A Graduate Course in Applied Cryptography

Dan Boneh and Victor Shoup

Version 0.6, Jan. 2023

Preface

Cryptography is an indispensable tool used to protect information in computing systems. It is used everywhere and by billions of people worldwide on a daily basis. It is used to protect data at rest and data in motion. Cryptographic systems are an integral part of standard protocols, most notably the Transport Layer Security (TLS) protocol, making it relatively easy to incorporate strong encryption into a wide range of applications.

While extremely useful, cryptography is also highly brittle. The most secure cryptographic system can be rendered completely insecure by a single specification or programming error. No amount of unit testing will uncover a security vulnerability in a cryptosystem.

Instead, to argue that a cryptosystem is secure, we rely on mathematical modeling and proofs to show that a particular system satisfies the security properties attributed to it. We often need to introduce certain plausible assumptions to push our security arguments through.

This book is about exactly that: constructing practical cryptosystems for which we can argue security under plausible assumptions. The book covers many constructions for different tasks in cryptography. For each task we define a precise security goal that we aim to achieve and then present constructions that achieve the required goal. To analyze the constructions, we develop a unified framework for doing cryptographic proofs. A reader who masters this framework will be capable of applying it to new constructions that may not be covered in the book.

Throughout the book we present many case studies to survey how deployed systems operate. We describe common mistakes to avoid as well as attacks on real-world systems that illustrate the importance of rigor in cryptography. We end every chapter with a fun application that applies the ideas in the chapter in some unexpected way.

Intended audience and how to use this book

The book is intended to be self contained. Some supplementary material covering basic facts from probability theory and algebra is provided in the appendices. The book is divided into three parts.

- Part I develops *symmetric encryption* which explains how two parties, Alice and Bob, can securely exchange information when they have a shared key unknown to the attacker. We discuss data confidentiality, data integrity, and the important concept of authenticated encryption.

- Part II develops the concepts of *public-key encryption* and *digital signatures*, which allow Alice and Bob to communicate securely, without having a pre-shared secret key.

- Part III is about *cryptographic protocols*, such as protocols for user identification, key exchange, zero knowledge, and secure computation.

A beginning reader can read though the book to learn how cryptographic systems work and why they are secure. Every security theorem in the book is followed by a proof idea that explains at a high level why the scheme is secure. On a first read one can skip over the detailed proofs without losing continuity. A beginning reader may also skip over the mathematical details sections that explore nuances of certain definitions.

An advanced reader may enjoy reading the detailed proofs to learn how to do proofs in cryptography. At the end of every chapter you will find many exercises that explore additional aspects of the material covered in the chapter. Some exercises rehearse what was learned, but many exercises expand on the material and present additional ideas that are not covered in the body. We recommend that readers read through the exercises, even if they do not intend to solve them.

Status of the book

The current draft is mostly complete, although there are a few missing sections here and there. Those sections, as well as the appendices, are forthcoming. We hope you enjoy this write-up. Please send us comments and let us know if you find typos or mistakes. We are very grateful to all the readers who have already sent us comments.

Citations: While the current draft is mostly complete, we have not yet included citations and references to the many works on which this book is based. Those will be coming soon and will be presented in the Notes section at the end of every chapter.

Dan Boneh and Victor Shoup
Jan. 2023

Contents

Part I

Secret key cryptography

Chapter 2

Encryption

Suppose Alice and Bob share a secret key k. Alice wants to transmit a message m to Bob over a network while maintaining the secrecy of m in the presence of an eavesdropping adversary. This chapter begins the development of basic techniques to solve this problem. Besides transmitting a message over a network, these same techniques allow Alice to store a file on a disk so that no one else with access to the disk can read the file, but Alice herself can read the file at a later time.

We should stress that while the techniques we develop in this chapter to solve this fundamental problem are important and interesting, they do not by themselves solve all problems related to "secure communication."

- The techniques only provide secrecy in the situation where Alice transmits a *single* message per key. If Alice wants to secretly transmit several messages using the *same* key, then she must use methods developed in Chapter 5.

- The techniques do not provide any assurances of *message integrity*: if the attacker has the ability to modify the bits of the ciphertext while it travels from Alice to Bob, then Bob may not realize that this happened, and accept a message other than the one that Alice sent. We will discuss techniques for providing message integrity in Chapter 6.

- The techniques do not provide a mechanism that allow Alice and Bob to come to share a secret key in the first place. Maybe they are able to do this using some secure network (or a physical, face-to-face meeting) at some point in time, while the message is sent at some later time when Alice and Bob must communicate over an insecure network. However, with an appropriate infrastructure in place, there are also protocols that allow Alice and Bob to exchange a secret key even over an insecure network: such protocols are discussed in Chapter 21.

2.1 Shannon ciphers and perfect security

2.1.1 Definition of a Shannon cipher

The basic mechanism for encrypting a message using a shared secret key is called a *cipher* (or *encryption scheme*). In this section, we introduce a slightly simplified notion of a cipher, which we call a **Shannon cipher**.

A **Shannon cipher** is a pair $\mathcal{E} = (E, D)$ of functions.

- The function E (the **encryption function**) takes as input a **key** k and a **message** m (also called a **plaintext**), and produces as output a **ciphertext** c. That is,

  $$c = E(k, m),$$

  and we say that c is the **encryption of** m **under** k.

- The function D (the **decryption function**) takes as input a key k and a ciphertext c, and produces a message m. That is,

  $$m = D(k, c),$$

  and we say that m is the **decryption of** c **under** k.

- We require that decryption "undoes" encryption; that is, the cipher must satisfy the following **correctness property**: for all keys k and all messages m, we have

  $$D(k,\ E(k,\ m)\ ) = m.$$

To be slightly more formal, let us assume that $\mathcal{K}$ is the set of all keys (the **key space**), $\mathcal{M}$ is the set of all messages (the **message space**), and that $\mathcal{C}$ is the set of all ciphertexts (the **ciphertext space**). With this notation, we can write:

$$E : \mathcal{K} \times \mathcal{M} \to \mathcal{C},$$
$$D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}.$$

Also, we shall say that $\mathcal{E}$ is **defined over** $(\mathcal{K}, \mathcal{M}, \mathcal{C})$.

Suppose Alice and Bob want to use such a cipher so that Alice can send a message to Bob. The idea is that Alice and Bob must somehow agree in advance on a key $k \in \mathcal{K}$. Assuming this is done, then when Alice wants to send a message $m \in \mathcal{M}$ to Bob, she encrypts m under k, obtaining the ciphertext $c = E(k, m) \in \mathcal{C}$, and then sends c to Bob via some communication network. Upon receiving c, Bob decrypts c under k, and the correctness property ensures that $D(k, c)$ is the same as Alice's original message m. For this to work, we have to assume that c is not tampered with in transit from Alice to Bob. Of course, the goal, intuitively, is that an eavesdropper, who may obtain c while it is in transit, does not learn too much about Alice's message m — this intuitive notion is what the formal definition of security, which we explore below, will capture.

In practice, keys, messages, and ciphertexts are often sequences of bytes. Keys are usually of some fixed length; for example, 16-byte (i.e., 128-bit) keys are very common. Messages and ciphertexts may be sequences of bytes of some fixed length, or of variable length. For example, a message may be a 1GB video file, a 10MB music file, a 1KB email message, or even a single bit encoding a "yes" or "no" vote in an electronic election.

Keys, messages, and ciphertexts may also be other types of mathematical objects, such as integers, or tuples of integers (perhaps lying in some specified interval), or other, more sophisticated types of mathematical objects (polynomials, matrices, or group elements). Regardless of how fancy these mathematical objects are, in practice, they must at some point be represented as sequences of bytes for purposes of storage in, and transmission between, computers.

For simplicity, in our mathematical treatment of ciphers, we shall assume that $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$ are sets of *finite* size. While this simplifies the theory, it means that if a real-world system allows messages of unbounded length, we will (somewhat artificially) impose a (large) upper bound on legal message lengths.

To exercise the above terminology, we take another look at some of the example ciphers discussed in Chapter 1.

Example 2.1. A **one-time pad** is a Shannon cipher $\mathcal{E} = (E, D)$, where the keys, messages, and ciphertexts are bit strings of the same length; that is, $\mathcal{E}$ is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where

$$\mathcal{K} := \mathcal{M} := \mathcal{C} := \{0,1\}^L,$$

for some fixed parameter L. For a key $k \in \{0,1\}^L$ and a message $m \in \{0,1\}^L$ the encryption function is defined as follows:

$$E(k, m) := k \oplus m,$$

and for a key $k \in \{0,1\}^L$ and ciphertext $c \in \{0,1\}^L$, the decryption function is defined as follows:

$$D(k, c) := k \oplus c.$$

Here, "$\oplus$" denotes bit-wise exclusive-OR, or in other words, component-wise addition modulo 2, and satisfies the following algebraic laws: for all bit vectors $x, y, z \in \{0,1\}^L$, we have

$$x \oplus y = y \oplus x, \quad x \oplus (y \oplus z) = (x \oplus y) \oplus z, \quad x \oplus 0^L = x, \quad \text{and} \quad x \oplus x = 0^L.$$

These properties follow immediately from the corresponding properties for addition modulo 2. Using these properties, it is easy to check that the correctness property holds for $\mathcal{E}$: for all $k, m \in \{0,1\}^L$, we have

$$D(k,\ E(k,\ m)\ ) = D(k,\ k \oplus m) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0^L \oplus m = m.$$

The encryption and decryption functions happen to be the same in this case, but of course, not all ciphers have this property. □

Example 2.2. A **variable length one-time pad** is a Shannon cipher $\mathcal{E} = (E, D)$, where the keys are bit strings of some fixed length L, while messages and ciphertexts are variable length bit strings, of length at most L. Thus, $\mathcal{E}$ is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where

$$\mathcal{K} := \{0,1\}^L \quad \text{and} \quad \mathcal{M} := \mathcal{C} := \{0,1\}^{\leq L}.$$

for some parameter L. Here, $\{0,1\}^{\leq L}$ denotes the set of all bit strings of length at most L (including the empty string). For a key $k \in \{0,1\}^L$ and a message $m \in \{0,1\}^{\leq L}$ of length ℓ, the encryption function is defined as follows:

$$E(k, m) := k[0 \mathinner{..} \ell - 1] \oplus m,$$

and for a key $k \in \{0,1\}^L$ and ciphertext $c \in \{0,1\}^{\leq L}$ of length ℓ, the decryption function is defined as follows:

$$D(k, c) := k[0 \mathinner{..} \ell - 1] \oplus c.$$

Here, $k[0 \mathinner{..} \ell - 1]$ denotes the truncation of k to its first ℓ bits. The reader may verify that the correctness property holds for $\mathcal{E}$. □

Example 2.3. A **substitution cipher** is a Shannon cipher $\mathcal{E} = (E, D)$ of the following form. Let Σ be a finite alphabet of symbols (e.g., the letters `A`–`Z`, plus a space symbol, ␣). The message space $\mathcal{M}$ and the ciphertext space $\mathcal{C}$ are both sequences of symbols from Σ of some fixed length L:

$$\mathcal{M} := \mathcal{C} := \Sigma^L.$$

The key space $\mathcal{K}$ consists of all permutations on Σ; that is, each $k \in \mathcal{K}$ is a one-to-one function from Σ onto itself. Note that $\mathcal{K}$ is a very large set; indeed, $|\mathcal{K}| = |\Sigma|!$ (for $|\Sigma| = 27$, $|\mathcal{K}| \approx 1.09 \cdot 10^{28}$).

Encryption of a message $m \in \Sigma^L$ under a key $k \in \mathcal{K}$ (a permutation on Σ) is defined as follows

$$E(k, m) := \big(\ k(m[0]), k(m[1]), \ldots, k(m[L-1])\ \big),$$

where $m[i]$ denotes the ith entry of m (counting from zero), and $k(m[i])$ denotes the application of the permutation k to the symbol $m[i]$. Thus, to encrypt m under k, we simply apply the permutation k component-wise to the sequence m. Decryption of a ciphertext $c \in \Sigma^L$ under a key $k \in \mathcal{K}$ is defined as follows:

$$D(k, c) := \big(\ k^{-1}(c[0]), k^{-1}(c[1]), \ldots, k^{-1}(c[L-1])\ \big).$$

Here, k^{-1} is the inverse permutation of k, and to decrypt c under k, we simply apply k^{-1} component-wise to the sequence c. The correctness property is easily verified: for a message $m \in \Sigma^L$ and key $k \in \mathcal{K}$, we have

$$\begin{aligned} D(k,\ E(k,\ m)\ ) &= D(k,\ (k(m[0]), k(m[1]), \ldots, k(m[L-1])\ ) \\ &= (k^{-1}(k(m[0])), k^{-1}(k(m[1])), \ldots, k^{-1}(k(m[L-1]))) \\ &= (m[0], m[1], \ldots, m[L-1]) = m. \quad \Box \end{aligned}$$

Example 2.4 (additive one-time pad). We may also define a "addition mod n" variation of the one-time pad. This is a cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{K} := \mathcal{M} := \mathcal{C} := \{0, \ldots, n-1\}$, where n is a positive integer. Encryption and decryption are defined as follows:

$$E(k, m) := m + k \bmod n \qquad D(k, c) := c - k \bmod n.$$

The reader may easily verify that the correctness property holds for $\mathcal{E}$. □

2.1.2 Perfect security

So far, we have just defined the basic syntax and correctness requirements of a Shannon cipher. Next, we address the question: what is a "secure" cipher? Intuitively, the answer is that a secure cipher is one for which an encrypted message remains "well hidden," even after seeing its encryption. However, turning this intuitive answer into one that is both mathematically meaningful and practically relevant is a real challenge. Indeed, although ciphers have been used for centuries, it is only in the last few decades that mathematically acceptable definitions of security have been developed.

In this section, we develop the mathematical notion of **perfect security** — this is the "gold standard" for security (at least, when we are only worried about encrypting a single message and do not care about integrity). We will also see that it is possible to achieve this level of security; indeed, we will show that the one-time pad satisfies the definition. However, the one-time pad is not very practical, in the sense that the keys must be as long as the messages: if Alice wants to send a 1GB file to Bob, they must already share a 1GB key! Unfortunately, this cannot be avoided: we will also prove that any perfectly secure cipher must have a key space at least as large as its message space. This fact provides the motivation for developing a definition of security that is weaker, but that is acceptable from a practical point of view, and which allows one to encrypt long messages using short keys.

If Alice encrypts a message m under a key k, and an eavesdropping adversary obtains the ciphertext c, Alice only has a hope of keeping m secret if the key k is hard to guess, and that means, at the very least, that the key k should be chosen at random from a large key space. To say that m is "well hidden" must at least mean that it is hard to completely determine m from c, without knowledge of k; however, this is not really enough. Even though the adversary may not know k, we assume that he does know the encryption algorithm and the distribution of k. In fact, we will assume that when a message is encrypted, the key k is always chosen at random, uniformly from among all keys in the key space. The adversary may also have some knowledge of the message encrypted — because of circumstances, he may know that the set of possible messages is quite small, and he may know something about how likely each possible message is. For example, suppose he knows the message m is either $m_0 =$ `"ATTACK␣AT␣DAWN"` or $m_1 =$ `"ATTACK␣AT␣DUSK"`, and that based on the adversary's available intelligence, Alice is equally likely to choose either one of these two messages. Without seeing the ciphertext c, the adversary would only have a 50% chance of guessing which message Alice sent. But we are assuming the adversary does know c. Even with this knowledge, both messages may be possible; that is, there may exist keys k_0 and k_1 such that $E(k_0, m_0) = c$ and $E(k_1, m_1) = c$, so he cannot *be sure* if $m = m_0$ or $m = m_1$. However, he can still guess. Perhaps it is a property of the cipher that there are 800 keys k_0 such that $E(k_0, m_0) = c$, and 600 keys k_1 such that $E(k_1, m_1) = c$. If that is the case, the adversary's best guess would be that $m = m_0$. Indeed, the probability that this guess is correct is equal to $800/(800 + 600) \approx 57\%$, which is better than the 50% chance he would have without knowledge of the ciphertext. Our formal definition of perfect security expressly rules out the possibility that knowledge of the ciphertext increases the probability of guessing the encrypted message, or for that matter, determining *any* property of the message whatsoever.

Without further ado, we formally define perfect security. In this definition, we will consider a probabilistic experiment in which the key is drawn uniformly from the key space. We write k to denote the random variable representing this random key. For a message m, $E(\mathsf{k}, m)$ is another random variable, which represents the application of the encryption function to our random key and the message m. Thus, every message m gives rise to a different random variable $E(\mathsf{k}, m)$.

Definition 2.1 (perfect security). *Let $\mathcal{E} = (E, D)$ be a Shannon cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Consider a probabilistic experiment in which the random variable k is uniformly distributed over $\mathcal{K}$. If for all $m_0, m_1 \in \mathcal{M}$, and all $c \in \mathcal{C}$, we have*

$$\Pr[E(\mathsf{k}, m_0) = c] = \Pr[E(\mathsf{k}, m_1) = c],$$

then we say that $\mathcal{E}$ is a ***perfectly secure*** *Shannon cipher.*

There are a number of equivalent formulations of perfect security that we shall explore. We state a couple of these here.

Theorem 2.1. *Let $\mathcal{E} = (E, D)$ be a Shannon cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. The following are equivalent:*

(i) *$\mathcal{E}$ is perfectly secure.*

(ii) *For every $c \in \mathcal{C}$, there exists an integer N_c (possibly depending on c) such that for all $m \in \mathcal{M}$, we have*

$$\big|\{k \in \mathcal{K} : E(k,m) = c\}\big| = N_c.$$

(iii) *If the random variable k is uniformly distributed over $\mathcal{K}$, then each of the random variables $E(\mathsf{k}, m)$, for $m \in \mathcal{M}$, has the same distribution.*

Proof. To begin with, let us restate (ii) as follows: for every $c \in \mathcal{C}$, there exists a number P_c (depending on c) such that for all $m \in \mathcal{M}$, we have $\Pr[E(\mathsf{k}, m) = c] = P_c$. Here, k is a random variable uniformly distributed over $\mathcal{K}$. Note that $P_c = N_c/|\mathcal{K}|$, where N_c is as in the original statement of (ii).

This version of (ii) is clearly the same as (iii).

(i) $\Longrightarrow$ (ii). We prove (ii) assuming (i). To prove (ii), let $c \in \mathcal{C}$ be some fixed ciphertext. Pick some arbitrary message $m_0 \in \mathcal{M}$, and let $P_c := \Pr[E(\mathsf{k}, m_0) = c]$. By (i), we know that for all $m \in \mathcal{M}$, we have $\Pr[E(\mathsf{k}, m) = c] = \Pr[E(\mathsf{k}, m_0) = c] = P_c$. That proves (ii).

(ii) $\Longrightarrow$ (i). We prove (i) assuming (ii). Consider any fixed $m_0, m_1 \in \mathcal{M}$ and $c \in \mathcal{C}$. (ii) says that $\Pr[E(\mathsf{k}, m_0) = c] = P_c = \Pr[E(\mathsf{k}, m_1) = c]$, which proves (i). □

As promised, we give a proof that the one-time pad (see Example 2.1) is perfectly secure.

Theorem 2.2. *The one-time pad is a perfectly secure Shannon cipher.*

Proof. Suppose that the Shannon cipher $\mathcal{E} = (E, D)$ is a one-time pad, and is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{K} := \mathcal{M} := \mathcal{C} := \{0,1\}^L$. For any fixed message $m \in \{0,1\}^L$ and ciphertext $c \in \{0,1\}^L$, there is a unique key $k \in \{0,1\}^L$ satisfying the equation

$$k \oplus m = c,$$

namely, $k := m \oplus c$. Therefore, $\mathcal{E}$ satisfies condition (ii) in Theorem 2.1 (with $N_c = 1$ for each c). □

Example 2.5. Consider again the variable length one-time pad, defined in Example 2.2. This does not satisfy our definition of perfect security, since a ciphertext has the same length as the corresponding plaintext. Indeed, let us choose an arbitrary string of length 1, call it m_0, and an arbitrary string of length 2, call it m_1. In addition, suppose that c is an arbitrary length 1 string, and that k is a random variable that is uniformly distributed over the key space. Then we have

$$\Pr[E(\mathsf{k}, m_0) = c] = 1/2 \quad \text{and} \quad \Pr[E(\mathsf{k}, m_1) = c] = 0,$$

which provides a direct counter-example to Definition 2.1.

Intuitively, the variable length one-time pad cannot satisfy our definition of perfect security simply because any ciphertext leaks the *length* of the corresponding plaintext. However, in some sense (which we do not make precise right now), this is the *only* information leaked. It is perhaps not clear whether this should be viewed as a problem with the cipher or with our definition of perfect security. On the one hand, one can imagine scenarios where the length of a message may vary greatly, and while we could always "pad" short messages to effectively make all messages equally long, this may be unacceptable from a practical point of view, as it is a waste of bandwidth. On the other hand, one must be aware of the fact that in certain applications, leaking just the length of a message may be dangerous: if you are encrypting a "yes" or "no" answer to a question, just the length of the obvious ASCII encoding of these strings leaks *everything*, so you better pad "no" out to three characters. □

Example 2.6. Consider again the substitution cipher defined in Example 2.3. There are a couple of different ways to see that this cipher is not perfectly secure.

For example, choose a pair of messages $m_0, m_1 \in \Sigma^L$ such that the first two components of m_0 are equal, yet the first two components of m_1 are not equal; that is,

$$m_0[0] = m_0[1] \quad \text{and} \quad m_1[0] \neq m_1[1].$$

Then for each key k, which is a permutation on Σ, if $c = E(k, m_0)$, then $c[0] = c[1]$, while if $c = E(k, m_1)$, then $c[0] \neq c[1]$. In particular, it follows that if k is uniformly distributed over the key space, then the distributions of $E(\mathsf{k}, m_0)$ and $E(\mathsf{k}, m_1)$ will not be the same.

Even the weakness described in the previous paragraph may seem somewhat artificial. Another, perhaps more realistic, type of attack on the substitution cipher works as follows. Suppose the substitution cipher is used to encrypt email messages. As anyone knows, an email starts with a "standard header," such as `"FROM"`. Suppose the ciphertext is $c \in \Sigma^L$ is intercepted by an adversary. The secret key is actually a permutation k on Σ. The adversary knows that

$$c[0 \ldots 3] = (k(\mathtt{F}), k(\mathtt{R}), k(\mathtt{O}), k(\mathtt{M})).$$

Thus, if the original message is $m \in \Sigma^L$, the adversary can now locate all positions in m where an `F` occurs, where an `R` occurs, where an `O` occurs, and where an `M` occurs. Based just on this information, along with specific, contextual information about the message, together with general information about letter frequencies, the adversary may be able to deduce quite a bit about the original message. □

Example 2.7. Consider the additive one-time pad, defined in Example 2.4. It is easy to verify that this is perfectly secure. Indeed, it satisfies condition (ii) in Theorem 2.1 (with $N_c = 1$ for each c). □

The next two theorems develop two more alternative characterizations of perfect security. For the first, suppose an eavesdropping adversary applies some predicate ϕ to a ciphertext he has obtained. The predicate ϕ (which is a boolean-valued function on the ciphertext space) may be something very simple, like the parity function (i.e., whether the number of 1 bits in the ciphertext is even or odd), or it might be some more elaborate type of statistical test. Regardless of how clever or complicated the predicate ϕ is, perfect security guarantees that the value of this predicate on the ciphertext reveals nothing about the message.

Theorem 2.3. *Let $\mathcal{E} = (E, D)$ be a Shannon cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Consider a probabilistic experiment in which k is a random variable uniformly distributed over $\mathcal{K}$. Then $\mathcal{E}$ is perfectly secure if and only if for every predicate ϕ on $\mathcal{C}$, for all $m_0, m_1 \in \mathcal{M}$, we have*

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \Pr[\phi(E(\mathsf{k}, m_1))].$$

Proof. This is really just a simple calculation. On the one hand, suppose $\mathcal{E}$ is perfectly secure, and let ϕ, m_0, and m_1 be given. Let $S := \{c \in \mathcal{C} : \phi(c)\}$. Then we have

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \sum_{c \in S} \Pr[E(\mathsf{k}, m_0) = c] = \sum_{c \in S} \Pr[E(\mathsf{k}, m_1) = c] = \Pr[\phi(E(\mathsf{k}, m_1))].$$

Here, we use the assumption that $\mathcal{E}$ is perfectly secure in establishing the second equality. On the other hand, suppose $\mathcal{E}$ is not perfectly secure, so there exist m_0, m_1, and c such that

$$\Pr[E(\mathsf{k}, m_0) = c] \neq \Pr[E(\mathsf{k}, m_1) = c].$$

Defining ϕ to be the predicate that is true for this particular c, and false for all other ciphertexts, we see that

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \Pr[E(\mathsf{k}, m_0) = c] \neq \Pr[E(\mathsf{k}, m_1) = c] = \Pr[\phi(E(\mathsf{k}, m_1))]. \quad \square$$

The next theorem states in yet another way that perfect security guarantees that the ciphertext reveals nothing about the message. Suppose that m is a random variable distributed over the message space $\mathcal{M}$. We do not assume that m is uniformly distributed over $\mathcal{M}$. Now suppose k is a random variable uniformly distributed over the key space $\mathcal{K}$, independently of m, and define $\mathsf{c} := E(\mathsf{k}, \mathsf{m})$, which is a random variable distributed over the ciphertext space $\mathcal{C}$. The following theorem says that perfect security guarantees that c and m are independent random variables.

One way of characterizing this independence is to say that for each ciphertext $c \in \mathcal{C}$ that occurs with nonzero probability, and each message $m \in \mathcal{M}$, we have

$$\Pr[\mathsf{m} = m \mid \mathsf{c} = c] = \Pr[\mathsf{m} = m].$$

Intuitively, this means that after seeing a ciphertext, we have no more information about the message than we did before seeing the ciphertext.

Another way of characterizing this independence is to say that for each message $m \in \mathcal{M}$ that occurs with nonzero probability, and each ciphertext $c \in \mathcal{C}$, we have

$$\Pr[\mathsf{c} = c \mid \mathsf{m} = m] = \Pr[\mathsf{c} = c].$$

Intuitively, this means that the choice of message has no impact on the distribution of the ciphertext.

The restriction that m and k are independent random variables is sensible: in using any cipher, it is a very bad idea to choose the key in a way that depends on the message, or vice versa (see Exercise 2.16).

Theorem 2.4. *Let $\mathcal{E} = (E, D)$ be a Shannon cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Consider a random experiment in which k and m are random variables, such that*

- *k is uniformly distributed over $\mathcal{K}$,*
- *m is distributed over $\mathcal{M}$, and*
- *k and m are independent.*

Define the random variable $\mathsf{c} := E(\mathsf{k}, \mathsf{m})$. Then we have:

- *if $\mathcal{E}$ is perfectly secure, then c and m are independent;*
- *conversely, if c and m are independent, and each message in $\mathcal{M}$ occurs with nonzero probability, then $\mathcal{E}$ is perfectly secure.*

Proof. For the first implication, assume that $\mathcal{E}$ is perfectly secure. Consider any fixed $m \in \mathcal{M}$ and $c \in \mathcal{C}$. We want to show that

$$\Pr[\mathsf{c} = c \wedge \mathsf{m} = m] = \Pr[\mathsf{c} = c] \Pr[\mathsf{m} = m].$$

We have

$$\begin{aligned}\Pr[\mathsf{c} = c \wedge \mathsf{m} = m] &= \Pr[E(\mathsf{k}, \mathsf{m}) = c \wedge \mathsf{m} = m] \\ &= \Pr[E(\mathsf{k}, m) = c \wedge \mathsf{m} = m] \\ &= \Pr[E(\mathsf{k}, m) = c] \Pr[\mathsf{m} = m] \quad \text{(by independence of } \mathsf{k} \text{ and } \mathsf{m}\text{).}\end{aligned}$$

So it will suffice to show that $\Pr[E(\mathsf{k}, m) = c] = \Pr[\mathsf{c} = c]$. But we have

$$\begin{aligned}\Pr[\mathsf{c} = c] &= \Pr[E(\mathsf{k}, \mathsf{m}) = c] \\ &= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, \mathsf{m}) = c \wedge \mathsf{m} = m'] \quad \text{(by total probability)} \\ &= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, m') = c \wedge \mathsf{m} = m'] \\ &= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, m') = c] \Pr[\mathsf{m} = m'] \quad \text{(by independence of } \mathsf{k} \text{ and } \mathsf{m}\text{)} \\ &= \sum_{m' \in \mathcal{M}} \Pr[E(\mathsf{k}, m) = c] \Pr[\mathsf{m} = m'] \quad \text{(by definition of perfect security)} \\ &= \Pr[E(\mathsf{k}, m) = c] \sum_{m' \in \mathcal{M}} \Pr[\mathsf{m} = m'] \\ &= \Pr[E(\mathsf{k}, m) = c] \quad \text{(probabilities sum to 1).}\end{aligned}$$

For the second implication, assume that c and m are independent, and each message in $\mathcal{M}$ occurs with nonzero probability. Let $m \in \mathcal{M}$ and $c \in \mathcal{C}$. We will show that $\Pr[E(\mathsf{k}, m) = c] = \Pr[\mathsf{c} = c]$, from which perfect security immediately follows. Since $\Pr[\mathsf{m} = m] \neq 0$, this is seen thusly:

$$\begin{aligned}\Pr[E(\mathsf{k}, m) = c] \Pr[\mathsf{m} = m] &= \Pr[E(\mathsf{k}, m) = c \wedge \mathsf{m} = m] \quad \text{(by independence of } \mathsf{k} \text{ and } \mathsf{m}\text{)} \\ &= \Pr[E(\mathsf{k}, \mathsf{m}) = c \wedge \mathsf{m} = m] \\ &= \Pr[\mathsf{c} = c \wedge \mathsf{m} = m] \\ &= \Pr[\mathsf{c} = c] \Pr[\mathsf{m} = m] \quad \text{(by independence of } \mathsf{c} \text{ and } \mathsf{m}\text{).} \quad \square\end{aligned}$$

2.1.3 The bad news

We have saved the bad news for last. The next theorem shows that perfect security is such a powerful notion that one can really do no better than the one-time pad: keys must be at least as long as messages. As a result, it is almost impossible to use perfectly secure ciphers in practice: if Alice wants to send Bob a 1GB video file, then Alice and Bob have to agree on a 1GB secret key in advance.

Theorem 2.5 (Shannon's theorem). *Let $\mathcal{E} = (E, D)$ be a Shannon cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. If $\mathcal{E}$ is perfectly secure, then $|\mathcal{K}| \geq |\mathcal{M}|$.*

Proof. Assume that $|\mathcal{K}| < |\mathcal{M}|$. We want to show that $\mathcal{E}$ is not perfectly secure. To this end, we show that there exist messages m_0 and m_1, and a ciphertext c, such that

$$\Pr[E(\mathsf{k}, m_0) = c] > 0, \quad \text{and} \tag{2.1}$$

$$\Pr[E(\mathsf{k}, m_1) = c] = 0. \tag{2.2}$$

Here, k is a random variable, uniformly distributed over $\mathcal{K}$.

To do this, choose any message $m_0 \in \mathcal{M}$, and any key $k_0 \in \mathcal{K}$. Let $c := E(k_0, m_0)$. It is clear that (2.1) holds.

Next, let

$$S := \{D(k_1, c) : k_1 \in \mathcal{K}\}.$$

Clearly,

$$|S| \leq |\mathcal{K}| < |\mathcal{M}|,$$

and so we can choose a message $m_1 \in \mathcal{M} \setminus S$.

To prove (2.2), we need to show that there is no key k_1 such that $E(k_1, m_1) = c$. Assume to the contrary that $E(k_1, m_1) = c$ for some k_1; then for this key k_1, by the correctness property for ciphers, we would have

$$D(k_1, c) = D(k_1,\ E(k_1,\ m_1)\ ) = m_1,$$

which would imply that m_1 belongs to S, which is not the case. That proves (2.2), and the theorem follows. □

2.2 Computational ciphers and semantic security

As we have seen in Shannon's theorem (Theorem 2.5), the only way to achieve perfect security is to have keys that are as long as messages. However, this is quite impractical: we would like to be able to encrypt a long message (say, a document of several megabytes) using a short key (say, a few hundred bits). The only way around Shannon's theorem is to relax our security requirements. The way we shall do this is to consider not all possible adversaries, but only *computationally feasible* adversaries, that is, "real world" adversaries that must perform their calculations on real computers using a reasonable amount of time and memory. This will lead to a weaker definition of security called **semantic security**. Furthermore, our definition of security will be flexible enough to allow ciphers with variable length message spaces to be considered secure so long as they do not leak any useful information about an encrypted message to an adversary *other than the length of the message.* Also, since our focus is now on the "practical," instead of the "mathematically possible," we shall also insist that the encryption and decryption functions are themselves efficient algorithms, and not just arbitrary functions.

2.2.1 Definition of a computational cipher

A **computational cipher** $\mathcal{E} = (E, D)$ is a pair of efficient algorithms, E and D. The encryption algorithm E takes as input a key k, along with a message m, and produces as output a ciphertext c. The decryption algorithm D takes as input a key k, a ciphertext c, and outputs a message m. Keys lie in some finite key space $\mathcal{K}$, messages lie in a finite message space $\mathcal{M}$, and ciphertexts lie in some finite ciphertext space $\mathcal{C}$. Just as for a Shannon cipher, we say that $\mathcal{E}$ is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$.

Although it is not really necessary for our purposes in this chapter, we will allow the encryption function E to be a *probabilistic* algorithm. This means that for fixed inputs k and m, the output of $E(k, m)$ may be one of many values. Probabilistic algorithms are discussed further in Appendix D. To emphasize the probabilistic nature of encryption we write

$$c \overset{\text{R}}{\leftarrow} E(k, m)$$

to denote the process of executing $E(k, m)$ and assigning the output to the program variable c. We shall use this notation throughout the book whenever we use probabilistic algorithms. Similarly, we write

$$k \overset{\text{R}}{\leftarrow} \mathcal{K}$$

to denote the process of assigning to the program variable k a random, uniformly distributed element from the key space $\mathcal{K}$. We shall use the analogous notation to sample uniformly from any finite set.

We will not see any examples of probabilistic encryption algorithms in this chapter (we will see our first examples of this in Chapter 5). Although one could allow the decryption algorithm to be probabilistic, we will have no need for this, and so will only discuss ciphers with deterministic decryption algorithms. However, it will be occasionally be convenient to allow the decryption algorithm to return a special reject value (distinct from all messages), indicating some kind of error occurred during the decryption process.

Since the encryption algorithm is probabilistic, for a given key k and message m, the encryption algorithm may output one of many possible ciphertexts; however, each of these possible ciphertexts should decrypt to m. We can state this **correctness requirement** more formally as follows: for all keys $k \in \mathcal{K}$ and messages $m \in \mathcal{M}$, if we execute

$$c \overset{\text{R}}{\leftarrow} E(k, m),\ m' \leftarrow D(k, c),$$

then $m = m'$ with probability 1.

From now on, whenever we refer to a ***cipher****, we shall mean a* ***computational cipher****, as defined above. Moreover, if the encryption algorithm happens to be deterministic, then we may call the cipher a* ***deterministic cipher****.*

Observe that any deterministic cipher is a Shannon cipher; however, a computational cipher need not be a Shannon cipher (if it has a probabilistic encryption algorithm), and a Shannon cipher need not be a computational cipher (if its encryption or decryption operations have no efficient implementations).

Example 2.8. The one-time pad (see Example 2.1) and the variable length one-time pad (see Example 2.2) are both deterministic ciphers, since their encryption and decryption operations may be trivially implemented as efficient, deterministic algorithms. The same holds for the substitution cipher (see Example 2.3), provided the alphabet Σ is not too large. Indeed, in the obvious implementation, a key — which is a permutation on Σ — will be represented by an array indexed by Σ, and so we will require $O(|\Sigma|)$ space just to store a key. This will only be practical for reasonably sized Σ. The additive one-time pad discussed in Example 2.4 is also a deterministic cipher, since both encryption and decryption operations may be efficiently implemented (if n is large, special software to do arithmetic with large integers may be necessary). □

2.2.2 Definition of semantic security

To motivate the definition of semantic security, consider a deterministic cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Consider again the formulation of perfect security in Theorem 2.3. This says that for all predicates ϕ on the ciphertext space, and all messages m_0, m_1, we have

$$\Pr[\phi(E(\mathsf{k}, m_0))] = \Pr[\phi(E(\mathsf{k}, m_1))], \tag{2.3}$$

where k is a random variable uniformly distributed over the key space $\mathcal{K}$. Instead of insisting that these probabilities are equal, we shall only require that they are very close; that is,

$$\Big|\Pr[\phi(E(\mathsf{k}, m_0))] - \Pr[\phi(E(\mathsf{k}, m_1))]\Big| \leq \epsilon, \tag{2.4}$$

for some very small, or *negligible*, value of ϵ. By itself, this relaxation does not help very much (see Exercise 2.5). However, instead of requiring that (2.4) holds for every possible ϕ, m_0, and m_1, we only require that (2.4) holds for all messages m_0 and m_1 that can be generated by some efficient algorithm, and all predicates ϕ that can be computed by some efficient algorithm (these algorithms could be probabilistic). For example, suppose it were the case that using the best possible algorithms for generating m_0 and m_1, and for testing some predicate ϕ, and using (say) 10,000 computers in parallel for 10 years to perform these calculations, (2.4) holds for $\epsilon = 2^{-100}$. While not perfectly secure, we might be willing to say that the cipher is *secure for all practical purposes.*

Also, in defining semantic security, we address an issue raised in Example 2.5. In that example, we saw that the variable length one-time pad did not satisfy the definition of perfect security. However, we want our definition to be flexible enough so that ciphers like the variable length one-time pad, which effectively leak no information about an encrypted message other than its length, may be considered secure as well.

Now the details. To precisely formulate the definition of semantic security, we shall describe an **attack game** played between two parties, a **challenger** and an **adversary**. Throughout this book, we shall formulate many attack games that capture various notions of security for different types of cryptographic primitives. In general, the challenger follows a very simple, fixed protocol. However, an adversary may follow an arbitrary (but still efficient) protocol. The challenger and the adversary send messages back and forth to each other, as specified by their protocols, and at the end of the game, the adversary may output some value. The attack game also defines a probability space, and this in turn defines the adversary's **advantage**, which is determined by the probability of one or more events.

Some attack games, such as the one defining the semantic security of a cipher, comprise two alternative "sub-games," or "experiments" — in both experiments, the adversary follows the same protocol, while the challenger behaves differently in each experiment. Specifically, in our attack game defining the semantic security of a cipher, the adversary generates two messages m_0 and m_1 (of the same length), and sends both messages to the challenger. In one experiment, the challenger encrypts m_0 under a random key, while in the other experiment, the challenger encrypts m_1, again, under a random key. In both experiments, the challenger sends the resulting ciphertext c back to the adversary. After examining c, the adversary outputs a bit $\hat{b} \in \{0, 1\}$. The advantage of the adversary is defined to be the absolute difference between the probability the adversary outputs $\hat{b} = 1$ in one experiment and the probability that it outputs $\hat{b} = 1$ the other experiment. We state this more formally as follows:

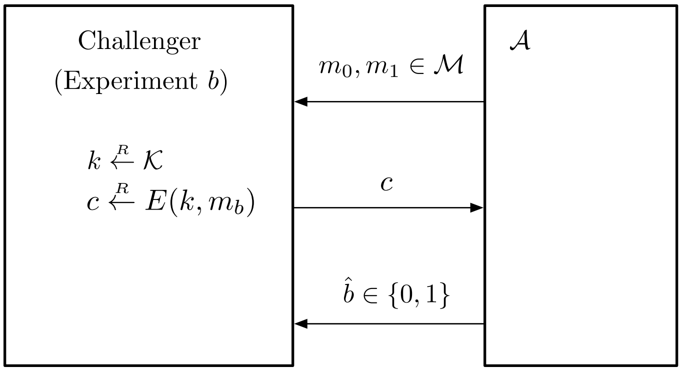

Challenger
(Experiment b)
$k \overset{R}{\leftarrow} \mathcal{K}$
$c \overset{R}{\leftarrow} E(k, m_b)$
$\mathcal{A}$
$m_0, m_1 \in \mathcal{M}$
c
$\hat{b} \in \{0,1\}$

Figure 2.1: Experiment b of Attack Game 2.1

Attack Game 2.1 (semantic security). For a given cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define

Experiment b:

- The adversary computes $m_0, m_1 \in \mathcal{M}$, of the same length, and sends them to the challenger.
- The challenger computes $k \overset{\text{R}}{\leftarrow} \mathcal{K}$, $c \overset{\text{R}}{\leftarrow} E(k, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **semantic security advantage** with respect to $\mathcal{E}$ as

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Note that in the above game, the events W_0 and W_1 are defined with respect to the probability space determined by the random choice of k, the random choices made (if any) by the encryption algorithm, and the random choices made (if any) by the adversary. The value $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ is a number between 0 and 1.

See Fig. 2.1 for a schematic diagram of Attack Game 2.1. As indicated in the diagram, $\mathcal{A}$'s "output" is really just a final message to the challenger.

Definition 2.2 (semantic security). *A cipher $\mathcal{E}$ is **semantically secure** if for all efficient adversaries $\mathcal{A}$, the value $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ is negligible.*

As a formal definition, this is not quite complete, as we have yet to define what we mean by "messages of the same length", "efficient adversaries", and "negligible". We will come back to this shortly.

Let us relate this formal definition to the discussion preceding it. Suppose that the adversary $\mathcal{A}$ in Attack Game 2.1 is deterministic. First, the adversary computes in a deterministic fashion messages m_0, m_1, and then evaluates a predicate ϕ on the ciphertext c, outputting 1 if true and 0 if false. Semantic security says that the value ϵ in (2.4) is negligible. In the case where $\mathcal{A}$ is probabilistic, we can view $\mathcal{A}$ as being structured as follows: it generates a random value r from some appropriate set, and deterministically computes messages $m_0^{(r)}$, $m_1^{(r)}$, which depend on r, and evaluates a predicate $\phi^{(r)}$ on c, which also depends on r. Here, semantic security says that the value ϵ in (2.4), with m_0, m_1, ϕ replaced by $m_0^{(r)}, m_1^{(r)}, \phi^{(r)}$, is negligible — but where now the probability is with respect to a randomly chosen key and a randomly chosen value of r.

Remark 2.1. Let us now say a few words about the requirement that the messages m_0 and m_1 computed by the adversary Attack Game 2.1 be of the same length.

- First, the notion of the "length" of a message is specific to the particular message space $\mathcal{M}$; in other words, in specifying a message space, one must specify a rule that associates a length (which is a non-negative integer) with any given message. For most concrete message spaces, this will be clear: for example, for the message space $\{0,1\}^{\leq L}$ (as in Example 2.2), the length of a message $m \in \{0,1\}^{\leq L}$ is simply its length, $|m|$, as a bit string. However, to make our definition somewhat general, we leave the notion of length as an abstraction. Indeed, some message spaces may have no particular notion of length, in which case all messages may be viewed as having length 0.

- Second, the requirement that m_0 and m_1 be of the same length means that the adversary is not deemed to have broken the system just because he can effectively distinguish an encryption of a message of one length from an encryption of a message of a different length. This is how our formal definition captures the notion that an encryption of a message is allowed to leak the length of the message (but nothing else).

  We already discussed in Example 2.5 how in certain applications, leaking the length of the message can be catastrophic. However, since there is no general solution to this problem, most real-world encryption schemes (for example, TLS) do not make any attempt at all to hide the length of the message. This can lead to real attacks. For example, Chen et al. [41] show that the lengths of encrypted messages can reveal considerable information about private data that a user supplies to a cloud application. They use an online tax filing system as their example, but other works show attacks of this type on many other systems. □

Example 2.9. Let $\mathcal{E}$ be a deterministic cipher that is perfectly secure. Then it is easy to see that for every adversary $\mathcal{A}$ (efficient or not), we have $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 0$. This follows almost immediately from Theorem 2.3 (the only slight complication is that our adversary $\mathcal{A}$ in Attack Game 2.1 may be probabilistic, but this is easily dealt with). In particular, $\mathcal{E}$ is semantically secure. Thus, if $\mathcal{E}$ is the one-time pad (see Example 2.1), we have $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 0$ for all adversaries $\mathcal{A}$; in particular, the one-time pad is semantically secure. Because the definition of semantic security is a bit more forgiving with regard to variable length message spaces, it is also easy to see that if $\mathcal{E}$ is the variable length one-time pad (see Example 2.2), then $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 0$ for all adversaries $\mathcal{A}$; in particular, the variable length one-time pad is also semantically secure. □

We need to say a few words about the terms "efficient" and "negligible". Below in Section 2.3 we will fill in the remaining details (they are somewhat tedious, and not really very enlightening). Intuitively, *negligible* means so small as to be "zero for all practical purposes": think of a number like 2^{-100} — if the probability that you spontaneously combust in the next year is 2^{-100}, then you would not worry about such an event occurring any more than you would an event that occurred with probability 0. We also use the following terms:

- An *efficient adversary* is one that runs in a "reasonable" amount time.
- A value N is called *super-poly* if $1/N$ is negligible.
- A *poly-bounded* value is a "reasonably" sized number. In particular, we can say that the running time of an efficient adversary is poly-bounded.

Fact 2.6. *If ϵ and ϵ' are negligible values, and Q and Q' are poly-bounded values, then:*

(i) $\epsilon + \epsilon'$ *is a negligible value,*

(ii) $Q + Q'$ *and* $Q \cdot Q'$ *are poly-bounded values, and*

(iii) $Q \cdot \epsilon$ *is a negligible value.*

For now, the reader can just take these facts as axioms. Instead of dwelling on these technical issues, we discuss an example that illustrates how one typically *uses* this definition in analyzing the security of a larger system that uses a semantically secure cipher.

2.2.3 Connections to weaker notions of security

2.2.3.1 Message recovery attacks

Intuitively, in a message recovery attack, an adversary is given an encryption of a random message, and is able to recover the message from the ciphertext with probability significantly better than random guessing, that is, probability $1/|\mathcal{M}|$. Of course, any reasonable notion of security should rule out such an attack, and indeed, semantic security does.

While this may seem intuitively obvious, we give a formal proof of this. One of our motivations for doing this is to illustrate in detail the notion of a *security reduction*, which is the main technique used to reason about the security of systems. Basically, the proof will argue that any efficient adversary $\mathcal{A}$ that can effectively mount a message recovery attack on $\mathcal{E}$ can be used to build an efficient adversary $\mathcal{B}$ that breaks the semantic security of $\mathcal{E}$; since semantic security implies that no such $\mathcal{B}$ exists, we may conclude that no such $\mathcal{A}$ exists.

To formulate this proof in more detail, we need a formal definition of a message recovery attack. As before, this is done by giving attack game, which is a protocol between a challenger and an adversary.

Attack Game 2.2 (message recovery). For a given cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, the attack game proceeds as follows:

- The challenger computes $m \overset{\text{R}}{\leftarrow} \mathcal{M}$, $k \overset{\text{R}}{\leftarrow} \mathcal{K}$, $c \overset{\text{R}}{\leftarrow} E(k, m)$, and sends c to the adversary.
- The adversary outputs a message $\hat{m} \in \mathcal{M}$.

Let W be the event that $\hat{m} = m$. We say that $\mathcal{A}$ wins the game in this case, and we define $\mathcal{A}$'s **message recovery advantage** with respect to $\mathcal{E}$ as

$$\text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \bigl|\Pr[W] - 1/|\mathcal{M}|\bigr|. \quad \Box$$

Definition 2.3 (security against message recovery). *A cipher $\mathcal{E}$ is* ***secure against message recovery*** *if for all efficient adversaries $\mathcal{A}$, the value $\text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible.*

Theorem 2.7. *Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. If $\mathcal{E}$ is semantically secure then $\mathcal{E}$ is secure against message recovery.*

Proof. Assume that $\mathcal{E}$ is semantically secure. Our goal is to show that $\mathcal{E}$ is secure against message recovery.

To prove that $\mathcal{E}$ is secure against message recovery, we have to show that every efficient adversary $\mathcal{A}$ has negligible advantage in Attack Game 2.2. To show this, we let an arbitrary but efficient adversary $\mathcal{A}$ be given, and our goal now is to show that $\mathcal{A}$'s message recovery advantage, $\text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, is negligible. Let p denote the probability that $\mathcal{A}$ wins the message recovery game, so that

$$\text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = \big|p - 1/|\mathcal{M}|\big|.$$

We shall show how to construct an efficient adversary $\mathcal{B}$ whose semantic security advantage in Attack Game 2.1 is related to $\mathcal{A}$'s message recovery advantage as follows:

$$\text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{2.5}$$

Since $\mathcal{B}$ is efficient, and since we are assuming that $\mathcal{E}$ is semantically secure, the right-hand side of (2.5) is negligible, and so we conclude that $\text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible.

So all that remains to complete the proof is to show how to construct an efficient $\mathcal{B}$ that satisfies (2.5). The idea is to use $\mathcal{A}$ as a "black box" — we do not have to understand the inner workings of $\mathcal{A}$ at all.

Here is how $\mathcal{B}$ works. Adversary $\mathcal{B}$ generates two random messages, m_0 and m_1 in $\mathcal{M}$, and sends these to its own SS challenger. This challenger sends $\mathcal{B}$ a ciphertext c, which $\mathcal{B}$ forwards to $\mathcal{A}$, *as if it were coming from $\mathcal{A}$'s MR challenger.* When $\mathcal{A}$ outputs a message $\hat{m}$, our adversary $\mathcal{B}$ outputs $\hat{b} = 1$ if $\hat{m} = m_1$, and outputs $\hat{b} = 0$ otherwise.

That completes the description of $\mathcal{B}$. Note that the running time of $\mathcal{B}$ is essentially the same as that of $\mathcal{A}$. We now analyze the $\mathcal{B}$'s SS advantage, and relate this to $\mathcal{A}$'s MR advantage.

For $b = 0, 1$, let p_b be the probability that $\mathcal{B}$ outputs 1 if $\mathcal{B}$'s SS challenger encrypts m_b. So by definition

$$\text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0|.$$

On the one hand, when c is an encryption of m_1, the probability p_1 is precisely equal to $\mathcal{A}$'s probability of winning the message recovery game, so $p_1 = p$. On the other hand, when c is an encryption of m_0, the adversary $\mathcal{A}$'s output is independent of m_1, and so $p_0 = 1/|\mathcal{M}|$. It follows that

$$\text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0| = \big|p - 1/|\mathcal{M}|\big| = \text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}].$$

This proves (2.5). In fact, equality holds in (2.5), but that is not essential to the proof. □

The reader should make sure that he or she understands the logic of this proof, as this type of proof will be used over and over again throughout the book. We shall review the important parts of the proof here, and give another way of thinking about it.

The core of the proof was establishing the following fact: for every efficient MR adversary $\mathcal{A}$ that attacks $\mathcal{E}$ as in Attack Game 2.2, there exists an efficient SS adversary $\mathcal{B}$ that attacks $\mathcal{E}$ as in Attack Game 2.1 such that

$$\text{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{2.6}$$

We are trying to prove that if $\mathcal{E}$ is semantically secure, then $\mathcal{E}$ is secure against message recovery. In the above proof, we argued that if $\mathcal{E}$ is semantically secure, then the right-hand side of (2.6) must be negligible, and hence so must the left-hand side; since this holds for all efficient $\mathcal{A}$, we conclude that $\mathcal{E}$ is secure against message recovery.

Another way to approach the proof of the theorem is to prove the contrapositive: if $\mathcal{E}$ is *not* secure against message recovery, then $\mathcal{E}$ is *not* semantically secure. So, let us assume that $\mathcal{E}$ is not secure against message recovery. This means there exists an efficient adversary $\mathcal{A}$ whose message recovery advantage is non-negligible. Using $\mathcal{A}$ we build an efficient adversary $\mathcal{B}$ that satisfies (2.6). By assumption, $\mathrm{MR}\mathsf{adv}[\mathcal{A},\mathcal{E}]$ is non-negligible, and (2.6) implies that $\mathrm{SS}\mathsf{adv}[\mathcal{B},\mathcal{E}]$ is non-negligible. From this, we conclude that $\mathcal{E}$ is not semantically secure.

Said even more briefly: to prove that semantic security implies security against message recovery, we show how to turn an efficient adversary that breaks message recovery into an efficient adversary that breaks semantic security.

We also stress that the adversary $\mathcal{B}$ constructed in the proof just uses $\mathcal{A}$ as a "black box." In fact, almost all of the constructions we shall see are of this type: $\mathcal{B}$ is essentially just a wrapper around $\mathcal{A}$, consisting of some simple and efficient "interface layer" between $\mathcal{B}$'s challenger and a single running instance of $\mathcal{A}$. Ideally, we want the computational complexity of the interface layer to not depend on the computational complexity of $\mathcal{A}$; however, some dependence is unavoidable: if an attack game allows $\mathcal{A}$ to make multiple queries to its challenger, the more queries $\mathcal{A}$ makes, the more work must be performed by the interface layer, but this work should just depend on the number of such queries and not on the running time of $\mathcal{A}$.

Thus, we will say adversary $\mathcal{B}$ is an **elementary wrapper** around adversary $\mathcal{A}$ when it can be structured as above, as an efficient interface interacting with $\mathcal{A}$. The salient properties are:

- If $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, and $\mathcal{A}$ is efficient, then $\mathcal{B}$ is efficient.
- If $\mathcal{C}$ is an elementary wrapper around $\mathcal{B}$ and $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, then $\mathcal{C}$ is an elementary wrapper around $\mathcal{A}$.

These notions are formalized in Section 2.3 (but again, they are extremely tedious).

2.2.3.2 Computing individual bits of a message

If an encryption scheme is secure, not only should it be hard to recover the whole message, but it should be hard to compute any partial information about the message.

We will not prove a completely general theorem here, but rather, consider a specific example.

Suppose $\mathcal{E} = (E, D)$ is a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{M} = \{0,1\}^L$. For $m \in \mathcal{M}$, we define $\mathrm{parity}(m)$ to be 1 if the number of 1's in m is odd, and 0 otherwise. Equivalently, $\mathrm{parity}(m)$ is the exclusive-OR of all the individual bits of m.

We will show that if $\mathcal{E}$ is semantically secure, then given an encryption c of a random message m, it is hard to predict $\mathrm{parity}(m)$. Now, since $\mathrm{parity}(m)$ is a single bit, any adversary can predict this value correctly with probability $1/2$ just by random guessing. But what we want to show is that no efficient adversary can do significantly better than random guessing.

As a warm up, suppose there were an efficient adversary $\mathcal{A}$ that could predict $\mathrm{parity}(m)$ with probability 1. This means that for every message m, every key k, and every encryption c of m, when we give $\mathcal{A}$ the ciphertext c, it outputs the parity of m. So we could use $\mathcal{A}$ to build an SS adversary $\mathcal{B}$ that works as follows. Our adversary chooses two messages, m_0 and m_1, arbitrarily, but with $\text{parity}(m_0) = 0$ and $\text{parity}(m_1) = 1$. Then it hands these two messages to its own SS challenger, obtaining a ciphertext c, which it then forwards to $\mathcal{A}$. After receiving c, adversary $\mathcal{A}$ outputs a bit $\hat{b}$, and $\mathcal{B}$ outputs this same bit $\hat{b}$ as its own output. It is easy to see that $\mathcal{B}$'s SS advantage is precisely 1: when its SS challenger encrypts m_0, it always outputs 0, and when its SS challenger encrypts m_1, it always outputs 1.

This shows that if $\mathcal{E}$ is semantically secure, there is no efficient adversary that can predict parity with probability 1. However, we can say even more: if $\mathcal{E}$ is semantically secure, there is no efficient adversary that can predict parity with probability significantly better than 1/2. To make this precise, we give an attack game:

Attack Game 2.3 (parity prediction). For a given cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, the attack game proceeds as follows:

- The challenger computes $m \overset{\text{R}}{\leftarrow} \mathcal{M}$, $k \overset{\text{R}}{\leftarrow} \mathcal{K}$, $c \overset{\text{R}}{\leftarrow} E(k, m)$, and sends c to the adversary.
- The adversary outputs $\hat{b} \in \{0, 1\}$.

Let W be the event that $\hat{b} = \text{parity}(m)$. We define $\mathcal{A}$'s **parity prediction advantage** with respect to $\mathcal{E}$ as

$$\text{Parity}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W] - 1/2\Big|. \quad \Box$$

Definition 2.4 (parity prediction). *A cipher $\mathcal{E}$ is* ***secure against parity prediction*** *if for all efficient adversaries $\mathcal{A}$, the value* $\text{Parity}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Theorem 2.8. *Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and $\mathcal{M} = \{0, 1\}^L$. If $\mathcal{E}$ is semantically secure, then $\mathcal{E}$ is secure against parity prediction.*

Proof. As in the proof of Theorem 2.7, we give a proof by reduction. In particular, we will show that for every parity prediction adversary $\mathcal{A}$ that attacks $\mathcal{E}$ as in Attack Game 2.3, there exists an SS adversary $\mathcal{B}$ that attacks $\mathcal{E}$ as in Attack Game 2.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{Parity}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = \frac{1}{2} \cdot \mathsf{SSadv}[\mathcal{B}, \mathcal{E}].$$

Let $\mathcal{A}$ be a parity prediction adversary that predicts parity with probability $1/2 + \epsilon$, so $\text{Parity}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = |\epsilon|$.

Here is how we construct our SS adversary $\mathcal{B}$.

Our adversary $\mathcal{B}$ generates a random message m_0, and sets $m_1 \leftarrow m_0 \oplus (0^{L-1} \parallel 1)$; that is, m_1 is the same as m_0, except that the last bit is flipped. In particular, m_0 and m_1 have opposite parity.

Our adversary $\mathcal{B}$ sends the pair m_0, m_1 to its own SS challenger, receives a ciphertext c from that challenger, and forwards c to $\mathcal{A}$. When $\mathcal{A}$ outputs a bit $\hat{b}$, our adversary $\mathcal{B}$ outputs 1 if $\hat{b} = \text{parity}(m_0)$, and outputs 0, otherwise.

For $b = 0, 1$, let p_b be the probability that $\mathcal{B}$ outputs 1 if $\mathcal{B}$'s SS challenger encrypts m_b. So by definition

$$\mathsf{SSadv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0|.$$

We claim that $p_0 = 1/2 + \epsilon$ and $p_1 = 1/2 - \epsilon$. This is because regardless of whether m_0 or m_1 is encrypted, the distribution of m_b is uniform over $\mathcal{M}$, and so in case $b = 0$, our parity predictor $\mathcal{A}$ will output $\text{parity}(m_0)$ with probability $1/2 + \epsilon$, and when $b = 1$, our parity predictor

$\mathcal{A}$ will output parity(m_1) with probability $1/2 + \epsilon$, and so outputs parity(m_0) with probability $1 - (1/2 + \epsilon) = 1/2 - \epsilon$.

Therefore,

$$\text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0| = 2|\epsilon| = 2 \cdot \text{Parity}\mathsf{adv}[\mathcal{A}, \mathcal{E}],$$

which proves the theorem. □

We have shown that if an adversary can effectively predict the parity of a message, then it can be used to break semantic security. Conversely, it turns out that if an adversary can break semantic security, he can effectively predict *some* predicate of the message (see Exercise 3.16).

2.2.4 Consequences of semantic security

In this section, we examine the consequences of semantic security in the context of a specific example, namely, electronic gambling. The specific details of the example are not so important, but the example illustrates how one typically uses the assumption of semantic security in applications.

Consider the following extremely simplified version of roulette, which is a game between the *house* and a *player*. The player gives the house 1 dollar. He may place one of two kinds of bets:

- “high or low,” or
- “even or odd.”

After placing his bet, the house chooses a random number $r \in \{0, 1, \ldots, 36\}$. The player wins if $r \neq 0$, and if

- he bet “high” and $r > 18$,
- he bet “low” and $r \leq 18$,
- he bet “even” and r is even,
- he bet “odd” and r is odd.

If the player wins, the house pays him 2 dollars (for a net win of 1 dollar), and if the player loses, the house pays nothing (for a net loss of 1 dollar). Clearly, the house has a small, but not insignificant advantage in this game: the probability that the player wins is $18/37 \approx 48.65\%$.

Now suppose that this game is played over the Internet. Also, suppose that for various technical reasons, the house publishes an encryption of r *before* the player places his bet (perhaps to be decrypted by some regulatory agency that shares a key with the house). The player is free to analyze this encryption before placing his bet, and of course, by doing so, the player could conceivably increase his chances of winning. However, if the cipher is any good, the player’s chances should not increase by much. Let us prove this, assuming r is encrypted using a semantically secure cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{M} = \{0, 1, \ldots, 36\}$ (we shall view all messages in $\mathcal{M}$ as having the same length in this example). Also, from now on, let us call the player $\mathcal{A}$, to stress the adversarial nature of the player, and assume that $\mathcal{A}$’s strategy can be modeled as an efficient algorithm. The game is illustrated in Fig. 2.2. Here, *bet* denotes one of “high,” “low,” “even,” “odd.” Player $\mathcal{A}$ sends *bet* to the house, who evaluates the function $W(r, \mathit{bet})$, which is 1 if *bet* is a winning bet with respect to r, and 0 otherwise. Let us define

$$\text{IR}\mathsf{adv}[\mathcal{A}] := \big|\Pr[W(r, \mathit{bet}) = 1] - 18/37\big|.$$

Our goal is to prove the following theorem.

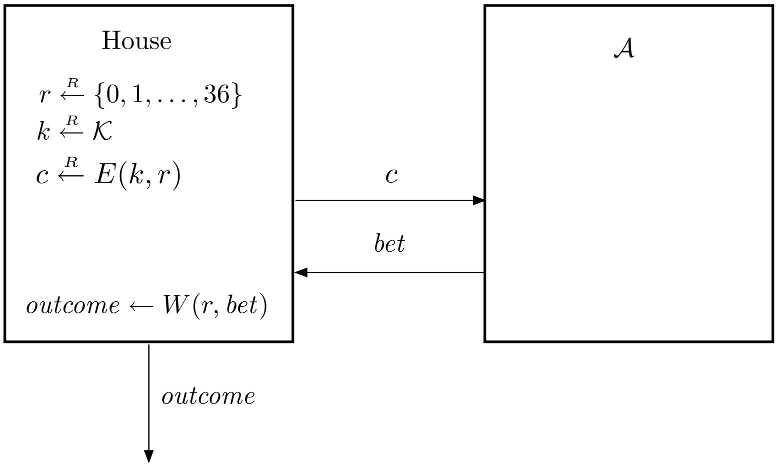

House
$r \xleftarrow{R} \{0,1,\ldots,36\}$
$k \xleftarrow{R} \mathcal{K}$
$c \xleftarrow{R} E(k,r)$
c
bet
$outcome \leftarrow W(r, bet)$
$\mathcal{A}$
$outcome$

Figure 2.2: Internet roulette

Theorem 2.9. *If $\mathcal{E}$ is semantically secure, then for every efficient player $\mathcal{A}$, the quantity* $\mathrm{IR}\mathsf{adv}[\mathcal{A}]$ *is negligible.*

As we did in Section 2.2.3, we prove this by reduction. More concretely, we shall show that for every player $\mathcal{A}$, there exists an SS adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{IR}\mathsf{adv}[\mathcal{A}] = \mathrm{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{2.7}$$

Thus, if there were an efficient player $\mathcal{A}$ with a non-negligible advantage, we would obtain an efficient SS adversary $\mathcal{B}$ that breaks the semantic security of $\mathcal{E}$, which we are assuming is impossible. Therefore, there is no such $\mathcal{A}$.

To motivate and analyze our new adversary $\mathcal{B}$, consider an "idealized" version of Internet roulette, in which instead of publishing an encryption of the actual value r, the house instead publishes an encryption of a "dummy"value, say 0. The logic of the ideal Internet roulette game is illustrated in Fig. 2.3. Note, however, that in the ideal Internet roulette game, the house still uses the *actual* value of r to determine the outcome of the game. Let p_0 be the probability that $\mathcal{A}$ wins at Internet roulette, and let p_1 be the probability that $\mathcal{A}$ wins at ideal Internet roulette.

Our adversary $\mathcal{B}$ is designed to play in Attack Game 2.1 so that if $\hat{b}$ denotes $\mathcal{B}$'s output in that game, then we have:

- if $\mathcal{B}$ is placed in Experiment 0, then $\Pr[\hat{b} = 1] = p_0$;
- if $\mathcal{B}$ is placed in Experiment 1, then $\Pr[\hat{b} = 1] = p_1$.

The logic of adversary $\mathcal{B}$ is illustrated in Fig. 2.4. It is clear by construction that $\mathcal{B}$ satisfies the properties claimed above, and so in particular,

$$\mathrm{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0|. \tag{2.8}$$

Now, consider the probability p_1 that $\mathcal{A}$ wins at ideal Internet roulette. No matter how clever $\mathcal{A}$'s strategy is, he wins with probability 18/37, since in this ideal Internet roulette game, the value

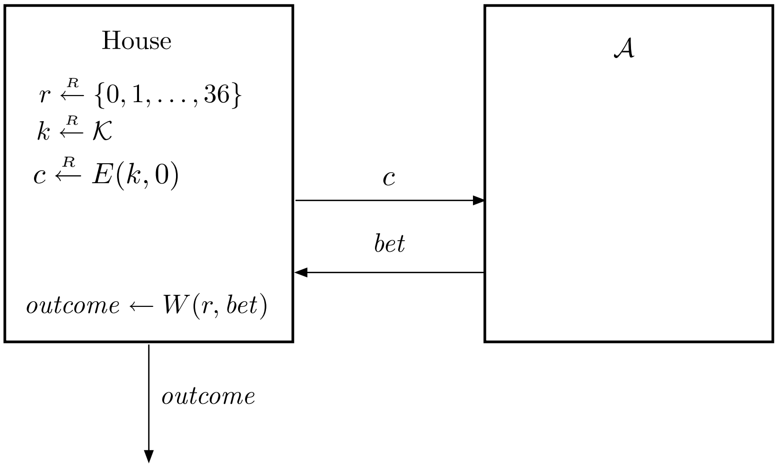

House
$r \overset{R}{\leftarrow} \{0, 1, \ldots, 36\}$
$k \overset{R}{\leftarrow} \mathcal{K}$
$c \overset{R}{\leftarrow} E(k, 0)$
$outcome \leftarrow W(r, bet)$
$\mathcal{A}$
c
bet
$outcome$

Figure 2.3: ideal Internet roulette

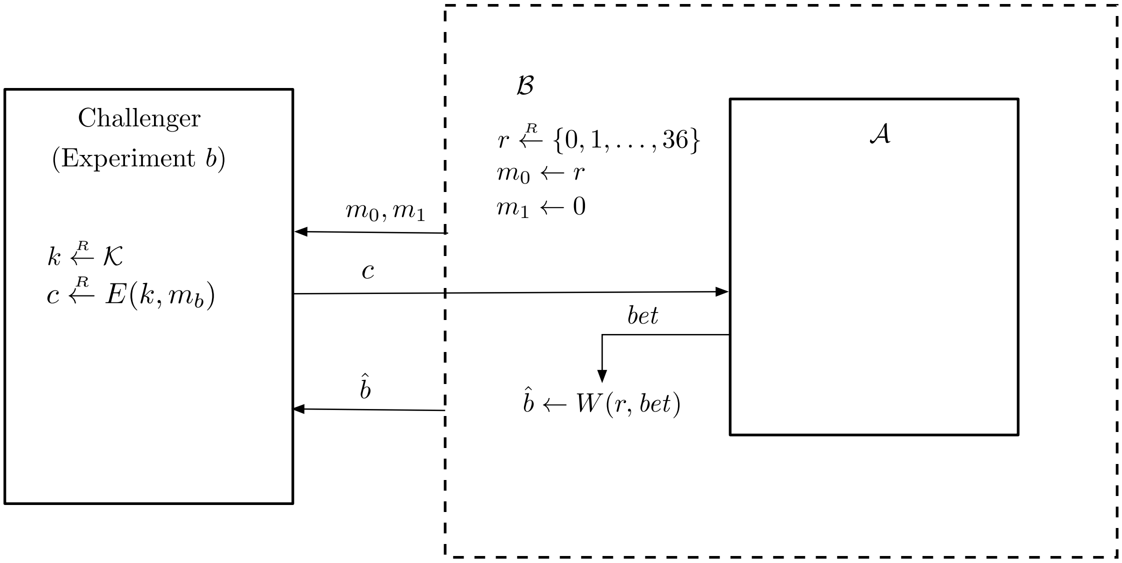

$\mathcal{B}$
$r \overset{R}{\leftarrow} \{0, 1, \ldots, 36\}$
$m_0 \leftarrow r$
$m_1 \leftarrow 0$
$\mathcal{A}$
Challenger
(Experiment b)
$k \overset{R}{\leftarrow} \mathcal{K}$
$c \overset{R}{\leftarrow} E(k, m_b)$
m_0, m_1
c
bet
$\hat{b}$
$\hat{b} \leftarrow W(r, bet)$

Figure 2.4: The SS adversary $\mathcal{B}$ in Attack Game 2.1

of *bet* is computed from c, which is statistically independent of the value of r. That is, ideal Internet roulette is equivalent to physical roulette. Therefore,

$$\text{IR}\mathsf{adv}[\mathcal{A}] = |p_1 - p_0|. \tag{2.9}$$

Combining (2.8) and (2.9), we obtain (2.7).

The approach we have used to analyze Internet roulette is one that we will see again and again. The basic idea is to replace a system component by an idealized version of that component, and then analyze the behavior of this new, idealized version of the system.

Another lesson to take away from the above example is that in reasoning about the security of a system, what we view as "the adversary" depends on what we are trying to do. In the above analysis, we cobbled together a new adversary $\mathcal{B}$ out of several components: one component was the original adversary $\mathcal{A}$, while other components were scavenged from other parts of the system (the algorithm of "the house," in this example). This will be very typical in our security analyses throughout this text. Intuitively, if we imagine a diagram of the system, at different points in the security analysis, we will draw a circle around different components of the system to identify what we consider to be "the adversary" at that point in the analysis.

2.2.5 Bit guessing: an alternative characterization of semantic security

The example in Section 2.2.4 was a typical example of how one could use the definition of semantic security to analyze the security properties of a larger system that makes use of a semantically secure cipher. However, there is another characterization of semantic security that is typically more convenient to work with when one is trying to prove that a given cipher satisfies the definition. In this alternative characterization, we define a new attack game. The role played by the adversary is exactly the same as before. However, instead of having two different experiments, there is just a single experiment. In this **bit-guessing version** of the attack game, the challenger chooses $b \in \{0, 1\}$ at random and runs Experiment b of Attack Game 2.1; it is the adversary's goal to guess the bit b with probability significantly better than 1/2. Here are the details:

Attack Game 2.4 (semantic security: bit-guessing version). For a given cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, the attack game runs as follows:

- The adversary computes $m_0, m_1 \in \mathcal{M}$, of the same length, and sends them to the challenger.
- The challenger computes $b \xleftarrow{\text{R}} \{0, 1\}$, $k \xleftarrow{\text{R}} \mathcal{K}$, $c \xleftarrow{\text{R}} E(k, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

We say that $\mathcal{A}$ **wins** the game if $\hat{b} = b$. □

Fig. 2.5 illustrates Attack Game 2.4. Note that in this game, the event that the $\mathcal{A}$ wins the game is defined with respect to the probability space determined by the random choice of b and k, the random choices made (if any) of the encryption algorithm, and the random choices made (if any) by the adversary.

Of course, any adversary can win the game with probability 1/2, simply by ignoring c completely and choosing $\hat{b}$ at random (or alternatively, always choosing $\hat{b}$ to be 0, or always choosing it to be 1). What we are interested in is how much *better* than random guessing an adversary can do. If W denotes the event that the adversary wins the bit-guessing version of the attack game, then we are interested in the quantity $|\Pr[W] - 1/2|$, which we denote by $\text{SS}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]$. Then we have:

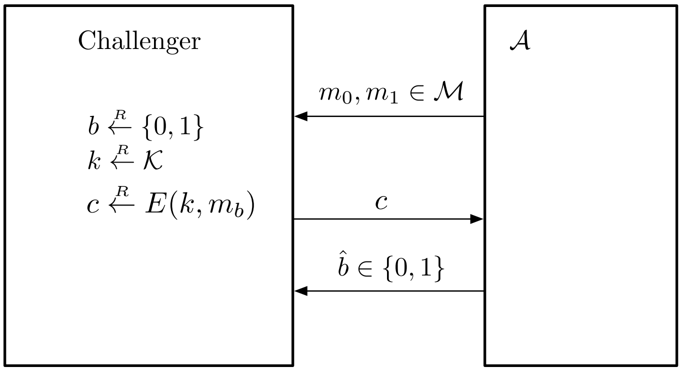

Challenger
$b \xleftarrow{R} \{0,1\}$
$k \xleftarrow{R} \mathcal{K}$
$c \xleftarrow{R} E(k, m_b)$
$\mathcal{A}$
$m_0, m_1 \in \mathcal{M}$
c
$\hat{b} \in \{0,1\}$

Figure 2.5: Attack Game 2.4

Theorem 2.10. *For every cipher $\mathcal{E}$ and every adversary $\mathcal{A}$, we have*

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]. \tag{2.10}$$

Proof. This is just a simple calculation. Let p_0 be the probability that the adversary outputs 1 in Experiment 0 of Attack Game 2.1, and let p_1 be the probability that the adversary outputs 1 in Experiment 1 of Attack Game 2.1.

Now consider Attack Game 2.4. From now on, all events and probabilities are with respect to this game. If we condition on the event that $b = 0$, then in this conditional probability space, all of the other random choices made by the challenger and the adversary are distributed in exactly the same way as the corresponding values in Experiment 0 of Attack Game 2.1. Therefore, if $\hat{b}$ is the output of the adversary in Attack Game 2.4, we have

$$\Pr[\hat{b} = 1 \mid b = 0] = p_0.$$

By a similar argument, we see that

$$\Pr[\hat{b} = 1 \mid b = 1] = p_1.$$

So we have

$$\begin{aligned}
\Pr[\hat{b} = b] &= \Pr[\hat{b} = b \mid b = 0] \Pr[b = 0] + \Pr[\hat{b} = b \mid b = 1] \Pr[b = 1] \\
&= \Pr[\hat{b} = 0 \mid b = 0] \cdot \tfrac{1}{2} + \Pr[\hat{b} = 1 \mid b = 1] \cdot \tfrac{1}{2} \\
&= \tfrac{1}{2}\Big(1 - \Pr[\hat{b} = 1 \mid b = 0] + \Pr[\hat{b} = 1 \mid b = 1]\Big) \\
&= \tfrac{1}{2}(1 - p_0 + p_1).
\end{aligned}$$

Therefore,

$$\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}] = \Big|\Pr[\hat{b} = b] - \tfrac{1}{2}\Big| = \tfrac{1}{2}|p_1 - p_0| = \tfrac{1}{2} \cdot \mathsf{SSadv}[\mathcal{A}, \mathcal{E}].$$

That proves the theorem. □

Just as it is convenient to refer to $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ as $\mathcal{A}$'s "SS advantage," we shall refer to $\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]$ as $\mathcal{A}$'s "bit-guessing SS advantage."

2.2.5.1 A generalization

As it turns out, the above situation is quite generic. Although we do not need it in this chapter, for future reference we indicate here how the above situation generalizes. There will be a number of situations we shall encounter where some particular security property, call it "X," for some cryptographic system, call it "$\mathcal{S}$," can be defined in terms of an attack game involving two experiments, Experiment 0 and Experiment 1, where the adversary $\mathcal{A}$'s protocol is the same in both experiments, while that of the challenger is different. For $b = 0, 1$, we define W_b to be the event that $\mathcal{A}$ outputs 1 in Experiment b, and we define

$$\mathsf{Xadv}[\mathcal{A}, \mathcal{S}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|$$

to be $\mathcal{A}$'s "X advantage." Just as above, we can always define a "bit-guessing" version of the attack game, in which the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b as its protocol. If W is the event that the adversary's output is equal to b, then we define

$$\mathsf{Xadv}^*[\mathcal{A}, \mathcal{S}] := \Big|\Pr[W] - 1/2\Big|$$

to be $\mathcal{A}$'s "bit-guessing X advantage."

Using exactly the same calculation as in the proof of Theorem 2.10, we have

$$\mathsf{Xadv}[\mathcal{A}, \mathcal{S}] = 2 \cdot \mathsf{Xadv}^*[\mathcal{A}, \mathcal{S}]. \tag{2.11}$$

2.3 Mathematical details

Up until now, we have used the terms *efficient* and *negligible* rather loosely, without a formal mathematical definition:

- we required that a computational cipher have *efficient* encryption and decryption algorithms;

- for a semantically secure cipher, we required that any *efficient* adversary have a *negligible* advantage in Attack Game 2.1.

The goal of this section is to provide precise mathematical definitions for these terms. While these definitions lead to a satisfying theoretical framework for the study of cryptography as a mathematical discipline, we should warn the reader:

- the definitions are rather complicated, requiring an unfortunate amount of notation; and

- the definitions model our intuitive understanding of these terms only very crudely.

We stress that the reader may safely skip this section without suffering a significant loss in understanding. Before marching headlong into the formal definitions, let us remind the reader of what we are trying to capture in these definitions.

- First, when we speak of an efficient encryption or decryption algorithm, we usually mean one that runs very quickly, encrypting data at a rate of, say, 10–100 computer cycles per byte of data.

- Second, when we speak of an efficient adversary, we usually mean an algorithm that runs in some large, but still feasible amount of time (and other resources). Typically, one assumes that an adversary that is trying to break a cryptosystem is willing to expend many more resources than a user of the cryptosystem. Thus, 10,000 computers running in parallel for 10 years may be viewed as an upper limit on what is feasibly computable by a determined, patient, and financially well-off adversary. However, in some settings, like the Internet roulette example in Section 2.2.4, the adversary may have a much more limited amount of time to perform its computations before they become irrelevant.

- Third, when we speak of an adversary's advantage as being negligible, we mean that it is so small that it may as well be regarded as being equal to zero for all practical purposes. As we saw in the Internet roulette example, if no efficient adversary has an advantage better than 2^{-100} in Attack Game 2.1, then no player can in practice improve his odds at winning Internet roulette by more than 2^{-100} relative to physical roulette.

Even though our intuitive understanding of the term *efficient* depends on the context, our formal definition will not make any such distinction. Indeed, we shall adopt the computational complexity theorist's habit of equating the notion of an *efficient* algorithm with that of a *(probabilistic) polynomial-time* algorithm. For better and for worse, this gives us a formal framework that is independent of the specific details of any particular model of computation.

2.3.1 Negligible, super-poly, and poly-bounded functions

We begin by defining the notions of *negligible*, *super-poly*, and *poly-bounded* functions.

Intuitively, a negligible function $f : \mathbb{Z}_{\geq 0} \to \mathbb{R}$ is one that not only tends to zero as $n \to \infty$, but does so faster than the inverse of any polynomial.

Definition 2.5. *A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is called **negligible** if for all $c \in \mathbb{R}_{>0}$ there exists $n_0 \in \mathbb{Z}_{\geq 1}$ such that for all integers $n \geq n_0$, we have $|f(n)| < 1/n^c$.*

An alternative characterization of a negligible function, which is perhaps easier to work with, is the following:

Theorem 2.11. *A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is negligible if and only if for all $c > 0$, we have*

$$\lim_{n \to \infty} f(n)n^c = 0.$$

Proof. Exercise. □

Example 2.10. Some examples of negligible functions:

$$2^{-n}, \quad 2^{-\sqrt{n}}, \quad n^{-\log n}.$$

Some examples of non-negligible functions:

$$\frac{1}{1000n^4 + n^2 \log n}, \quad \frac{1}{n^{100}}. \quad \Box$$

Once we have the term "negligible" formally defined, defining "super-poly" is easy:

Definition 2.6. *A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is called **super-poly** if $1/f$ is negligible.*

Essentially, a poly-bounded function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is one that is bounded (in absolute value) by some polynomial. Formally:

Definition 2.7. *A function $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ is called **poly-bounded**, if there exists $c, d \in \mathbb{R}_{>0}$ such that for all integers $n \geq 0$, we have $|f(n)| \leq n^c + d$.*

Note that if f is a poly-bounded function, then $1/f$ is definitely *not* a negligible function. However, as the following example illustrates, one must take care not to draw erroneous inferences.

Example 2.11. Define $f : \mathbb{Z}_{\geq 1} \to \mathbb{R}$ so that $f(n) = 1/n$ for all even integers n and $f(n) = 2^{-n}$ for all odd integers n. Then f is not negligible, and $1/f$ is neither poly-bounded nor super-poly. □

2.3.2 Computational ciphers: the formalities

Now the formalities. We begin by admitting a lie: when we said a computational cipher $\mathcal{E} = (E, D)$ is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{K}$ is the key space, $\mathcal{M}$ is the message space, and $\mathcal{C}$ is the ciphertext space, and with each of these spaces being finite sets, we were not telling the whole truth. In the mathematical model (though not always in real-world systems), we associate with $\mathcal{E}$ *families* of key, message, and ciphertext spaces, indexed by

- a **security parameter**, which is a positive integer, and is denoted by λ, and
- a **system parameter**, which is a bit string, and is denoted by Λ.

Thus, instead of just finite sets $\mathcal{K}$, $\mathcal{M}$, and $\mathcal{C}$, we have families of finite sets

$$\{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \text{and} \quad \{\mathcal{C}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

which for the purposes of this definition, we view as sets of bit strings (which may represent mathematical objects by way of some canonical encoding functions).

The idea is that when the cipher $\mathcal{E}$ is deployed, the security parameter λ is fixed to some value. Generally speaking, larger values of λ imply higher levels of security (i.e., resistance against adversaries with more computational resources), but also larger key sizes, as well as slower encryption and decryption speeds. Thus, the security parameter is like a "dial" we can turn, setting a trade-off between security and efficiency.

Once λ is chosen, a system parameter Λ is generated using an algorithm specific to the cipher. The idea is that the system parameter Λ (together with λ) gives a detailed description of a fixed instance of the cipher, with

$$(\mathcal{K}, \mathcal{M}, \mathcal{C}) = (\mathcal{K}_{\lambda,\Lambda},\ \mathcal{M}_{\lambda,\Lambda},\ \mathcal{C}_{\lambda,\Lambda}).$$

This one, fixed instance may be deployed in a larger system and used by many parties — the values of λ and Λ are public and known to everyone (including the adversary).

Example 2.12. Consider the additive one-time pad discussed in Example 2.4. This cipher was described in terms of a modulus n. To deploy such a cipher, a suitable modulus n is generated, and is made public (possibly just "hardwired" into the software that implements the cipher). The modulus n is the system parameter for this cipher. Each specific value of the security parameter determines the length, in bits, of n. The value n itself is generated by some algorithm that may be probabilistic and whose output distribution may depend on the intended application. For example, we may want to insist that n is a prime in some applications. □

Before going further, we define the notion of an efficient algorithm. For the purposes of this definition, we shall only consider algorithms A that take as input a security parameter λ, as well as other parameters whose total length is bounded by some fixed polynomial in λ. Basically, we want to say that the running time of A is bounded by a polynomial in λ, but things are complicated if A is probabilistic:

Definition 2.8 (efficient algorithm). *Let A be an algorithm (possibly probabilistic) that takes as input a security parameter $\lambda \in \mathbb{Z}_{\geq 1}$, as well as other parameters encoded as a bit string $x \in \{0,1\}^{\leq p(\lambda)}$ for some fixed polynomial p. We call A an* ***efficient algorithm*** *if there exist a poly-bounded function t and a negligible function ϵ such that for all $\lambda \in \mathbb{Z}_{\geq 1}$, and all $x \in \{0,1\}^{\leq p(\lambda)}$, the probability that the running time of A on input (λ, x) exceeds $t(\lambda)$ is at most $\epsilon(\lambda)$.*

We stress that the probability in the above definition is with respect to the coin tosses of A: this bound on the probability must hold for all possible inputs x.[1]

Here is a formal definition that captures the basic requirements of systems that are parameterized by a security and system parameter, and introduces some more terminology. In the following definition we use the notation $\mathrm{Supp}(P(\lambda))$ to refer to the **support** of the distribution $P(\lambda)$, which is the set of all possible outputs of algorithm P on input λ.

Definition 2.9. *A* ***system parameterization*** *is an efficient probabilistic algorithm P that given a security parameter $\lambda \in \mathbb{Z}_{\geq 1}$ as input, outputs a bit string Λ, called a* ***system parameter****, whose length is always bounded by a polynomial in λ. We also define the following terminology:*

- *A collection $\mathbf{S} = \{\mathcal{S}_{\lambda,\Lambda}\}_{\lambda,\Lambda}$ of finite sets of bits strings, where λ runs over $\mathbb{Z}_{\geq 1}$ and Λ runs over $\mathrm{Supp}(P(\lambda))$, is called a* ***family of spaces with system parameterization*** *P, provided the lengths of all the strings in each of the sets $\mathcal{S}_{\lambda,\Lambda}$ are bounded by some polynomial p in λ.*

- *We say that $\mathbf{S}$ is* ***efficiently recognizable*** *if there is an efficient* deterministic *algorithm that on input $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, and $s \in \{0,1\}^{\leq p(\lambda)}$, determines if $s \in \mathcal{S}_{\lambda,\Lambda}$.*

- *We say that $\mathbf{S}$ is* ***efficiently sampleable*** *if there is an efficient probabilistic algorithm that on input $\lambda \in \mathbb{Z}_{\geq 1}$ and $\Lambda \in \mathrm{Supp}(P(\lambda))$, outputs an element uniformly distributed over $\mathcal{S}_{\lambda,\Lambda}$.*

- *We say that $\mathbf{S}$* ***has an effective length function*** *if there is an efficient* deterministic *algorithm that on input $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, and $s \in \mathcal{S}_{\lambda,\Lambda}$, outputs a non-negative integer, called the* ***length*** *of s.*

We can now state the complete, formal definition of a computational cipher:

[1] By not insisting that a probabilistic algorithm halts in a specified time bound with probability 1, we give ourselves a little "wiggle room," which allows us to easily do certain types of random sampling procedure that have no *a priori* running time bound, but are very unlikely to run for too long (e.g., think of flipping a coin until it comes up "heads"). An alternative approach would be to bound the *expected* running time, but this turns out to be somewhat problematic for technical reasons.

Note that this definition of an efficient algorithm does not require that the algorithm halt with probability 1 on all inputs. An algorithm that with probability $2^{-\lambda}$ entered an infinite loop would satisfy the definition, even though it does not halt with probability 1. These issues are rather orthogonal. In general, we shall only consider algorithms that halt with probability 1 on all inputs: this can more naturally be seen as a requirement on the output distribution of the algorithm, rather than on its running time.

Definition 2.10 (computational cipher). *A **computational cipher** consists of a pair of algorithms E and D, along with three families of spaces with system parameterization P:*

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \textit{and} \quad \mathbf{C} = \{\mathcal{C}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

such that

1. $\mathbf{K}$, $\mathbf{M}$, *and* $\mathbf{C}$ *are efficiently recognizable.*

2. $\mathbf{K}$ *is efficiently sampleable.*

3. $\mathbf{M}$ *has an effective length function.*

4. *Algorithm E is an efficient probabilistic algorithm that on input λ, Λ, k, m, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, and $m \in \mathcal{M}_{\lambda,\Lambda}$, always outputs an element of $\mathcal{C}_{\lambda,\Lambda}$.*

5. *Algorithm D is an efficient* deterministic *algorithm that on input λ, Λ, k, c, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, and $c \in \mathcal{C}_{\lambda,\Lambda}$, outputs either an element of $\mathcal{M}_{\lambda,\Lambda}$, or a special symbol* reject $\notin \mathcal{M}_{\lambda,\Lambda}$.

6. *For all $\lambda, \Lambda, k, m, c$, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, $m \in \mathcal{M}_{\lambda,\Lambda}$, and $c \in \mathrm{Supp}(E(\lambda, \Lambda; k, m))$, we have $D(\lambda, \Lambda; k, c) = m$.*

Note that in the above definition, the encryption and decryption algorithms take λ and Λ as auxiliary inputs. So as to be somewhat consistent with the notation already introduced in Section 2.2.1, we write this as $E(\lambda, \Lambda; \cdots)$ and $D(\lambda, \Lambda; \cdots)$.

Example 2.13. Consider the additive one-time pad (see Example 2.12). In our formal framework, the security parameter λ determines the bit length $L(\lambda)$ of the modulus n, which is the system parameter. The system parameter generation algorithm takes as input λ and generates a modulus n of length $L(\lambda)$. The function $L(\cdot)$ should be polynomially bounded. With this assumption, it is clear that the system parameter generation algorithm satisfies its requirements. The requirements on the key, message, and ciphertext spaces are also satisfied:

1. Elements of these spaces have polynomially bounded lengths: this again follows from our assumption that $L(\cdot)$ is polynomially bounded.

2. The key space is efficiently sampleable: just choose $k \xleftarrow{\mathrm{R}} \{0, \ldots, n-1\}$.

3. The key, message, and ciphertext spaces are efficiently recognizable: just test if a bit string s is the binary encoding of an integer between 0 and $n - 1$.

4. The message space also has an effective length function: just output (say) 0. □

We note that some ciphers (for example the one-time pad) may not need a system parameter. In this case, we can just pretend that the system parameter is, say, the empty string. We also note that some ciphers do not really have a security parameter either; indeed, many industry-standard ciphers simply come ready-made with a fixed key size, with no security parameter that can be tuned. This is simply mismatch between theory and practice — that is just the way it is.

That completes our formal mathematical description of a computational cipher, in all its glorious detail.[2] The reader should hopefully appreciate that while these formalities may allow us to make mathematically precise and meaningful statements, they are not very enlightening, and mostly serve to obscure what is really going on. Therefore, in the main body of the text, we will continue to discuss ciphers using the simplified terminology and notation of Section 2.2.1, with the understanding that all statements made have a proper and natural interpretation in the formal framework discussed in this section. This will be a pattern that is repeated in the sequel: we shall mainly discuss various types of cryptographic schemes using a simplified terminology, without mention of security parameters and system parameters — these mathematical details will be discussed in a separate section, but will generally follow the same general pattern established here.

2.3.3 Efficient adversaries and attack games

In defining the notion of semantic security, we have to define what we mean by an *efficient adversary*. Since this concept will be used extensively throughout the text, we present a more general framework here.

For any type of cryptographic scheme, security will be defined using an attack game, played between an adversary $\mathcal{A}$ and a challenger: $\mathcal{A}$ follows an arbitrary protocol, while the challenger follows some simple, fixed protocol determined by the cryptographic scheme and the notion of security under discussion. Furthermore, both adversary and challenger take as input a common security parameter λ, and the challenger starts the game by computing a corresponding system parameter Λ, and sending this to the adversary.

To model these types of interactions, we introduce the notion of an **interactive machine**. Before such a machine M starts, it always gets the security parameter λ written in a special buffer, and the rest of its internal state is initialized to some default value. Machine M has two other special buffers: an *incoming message buffer* and an *outgoing message buffer*. Machine M may be invoked many times: each invocation starts when M's external environment writes a string to M's incoming message buffer; M reads the message, performs some computation, updates its internal state, and writes a string on its outgoing message buffer, ending the invocation, and the outgoing message is passed to the environment. Thus, M interacts with its environment via a simple message passing system. We assume that M may indicate that it has halted by including some signal in its last outgoing message, and M will essentially ignore any further attempts to invoke it.

We shall assume messages to and from the machine M are restricted to be of *constant length*. This is not a real restriction: we can always simulate the transmission of one long message by sending many shorter ones. However, making a restriction of this type simplifies some of the technicalities. We assume this restriction from now on, for adversaries as well as for any other type of interactive machine.

For any given environment, we can measure the **total running time** of M by counting the number of steps it performs across all invocations until it signals that it has halted. This running time depends not only on M and its random choices, but also on the environment in which M runs.[3]

[2] Note that the definition of a Shannon cipher in Section 2.1.1 remains unchanged. The claim made at the end of Section 2.2.1 that any deterministic computational cipher is also a Shannon cipher needs to be properly interpreted: for each λ and Λ, we get a Shannon cipher defined over $(\mathcal{K}_{\lambda,\Lambda}, \mathcal{M}_{\lambda,\Lambda}, \mathcal{C}_{\lambda,\Lambda})$.

[3] Analogous to the discussion in footnote 1 on page 30, our definition of an efficient interactive machine will not require that it halts with probability 1 for all environments. This is an orthogonal issue, but it will be an implicit

Definition 2.11 (efficient interactive machine). *We say that M is an* ***efficient interactive machine*** *if there exist a poly-bounded function t and a negligible function ϵ, such that for all environments (not even computationally unbounded ones), the probability that the total running time of M exceeds $t(\lambda)$ is at most $\epsilon(\lambda)$.*

We naturally model an adversary as an interactive machine. An **efficient adversary** is simply an efficient interactive machine.

We can connect two interactive machines together, say M' and M, to create a new interactive machine $M'' = \langle M', M \rangle$. Messages from the environment to M'' always get routed to M'. The machine M' may send a message to the environment, or to M; in the latter case, the output message sent by M gets sent to M'. We assume that if M halts, then M' does not send it any more messages.

Thus, when M'' is invoked, its incoming message is routed to M', and then M' and M may interact some number of times, and then the invocation of M'' ends when M' sends a message to the environment. We call M' the "open" machine (which interacts with the outside world), and M the "closed" machine (which interacts only with M').

Naturally, we can model the interaction of a challenger and an adversary by connecting two such machines together as above: the challenger becomes the open machine, and the adversary becomes the closed machine.

In our security reductions, we typically show how to use an adversary $\mathcal{A}$ that breaks some system to build an adversary $\mathcal{B}$ that breaks some other system. The essential property that we want is that if $\mathcal{A}$ is efficient, then so is $\mathcal{B}$. However, our reductions are almost always of a very special form, where $\mathcal{B}$ is a wrapper around $\mathcal{A}$, consisting of some simple and efficient "interface layer" between $\mathcal{B}$'s challenger and a single running instance of $\mathcal{A}$.

Ideally, we want the computational complexity of the interface layer to not depend on the computational complexity of $\mathcal{A}$; however, some dependence is unavoidable: the more queries $\mathcal{A}$ makes to its challenger, the more work must be performed by the interface layer, but this work should just depend on the number of such queries and not on the running time of $\mathcal{A}$.

To formalize this, we build $\mathcal{B}$ as a composed machine $\langle M', M \rangle$, where M' represents the interface layer (the "open" machine), and M represents the instance of $\mathcal{A}$ (the "closed" machine). This leads us to the following definition.

Definition 2.12 (elementary wrapper). *An interactive machine M' is called an* ***efficient interface*** *if there exists a poly-bounded function t and a negligible function ϵ, such that for all M (not necessarily computationally bounded), when we execute the composed machine $\langle M', M \rangle$ in an arbitrary environment (again, not necessarily computationally bounded), the following property holds:*

> *at every point in the execution of $\langle M', M \rangle$, if I is the number of interactions between M' and M up to at that point, and T is the total running time of M' up to that point, then the probability that $T > t(\lambda + I)$ is at most $\epsilon(\lambda)$.*

If M' is an efficient interface, and M is any machine, then we say $\langle M', M \rangle$ is an ***elementary wrapper around*** *M.*

requirement of any machines we consider.

Thus, we will say adversary $\mathcal{B}$ is an elementary wrapper around adversary $\mathcal{A}$ when it can be structured as above, as an efficient interface interacting with $\mathcal{A}$. Our definitions were designed to work well together. The salient properties are:

- If $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, and $\mathcal{A}$ is efficient, then $\mathcal{B}$ is efficient.
- If $\mathcal{C}$ is an elementary wrapper around $\mathcal{B}$ and $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, then $\mathcal{C}$ is an elementary wrapper around $\mathcal{A}$.

Also note that in our attack games, the challenger typically satisfies our definition of an efficient interface. For such a challenger and any efficient adversary $\mathcal{A}$, we can view their entire interaction as a that of a single, efficient machine.

Query bounded adversaries. In the attack games we have seen so far, the adversary makes just a fixed number of queries. Later in the text, we will see attack games in which the adversary $\mathcal{A}$ is allowed to make many queries — even though there is no a priori bound on the number of queries it is allowed to make, if $\mathcal{A}$ is efficient, the number of queries will be bounded by some poly-bounded value Q (at least with all but negligible probability). In proving security for such attack games, in designing an elementary wrapper $\mathcal{B}$ from $\mathcal{A}$, it will usually be convenient to tell $\mathcal{B}$ *in advance* an upper bound Q on how many queries $\mathcal{A}$ will ultimately make. To fit this into our formal framework, we can set things up so that $\mathcal{A}$ starts out by sending a sequence of Q special messages to "signal" this query bound to $\mathcal{B}$. If we do this, then not only can $\mathcal{B}$ use the value Q in its logic, it is also allowed to run in time that depends on Q, without violating the time constraints in Definition 2.12. This is convenient, as then $\mathcal{B}$ is allowed to initialize data structures whose size may depend on Q. Of course, all of this is just a legalistic "hack" to work around technical constraints that would otherwise be too restrictive, and should not be taken too seriously. We will never make this "signaling" explicit in any of our presentations.

2.3.4 Semantic security: the formalities

In defining any type of security, we will define the adversary's advantage in the attack game as a function $\mathrm{Adv}(\lambda)$. This will be defined in terms of probabilities of certain events in the attack game: for each value of λ we get a different probability space, determined by the random choices of the challenger, and the random choices of the adversary. Security will mean that for every efficient adversary, the function $\mathrm{Adv}(\cdot)$ is negligible.

Turning now to the specific situation of semantic security of a cipher, in Attack Game 2.1, we defined the value $\mathrm{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$. This value is actually a function of the security parameter λ. The proper interpretation of Definition 2.2 is that $\mathcal{E}$ is secure if for all efficient adversaries $\mathcal{A}$ (modeled as an interactive machine, as described above), the function $\mathrm{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}](\lambda)$ in the security parameter λ is negligible (as defined in Definition 2.5). Recall that both challenger and adversary receive λ as a common input. Control begins with the challenger, who sends the system parameter to the adversary. The adversary then sends its query to the challenger, which consists of two plaintexts, who responds with a ciphertext. Finally, the adversary outputs a bit (technically, in our formal machine model, this "output" is a message sent to the challenger, and then the challenger halts). The value of $\mathrm{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}](\lambda)$ is determined by the random choices of the challenger (including the choice of system parameter) and the random choices of the adversary. See Fig. 2.6 for a complete picture of Attack Game 2.1.

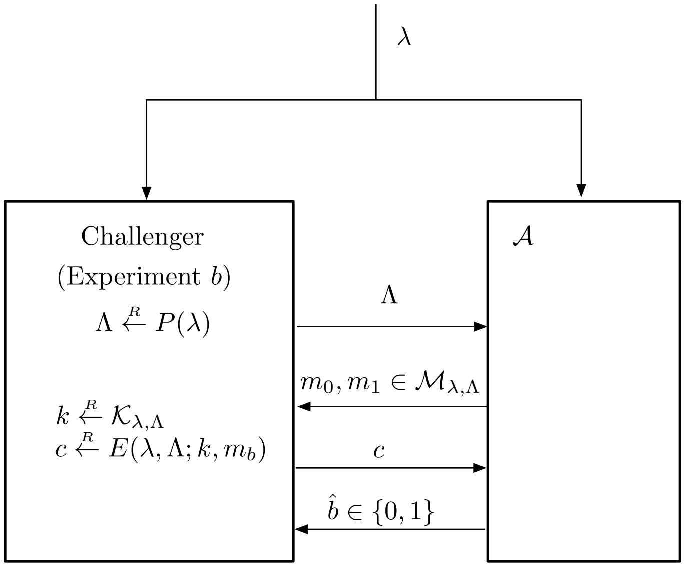

λ
Challenger
(Experiment b)
$\Lambda \overset{R}{\leftarrow} P(\lambda)$
$k \overset{R}{\leftarrow} \mathcal{K}_{\lambda,\Lambda}$
$c \overset{R}{\leftarrow} E(\lambda, \Lambda; k, m_b)$
Λ
$m_0, m_1 \in \mathcal{M}_{\lambda,\Lambda}$
c
$\hat{b} \in \{0,1\}$
$\mathcal{A}$

Figure 2.6: The fully detailed version of Attack Game 2.1

Also, in Attack Game 2.1, the requirement that the two messages presented by the adversary have the same length means that the length function provided in part 3 of Definition 2.10 evaluates to the same value on the two messages.

It is perhaps useful to see what it means for a cipher $\mathcal{E}$ to be insecure according to this formal definition. This means that there exists an adversary $\mathcal{A}$ such that $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ is a non-negligible function in the security parameter. This means that $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}](\lambda) \geq 1/\lambda^c$ for some $c > 0$ and for infinitely many values of the security parameter λ. So this does not mean that $\mathcal{A}$ can "break" $\mathcal{E}$ for all values of the security parameter, but only infinitely many values of the security parameter.

In the main body of the text, we shall mainly ignore security parameters, system parameters, and the like, but it will always be understood that all of our "shorthand" has a precise mathematical interpretation. In particular, we will often refer to certain values v as being negligible (resp., poly-bounded), which really means that v is a negligible (resp., poly-bounded) function of the security parameter.

2.4 A fun application: anonymous routing

Our friend Alice wants to send a message m to Bob, but she does not want Bob or anyone else to know that the message m is from Alice. For example, Bob might be running a public discussion forum and Alice wants to post a comment anonymously on the forum. Posting anonymously lets Alice discuss health issues or other matters without identifying herself. In this section we will assume Alice only wants to post a *single* message to the forum.

One option is for Alice to choose a proxy, Carol, send m to Carol, and ask Carol to forward the message to Bob. This clearly does not provide anonymity for Alice since anyone watching the network will see that m was sent from Alice to Carol and then from Carol to Bob. By tracing the path of m through the network anyone can see that the post came from Alice.

A better approach is for Alice to establish a shared key k with Carol and send $c := E(k, m)$ to Carol, where $\mathcal{E} = (E, D)$ is a semantically secure cipher. Carol decrypts c and forwards m to Bob. Now, someone watching the network will see one message sent from Alice to Carol and a different message sent from Carol to Bob. Nevertheless, this method still does not ensure anonymity for Alice: if on a particular day the only message that Carol receives is the one from Alice and the only message she sends goes to Bob, then an observer can link the two and still learn that the posted message came from Alice.

We solve this problem by having Carol provide a *mixing service*, that is, a service that mixes incoming messages from many different parties $A_1, \ldots, A_n$. For $i = 1, \ldots, n$, Carol establishes a secret key k_i with party A_i and each party A_i sends to Carol an encrypted message $c_i := E\big(k_i,\ \langle destination_i, m_i\rangle\big)$. Carol collects all n incoming ciphertexts, decrypts each of them with the correct key, and forwards the resulting plaintexts in some random order to their destinations. Now an observer examining Carol's traffic sees n messages going in and n messages going out, but cannot tell which message was sent where. Alice's message is one of the n messages sent out by Carol, but the observer cannot tell which one. We say that Alice's *anonymity set* is of size n.

The remaining problem is that Carol can still tell that Alice is the one who posted a specific message on the discussion forum. To eliminate this final risk Alice uses multiple mixing services, say, Carol and David. She establishes a secret key k_c with Carol and a secret key k_d with David. To send her message to Bob she constructs the following nested ciphertext c_2:

$$c_2 := E\big(k_\mathrm{c},\ E(k_\mathrm{d}, m)\big)\ . \tag{2.12}$$

For completeness Alice may want to embed routing information inside the ciphertext so that c_2 is actually constructed as:

$$c_2 := E\big(k_\mathrm{c},\ \langle \mathsf{David}, c_1\rangle\big) \qquad \text{where} \qquad c_1 := E\big(k_\mathrm{d},\ \langle \mathsf{Bob}, m\rangle\big)\ .$$

Next, Alice sends c_2 to Carol. Carol decrypts c_2 and obtains the plaintext $\langle \mathsf{David}, c_1\rangle$ which tells her to send c_1 to David. David decrypts c_1 and obtains the plaintext $\langle \mathsf{Bob}, m\rangle$ which tells him to send m to Bob. This process of decrypting a nested ciphertext, illustrated in Fig. 2.7, is similar to peeling an onion one layer at a time. For this reason this routing procedure is often called *onion routing.*

Now even if Carol observes all network traffic she cannot tell with certainty who posted a particular message on Bob's forum. The same holds for David. However, if Carol and David collude they can figure it out. For this reason Alice may want to route her message through more than two mixes. As long as one of the mixes does not collude with the others, Alice's anonymity will be preserved.

One complication is that when Alice establishes her shared secret key k_d with David, she must do so without revealing her identity to David. Otherwise, David will know that c_1 came from Alice, which we do not want. This is not difficult to do, and we will see how later in the book (Section 21.13).

Security of nested encryption. To preserve Alice's anonymity it is necessary that Carol, who knows k_c, learn no information about m from the nested ciphertext c_2 in (2.12). Otherwise, Carol could potentially use the information she learns about m from c_2 to link Alice to her post on Bob's

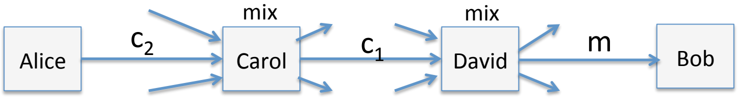


Figure 2.7: An example onion routing using two mixes

discussion forum. For example, suppose Carol could learn the first few characters of m from c_2 and later find that there is only one post on Bob's forum starting with those characters. Carol could then link the entire post to Alice because she knows that c_2 came from Alice.

The same should hold for David. David has $k_{\rm d}$, and by observing network traffic, knows that Alice sent c_2. As in the previous paragraph, it is important that David learn nothing about m from c_2.

Let us argue that if $\mathcal{E}$ is semantically secure, then no efficient adversary can learn information about m given c_2 and one of $k_{\rm c}$ or $k_{\rm d}$. More generally, for a cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, let us define the n**-way nested cipher** $\mathcal{E}_n = (E_n, D_n)$ as

$$E_n\big((k_1, \ldots, k_n),\ m\big) := E\big(k_n,\ E(k_{n-1},\ \cdots E(k_1, m))\big)\ .$$

Decryption applies the keys in the reverse order:

$$D_n\big((k_1, \ldots, k_n),\ c\big) := D\big(k_1,\ D(k_2,\ \cdots D(k_n, c))\big)\ .$$

Our goal is to show that if $\mathcal{E}$ is semantically secure then $\mathcal{E}_n$ is semantically secure even if the adversary is given all but one of the keys $k_1, \ldots, k_n$. To make this precise, we define two experiments, Experiment 0 and Experiment 1, where for $b = 0, 1$, Experiment b is:

- The adversary gives the challenger (m_0, m_1, d) where $m_0, m_1 \in \mathcal{M}$ are equal length messages and $1 \leq d \leq n$.
- The challenger chooses n keys $k_1, \ldots, k_n \overset{\rm R}{\leftarrow} \mathcal{K}$ and computes $c \overset{\rm R}{\leftarrow} E_n\big((k_1, \ldots, k_n),\ m_b\big)$. It sends c to the adversary along with all keys $k_1, \ldots, k_n$, but excluding the key k_d.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

This game captures the fact that the adversary sees all keys $k_1, \ldots, k_n$ except for k_d and tries to break semantic security.

We define the adversary's advantage, $\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, as in the definition of semantic security:

$$\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \big|\Pr[W_0] - \Pr[W_1]\big|$$

where W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, for $b = 0, 1$. We say that $\mathcal{E}$ is semantically secure for n-way nesting if $\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible.

Theorem 2.12. *For every constant $n > 0$, if $\mathcal{E} = (E, D)$ is semantically secure then $\mathcal{E}$ is semantically secure for n-way nesting.*

> *In particular, for every n-way nested adversary $\mathcal{A}$ attacking $\mathcal{E}_n$, there exists a semantic security adversary $\mathcal{B}$ attacking $\mathcal{E}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{NE}^{(\mathrm{n})}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = \mathrm{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}]\ .$$

The proof of this theorem is a good exercise in security reductions. We leave it for Exercise 2.15.

2.5 Notes

The one time pad is due to Gilbert Vernam in 1917, although there is evidence that it was discovered earlier [15].

Citations to the literature to be added.

2.6 Exercises

2.1 (multiplicative one-time pad). We may also define a "multiplication mod p" variation of the one-time pad. This is a cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{K} := \mathcal{M} := \mathcal{C} := \{1, \ldots, p-1\}$, where p is a prime. Encryption and decryption are defined as follows:

$$E(k, m) := k \cdot m \bmod p \qquad D(k, c) := k^{-1} \cdot c \bmod p.$$

Here, k^{-1} denotes the multiplicative inverse of k modulo p. Verify the correctness property for this cipher and prove that it is perfectly secure.

2.2 (A good substitution cipher). Consider a variant of the substitution cipher $\mathcal{E} = (E, D)$ defined in Example 2.3 where every symbol of the message is encrypted using an *independent* permutation. That is, let $\mathcal{M} = \mathcal{C} = \Sigma^L$ for some a finite alphabet of symbols Σ and some L. Let the key space be $\mathcal{K} = S^L$ where S is the set of all permutations on Σ. The encryption algorithm $E(k, m)$ is defined as

$$E(k, m) := \big(\ k[0](m[0]),\ k[1](m[1]),\ \ldots,\ k[L-1](m[L-1])\ \big)$$

Show that $\mathcal{E}$ is perfectly secure.

2.3 (A broken one-time pad). Consider a variant of the one time pad with message space $\{0,1\}^L$ where the key space $\mathcal{K}$ is restricted to all L-bit strings with an even number of 1's. Give an efficient adversary whose semantic security advantage is 1.

2.4 (Encryption chain). Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where $\mathcal{K} = \mathcal{M}$. Let $\mathcal{E}' = (E', D')$ be a cipher where encryption is defined as

$$E'\big((k_1, k_2), m\big) := \Big(E(k_1, k_2),\ E(k_2, m)\Big) \in \mathcal{C}^2.$$

Show that if $\mathcal{E}$ is perfectly secure then so is $\mathcal{E}'$. Exercise 3.2 describes an application for this encryption scheme.

2.5 (A stronger impossibility result). This exercise generalizes Shannon's theorem (Theorem 2.5). Let $\mathcal{E}$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Suppose that $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] \le \epsilon$ for *all* adversaries $\mathcal{A}$, even including *computationally unbounded* ones. Show that $|\mathcal{K}| \ge (1-\epsilon)|\mathcal{M}|$.

2.6 (A matching bound). This exercise develops a converse of sorts for the previous exercise. For $j = 1, \ldots, L$, let $\epsilon := 1/2^j$. Consider the L-bit one-time pad variant $\mathcal{E}$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where $\mathcal{M} = \mathcal{C} = \{0,1\}^L$. The key space $\mathcal{K}$ is restricted to all L-bit strings whose first j bits are not all zero, so that $|\mathcal{K}| = (1-\epsilon)|\mathcal{M}|$. Show that:

(a) there is an efficient adversary $\mathcal{A}$ such that $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = \epsilon/(1-\epsilon)$;

(b) for all adversaries $\mathcal{A}$, even including *computationally unbounded* ones, $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] \leq \epsilon/(1-\epsilon)$.

Note: Since the advantage of $\mathcal{A}$ in part (a) is non-zero, the cipher $\mathcal{E}$ cannot be perfectly secure.

2.7 (Deterministic ciphers). In this exercise, you are asked to prove in detail the claims made in Example 2.9. Namely, show that if $\mathcal{E}$ is a deterministic cipher that is perfectly secure, then $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 0$ for every adversary $\mathcal{A}$ (bearing in mind that $\mathcal{A}$ may be probabilistic); also show that if $\mathcal{E}$ is the variable length one-time pad, then $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 0$ for all adversaries $\mathcal{A}$.

2.8 (Roulette). In Section 2.2.4, we argued that if value r is encrypted using a semantically secure cipher, then a player's odds of winning at Internet roulette are very close to those of real roulette. However, our "roulette" game was quite simple. Suppose that we have a more involved game, where different outcomes may result in different winnings. The rules are not so important, but assume that the rules are easy to evaluate (given a bet and the number r) and that every bet results in a payout of $0, 1, \ldots, n$ dollars, where n is poly-bounded. Let μ be the expected winnings in an optimal strategy for a real version of this game (with no encryption). Let μ' be the expected winnings of some (efficient) player in an Internet version of this game (with encryption). Show that $\mu' \leq \mu + \epsilon$, where ϵ is negligible, assuming the cipher is semantically secure.

Hint: You may want to use the fact that if $\mathbf{X}$ is a random variable taking values in the set $\{0, 1, \ldots, n\}$, the expected value of $\mathbf{X}$ is equal to $\sum_{i=1}^{n} \Pr[\mathbf{X} \geq i]$.

2.9. Prove Fact 2.6, using the formal definitions in Section 2.3.

2.10 (Exercising the definition of semantic security). Let $\mathcal{E} = (E, D)$ be a semantically secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{M} = \mathcal{C} = \{0,1\}^L$. Which of the following encryption algorithms yields a semantically secure scheme? Either give an attack or provide a security proof via an explicit reduction.

(a) $E_1(k, m) := 0 \parallel E(k, m)$

(b) $E_2(k, m) := E(k, m) \parallel \text{parity}(m)$

(c) $E_3(k, m) := \text{reverse}(E(k, m))$

(d) $E_4(k, m) := E(k, \text{reverse}(m))$

Here, for a bit string s, $\text{parity}(s)$ is 1 if the number of 1's in s is odd, and 0 otherwise; also, $\text{reverse}(s)$ is the string obtained by reversing the order of the bits in s, e.g., $\text{reverse}(1011) = 1101$.

2.11 (Key recovery attacks). Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. A key recovery attack is modeled by the following game between a challenger and an adversary $\mathcal{A}$: the challenger chooses a random key k in $\mathcal{K}$, a random message m in $\mathcal{M}$, computes $c \xleftarrow{R} E(k, m)$, and sends (m, c) to $\mathcal{A}$. In response $\mathcal{A}$ outputs a guess $\hat{k}$ in $\mathcal{K}$. We say that $\mathcal{A}$ wins the game if $D(\hat{k}, c) = m$ and define $\mathsf{KRadv}[\mathcal{A}, \mathcal{E}]$ to be the probability that $\mathcal{A}$ wins the game. As usual, we say that $\mathcal{E}$ is secure against key recovery attacks if for all efficient adversaries $\mathcal{A}$ the advantage $\mathsf{KRadv}[\mathcal{A}, \mathcal{E}]$ is negligible.

(a) Show that the one-time pad is not secure against key recovery attacks.

(b) Show that if $\mathcal{E}$ is semantically secure and $\epsilon = |\mathcal{K}|/|\mathcal{M}|$ is negligible, then $\mathcal{E}$ is secure against key recovery attacks. In particular, show that for every efficient key-recovery adversary $\mathcal{A}$ there is an efficient semantic security adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{KR}\mathsf{adv}[\mathcal{A},\mathcal{E}] \leq \text{SS}\mathsf{adv}[\mathcal{B},\mathcal{E}] + \epsilon$$

Hint: Your semantic security adversary $\mathcal{B}$ will output 1 with probability $\text{KR}\mathsf{adv}[\mathcal{A},\mathcal{E}]$ in the semantic security Experiment 1, and output 1 with probability at most ϵ in Experiment 0. Deduce from this a lower bound on $\text{SS}\mathsf{adv}[\mathcal{B},\mathcal{E}]$ in terms of ϵ and $\text{KR}\mathsf{adv}[\mathcal{A},\mathcal{E}]$ from which the result follows.

(c) Deduce from part (b) that if $\mathcal{E}$ is semantically secure and $|\mathcal{M}|$ is super-poly, then $|\mathcal{K}|$ cannot be poly-bounded.

Note: $|\mathcal{K}|$ can be poly-bounded when $|\mathcal{M}|$ is poly-bounded, as in the one-time pad.

2.12 (Security against message recovery). In Section 2.2.3.1 we developed the notion of security against message recovery. Construct a cipher that is secure against message recovery, but is not semantically secure.

2.13 (Advantage calculations in simple settings). Consider the following two experiments Experiment 0 and Experiment 1:

- In Experiment 0 the challenger flips a fair coin (probability 1/2 for HEADS and 1/2 for TAILS) and sends the result to the adversary $\mathcal{A}$.
- In Experiment 1 the challenger always sends TAILS to the adversary.

The adversary's goal is to distinguish these two experiments: at the end of each experiment the adversary outputs a bit 0 or 1 for its guess for which experiment it is in. For $b = 0, 1$ let W_b be the event that in experiment b the adversary output 1. The adversary tries to maximize its distinguishing advantage, namely the quantity

$$\big|\Pr[W_0] - \Pr[W_1]\big| \quad \in [0,1] \ .$$

If the advantage is negligible for all efficient adversaries then we say that the two experiments are indistinguishable.

(a) Calculate the advantage of each of the following adversaries:

(i) $\mathcal{A}_1$: Always output 1.

(ii) $\mathcal{A}_2$: Ignore the result reported by the challenger, and randomly output 0 or 1 with even probability.

(iii) $\mathcal{A}_3$: Output 1 if HEADS was received from the challenger, else output 0.

(iv) $\mathcal{A}_4$: Output 0 if HEADS was received from the challenger, else output 1.

(v) $\mathcal{A}_5$: If HEADS was received, output 1. If TAILS was received, randomly output 0 or 1 with even probability.

(b) What is the maximum advantage possible in distinguishing these two experiments? Explain why.

2.14 (Permutation cipher). Consider the following cipher (E, D) defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where $\mathcal{C} = \mathcal{M} = \{0,1\}^\ell$ and $\mathcal{K}$ is the set of all $\ell!$ permutations of the set $\{0, \dots, \ell - 1\}$. For a key $k \in \mathcal{K}$ and message $m \in \mathcal{M}$ define $E(k, m)$ to be result of permuting the bits of m using the permutation k, namely $E(k, m) = m[k(0)] ... m[k(\ell - 1)]$. Show that this cipher is not semantically secure by showing an adversary that achieves advantage 1.

2.15 (Nested encryption). For a cipher $\mathcal{E} = (E, D)$ with key space $\mathcal{K}$ define the nested cipher $\mathcal{E}' = (E', D')$ as

$$E'\big((k_0, k_1), m\big) := E\big(k_1, E(k_0, m)\big) \quad \text{and} \quad D'\big((k_0, k_1), c\big) := D(k_0, D(k_1, c)) \ .$$

Our goal is to show that if $\mathcal{E}$ is semantically secure then $\mathcal{E}'$ is semantically secure even if the adversary is given one of the keys k_0 or k_1.

(a) Consider the following two semantic security experiments, Experiments 0 and 1. For $b = 0, 1$, in Experiment b the adversary first sends to the challenger two messages m_0 and m_1. The challenger chooses keys $k_0, k_1 \xleftarrow{\text{R}} \mathcal{K}$ and sends back the pair (k_1, c) where $c \xleftarrow{\text{R}} E'\big((k_0, k_1),\ m_b\big)$. Finally, the adversary outputs $\hat{b}$ in $\{0, 1\}$. We define the adversary's advantage, $\text{NE}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, as in the usual the definition of semantic security. Show that for every nested encryption adversary $\mathcal{A}$ attacking $\mathcal{E}'$, there is a semantic security adversary $\mathcal{B}$ attacking $\mathcal{E}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{NE}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}] \ .$$

Draw a diagram with $\mathcal{A}$ on the right, $\mathcal{B}$ in the middle, and $\mathcal{B}$'s challenger on the left. Show the message flow between these three parties that takes place in your proof of security.

(b) Repeat part (a), but now when the adversary gets back the pair (k_0, c) from the challenger (i.e., k_1 is replaced by k_0), where $c \xleftarrow{\text{R}} E'\big((k_0, k_1),\ m_b\big)$ as before. Draw a diagram describing the message flow in your proof of security as you did in part (a).

This problem comes up in the context of anonymous routing on the Internet as discussed in Section 2.4.

2.16 (Self referential encryption). Let us show that encrypting a key under itself can be dangerous. Let $\mathcal{E}$ be a semantically secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{K} \subseteq \mathcal{M}$, and let $k \xleftarrow{\text{R}} \mathcal{K}$. A ciphertext $c_* := E(k, k)$, namely encrypting k using k, is called a **self referential encryption**.

(a) Construct a cipher $\tilde{\mathcal{E}} = (\tilde{E}, \tilde{D})$ derived from $\mathcal{E}$ such that $\tilde{\mathcal{E}}$ is semantically secure, but becomes insecure if the adversary is given $\tilde{E}(k, k)$. You have just shown that semantic security does not imply security when one encrypts one's key.

(b) Construct a cipher $\hat{\mathcal{E}} = (\hat{E}, \hat{D})$ derived from $\mathcal{E}$ such that $\hat{\mathcal{E}}$ is semantically secure and remains semantically secure (provably) even if the adversary is given $\hat{E}(k, k)$. To prove that $\hat{\mathcal{E}}$ is semantically secure, you should show the following: for every adversary $\mathcal{A}$ that attacks $\hat{\mathcal{E}}$, there exists and adversary $\mathcal{B}$ that attacks $\mathcal{E}$ such that (i) the running time $\mathcal{B}$ is about the same as that of $\mathcal{A}$, and (ii) $\text{SS}\mathsf{adv}[\mathcal{A}, \hat{\mathcal{E}}] \le \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}]$.

2.17 (Compression and encryption). Two standards committees propose to save bandwidth by combining compression (such as the Lempel-Ziv algorithm used in the zip and gzip programs) with encryption. Both committees plan on using the variable length one time pad for encryption.

- One committee proposes to compress messages before encrypting them. Explain why this is a bad idea.

  Hint: Recall that compression can significantly shrink the size of some messages while having little impact on the length of other messages.

- The other committee proposes to compress ciphertexts after encryption. Explain why this is a bad idea.

Over the years many problems have surfaced when combining encryption and compression. The CRIME [136] and BREACH [131] attacks are good representative examples.

2.18 (Voting protocols). This exercise develops a simple voting protocol based on the additive one-time pad (Example 2.4). Suppose we have t voters and a counting center. Each voter is going to vote 0 or 1, and the counting center is going to tally the votes and broadcast the total sum S. However, they will use a protocol that guarantees that no party (voter or counting center) learns anything other than S (but we shall assume that each party faithfully follows the protocol).

The protocol works as follows. Let $n > t$ be an integer. The counting center generates an encryption of 0: $c_0 \overset{\text{R}}{\leftarrow} \{0, \ldots, n-1\}$, and passes c_0 to voter 1. Voter 1 adds his vote v_1 to c_0, computing $c_1 \leftarrow c_0 + v_1 \bmod n$, and passes c_1 to voter 2. This continues, with each voter i adding v_i to c_{i-1}, computing $c_i \leftarrow c_{i-1} + v_i \bmod n$, and passing c_i to voter $i+1$, except that voter t passes c_t to the counting center. The counting center computes the total sum as $S \leftarrow c_t - c_0 \bmod n$, and broadcasts S to all the voters.

(a) Show that the protocol correctly computes the total sum.

(b) Show that the protocol is perfectly secure in the following sense. For voter $i = 1, \ldots, t$, define $\mathit{View}_i := (S, c_{i-1})$, which represents the "view" of voter i. We also define $\mathit{View}_0 := (c_0, c_t)$, which represents the "view" of the counting center. Show that for each $i = 0, \ldots, t$ and $S = 0, \ldots, t$, the following holds:

> as the choice of votes $v_1, \ldots, v_t$ varies, subject to the restrictions that each $v_j \in \{0,1\}$ and $\sum_{j=1}^{t} v_j = S$, the distribution of View_i remains the same.

(c) Show that if two voters i, j collude, they can determine the vote of a third voter k. You are free to choose the indices i, j, k.

2.19 (Two-way split keys). Let $\mathcal{E} = (E, D)$ be a semantically secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where $\mathcal{K} = \{0,1\}^d$. Suppose we wish to split the ability to decrypt ciphertexts across two parties, Alice and Bob, so that both parties are needed to decrypt ciphertexts. For a random key k in $\mathcal{K}$ choose a random r in $\mathcal{K}$ and define $k_{\text{a}} := r$ and $k_{\text{b}} := k \oplus r$. Now if Alice and Bob get together they can decrypt a ciphertext c by first reconstructing the key k as $k = k_{\text{a}} \oplus k_{\text{b}}$ and then computing $D(k, c)$. Our goal is to show that neither Alice nor Bob can decrypt ciphertexts on their own.

(a) Formulate a security notion that captures the advantage that an adversary has in breaking semantic security given Bob's key k_{b}. Denote this 2-way key splitting advantage by $2\text{KS}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$.

(b) Show that for every 2-way key splitting adversary $\mathcal{A}$ there is a semantic security adversary $\mathcal{B}$ such that $2\text{KS}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}]$.

2.20 (Simple secret sharing). Let $\mathcal{E} = (E, D)$ be a semantically secure cipher with key space $\mathcal{K} = \{0,1\}^L$. A bank wishes to split a decryption key $k \in \{0,1\}^L$ into three shares p_0, p_1, and p_2 so that two of the three shares are needed for decryption. Each share can be given to a different bank executive, and two of the three must contribute their shares for decryption to proceed. This way, decryption can proceed even if one of the executives is out sick, but at least two executives are needed for decryption.

(a) To do so the bank generates two random pairs (k_0, k_0') and (k_1, k_1') so that $k_0 \oplus k_0' = k_1 \oplus k_1' = k$. How should the bank assign shares so that any two shares enable decryption using k, but no single share can decrypt?

Hint: The first executive will be given the share $p_0 := (k_0, k_1)$.

(b) Generalize the scheme from part (a) so that 3-out-of-5 shares are needed for decryption. Reconstituting the key only uses XOR of key shares. Two shares should reveal nothing about the key k.

(c) More generally, we can design a t-out-of-w system this way for any $t < w$. How does the size of each share scale with t? We will see a much better way to do this in Chapter 22

2.21 (Simple threshold decryption). Let $\mathcal{E} = (E, D)$ be a semantically secure cipher with key space $\mathcal{K}$. In this exercise we design a system that lets a bank split a key k into three shares p_0, p_1, and p_2 so that two of the three shares are needed for decryption, as in Exercise 2.20. However, decryption is done without ever reconstituting the complete key at a single location.

We use nested encryption from Exercise 2.15. Choose a random key $k := (k_0, k_1, k_2, k_3)$ in $\mathcal{K}^4$ and encrypt a message m as:

$$c \xleftarrow{\text{R}} \Big(E\big(k_1, E(k_0, m)\big), \quad E\big(k_3, E(k_2, m)\big) \Big).$$

(a) Construct the shares p_0, p_1, p_2 so that any two shares enable decryption, but no single share can decrypt. Hint: the first share is $p_0 := (k_0, k_3)$.

Discussion: Suppose the entities holding shares p_0 and p_2 are available to decrypt. To decrypt a ciphertext c, first send c to the entity holding p_2 to partially decrypt c. Then forward the result to the entity holding p_0 to complete the decryption. This way, decryption is done without reconstituting the complete key k at a single location.

(b) Generalize the scheme from part (a) so that 3-out-of-5 shares are needed for decryption. Explain how decryption can be done without reconstituting the key in a single location.

An encryption scheme where the key can be split into shares so that t-out-of-w shares are needed for decryption, and decryption does not reconstitute the key at a single location, is said to provide **threshold decryption**. We will see a much better way to do this in Chapter 22.

2.22 (Bias correction). Consider again the bit-guessing version of the semantic security attack game (i.e., Attack Game 2.4). Suppose an efficient adversary $\mathcal{A}$ wins the game (i.e., guesses the hidden bit b) with probability $1/2 + \epsilon$, where ϵ is non-negligible. Note that ϵ could be positive or negative (the definition of negligible works on absolute values). Our goal is to show that there is another efficient adversary $\mathcal{B}$ that wins the game with probability $1/2+\epsilon'$, where ϵ' is non-negligible and positive.

(a) Consider the following adversary $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine in Attack Game 2.4 in the following two-stage attack. In the first stage, $\mathcal{B}$ plays challenger to $\mathcal{A}$, but $\mathcal{B}$ generates its own hidden bit b_0, its own key k_0, and eventually $\mathcal{A}$ outputs its guess-bit $\hat{b}_0$. Note that in this stage, $\mathcal{B}$'s challenger in Attack Game 2.4 is not involved at all. In the second stage, $\mathcal{B}$ restarts $\mathcal{A}$, and lets $\mathcal{A}$ interact with the "real" challenger in Attack Game 2.4, and eventually $\mathcal{A}$ outputs a guess-bit $\hat{b}$. When this happens, $\mathcal{B}$ outputs $\hat{b} \oplus \hat{b}_0 \oplus b_0$. Note that this run of $\mathcal{A}$ is completely independent of the first — the coins of $\mathcal{A}$ and also the system parameters are generated independently in these two runs.

Show that $\mathcal{B}$ wins Attack Game 2.4 with probability $1/2 + 2\epsilon^2$.

(b) One might be tempted to argue as follows. Just construct an adversary $\mathcal{B}$ that runs $\mathcal{A}$, and when $\mathcal{A}$ outputs $\hat{b}$, adversary $\mathcal{B}$ outputs $\hat{b} \oplus 1$. Now, we do not know if ϵ is positive or negative. If it is positive, then $\mathcal{A}$ satisfies our requirements. If it is negative, then $\mathcal{B}$ satisfies our requirements. Although we do not know which one of these two adversaries satisfies our requirements, we know that one of them definitely does, and so existence is proved.

What is wrong with this argument? The explanation requires an understanding of the mathematical details regarding security parameters (see Section 2.3).

(c) Can you come up with another efficient adversary $\mathcal{B}'$ that wins the bit-guessing game with probability at least $1/2 + |\epsilon|/2$? Your adversary $\mathcal{B}'$ will be less efficient than $\mathcal{B}$.

Hint: try running the first stage of adversary $\mathcal{B}$ multiple times.

Chapter 3

Stream ciphers

In the previous chapter, we introduced the notions of perfectly secure encryption and semantically secure encryption. The problem with perfect security is that to achieve it, one must use very long keys. Semantic security was introduced as a weaker notion of security that would perhaps allow us to build secure ciphers that use reasonably short keys; however, we have not yet produced any such ciphers. This chapter studies one type of cipher that does this: the stream cipher.

3.1 Pseudo-random generators

Recall the one-time pad. Here, keys, messages, and ciphertexts are all L-bit strings. However, we would like to use a key that is much shorter. So the idea is to instead use a short, ℓ-bit "seed" s as the encryption key, where ℓ is much smaller than L, and to "stretch" this seed into a longer, L-bit string that is used to mask the message (and unmask the ciphertext). The string s is stretched using some efficient, deterministic algorithm G that maps ℓ-bit strings to L-bit strings. Thus, the key space for this modified one-time pad is $\{0,1\}^\ell$, while the message and ciphertext spaces are $\{0,1\}^L$. For $s \in \{0,1\}^\ell$ and $m, c \in \{0,1\}^L$, encryption and decryption are defined as follows:

$$E(s,m) := G(s) \oplus m \quad \text{and} \quad D(s,c) := G(s) \oplus c.$$

This modified one-time pad is called a **stream cipher**, and the function G is called a **pseudo-random generator**.

If $\ell < L$, then by Shannon's Theorem, this stream cipher cannot achieve perfect security; however, if G satisfies an appropriate security property, then this cipher is semantically secure. Suppose s is a random ℓ-bit string and r is a random L-bit string. Intuitively, if an adversary cannot effectively tell the difference between $G(s)$ and r, then he should not be able to tell the difference between this stream cipher and a one-time pad; moreover, since the latter cipher is semantically secure, so should be the former. To make this reasoning rigorous, we need to formalize the notion that an adversary cannot "effectively tell the difference between $G(s)$ and r."

An algorithm that is used to distinguish a pseudo-random string $G(s)$ from a truly random string r is called a **statistical test**. It takes a string as input, and outputs 0 or 1. Such a test is called **effective** if the probability that it outputs 1 on a pseudo-random input is significantly different than the probability that it outputs 1 on a truly random input. Even a relatively small difference in probabilities, say 1%, is considered significant; indeed, even with a 1% difference, if we can obtain a few hundred independent samples, which are either all pseudo-random or all truly random, then we will be able to infer with high confidence whether we are looking at pseudo-random strings or at truly random strings. However, a non-zero but negligible difference in probabilities, say 2^{-100}, is not helpful.

How might one go about designing an effective statistical test? One basic approach is the following: given an L-bit string, calculate some statistic, and then see if this statistic differs greatly from what one would expect if the string were truly random.

For example, a very simple statistic that is easy to compute is the number k of 1's appearing in the string. For a truly random string, we would expect $k \approx L/2$. If the PRG G had some bias towards either 0-bits or 1-bits, we could effectively detect this with a statistical test that, say, outputs 1 if $|k - 0.5L| < 0.01L$, and otherwise outputs 0. This statistical test would be quite effective if the PRG G did indeed have some significant bias towards either 0 or 1.

The test in the previous example can be strengthened by considering not just individual bits, but pairs of bits. One could break the L-bit string up into $\approx L/2$ bit pairs, and count the number k_{00} of pairs 00, the number k_{01} of pairs 01, the number k_{10} of pairs 10, and the number k_{11} of pairs 11. For a truly random string, one would expect each of these numbers to be $\approx L/2 \cdot 1/4 = L/8$. Thus, a natural statistical test would be one that tests if the distance from $L/8$ of each of these numbers is less than some specified bound. Alternatively, one could sum up the squares of these distances, and test whether this sum is less than some specified bound — this is the classical χ-squared test from statistics. Obviously, this idea generalizes from pairs of bits to tuples of any length.

There are many other simple statistics one might check. However, simple tests such as these do not tend to exploit deeper mathematical properties of the algorithm G that a malicious adversary may be able to exploit in designing a statistical test specifically geared towards G. For example, there are PRG's for which the simple tests in the previous two paragraphs are completely ineffective, but yet are completely predictable, given sufficiently many output bits; that is, given a prefix of $G(s)$ of sufficient length, the adversary can compute all the remaining bits of $G(s)$, or perhaps even compute the seed s itself.

Our definition of security for a PRG formalizes the notion that there should be no effective (and efficiently computable) statistical test.

3.1.1 Definition of a pseudo-random generator

A **pseudo-random generator**, or **PRG** for short, is an efficient, deterministic algorithm G that, given as input a **seed** s, computes an output r. The seed s comes from a finite **seed space** $\mathcal{S}$ and the output r belongs to a finite **output space** $\mathcal{R}$. Typically, $\mathcal{S}$ and $\mathcal{R}$ are sets of bit strings of some prescribed length (for example, in the discussion above, we had $\mathcal{S} = \{0,1\}^\ell$ and $\mathcal{R} = \{0,1\}^L$). We say that G is a PRG defined over $(\mathcal{S}, \mathcal{R})$.

Our definition of security for a PRG captures the intuitive notion that if s is chosen at random from $\mathcal{S}$ and r is chosen at random from $\mathcal{R}$, then no efficient adversary can effectively tell the difference between $G(s)$ and r: the two are **computationally indistinguishable**. The definition is formulated as an attack game.

Attack Game 3.1 (PRG). For a given PRG G, defined over $(\mathcal{S}, \mathcal{R})$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define:

Experiment b:

- The challenger computes $r \in \mathcal{R}$ as follows:

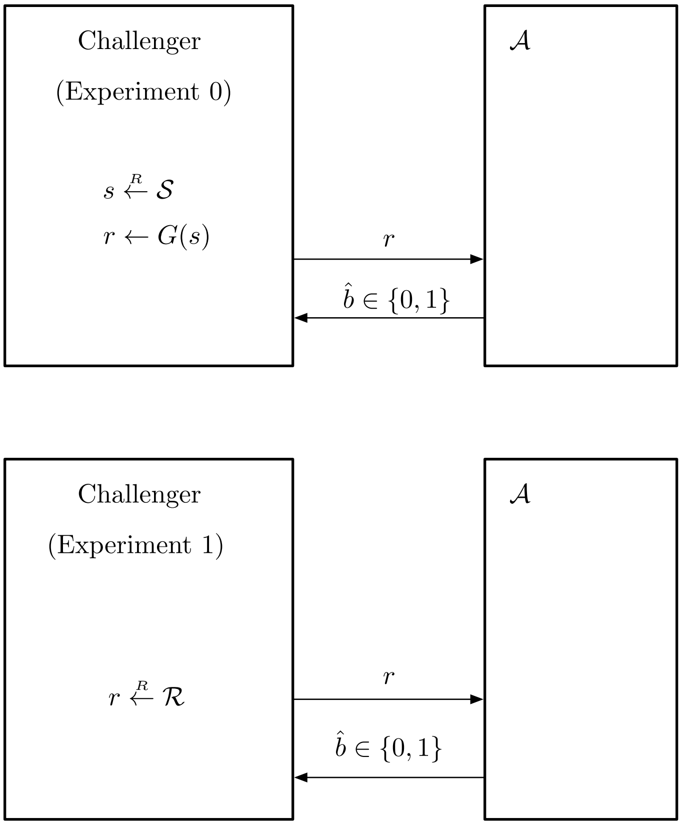


Figure 3.1: Experiments 0 and 1 of Attack Game 3.1

- if $b = 0$: $s \xleftarrow{\mathrm{R}} \mathcal{S}$, $r \leftarrow G(s)$;
- if $b = 1$: $r \xleftarrow{\mathrm{R}} \mathcal{R}$.

and sends r to the adversary.

- Given r, the adversary computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to G as

$$\mathrm{PRG}\mathsf{adv}[\mathcal{A}, G] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

The attack game is illustrated in Fig. 3.1.

Definition 3.1 (secure PRG). *A PRG G is* **secure** *if the value* $\mathrm{PRG}\mathsf{adv}[\mathcal{A}, G]$ *is negligible for all efficient adversaries* $\mathcal{A}$*.*

As discussed in Section 2.2.5, Attack Game 3.1 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage*

$\text{PRG}\mathsf{adv}^*[\mathcal{A}, G]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.2.5 (namely, (2.11)) applies here as well:

$$\text{PRG}\mathsf{adv}[\mathcal{A}, G] = 2 \cdot \text{PRG}\mathsf{adv}^*[\mathcal{A}, G]. \tag{3.1}$$

We also note that a PRG can only be secure if the cardinality of the seed space is super-poly (see Exercise 3.5).

3.1.2 Mathematical details

Just as in Section 2.3, we give here more of the mathematical details pertaining to PRGs. Just like Section 2.3, this section may be safely skipped on first reading with very little loss in understanding.

First, we state the precise definition of a PRG, using the terminology introduced in Definition 2.9.

Definition 3.2 (pseudo-random generator). *A* ***pseudo-random generator*** *consists of an algorithm G, along with two families of spaces with system parameterization P:*

$$\mathbf{S} = \{\mathcal{S}_{\lambda,\Lambda}\}_{\lambda,\Lambda} \quad \textit{and} \quad \mathbf{R} = \{\mathcal{R}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

such that

1. $\mathbf{S}$ *and* $\mathbf{R}$ *are efficiently recognizable and sampleable.*

2. *Algorithm G is an efficient deterministic algorithm that on input λ, Λ, s, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \text{Supp}(P(\lambda))$, and $s \in \mathcal{S}_{\lambda,\Lambda}$, outputs an element of $\mathcal{R}_{\lambda,\Lambda}$.*

Next, Definition 3.1 needs to be properly interpreted. First, in Attack Game 3.1, it is to be understood that for each value of the security parameter λ, we get a different probability space, determined by the random choices of the challenger and the random choices of the adversary. Second, the challenger generates a system parameter Λ, and sends this to the adversary at the very start of the game. Third, the advantage $\text{PRG}\mathsf{adv}[\mathcal{A}, G]$ is a function of the security parameter λ, and security means that this function is a negligible function.

3.2 Stream ciphers: encryption with a PRG

Let G be a PRG defined over $(\{0,1\}^\ell, \{0,1\}^L)$; that is, G stretches an ℓ-bit seed to an L-bit output. The **stream cipher $\mathcal{E} = (E, D)$ constructed from** G is defined over $(\{0,1\}^\ell, \{0,1\}^{\leq L}, \{0,1\}^{\leq L})$; for $s \in \{0,1\}^\ell$ and $m, c \in \{0,1\}^{\leq L}$, encryption and decryption are defined as follows: if $|m| = v$, then

$$E(s, m) := G(s)[0 \mathinner{..} v-1] \;\oplus\; m,$$

and if $|c| = v$, then

$$D(s, c) := G(s)[0 \mathinner{..} v-1] \;\oplus\; c.$$

As the reader may easily verify, this satisfies our definition of a cipher (in particular, the correctness property is satisfied).

Note that for the purposes of analyzing the semantic security of $\mathcal{E}$, the length associated with a message m in Attack Game 2.1 is the natural length $|m|$ of m in bits. Also, note that if v is much smaller than L, then for many practical PRGs, it is possible to compute the first v bits of $G(s)$ much faster than actually computing all the bits of $G(s)$ and then truncating.

The main result of this section is the following:

Theorem 3.1. *If G is a secure PRG, then the stream cipher $\mathcal{E}$ constructed from G is a semantically secure cipher.*

> *In particular, for every SS adversary $\mathcal{A}$ that attacks $\mathcal{E}$ as in Attack Game 2.1, there exists a PRG adversary $\mathcal{B}$ that attacks G as in Attack Game 3.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
> $$\mathsf{SSadv}[\mathcal{A},\mathcal{E}] = 2 \cdot \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G]. \tag{3.2}$$

Proof idea. The basic idea is to argue that we can replace the output of the PRG by a truly random string, without affecting the adversary's advantage by more than a negligible amount. However, after making this replacement, the adversary's advantage is zero. □

Proof. Let $\mathcal{A}$ be an efficient adversary attacking $\mathcal{E}$ as in Attack Game 2.1. We want to show that $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]$ is negligible, assuming that G is a secure PRG. It is more convenient to work with the bit-guessing version of the SS attack game. We prove:

$$\mathsf{SSadv}^*[\mathcal{A},\mathcal{E}] = \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G] \tag{3.3}$$

for some efficient adversary $\mathcal{B}$. Then (3.2) follows from Theorem 2.10. Moreover, by the assumption that G is a secure PRG, the quantity $\mathrm{PRG}\mathsf{adv}[\mathcal{B}, G]$ must be negligible, and so the quantity $\mathsf{SSadv}[\mathcal{A},\mathcal{E}]$ is negligible as well.

So consider the adversary $\mathcal{A}$'s attack of $\mathcal{E}$ in the bit-guessing version of Attack Game 2.1. In this game, $\mathcal{A}$ presents the challenger with two messages m_0, m_1 of the same length; the challenger then chooses a random key s and a random bit b, and encrypts m_b under s, giving the resulting ciphertext c to $\mathcal{A}$; finally, $\mathcal{A}$ outputs a bit $\hat{b}$. The adversary $\mathcal{A}$ wins the game if $\hat{b} = b$.

Let us call this **Game 0.** The logic of the challenger in this game may be written as follows:

```
Upon receiving $m_0, m_1 \in \{0,1\}^v$ from $\mathcal{A}$, for some $v \le L$, do:
    $b \overset{\mathrm{R}}{\leftarrow} \{0,1\}$
    $s \overset{\mathrm{R}}{\leftarrow} \{0,1\}^\ell$, $r \leftarrow G(s)$
    $c \leftarrow r[0\,..\,v-1] \oplus m_b$
    send $c$ to $\mathcal{A}$.
```

Game 0 is illustrated in Fig. 3.2.

Let W_0 be the event that $\hat{b} = b$ in Game 0. By definition, we have

$$\mathsf{SSadv}^*[\mathcal{A},\mathcal{E}] = |\Pr[W_0] - 1/2|. \tag{3.4}$$

Next, we modify the challenger of Game 0, obtaining a new game, called **Game 1**, which is exactly the same as Game 0, except that the challenger uses a truly random string in place of a pseudo-random string. The logic of the challenger in Game 1 is as follows:

```
Upon receiving $m_0, m_1 \in \{0,1\}^v$ from $\mathcal{A}$, for some $v \le L$, do:
    $b \overset{\mathrm{R}}{\leftarrow} \{0,1\}$
    $r \overset{\mathrm{R}}{\leftarrow} \{0,1\}^L$
    $c \leftarrow r[0\,..\,v-1] \oplus m_b$
    send $c$ to $\mathcal{A}$.
```

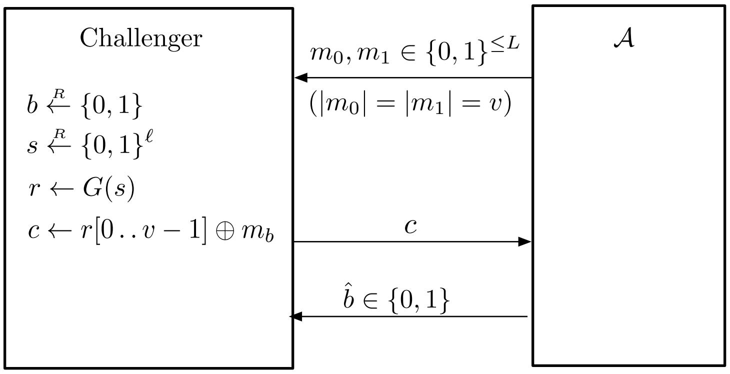

Challenger
$b \overset{R}{\leftarrow} \{0,1\}$
$s \overset{R}{\leftarrow} \{0,1\}^\ell$
$r \leftarrow G(s)$
$c \leftarrow r[0\,..\,v-1] \oplus m_b$
$\mathcal{A}$
$m_0, m_1 \in \{0,1\}^{\leq L}$
$(|m_0| = |m_1| = v)$
c
$\hat{b} \in \{0,1\}$

Figure 3.2: Game 0 in the proof of Theorem 3.1

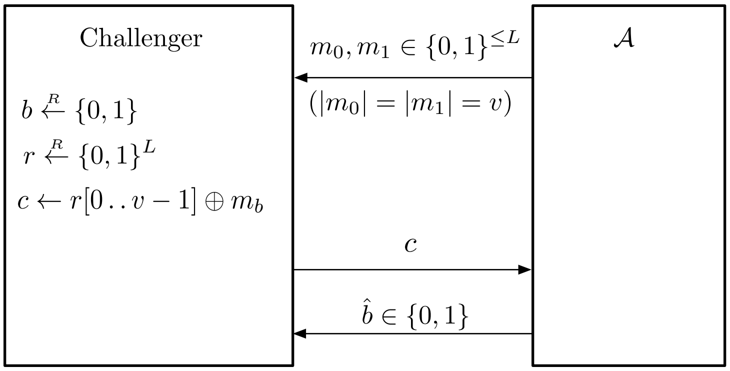

Challenger
$b \overset{R}{\leftarrow} \{0,1\}$
$r \overset{R}{\leftarrow} \{0,1\}^L$
$c \leftarrow r[0\,..\,v-1] \oplus m_b$
$\mathcal{A}$
$m_0, m_1 \in \{0,1\}^{\leq L}$
$(|m_0| = |m_1| = v)$
c
$\hat{b} \in \{0,1\}$

Figure 3.3: Game 1 in the proof of Theorem 3.1

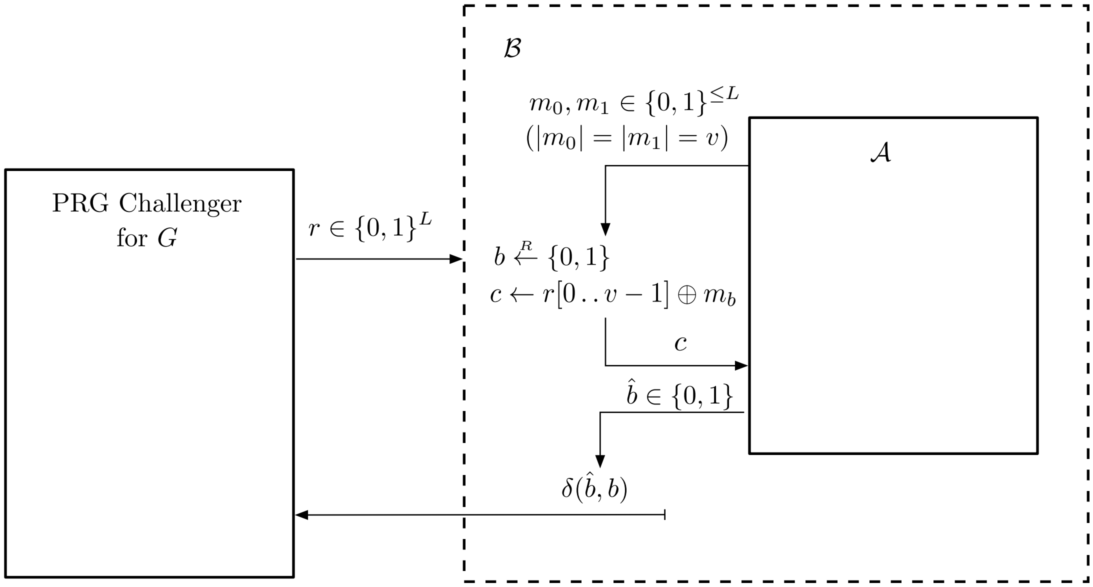


Figure 3.4: The PRG adversary $\mathcal{B}$ in the proof of Theorem 3.1

As usual, $\mathcal{A}$ outputs a bit $\hat{b}$ at the end of this game. We have highlighted the changes from Game 0 in gray. Game 1 is illustrated in Fig. 3.3.

Let W_1 be the event that $\hat{b} = b$ in Game 1. We claim that

$$\Pr[W_1] = 1/2. \tag{3.5}$$

This is because in Game 1, the adversary is attacking the variable length one-time pad. In particular, it is easy to see that the adversary's output $\hat{b}$ and the challenger's hidden bit b are independent.

Finally, we show how to construct an efficient PRG adversary $\mathcal{B}$ that uses $\mathcal{A}$ as a subroutine, such that

$$|\Pr[W_0] - \Pr[W_1]| = \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G]. \tag{3.6}$$

This is actually quite straightforward. The logic of our new adversary $\mathcal{B}$ is illustrated in Fig. 3.4. Here, δ is defined as follows:

$$\delta(x,y) := \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{if } x \neq y. \end{cases} \tag{3.7}$$

Also, the box labeled "PRG Challenger" is playing the role of the challenger in Attack Game 3.1 with respect to G.

In words, adversary $\mathcal{B}$, which is a PRG adversary designed to attack G (as in Attack Game 3.1), receives $r \in \{0,1\}^L$ from its PRG challenger, and then plays the role of challenger to $\mathcal{A}$, as follows:

```
Upon receiving $m_0, m_1 \in \{0,1\}^v$ from $\mathcal{A}$, for some $v \leq L$, do:
    $b \overset{R}{\leftarrow} \{0,1\}$
    $c \leftarrow r[0 \mathinner{..} v-1] \oplus m_b$
    send $c$ to $\mathcal{A}$.
```

Finally, when $\mathcal{A}$ outputs a bit $\hat{b}$, $\mathcal{B}$ outputs the bit $\delta(\hat{b}, b)$.

Let p_0 be the probability that $\mathcal{B}$ outputs 1 when the PRG challenger is running Experiment 0 of Attack Game 3.1, and let p_1 be the probability that $\mathcal{B}$ outputs 1 when the PRG challenger is running Experiment 1 of Attack Game 3.1. By definition, $\text{PRG}\mathsf{adv}[\mathcal{B}, G] = |p_1 - p_0|$. Moreover, if the PRG challenger is running Experiment 0, then adversary $\mathcal{A}$ is essentially playing our Game 0, and so $p_0 = \Pr[W_0]$, and if the PRG challenger is running Experiment 1, then $\mathcal{A}$ is essentially playing our Game 1, and so $p_1 = \Pr[W_1]$. Equation (3.6) now follows immediately.

Combining (3.4), (3.5), and (3.6), yields (3.3). □

In the above theorem, we reduced the security of $\mathcal{E}$ to that of G by showing that if $\mathcal{A}$ is an efficient SS adversary that attacks $\mathcal{E}$, then there exists an efficient PRG adversary $\mathcal{B}$ that attacks G, such that

$$\text{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq 2 \cdot \text{PRG}\mathsf{adv}[\mathcal{B}, G].$$

(Actually, we showed that equality holds, but that is not so important.) In the proof, we argued that if G is secure, then $\text{PRG}\mathsf{adv}[\mathcal{B}, G]$ is negligible, hence by the above inequality, we conclude that $\text{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is also negligible. Since this holds for all efficient adversaries $\mathcal{A}$, we conclude that $\mathcal{E}$ is semantically secure.

Analogous to the discussion after the proof of Theorem 2.7, another way to structure the proof is by proving the contrapositive: indeed, if we assume that $\mathcal{E}$ is insecure, then there must be an efficient adversary $\mathcal{A}$ such that $\text{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is non-negligible, and the reduction (and the above inequality) gives us an efficient adversary $\mathcal{B}$ such that $\text{PRG}\mathsf{adv}[\mathcal{B}, G]$ is also non-negligible. That is, if we can break $\mathcal{E}$, we can also break G. While logically equivalent, such a proof has a different "feeling": one starts with an adversary $\mathcal{A}$ that breaks $\mathcal{E}$, and shows how to use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ that breaks G.

The reader should notice that the proof of the above theorem follows the same basic pattern as our analysis of Internet roulette in Section 2.2.4. In both cases, we started with an attack game (Fig. 2.2 or Fig. 3.2) which we modified to obtain a new attack game (Fig. 2.3 or Fig. 3.3); in this new attack game, it was quite easy to compute the adversary's advantage. Also, we used an appropriate security assumption to show that the difference between the adversary's advantages in the original and the modified games was negligible. This was done by exhibiting a new adversary (Fig. 2.4 or Fig. 3.4) that attacked the underlying cryptographic primitive (cipher or PRG) with an advantage equal to this difference. Assuming the underlying primitive was secure, this difference must be negligible; alternatively, one could argue the contrapositive: if this difference were not negligible, the new adversary would "break" the underlying cryptographic primitive.

This is a pattern that will be repeated and elaborated upon throughout this text. The reader is urged to study both of these analyses to make sure he or she completely understands what is going on.

3.3 Stream cipher limitations: attacks on the one time pad

Although stream ciphers are semantically secure, they are quite brittle and become insecure if used incorrectly.

3.3.1 The two-time pad is insecure

A stream cipher is well equipped to encrypt a *single* message from Alice to Bob. Alice, however, may wish to send several messages to Bob. For simplicity suppose Alice wishes to encrypt two messages m_1 and m_2. The naive solution is to encrypt both messages using the same stream cipher key s:

$$c_1 \leftarrow m_1 \oplus G(s) \qquad \text{and} \qquad c_2 \leftarrow m_2 \oplus G(s) \tag{3.8}$$

A moments reflection shows that this construction is insecure in a very strong sense. An adversary who intercepts c_1 and c_2 can compute

$$\Delta := c_1 \oplus c_2 = \big(m_1 \oplus G(s)\big) \oplus \big(m_2 \oplus G(s)\big) = m_1 \oplus m_2$$

and obtain the xor of m_1 and m_2. Not surprisingly, English text contains enough redundancy that given $\Delta = m_1 \oplus m_2$ the adversary can recover both m_1 and m_2 in the clear. Hence, the construction in (3.8) leaks the plaintexts after seeing only two sufficiently long ciphertexts.

The construction in (3.8) is jokingly called the **two-time pad**. We just argued that the two-time pad is totally insecure. In particular, **a stream cipher key should never be used to encrypt more than one message**. Throughout the book we will see many examples where a one-time cipher is sufficient. For example, when choosing a new random key for every message as in Section 5.4.1. However, in settings where a single key is used multiple times, one should never use a stream cipher directly. We build multi-use ciphers in Chapter 5.

Incorrectly reusing a stream cipher key is a common error in deployed systems. For example, a protocol called PPTP enables two parties A and B to send encrypted messages to one another. Microsoft's implementation of PPTP in Windows NT uses a stream cipher called RC4. The original implementation encrypts messages from A to B using the same RC4 key as messages from B to A [140]. Consequently, by eavesdropping on two encrypted messages headed in opposite directions an attacker could recover the plaintext of both messages.

Another amusing story about the two-time pad is relayed by Klehr [85] who describes in great detail how Russian spies in the US during World War II were sending messages back to Moscow, encrypted with the one-time pad. The system had a critical flaw, as explained by Klehr:

> During WWII the Soviet Union could not produce enough one-time pads ... to keep up with the enormous demand .... So, they used a number of one-time pads twice, thinking it would not compromise their system. American counter-intelligence during WWII collected all incoming and outgoing international cables. Beginning in 1946, it began an intensive effort to break into the Soviet messages with the cooperation of the British and by ... the Soviet error of using some one-time pads as two-time pads, was able, over the next 25 years, to break some 2900 messages, containing 5000 pages of the hundreds of thousands of messages that had been sent between 1941 and 1946 (when the Soviets switched to a different system).

The decryption effort was codenamed project Venona. The Venona files are most famous for exposing Julius and Ethel Rosenberg and helped give evidence of their involvement with the Soviet spy ring. Starting in 1995 all 3000 Venona decrypted messages were made public.

3.3.2 The one-time pad is malleable

Although semantic security ensures that an adversary cannot read the plaintext, it provides no guarantees for integrity. When using a stream cipher, an adversary can change a ciphertext and the modification will never be detected by the decryptor. Even worse, let us show that by changing the ciphertext, the attacker can control how the decrypted plaintext will change.

Suppose an attacker intercepts a ciphertext $c := E(s, m) = m \oplus G(s)$. The attacker changes c to $c' := c \oplus \Delta$ for some Δ of the attacker's choice. Consequently, the decryptor receives the modified message

$$D(s, c') = c' \oplus G(s) = (c \oplus \Delta) \oplus G(s) = m \oplus \Delta.$$

Hence, without knowledge of either m or s, the attacker was able to cause the decrypted message to become $m \oplus \Delta$ for Δ of the attacker's choosing. We say that stream-ciphers are **malleable** since an attacker can cause predictable changes to the plaintext. We will construct ciphers that provide both privacy and integrity in Chapter 9.

A simple example where malleability could help an attacker is an encrypted file system. To make things concrete, suppose Bob is a professor and that Alice and Molly are students. Bob's students submit their homework by email, and then Bob stores these emails on a disk encrypted using a stream cipher. An email always starts with a standard header. Simplifying things a bit, we can assume that an email from, say, Alice, always starts with the characters `From:Alice`.

Now suppose Molly is able to gain access to Bob's disk and locate the encryption of the email from Alice containing her homework. Molly can effectively steal Alice's homework, as follows. She simply XORs the appropriate five-character string into the ciphertext in positions 6 to 10, so as to change the header `From:Alice` to the header `From:Molly`. Molly makes this change by only operating on ciphertexts and without knowledge of Bob's secret key. Bob will never know that the header was changed, and he will grade Alice's homework, thinking it is Molly's, and Molly will get the credit instead of Alice.

Of course, for this attack to be effective, Molly must somehow be able to find the email from Alice on Bob's encrypted disk. However, in some implementations of encrypted file systems, file metadata (such as file names, modification times, etc) are not encrypted. Armed with this metadata, it may be straightforward for Molly to locate the encrypted email from Alice and carry out this attack.

3.4 Composing PRGs

In this section, we discuss two constructions that allow one to build new PRGs out of old PRGs. These constructions allow one to increase the size of the output space of the original PRG while at the same time preserving its security. Perhaps more important than the constructions themselves is the proof technique, which is called a **hybrid argument**. This proof technique is used pervasively throughout modern cryptography.

3.4.1 A parallel construction

Let G be a PRG defined over $(\mathcal{S}, \mathcal{R})$. Suppose that in some application, we want to use G many times. We want all the outputs of G to be computationally indistinguishable from random elements of $\mathcal{R}$. If G is a secure PRG, and if the seeds are independently generated, then this will indeed be the case.

We can model the use of many applications of G as a new PRG G'. That is, we construct a new PRG G' that applies G to n seeds, and concatenates the outputs. Thus, G' is defined over $(\mathcal{S}^n, \mathcal{R}^n)$, and for $s_1, \ldots, s_n \in \mathcal{S}$,

$$G'(s_1, \ldots, s_n) := (G(s_1), \ldots, G(s_n)).$$

We call G' the **n-wise parallel composition of G**. The value n is called a **repetition parameter**, and we require that it is a poly-bounded value.

Theorem 3.2. *If G is a secure PRG, then the n-wise parallel composition G' of G is also a secure PRG.*

> *In particular, for every PRG adversary $\mathcal{A}$ that attacks G' as in Attack Game 3.1, there exists a PRG adversary $\mathcal{B}$ that attacks G as in Attack Game 3.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
> $$\mathrm{PRG}\mathsf{adv}[\mathcal{A}, G'] = n \cdot \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G].$$

As a warm up, we first prove this theorem in the special case $n = 2$. Let $\mathcal{A}$ be an efficient PRG adversary that has advantage ϵ in attacking G' in Attack Game 3.1. We want to show that ϵ is negligible, under the assumption that G is a secure PRG. To do this, let us define **Game 0** to be Experiment 0 of Attack Game 3.1 with $\mathcal{A}$ and G'. The challenger in this game works as follows:

```
$s_1 \xleftarrow{R} \mathcal{S}$, $r_1 \leftarrow G(s_1)$
$s_2 \xleftarrow{R} \mathcal{S}$, $r_2 \leftarrow G(s_2)$
send $(r_1, r_2)$ to $\mathcal{A}$.
```

Let p_0 denote the probability with which $\mathcal{A}$ outputs 1 in this game.

Next, we define **Game 1**, which is played between $\mathcal{A}$ and a challenger that works as follows:

```
$r_1 \xleftarrow{R} \mathcal{R}$
$s_2 \xleftarrow{R} \mathcal{S}$, $r_2 \leftarrow G(s_2)$
send $(r_1, r_2)$ to $\mathcal{A}$.
```

Note that Game 1 corresponds to neither Experiment 0 nor Experiment 1 of Attack Game 3.1; rather, it is a "hybrid" experiment corresponding to something in between Experiments 0 and 1. All we have done is replace the pseudo-random value r_1 in Game 0 by a truly random value (as highlighted). Intuitively, under the assumption that G is a secure PRG, the adversary $\mathcal{A}$ should not notice the difference. To make this argument precise, let p_1 be the probability that $\mathcal{A}$ outputs 1 in Game 1.

Let $\delta_1 := |p_1 - p_0|$. We claim that δ_1 is negligible, assuming that G is a secure PRG. Indeed, we can easily construct an efficient PRG adversary $\mathcal{B}_1$ whose advantage in attacking G in Attack Game 3.1 is precisely equal to δ_1. The adversary $\mathcal{B}_1$ works as follows:

> Upon receiving $r \in \mathcal{R}$ from its challenger, $\mathcal{B}_1$ plays the role of challenger to $\mathcal{A}$, as follows:
>
> ```
> $r_1 \leftarrow r$
> $s_2 \xleftarrow{R} \mathcal{S}$, $r_2 \leftarrow G(s_2)$
> send $(r_1, r_2)$ to $\mathcal{A}$.
> ```
>
> Finally, $\mathcal{B}_1$ outputs whatever $\mathcal{A}$ outputs.

Observe that when $\mathcal{B}_1$ is in Experiment 0 of its attack game, it perfectly mimics the behavior of the challenger in Game 0, while in Experiment 1, it perfectly mimics the behavior of the challenger in Game 1. Thus, p_0 is equal to the probability that $\mathcal{B}_1$ outputs 1 in Experiment 0 of Attack Game 3.1, while p_1 is equal to the probability that $\mathcal{B}_1$ outputs 1 in Experiment 1 of Attack Game 3.1. Thus, $\mathcal{B}_1$'s advantage in attacking G is precisely $|p_1 - p_0|$, as claimed.

Next, we define **Game 2**, which is played between $\mathcal{A}$ and a challenger that works as follows:

```
$r_1 \overset{\mathrm{R}}{\leftarrow} \mathcal{R}$
$r_2 \overset{\mathrm{R}}{\leftarrow} \mathcal{R}$
send $(r_1, r_2)$ to $\mathcal{A}$.
```

All we have done is replace the pseudo-random value r_2 in Game 1 by a truly random value (as highlighted). Let p_2 be the probability that $\mathcal{A}$ outputs 1 in Game 2. Note that Game 2 corresponds to Experiment 1 of Attack Game 3.1 with $\mathcal{A}$ and G', and so p_2 is equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 1 of Attack Game 3.1 with respect to G'.

Let $\delta_2 := |p_2 - p_1|$. By an argument similar to that above, it is easy to see that δ_2 is negligible, assuming that G is a secure PRG. Indeed, we can easily construct an efficient PRG adversary $\mathcal{B}_2$ whose advantage in Attack Game 3.1 with respect to G is precisely equal to δ_2. The adversary $\mathcal{B}_2$ works as follows:

```
Upon receiving $r \in \mathcal{R}$ from its challenger, $\mathcal{B}_2$ plays the role of challenger to $\mathcal{A}$, as follows:

    $r_1 \overset{\mathrm{R}}{\leftarrow} \mathcal{R}$
    $r_2 \leftarrow r$
    send $(r_1, r_2)$ to $\mathcal{A}$.

Finally, $\mathcal{B}_2$ outputs whatever $\mathcal{A}$ outputs.
```

It should be clear that p_1 is equal to the probability that $\mathcal{B}_2$ outputs 1 in Experiment 0 of Attack Game 3.1, while p_2 is equal to the probability that $\mathcal{B}_2$ outputs 1 in Experiment 1 of Attack Game 3.1.

Recalling that $\epsilon = \mathrm{PRG}\mathsf{adv}[\mathcal{A}, G']$, then from the above discussion, we have

$$\epsilon = |p_2 - p_0| = |p_2 - p_1 + p_1 - p_0| \leq |p_1 - p_0| + |p_2 - p_1| = \delta_1 + \delta_2.$$

Since both δ_1 and δ_2 are negligible, then so is ϵ (see Fact 2.6).

That completes the proof that G' is secure in the case $n = 2$. Before giving the proof in the general case, we give another proof in the case $n = 2$. While our first proof involved the construction of two adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, our second proof combines these two adversaries into a single PRG adversary $\mathcal{B}$ that plays Attack Game 3.1 with respect to G, and which runs as follows:

> upon receiving $r \in \mathcal{R}$ from its challenger, adversary $\mathcal{B}$ chooses $\omega \in \{1, 2\}$ at random, and gives r to $\mathcal{B}_\omega$; finally, $\mathcal{B}$ outputs whatever $\mathcal{B}_\omega$ outputs.

Let W_0 be the event that $\mathcal{B}$ outputs 1 in Experiment 0 of Attack Game 3.1, and W_1 be the event that $\mathcal{B}$ outputs 1 in Experiment 1 of Attack Game 3.1. Conditioning on the events $\omega = 1$ and $\omega = 2$, we have

$$\begin{aligned}
\Pr[W_0] &= \Pr[W_0 \mid \omega = 1] \Pr[\omega = 1] + \Pr[W_0 \mid \omega = 2] \Pr[\omega = 2] \\
&= \tfrac{1}{2}\Big(\Pr[W_0 \mid \omega = 1] + \Pr[W_0 \mid \omega = 2]\Big) \\
&= \tfrac{1}{2}(p_0 + p_1).
\end{aligned}$$

Similarly, we have

$$\begin{aligned}
\Pr[W_1] &= \Pr[W_1 \mid \omega = 1] \Pr[\omega = 1] + \Pr[W_1 \mid \omega = 2] \Pr[\omega = 2] \\
&= \tfrac{1}{2}\Big(\Pr[W_1 \mid \omega = 1] + \Pr[W_1 \mid \omega = 2]\Big) \\
&= \tfrac{1}{2}(p_1 + p_2).
\end{aligned}$$

Therefore, if δ is the advantage of $\mathcal{B}$ in Attack Game 3.1 with respect to G, we have

$$\delta = \bigl|\Pr[W_1] - \Pr[W_0]\bigr| = \bigl|\tfrac{1}{2}(p_1 + p_2) - \tfrac{1}{2}(p_0 + p_1)\bigr| = \tfrac{1}{2}|p_2 - p_0| = \epsilon/2.$$

Thus, $\epsilon = 2\delta$, and since δ is negligible, so is ϵ (see Fact 2.6).

Now, finally, we present the proof of Theorem 3.2 for general, poly-bounded n.

Proof idea. We could try to extend the first strategy outlined above from $n = 2$ to arbitrary n. That is, we could construct a sequence of $n + 1$ games, starting with a challenger that produces a sequence $(G(s_1), \ldots, G(s_n))$, of pseudo-random elements replacing elements one at a time with truly random elements of $\mathcal{R}$, ending up with a sequence $(r_1, \ldots, r_n)$ of truly random elements of $\mathcal{R}$. Intuitively, the adversary should not notice any of these replacements, since G is a secure PRG; however, proving this formally would require the construction of n different adversaries, each of which attacks G in a slightly different way. As it turns out, this leads to some annoying technical difficulties when n is not an absolute constant, but is simply poly-bounded; it is much more convenient to extend the second strategy outlined above, constructing a single adversary that attacks G "in one blow." □

Proof. Let $\mathcal{A}$ be an efficient PRG adversary that plays Attack Game 3.1 with respect to G'. We first introduce a sequence of $n + 1$ **hybrid games**, called Hybrid 0, Hybrid 1, ..., Hybrid n. For $j = 0, 1, \ldots, n$, Hybrid j is a game played between $\mathcal{A}$ and a challenger that prepares a tuple of n values, the first j of which are truly random, and the remaining $n - j$ of which are pseudo-random outputs of G; that is, the challenger works as follows:

```
$r_1 \xleftarrow{R} \mathcal{R}$
    $\vdots$
$r_j \xleftarrow{R} \mathcal{R}$
$s_{j+1} \xleftarrow{R} \mathcal{S}$, $r_{j+1} \leftarrow G(s_{j+1})$
    $\vdots$
$s_n \xleftarrow{R} \mathcal{S}$, $r_n \leftarrow G(s_n)$
send $(r_1, \ldots, r_n)$ to $\mathcal{A}$.
```

As usual, $\mathcal{A}$ outputs 0 or 1 at the end of the game. Fig. 3.5 illustrates the values prepared by the challenger in each of these $n+1$ games. Let p_j denote the probability that $\mathcal{A}$ outputs 1 in Hybrid j. Note that p_0 is also equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 0 of Attack Game 3.1, while p_n is equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 1. Thus, we have

$$\text{PRG}\mathsf{adv}[\mathcal{A}, G'] = |p_n - p_0|. \tag{3.9}$$

We next define a PRG adversary $\mathcal{B}$ that plays Attack Game 3.1 with respect to G, and which works as follows:

Upon receiving $r \in \mathcal{R}$ from its challenger, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$, as follows:

Hybrid 0:	$G(s_1)$	$G(s_2)$	$G(s_3)$	$\cdots$	$G(s_n)$
Hybrid 1:	r_1	$G(s_2)$	$G(s_3)$	$\cdots$	$G(s_n)$
Hybrid 2:	r_1	r_2	$G(s_3)$	$\cdots$	$G(s_n)$
$\vdots$					
Hybrid $n-1$:	r_1	r_2	r_3	$\cdots$	$G(s_n)$
Hybrid n:	r_1	r_2	r_3	$\cdots$	r_n

Figure 3.5: Values prepared by challenger in Hybrids $0, 1, \ldots, n$. Each r_i is a random element of $\mathcal{R}$, and each s_i is a random element of $\mathcal{S}$.

```
$\omega \xleftarrow{\text{R}} \{1, \ldots, n\}$
$r_1 \xleftarrow{\text{R}} \mathcal{R}$
    $\vdots$
$r_{\omega-1} \xleftarrow{\text{R}} \mathcal{R}$
$r_\omega \leftarrow r$
$s_{\omega+1} \xleftarrow{\text{R}} \mathcal{S}$, $r_{\omega+1} \leftarrow G(s_{\omega+1})$
    $\vdots$
$s_n \xleftarrow{\text{R}} \mathcal{S}$, $r_n \leftarrow G(s_n)$
send $(r_1, \ldots, r_n)$ to $\mathcal{A}$.
```

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Let W_0 be the event that $\mathcal{B}$ outputs 1 in Experiment 0 of Attack Game 3.1, and W_1 be the event that $\mathcal{B}$ outputs 1 in Experiment 1 of Attack Game 3.1. The key observation is this:

> *conditioned on $\omega = j$ for every fixed $j = 1, \ldots, n$, Experiment 0 of $\mathcal{B}$'s attack game is equivalent to Hybrid $j-1$, while Experiment 1 of $\mathcal{B}$'s attack game is equivalent to Hybrid j.*

Therefore,

$$\Pr[W_0 \mid \omega = j] = p_{j-1} \quad \text{and} \quad \Pr[W_1 \mid \omega = j] = p_j.$$

So we have

$$\Pr[W_0] = \sum_{j=1}^{n} \Pr[W_0 \mid \omega = j] \Pr[\omega = j] = \frac{1}{n} \sum_{j=1}^{n} \Pr[W_0 \mid \omega = j] = \frac{1}{n} \sum_{j=1}^{n} p_{j-1},$$

and similarly,

$$\Pr[W_1] = \sum_{j=1}^{n} \Pr[W_1 \mid \omega = j] \Pr[\omega = j] = \frac{1}{n} \sum_{j=1}^{n} \Pr[W_1 \mid \omega = j] = \frac{1}{n} \sum_{j=1}^{n} p_j.$$

Finally, we have

$$\begin{aligned}\text{PRG}\mathsf{adv}[\mathcal{B}, G] &= |\Pr[W_1] - \Pr[W_0]| \\ &= \left|\frac{1}{n}\sum_{j=1}^{n} p_j - \frac{1}{n}\sum_{j=1}^{n} p_{j-1}\right| \\ &= \frac{1}{n}|p_n - p_0|,\end{aligned}$$

and combining this with (3.9), we have

$$\text{PRG}\mathsf{adv}[\mathcal{A}, G'] = n \cdot \text{PRG}\mathsf{adv}[\mathcal{B}, G].$$

Since we are assuming G is a secure PRG, it follows that PRG$\mathsf{adv}[\mathcal{B}, G]$ is negligible, and since n is poly-bounded, it follows that PRG$\mathsf{adv}[\mathcal{A}, G']$ is negligible (see Fact 2.6). That proves the theorem. □

Theorem 3.2 says that the security of a PRG degrades at most linearly in the number of times that we use it. One might ask if this bound is tight; that is, might security indeed degrade linearly in the number of uses? The answer is in fact “yes” (see Exercise 3.15).

3.4.2 A sequential construction: the Blum-Micali method

We now present a sequential construction, invented by Blum and Micali, which uses a PRG that stretches just a little, and builds a PRG that stretches an arbitrary amount.

Let G be a PRG defined over $(\mathcal{S}, \mathcal{R} \times \mathcal{S})$, for some finite sets $\mathcal{S}$ and $\mathcal{R}$. For every poly-bounded value $n \geq 1$, we can construct a new PRG G', defined over $(\mathcal{S}, \mathcal{R}^n \times \mathcal{S})$. For $s \in \mathcal{S}$, we let

```
$G'(s) :=$
    $s_0 \leftarrow s$
    for $i \leftarrow 1$ to $n$ do
        $(r_i, s_i) \leftarrow G(s_{i-1})$
    output $(r_1, \ldots, r_n, s_n)$.
```

We call G' the **n-wise sequential composition of** G. See Fig. 3.6 for a schematic description of G' for $n = 3$.

We shall prove below in Theorem 3.3 that if G is a secure PRG, then so is G'. As a special case of this construction, suppose G is a PRG defined over $(\{0,1\}^\ell, \{0,1\}^{t+\ell})$, for some positive integers ℓ and t; that is, G stretches ℓ-bit strings to $(t+\ell)$-bit strings. We can naturally view the output space of G as $\{0,1\}^t \times \{0,1\}^\ell$, and applying the above construction, and interpreting outputs as bit strings, we get a PRG G' that stretches ℓ-bit strings to $(nt+\ell)$-bit strings.

Theorem 3.3. *If G is a secure PRG, then the n-wise sequential composition G' of G is also a secure PRG.*

> *In particular, for every PRG adversary $\mathcal{A}$ that plays Attack Game 3.1 with respect to G', there exists a PRG adversary $\mathcal{B}$ that plays Attack Game 3.1 with respect to G, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{PRG}\mathsf{adv}[\mathcal{A}, G'] = n \cdot \text{PRG}\mathsf{adv}[\mathcal{B}, G].$$

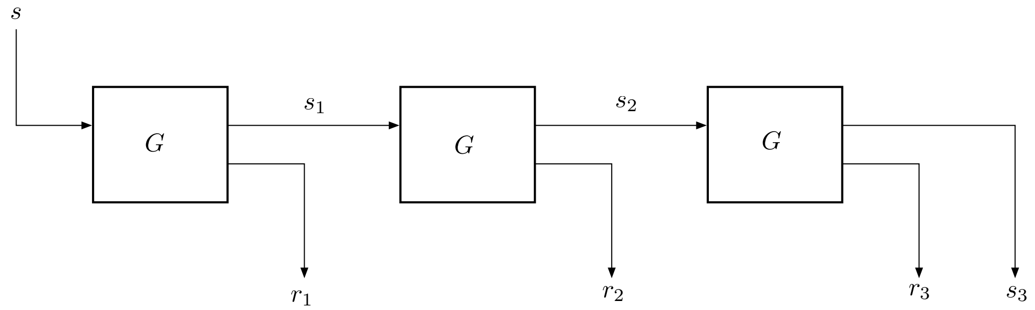


Figure 3.6: The sequential construction for $n = 3$

Proof idea. The proof of this is a hybrid argument that is very similar in spirit to the proof of Theorem 3.2. The intuition behind the proof is as follows: Consider a PRG adversary $\mathcal{A}$ who receives the $(r_1, \ldots, r_n, s_n)$ in Experiment 0 of Attack Game 3.1. Since $s = s_0$ is random and G is a secure PRG, we may replace (r_1, s_1) by a completely random element of $\mathcal{R} \times \mathcal{S}$, and the probability that $\mathcal{A}$ outputs 1 in this new, hybrid game should change by only a negligible amount. Now, since s_1 is random (and again, since G is a secure PRG), we may replace (r_2, s_2) by a completely random element of $\mathcal{R} \times \mathcal{S}$, and the probability that $\mathcal{A}$ outputs 1 in this second hybrid game should again change by only a negligible amount. Continuing in this way, we may incrementally replace (r_3, s_3) through (r_n, s_n) by random elements of $\mathcal{R} \times \mathcal{S}$, and the probability that $\mathcal{A}$ outputs 1 should change by only a negligible amount after making all these changes (assuming n is poly-bounded). However, at this point, $\mathcal{A}$ outputs 1 with the same probability with which he would output 1 in Experiment 1 in Attack Game 3.1, and therefore, this probability is negligibly close to the probability that $\mathcal{A}$ outputs 1 in Experiment 0 of Attack Game 3.1.

That is the idea; however, just as in the proof of Theorem 3.2, for technical reasons, we design a single PRG adversary that attacks G. □

Proof. Let $\mathcal{A}$ be a PRG adversary that plays Attack Game 3.1 with respect to G'. We first introduce a sequence of $n+1$ hybrid games, called Hybrid 0, Hybrid 1, …, Hybrid n. For $j = 0, 1, \ldots, n$, we define Hybrid j to be the game played between $\mathcal{A}$ and the following challenger:

```
$r_1 \xleftarrow{R} \mathcal{R}$
    $\vdots$
$r_j \xleftarrow{R} \mathcal{R}$
$s_j \xleftarrow{R} \mathcal{S}$
$(r_{j+1}, s_{j+1}) \leftarrow G(s_j)$
    $\vdots$
$(r_n, s_n) \leftarrow G(s_{n-1})$
send $(r_1, \ldots, r_n, s_n)$ to $\mathcal{A}$.
```

As usual, $\mathcal{A}$ outputs 0 or 1 at the end of the game. See Fig. 3.7 for a schematic description of how these challengers work in the case $n = 3$. Let p_j denote the probability that $\mathcal{A}$ outputs 1 in Hybrid j. Note that p_0 is also equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 0 of

Attack Game 3.1, while p_n is equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 1 of Attack Game 3.1. Thus, we have

$$\text{PRG}\mathsf{adv}[\mathcal{A}, G'] = |p_n - p_0|. \tag{3.10}$$

We next define a PRG adversary $\mathcal{B}$ that plays Attack Game 3.1 with respect to G, and which works as follows:

> Upon receiving $(r, s) \in \mathcal{R} \times \mathcal{S}$ from its challenger, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$, as follows:
>
> ```
> $\omega \xleftarrow{R} \{1, \ldots, n\}$
> $r_1 \xleftarrow{R} \mathcal{R}, \ldots, r_{\omega-1} \xleftarrow{R} \mathcal{R}$
> $(r_\omega, s_\omega) \leftarrow (r, s)$
> $(r_{\omega+1}, s_{\omega+1}) \leftarrow G(s_\omega), \ldots, (r_n, s_n) \leftarrow G(s_{n-1})$
> send $(r_1, \ldots, r_n, s_n)$ to $\mathcal{A}$.
> ```
>
> Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Let W_0 be the event that $\mathcal{B}$ outputs 1 in Experiment 0 of Attack Game 3.1, and W_1 be the event that $\mathcal{B}$ outputs 1 in Experiment 1 of Attack Game 3.1. The key observation is this:

> *conditioned on $\omega = j$ for every fixed $j = 1, \ldots, n$, Experiment 0 of $\mathcal{B}$'s attack game is equivalent to Hybrid $j - 1$, while Experiment 1 of $\mathcal{B}$'s attack game is equivalent to Hybrid j.*

Therefore,

$$\Pr[W_0 \mid \omega = j] = p_{j-1} \quad \text{and} \quad \Pr[W_1 \mid \omega = j] = p_j.$$

The remainder of the proof is a simple calculation that is *identical* to that in the last paragraph of the proof of Theorem 3.2. □

One criteria for evaluating a PRG is its **expansion rate**: a PRG that stretches an n-bit seed to an m-bit output has expansion rate of m/n; more generally, if the seed space is $\mathcal{S}$ and the output space is $\mathcal{R}$, we would define the expansion rate as $\log|\mathcal{R}|/\log|\mathcal{S}|$. The sequential composition achieves a better expansion rate than the parallel composition. However, it suffers from the drawback that it cannot be parallelized. In fact, we can obtain the best of both worlds: a large expansion rate with a highly parallelizable construction (see Section 4.4.4).

3.4.3 Mathematical details

There are some subtle points in the proofs of Theorems 3.2 and 3.3 that merit discussion.

First, in both constructions, the underlying PRG G may have system parameters. That is, there may be a probabilistic algorithm that takes as input the security parameter λ, and outputs a system parameter Λ. Recall that a system parameter is public data that fully instantiates the scheme (in this case, it might define the seed and output spaces). For both the parallel and sequential constructions, one could use the same system parameter for all n instances of G; in fact, for the sequential construction, this is necessary to ensure that outputs from one round may be used as inputs in the next round. The proofs of these security theorems are perfectly valid if the same system parameter is used for all instances of G, or if different system parameters are used.

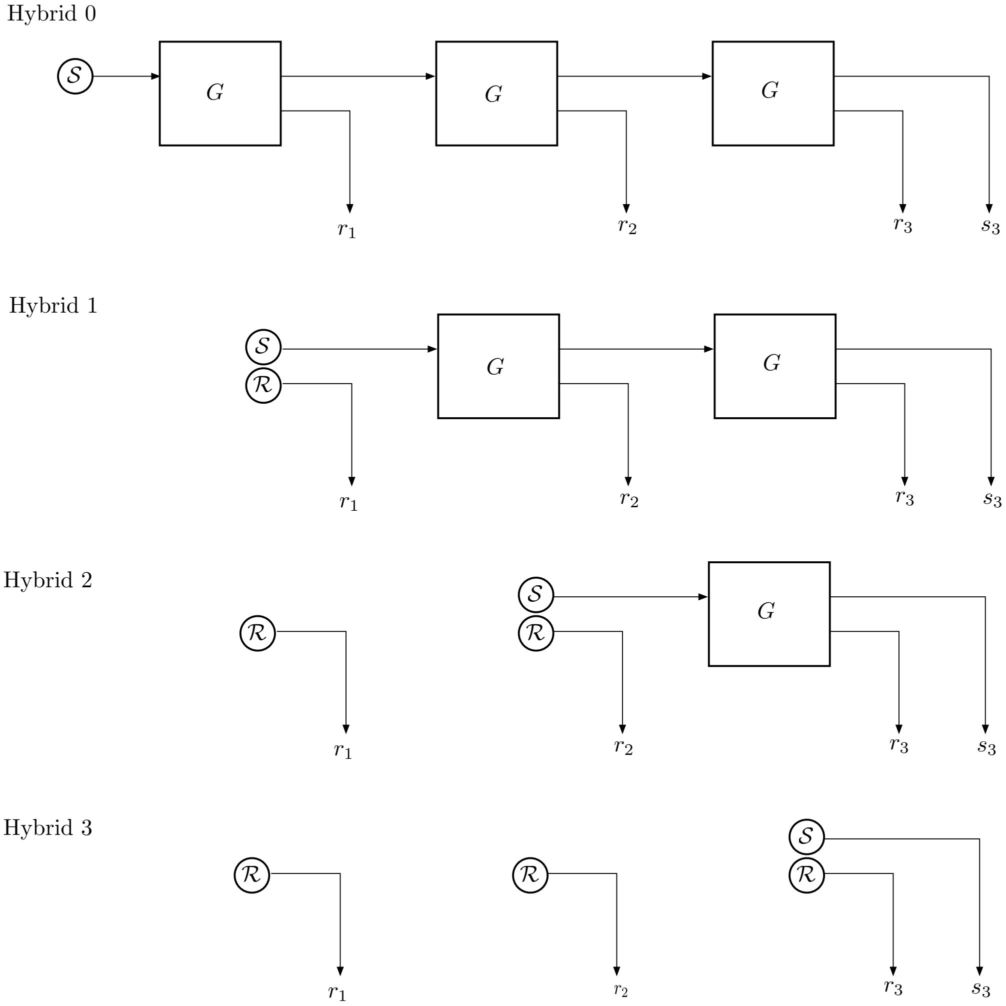


Figure 3.7: The challenger's computation in the hybrid games for $n = 3$. The circles indicate randomly generated elements of $\mathcal{S}$ or $\mathcal{R}$, as indicated by the label.

Second, we briefly discuss a rather esoteric point regarding hybrid arguments. To make things concrete, we focus attention on the proof of Theorem 3.2 (although analogous remarks apply to the proof of Theorem 3.3, or any other hybrid argument). In proving this theorem, we ultimately want to show that if there is an efficient adversary $\mathcal{A}$ that breaks G', then there is an efficient adversary that breaks G. Suppose that $\mathcal{A}$ is an efficient adversary that breaks G', so that its advantage $\epsilon(\lambda)$ (which we write here explicitly as a function of the security parameter λ) with respect to G' is not negligible. This means that there exists a constant c such that $\epsilon(\lambda) \geq 1/\lambda^c$ for infinitely many λ.

Now, in the discussion preceding the proof of Theorem 3.2, we considered the special case $n = 2$, and showed that there exist efficient adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, such that $\epsilon(\lambda) \leq \delta_1(\lambda) + \delta_2(\lambda)$ for all λ, where $\delta_j(\lambda)$ is the advantage of $\mathcal{B}_j$ with respect to G. It follows that either $\delta_1(\lambda) \geq 1/2\lambda^c$ infinitely often, or $\delta_2(\lambda) \geq 1/2\lambda^c$ infinitely often. So we may conclude that either $\mathcal{B}_1$ breaks G or $\mathcal{B}_2$ breaks G (or possibly both). Thus, *there exists* an efficient adversary that breaks G: it is either $\mathcal{B}_1$ or $\mathcal{B}_2$, which one we do not say (and we do not have to). However, whichever one it is, it is a fixed adversary that is defined uniformly for all λ; that is, it is a fixed machine that takes λ as input.

This argument is perfectly valid, and extends to every *constant* n: we would construct n adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_n$, and argue that for some $j = 1, \ldots, n$, adversary $\mathcal{B}_j$ must have advantage $1/n\lambda^c$ infinitely often, and thus break G. However, this argument does not extend to the case where n is a function of λ, which we now write explicitly as $n(\lambda)$. The problem is not that $1/(n(\lambda)\lambda^c)$ is perhaps too small (it is not). The problem is quite subtle, so before we discuss it, let us first review the (valid) proof that we did give. For each λ, we defined a sequence of $n(\lambda) + 1$ hybrid games, so that for each λ, we actually get a different sequence of games. Indeed, we cannot speak of a single, finite sequence of games that works for all λ, since $n(\lambda) \to \infty$. Nevertheless, we explicitly constructed a fixed adversary $\mathcal{B}$ that is defined uniformly for all λ; that is, $\mathcal{B}$ is a fixed machine that takes λ as input. The sequence of hybrid games that we define for each λ is a mathematical object for which we make no claims as to its computability — it is simply a convenient device used in the analysis of $\mathcal{B}$.

Hopefully by now the reader has at least a hint of the problem that arises if we attempt to generalize the argument for constant n to a function $n(\lambda)$. First of all, it is not even clear what it means to talk about $n(\lambda)$ adversaries $\mathcal{B}_1, \ldots, \mathcal{B}_{n(\lambda)}$: our adversaries are supposed to be fixed machines that take λ as input, and the machines themselves should not depend on λ. Such linguistic confusion aside, our proof for the constant case only shows that there exists an "adversary" that for infinitely many values of λ somehow knows the "right" value of $j = j(\lambda)$ to use in the $(n(\lambda) + 1)$-game hybrid argument — no single, constant value of j necessarily works for infinitely many λ. One can actually make sense of this type of argument if one uses a non-uniform model of computation, but we shall not take this approach in this text.

All of these problems simply go away when we use a hybrid argument that constructs a single adversary $\mathcal{B}$, as we did in the proofs of Theorems 3.2 and 3.3. However, we reiterate that the original analysis we did in the case where $n = 2$, or its natural extension to every constant n, is perfectly valid. In that case, we construct a single, fixed sequence of $n + 1$ games, with each individual game uniformly defined for all λ (just as our attack games are in our security definitions), as well as a finite collection of adversaries, each of which is a fixed machine. We reiterate this because in the sequel we shall often be constructing proofs that involve finite sequences of games like this (indeed, the proof of Theorem 3.1 was of this type). In such cases, each game will be uniformly defined for all λ, and will be denoted Game 0, Game 1, etc. In contrast, when we make a hybrid argument that uses non-uniform sequences of games, we shall denote these games Hybrid 0, Hybrid 1, etc., so as to avoid any possible confusion.

3.5 The next bit test

Let G be a PRG defined over $(\{0,1\}^\ell, \{0,1\}^L)$, so that it stretches ℓ-bit strings to L-bit strings. There are a number of ways an adversary might be able to distinguish a pseudo-random output of G from a truly random bit string. Indeed, suppose that an efficient adversary were able to compute, say, the last bit of G's output, given the first $L-1$ bits of G's output. Intuitively, the existence of such an adversary would imply that G is insecure, since given the first $L-1$ bits of a truly random L-bit string, one has at best a 50-50 chance of guessing the last bit. It turns out that an interesting converse, of sorts, is also true.

We shall formally define the notion of **unpredictability** for a PRG, which essentially says that given the first i bits of G's output, it is hard to predict the next bit (i.e., the $(i+1)$-st bit) with probability significantly better that 1/2 (here, i is an adversarially chosen index). We shall then prove that unpredictability and security are equivalent. The fact that security implies unpredictability is fairly obvious: the ability to effectively predict the next bit in the pseudo-random output string immediately gives an effective statistical test. However, the fact that unpredictability implies security is quite interesting (and requires more effort to prove): it says that if there is any effective statistical test at all, then there is in fact an effective method for predicting the next bit in a pseudo-random output string.

Attack Game 3.2 (Unpredictable PRG). For a given PRG G, defined over $(\mathcal{S}, \{0,1\}^L)$, and a given adversary $\mathcal{A}$, the attack game proceeds as follows:

- The adversary sends an index i, with $0 \le i \le L-1$, to the challenger.
- The challenger computes
$$s \xleftarrow{\text{R}} \mathcal{S}, \;\; r \leftarrow G(s)$$
and sends $r[0\,..\,i-1]$ to the adversary.
- The adversary outputs $g \in \{0,1\}$.

We say that $\mathcal{A}$ **wins** if $r[i] = g$, and we define $\mathcal{A}$'s **advantage** $\text{Pred}\mathsf{adv}[\mathcal{A}, G]$ to be $|\Pr[\mathcal{A} \text{ wins}] - 1/2|$. □

Definition 3.3 (Unpredictable PRG). *A PRG G is* ***unpredictable*** *if the value* $\text{Pred}\mathsf{adv}[\mathcal{A}, G]$ *is negligible for all efficient adversaries $\mathcal{A}$.*

We begin by showing that security implies unpredictability.

Theorem 3.4. *Let G be a PRG, defined over $(\mathcal{S}, \{0,1\}^L)$. If G is secure, then G is unpredictable.*

In particular, for every adversary $\mathcal{A}$ breaking the unpredictability of G, as in Attack Game 3.2, there exists an adversary $\mathcal{B}$ breaking the security of G as in Attack Game 3.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{Pred}\mathsf{adv}[\mathcal{A}, G] = \text{PRG}\mathsf{adv}[\mathcal{B}, G].$$

Proof. Let $\mathcal{A}$ be an adversary breaking the unpredictability of G, and let i denote the index chosen by $\mathcal{A}$. Also, suppose $\mathcal{A}$ wins Attack Game 3.2 with probability $1/2+\epsilon$, so that $\text{Pred}\mathsf{adv}[\mathcal{A}, G] = |\epsilon|$.

We build an adversary $\mathcal{B}$ breaking the security of G, using $\mathcal{A}$ as a subroutine, as follows:

> Upon receiving $r \in \{0,1\}^L$ from its challenger, $\mathcal{B}$ does the following:
>
> - $\mathcal{B}$ gives $r[0\mathinner{..}i-1]$ to $\mathcal{A}$, obtaining $\mathcal{A}$'s output $g \in \{0,1\}$;
> - if $r[i] = g$, then output 1, and otherwise, output 0.

For $b = 0, 1$, let W_b be the event that $\mathcal{B}$ outputs 1 in Experiment b of Attack Game 3.1. In Experiment 0, r is a pseudo-random output of G, and W_0 occurs if and only if $r[i] = g$, and so by definition

$$\Pr[W_0] = 1/2 + \epsilon.$$

In Experiment 1, r is a truly random bit string, but again, W_1 occurs if and only if $r[i] = g$; in this case, however, as random variables, the values of $r[i]$ and g are independent, and so

$$\Pr[W_1] = 1/2.$$

It follows that

$$\text{PRG}\mathsf{adv}[\mathcal{B}, G] = |\Pr[W_1] - \Pr[W_0]| = |\epsilon| = \text{Pred}\mathsf{adv}[\mathcal{A}, G]. \quad \Box$$

The more interesting, and more challenging, task is to show that unpredictability implies security. Before getting into all the details of the proof, we sketch the high level ideas.

First, we shall employ a hybrid argument, which will essentially allow us to argue that if $\mathcal{A}$ is an efficient adversary that can effectively distinguish a pseudo-random L-bit string from a random L-bit string, then we can construct an efficient adversary $\mathcal{B}$ that can effectively distinguish

$$x_1 \cdots x_j \; x_{j+1}$$

from

$$x_1 \cdots x_j \; r,$$

where j is a randomly chosen index, $x_1, \ldots, x_L$ is the pseudo-random output, and r is a random bit. Thus, adversary $\mathcal{B}$ can distinguish the pseudo-random bit x_{j+1} from the random bit r, given the "side information" $x_1, \ldots, x_j$.

We want to turn $\mathcal{B}$'s distinguishing advantage into a predicting advantage. The rough idea is this: given $x_1, \ldots, x_j$, we feed $\mathcal{B}$ the string $x_1, \ldots, x_j \; r$ for a randomly chosen bit r; if $\mathcal{B}$ outputs 1, our prediction for x_{j+1} is r; otherwise, our prediction for x_{j+1} is $\bar{r}$ (the complement of r).

That this prediction strategy works is justified by the following general result, which we call the *distinguisher/predictor lemma.* The general setup is as follows. We have:

- a random variable $\mathbf{X}$, which corresponds to the "side information" $x_1, \ldots, x_j$ above, as well as any random coins used by the adversary $\mathcal{B}$;

- a 0/1-valued random variable $\mathbf{B}$, which corresponds to x_{j+1} above, and which may be correlated with $\mathbf{X}$;

- a 0/1-valued random variable $\mathbf{R}$, which corresponds to r above, and which is independent of $(\mathbf{X}, \mathbf{B})$;

- a function d, which corresponds to $\mathcal{B}$'s strategy, so that $\mathcal{B}$'s distinguishing advantage is equal to $|\epsilon|$, where $\epsilon = \Pr[d(\mathsf{X}, \mathsf{B}) = 1] - \Pr[d(\mathsf{X}, \mathsf{R}) = 1]$.

The lemma says that if we define B' using the predicting strategy outlined above, namely $\mathsf{B}' = \mathsf{R}$ if $d(\mathsf{X}, \mathsf{R}) = 1$, and $\mathsf{B}' = \overline{\mathsf{R}}$ otherwise, then the probability that the prediction B' is equal to the actual value B is precisely $1/2 + \epsilon$. Here is the precise statement of the lemma:

Lemma 3.5 (Distinguisher/predictor lemma). *Let X be a random variable taking values in some set S, and let B and R be a 0/1-valued random variables, where R is uniformly distributed over $\{0,1\}$ and is independent of (X, B). Let $d : S \times \{0,1\} \to \{0,1\}$ be an arbitrary function, and let*

$$\epsilon := \Pr[d(\mathsf{X}, \mathsf{B}) = 1] - \Pr[d(\mathsf{X}, \mathsf{R}) = 1].$$

Define the random variable B' as follows:

$$\mathsf{B}' := \begin{cases} \mathsf{R} & \text{if } d(\mathsf{X}, \mathsf{R}) = 1; \\ \overline{\mathsf{R}} & \text{otherwise.} \end{cases}$$

Then

$$\Pr[\mathsf{B}' = \mathsf{B}] = 1/2 + \epsilon.$$

Proof. We calculate $\Pr[\mathsf{B}' = \mathsf{B}]$, conditioning on the events $\mathsf{B} = \mathsf{R}$ and $\mathsf{B} = \overline{\mathsf{R}}$:

$$\begin{aligned}
\Pr[\mathsf{B}' = \mathsf{B}] &= \Pr[\mathsf{B}' = \mathsf{B} \mid \mathsf{B} = \mathsf{R}] \Pr[\mathsf{B} = \mathsf{R}] + \Pr[\mathsf{B}' = \mathsf{B} \mid \mathsf{B} = \overline{\mathsf{R}}] \Pr[\mathsf{B} = \overline{\mathsf{R}}] \\
&= \Pr\big[d(\mathsf{X}, \mathsf{R}) = 1 \bigm| \mathsf{B} = \mathsf{R}\big]\frac{1}{2} + \Pr\big[d(\mathsf{X}, \mathsf{R}) = 0 \bigm| \mathsf{B} = \overline{\mathsf{R}}\big]\frac{1}{2} \\
&= \frac{1}{2}\Big(\Pr\big[d(\mathsf{X}, \mathsf{R}) = 1 \bigm| \mathsf{B} = \mathsf{R}\big] + \big(1 - \Pr\big[d(\mathsf{X}, \mathsf{R}) = 1 \bigm| \mathsf{B} = \overline{\mathsf{R}}\big]\big)\Big) \\
&= \frac{1}{2} + \frac{1}{2}(\alpha - \beta),
\end{aligned}$$

where

$$\alpha := \Pr[d(\mathsf{X}, \mathsf{R}) = 1 \mid \mathsf{B} = \mathsf{R}] \text{ and } \beta := \Pr[d(\mathsf{X}, \mathsf{R}) = 1 \mid \mathsf{B} = \overline{\mathsf{R}}].$$

By independence, we have

$$\alpha = \Pr[d(\mathsf{X}, \mathsf{R}) = 1 \mid \mathsf{B} = \mathsf{R}] = \Pr[d(\mathsf{X}, \mathsf{B}) = 1 \mid \mathsf{B} = \mathsf{R}] = \Pr[d(\mathsf{X}, \mathsf{B}) = 1].$$

To see the last equality, the result of Exercise 3.26 may be helpful.

We thus calculate that

$$\begin{aligned}
\epsilon &= \Pr[d(\mathsf{X}, \mathsf{B}) = 1] - \Pr[d(\mathsf{X}, \mathsf{R}) = 1] \\
&= \alpha - \Big(\Pr[d(\mathsf{X}, \mathsf{R}) = 1 \mid \mathsf{B} = \mathsf{R}] \Pr[\mathsf{B} = \mathsf{R}] + \Pr[d(\mathsf{X}, \mathsf{R}) = 1 \mid \mathsf{B} = \overline{\mathsf{R}}] \Pr[\mathsf{B} = \overline{\mathsf{R}}]\Big) \\
&= \alpha - \frac{1}{2}(\alpha + \beta) \\
&= \frac{1}{2}(\alpha - \beta),
\end{aligned}$$

which proves the lemma. □

Theorem 3.6. *Let G be a PRG, defined over $(\mathcal{S}, \{0,1\}^L)$. If G is unpredictable, then G is secure.*

> *In particular, for every adversary $\mathcal{A}$ breaking the security of G as in Attack Game 3.1, there exists an adversary $\mathcal{B}$, breaking the unpredictability of G as in Attack Game 3.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{PRG}\mathsf{adv}[\mathcal{A}, G] = L \cdot \text{Pred}\mathsf{adv}[\mathcal{B}, G].$$

Proof. Let $\mathcal{A}$ attack G as in Attack Game 3.1. Using $\mathcal{A}$, we build a predictor $\mathcal{B}$, which attacks G as in Attack Game 3.2, and works as follows:

- Choose $\omega \in \{1, \ldots, L\}$ at random.
- Send $L - \omega$ to the challenger, obtaining a string $x \in \{0,1\}^{L-\omega}$.
- Generate ω random bits $r_1, \ldots, r_\omega$, and give the L-bit string $x \parallel r_1 \cdots r_\omega$ to $\mathcal{A}$.
- If $\mathcal{A}$ outputs 1, then output r_1; otherwise, output $\overline{r}_1$.

To analyze $\mathcal{B}$, we consider $L+1$ hybrid games, called Hybrid 0, Hybrid 1, …, Hybrid L. For $j = 0, \ldots, L$, we define Hybrid j to be the game played between $\mathcal{A}$ and a challenger that generates a bit string r consisting of $L - j$ pseudo-random bits, followed by j truly random bits; that is, the challenger chooses $s \in \mathcal{S}$ and $t \in \{0,1\}^j$ at random, and sends $\mathcal{A}$ the bit string

$$r := G(s)[0 \mathinner{..} L - j - 1] \parallel t.$$

As usual, $\mathcal{A}$ outputs 0 or 1 at the end of the game, and we define p_j to be the probability that $\mathcal{A}$ outputs 1 in Hybrid j. Note that p_0 is the probability that $\mathcal{A}$ outputs 1 in Experiment 0 of Attack Game 3.1, while p_L is the probability that $\mathcal{A}$ outputs 1 in Experiment 1 of Attack Game 3.1.

Let W be the event that $\mathcal{B}$ wins in Attack Game 3.2 (that is, correctly predicts the next bit). Then we have

$$\begin{aligned}
\Pr[W] &= \sum_{j=1}^{L} \Pr[W \mid \omega = j] \Pr[\omega = j] \\
&= \frac{1}{L} \sum_{j=1}^{L} \Pr[W \mid \omega = j] \\
&= \frac{1}{L} \sum_{j=1}^{L} \Big(\frac{1}{2} + p_{j-1} - p_j\Big) \quad \text{(by Lemma 3.5)} \\
&= \frac{1}{2} + \frac{1}{L}(p_0 - p_L),
\end{aligned}$$

and the theorem follows. □

3.6 Case study: the Salsa and ChaCha PRGs

There are many ways to build PRGs and stream ciphers in practice. One approach builds PRGs using the Blum-Micali paradigm discussed in Section 3.4.2. Another approach, discussed more generally in Chapter 5, builds them from a more versatile primitive called a *pseudorandom function* in counter mode. We start with a construction that uses this latter approach.

Salsa20/12 and Salsa20/20 are fast stream ciphers designed by Dan Bernstein in 2005. Salsa20/12 is one of four Profile 1 stream ciphers selected for the eStream portfolio of stream ciphers. eStream is a project that identifies fast and secure stream ciphers that are appropriate for practical use. Variants of Salsa20/12 and Salsa20/20, called ChaCha12 and ChaCha20 respectively, were proposed by Bernstein in 2008. These stream ciphers have been incorporated into several widely deployed protocols such as TLS and SSH.

Let us briefly describe the PRGs underlying the Salsa and ChaCha stream cipher families. These PRGs take as input a 256-bit seed and a 64-bit nonce. For now we ignore the nonce and simply set it to 0. We discuss the purpose of the nonce at the end of this section. The Salsa and ChaCha PRGs follow the same high level structure shown in Fig. 3.8. They make use of two components:

- A padding function denoted $\text{pad}(s, j, 0)$ that combines a 256-bit seed s with a 64-bit counter j to form a 512-bit block. The third input, a 64-bit nonce, is always set to 0 for now.

- A fixed public permutation $\pi : \{0,1\}^{512} \to \{0,1\}^{512}$.

These components are used to output $L < 2^{64}$ pseudorandom blocks, each 512 bits long, using the following algorithm (Fig. 3.8):

```
input: seed $s \in \{0,1\}^{256}$
1.  for $j \leftarrow 0$ to $L-1$
2.      $h_j \leftarrow \text{pad}(s, j, 0) \in \{0,1\}^{512}$
3.      $r_j \leftarrow \pi(h_j) \oplus h_j$
4.  output $(r_0, \ldots, r_{L-1})$.
```

The final PRG output is $512 \cdot L$ bits long. We note that in Salsa and ChaCha the XOR on line 3 is a slightly more complicated operation: the 512-bit operands h_j and $\pi(h_j)$ are split into 16 words each 32-bits long and then added word-wise mod 2^{32}.

The design of Salsa and ChaCha is highly parallelizable and can take advantage of multiple processor cores to speed-up encryption. Moreover, it enables random access to output blocks: output block number j can be computed without having to first compute all previous blocks. Generators based on the Blum-Micali paradigm do not have these properties.

We analyze the security of the Salsa and ChaCha design in Exercise 4.23 in the next chapter, after we develop a few more tools.

The details. We briefly describe the padding function $\text{pad}(s, j, n)$ and the permutation π used in ChaCha20. The padding function takes as input a 256-bit seed $s_0, \ldots, s_7 \in \{0,1\}^{32}$, a 64-bit counter $j_0, j_1 \in \{0,1\}^{32}$, and 64-bit nonce $n_0, n_1 \in \{0,1\}^{32}$. It outputs a 512-bit block denoted $x_0, \ldots, x_{15} \in \{0,1\}^{32}$. The output is arranged in a 4×4 matrix of 32-bit words as follows:

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \longleftarrow \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ j_0 & j_1 & n_0 & n_1 \end{pmatrix} \tag{3.11}$$

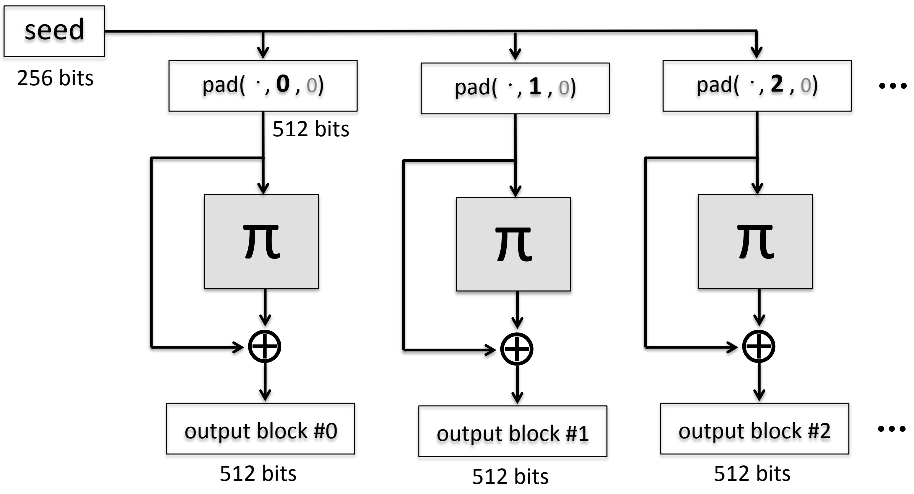


Figure 3.8: A schematic of the Salsa and ChaCha PRGs

where c_0, c_1, c_2, c_3 are fixed 32-bit constants.

The permutation $\pi : \{0,1\}^{512} \to \{0,1\}^{512}$ is constructed by iterating a simple permutation a fixed number of times. The 512-bit input to π is treated as a 4×4 array of 32-bit words denoted by $x_0, \dots, x_{15}$. In ChaCha20 the function π is implemented by repeating the following sequence of steps ten times:

(1) QuarterRound(x_0, x_4, x_8, x_{12}), (2) QuarterRound(x_1, x_5, x_9, x_{13}),
(3) QuarterRound$(x_2, x_6, x_{10}, x_{14})$, (4) QuarterRound$(x_3, x_7, x_{11}, x_{15})$,
(5) QuarterRound$(x_0, x_5, x_{10}, x_{15})$, (6) QuarterRound$(x_1, x_6, x_{11}, x_{12})$,
(7) QuarterRound(x_2, x_7, x_8, x_{13}), (8) QuarterRound(x_3, x_4, x_9, x_{14}).

Here QuarterRound(`a`, `b`, `c`, `d`) is defined as the following sequence of steps written as C code using a macro `ROTL(a,b)` that rotates left a 32-bit word `a` by `b` bits:

```
#define ROTL(a,b) (((a) << (b)) | ((a) >> (32 - (b))))
a += b;  d ^= a;  ROTL(d, 16);
c += d;  b ^= c;  ROTL(b, 12);
a += b;  d ^= a;  ROTL(d, 8);
c += d;  b ^= c;  ROTL(b, 7);
```

Observe that the first four invocations of QuarterRound, steps (1-4), are applied to each of the four columns of the 4×4 matrix, from left to right. The next four invocations, steps (5-8), are applied to each of the four diagonals, with wrap around. This completes our description of ChaCha20, except that we still need to discuss the use of nonces.

Using nonces. While the PRGs we discussed so far only take the seed as input, many PRGs used in practice take an additional input called a *nonce*. That is, the PRG is a function $G : \mathcal{S} \times \mathcal{N} \to \mathcal{R}$ where $\mathcal{S}$ and $\mathcal{R}$ are as before and $\mathcal{N}$ is called a *nonce space*. The nonce lets us generate multiple pseudorandom outputs from a single seed s. That is, $G(s, n_0)$ is one pseudorandom output and

$G(s, n_1)$ for $n_1 \neq n_0$ is another. The nonce turns the PRG into a more powerful primitive called a *pseudorandom function* discussed in the next chapter. As we will see, secure pseudorandom functions make it possible to use the same seed to encrypt multiple messages securely.

3.7 Case study: linear generators

In this section we look at two example PRGs built from linear functions. Both generators follow the Blum-Micali paradigm presented in Section 3.4.2. Our first example, called a *linear congruential generator*, is completely insecure. We present it to show the beautiful mathematics that comes up when attacking PRGs. Our second example, called a *subset sum generator*, is a provably secure PRG assuming a certain version of the classic subset-sum problem is hard.

3.7.1 An example cryptanalysis: the linear congruential generator

Linear congruential generators (LCG) are used in statistical simulations to generate pseudorandom values. They are fast, easy to implement, and widely deployed. Variants of LCG are used to generate randomness in early versions of `glibc`, Microsoft Visual Basic, and the Java runtime. While these generators may be sufficient for simulations, they should *never* be used for cryptographic applications because they are insecure as PRGs. In particular, they are predictable: given a few consecutive outputs of an LCG generator it is easy to compute all subsequent outputs. In this section we describe an attack on LCG generators by showing a prediction algorithm.

The basic linear congruential generator is specified by four public system parameters: an integer q, two constants $a, b \in \{0, \ldots, q-1\}$, and a positive integer $w \leq q$. The constant a is taken to be relatively prime to q. We use $\mathcal{S}_q$ and $\mathcal{R}$ to denote the sets:

$$\mathcal{S}_q := \{0, \ldots, q-1\}; \quad \mathcal{R} := \big\{0, \ldots, \lfloor (q-1)/w \rfloor\ \big\}.$$

Here $\lfloor \cdot \rfloor$ is the floor function: for a real number x, $\lfloor x \rfloor$ is the biggest integer less than or equal to x. Now, the generator $G_{\text{lcg}} : \mathcal{S}_q \to \mathcal{R} \times \mathcal{S}_q$ with seed $s \in \mathcal{S}_q$ is defined as follows:

$$G_{\text{lcg}}(s) := \big(\ \lfloor s/w \rfloor, \quad as + b \bmod q\ \big).$$

When w is a power of 2, say $w = 2^t$, then the operation $\lfloor s/w \rfloor$ simply erases the t least significant bits of s. Hence, the left part of $G_{\text{lcg}}(s)$ is the result of dropping the t least significant bits of s.

The generator G_{lcg} is clearly insecure since given $s' := as + b \bmod q$ it is straight-forward to recover s and then distinguish $\lfloor s/w \rfloor$ from random. Nevertheless, consider a variant of the Blum-Micali construction in which the final $\mathcal{S}_q$-value is not output:

```
$G^{(n)}_{\text{lcg}}(s) :=$    $s_1 \leftarrow s$
                 for $i \leftarrow 1$ to $n$ do
                     $r_i \leftarrow \lfloor s_i/w \rfloor$,   $s_{i+1} \leftarrow as_i + b \bmod q$
                 output $(r_1, \ldots, r_n)$.
```

We refer to each iteration of the loop as a single iteration of the LCG generator and call each one of $r_1, \ldots, r_n$ the output of a single iteration.

Different implementations use different system parameters q, a, b, w. For example, the `Math.random` function in the Java 8 Development Kit (JDKv8) uses $q := 2^{48}$, $w := 2^{22}$, and the hexadecimal constants $a := \text{0x5DEECE66D}$ and $b := \text{0x0B}$. Thus, every iteration of the LCG generator outputs the top $48 - 22 = 26$ bits of the 48-bit state s_i.

The parameters used by this Java 8 generator are clearly too small for security applications since the output $r_1 \in \mathcal{R}$ of the first iteration of the generator reveals all but 22 bits of the seed $s \in \mathcal{S}_q$. An attacker can easily recover the unknown 22 bits of s by exhaustive search. For every possible value of the 22 bits, the attacker forms a candidate seed $\hat{s} \in \mathcal{S}_q$. It tests if $\hat{s}$ is the correct seed by checking if the outputs $\hat{r}_1, \hat{r}_2, \hat{r}_3 \in \mathcal{R}$ computed from the seed $\hat{s}$ are equal to the outputs $r_1, r_2, r_3 \in \mathcal{R}$ observed from the actual generator. By trying all 2^{22} candidates (about four million) the attacker eventually finds the correct seed s and can then predict *all* subsequent outputs of the generator. This attack runs in under a second on a modern processor.

Even when the LCG parameters are sufficiently large to prevent exhaustive search, say $q = 2^{512}$, the generator $G_{\text{lcg}}^{(n)}$ is insecure and should never be used for security applications despite its wide availability in software libraries. Known attacks [65] on the LCG show that even if the generator outputs only a few bits per iteration, it is still possible to predict the entire sequence from just a few consecutive outputs. Let us see an elegant version of this attack.

Cryptanalysis. Suppose that q is large (e.g. $q = 2^{512}$) and the LCG generator $G_{\text{lcg}}^{(n)}$ outputs about half the bits of the state s per iteration, as in the Java 8 `Math.random` generator. An exhaustive search on the seed s is not possible given its size. Nevertheless, we show how to quickly predict the generator from the output of only two consecutive iterations.

Suppose that $w \le \lfloor \sqrt{q}/c \rfloor$ for some fixed $c > 0$, say $c = 32$. This means that at every iteration the generator outputs slightly more than half the bits of the current internal state s_i. For example, when $q = 2^{512}$ and $c = 32$ the generator would output at least 261 bits per iteration.

Suppose the attacker is given two consecutive outputs of the generator $r_i, r_{i+1} \in \mathcal{R}$. We show how it can predict the remaining sequence. The attacker knows that

$$r_i = \lfloor s_i/w \rfloor \qquad \text{and} \qquad r_{i+1} = \lfloor s_{i+1}/w \rfloor = \lfloor (as_i + b \bmod q)/w \rfloor \ .$$

for some unknown $s_i \in \mathcal{S}_q$. By multiplying both equation by w we obtain

$$r_i \cdot w + e_0 = s_i \qquad \text{and} \qquad r_{i+1} \cdot w + e_1 = (as_i + b \bmod q),$$

where e_0 and e_1 are the remainders after dividing s_i and s_{i+1} by w. In particular,

$$0 \le e_0, e_1 < w \le \lfloor \sqrt{q}/c \rfloor.$$

The fact that e_0, e_1 are smaller than $\sqrt{q}/c$ is an essential ingredient of the attack. Next, let us write s in place of s_i, and eliminate the mod q by introducing an integer variable x, to obtain

$$r_i \cdot w + e_0 = s \qquad \text{and} \qquad r_{i+1} \cdot w + e_1 = as + b + qx \ .$$

The integers x, s, e_0, e_1 are unknown to the attacker, but it has the integers r_i, r_{i+1}, w, a, b. Rearranging terms to put the terms involving x and s on the left gives

$$s = r_i \cdot w + e_0 \qquad \text{and} \qquad as + qx = r_{i+1}w - b + e_1 \ . \tag{3.12}$$

Finally, we can write (3.12) in vector form as

$$s \cdot \begin{pmatrix} 1 \\ a \end{pmatrix} + x \cdot \begin{pmatrix} 0 \\ q \end{pmatrix} = \boldsymbol{g} + \boldsymbol{e} \qquad \text{where} \qquad \boldsymbol{g} := \begin{pmatrix} r_i w \\ r_{i+1}w - b \end{pmatrix} \quad \text{and} \quad \boldsymbol{e} := \begin{pmatrix} e_0 \\ e_1 \end{pmatrix} \ . \tag{3.13}$$

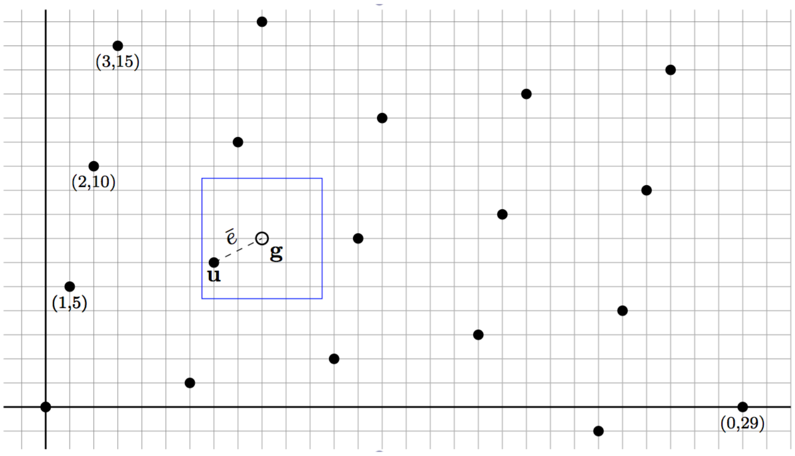


Figure 3.9: The two-dimensional lattice $\mathcal{L}_a$ associated with attacking the LCG. Here the lattice is generated by the vectors $(1,5)^{\intercal}$ and $(0,29)^{\intercal}$. The attacker has a vector $\boldsymbol{g} = (9,7)^{\intercal} \in \mathbb{Z}^2$ and wants to find the closest lattice vector $\boldsymbol{u} \in \mathcal{L}_a$. In this picture there is indeed only one "close" lattice vector to $\boldsymbol{g}$, namely $\boldsymbol{u} = (7,6)^{\intercal}$.

The attacker knows $\boldsymbol{g} \in \mathbb{Z}^2$, but it does not know s, x, or $\boldsymbol{e} \in \mathbb{Z}^2$. However, it knows that $\boldsymbol{e}$ is short, namely $\|\boldsymbol{e}\|_\infty$ is less than $\lfloor\sqrt{q}/c\rfloor$.

Let $\boldsymbol{u} \in \mathbb{Z}^2$ denote the unknown vector $\boldsymbol{u} := \boldsymbol{g} + \boldsymbol{e} = s \cdot (1,a)^{\intercal} + x \cdot (0,q)^{\intercal}$. If the attacker could find $\boldsymbol{u}$ then it could easily recover s and x from $\boldsymbol{u}$ by linear algebra. Using s it could predict the rest of the PRG output. Thus, to break the generator it suffices to find the vector $\boldsymbol{u} \in \mathbb{Z}^2$. The attacker has $\boldsymbol{g} \in \mathbb{Z}^2$, and it knows that $\|\boldsymbol{g} - \boldsymbol{u}\|_\infty = \|\boldsymbol{e}\|_\infty$ is short, meaning that $\boldsymbol{u}$ is "close" to $\boldsymbol{g}$.

We show how to find $\boldsymbol{u}$ from $\boldsymbol{g}$. Consider the set of all integer linear combinations of the vectors $(1,a)^{\intercal}$ and $(0,q)^{\intercal}$. This set, denoted by $\mathcal{L}_a$, is a subset of $\mathbb{Z}^2$ and contains vectors like $(1,a)^{\intercal}$, $(2,2a)^{\intercal}$, $(3,3a-2q)^{\intercal}$, and so on. The set $\mathcal{L}_a$ is illustrated in Fig. 3.9 where the solid dots in the figure are the integer linear combinations of the vectors $(1,a)^{\intercal}$ and $(0,q)^{\intercal}$. The set $\mathcal{L}_a$ is called the two-dimensional **lattice** generated by the vectors $(1,a)^{\intercal}$ and $(0,q)^{\intercal}$.

Now, the attacker has a vector $\boldsymbol{g} \in \mathbb{Z}^2$ and knows that his target vector $\boldsymbol{u} \in \mathcal{L}_a$ is close to $\boldsymbol{g}$. If it could find the *closest* vector in $\mathcal{L}_a$ to $\boldsymbol{g}$ then there is a good chance that this vector is the desired vector $\boldsymbol{u}$. The following lemma shows that indeed this is the case for most $a \in \mathcal{S}_q$.

Lemma 3.7. *For at least $(1 - 16/c^2) \cdot q$ of the a in $\mathcal{S}_q$, the lattice $\mathcal{L}_a \subseteq \mathbb{Z}^2$ has the following property: for every $\boldsymbol{g} \in \mathbb{Z}^2$ there is at most one vector $\boldsymbol{u} \in \mathcal{L}_a$ such that $\|\boldsymbol{g} - \boldsymbol{u}\|_\infty < \lfloor\sqrt{q}/c\rfloor$.*

Taking $c = 32$ in Lemma 3.7 shows that for 98% of the $a \in \mathcal{S}_q$, the closest vector to $\boldsymbol{g}$ in $\mathcal{L}_a$ is precisely the desired vector $\boldsymbol{u}$. Before proving the lemma, let us first complete the description of the attack.

It remains to efficiently find the closest vector to $\boldsymbol{g}$ in $\mathcal{L}_a$. This problem is a special case of a general problem called the **closest vector problem**: given a lattice $\mathcal{L}$ and a vector $\boldsymbol{g}$, find the vector in $\mathcal{L}$ that is closest to $\boldsymbol{g}$. When the lattice $\mathcal{L}$ is two dimensional there is an efficient polynomial time algorithm for this problem [153]. Armed with this algorithm the attacker can recover the internal state s_i of the LCG generator from just two outputs r_i, r_{i+1} of the generator and predict the remaining sequence. This attack works for 98% of the $a \in \mathcal{S}_q$.

For completeness we note that $a = 1$ and $a = 2$ are examples where Lemma 3.7 fails. For these a there may be many lattice vectors in $\mathcal{L}_a$ close to a given $\boldsymbol{g} \in \mathbb{Z}^2$, and the attack will fail. We leave it as a fun exercise to devise an attack that works for the a in $\mathcal{S}_q$ for which Lemma 3.7 does not apply. We conclude this section with a proof of Lemma 3.7.

Proof of Lemma 3.7. Let $\boldsymbol{g} \in \mathbb{Z}^2$ and suppose there are two vectors $\boldsymbol{u}_0$ and $\boldsymbol{u}_1$ in $\mathcal{L}_a$ that are close to $\boldsymbol{g}$, that is, $\|\boldsymbol{u}_i - \boldsymbol{g}\|_\infty < \lfloor \sqrt{q}/c \rfloor$ for $i = 0, 1$. Then $\boldsymbol{u}_0$ and $\boldsymbol{u}_1$ must be close to each other. Indeed, by the triangle inequality, we have

$$\|\boldsymbol{u}_0 - \boldsymbol{u}_1\|_\infty \le \|\boldsymbol{u}_0 - \boldsymbol{g}\|_\infty + \|\boldsymbol{g} - \boldsymbol{u}_1\|_\infty < 2\lfloor \sqrt{q}/c \rfloor \ .$$

Since any lattice is closed under addition, we know that $\boldsymbol{u} := \boldsymbol{u}_0 - \boldsymbol{u}_1$ is a vector in the lattice $\mathcal{L}_a$, and we conclude that $\mathcal{L}_a$ must contain a "short" vector, namely, a non-zero vector of norm less than $B := 2\lfloor \sqrt{q}/c \rfloor$. So, let us bound the number of "bad" $a \in \mathcal{S}_q$ for which $\mathcal{L}_a$ contains a short vector of norm less than B.

First, consider the case when q is a prime. We show that every short vector in $\mathbb{Z}^2$ is contained in at most one lattice $\mathcal{L}_a$. Therefore, the number of bad a's is at most the number of short vectors in $\mathbb{Z}^2$. Let $\boldsymbol{t} = (s, y)^\intercal \in \mathbb{Z}^2$ be some non-zero vector such that $\|\boldsymbol{t}\|_\infty < B$. Suppose that $\boldsymbol{t} \in \mathcal{L}_a$ for some $a \in \mathcal{S}_q$. Then there exist integers s_a and x_a such that $s_a \cdot (1, a)^\intercal + x_a \cdot (0, q)^\intercal = \boldsymbol{t} = (s, y)^\intercal$. From this we obtain that $s = s_a$ and $y = as \bmod q$. Moreover, $s \neq 0$ since otherwise $\boldsymbol{t} = \boldsymbol{0}$. Since $y = as \bmod q$ and $s \neq 0$, the value of a is uniquely determined, namely, $a = ys^{-1} \bmod q$. Hence, when q is prime, every non-zero short vector $\boldsymbol{t}$ is contained in at most one lattice $\mathcal{L}_a$ for some $a \in \mathcal{S}_q$. It follows that the number of bad a in $\mathcal{S}_q$ is at most the number of vectors in $\boldsymbol{t} \in \mathbb{Z}^2$ where $\|\boldsymbol{t}\|_\infty < B$, which is at most $(2B)^2 \le 16q/c^2$.

The same bound on the number of bad a's holds when q is not a prime. To see why consider a specific non-zero $s \in \mathcal{S}_q$ and let $d = \gcd(s, q)$. As above, a vector $\boldsymbol{t} = (s, y)^\intercal$ is contained in some lattice $\mathcal{L}_a$ only if there is an $a \in \mathcal{S}_q$ satisfying $as \equiv y \pmod{q}$. This implies that y must be a multiple of d so that we need only consider $2B/d$ possible values of y. For each such y the vector $\boldsymbol{t} = (s, y)^\intercal$ is in at most d lattices $\mathcal{L}_a$. Since $\|\boldsymbol{t}\|_\infty < B$, there are at most $2B$ possible values for s. Hence, the number of bad a's is bounded by $(d \cdot 2B/d) \cdot 2B = (2B)^2$ as in the case when q is prime.

To conclude, there are at most $16q/c^2$ bad values of a in $\mathcal{S}_q$. Therefore, for at least $(1 - 16/c^2) \cdot q$ of the a values in $\mathcal{S}_q$, the lattice $\mathcal{L}_a$ contains no non-zero short vectors and the lemma follows. □

3.7.2 The subset sum generator

We next show how to construct a pseudorandom generator from simple linear operations. The generator is secure assuming that a certain randomized version of the classic *subset sum problem* is hard.

The modular subset problem. Let q be a positive integer and set $\mathcal{S}_q := \{0, \ldots, q-1\} \subseteq \mathbb{Z}$. Choose n integers $\boldsymbol{a} := (a_1, \ldots, a_n)$ in $\mathcal{S}_q$ and define the subset sum function $f_{\boldsymbol{a}} : \{0,1\}^n \to \mathcal{S}_q$ as

$$\text{for } \boldsymbol{s} = (s_1, \ldots, s_n) \in \{0,1\}^n \ \text{ define } \quad f_{\boldsymbol{a}}(\boldsymbol{s}) := \sum_{i=1}^{n} a_i \cdot s_i \bmod q \ .$$

For example, $f_{\boldsymbol{a}}(101101) = a_1 + a_3 + a_4 + a_6 \bmod q$. Now, for a target integer $t \in \mathcal{S}_q$ the modular subset problem is defined as follows:

given $(q, \boldsymbol{a}, t)$ as input, output a vector $\boldsymbol{s} \in \{0,1\}^n$ such that $f_{\boldsymbol{a}}(\boldsymbol{s}) = t$, if one exists.

In other words, the problem is to find a pre-image of t for the function $f_{\boldsymbol{a}}(\cdot)$, if one exists. The modular subset problem is known to be $\mathcal{NP}$ hard.

The subset sum PRG. The subset problem naturally suggests the following PRG: at setup time fix an integer q and choose n random integers $\boldsymbol{a} := (a_1, \dots, a_n)$ in $\mathcal{S}_q$. The PRG $G_{q,\boldsymbol{a}}$ takes a seed $\boldsymbol{s} \in \{0,1\}^n$ and outputs a pseudorandom value in $\mathcal{S}_q$. It is defined as

$$G_{q,\boldsymbol{a}}(\boldsymbol{s}) := \sum_{i=1}^{n} a_i \cdot s_i \bmod q \ .$$

The PRG expands an n bit seed to an element of $\mathcal{S}_q$, which is about $\log_2 q$ bits of output. Choosing n and q so that $\log_2 q$ is somewhat bigger than $2n$ gives a PRG whose output is about twice the size of the input. We can then plug this PRG into the Blum-Micali construction to expand the output further. Note that if the output of the PRG needs to be converted to a binary string, then q needs to be close to a power of 2, otherwise the most significant bit of the output will be biased.

While $G_{q,\boldsymbol{a}}$ is far slower than custom PRG constructions like ChaCha20 from Section 3.6, the work on average per bit of output is a single modular addition in $\mathcal{S}_q$, which may be appropriate for some applications that are not time sensitive.

Impagliazzo and Naor [91] show that attacking $G_{q,\boldsymbol{a}}$ as a PRG is as hard as solving a certain randomized variant of the modular subset sum problem. While there is considerable work on solving the modular subset problem, the problem appears to be hard when $\log_2 q$ is approximately $2n$, and n is large, say $n > 1000$. This shows the security of $G_{q,\boldsymbol{a}}$ as a PRG.

Variants. Fischer and Stern [62] and others propose the following variation of the subset sum generator:

$$G_{q,A}(\boldsymbol{s}) := A \cdot \boldsymbol{s} \bmod q$$

where q is a small prime, A is a random matrix in $\mathcal{S}_q^{n \times m}$ for $n < m$, and the seed $\boldsymbol{s}$ is uniform in $\{0,1\}^m$. The generator maps an m-bit seed to $n \log_2 q$ bits of output. We discuss this generator further in Chapter 17.

3.8 Case study: cryptanalysis of the DVD encryption system

The Content Scrambling System (CSS) is a system used for protecting movies on DVD disks. It uses a stream cipher, called the CSS stream cipher, to encrypt movie contents. CSS was designed in the 1980's when exportable encryption was restricted to 40-bit keys. As a result, CSS encrypts movies using a 40-bit secret key. While ciphers using 40-bit keys are woefully insecure, we show that the CSS stream cipher is particularly weak and can be broken in far less time than an exhaustive search over all 2^{40} keys. It provides a fun opportunity for cryptanalysis.

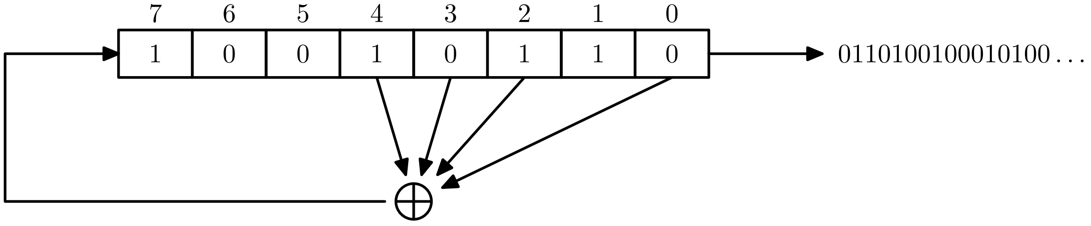


Figure 3.10: The 8 bit linear feedback shift register $\{4, 3, 2, 0\}$

Linear feedback shift registers (LFSR). The CSS stream cipher is built from two LFSRs. An n-bit LFSR is defined by a set of integers $V := \{v_1, \ldots, v_d\}$ where each v_i is in the range $\{0, \ldots, n-1\}$. The elements of V are called **tap positions**. An LFSR gives a PRG as follows (Fig. 3.10):

```
Input:    $s = (b_{n-1}, \ldots, b_0) \in \{0,1\}^n$ and $s \neq 0^n$
Output:   $y \in \{0,1\}^\ell$ where $\ell > n$

for $i \leftarrow 1 \ldots \ell$ do
    output $b_0$                              // output one bit
    $b \leftarrow b_{v_1} \oplus \cdots \oplus b_{v_d}$        // compute feedback bit
    $s \leftarrow (b,\ b_{n-1}, \ldots,\ b_1)$          // shift register bits to the right
```

The LFSR outputs one bit per clock cycle. Note that if an LFSR is started in state $s = 0^n$ then its output is degenerate, namely all 0. For this reason one of the seed bits is always set to 1.

LFSR can be implemented in hardware with few transistors. As a result, stream ciphers built from LFSR are attractive for low-cost consumer electronics such as DVD players, cell phones, and Bluetooth devices.

Stream ciphers from LSFRs. A single LFSR is completely insecure as a PRG since given n consecutive bits of its output it is trivial to compute all subsequent bits. Nevertheless, by combining several LFSRs using a non-linear component it is possible to get some (weak) security as a PRG. Trivium, one of the eStream portfolio stream ciphers, is built this way.

One approach to building stream ciphers from LFSRs is to run several LFSRs in parallel and combine their output using a non-linear operation. The CSS stream cipher, described next, combines two LFSRs using addition over the integers. The A5/1 stream cipher used to encrypt GSM cell phone traffic combines the outputs of three LFSRs. The Bluetooth E0 stream cipher combines four LFSRs using a 2-bit finite state machine. All these algorithms have been shown to be insecure and should not be used: recovering the plaintext takes far less time than an exhaustive search on the key space.

Another approach is to run a single LFSR and generate the output from a non-linear operation on its internal state. The `snow` 3G cipher used to encrypt 3GPP cell phone traffic operates this way.

The CSS stream cipher. The CSS stream cipher is built from the PRG shown in Fig. 3.11. The PRG works as follows:

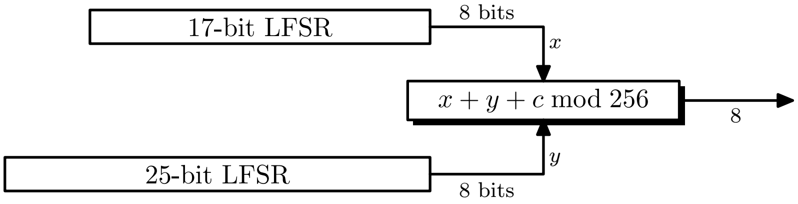


Figure 3.11: The CSS stream cipher

```
Input:    seed $s \in \{0,1\}^{40}$
Output:   $\ell$ bytes
write $s = s_1 \| s_2$ where $s_1 \in \{0,1\}^{16}$ and $s_2 \in \{0,1\}^{24}$
load $1 \| s_1$ into a 17-bit LFSR
load $1 \| s_2$ into a 25-bit LFSR
$c \leftarrow 0$   // carry bit
for $i = 1, \ldots, \ell$:
    run both LFSRs for eight cycles to obtain $x_i, y_i \in \{0,1\}^8$
    treat $x_i$ and $y_i$ as integers in $0 \ldots 255$
    output $z_i := x_i + y_i + c \bmod 256$
    if $x_i + y_i > 255$ then $c \leftarrow 1$ else $c \leftarrow 0$   // carry bit
```

The PRG outputs one byte per iteration. Prepending 1 to both s_1 and s_2 ensures that the LFSRs are never initialized to the all 0 state. The taps for both LFSRs are fixed. The 17-bit LFSR uses taps $\{14, 0\}$. The 25-bit LFSR uses taps $\{12, 4, 3, 0\}$.

The CSS PRG we presented is a minor variation of CSS that is a little easier to describe, but has the same security. In the real CSS, instead of prepending a 1 to the initial seeds, one inserts the 1 in bit position 9 for the 17-bit LFSR and in bit position 22 for the 25-bit LFSR. In addition, the real CSS discards the first byte output by the 17-bit LFSR and the first two bytes output by the 25-bit LFSR. Neither issue affects the analysis presented next.

Insecurity of CSS. Given the PRG output, one can clearly recover the secret seed in time 2^{40} by exhaustive search over the seed space. We show a much faster attack that takes only 2^{16} guesses. Suppose we are given the first 100 bytes $\bar{z} := (z_1, z_2, \ldots)$ output by the PRG. The attack is based on the following observation:

> Let (x_1, x_2, x_3) and (y_1, y_2, y_3) be the first three bytes output by the 17-bit and 25-bit LFSR, respectively. Then
>
> $$(2^{16}x_3 + 2^8 x_2 + x_1) + (2^{16}y_3 + 2^8 y_2 + y_1) \equiv (2^{16}z_3 + 2^8 z_2 + z_1) \pmod{2^{24}}.$$
>
> Therefore, once both (z_1, z_2, z_3) and (x_1, x_2, x_3) are known, one can easily compute (y_1, y_2, y_3), from which the initial state s_2 of the 25-bit LFSR is easily obtained.

With this observation the attacker can recover the seed s by trying all possible 16-bit values for s_1. For each guess for s_1 compute the corresponding (x_1, x_2, x_3) output from the 17-bits LFSR. Use

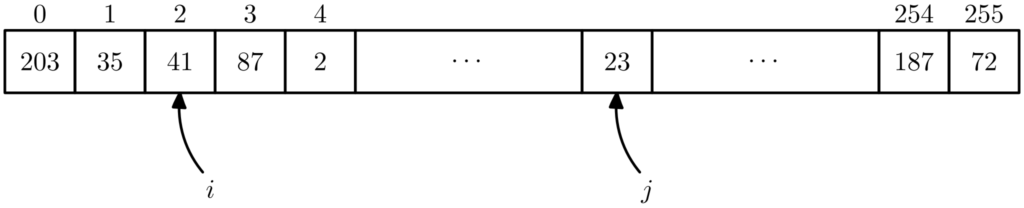


Figure 3.12: An example RC4 internal state

the observation above to obtain a candidate seed s_2 for the 25-bit LFSR. Then to confirm that $\hat{s} := s_1 \| s_2$ is the correct secret seed, run the PRG using the seed $\hat{s}$ for 100 iterations, and compare the resulting output to the given sequence $\bar{z}$. If the sequences do not match, try another guess for s_1. Once the attacker hits the correct value for s_1, the generated sequence will match the given $\bar{z}$, in which case the attacker has the correct secret seed $s := s_1 \| s_2$.

We just showed that the entire seed s can be found after an expected 2^{15} guesses for s_1. This is much faster than the naive 2^{40}-time exhaustive search attack.

3.9 Case study: cryptanalysis of the RC4 stream cipher

The RC4 stream cipher, designed by Ron Rivest in 1987, was historically used for securing Web traffic (in the SSL/TLS protocol) and wireless traffic (in the 802.11b WEP protocol). It is designed to operate on 8-bit processors with little internal memory. While RC4 is still in use, it has been shown to be vulnerable to a number of significant attacks and should not be used in new projects. Our discussion of RC4 serves as an elegant example of stream cipher cryptanalysis.

At the heart of the RC4 cipher is a PRG, called the RC4 PRG. The PRG maintains an internal state consisting of an array S of 256 bytes plus two additional bytes i, j used as pointers into S. The array S contains all the numbers $0 \ldots 255$ and each number appears exactly once. Fig. 3.12 gives an example of an RC4 state.

The RC4 stream cipher key s is a seed for the PRG and is used to initialize the array S to a pseudo-random permutation of the numbers $0 \ldots 255$. Initialization is performed using the following **setup algorithm**:

```
input: string of bytes $s$
for $i \leftarrow 0$ to 255 do:    $S[i] \leftarrow i$
$j \leftarrow 0$
for $i \leftarrow 0$ to 255 do
    $k \leftarrow s\big[i \bmod |s|\big]$    // extract one byte from seed
    $j \leftarrow \big(\, j + S[i] + k \,\big) \bmod 256$
    swap$(S[i], S[j])$
```

During the loop the index i runs linearly through the array while the index j jumps around. At each iteration the entry at index i is swapped with the entry at index j.

Once the array S is initialized, the PRG generates pseudo-random output one byte at a time using the following **stream generator**:

cipher	speed[1](MB/sec)
RC4	126
SEAL	375
Salsa20	408
Sosemanuk	727

Table 3.1: Software stream cipher speeds (higher speed is better)

```
$i \leftarrow 0, \quad j \leftarrow 0$
repeat
    $i \leftarrow (i + 1) \bmod 256$
    $j \leftarrow (j + S[i]) \bmod 256$
    swap$(S[i], S[j])$
    output $S\big[\ (S[i] + S[j]) \bmod 256\ \big]$
forever
```

The procedure runs for as long as necessary. Again, the index i runs linearly through the array while the index j jumps around. Swapping $S[i]$ and $S[j]$ continuously shuffles the array S.

RC4 encryption speed. RC4 is well suited for software implementations. Other stream ciphers, such as Grain and Trivium, are designed for hardware and perform poorly when implemented in software. Table 3.1 provides running times for RC4 and a few other software stream ciphers. Modern processors operate on 64-bit words, making the 8-bit design of RC4 relatively slow on these architectures.

3.9.1 Security of RC4

At one point RC4 was believed to be a secure stream cipher and was widely deployed in applications. The cipher fell from grace after a number of attacks showed that its output is somewhat biased. We present two attacks that distinguish the output of RC4 from a random string. Throughout the section we let n denote the size of the array S. $n = 256$ for RC4.

Bias in the initial RC4 output. The RC4 setup algorithm initializes the array S to a permutation of $0 \dots 255$ generated from the given random seed. For now, let us assume that the RC4 setup algorithm is perfect and generates a uniform permutation from the set of all 256! permutations. Mantin and Shamir [107] showed that, even assuming perfect initialization, the output of RC4 is biased.

Lemma 3.8 (Mantin-Shamir). *Suppose the array S is set to random permutation of $0 \dots n-1$ and that i, j are set to 0. Then the probability that the second byte of the output of RC4 is equal to 0 is $2/n$.*

[1] Performance numbers were obtained using the Crypto++ 5.6.0 benchmarks running on a 1.83 GhZ Intel Core 2 processor.

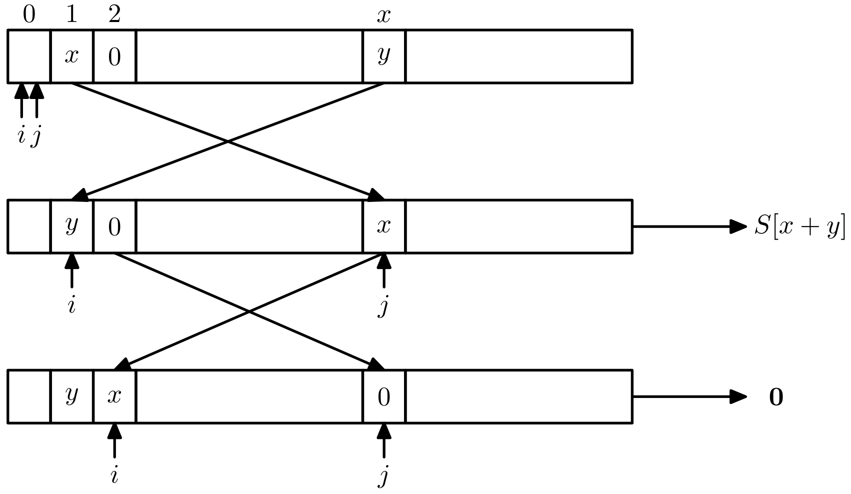


Figure 3.13: Proof of Lemma 3.8

Proof idea. Let z_2 be the second byte output by RC4. Let P be the event that $S[2] = 0$ and $S[1] \neq 2$. The key observation is that when event P happens then $z_2 = 0$ with probability 1. See Fig. 3.13. However, when P does not happen then z_2 is uniformly distributed in $0 \ldots n-1$ and hence equal to 0 with probability $1/n$. Since $\Pr[P]$ is about $1/n$ we obtain (approximately) that

$$\begin{aligned}\Pr[z_2 = 0] &= \Pr\left[(z_2 = 0) \mid P\right] \cdot \Pr[P] + \Pr\left[(z_2 = 0) \mid \neg P\right] \cdot \Pr[\neg P] \\ &\approx 1 \cdot (1/n) + (1/n) \cdot (1 - 1/n) \approx 2/n \quad \Box\end{aligned}$$

The lemma shows that the probability that the second byte in the output of RC4 is 0 is twice what it should be. This leads to a simple distinguisher for the RC4 PRG. Given a string $x \in \{0 \ldots 255\}^\ell$, for $\ell \geq 2$, the distinguisher outputs 0 if the second byte of x is 0 and outputs 1 otherwise. By Lemma 3.8 this distinguisher has advantage approximately $1/n$, which is 0.39% for RC4.

The Mantin-Shamir distinguisher shows that the second byte of the RC4 output are biased. This was generalized by AlFardan et al. [4] who showed, by measuring the bias over many random keys, that there is bias in every one of the first 256 bytes of the output: the distribution on each byte is quite far from uniform. The bias is not as noticeable as in the second byte, but it is non-negligible and sufficient to attack the cipher. They show, for example, that given the encryption of a single plaintext encrypted under 2^{30} random keys, it is possible to recover the first 128 bytes of the plaintext with probability close to 1. This attack is easily carried out on the Web where a secret cookie is often embedded in the first few bytes of a message. This cookie is re-encrypted over and over with fresh keys every time the browser connects to a victim web server. Using Javascript the attacker can make the user's browser repeatedly re-connect to the target site giving the attacker the 2^{30} ciphertexts needed to mount the attack and expose the cookie.

In response, RSA Labs issued a recommendation suggesting that one discard the first 1024 bytes output by the RC4 stream generator and only use bytes 1025 and onwards. This defeats the initial key stream bias distinguishers, but does not defeat other attacks, which we discuss next.

Bias in the RC4 stream generator. Suppose the RC4 setup algorithm is modified so that the attack of the previous paragraph is ineffective. Fluhrer and McGrew [64] gave a direct attack on the stream generator. They argue that the number of times that the pair of bytes $(0, 0)$ appears in the RC4 output is larger than what it should be for a random sequence. This is sufficient to distinguish the output of RC4 from a random string.

Let $\mathrm{ST}_{\mathrm{RC4}}$ be the set of all possible internal states of RC4. Since there are $n!$ possible settings for the array S and n possible settings for each of i and j, the size of $\mathrm{ST}_{\mathrm{RC4}}$ is $n! \cdot n^2$. For $n = 256$, as used in RC4, the size of $\mathrm{ST}_{\mathrm{RC4}}$ is gigantic, namely about 10^{511}.

Lemma 3.9 (Fluhrer-McGrew). *Suppose RC4 is initialized with a random state T in ST_{RC4}. Let (z_1, z_2) be the first two bytes output by RC4 when started in state T. Then*

$$\begin{aligned} i \neq n-1 \quad &\Longrightarrow \quad \Pr[(z_1, z_2) = (0, 0)] \geq (1/n^2) \cdot \big(1 + (1/n)\big) \\ i \neq 0, 1 \quad &\Longrightarrow \quad \Pr[(z_1, z_2) = (0, 1)] \geq (1/n^2) \cdot \big(1 + (1/n)\big) \end{aligned}$$

A pair of consecutive outputs (z_1, z_2) is called a **digraph**. In a truly random string, the probability of all digraphs (x, y) is exactly $1/n^2$. The lemma shows that for RC4 the probability of $(0, 0)$ is greater by $1/n^3$ from what it should be. The same holds for the digraph $(0, 1)$. In fact, Fluhrer-McGrew identify several other anomalous digraphs, beyond those stated in Lemma 3.9.

The lemma suggests a simple distinguisher D between the output of RC4 and a random string. If the distinguisher finds more $(0, 0)$ pairs in the given string than are likely to be in a random string it outputs 1, otherwise it outputs 0. More precisely, the distinguisher D works as follows:

```
input: string $x \in \{0 \dots n\}^\ell$
output: 0 or 1

let $q$ be the number of times the pair $(0, 0)$ appears in $x$
if $(q/\ell) - (1/n^2) > 1/(2n^3)$ output 0, else output 1
```

Using Theorem B.3 we can estimate D's advantage as a function of the input length ℓ. In particular, the distinguisher D achieves the following advantages:

$$\begin{aligned} \ell = 2^{14} \text{ bytes:} \qquad & \mathrm{PRG}\mathsf{adv}[D, RC4] \geq 2^{-8} \\ \ell = 2^{34} \text{ bytes:} \qquad & \mathrm{PRG}\mathsf{adv}[D, RC4] \geq 0.5 \end{aligned}$$

Using all the anomalous digraphs provided by Fluhrer and McGrew one can build a distinguisher that achieves advantage 0.8 using only $2^{30.6}$ bytes of output.

Related key attacks on RC4. Fluhrer, Mantin, and Shamir [63] showed that RC4 is insecure when used with related keys. We discuss this attack and its impact on the 802.11b WiFi protocol in Section 9.10, attack 2.

3.10 Generating random bits in practice

Random bits are needed in cryptography for many tasks, such as generating keys and other ephemeral values called nonces. Throughout the book we assume all parties have access to a good source of randomness, otherwise many desirable cryptographic goals are impossible. So far we used a PRG to stretch a short uniformly distributed secret seed to a long pseudorandom string.

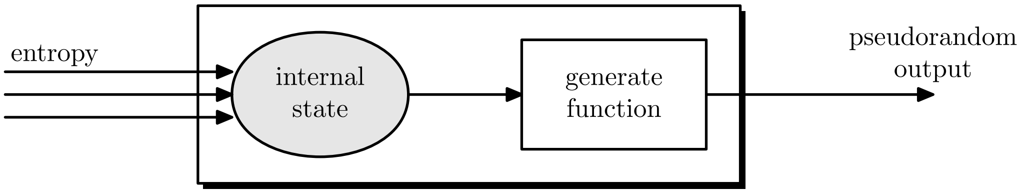


Figure 3.14: A random number generator

While a PRG is an important tool in generating random (or pseudorandom) bits it is only part of the story.

In practice, random bits are generated using a **random number generator**, or RNG. An RNG, like a PRG, outputs a sequence of random or pseudorandom bits. RNGs, however, have an additional interface that is used to continuously add entropy to the RNG's internal state, as shown in Fig. 3.14. The idea is that whenever the system has more random entropy to contribute to the RNG, this entropy is added into the RNG internal state. Whenever someone reads bits from the RNG, these bits are generated using the current internal state.

An example is the Linux RNG which is implemented as a device called `/dev/random`. Anyone can read data from the device to obtain random bits. To play with the `/dev/random` try typing `cat /dev/random` at a UNIX shell. You will see an endless sequence of random-looking characters. The UNIX RNG obtains its entropy from a number of hardware sources:

- keyboard events: inter-keypress timings provide entropy;
- mouse events: both interrupt timing and reported mouse positions are used;
- hardware interrupts: time between hardware interrupts is a good source of entropy;

These sources generate a continuous stream of randomness that is periodically XORed into the RNG internal state. Notice that keyboard input is not used as a source of entropy; only keypress timings are used. This ensures that user input is not leaked to other users in the system via the Linux RNG.

High entropy random generation. The entropy sources described above generate randomness at a relatively slow rate. To generate true random bits at a faster rate, Intel added a hardware random number generator starting with the Ivy Bridge processor family in 2012. Output from the generator is read using the `RdRand` instruction that is intended to provide a fast uniform bit generator.

To reduce biases in the generator output, the raw bits are first passed through a function called a "conditioner" designed to ensure that the output is a sequence of uniformly distributed bits, assuming sufficient entropy is provided as input. We discuss this in more detail in Section 8.10 where we discuss the key derivation problem.

The `RdRand` generator should not replace other entropy sources such as the four sources described above; it should only augment them as an *additional* entropy source for the RNG. This way, if the generator is defective it will not completely compromise the cryptographic application.

One difficulty with Intel's approach is that, over time, the hardware generator may stop producing high entropy random bits due to a hardware glitch. For example, the raw hardware generator may always output ‘0’, resulting in highly non-random output. To prevent this from happening the processor periodically tests the raw bits produced by the hardware using a fixed set of statistical tests. If any of the tests reports “non-random” the hardware generator is declared to be defective.

3.11 A broader perspective: computational and statistical indistinguishability

Our definition of security for a pseudo-random generator G formalized the intuitive idea that an adversary should not be able to effectively distinguish between $G(s)$ and r, where s is a randomly chosen seed, and r is a random element of the output space.

This idea generalizes quite naturally and usefully to other settings. Suppose P_0 and P_1 are probability distributions on some finite set $\mathcal{R}$. Our goal is to formally define the intuitive notion that an adversary cannot effectively distinguish between P_0 and P_1. As usual, this is done via an attack game. For $b = 0, 1$, we write $x \overset{\text{R}}{\leftarrow} P_b$ to denote the assignment to x of a value chosen at random from the set $\mathcal{R}$, according to the probability distribution P_b.

Attack Game 3.3 (Distinguishing P_0 from P_1). For given probability distributions P_0 and P_1 on a finite set $\mathcal{R}$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define:

Experiment b:

- The challenger computes x as follows:

  $x \overset{\text{R}}{\leftarrow} P_b$

  and sends x to the adversary.

- Given x, the adversary computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$’s **advantage** with respect to P_0 and P_1 as

$$\mathsf{Distadv}[\mathcal{A}, P_0, P_1] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 3.4 (Computational indistinguishability). *Distributions P_0 and P_1 are called **computationally indistinguishable** if the value* $\mathsf{Distadv}[\mathcal{A}, P_0, P_1]$ *is negligible for all efficient adversaries $\mathcal{A}$.*

Using this definition we can restate the definition of a secure PRG more simply: a PRG G defined over $(\mathcal{S}, \mathcal{R})$ is secure if and only if P_0 and P_1 are computationally indistinguishable, where P_1 is the uniform distribution on $\mathcal{R}$, and P_0 is the distribution that assigns to each $r \in \mathcal{R}$ the weight

$$P_0(r) := \frac{|\{s \in \mathcal{S} : G(s) = r\}|}{|\mathcal{S}|}.$$

Again, as discussed in Section 2.2.5, Attack Game 3.3 can be recast as a “bit guessing” game, where instead of having two separate experiments, the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$’s *bit-guessing* *advantage* $\mathsf{Distadv}^*[\mathcal{A}, P_0, P_1]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.2.5 (namely, (2.11)) applies here as well:

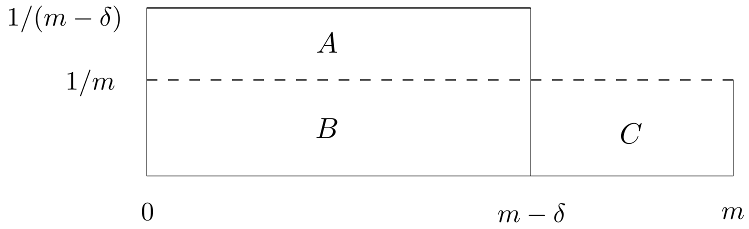
$$\mathsf{Distadv}[\mathcal{A}, P_0, P_1] = 2 \cdot \mathsf{Distadv}^*[\mathcal{A}, P_0, P_1]. \tag{3.14}$$

Typically, to prove that two distributions are computationally indistinguishable, we will have to make certain other computational assumptions. However, sometimes two distributions are so similar that no adversary can effectively distinguish between them, regardless of how much computing power the adversary may have. To make this notion of "similarity" precise, we introduce a useful tool, called **statistical distance**:

Definition 3.5 (Statistical distance). *Suppose P_0 and P_1 are probability distributions on a finite set $\mathcal{R}$. Then their* ***statistical distance*** *is defined as*

$$\Delta[P_0, P_1] := \frac{1}{2} \sum_{r \in \mathcal{R}} \big|P_0(r) - P_1(r)\big|.$$

Example 3.1. Suppose P_0 is the uniform distribution on $\{1, \ldots, m\}$, and P_1 is the uniform distribution on $\{1, \ldots, m - \delta\}$, where $\delta \in \{0, \ldots, m - 1\}$. Let us compute $\Delta[P_0, P_1]$. We could apply the definition directly; however, consider the following graph of P_0 and P_1:



The statistical distance between P_0 and P_1 is just 1/2 times the area of regions A and C in the diagram. Moreover, because probability distributions sum to 1, we must have

$$\text{area of } B + \text{area of } A = 1 = \text{area of } B + \text{area of } C,$$

and hence, the areas of region A and region C are the same. Therefore,

$$\Delta[P_0, P_1] = \text{area of } A = \text{area of } C = \delta/m. \quad \square$$

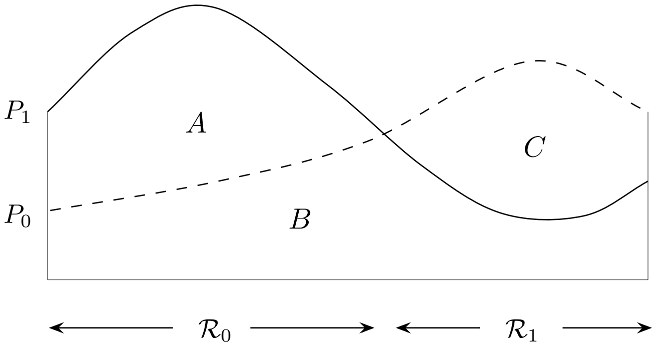
The following theorem allows us to make a connection between the notions of computational indistinguishability and statistical distance:

Theorem 3.10. *Let P_0 and P_1 be probability distributions on a finite set $\mathcal{R}$. Then we have*

$$\max_{\mathcal{R}' \subseteq \mathcal{R}} \big|P_0[\mathcal{R}'] - P_1[\mathcal{R}']\big| = \Delta[P_0, P_1],$$

where the maximum is taken over all subsets $\mathcal{R}'$ of $\mathcal{R}$.

Proof. Suppose we split the set $\mathcal{R}$ into two disjoint subsets: the set $\mathcal{R}_0$ consisting of those $r \in \mathcal{R}$ such that $P_0(r) < P_1(r)$, and the set $\mathcal{R}_1$ consisting of those $r \in \mathcal{R}$ such that $P_0(r) \geq P_1(r)$. Consider the following rough graph of the distributions of P_0 and P_1, where the elements of $\mathcal{R}_0$ are placed to the left of the elements of $\mathcal{R}_1$:



Now, as in Example 3.1,

$$\Delta[P_0, P_1] = \text{area of } A = \text{area of } C.$$

Observe that for every subset $\mathcal{R}'$ of $\mathcal{R}$, we have

$$P_0[\mathcal{R}'] - P_1[\mathcal{R}'] = \text{area of } C' - \text{area of } A',$$

where C' is the subregion of C that lies above $\mathcal{R}'$, and A' is the subregion of A that lies above $\mathcal{R}'$. It follows that $|P_0[\mathcal{R}'] - P_1[\mathcal{R}']|$ is maximized when $\mathcal{R}' = \mathcal{R}_0$ or $\mathcal{R}' = \mathcal{R}_1$, in which case it is equal to $\Delta[P_0, P_1]$. □

The connection to computational indistinguishability is as follows:

Theorem 3.11. *Let P_0 and P_1 be probability distributions on a finite set $\mathcal{R}$. Then for every adversary $\mathcal{A}$, we have*

$$\mathsf{Distadv}[\mathcal{A}, P_0, P_1] \leq \Delta[P_0, P_1].$$

Proof. Consider an adversary $\mathcal{A}$ that tries to distinguish P_0 from P_1, as in Attack Game 3.3.

First, we consider the case where $\mathcal{A}$ is deterministic. In this case, the output of $\mathcal{A}$ is a function $f(r)$ of the value $r \in \mathcal{R}$ presented to it by the challenger. Let $\mathcal{R}' := \{r \in \mathcal{R} : f(r) = 1\}$. If W_0 and W_1 are the events defined in Attack Game 3.3, then for $b = 0, 1$, we have

$$\Pr[W_b] = P_b[\mathcal{R}'].$$

By the previous theorem, we have

$$\mathsf{Distadv}[\mathcal{A}, P_0, P_1] = \big|P_0[\mathcal{R}'] - P_1[\mathcal{R}']\big| \leq \Delta[P_0, P_1].$$

We now consider the case where $\mathcal{A}$ is probabilistic. We can view $\mathcal{A}$ as taking an auxiliary input t, representing its random choices. We view t as being chosen uniformly at random from some finite set $\mathcal{T}$. Thus, the output of $\mathcal{A}$ is a function $g(r, t)$ of the value $r \in \mathcal{R}$ presented to it by the challenger, and the value $t \in \mathcal{T}$ representing its random choices. For a given $t \in \mathcal{T}$, let $\mathcal{R}'_t := \{r \in \mathcal{R} : g(r, t) = 1\}$. Then, averaging over the random choice of t, we have

$$\Pr[W_b] = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} P_b[\mathcal{R}'_t].$$

It follows that

$$\begin{aligned}\mathsf{Distadv}[\mathcal{A}, P_0, P_1] &= |\Pr[W_0] - \Pr[W_1]| \\ &= \frac{1}{|\mathcal{T}|}\Big|\sum_{t\in\mathcal{T}}(P_0[\mathcal{R}'_t] - P_1[\mathcal{R}'_t])\Big| \\ &\leq \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}|P_0[\mathcal{R}'_t] - P_1[\mathcal{R}'_t]| \\ &\leq \frac{1}{|\mathcal{T}|}\sum_{t\in\mathcal{T}}\Delta[P_0, P_1] \\ &= \Delta[P_0, P_1]. \quad \square\end{aligned}$$

Analogous to the definition of computational indistinguishability, we have:

Definition 3.6 (Statistical indistinguishability). *Let P_0 and P_1 be probability distributions on a finite set $\mathcal{R}$. We say that P_0 and P_1 are **statistically indistinguishable** if the statistical distance $\Delta[P_0, P_1]$ is negligible.*

An immediate consequence of Theorem 3.11 is that two distributions that are statistically indistinguishable are also computationally indistinguishable:

Corollary 3.12. *Let P_0 and P_1 be probability distributions on a finite set $\mathcal{R}$. If P_0 and P_1 are statistically indistinguishable, then they are also computationally indistinguishable.*

Statistical distance between random variables. One also defines the statistical distance between two random variables as the statistical distance between their corresponding distributions. That is, if X and Y are random variables taking values in a finite set $\mathcal{R}$, then their **statistical distance** is

$$\Delta[\mathsf{X}, \mathsf{Y}] := \frac{1}{2}\sum_{r\in\mathcal{R}}|\Pr[\mathsf{X} = r] - \Pr[\mathsf{Y} = r]|.$$

In this case, Theorem 3.10 says that

$$\max_{\mathcal{R}'\subseteq\mathcal{R}}\Big|\Pr[\mathsf{X} \in \mathcal{R}'] - \Pr[\mathsf{Y} \in \mathcal{R}']\Big| = \Delta[\mathsf{X}, \mathsf{Y}],$$

where the maximum is taken over all subsets $\mathcal{R}'$ of $\mathcal{R}$. Definition 3.6, which defines statistical inistinguishability, extends to random variables as well.

Analogously, one can define distinguishing advantage (as in Attack Game 3.3) and computational indistinguishability (as in Definition 3.4) with respect to random variables, rather than distributions. The advantage of working with random variables is that we can more conveniently work with distributions that are related to one another, as exemplified in the following theorem.

Theorem 3.13. *If $\mathcal{S}$ and $\mathcal{T}$ are finite sets, X and Y are random variables taking values in $\mathcal{S}$, and $f : \mathcal{S} \to \mathcal{T}$ is a function, then $\Delta[f(\mathsf{X}), f(\mathsf{Y})] \leq \Delta[\mathsf{X}, \mathsf{Y}]$.*

Proof. We have

$$\begin{aligned}\Delta[f(\mathsf{X}), f(\mathsf{Y})] &= |\Pr[f(\mathsf{X}) \in \mathcal{T}'] - \Pr[f(\mathsf{Y}) \in \mathcal{T}']| \text{ for some } \mathcal{T}' \subseteq \mathcal{T} \\ &\qquad \text{(by Theorem 3.10)} \\ &= |\Pr[\mathsf{X} \in f^{-1}(\mathcal{T}')] - \Pr[\mathsf{Y} \in f^{-1}(\mathcal{T}')]| \\ &\leq \Delta[\mathsf{X}, \mathsf{Y}] \text{ (again by Theorem 3.10).} \quad \Box\end{aligned}$$

Example 3.2. Let X be uniformly distributed over the set $\{0, \ldots, m-1\}$, and let Y be uniformly distributed over the set $\{0, \ldots, N-1\}$, for $N \geq m$. Let $f(t) := t \bmod m$. We want to compute an upper bound on the statistical distance between X and $f(\mathsf{Y})$. We can do this as follows. Let $N = qm - r$, where $0 \leq r < m$, so that $q = \lceil N/m \rceil$. Also, let Z be uniformly distributed over $\{0, \ldots, qm-1\}$. Then $f(\mathsf{Z})$ is uniformly distributed over $\{0, \ldots, m-1\}$, since every element of $\{0, \ldots, m-1\}$ has the same number (namely, q) of pre-images under f which lie in the set $\{0, \ldots, qm-1\}$. Since statistical distance depends only on the distributions of the random variables, by the previous theorem, we have

$$\Delta[\mathsf{X}, f(\mathsf{Y})] = \Delta[f(\mathsf{Z}), f(\mathsf{Y})] \leq \Delta[\mathsf{Z}, \mathsf{Y}],$$

and as we saw in Example 3.1,

$$\Delta[\mathsf{Z}, \mathsf{Y}] = \frac{r}{qm} < \frac{1}{q} \leq \frac{m}{N}.$$

Therefore,

$$\Delta[\mathsf{X}, f(\mathsf{Y})] < \frac{m}{N}. \quad \Box$$

Example 3.3. Suppose we want to generate a pseudo-random number in a given interval $\{0, \ldots, m-1\}$. However, suppose that we have at our disposal a PRG G that outputs L-bit strings. Of course, an L-bit string can be naturally viewed as a number in the range $\{0, \ldots, N-1\}$, where $N := 2^L$. Let us assume that $N \geq m$.

To generate a pseudo-random number in the interval $\{0, \ldots, m-1\}$, we can take the output of G, view it as a number in the interval $\{0, \ldots, N-1\}$, and reduce it modulo m. We will show that this produces a number that is computationally indistinguishable from a truly random number in the interval $\{0, \ldots, m-1\}$, assuming G is secure and m/N is negligible (e.g., $N \geq 2^{100} \cdot m$).

To this end, let P_0 be the distribution representing the output of G, reduced modulo m, and let P_1 be the uniform distribution on $\{0, \ldots, m-1\}$. Let $\mathcal{A}$ be an adversary trying to distinguish P_0 from P_1, as in Attack Game 3.3.

Let Game 0 be Experiment 0 of Attack Game 3.3, in which $\mathcal{A}$ is presented with a random sample distributed according to P_0, and let W_0 be the event that $\mathcal{A}$ outputs 1 in this game.

Now define Game 1 to be the same as Game 0, except that we replace the output of G by a truly random value chosen from the interval $\{0, \ldots, N-1\}$. This value is then reduced modulo m, as in Game 0. Let W_1 be the event that $\mathcal{A}$ outputs 1 in Game 1. One can easily construct an efficient adversary $\mathcal{B}$ that attacks G as in Attack Game 3.1, such that

$$\mathsf{PRGadv}[\mathcal{B}, G] = \big|\Pr[W_0] - \Pr[W_1]\big|.$$

The idea is that $\mathcal{B}$ takes its challenge value, reduces it modulo m, gives this value to $\mathcal{A}$, and outputs whatever $\mathcal{A}$ outputs.

Finally, we define Game 2 to be Experiment 1 of Attack Game 3.3, in which $\mathcal{A}$ is presented with a random sample distributed according to P_1, the uniform distribution on $\{0, \ldots, m-1\}$. Let W_2 be the event that $\mathcal{A}$ outputs 1 in Game 2. If P is the distribution of the value presented to $\mathcal{A}$ in Game 1, then by Theorem 3.11, we have $|\Pr[W_1] - \Pr[W_2]| \leq \Delta[P, P_1]$; moreover, by Example 3.2, we have $\Delta[P, P_1] \leq m/N$.

Putting everything together, we see that

$$\begin{aligned}\mathsf{Distadv}[\mathcal{A}, P_0, P_1] = \big|\Pr[W_0] - \Pr[W_2]\big| &\leq \big|\Pr[W_0] - \Pr[W_1]\big| + \big|\Pr[W_1] - \Pr[W_2]\big| \\ &\leq \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G] + \frac{m}{N},\end{aligned}$$

which, by assumption, is negligible. □

Remark 3.1. In statistics, a **divergence function** $D(P_0, P_1)$ establishes a way to measure the distance between two distributions P_0 and P_1. The statistical distance $\Delta[P_0, P_1]$, developed in this section, is one example of a divergence function. Many other divergence functions have been developed, such as Kullback–Leibler (KL) divergence and Rényi Divergence. In some cases, these alternate divergence functions lead to much tighter bounds than what can be proved using statistical distance. Exercise 3.12 introduces a divergence function with important applications to cryptography (see Exercises 3.14 and 7.12). Other divergence functions and applications may be found in [133, 1]. □

3.11.1 Mathematical details

As usual, we fill in the mathematical details needed to interpret the definitions and results of this section from the point of view of asymptotic complexity theory.

In defining computational and statistical indistinguishability (Definitions 3.4 and 3.6), one should consider two families of probability distributions $P_0 = \{P_{0,\lambda}\}_\lambda$ and $P_1 = \{P_{1,\lambda}\}_\lambda$, indexed by a security parameter λ. For each λ, the distributions $P_{0,\lambda}$ and $P_{1,\lambda}$ should take values in a finite set of bit strings $\mathcal{R}_\lambda$, where the strings in $\mathcal{R}_\lambda$ are bounded in length by a polynomial in λ. In Attack Game 3.3, the security parameter λ is an input to both the challenger and adversary, and in Experiment b, the challenger produces a sample, distributed according to $P_{b,\lambda}$. The advantage should properly be written $\mathsf{Distadv}[\mathcal{A}, P_0, P_1](\lambda)$, which is a function of λ. Computational indistinguishability means that this is a negligible function. Similarly, the definition of statistical indistinguishability says that the value $\Delta[P_{0,\lambda}, P_{1,\lambda}]$ grows negligibly, as a function of λ.

In some situations, it may be natural to introduce a probabilistically generated system parameter; however, from a technical perspective, this is not necessary, as such a system parameter can be incorporated in the distributions $P_{0,\lambda}$ and $P_{1,\lambda}$. One could also impose the requirement that $P_{0,\lambda}$ and $P_{1,\lambda}$ be efficiently sampleable; however, to keep the definition simple, we will not require this.

The definition of statistical distance (Definition 3.5) makes perfect sense from a non-asymptotic point of view, and does not require any modification or elaboration. Theorem 3.10 holds as stated, for specific distributions P_0 and P_1. Theorem 3.11 may be viewed asymptotically as stating that for all distribution families $P_0 = \{P_{0,\lambda}\}_\lambda$ and $P_1 = \{P_{1,\lambda}\}_\lambda$, for all adversaries (even computationally unbounded ones), and for all λ, we have

$$\mathsf{Distadv}[\mathcal{A}, P_0, P_1](\lambda) \leq \Delta[P_{0,\lambda}, P_{1,\lambda}].$$

3.12 A fun application: coin flipping and bit commitment

Alice and Bob are going out on a date. Alice wants to see one movie and Bob wants to see another. They decide to flip a random coin to choose the movie. If the coin comes up "heads" they will go to Alice's choice; otherwise, they will go to Bob's choice. When Alice and Bob are in close proximity this is easy: one of them, say Bob, flips a coin and they both verify the result. When they are far apart and are speaking on the phone this is harder. Bob can flip a coin on his side and tell Alice the result, but Alice has no reason to believe the outcome. Bob could simply claim that the coin came up "tails" and Alice would have no way to verify this. Not a good way to start a date.

A simple solution to their problem makes use of a cryptographic primitive called **bit commitment**. It lets Bob commit to a bit $b \in \{0, 1\}$ of his choice. Later, Bob can open the commitment and convince Alice that b was the value he committed to. Committing to a bit b results in a **commitment string** c, that Bob sends to Alice, and an **opening string** s that Bob uses for opening the commitment later. A commitment scheme is secure if it satisfies the following two properties:

- **Hiding:** The commitment string c reveals no information about the committed bit b. More precisely, the distribution on c when committing to the bit 0 is indistinguishable from the distribution on c when committing to the bit 1. In the bit commitment scheme we present, the hiding property depends on the security of a certain PRG G.

- **Binding:** Let c be a commitment string output by Bob. If Bob can open the commitment as some $b \in \{0, 1\}$ then he cannot open it as $\bar{b}$. This ensures that once Bob commits to a bit b he can open it as b and nothing else. In the commitment scheme we present the binding property holds unconditionally.

Coin flipping. Using a commitment scheme, Alice and Bob can generate a random bit $b \in \{0, 1\}$ so that no side can bias the result towards their preferred outcome, assuming the protocol terminates successfully. Such protocols are called **coin flipping protocols**. The resulting bit b determines what movie they go to.

Alice and Bob use the following simple coin flipping protocol:

Step 1: Bob chooses a random bit $b_0 \xleftarrow{R} \{0, 1\}$.
Alice and Bob execute the commitment protocol by which Alice obtains
a commitment c to b_0 and Bob obtains an opening string s.
Step 2: Alice chooses a random bit $b_1 \xleftarrow{R} \{0, 1\}$ and sends b_1 to Bob in the clear.
Step 3: Bob opens the commitment by revealing b_0 and s to Alice.
Alice verifies that c is indeed a commitment to b_0 and aborts if verification fails.

Output: the resulting bit is $b := b_0 \oplus b_1$.

We argue that if the protocol terminates successfully and one side is honestly following the protocol then the other side cannot bias the result towards their preferred outcome. By the hiding property, Alice learns nothing about b_0 at the end of Step 1 and therefore her choice of bit b_1 is independent of the value of b_0. By the binding property, Bob can only open the commitment c in Step 3 to the bit b_0 he chose in Step 1. Because he chose b_0 before Alice chose b_1, Bob's choice of b_0 is independent of b_1. We conclude that the output bit b is the XOR of two independent bits. Therefore, if one side is honestly following the protocol, the other side cannot bias the resulting bit.

One issue with this protocol is that Bob learns the generated bit at the end of Step 2, before Alice learns the bit. In principle, if the outcome is not what Bob wants he could abort the protocol at the end of Step 2 and try to re-initiate the protocol hoping that the next run will go his way. More sophisticated coin flipping protocols avoid this problem, but at the cost of many more rounds of interaction (see, e.g., [116]).

Bit commitment from secure PRGs. It remains to construct a secure bit commitment scheme that lets Bob commit to his bit $b_0 \in \{0,1\}$. We do so using an elegant construction due to Naor [122].

Let $G : \mathcal{S} \to \mathcal{R}$ be a secure PRG where $|\mathcal{R}| \geq |\mathcal{S}|^3$ and $\mathcal{R} = \{0,1\}^n$ for some n. To commit to the bit b_0, Alice and Bob engage in the following protocol:

Bob commits to bit $b_0 \in \{0,1\}$:

Step 1: Alice chooses a random $r \in \mathcal{R}$ and sends r to Bob.

Step 2: Bob chooses a random $s \in \mathcal{S}$ and computes $c \leftarrow \mathrm{com}(s, r, b_0)$ where $\mathrm{com}(s, r, b_0)$ is the following function:

$$c = \mathrm{com}(s, r, b_0) := \begin{cases} G(s) & \text{if } b_0 = 0, \\ G(s) \oplus r & \text{if } b_0 = 1. \end{cases}$$

Bob outputs c as the commitment string and uses s as the opening string.

When it comes time to open the commitment Bob sends (b_0, s) to Alice. Alice accepts the opening if $c = \mathrm{com}(s, r, b_0)$ and rejects otherwise.

The hiding property follows directly from the security of the PRG: because the output $G(s)$ is computationally indistinguishable from a uniform random string in $\mathcal{R}$ it follows that $G(s) \oplus r$ is also computationally indistinguishable from a uniform random string in $\mathcal{R}$. Therefore, whether $b_0 = 0$ or $b_0 = 1$, the commitment string c is computationally indistinguishable from a uniform string in $\mathcal{R}$, as required.

The binding property holds unconditionally as long as $1/|\mathcal{S}|$ is negligible. The only way Bob can open a commitment $c \in \mathcal{R}$ as both 0 and 1 is if there exist two seeds $s_0, s_1 \in \mathcal{S}$ such that $c = G(s_0) = G(s_1) \oplus r$ which implies that $G(s_0) \oplus G(s_1) = r$. Let us say that $r \in \mathcal{R}$ is "bad" if there are seeds $s_0, s_1 \in \mathcal{S}$ such that $G(s_0) \oplus G(s_1) = r$. The number of pairs of seeds (s_0, s_1) is $|\mathcal{S}|^2$, and therefore the number of bad r is at most $|\mathcal{S}|^2$. It follows that the probability that Alice chooses a bad r is at most $|\mathcal{S}|^2/|\mathcal{R}| < |\mathcal{S}|^2/|\mathcal{S}|^3 = 1/|\mathcal{S}|$ which is negligible. Therefore, the probability that Bob can open the commitment c as both 0 and 1 is negligible.

This completes the description of the bit commitment scheme. We will see a more efficient commitment scheme and more applications for commitments in Section 8.12, after we develop a few more tools.

3.13 Notes

Citations to the literature to be added.

3.14 Exercises

3.1 (Semantic security for random messages). One can define a notion of semantic security for random messages. Here, one modifies Attack Game 2.1 so that instead of the adversary choosing the messages m_0, m_1, the challenger generates m_0, m_1 at random from the message space. Otherwise, the definition of advantage and security remains unchanged.

(a) Suppose that $\mathcal{E} = (E, D)$ is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{M} = \{0,1\}^L$. Assuming that $\mathcal{E}$ is semantically secure for random messages, show how to construct a new cipher $\mathcal{E}'$ that is secure in the ordinary sense. Your new cipher should be defined over $(\mathcal{K}', \mathcal{M}', \mathcal{C}')$, where $\mathcal{K}' = \mathcal{K}$ and $\mathcal{M}' = \mathcal{M}$.

(b) Give an example of a cipher that is semantically secure for random messages but that is not semantically secure in the ordinary sense.

3.2 (Encryption chain). Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where $\mathcal{K} = \mathcal{M}$. Let $\mathcal{E}' = (E', D')$ be a cipher where encryption is defined as $E'((k_1, k_2), m) := \big(E(k_1, k_2), E(k_2, m)\big)$. Show that if $\mathcal{E}$ is semantically secure then so is $\mathcal{E}'$.
Hint: Use a three move hybrid argument. In particular, define four games where in each game the adversary outputs $m_0, m_1 \in \mathcal{M}$, and receives back an $\mathcal{E}'$ ciphertext c, as:

$$\begin{array}{ll} \text{Game 1: } c \xleftarrow{\text{R}} \big(E(k_1,k_2), E(k_2,m_0)\big), & \text{Game 3: } c \xleftarrow{\text{R}} \big(E(k_1,0), E(k_2,m_1)\big), \\ \text{Game 2: } c \xleftarrow{\text{R}} \big(E(k_1,0), E(k_2,m_0)\big), & \text{Game 4: } c \xleftarrow{\text{R}} \big(E(k_1,k_2), E(k_2,m_1)\big). \end{array}$$

Argue that the adversary cannot distinguish each game from the one before it. Deduce that $\mathcal{E}'$ is semantically secure.

Discussion: This encryption scheme can be used to distribute large protected files. For example, a movie rental service can place a large encrypted movie, $E(k_2, m)$, on a content distribution network for anyone to download. When a customer, Bob, wants to watch the movie m, he pays the rental service, and the service sends back a short ticket $E(k_1, k_2)$, where Bob knows k_1. Bob can now stream the encrypted file from the content distribution network, and decrypt it locally. Exercise 5.4 gives another application of this construction.

3.3 (Indistinguishability from a random message). This exercise develops an alternative characterization of semantic security. Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Assume that one can efficiently generate messages from the message space $\mathcal{M}$ at random. We define an attack game between an adversary $\mathcal{A}$ and a challenger as follows. The adversary selects a message $m \in \mathcal{M}$ and sends m to the challenger. The challenger then computes:

$$b \xleftarrow{\text{R}} \{0,1\}, \ k \xleftarrow{\text{R}} \mathcal{K}, \ \ m_0 \leftarrow m, \ m_1 \xleftarrow{\text{R}} \mathcal{M}, \ c \xleftarrow{\text{R}} E(k, m_b),$$

and sends the ciphertext c to $\mathcal{A}$, who then computes and outputs a bit $\hat{b}$. That is, the challenger encrypts either m or a random message, depending on b. We define $\mathcal{A}$'s advantage to be $|\Pr[\hat{b} = b] - 1/2|$, and we say the $\mathcal{E}$ is *real/random semantically secure* if this advantage is negligible for all efficient adversaries.

Show that $\mathcal{E}$ is real/random semantically secure if and only if it is semantically secure in the ordinary sense.

3.4 (Pseudo-random ciphertexts). In this exercise, we develop a notion of security for a cipher, called *psuedo-random ciphertext security*, which intuitively says that no efficient adversary can distinguish an encryption of a chosen message from a random ciphertext.

Let $\mathcal{E} = (E, D)$ be defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Assume that one can efficiently generate ciphertexts from the ciphertext space $\mathcal{C}$ at random. We define an attack game between an adversary $\mathcal{A}$ and a challenger as follows. The adversary selects a message $m \in \mathcal{M}$ and sends m to the challenger. The challenger then computes:

$$b \xleftarrow{R} \{0,1\}, \; k \xleftarrow{R} \mathcal{K}, \;\; c_0 \xleftarrow{R} E(k, m), \;\; c_1 \xleftarrow{R} \mathcal{C}, \quad c \leftarrow c_b$$

and sends the ciphertext c to $\mathcal{A}$, who then computes and outputs a bit $\hat{b}$. We define $\mathcal{A}$'s advantage to be $|\Pr[\hat{b} = b] - 1/2|$, and we say the $\mathcal{E}$ is *pseudo-random ciphertext secure* if this advantage is negligible for all efficient adversaries.

(a) Show that if a cipher is pseudo-random ciphertext secure, then it is semantically secure.

(b) Show that the one-time pad is pseudo-random ciphertext secure.

(c) Give an example of a cipher that is semantically secure, but not pseudo-random ciphertext secure.

3.5 (Small seed spaces are insecure). Suppose G is a PRG defined over $(\mathcal{S}, \mathcal{R})$ where $|\mathcal{R}| \geq 2|\mathcal{S}|$. Let us show that $|\mathcal{S}|$ must be super-poly. To do so, show that there is an adversary that achieves advantage at least 1/2 in attacking the PRG G whose running time is linear in $|\mathcal{S}|$.

3.6 (Another malleability example). Let us give another example illustrating the malleability of stream ciphers. Suppose you are told that the stream cipher encryption of the message "attack at dawn" is 6c73d5240a948c86981bc294814d (the plaintext letters are encoded as 8-bit ASCII and the given ciphertext is written in hex). What would be the stream cipher encryption of the message "attack at dusk" under the same key?

3.7 (Exercising the definition of a secure PRG). Suppose $G(s)$ is a secure PRG that outputs bit-strings in $\{0,1\}^n$. Which of the following derived generators are secure?

(a) $G_1(s_1 \parallel s_2) := G(s_1) \wedge G(s_2)$ where $\wedge$ denotes bit-wise AND.

(b) $G_2(s_1 \parallel s_2) := G(s_1) \oplus G(s_2)$.

(c) $G_3(s) := G(s) \oplus 1^n$.

(d) $G_4(s) := G(s)[0 \mathinner{..} n-1]$.

(e) $G_5(s) := (G(s), G(s))$.

(f) $G_6(s_1 \parallel s_2) := (s_1, G(s_2))$.

3.8 (The converse of Theorem 3.1). In Section 3.2, we showed how to build a stream cipher from a PRG. In Theorem 3.1, we proved that this encryption scheme is semantically secure if the PRG is secure. Prove the converse: the PRG is secure if this encryption scheme is semantically secure.

3.9 (Predicting the next character). In Section 3.5, we showed that if one could effectively distinguish a random bit string from a pseudo-random bit string, then one could succeed in predicting the next bit of a pseudo-random bit string with probability significantly greater than 1/2 (where the position of the "next bit" was chosen at random). Generalize this from bit strings to strings over the alphabet $\{0, \ldots, n-1\}$, for all $n \geq 2$, assuming that n is poly-bounded.

Hint: First generalize the distinguisher/predictor lemma (Lemma 3.5).

3.10 (Simple statistical distance calculations).

(a) Let X and Y be independent random variables, each uniformly distributed over $\mathbb{Z}_p$, where p is prime. Calculate $\Delta[\ (\mathsf{X}, \mathsf{Y}),\ (\mathsf{X}, \mathsf{XY})\ ]$.

(b) Let X and Y be random variables, each taking values in the interval $[0, t]$. Show that $|E[\mathsf{X}] - E[\mathsf{Y}]| \leq t\Delta[\mathsf{X}, \mathsf{Y}]$.

3.11 (A converse to Example 3.2). In Example 3.2, we saw that if Y be uniformly distributed over the set $\{0, \ldots, N-1\}$, then the statistical between $\mathsf{R} := (\mathsf{Y} \bmod m)$ and the uniform distribution on $\{0, \ldots, m-1\}$ is less than m/N. This exercise develops a converse of sorts.

Let R be uniformly distributed over $\{0, \ldots, m-1\}$ and Q be uniformly distributed over $\{0, \ldots, \lfloor N/m \rfloor - 1\}$, where R and Q are independent, and set $\mathsf{Y} := m\mathsf{Q} + \mathsf{R}$. Show that (i) $0 \leq \mathsf{Y} < N$, (ii) $R = (\mathsf{Y} \bmod m)$, and (iii) the statistical distance between Y and the uniform distribution on $\{0, \ldots, N-1\}$ is less than m/N.

The following three exercises should be done together; they will be used in exercises in the following chapters.

3.12 (Distribution ratio). This exercise develops another way of comparing two probability distributions, which considers ratios of probabilities, rather than differences. Let X and Y be two random variables taking values on a finite set $\mathcal{R}$, and assume that $\Pr[\mathsf{X} = r] > 0$ for all $r \in \mathcal{R}$. Define

$$\rho[\mathsf{X}, \mathsf{Y}] := \max\bigl\{\Pr[\mathsf{Y} = r] / \Pr[\mathsf{X} = r] : r \in \mathcal{R}\bigr\}$$

We call this the **max ratio distance** between X and Y. Show that for every subset $\mathcal{R}'$ of $\mathcal{R}$, we have $\Pr[\mathsf{Y} \in \mathcal{R}'] \leq \rho[\mathsf{X}, \mathsf{Y}] \cdot \Pr[\mathsf{X} \in \mathcal{R}']$.

3.13 (A variant of Bernoulli's inequality). The following is a useful fact that will be used in the following exercise. Prove the following statement by induction on n: for any real numbers $x_1, \ldots, x_n$ in the interval $[0, 1]$, we have

$$\prod_{i=1}^{n}(1 - x_i) \geq 1 - \sum_{i=1}^{n} x_i.$$

3.14 (Sampling with and without replacement: distance and ratio). Let $\mathcal{X}$ be a finite set of size N, and let $Q \leq N$. Define random variables X and Y, where X is uniformly distributed over all sequences of Q elements in $\mathcal{X}$, and Y is uniformly distributed over all sequences of Q *distinct* elements in $\mathcal{X}$. Let $\Delta[\mathsf{X}, \mathsf{Y}]$ be the statistical distance between X and Y, and let $\rho[\mathsf{X}, \mathsf{Y}]$ be the max ratio distance defined in Exercise 3.12. Using the previous exercise, prove the following:

(a) $\Delta[\mathsf{X}, \mathsf{Y}] = 1 - \displaystyle\prod_{i=0}^{Q-1}(1 - i/N) \leq \frac{Q^2}{2N}$,

(b) $\rho[\mathsf{X}, \mathsf{Y}] = \dfrac{1}{\prod_{i=0}^{Q-1}(1 - i/N)} \leq \dfrac{1}{1 - \frac{Q^2}{2N}}$ (assuming $Q^2 < 2N$).

Discussion: The result of part (b) has applications to the security analysis of message authentication codes. See Exercise 7.12.

3.15 (Theorem 3.2 is tight). Let us show that the bounds in the parallel composition theorem, Theorem 3.2, are tight. Consider the following, rather silly PRG G_0, which "stretches" ℓ-bit strings to ℓ-bit strings, with ℓ even: for $s \in \{0,1\}^\ell$, we define

```
$G_0(s) :=$
    if $s[0 \mathinner{..} \ell/2 - 1] = 0^{\ell/2}$
        then output $0^\ell$
        else  output $s$.
```

That is, if the first $\ell/2$ bits of s are zero, then $G_0(s)$ outputs the all-zero string, and otherwise, $G_0(s)$ outputs s.

Next, define the following PRG adversary $\mathcal{B}_0$ that attacks G_0:

> When the challenger presents $\mathcal{B}_0$ with $r \in \{0,1\}^\ell$, if r is of the form $0^{\ell/2} \parallel t$, for some $t \neq 0^{\ell/2}$, $\mathcal{B}_0$ outputs 1; otherwise, $\mathcal{B}_0$ outputs 0.

Now, let G_0' be the n-wise parallel composition of G_0. Using $\mathcal{B}_0$, we construct a PRG adversary $\mathcal{A}_0$ that attacks G_0':

> when the challenger presents $\mathcal{A}_0$ with the sequence of strings $(r_1, \ldots, r_n)$, $\mathcal{A}_0$ presents each r_i to $\mathcal{B}_0$, and outputs 1 if $\mathcal{B}_0$ ever outputs 1; otherwise, $\mathcal{A}_0$ outputs 0.

(a) Show that $\text{PRG}\mathsf{adv}[\mathcal{B}_0, G_0] = 2^{-\ell/2} - 2^{-\ell}$.

(b) Show that $\text{PRG}\mathsf{adv}[\mathcal{A}_0, G_0'] \geq n2^{-\ell/2} - n(n+1)2^{-\ell}$.

(c) Show that no adversary attacking G_0 has a better advantage than $\mathcal{B}_0$ (hint: make an argument based on statistical distance).

(d) Using parts (a)–(c), argue that Theorem 3.2 cannot be substantially improved; in particular, show that the following *cannot* be true:

> *There exists a constant $c < 1$ such that for every PRG G, poly-bounded n, and efficient adversary $\mathcal{A}$, there exists an efficient adversary $\mathcal{B}$ such that*
>
> $$\text{PRG}\mathsf{adv}[\mathcal{A}, G'] \leq cn \cdot \text{PRG}\mathsf{adv}[\mathcal{B}, G],$$
>
> *where G' is the n-wise parallel composition of G.*

3.16 (A converse (of sorts) to Theorem 2.8). Let $\mathcal{E} = (E, D)$ be a semantically secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{M} = \{0,1\}$. Let $\mathcal{A}$ be an efficient adversary that takes as input an encryption $c \xleftarrow{\text{R}} E(k, b)$ of a random bit $b \xleftarrow{\text{R}} \{0,1\}$, where $k \xleftarrow{\text{R}} \mathcal{K}$, and outputs a guess $b' \in \{0,1\}$ for b. Show that the probability that $b = b'$ is at most $1/2 + \epsilon$, where ϵ is negligible.

Hint: Use Lemma 3.5.

3.17 (Previous-bit prediction). Suppose that $\mathcal{A}$ is an effective next-bit predictor. That is, suppose that $\mathcal{A}$ is an efficient adversary whose advantage in Attack Game 3.2 is non-negligible. Show how to use $\mathcal{A}$ to build an explicit, effective *previous-bit* predictor $\mathcal{B}$ that uses $\mathcal{A}$ as a black box. Here, one defines a *previous-bit prediction game* that is the same as Attack Game 3.2, except that the challenger sends $r[i+1\,..\,L-1]$ to the adversary. Also, express $\mathcal{B}$'s previous-bit prediction advantage in terms of $\mathcal{A}$'s next-bit prediction advantage.

3.18 (An insecure PRG based on linear algebra). Let A be a fixed $m \times n$ matrix with $m > n$ whose entries are all binary. Consider the following PRG $G : \{0,1\}^n \to \{0,1\}^m$ defined by

$$G(s) := A \cdot s \pmod 2$$

where $A \cdot s \bmod 2$ denotes a matrix-vector product where all elements of the resulting vector are reduced modulo 2. Show that this PRG is insecure no matter what matrix A is used.

3.19 (Generating an encryption key using a PRG). Let $G : \mathcal{S} \to \mathcal{R}$ be a secure PRG. Let $\mathcal{E} = (E, D)$ be a semantically secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Assume $\mathcal{K} = \mathcal{R}$. Construct a new cipher $\mathcal{E}' = (E', D')$ defined over $(\mathcal{S}, \mathcal{M}, \mathcal{C})$, where $E'(s, m) := E(G(s), m)$ and $D'(s, c) := D(G(s), c)$. Show that $\mathcal{E}'$ is semantically secure.

3.20 (Nested PRG construction). Let $G_0 : \mathcal{S} \to \mathcal{R}_1$ and $G_1 : \mathcal{R}_1 \to \mathcal{R}_2$ be two secure PRGs. Show that $G(s) := G_1(G_0(s))$ mapping $\mathcal{S}$ to R_2 is a secure PRG.

3.21 (Self-nested PRG construction). Let G be a PRG that stretches n-bit strings to $2n$-bit strings. For $s \in \{0,1\}^n$, write $G(s) = G_0(s) \parallel G_1(s)$, so that $G_0(s)$ represents the first n bits of $G(s)$, and $G_1(s)$ represents the last n bits of $G(s)$. Define a new PRG G' that stretches n-bit strings to $4n$-bit strings, as follows: $G'(s) := G(G_0(s)) \parallel G(G_1(s))$. Show that if G is a secure PRG, then so is G'.

Hint: You can give a direct proof; alternatively, you can use the previous exercise together with Theorem 3.2.

Note: This construction is a special case of a more general construction discussed in Section 4.6.

3.22 (Bad seeds). Show that a secure PRG $G : \{0,1\}^n \to \mathcal{R}$ can become insecure if the seed is not uniformly random in $\mathcal{S}$.

(a) Consider the PRG $G' : \{0,1\}^{n+1} \to \mathcal{R} \times \{0,1\}$ defined as $G'(s_0 \parallel s_1) = (G(s_0), s_1)$. Show that G' is a secure PRG assuming G is secure.

(b) Show that G' becomes insecure if its random seed $s_0 \parallel s_1$ is chosen so that its last bit is always 0.

(c) Construct a secure PRG $G'' : \{0,1\}^{n+1} \to \mathcal{R} \times \{0,1\}$ that becomes insecure if its seed s is chosen so that the *parity* of the bits in s is always 0.

3.23 (Good intentions, bad idea). Let us show that a natural approach to strengthening a PRG is insecure. Let $m > n$ and let $G : \{0,1\}^n \to \{0,1\}^m$ be a PRG. Define a new generator $G'(s) := G(s) \oplus (0^{m-n} \parallel s)$ derived from G. Show that there is a secure PRG G for which G' is insecure.

Hint: Use the construction from part (a) of Exercise 3.22.

3.24 (Seed recovery attacks). Let G be a PRG defined over $(\mathcal{S}, \mathcal{R})$ where, $|\mathcal{S}|/|\mathcal{R}|$ is negligible, and suppose $\mathcal{A}$ is an adversary that given $G(s)$ outputs s with non-negligible probability. Show how to use $\mathcal{A}$ to construct a PRG adversary $\mathcal{B}$ that has non-negligible advantage in attacking G as a PRG. This shows that for a secure PRG it is intractable to recover the seed from the output.

3.25 (A PRG combiner). Suppose that G_1 and G_2 are PRG's defined over $(\mathcal{S}, \mathcal{R})$, where $\mathcal{R} = \{0,1\}^L$. Define a new PRG G' defined over $(\mathcal{S} \times \mathcal{S}, \mathcal{R})$, where $G'(s_1, s_2) = G_1(s_1) \oplus G_2(s_2)$. Show that if either G_1 or G_2 is secure (we may not know which one is secure), then G' is secure.

3.26 (A technical step in the proof of Lemma 3.5). This exercise develops a simple fact from probability that is helpful in understanding the proof of Lemma 3.5. Let X and Y be independent random variables, taking values in S and T, respectively, where Y is uniformly distributed over T. Let $f : S \to \{0,1\}$ and $g : S \to T$ be functions. Show that the events $f(\mathsf{X}) = 1$ and $g(\mathsf{X}) = \mathsf{Y}$ are independent, and the probability of the latter is $1/|T|$.

Chapter 4

Block ciphers

This chapter continues the discussion begun in the previous chapter on achieving privacy against eavesdroppers. Here, we study another kind of cipher, called a **block cipher**. We also study the related concept of a **pseudo-random function**.

Block ciphers are the "work horse" of practical cryptography: not only can they can be used to build a stream cipher, but they can be used to build ciphers with stronger security properties (as we will explore in Chapter 5), as well as many other cryptographic primitives.

4.1 Block ciphers: basic definitions and properties

Functionally, a **block cipher** is a deterministic cipher $\mathcal{E} = (E, D)$ whose message space and ciphertext space are the same (finite) set $\mathcal{X}$. If the key space of $\mathcal{E}$ is $\mathcal{K}$, we say that $\mathcal{E}$ is a block cipher **defined over** $(\mathcal{K}, \mathcal{X})$. We call an element $x \in \mathcal{X}$ a **data block**, and refer to $\mathcal{X}$ as the **data block space** of $\mathcal{E}$.

For every fixed key $k \in \mathcal{K}$, we can define the function $f_k := E(k, \cdot)$; that is, $f_k : \mathcal{X} \to \mathcal{X}$ sends $x \in \mathcal{X}$ to $E(k, x) \in \mathcal{X}$. The usual correctness requirement for any cipher implies that for every fixed key k, the function f_k is one-to-one, and as $\mathcal{X}$ is finite, f_k must be onto as well. Thus, f_k is a permutation on $\mathcal{X}$, and $D(k, \cdot)$ is the inverse permutation f_k^{-1}.

Although syntactically a block cipher is just a special kind of cipher, the security property we shall expect for a block cipher is actually much stronger than semantic security: for a randomly chosen key k, the permutation $E(k, \cdot)$ should — for all practical purposes — "look like" a random permutation. This is a notion that we will soon make more precise.

One very important and popular block cipher is AES (the Advanced Encryption Standard). We will study the internal design of AES in more detail below, but for now, we just give a very high-level description. AES keys are 128-bit strings (although longer key sizes may be used, such as 192-bits or 256-bits). AES data blocks are 128-bit strings. See Fig. 4.1. AES was designed to be quite efficient: one evaluation of the encryption (or decryption) function takes just a few hundred cycles on a typical computer.

The definition of security for a block cipher is formulated as a kind of "black box test." The intuition is the following: an efficient adversary is given a "black box." Inside the box is a permutation f on $\mathcal{X}$, which is generated via one of two random processes:

- $f := E(k, \cdot)$, for a randomly chosen key k, or

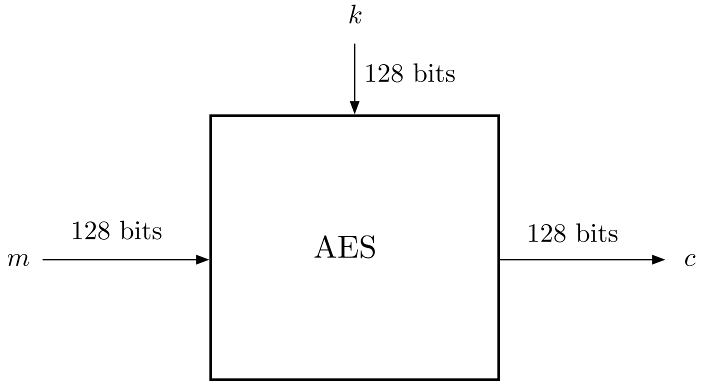


Figure 4.1: The block cipher AES

- f is a truly random permutation, chosen uniformly from among *all* permutations on $\mathcal{X}$.

The adversary cannot see inside the box, but he can "probe" it with questions: he can give the box a value $x \in \mathcal{X}$, and obtain the value $y := f(x) \in \mathcal{X}$. We allow the adversary to ask many such questions, and we quite liberally allow him to choose the questions in any way he likes; in particular, each question may even depend in some clever way on the answers to previous questions. Security means that the adversary should not be able to tell which type of function is inside the box — a randomly keyed block cipher, or a truly random permutation. Put another way, a secure block cipher should be **computationally indistinguishable** from a random permutation.

To make this definition more formal, let us introduce some notation:

$$\text{Perms}[\mathcal{X}]$$

denotes the set of *all* permutations on $\mathcal{X}$. Note that this is a very large set:

$$\big|\text{Perms}[\mathcal{X}]\big| = |\mathcal{X}|!.$$

For AES, with $|\mathcal{X}| = 2^{128}$, the number of permutations is about

$$\text{Perms}[\mathcal{X}] \approx 2^{2^{135}},$$

while the number of permutations defined by 128-bit AES keys is at most 2^{128}.

As usual, to define security, we introduce an attack game. Just like the attack game used to define a PRG, this attack game comprises two separate experiments. In both experiments, the adversary follows the same protocol; namely, it submits a sequence of queries $x_1, x_2, \ldots$ to the challenger; the challenger responds to query x_i with $f(x_i)$, where in the first experiment, $f := E(k, \cdot)$ for a randomly chosen $k \in \mathcal{K}$, while in the second experiment, f is randomly selected from $\text{Perms}[\mathcal{X}]$; throughout each experiment, the same f is used to answer all queries. When the adversary tires of querying the challenger, it outputs a bit.

Attack Game 4.1 (block cipher). For a given block cipher (E, D), defined over $(\mathcal{K}, \mathcal{X})$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define:

Experiment b:

- The challenger selects $f \in \text{Perms}[\mathcal{X}]$ as follows:

  if $b = 0$: $k \xleftarrow{\text{R}} \mathcal{K}$, $f \leftarrow E(k, \cdot)$;
  if $b = 1$: $f \xleftarrow{\text{R}} \text{Perms}[\mathcal{X}]$.

- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots$, the ith query is a data block $x_i \in \mathcal{X}$.

  The challenger computes $y_i \leftarrow f(x_i) \in \mathcal{X}$, and gives y_i to the adversary.

- The adversary computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \big|\Pr[W_0] - \Pr[W_1]\big|.$$

Finally, we say that $\mathcal{A}$ is a **Q-query BC adversary** if $\mathcal{A}$ issues at most Q queries. □

Fig. 4.2 illustrates Attack Game 4.1.

Definition 4.1 (secure block cipher). *A block cipher $\mathcal{E}$ is* ***secure*** *if for all efficient adversaries $\mathcal{A}$, the value* $\text{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

We stress that the queries made by the adversary in Attack Game 4.1 are allowed to be *adaptive*; that is, the adversary need not choose all its queries in advance; rather, it is allowed to concoct each query in some clever way that depends on the previous responses from the challenger (see Exercise 4.6).

As discussed in Section 2.2.5, Attack Game 4.1 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\text{BC}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.2.5 (namely, (2.11)) applies here as well:

$$\text{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \text{BC}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]. \tag{4.1}$$

4.1.1 Some implications of security

Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$. To exercise the definition of security a bit, we prove a couple of simple implications. For simplicity, we assume that $|\mathcal{X}|$ is large (i.e., super-poly).

4.1.1.1 A secure block cipher is unpredictable

We show that if $\mathcal{E}$ is secure in the sense of Definition 4.1, then it must be *unpredictable*, which means that every efficient adversary wins the following *prediction game* with negligible probability. In this game, the challenger chooses a random key k, and the adversary submits a sequence of queries $x_1, \ldots, x_Q$; in response to the ith query x_i, the challenger responds with $E(k, x_i)$. These queries are adaptive, in the sense that each query may depend on the previous responses. Finally, the adversary outputs a pair of values (x_{Q+1}, y), where $x_{Q+1} \notin \{x_1, \ldots, x_Q\}$. The adversary wins the game if $y = E(k, x_{Q+1})$.

To prove this implication, suppose that $\mathcal{E}$ is not unpredictable, which means there is an efficient adversary $\mathcal{A}$ that wins the above prediction game with non-negligible probability p. Then we can

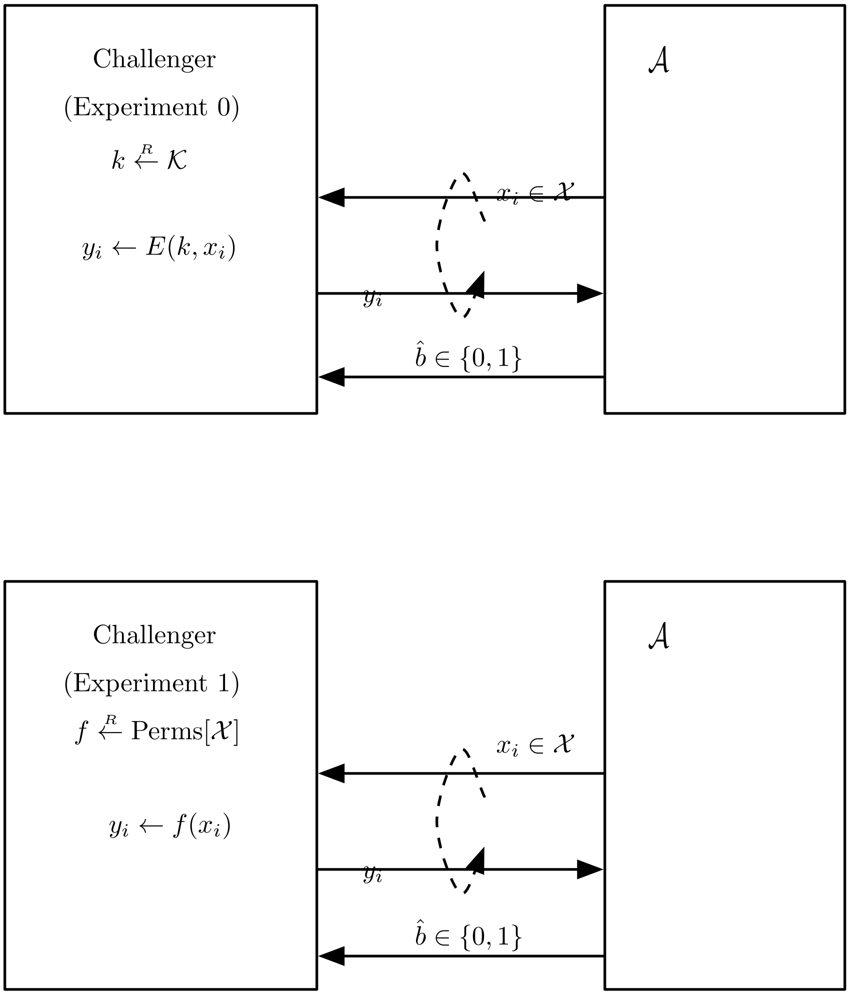


Figure 4.2: Attack Game 4.1

use $\mathcal{A}$ to break the security of $\mathcal{E}$ in the sense of Definition 4.1. To this end, we design an adversary $\mathcal{B}$ that plays Attack Game 4.1, and plays the role of challenger to $\mathcal{A}$ in the above prediction game. Whenever $\mathcal{A}$ makes a query x_i, adversary $\mathcal{B}$ passes x_i through to its own challenger, obtaining a response y_i, which it passes back to $\mathcal{A}$. Finally, when $\mathcal{A}$ outputs (x_{Q+1}, y), adversary $\mathcal{B}$ submits x_{Q+1} to its own challenger, obtaining y_{Q+1}, and outputs 1 if $y = y_{Q+1}$, and 0, otherwise.

On the one hand, if $\mathcal{B}$'s challenger is running Experiment 0, then $\mathcal{B}$ outputs 1 with probability p. On the other hand, if $\mathcal{B}$'s challenger is running Experiment 1, then $\mathcal{B}$ outputs 1 with negligible probability ϵ (since we are assuming $|\mathcal{X}|$ is super-poly). This implies that $\mathcal{B}$'s advantage in Attack Game 4.1 is $|p - \epsilon|$, which is non-negligible.

4.1.1.2 Unpredictability implies security against key recovery

Next, we show that if $\mathcal{E}$ is unpredictable, then it is *secure against key recovery*, which means that every efficient adversary wins the following *key-recovery game* with negligible probability. In this game, the adversary interacts with the challenger exactly as in the prediction game, except that at the end, it outputs a candidate key $\underset{\sim}{k} \in \mathcal{K}$, and wins the game if $\underset{\sim}{k} = k$.

To prove this implication, suppose that $\mathcal{E}$ is not secure against key recovery, which means that there is an efficient adversary $\mathcal{A}$ that wins the key-recovery game with non-negligible probability p. Then we can use $\mathcal{A}$ to build an efficient adversary $\mathcal{B}$ that wins the prediction game with probability at least p. Adversary $\mathcal{B}$ simply runs $\mathcal{A}$'s attack, and when $\mathcal{A}$ outputs $\underset{\sim}{k}$, adversary $\mathcal{B}$ chooses an arbitrary $x_{Q+1} \notin \{x_1, \ldots, x_Q\}$, computes $y \leftarrow E(\underset{\sim}{k}, x_{Q+1})$, and outputs (x_{Q+1}, y).

It is easy to see that if $\mathcal{A}$ wins the key-recovery game, then $\mathcal{B}$ wins the prediction game.

4.1.1.3 Key space size and exhaustive-search attacks

Combining the above two implications, we conclude that if $\mathcal{E}$ is a secure block cipher, then it must be secure against key recovery. Moreover, if $\mathcal{E}$ is secure against key recovery, it must be the case that $|\mathcal{K}|$ is large.

One way to see this is as follows. An adversary can always win the key-recovery game with probability $1/|\mathcal{K}|$ by simply choosing $\underset{\sim}{k}$ from $\mathcal{K}$ at random. If $|\mathcal{K}|$ is not super-poly, then $1/|\mathcal{K}|$ is non-negligible. Hence, when $|\mathcal{K}|$ is not super-poly this simple key guessing adversary wins the key-recovery game with non-negligible probability.

We can trade success probability for running time using a different attack, called an *exhaustive-search attack*. In this attack, our adversary makes a few, arbitrary queries $x_1, \ldots, x_Q$ in the key-recovery game, obtaining responses $y_1, \ldots, y_Q$. One can argue — heuristically, at least, assuming that $|\mathcal{X}| \geq |\mathcal{K}|$ and $|\mathcal{X}|$ is super-poly — that for fairly small values of Q ($Q = 2$, in fact), with all but negligible probability, only one key k satisfies

$$y_i = E(k, x_i) \quad \text{for} \quad i = 1, \ldots, Q. \tag{4.2}$$

So our adversary simply tries all possible keys to find one that satisfies (4.2). If there is only one such key, then the key that our adversary finds will be the key chosen by the challenger, and the adversary will win the game. Thus, our adversary wins the key-recovery game with all but negligible probability; however, its running time is linear in $|\mathcal{K}|$.

This time/advantage trade-off can be easily generalized. Indeed, consider an adversary that chooses t keys at random, testing if each such key satisfies (4.2). The running time of such an adversary is linear in t, and it wins the key-recovery game with probability $\approx t/|\mathcal{K}|$.

We describe a few real-world exhaustive search attacks in Section 4.2.2. We present a detailed treatment of exhaustive search in Section 4.7.2 where, in particular, we justify the heuristic assumption used above that with high probability there is at most one key satisfying (4.2).

So it is clear that if a block cipher has any chance of being secure, it must have a large key space, simply to avoid a key-recovery attack.

4.1.2 Efficient implementation of random permutations

Note that the challenger's protocol in Experiment 1 of Attack Game 4.1 is not very efficient: he is supposed to choose a *very* large random object. Indeed, just writing down an element of Perms[$\mathcal{X}$] would require about $|\mathcal{X}| \log_2 |\mathcal{X}|$ bits. For AES, with $|\mathcal{X}| = 2^{128}$, this means about 10^{40} bits!

While this is not a problem from a purely definitional point of view, for both aesthetic and technical reasons, it would be nice to have a more efficient implementation. We can do this by using a "lazy" implementation of f. That is, the challenger represents the random permutation f by keeping track of input/output pairs (x_i, y_i). When the challenger receives the ith query x_i, he tests whether $x_i = x_j$ for some $j < i$; if so, he sets $y_i \leftarrow y_j$ (this ensures that the challenger implements a function); otherwise, he chooses y_i at random from the set $\mathcal{X} \setminus \{y_1, \ldots, y_{i-1}\}$ (this ensures that the function is a permutation); finally, he sends y_i to the adversary. We can write the logic of this implementation of the challenger as follows:

```
upon receiving the $i$th query $x_i \in \mathcal{X}$ from $\mathcal{A}$ do:
    if $x_i = x_j$ for some $j < i$
        then $y_i \leftarrow y_j$
        else $y_i \xleftarrow{R} \mathcal{X} \setminus \{y_1, \ldots, y_{i-1}\}$
    send $y_i$ to $\mathcal{A}$.
```

To make this implementation as fast as possible, one would implement the test "if $x_i = x_j$ for some $j < i$" using an appropriate dictionary data structure (hash tables, digital search tries, balanced trees, etc.). Assuming random elements of $\mathcal{X}$ can be generated efficiently, one way to implement the step "$y_i \xleftarrow{R} \mathcal{X} \setminus \{y_1, \ldots, y_{i-1}\}$" is as follows:

```
repeat   $y \xleftarrow{R} \mathcal{X}$   until $y \notin \{y_1, \ldots, y_{i-1}\}$
$y_i \leftarrow y$,
```

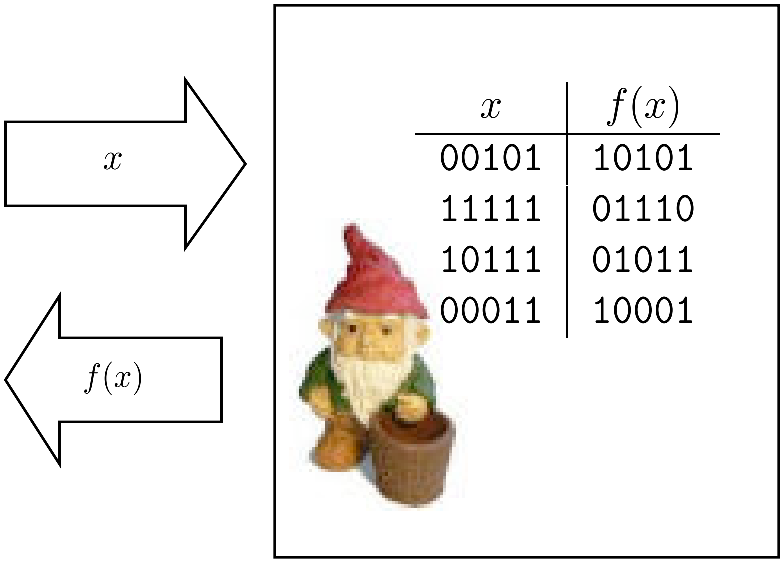
again, using appropriate dictionary data structure for the tests "$y \notin \{y_1, \ldots, y_{i-1}\}$." When $i < |\mathcal{X}|/2$ the loop will run for only two iterations in expectation.

One way to visualize this implementation is that the challenger in Experiment 1 is a "black box," but inside the box is a little **faithful gnome** whose job it is to maintain the table of input/output pairs which represents a random permutation f. See Fig. 4.3.

4.1.3 Strongly secure block ciphers

Note that in Attack Game 4.1, the decryption algorithm D was never used. One can in fact define a stronger notion of security by defining an attack game in which the adversary is allowed to make two types of queries to the challenger:

forward queries: the adversary sends a value $x_i \in \mathcal{X}$ to the challenger, who sends $y_i := f(x_i)$ to the adversary;



x	$f(x)$
00101	10101
11111	01110
10111	01011
00011	10001

Figure 4.3: A faithful gnome implementing random permutation f

inverse queries: the adversary sends a value $y_i \in \mathcal{X}$ to the challenger, who sends $x_i := f^{-1}(y_i)$ to the adversary (in Experiment 0 in the attack game, this is done using algorithm D).

One then defines a corresponding advantage for this attack game. A block cipher is then called **strongly secure** if for all efficient adversaries, this advantage is negligible. We leave it to the reader to work out the details of this definition (see Exercise 4.9). We will not make use of this notion in this text, other than an example application in a later chapter (Exercise 9.12).

4.1.4 Using a block cipher directly for encryption

Since a block cipher is a special kind of cipher, we can of course consider using it directly for encryption. The question is: is a secure block cipher also semantically secure?

The answer to this question is "yes," provided the message space is equal to the data block space. This will be implied by Theorem 4.1 below. However, data blocks for practical block ciphers are very short: as we mentioned, data blocks for AES are just 128-bits long. If we want to encrypt longer messages, a natural idea would be to break up a long message into a sequence of data blocks, and encrypt each data block separately. This use of a block cipher to encrypt long messages is called **electronic codebook mode**, or **ECB mode** for short.

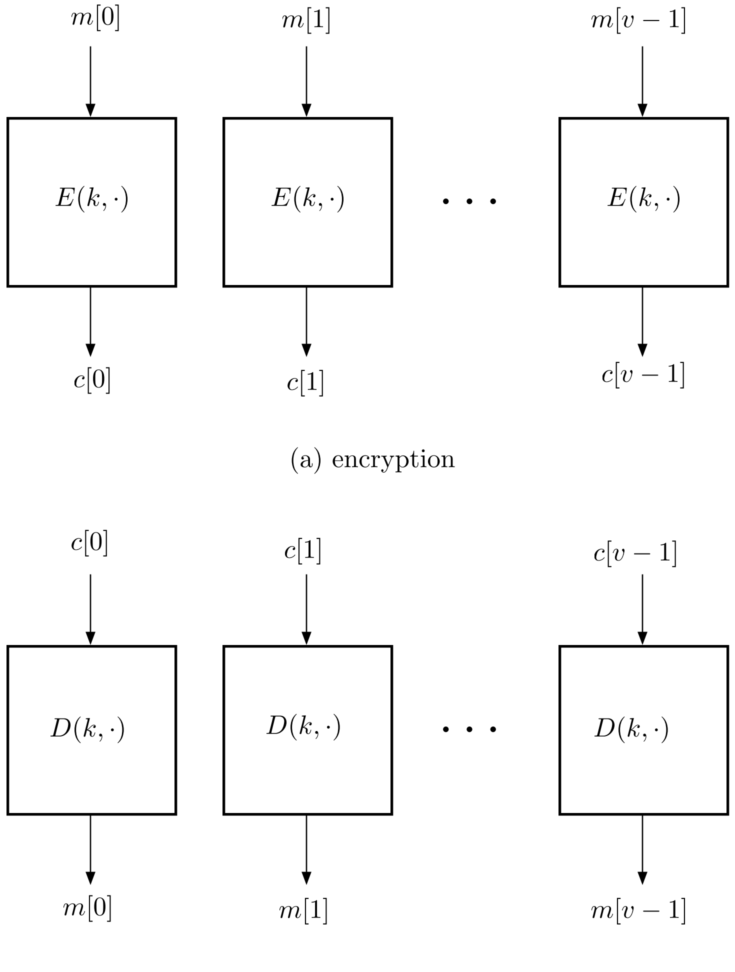
More precisely, suppose $\mathcal{E} = (E, D)$ is a block cipher defined over $(\mathcal{K}, \mathcal{X})$. For any poly-bounded $\ell \geq 1$, we can define a cipher $\mathcal{E}' = (E', D')$, defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{X}^{\leq \ell})$, as follows.

- For $k \in \mathcal{K}$ and $m \in \mathcal{X}^{\leq \ell}$, with $v := |m|$, we define

$$E'(k, m) := \big(E(k, m[0]), \dots, E(k, m[v-1])\big).$$

- For $k \in \mathcal{K}$ and $c \in \mathcal{X}^{\leq \ell}$, with $v := |c|$, we define

$$D'(k, c) := \big(D(k, c[0]), \dots, D(k, c[v-1])\big).$$



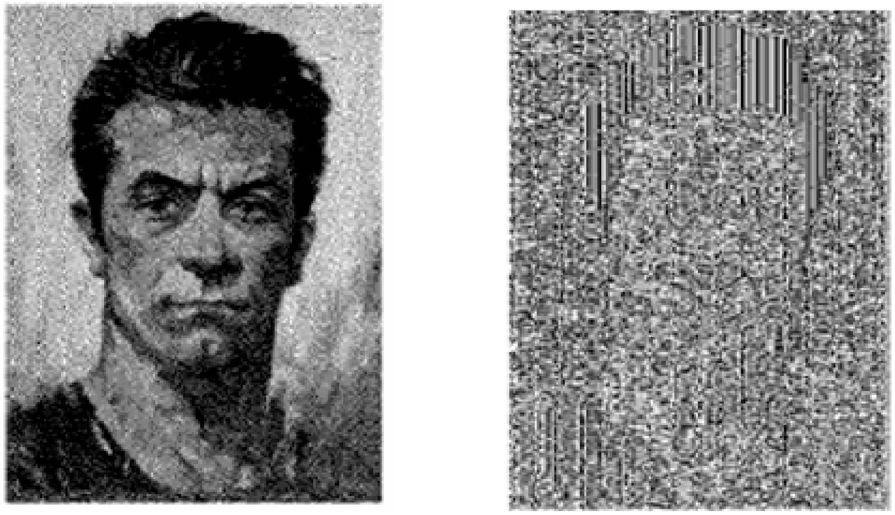
(a) encryption

(b) decryption

Figure 4.4: Encryption and decryption for ECB mode

Fig. 4.4 illustrates encryption and decryption. We call $\mathcal{E}'$ the **ℓ-wise ECB cipher derived from $\mathcal{E}$**.

The ECB cipher is very closely related to the substitution cipher discussed in Examples 2.3 and 2.6. The main difference is that instead of choosing a permutation at random from among all possible permutations on $\mathcal{X}$, we choose one from the much smaller set of permutations $\{E(k,\cdot) : k \in \mathcal{K}\}$. A less important difference is that in Example 2.3, we defined our substitution cipher to have a fixed length, rather than a variable length message space (this was really just an arbitrary choice — we could have defined the substitution cipher to have a variable length message space). Another difference is that in Example 2.3, we suggested an alphabet of size 27, while if we use a block cipher like AES with a 128-bit block size, the "alphabet" is much larger — it has 2^{128} elements. Despite these differences, some of the vulnerabilities discussed in Example 2.6 apply here as well. For example, an adversary can easily distinguish an encryption of two messages $m_0, m_1 \in \mathcal{X}^2$, where m_0 consists of two equal blocks (i.e., $m_0[0] = m_0[1]$) and m_1 consists of two unequal blocks (i.e.,



(a) plaintext

(b) plaintext encrypted in ECB mode using AES

Figure 4.5: Encrypting in ECB mode

$m_1[0] \neq m_1[1]$). For this reason alone, the *ECB cipher does not satisfy our definition of semantic security, and its use as an encryption scheme is strongly discouraged.*

This ability to easily tell which plaintext blocks are the same is graphically illustrated in Fig. 4.5 (due to B. Preneel). Here, visual data is encrypted in ECB mode, with each data block encoding some small patch of pixels in the original data. Since identical patches of pixels get mapped to identical blocks of ciphertext, some patterns in the original picture are visible in the ciphertext.

Note, however, that some of the vulnerabilities discussed in Example 2.6 do not apply directly here. Suppose we are encrypting ASCII text. If the block size of the cipher is 128-bits, then each character of text will be typically encoded as a byte, with 16 characters packed into a data block. Therefore, an adversary will not be able to trivially locate positions where individual characters are repeated, as was the case in Example 2.6.

We close this section with a proof that ECB mode is in fact secure if the message space is restricted to sequences on *distinct* data blocks. This includes as a special case the encryption of single-block messages. It is also possible to encode longer messages as sequences of distinct data blocks. For example, suppose we are using AES, which has 128-bit data blocks. Then we could allocate, say, 32 bits out of each block as a counter, and use the remaining 96 bits for bits of the message. With such a strategy, we can encode any message of up to $2^{32} \cdot 96$ bits as a sequence of distinct data blocks. Of course, this strategy has the disadvantage that ciphertexts are 33% longer than plaintexts.

Theorem 4.1. *Let $\mathcal{E} = (E, D)$ be a block cipher. Let $\ell \geq 1$ be any poly-bounded value, and let $\mathcal{E}' = (E', D')$ be the ℓ-wise ECB cipher derived from $\mathcal{E}$, but with the message space restricted to all sequences of at most ℓ* distinct *data blocks. If $\mathcal{E}$ is a secure block cipher, then $\mathcal{E}'$ is a semantically* *secure cipher.*

> *In particular, for every SS adversary $\mathcal{A}$ that plays Attack Game 2.1 with respect to $\mathcal{E}'$, there exists a BC adversary $\mathcal{B}$ that plays Attack Game 4.1 with respect to $\mathcal{E}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}'] = 2 \cdot \mathrm{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{4.3}$$

Proof idea. The basic idea is that if an adversary is given an encryption of a message, which is a sequence of distinct data blocks, then what he sees is effectively just a sequence of random data blocks (sampled without replacement). □

Proof. If $\mathcal{E}$ is defined over $(\mathcal{K}, \mathcal{X})$, let $\mathcal{X}_*^{\leq \ell}$ denote the set of all sequences of at most ℓ distinct elements of $\mathcal{X}$.

Let $\mathcal{A}$ be an efficient adversary that attacks $\mathcal{E}'$ as in Attack Game 2.1. Our goal is to show that $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}']$ is negligible, assuming that $\mathcal{E}$ is a secure block cipher. It is more convenient to work with the bit-guessing version of the SS attack game. We prove:

$$\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}'] = \mathrm{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}] \tag{4.4}$$

for some efficient adversary $\mathcal{B}$. Then (4.3) follows from Theorem 2.10.

So consider the adversary $\mathcal{A}$'s attack of $\mathcal{E}'$ in the bit-guessing version of Attack Game 2.1. In this game, $\mathcal{A}$ presents the challenger with two messages m_0, m_1 of the same length; the challenger then chooses a random key k and a random bit b, and encrypts m_b under k, giving the resulting ciphertext c to $\mathcal{A}$; finally, $\mathcal{A}$ outputs a bit $\hat{b}$. The adversary $\mathcal{A}$ wins the game if $\hat{b} = b$.

The logic of the challenger in this game may be written as follows:

```
upon receiving $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$, with $v := |m_0| = |m_1|$, do:
    $b \xleftarrow{R} \{0, 1\}$
    $k \xleftarrow{R} \mathcal{K}$
    $c \leftarrow (E(k, m_b[0]), \ldots, E(k, m_b[v-1]))$
    send $c$ to $\mathcal{A}$.
```

Let us call this **Game 0.** We will define two more games: Game 1 and Game 2. For $j = 0, 1, 2$, we define W_j to be the event that $\hat{b} = b$ in Game j. By definition, we have

$$\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}'] = |\Pr[W_0] - 1/2|. \tag{4.5}$$

Game 1. This is the same as Game 0, except the challenger uses a random $f \in \mathrm{Perms}[\mathcal{X}]$ in place of $E(k, \cdot)$. Our challenger now looks like this:

```
upon receiving $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$, with $v := |m_0| = |m_1|$, do:
    $b \xleftarrow{R} \{0, 1\}$
    $f \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$
    $c \leftarrow (f(m_b[0]), \ldots, f(m_b[v-1]))$
    send $c$ to $\mathcal{A}$.
```

Intuitively, the fact that $\mathcal{E}$ is a secure block cipher implies that the adversary should not notice the switch. To prove this rigorously, we show how to build a BC adversary $\mathcal{B}$ that is an elementary wrapper around $\mathcal{A}$, such that

$$|\Pr[W_0] - \Pr[W_1]| = \mathrm{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{4.6}$$

The design of $\mathcal{B}$ follows directly from the logic of Games 0 and 1. Adversary $\mathcal{B}$ plays Attack Game 4.1 with respect to $\mathcal{E}$, and works as follows:

> Let f be the function chosen by $\mathcal{B}$'s BC challenger in Attack Game 4.1. We let $\mathcal{B}$ play the role of challenger to $\mathcal{A}$, as follows:
>
> ```
> upon receiving $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$ from $\mathcal{A}$, with $v := |m_0| = |m_1|$, do:
>     $b \xleftarrow{R} \{0,1\}$
>     $c \leftarrow \big(f(m_b[0]), \ldots, f(m_b[v-1])\big)$
>     send $c$ to $\mathcal{A}$.
> ```
>
> Note that $\mathcal{B}$ computes the values $f(m_b[0]), \ldots, f(m_b[v-1])$ by querying its own BC challenger. Finally, when $\mathcal{A}$ outputs a bit $\hat{b}$, $\mathcal{B}$ outputs the bit $\delta(\hat{b}, b)$, as defined in (3.7).

It should be clear that when $\mathcal{B}$ is in Experiment 0 of its attack game, it outputs 1 with probability $\Pr[W_0]$, while when $\mathcal{B}$ is in Experiment 1 of its attack game, it outputs 1 with probability $\Pr[W_1]$. The equation (4.6) now follows.

Game 2. We now rewrite the challenger in Game 1 so that it uses the "faithful gnome" implementation of a random permutation, discussed in Section 4.1.2. Each of the messages m_0 and m_1 is required to consist of distinct data blocks (our challenger does not have to verify this), and so our gnome's job is quite easy: it does not even have to look at the input data blocks, as these are guaranteed to be distinct; however, it still has to ensure that the output blocks it generates are distinct.

We can express the logic of our challenger as follows:

```
$y_0 \xleftarrow{R} \mathcal{X}$, $y_1 \xleftarrow{R} \mathcal{X} \setminus \{y_0\}$, ..., $y_{\ell-1} \xleftarrow{R} \mathcal{X} \setminus \{y_0, \ldots, y_{\ell-2}\}$
upon receiving $m_0, m_1 \in \mathcal{X}_*^{\leq \ell}$, with $v := |m_0| = |m_1|$, do:
    $b \xleftarrow{R} \{0,1\}$
    $c \leftarrow (y_0, \ldots, y_{v-1})$
    send $c$ to $\mathcal{A}$.
```

Since our gnome is faithful, we have

$$\Pr[W_1] = \Pr[W_2]. \tag{4.7}$$

Moreover, we claim that

$$\Pr[W_2] = 1/2. \tag{4.8}$$

This follows from the fact that in Game 2, the adversary's output $\hat{b}$ is a function of its own random choices, together with $y_0, \ldots, y_{\ell-1}$; since these values are (by definition) independent of b, it follows that $\hat{b}$ and b are independent. The equation (4.8) now follows.

Combining (4.5), (4.6), (4.7), and (4.8), yields (4.4), which completes the proof. □

4.1.5 Mathematical details

As usual, we address a few mathematical details that were glossed over above.

Since a block cipher is just a special kind of cipher, there is really nothing to say about the definition of a block cipher that was not already said in Section 2.3. As usual, Definition 4.1 needs

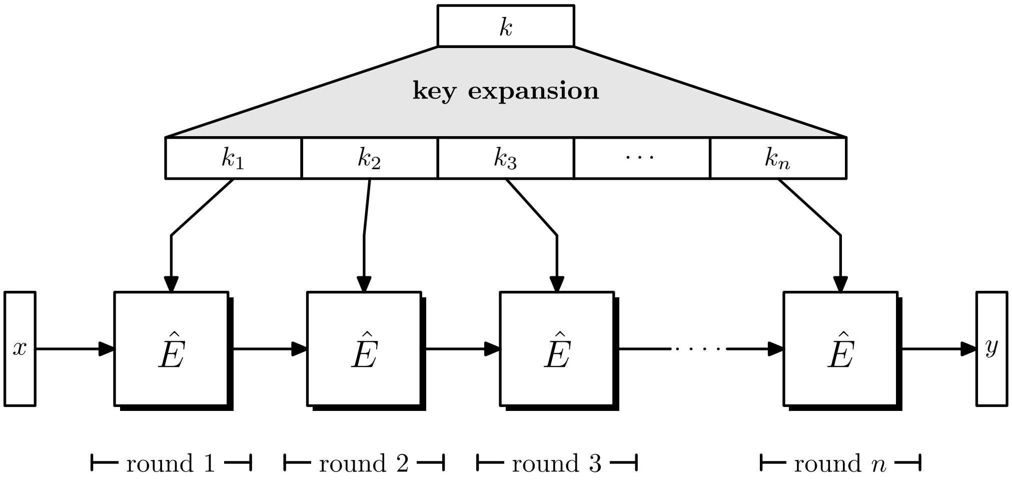


Figure 4.6: Encryption in a real-world block cipher

to be properly interpreted. First, in Attack Game 4.1, it is to be understood that for each value of the security parameter λ, we get a different probability space, determined by the random choices of the challenger and the random choices of the adversary. Second, the challenger generates a system parameter Λ, and sends this to the adversary at the very start of the game. Third, the advantage $\mathrm{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is a function of the security parameter λ, and security means that this function is a negligible function.

4.2 Constructing block ciphers in practice

Block ciphers are a basic primitive in cryptography from which many other systems are built. Virtually all block ciphers used in practice use the same basic framework called the **iterated cipher** paradigm. To construct an iterated block cipher the designer makes two choices:

- First, he picks a simple block cipher $\hat{\mathcal{E}} := (\hat{E}, \hat{D})$ that is clearly insecure on its own. We call $\hat{\mathcal{E}}$ the **round cipher**.
- Second, he picks a simple (not necessarily secure) PRG G that is used to expand the key k into d keys $k_1, \ldots, k_d$ for $\hat{\mathcal{E}}$. We call G the **key expansion function**.

Once these two choices are made, the iterated block cipher $\mathcal{E}$ is completely specified. The encryption algorithm $E(k, x)$ works as follows (see Fig. 4.6):

	key size (bits)	block size (bits)	number of rounds	performance[1] (MB/sec)
DES	56	64	16	80
3DES	168	64	48	30
AES-128	128	128	10	163
AES-256	256	128	14	115

Table 4.1: Sample block ciphers

Algorithm $E(k, x)$:

- step 1. **key expansion**: use the key expansion function G to stretch the key k of $\mathcal{E}$ to d keys of $\hat{\mathcal{E}}$:

$$(k_1, \ldots, k_d) \leftarrow G(k)$$

- step 2. **iteration**: for $i = 1, \ldots, d$ apply $\hat{E}(k_i, \cdot)$, namely:

$$y \leftarrow \hat{E}(k_d,\ \hat{E}(k_{d-1}, \ldots, \hat{E}(k_2,\ \hat{E}(k_1,\ x)) \ldots))$$

Each application of $\hat{E}$ is called a **round** and the total number of rounds is d. The keys $k_1, \ldots, k_d$ are called **round keys**. The decryption algorithm $D(k, y)$ is identical except that the round keys are applied in reverse order. $D(k, y)$ is defined as:

$$x \leftarrow \hat{D}(k_1,\ \hat{D}(k_2, \ldots, \hat{D}(k_{d-1},\ \hat{D}(k_d,\ y)) \ldots))$$

Table 4.1 lists a few common block ciphers and their parameters. We describe DES and AES in the next section.

Does iteration give a secure block cipher? Nobody knows. However, heuristic evidence suggests that security of a block cipher comes from iterating a simple cipher many times. Not all round ciphers will work. For example, iterating a linear function

$$\hat{E}(k, x) := k \cdot x \bmod q$$

will never result in a secure block cipher since the iterate of $\hat{E}$ is just another linear function. There is currently no way to classify which round ciphers will eventually result in a secure block cipher. Moreover, for a candidate round cipher $\hat{E}$ there is no rigorous methodology to gauge how many times it needs to be iterated before it becomes a secure block cipher. All we know is that certain functions, like linear functions, never lead to secure block ciphers, while simple non-linear functions appear to give a secure block cipher after a few iterations.

The challenge for the cryptographer is to come up with a fast round cipher that converges to a secure block cipher within a few rounds. Looking at Table 4.1 one is impressed that AES-128 uses a simple round cipher and yet seems to produce a secure block cipher after only ten rounds.

[1] OpenSSL 1.0.1e on Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz (Haswell).

A word of caution. While this section explains the inner workings of several block ciphers, it does not teach how to design new block ciphers. In fact, one of the main take-away messages from this section is that readers should not design block ciphers on their own, but instead always use the standard ciphers described here. Block-cipher design is non-trivial and many years of analysis are needed before one gains confidence in a specific proposal. Furthermore, readers should not even implement block ciphers on their own since implementations of block-ciphers tend to be vulnerable to timing and power attacks, as discussed in Section 4.3.2. It is much safer to use one of the standard implementations freely available in crypto libraries such as OpenSSL. These implementations have gone through considerable analysis over the years and have been hardened to resist attack.

4.2.1 Case study: DES

The Data Encryption Standard (DES) was developed at IBM in response to a solicitation for proposals from the National Bureau of Standards (now the National Institute of Standards). It was published in the Federal Register in 1975 and was adopted as a standard for "unclassified" applications in 1977. The DES algorithm single-handedly jump started the field of cryptanalysis; everyone wanted to break it. Since inception, DES has undergone considerable analysis that led to the development of many new tools for analyzing block ciphers.

The precursor to DES is an earlier IBM block cipher called Lucifer. Certain variants of Lucifer operated on 128-bit blocks using 128-bit keys. The National Bureau of Standards, however, asked for a block cipher that used shorter blocks (64 bits) and shorter keys (56 bits). In response, the IBM team designed a block cipher that met these requirements and eventually became DES. Setting the DES key size to 56 bits was widely criticized and lead to speculation that DES was deliberately made weak due to pressure from US intelligence agencies. In the coming chapters, we will see that reducing the block size to 64 bits also creates problems.

Due to its short key size, the DES algorithm is now considered insecure and should not be used. However, a strengthened version of DES called Triple-DES (3DES) was reaffirmed as a US standard in 1998. The National Institute of Standards, NIST, has approved Triple-DES through the year 2030 for government use. In 2002 DES was superseded by a new and more efficient block cipher standard called AES that uses 128-bit (or longer) keys, and operates on 128-bit blocks.

4.2.1.1 The DES algorithm

The DES algorithm consists of 16 iterations of a simple round cipher. To describe DES it suffices to describe the DES round cipher and the DES key expansion function. We describe each in turn.

The Feistel permutation. One of the key innovations in DES, invented by Horst Feistel at IBM, builds a permutation from an arbitrary function. Let $f : \mathcal{X} \to \mathcal{X}$ be a function. We construct a permutations $\pi : \mathcal{X}^2 \to \mathcal{X}^2$ as follows (Fig. 4.7):

$$\pi(x, y) := \big(y,\ x \oplus f(y)\big)$$

To show that π is one-to-one we construct its inverse, which is given by:

$$\pi^{-1}(u, v) = \big(v \oplus f(u),\ u\big)$$

The function π is called a **Feistel permutation** and is used to build the DES round cipher. The composition of n Feistel permutations is called an **n-round Feistel network**. Block ciphers

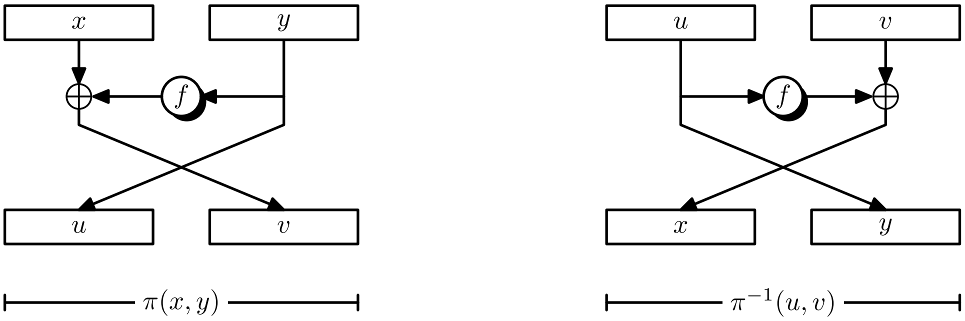


Figure 4.7: The Feistel permutation

designed as a Feistel network are called **Feistel ciphers**. For DES, the function f takes 32-bit inputs and the resulting permutation π operates on 64-bit blocks.

Note that the Feistel inverse function π^{-1} is almost identical to π. As a result the same hardware can be used for evaluating both π and π^{-1}. This in turn means that the encryption and decryption circuits can use the same hardware.

The DES round function $F(k,x)$. The DES encryption algorithm is a 16-round Feistel network where each round uses a different function $f : \mathcal{X} \to \mathcal{X}$. In round number i the function f is defined as

$$f(x) := F(k_i, x)$$

where k_i is a 48-bit key for round number i and F is a fixed function called the **DES round function**. The function F is the centerpiece of the DES algorithm and is shown in Fig. 4.8. F uses several auxiliary functions E, P, and $S_1, \ldots, S_8$ defined as follows:

- The function E expands a 32-bit input to a 48-bit output by rearranging and replicating the input bits. For example, E maps input bit number 1 to output bits 2 and 48; it maps input bit 2 to output bit number 3, and so on.

- The function P, called the **mixing permutation**, maps a 32-bit input to a 32-bit output by rearranging the bits of the input. For example, P maps input bit number 1 to output bit number 9; input bit number 2 to output number 15, and so on.

- At the heart of the DES algorithm are the functions $S_1, \ldots, S_8$ called **S-boxes**. Each S-box S_i maps a 6-bit input to a 4-bit output by a lookup table. The DES standard lists these 8 look-up tables, where each table contains 64 entries.

Given these functions, the DES round function $F(k,x)$ works as follows:

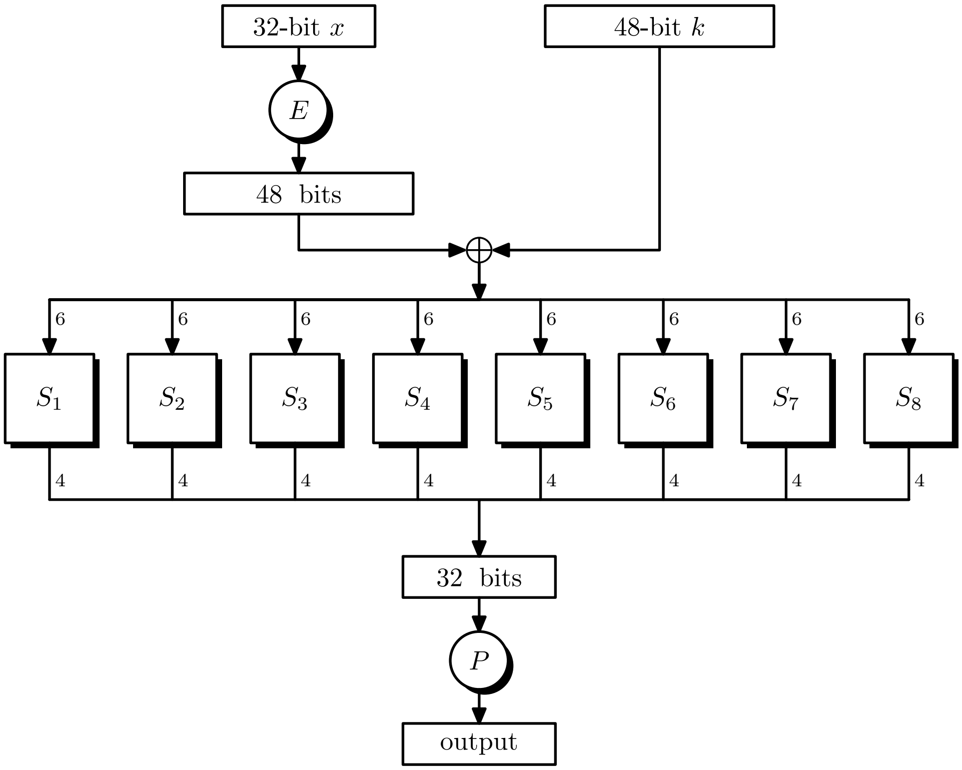


Figure 4.8: The DES round function $F(k, x)$

```
input: $k \in \{0,1\}^{48}$ and $x \in \{0,1\}^{32}$
output: $y \in \{0,1\}^{32}$

$F(k, x)$:
    $t \leftarrow E(x) \oplus k \quad \in \{0,1\}^{48}$
    separate $t$ into 8 groups of 6-bits each: $t := t_1 \parallel \cdots \parallel t_8$
    for $i = 1$ to $8$ :    $s_i \leftarrow S_i(t_i)$
    $s \leftarrow s_1 \parallel \cdots \parallel s_8 \quad \in \{0,1\}^{32}$
    $y \leftarrow P(s) \quad \in \{0,1\}^{32}$
    output $y$
```

Except for the S-boxes, the DES round cipher is made up entirely of XORs and bit permutations. The eight S-boxes are the only components that introduce non-linearity into the design. IBM published the criteria used to design the S-boxes in 1994 [44], after the discovery of a powerful attack technique called "differential cryptanalysis" in the open literature. This IBM report makes it clear that the designers of DES knew in 1973 of attack techniques that would only become known in the open literature many years later. They designed DES to resist these attacks. The reason for keeping the S-box design criteria secret is explained in the following quote [44]:

> The design [of DES] took advantage of knowledge of certain cryptanalytic techniques, most prominently the technique of "differential cryptanalysis," which were not known in the published literature. After discussions with the NSA, it was decided that disclosure of the design considerations would reveal the technique of differential cryptanalysis, a powerful technique that can be used against many ciphers. This in turn would weaken the competitive advantage that the United States enjoyed over other countries in the field of cryptography.

Once differential cryptanalysis became public, there was no longer any reason to keep the design of DES secret. Due to the importance of the S-boxes we list a few of the criteria that went into their design, as explained in [44].

1. The size of the look-up tables, mapping 6-bits to 4-bits, was the largest that could be accommodated on a single chip using 1974 technology.

2. No output bit of an S-box should be close to a linear function of the input bits. That is, if we select any output bit and any subset of the 6 input bits, then the fraction of inputs for which this output bit equals the XOR of these input bits should be close to $1/2$.

3. If we fix the leftmost and rightmost bits of the input to an S-box then the resulting 4-bit to 4-bit function is one-to-one. In particular, this implies that each S-box is a 4-to-1 map.

4. Changing one bit of the input to an S-box changes at least two bits of the output.

5. For each $\Delta \in \{0,1\}^6$, among the 64 pairs $x, y \in \{0,1\}^6$ such that $x \oplus y = \Delta$, the quantity $S_i(x) \oplus S_i(y)$ must not attain a single value more than eight times.

These criteria were designed to make DES as strong as possible, given the 56-bit key-size constraints. It is now known that if the S-boxes were simply chosen at random, then with high probability the resulting DES cipher would be insecure. In particular, the secret key could be recovered after only several million queries to the challenger.

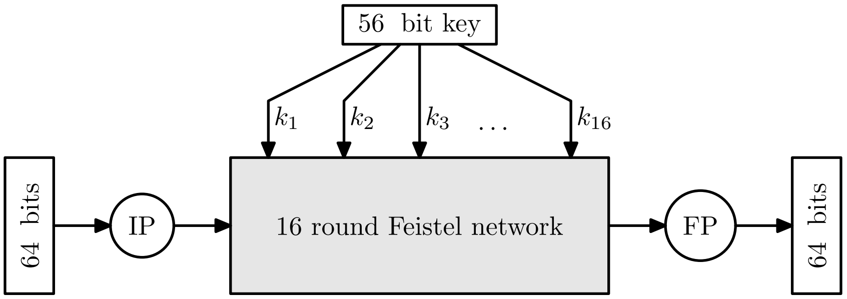


Figure 4.9: The complete DES circuit

Beyond the S-boxes, the mixing permutation P also plays an important role. It ensures that the S-boxes do not always operate on the same group of 6 bits. Again, [44] lists a number of criteria used to choose the permutation P. If the permutation P was simply chosen at random then DES would be far less secure.

The key expansion function. The DES key expansion function G takes as input the 56-bit key k and outputs 16 keys $k_1, \ldots, k_{16}$, each 48-bits long. Each key k_i consists of 48 bits chosen from the 56-bit key, with each k_i using a different subset of bits from k.

The DES algorithm. The complete DES algorithm is shown in Fig. 4.9. It consists of 16 iterations of the DES round cipher plus initial and final permutations called IP and FP. These permutations simply rearrange the 64 incoming and outgoing bits. The permutation FP is the inverse of IP.

IP and FP have no cryptographic significance and were included for unknown reasons. Since bit permutations are slow in software, but fast in hardware, one theory is that IP and FP are intended to deliberately slow down software implementations of DES.

4.2.2 Exhaustive search on DES: the DES challenges

Recall that an exhaustive search attack on a block cipher (E, D) (Section 4.1.1.2) refers to the following attack: the adversary is given a small number of plaintext blocks $x_1, \ldots, x_Q \in \mathcal{X}$ and their encryption $y_1, \ldots, y_Q$ using a block cipher key k in $\mathcal{K}$. The adversary finds k by trying all possible keys $\underset{\sim}{k} \in \mathcal{K}$ until it finds a key that maps all the given plaintext blocks to the given ciphertext blocks. If enough ciphertext blocks are given, then k is the only such key, and it will be found by the adversary.

For block ciphers like DES and AES-128 *three* blocks are enough to ensure that with high probability there is a unique key mapping the given plaintext blocks to the given ciphertext blocks. We will see why in Section 4.7.2 where we discuss ideal ciphers and their properties. For now it suffices to know that given three plaintext/ciphertext blocks an attacker can use exhaustive search to find the secret key k.

In 1974, when DES was designed, an exhaustive search attack on a key space of size 2^{56} was believed to be infeasible. With improvements in computer hardware it was shown that a 56-bit key is woefully inadequate.

To prove that exhaustive search on DES is feasible, RSA data security set up a sequence of challenges, called the **DES challenges**. The rules were simple: on a pre-announced date RSA data security posted three input/output pairs for DES. The first group to find the corresponding key wins ten thousand US dollars. To make the challenge more entertaining, the challenge consisted of n DES outputs $y_1, y_2, \dots, y_n$ where the first three outputs, y_1, y_2, y_3, were the result of applying DES to the 24-byte plaintext message:

$$\underbrace{\texttt{The unkn}}_{x_1}\underbrace{\texttt{own mess}}_{x_2}\underbrace{\texttt{age is:\ \ }}_{x_3}$$

which consists of three DES blocks: each block is 8 bytes which is 64 bits, a single DES block. The goal was to find a DES key that maps x_i to y_i for all $i = 1, 2, 3$ and then use this key to decrypt the secret message encoded in $y_4 \dots y_n$.

The first challenge was posted in January 1997. It was solved by the DESCHALL project in 96 days. The team used a distributed Internet search with the help of 78,000 volunteers who contributed idle cycles on their machines. The person whose machine found the secret-key received 40% of the prize money. Once decrypted, the secret message encoded in $y_4 \dots y_n$ was "Strong cryptography makes the world a safer place."

A second challenge, posted in January 1998, was solved by the **distributed.net** project in only 41 days by conducting a similar Internet search, but on a larger scale.

In early 1998, the Electronic Frontiers Foundation (EFF) contracted Paul Kocher to construct a dedicated machine to do DES exhaustive key search. The machine, called **DeepCrack**, cost 250,000 US dollars and contained about 1900 dedicated DES chips housed in six cabinets. The chips worked in parallel, each searching through an assigned segment of the key space. When RSA data security posted the next challenge in July 1998, DeepCrack solved it in 56 hours and easily won the ten thousand dollar prize: not quite enough to cover the cost of the machine, but more than enough to make an important point about DES.

The final challenge was posted in January 1999. It was solved within 22 hours using a combined DeepCrack and *distributed.net* effort. This put the final nail in DES's coffin showing that a 56-bit secret key can be recovered in just a few hours.

To complete the story, in 2007 the COPACOBANA team built a cluster of 120 off the shelf FPGA boards at a total cost of about ten thousand US dollars. The cluster can search through the entire 2^{56} DES key space in about 12.8 days [81].

The conclusion from all this work is that a 56-bit key is way too short. The minimum safe key size these days is 128 bits.

4.2.2.1 Is AES-128 vulnerable to exhaustive search?

Let us extrapolate the DES results to AES. While these estimates are inherently imprecise, they give some indication as to the complexity of exhaustive search on AES. The minimum AES key space size is 2^{128}. If scanning a space of size 2^{56} takes 22 hours then scanning a space of size 2^{128} will take time:

$$(22 \text{ hours}) \times 2^{128-56} \approx 1.18 \cdot 10^{20} \text{ years}.$$

Even allowing for a billion fold improvement in computing speed and computing resources and accounting for the fact that evaluating AES is faster than evaluating DES, the required time far exceeds our capabilities. It is fair to conclude that a brute-force exhaustive search attack on AES will never be practical. However, more sophisticated brute-force attacks on AES-128 exploiting time-space tradeoffs may come within reach, as discussed in Section 18.7.

4.2.3 Strengthening ciphers against exhaustive search: the $3\mathcal{E}$ construction

The DES cipher has proved to be remarkably resilient to sophisticated attacks. Despite many years of analysis the most practical attack on DES is a brute force exhaustive search over the entire key space. Unfortunately, the 56-bit key space is too small.

A natural question is whether we can strengthen the cipher against exhaustive search without changing its inner structure. The simplest solution is to iterate the cipher several time using independent keys.

Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$. We define the block cipher $3\mathcal{E} = (E_3, D_3)$ as

$$E_3(\ (k_1, k_2, k_3),\ x) := E\big(k_3,\ E(k_2,\ E(k_1, x))\big)$$

The $3\mathcal{E}$ block cipher takes keys in $\mathcal{K}^3$. For DES the $3\mathcal{E}$ block cipher, called **Triple-DES**, uses keys whose length is $3 \times 56 = 168$ bits.

Security. To analyze the security of $3\mathcal{E}$ we will need a framework called the *ideal cipher model* which we present at the end of this chapter. We analyze the security of $3\mathcal{E}$ in that section.

The Triple-DES standard. NIST approved Triple-DES for government use through the year 2030. Strictly speaking, the NIST version of Triple-DES is defined as

$$E_3(\ (k_1, k_2, k_3),\ x) := E\big(k_3,\ D(k_2,\ E(k_1, x))\big).$$

The reason for this is that setting $k_1 = k_2 = k_3$ reduces the NIST Triple-DES to ordinary DES and hence Triple-DES hardware can be used to implement single DES. This will not affect our discussion of security of Triple-DES. Another variant of Triple-DES is discussed in Exercise 4.5.

4.2.3.1 The $2\mathcal{E}$ construction is insecure

While Triple-DES is not vulnerable to exhaustive search, its performance is three times slower than single DES, as shown in Table 4.1.

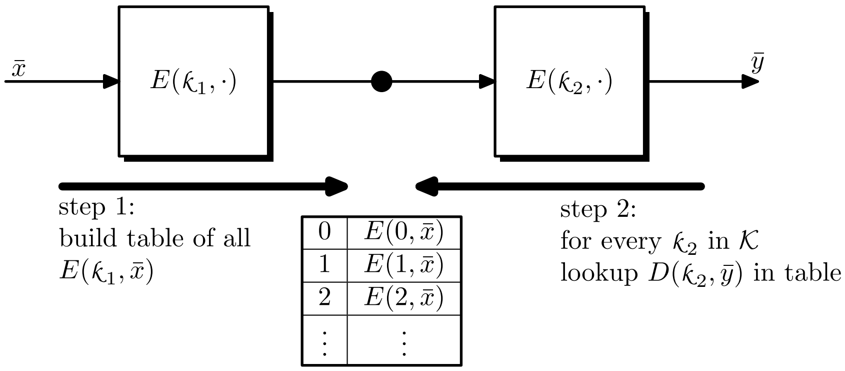
Why not use Double-DES? Its key size is $2 \times 56 = 112$ bits, which is already sufficient to defeat exhaustive search. Its performance is much better than Triple-DES.

Unfortunately, Double-DES is no more secure than single DES. More generally, let $\mathcal{E} = (E, D)$ be a block cipher with key space $\mathcal{K}$. We show that the $2\mathcal{E} = (E_2, D_2)$ construction, defined as

$$E_2(\ (k_1, k_2),\ x) := E\big(k_2,\ E(k_1, x)\big)$$

is no more secure than $\mathcal{E}$. The attack strategy is called **meet in the middle**.

We are given Q plaintext blocks $x_1, \ldots, x_Q$ and their $2\mathcal{E}$ encryptions $y_i = E_2\big(\ (k_1, k_2),\ x_i\big)$ for $i = 1, \ldots, Q$. We show how to recover the secret key (k_1, k_2) in time proportional to $|\mathcal{K}|$, even though the key space has size $|\mathcal{K}|^2$. As with exhaustive search, a small number of plaintext/ciphertext pairs is sufficient to ensure that there is a unique key (k_1, k_2) with high probability. Ten pairs are more than enough to ensure uniqueness for block ciphers like Double-DES.


$\bar{x}$
$E(\mathscr{k}_1, \cdot)$
$E(\mathscr{k}_2, \cdot)$
$\bar{y}$
step 1:
build table of all
$E(\mathscr{k}_1, \bar{x})$
step 2:
for every $\mathscr{k}_2$ in $\mathcal{K}$
lookup $D(\mathscr{k}_2, \bar{y})$ in table

0	$E(0, \bar{x})$
1	$E(1, \bar{x})$
2	$E(2, \bar{x})$
$\vdots$	$\vdots$

Figure 4.10: Meet in the middle attack on $2\mathcal{E}$

Theorem 4.2. *Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$. There is an algorithm $\mathcal{A}_{\text{EX}}$ that takes as input Q plaintext/ciphertext pairs $(x_i, y_i) \in \mathcal{X}^2$ for $i = 1, \dots, Q$ and outputs a key pair $(k_1, k_2) \in \mathcal{K}^2$ such that*

$$y_i = E_2\big(\ (k_1, k_2),\ x_i\big) \quad \textit{for all } i = 1, \dots, Q. \tag{4.9}$$

Its running time is dominated by a total of $2Q \cdot |\mathcal{K}|$ evaluations of algorithms E and D.

Proof. Let $\bar{x} := (x_1, \dots, x_Q)$ and $\bar{y} := (y_1, \dots, y_Q)$. We can capture the Q relations in (4.9) by writing

$$\bar{y} = E_2\big((k_1, k_2), \bar{x}\big) = E(k_2,\ E(k_1, \bar{x})).$$

This is equivalent to

$$D(k_2, \bar{y}) = E(k_1, \bar{x}). \tag{4.10}$$

To find a pair (k_1, k_2) satisfying (4.10) the algorithm $\mathcal{A}_{\text{EX}}$ does the following:

```
step 1:   construct a table $T$ containing all pairs $\big(\mathscr{k}_1,\ E(\mathscr{k}_1, \bar{x})\ \big)$ for all $\mathscr{k}_1 \in \mathcal{K}$
step 2:   for all $\mathscr{k}_2 \in \mathcal{K}$ do:
              $\bar{z} \leftarrow D(\mathscr{k}_2, \bar{y})$
              table lookup: if $T$ contains a pair $(\cdot,\ \bar{z})$ then
                  let $(\mathscr{k}_1, \bar{z})$ be that pair, output $(\mathscr{k}_1, \mathscr{k}_2)$ and halt
```

This meet in the middle attack is depicted in Fig. 4.10. By construction, the pair $(\mathscr{k}_1, \mathscr{k}_2)$ output by the algorithm must satisfy $D(\mathscr{k}_2, \bar{y}) = E(\mathscr{k}_1, \bar{x})$, as required.

Step 1 requires $Q \cdot |\mathcal{K}|$ evaluations of E. Step 2 similarly requires $Q \cdot |\mathcal{K}|$ evaluations of D. Therefore, the total number of evaluation of E and D is $2Q \cdot |\mathcal{K}|$. We assume that the time to insert and look-up elements in the data structure holding the table T is less than the time to evaluate algorithms E and D. □

As discussed above, for relatively small values of Q, with overwhelming probability there will be only one key pair satisfying (4.9), and this will be the output of Algorithm $\mathcal{A}_{\text{EX}}$ in Theorem 4.2.

The running time of algorithm $\mathcal{A}$ in Theorem 4.2 is about the same as the time to do exhaustive search on $\mathcal{E}$, suggesting that $2\mathcal{E}$ does not strengthen $\mathcal{E}$ against exhaustive search. The theorem, however, only considers the running time of $\mathcal{A}$. Notice that $\mathcal{A}$ must keep a large table in memory which can be difficult. To attack Double-DES, $\mathcal{A}$ would need to store a table of size 2^{56} where each table entry contains a DES key and a short ciphertext. Overall this amounts to at least 2^{60} bytes, which is about a million Terrabytes. While not impossible, obtaining sufficient storage can be difficult. Alternatively an attacker can trade-off storage space for running time — it is easy to modify $\mathcal{A}$ so that at any given time it only stores an ϵ fraction of the table at the cost of increasing the running time by a factor of $1/\epsilon$.

A meet in the middle attack on Triple-DES. A similar meet in the middle attack applies to the $3\mathcal{E}$ construction from the previous section. While $3\mathcal{E}$ has key space $\mathcal{K}^3$, the meet in the middle attack on $3\mathcal{E}$ runs in time about $|\mathcal{K}|^2$ and takes space $|\mathcal{K}|$. In the case of Triple-DES, the attack requires about $|\mathcal{K}|^2 = 2^{112}$ evaluations of DES which is too long to run in practice. Hence, Triple-DES resists this meet in the middle attack and is the reason why Triple-DES is used in practice.

4.2.4 Case study: AES

Although Triple-DES is a NIST approved cipher, it has a number of significant drawbacks. First, Triple-DES is three times slower than DES and performs poorly when implemented in software. Second, the 64-bit block size is problematic for a number of important applications, such as those discussed in Chapter 6. By the mid-1990s it became apparent that a new federal block cipher standard was needed.

The AES process. In 1997 NIST put out a request for proposals for a new block cipher standard to be called the **Advanced Encryption Standard** or AES. The AES block cipher had to operate on 128-bit blocks and support three key sizes: 128, 192, and 256 bits. In September of 1997, NIST received 15 submissions, many of which were developed outside of the United States. After holding two open conferences to discuss the proposals, in 1999 NIST narrowed down the list to five candidates. A further round of intense cryptanalysis followed, culminating in the AES3 conference in April of 2000, at which a representative of each of the final five teams made a presentation arguing why their submission should be chosen as the standard. In October of 2000, NIST announced that **Rijndael**, a Belgian block cipher, had been selected as the AES cipher. AES became an official standard in November of 2001 when it was published as a NIST standard in FIPS 197. This concluded a five year process to standardize a replacement to DES.

Rijndael was designed by Belgian cryptographers Joan Daemen and Vincent Rijmen [49]. AES is slightly different from the original Rijndael cipher. For example, Rijndael supports blocks of size 128, 192, or 256 bits while AES only supports 128-bit blocks.

4.2.4.1 The AES algorithm

Like many real-world block ciphers, AES is an iterated cipher that iterates a simple round cipher several times. The number of iterations depends on the size of the secret key:

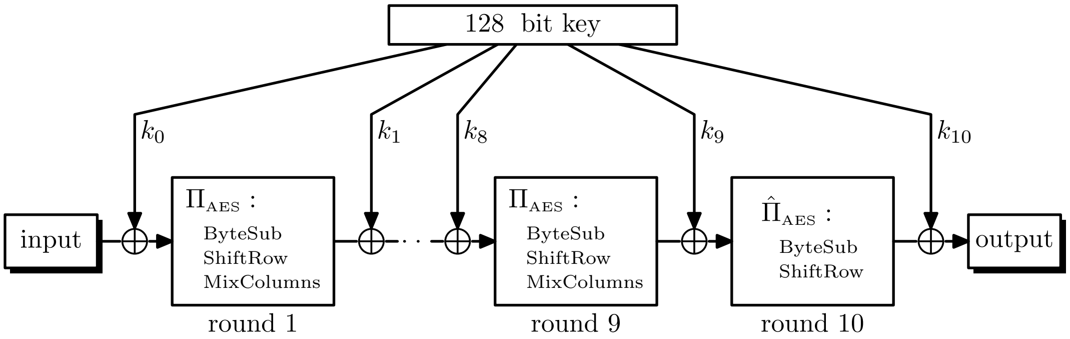


Figure 4.11: Schematic of the AES-128 block cipher

cipher name	key-size (bits)	block-size (bits)	number of rounds
AES-128	128	128	10
AES-192	192	128	12
AES-256	256	128	14

For example, the structure of the cipher AES-128 with its ten rounds is shown in Fig. 4.11. Here Π_{AES} is a fixed permutation (a one-to-one function) on $\{0,1\}^{128}$ that does not depend on the key. The last step of each round is to XOR the current round key with the output of Π_{AES}. This is repeated 9 times until in the last round a slightly modified permutation $\hat{\Pi}_{\text{AES}}$ is used. Inverting the AES algorithm is done by running the entire structure in the reverse direction. This is possible because every step is easily invertible.

Ciphers that follow the structure shown in Fig. 4.11 are called **alternating key ciphers**. They are also known as **iterated Even-Mansour ciphers**. They can be proven secure under certain "ideal" assumptions about the permutation Π_{AES} in each round. We present this analysis in Theorem 4.14 later in this chapter.

To complete the description of AES it suffices to describe the permutation Π_{AES}, and the AES key expansion PRG. We describe each in turn.

The AES round permutation. The permutation Π_{AES} is made up of a sequence of three invertible operations on the set $\{0,1\}^{128}$. The 128 bits are organized as a 4×4 array of cells, where each cell is made up of eight bits. The following three invertible operations are then carried out in sequence, one after the other, on this 4×4 array:

1. `SubBytes`: Let $S : \{0,1\}^8 \to \{0,1\}^8$ be a fixed permutation (a one-to-one function). This permutation is applied to each of the 16 cells, one cell at a time. The permutation S is specified in the AES standard as a hard-coded table of 256 entries. It is designed to have no fixed points, namely $S(x) \neq x$ for all $x \in \{0,1\}^8$, and no inverse fixed points, namely $S(x) \neq \bar{x}$ where $\bar{x}$ is the bit-wise complement of x. These requirements are needed to defeat certain attacks discussed in Section 4.3.1.

2. `ShiftRows`: This step performs a cyclic shift on the four rows of the input 4×4 array: the first row is unchanged, the second row is cyclically shifted one byte to the left, the third row is

cyclically shifted two bytes, and the fourth row is cyclically shifted three bytes. In a diagram, this step performs the following transformation:

$$\begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \end{pmatrix} \Longrightarrow \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_5 & a_6 & a_7 & a_4 \\ a_{10} & a_{11} & a_8 & a_9 \\ a_{15} & a_{12} & a_{13} & a_{14} \end{pmatrix} \tag{4.11}$$

3. `MixColumns`: In this step the 4×4 array is treated as a matrix and this matrix is multiplied by a fixed matrix where arithmetic is interpreted in the finite field $\mathrm{GF}(2^8)$. Elements in the field $\mathrm{GF}(2^8)$ are represented as polynomials over $\mathrm{GF}(2)$ of degree less than eight where multiplication is done modulo the irreducible polynomial $x^8 + x^4 + x^3 + x + 1$. Specifically, the `MixColumns` transformation does:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_5 & a_6 & a_7 & a_4 \\ a_{10} & a_{11} & a_8 & a_9 \\ a_{15} & a_{12} & a_{13} & a_{14} \end{pmatrix} \Longrightarrow \begin{pmatrix} a'_0 & a'_1 & a'_2 & a'_3 \\ a'_4 & a'_5 & a'_6 & a'_7 \\ a'_8 & a'_9 & a'_{10} & a'_{11} \\ a'_{12} & a'_{13} & a'_{14} & a'_{15} \end{pmatrix} \tag{4.12}$$

   Here the scalars $01, 02, 03$ are interpreted as elements of $\mathrm{GF}(2^8)$ using their binary representation (e.g., 03 represents the element $x + 1$ in $\mathrm{GF}(2^8)$). This fixed matrix is invertible over $\mathrm{GF}(2^8)$ so that the entire transformation is invertible.

The permutation Π_{AES} used in the AES circuit of Fig. 4.11 is the sequential composition of the three permutation `SubBytes`, `ShiftRows`, and `MixColumns` in that order. In the very last round AES uses a slightly different function we call $\hat{\Pi}_{\text{AES}}$. This function is the same as Π_{AES} except that the MixColumns step is omitted. This omission is done so that the AES decryption circuit looks somewhat similar to the AES encryption circuit. Security implications of this omission are discussed in [57].

Because each step in Π_{AES} is easily invertible, the entire permutation Π_{AES} is easily invertible, as required for decryption.

Implementing AES using pre-computed tables. The AES round function is built from a permutation we called Π_{AES} defined as a sequence of three steps: `SubBytes`, `ShiftRows`, and `MixColumns`. The designers of AES did not intend for AES to be implemented that way on modern processors. Instead, they proposed an implementation of Π_{AES} the does all three steps at once using four fixed lookup tables called T_0, T_1, T_2, T_3.

To explain how this works, recall that Π_{AES} takes as input a 4×4 matrix $A = (a_i)_{i=0,\ldots,15}$ and outputs a matrix $A' := \Pi_{\text{AES}}(A)$ of the same dimensions. Let us use $S[a]$ to denote the result of applying `SubBytes` to an input $a \in \{0,1\}^8$. Similarly, recall that the `MixColumns` step multiplies the current state by a fixed 4×4 matrix M. Let us use $M[i]$ to denote column number i of M, and $A'[i]$ to denote column number i of A'.

Now, looking at (4.12), we can write the four columns of the output of $\Pi_{\text{AES}}(A)$ as:

$$\begin{aligned} A'[0] &= M[0] \cdot S[a_0] + M[1] \cdot S[a_5] + M[2] \cdot S[a_{10}] + M[3] \cdot S[a_{15}] \\ A'[1] &= M[0] \cdot S[a_1] + M[1] \cdot S[a_6] + M[2] \cdot S[a_{11}] + M[3] \cdot S[a_{12}] \\ A'[2] &= M[0] \cdot S[a_2] + M[1] \cdot S[a_7] + M[2] \cdot S[a_8] + M[3] \cdot S[a_{13}] \\ A'[3] &= M[0] \cdot S[a_3] + M[1] \cdot S[a_4] + M[2] \cdot S[a_9] + M[3] \cdot S[a_{14}] \end{aligned} \tag{4.13}$$

where addition and multiplication is done in GF(2^8). Each column $M[i],\ i = 0, 1, 2, 3$, is a vector of four bytes over GF(2^8), while the quantities $S[a_i]$ are 1-byte scalars in GF(2^8).

Every term in (4.13) can be evaluated quickly using a fixed pre-computed table. For $i = 0, 1, 2, 3$ let us define a table T_i with 256 entries as follows:

$$\text{for } a \in \{0,1\}^8\text{:} \qquad T_i[a] := M[i] \cdot S[a] \quad \in \{0,1\}^{32} \ .$$

Plugging these tables into (4.13) gives a fast way to evaluate $\Pi_{\text{AES}}(A)$:

$$\begin{aligned}
A'[0] &= T_0[a_0] + T_1[a_5] + T_2[a_{10}] + T_3[a_{15}] \\
A'[1] &= T_0[a_1] + T_1[a_6] + T_2[a_{11}] + T_3[a_{12}] \\
A'[2] &= T_0[a_2] + T_1[a_7] + T_2[a_8] + T_3[a_{13}] \\
A'[3] &= T_0[a_3] + T_1[a_4] + T_2[a_9] + T_3[a_{14}]
\end{aligned}$$

The entire AES circuit written this way is a simple sequence of table lookups. Since each table T_i contains 256 entries, four bytes each, the total size of all four tables is 4KB. The circular structure of the matrix M makes it possible to compress the four tables to only 2KB with little impact on performance.

The one exception to (4.13) is the very last round of AES where the `MixColumns` step is omitted. To evaluate the last round we need a fifth 256-byte table S that only implements the `SubBytes` operation.

This optimization of AES is optional. Implementations in constrained environments where there is no room to store a 4KB table can choose to implement the three steps of Π_{AES} in code, which takes less than 4KB, but is not as fast. Thus AES can be adapted for both constrained and unconstrained environments.

As a word of caution, we note that a simplistic implementation of AES using this table lookup optimization is most likely vulnerable to cache timing attacks discussed in Section 4.3.2.

The AES-128 key expansion method. Looking back at Fig. 4.11 we see that key expansion for AES-128 needs to generate 11 rounds keys $k_0, \ldots, k_{10}$ where each round key is 128 bits. To do so, the 128-bit AES key is partitioned into four 32-bit words $w_{0,0}, w_{0,1}, w_{0,2}, w_{0,3}$ and these form the first round key k_0. The remaining ten round keys are generated sequentially: for $i = 1, \ldots, 10$, the 128-bit round key $k_i = (w_{i,0}, w_{i,1}, w_{i,2}, w_{i,3})$ is generated from the preceding round key $k_{i-1} = (w_{i-1,0}, w_{i-1,1}, w_{i-1,2}, w_{i-1,3})$ as follows:

$$\begin{aligned}
& w_{i,0} \leftarrow w_{i-1,0} \oplus g_i(w_{i-1,3}) \\
& w_{i,1} \leftarrow w_{i-1,1} \oplus w_{i,0} \\
& w_{i,2} \leftarrow w_{i-1,2} \oplus w_{i,1} \\
& w_{i,3} \leftarrow w_{i-1,3} \oplus w_{i,2} \quad .
\end{aligned}$$

Here the function $g_i : \{0,1\}^{32} \to \{0,1\}^{32}$ is a fixed function specified in the AES standard. It operates on its four byte input in three steps: (1) perform a one-byte left circular rotation on the 4-byte input, (2) apply `SubBytes` to each of the four bytes obtained, and (3) XOR the left most byte with a fixed round constant c_i. The round constants $c_1, \ldots, c_{10}$ are specified in the AES standard: round constant number i is the element x^{i-1} of the field GF(2^8) treated as an 8-bit string.

The key expansion procedures for AES-192 and AES-256 are similar to those of AES-128. For AES-192 each iteration generates six 32-bit words (192 bits total) in a similar manner to AES-128, but only the first four 32-bit words (128 bits total) are used as the AES round key. For AES-256 each iteration generates eight 32-bit words (256 bits total) in a similar manner to AES-128, but only the first four 32-bit words (128 bits total) are used as the AES round key.

The AES key expansion method is intentionally designed to be invertible: given the last round key, one can work backwards to recover the full AES secret key k. The reason for this is to ensure that every AES-128 round key, on its own, has the same amount of entropy as the AES-128 secret key k. If AES-128 key expansion were not invertible then the last round key would not be uniform in $\{0,1\}^{128}$. Unfortunately, invertability also aids attacks: it is used in related key attacks and in side-channel attacks on AES, discussed next.

Security of AES. The AES algorithm withstood fairly sophisticated attempts at cryptanalysis lobbed at it. At the time of this writing, the best known attacks are as follows:

- **Key recovery:** Key recovery attacks refer to an adversary who is given multiple plaintext/ciphertext pairs and is able to recover the secret key from these pairs, as in an exhaustive search attack. The best known key recovery attack on AES-128 takes $2^{126.1}$ evaluations of AES [26]. This is about four times faster than exhaustive search and takes a prohibitively long time. Therefore this attack has little impact on the security of AES-128.

  The best known attack on AES-192 takes $2^{189.74}$ evaluation of AES, which is again only about four times faster than exhaustive search. The best known attack on AES-256 takes $2^{254.42}$ evaluation of AES, which is about three times faster than exhaustive search. These attacks have little impact on the security of AES-192 or AES-256.

- **Related key attacks:** In an ℓ-way related key attack, the adversary is given ℓ lists of plaintext/ciphertext pairs: for $i = 1, \ldots, \ell$, list number i is generated using key k_i. The point is that all ℓ keys $k_1, \ldots, k_\ell$ must satisfy some fixed relation chosen by the adversary. The attacker's goal is to recover one of the keys, say k_1. In well-implemented cryptosystems, keys are always generated independently at random and are unlikely to satisfy the required relation. Therefore related key attacks do not typically affect correct crypto implementations.

  AES-256 is vulnerable to a related key attack that exploits its relatively simple key expansion mechanism [21]. The attack requires four related keys k_1, k_2, k_3, k_4 where the relation is a simple XOR relation: it requires that certain bits of the quantities $k_1 \oplus k_2$, $k_1 \oplus k_3$, and $k_2 \oplus k_4$ are set to specific values. Then, given lists of plaintext/ciphertext pairs generated for each of the four keys, the attacker can recover the four keys in time $2^{99.5}$. This is far faster than the time it would take to mount an exhaustive search on AES-256. While the attack is quite interesting, it does not affect the security of AES-256 in well-implemented systems.

Hardware implementation of AES. At the time AES was standardized as a federal encryption standard, most implementations were software based. The wide-spread adoption of AES in software products prompted all major processor vendors to extend their instruction set to add support for a hardware implementation of AES.

Intel, for example, added new instructions to its Xeon and Core families of processors called **AES-NI** (AES new instructions) that speed-up and simplify the process of using AES in software. The new instructions work as follows:

- `AESKEYGENASSIST`: runs the key expansion procedure to generate the AES round keys from the AES key.

- `AESENC`: runs one round of the AES encryption algorithm. The instruction is called as:

  ```
  AESENC xmm15, xmm1
  ```

  where the `xmm15` register holds the 128-bit data block and the `xmm1` register holds the 128-bit round key for that round. The resulting 128-bit data block is written to register `xmm15`. Running this instruction nine times with the appropriate round keys loaded into registers `xmm1`, ..., `xmm9` executes the first nine rounds of AES encryption.

- `AESENCLAST`: invoked similar to `AESENC` to run last round of the AES algorithm. Recall that the last round function is different from the others: it omits the MixColumns step.

- `AESDEC` and `AESDECLAST`: runs one round of the AES decryption algorithm, analogous to the encryption instructions.

These AES-NI hardware instructions provide a significant speed-up over a heavily optimized software implementation of AES. Experiments by Emilia Käsper in 2009 show that on an Intel Core 2 processor, AES using the AES-NI instructions takes 1.35 cycles/byte (pipelined), while an optimized software implementation takes 7.59 cycles/byte.

In Intel's Skylake processors introduced in 2015 the `AESENC`, `AESDEC` and `AESENCLAST` instructions each take four cycles to complete. These instructions are fully pipelined so that a new instruction can be dispatched every cycle. In other words, Intel partitioned the execution of `AESENC` into a pipeline of four stages. Four AES blocks can be processed concurrently by different stages of the pipeline. While processing a single AES-128 block takes (4 cycles) $\times$ (10 rounds) $=$ 40 cycles (or 2.5 cycles/byte), processing four blocks in a pipeline takes only 44 cycles (or 0.69 cycles/byte). Hence, pipelining can speed up AES by almost a factor of four. As we will see in the next chapter, this plays an important role in choosing the exact method we use to encrypt long messages: it is best to choose an encryption method that can leverage the available parallelism to keep the pipeline busy.

Beyond speed, the hardware implementation of AES offers better security because it is resistant to the side-channel attacks discussed in the next section.

4.3 Sophisticated attacks on block ciphers

Widely deployed block ciphers like AES go through a lengthy selection process before they are standardized and continue to be subjected to cryptanalysis. In this section we survey some attack techniques that have been developed over the years.

In Section 4.3.1, we begin with attacks on the design of the cipher that may result in key compromise from observing plaintext/ciphertext pairs. Unlike brute-force exhaustive search attacks, these **algorithmic attacks** rely on clever analysis of the internal structure of a particular block cipher.

In Section 4.3.2, we consider a very different class of attacks, called **side-channel attacks**. In analyzing any cryptosystem, we consider scenarios in which an adversary interacts with the users of a cryptosystem. During the course of these interactions, the adversary collects information that may help it break the system. Throughout this book, we generally assume that this information is limited to the input/output behavior of the users (for example, plaintext/ciphertext pairs). However, this assumption ignores the fact that **computation is a physical process**. As we shall see, in some scenarios it is possible for the adversary to break a cryptosystem by measuring physical characteristics of the users' computations, for example, running time or power consumption.

Another class of attacks on the physical implementation of a cryptosystem is a **fault injection attack**, which is discussed in Section 4.3.3. Finally, in Section 4.3.4, we consider another class of algorithmic attacks, in which the adversary can harness the laws of **quantum mechanics** to speed up its computations.

These clever attacks make two very important points:

1. Casual users of cryptography should only ever use standardized algorithms like AES, and not design their own block ciphers.

2. It is best to not implement algorithms on your own since, most likely the resulting implementations will be vulnerable to side-channel attacks; instead, it is better to use vetted implementations in widely used crypto libraries.

To further emphasize these points we encourage anyone who first learns about the inner-workings of AES to take the following entertaining pledge (originally due to Jeff Moser):

> I promise that once I see how simple AES really is, I will not implement it in production code even though it will be really fun. This agreement will remain in effect until I learn all about side-channel attacks and countermeasures to the point where I lose all interest in implementing AES myself.

4.3.1 Algorithmic attacks

Attacking the design of block ciphers is a vast field with many sophisticated techniques: linear cryptanalysis, differential cryptanalysis, slide attacks, boomerang attacks, and many others. We refer to [150] for a survey of the many elegant ideas that have been developed. Here we briefly describe a technique called *linear cryptanalysis* that has been used successfully against the DES block cipher. This technique, due to Matsui [109, 108], illustrates why designing efficient block-ciphers is so challenging. This method has been shown to not work against AES.

Linear cryptanalysis. Let (E, D) be a block cipher where data blocks and keys are bit strings. That is, $\mathcal{M} = \mathcal{C} = \{0,1\}^n$ and $\mathcal{K} = \{0,1\}^h$.

For a bit string $m \in \{0,1\}^n$ and a set of bit positions $S \subseteq \{0, \dots, n-1\}$ we use $m[S]$ to denote the XOR of the bits in positions in S. That is, if $S = \{i_1, \dots, i_\ell\}$ then $m[S] := m[i_1] \oplus \cdots \oplus m[i_\ell]$.

We say that the block cipher (E, D) has a **linear relation** if there exist sets of bit positions $S_0, S_1 \subseteq \{0, \dots, n-1\}$ and $S_2 \subseteq \{0, \dots, h-1\}$, such that *for all* keys $k \in \mathcal{K}$ and *for randomly chosen* $m \in \mathcal{M}$, we have

$$\Pr\Big[\ m[S_0] \oplus E(k,m)[S_1] = k[S_2]\ \Big] \geq \frac{1}{2} + \epsilon \tag{4.14}$$

for some non-negligible ϵ called the **bias**. For an "ideal" cipher the plaintext and ciphertext behave like independent strings so that the relation $m[S_0] \oplus E(k,m)[S_1] = k[S_2]$ in (4.14) holds with probability exactly 1/2, and therefore $\epsilon = 0$. Surprisingly, the DES block cipher has a linear relation with a small, but non-negligible bias.

Let us see how a linear relation leads to an attack. Consider a cipher (E, D) that has a linear relation as in (4.14) for some non-negligible $\epsilon > 0$. We assume the linear relation is explicit so that the attacker knows the sets S_0, S_1 and S_2 used in the relation. Suppose that for some unknown secret key $k \in \mathcal{K}$ the attacker obtains many plaintext/ciphertext pairs (m_i, c_i) for $i = 1, \ldots, t$. We assume that the messages $m_1, \ldots, m_t$ are sampled uniformly and independently from $\mathcal{M}$ and that $c_i = E(k, m_i)$ for $i = 1, \ldots, t$. Using this information the attacker can learn one bit of information about the secret key k, namely the bit $k[S_2] \in \{0, 1\}$ assuming sufficiently many plaintext/ciphertext pairs are given. The following lemma shows how.

Lemma 4.3. *Let (E, D) be a block cipher for which (4.14) holds. Let $m_1, \ldots, m_t$ be messages sampled uniformly and independently from the message space $\mathcal{M}$ and let $c_i := E(k, m_i)$ for $i = 1, \ldots, t$. Then*

$$\Pr\Big[\ k[S_2] = \text{Majority}_{i=1}^{t}(m_i[S_0] \oplus c_i[S_1])\ \Big] \geq 1 - e^{-t\epsilon^2/2}\ . \tag{4.15}$$

Here, Majority takes a majority vote on the given bits; for example, on input $(0, 0, 1)$, the majority is 0, and on input $(0, 1, 1)$, the majority is 1. The proof of the lemma is by a direct application of the classic Chernoff bound.

The bound in (4.15) shows that once the number of known plaintext/ciphertext pairs exceeds $4/\epsilon^2$, the output of the majority function equals $k[S_2]$ with more than 86% probability. Hence, the attacker can compute $k[S_2]$ from the given plaintext/ciphertext pairs and obtain one bit of information about the secret key. While this single key bit may not seem like much, it is a stepping stone towards a more powerful attack that can expose the entire key.

Linear cryptanalysis of DES. Matsui showed that 14-rounds of the DES block cipher has a linear relation where the bias is at least $\epsilon \geq 2^{-21}$. In fact, two linear relations are obtained: one by exploiting linearity in the DES encryption circuit and another from linearity in the DES decryption circuit. For a 64-bit plaintext m let m_{L} and m_{R} be the left and right 32-bits of m respectively. Similarly, for a 64-bit ciphertext c let c_{L} and c_{R} be the left and right 32-bits of c respectively. Then two linear relations for 14-rounds of DES are:

$$\begin{gathered} m_{\text{R}}[17, 18, 24] \oplus c_{\text{L}}[7, 18, 24, 29] \oplus c_{\text{R}}[15] = k[S_{\text{e}}] \\ c_{\text{R}}[17, 18, 24] \oplus m_{\text{L}}[7, 18, 24, 29] \oplus m_{\text{R}}[15] = k[S_{\text{d}}] \end{gathered} \tag{4.16}$$

for some bit positions $S_{\text{e}}, S_{\text{d}} \subseteq \{0, \ldots, 55\}$ in the 56-bit key k. Both relations have a bias of $\epsilon \geq 2^{-21}$ when applied to 14-rounds of DES.

These relations are extended to the entire 16-round DES by incorporating the first and last rounds of DES — rounds number 1 and 16 — into the relations. Let k_1 be the first round key and let k_{16} be the last round key. Then by definition of the DES round function we obtain from (4.16) the following relations on the entire 16-round DES circuit:

$$\Big(m_{\text{L}} \oplus F(k_1, m_{\text{R}})\Big)[17, 18, 24] \oplus c_{\text{R}}[7, 18, 24, 29] \oplus \Big(c_{\text{L}} \oplus F(k_{16}, c_{\text{R}})\Big)[15] = k[S'_{\text{e}}] \tag{4.17}$$

$$\Big(c_{\text{L}} \oplus F(k_{16}, c_{\text{R}})\Big)[17, 18, 24] \oplus m_{\text{R}}[7, 18, 24, 29] \oplus \Big(m_{\text{L}} \oplus F(k_1, m_{\text{R}})\Big)[15] = k[S'_{\text{d}}] \tag{4.18}$$

for appropriate bit positions $S'_{\text{e}}, S'_{\text{d}} \subseteq \{0, \ldots, 55\}$ in the 56-bit key.

Let us first focus on relation (4.17). Bits 17,18,24 of $F(k_1, m_{\text{R}})$ are the result of a single S-box and therefore they depend on only six bits of k_1. Similarly $F(k_{16}, c_{\text{R}})[15]$ depends on six bits of k_{16}.

Hence, the left hand side of (4.17) depends on only 12 bits of the secret key k. Let us denote these 12 bits by $k^{(12)}$. We know that when the 12 bits are set to their correct value, the left hand side of (4.17), evaluated at a random plaintext/ciphertext pair, exhibits a bias of about 2^{-21} towards the bit $k[S'_{\mathrm{e}}]$. When the 12 key bits of the key are set incorrectly one assumes that the bias in (4.17) is far less. As we will see, this has been verified experimentally.

This observation lets an attacker recover the 12 bits $k^{(12)}$ of the secret key k as follows. Given a list L of t plaintext/ciphertext pairs (e.g., $t = 2^{43}$) do:

- Step 1: for each of the 2^{12} candidates for the key bits $k^{(12)}$ compute the bias in (4.17). That is, evaluate the left hand side of (4.17) on all t plaintext/ciphertext pairs in L and let t_0 be the number of times that the expression evaluates to 0. The bias is computed as $\epsilon = |(t_0/t) - (1/2)|$. This produces a vector of 2^{12} biases, one for each candidate 12 bits for $k^{(12)}$.

- Step 2: sort the 2^{12} candidates by their bias, from largest to smallest. If the list L of given plaintext/ciphertext pairs is sufficiently large then the 12-bit candidate producing the highest bias is the most likely to be equal to $k^{(12)}$. This recovers 12 bits of the key. Once $k^{(12)}$ is known we can determine the bit $k[S'_{\mathrm{e}}]$ using Lemma 4.3, giving a total of 13 bits of k.

The relation (4.18) can be used to recover an additional 13 bits of the key k in exactly the same way. This gives the attacker a total 26 bits of the key. The remaining $56 - 26 = 30$ bits are recovered by exhaustive search.

Naively computing the biases in Step 1 takes time $2^{12} \times t$: for each candidate for $k^{(12)}$ one has to evaluate (4.17) on all t plaintext/ciphertext pairs in L. The following insight reduces the work to approximately time t. For a given pair (m, c), the left hand side of (4.17) can be computed from only thirteen bits of (m, c): six bits of m are needed to compute $F(k_1, m_{\mathrm{R}})[17, 18, 24]$, six bits of c are needed to compute $F(k_{16}, c_{\mathrm{R}})[15]$, and finally the single bit $m_{\mathrm{L}}[17, 18, 24] \oplus c_{\mathrm{R}}[7, 18, 24, 29] \oplus c_{\mathrm{L}}[15]$ is needed. These 13 bits are sufficient to evaluate the left hand side of (4.17) for any candidate key. Two plaintext/ciphertext pairs that agree on these 13 bits will always result in the same value for (4.17). We refer to these 13 bits as the *type* of the plaintext/ciphertext pair.

Before computing the biases in Step 1 we build a table of size 2^{13} that counts the number of plaintext/ciphertext pairs in L of each type. For $b \in \{0, 1\}^{13}$ table entry b is the number of plaintext/ciphertext pairs of type b. Constructing this table takes time t, but once the table is constructed computing all the biases in Step 1 can be done in time $2^{12} \times 2^{13} = 2^{25}$ which is much less than t. Therefore, the bulk of the work in Step 1 is counting the number of plaintext/ciphertext pairs of each type.

Matsui shows that given a list of 2^{43} plaintext/ciphertext pairs this attack succeeds with probability 85% using about 2^{43} evaluations of the DES circuit. Experimental results by Junod [97] show that with 2^{43} plaintext/ciphertext pairs, the correct 26 bits of the key are among the 2700 most likely candidates from Step 1 on average. In other words, the exhaustive search for the remaining 30 bits is carried out on average $2700 \approx 2^{11.4}$ times to recover the entire 56-bit key. Overall, the attack is dominated by the time to evaluate the DES circuit $2^{30} \times 2^{11.4} = 2^{41.4}$ times on average [97].

Lesson. Linear cryptanalysis of DES is possible because the fifth S-box, S_5, happens to be somewhat approximated by a linear function. The linearity of S_5 introduced a linear relation on the cipher that could be exploited to recover the secret key using 2^{41} DES evaluations, far less than the

2^{56} evaluations that would be needed in an exhaustive search. However, unlike exhaustive search, this attack requires a large number of plaintext/ciphertext pairs: the required 2^{43} pairs correspond to 64 *terabytes* of plaintext data. Nevertheless, this is a good illustration of how difficult it is to design secure block ciphers and why one should only use standardized and well-studied ciphers.

Linear cryptanalysis has been generalized over the years to allow for more complex non-linear relations among plaintext, ciphertext, and key bits. These generalizations have been used against other block ciphers such as LOKI91 and Q.

4.3.2 Side-channel attacks

Side-channel attacks do not attack the cryptosystem as a mathematical object. Instead, they exploit information inadvertently leaked by its physical implementation.

Consider an attacker who observes a cryptosystem as it operates on secret data, such as a secret key. The attacker can learn far more information than just the input/output behavior of the system. Two important examples are:

- **Timing side channel:** In a vulnerable implementation, the time it takes to encrypt a block of plaintext may depend on the value of the secret key. An attacker who measures encryption time can learn information about the key, as shown below.

- **Power side channel:** In a vulnerable implementation, the amount of power used by the hardware as it encrypts a block of plaintext can depend on the value of the secret key. An attacker who wants to extract a secret key from a device like a smartcard can measure the device's power usage as it operates and learn information about the key.

Many other side channels have been used to attack implementations: electromagnetic radiation emanating from a device as it encrypts, heat emanating from a device as it encrypts [117], and even sound [71].

4.3.2.1 Timing attacks

Timing attacks are a significant threat to crypto implementations. Timing information can be measured by a remote network attacker who interacts with a victim server and measures the server's response time to certain requests. For a vulnerable implementation, the response time can leak information about a secret key. Timing information can also be obtained by a local attacker on the same machine as the victim, for example, when a low-privilege process tries to extract a secret key from a high-privilege process. In this case, the attacker obtains very accurate timing measurements about its target. Timing attacks have been demonstrated in both the local and remote settings.

In this section, we describe a timing attack on AES that exploits memory caching behavior on the victim machine. We will assume that the adversary can accurately measure the victim's running time as it encrypts a block of plaintext with AES. The attack we present exploits timing variations due to caching in the machine's memory hierarchy.

Modern processors use a hierarchy of caches to speed up reads and writes to memory. The fastest layer, called the L1 cache, is relatively small (e.g. 64KB). Data is loaded into the L1 cache in blocks (called lines) of 64 bytes. Loading a line into L1 cache takes considerably more time than reading a line already in cache.

This cache-induced difference in timing leads to a devastating key recovery attack against the fast table-based implementation of AES presented on page 119. An implementation that ignores these caching effects will be easily broken by a timing attack.

Recall that the table-based implementation of AES uses four tables T_0, T_1, T_2, T_3 for all but the last round. The last round does not include the MixColumns step and evaluation of this last round uses an explicit S table instead of the tables T_0, T_1, T_2, T_3. Suppose that when each execution of AES begins, the S table is not in the L1 cache. The first time a table entry is read, that part of the table will be loaded into L1 cache. Consequently, this first read will be slow, but subsequent reads to the same entry will be much faster since the data is already cached. Since the S table is only used in the last round of AES no parts of the table will be loaded in cache prior to the last round.

Letting $A = (a_i)_{i=0,\ldots,15}$ denote the 4×4 input to the last round, and letting $(w_i)_{i=0,\ldots,15}$ denote the 4×4 last round key, the final AES output is computed as the 4×4 matrix:

$$C = (c_{i,j}) = \begin{pmatrix} S[a_0] + w_0 & S[a_1] + w_1 & S[a_2] + w_2 & S[a_3] + w_3 \\ S[a_5] + w_4 & S[a_6] + w_5 & S[a_7] + w_6 & S[a_4] + w_7 \\ S[a_{10}] + w_8 & S[a_{11}] + w_9 & S[a_8] + w_{10} & S[a_9] + w_{11} \\ S[a_{15}] + w_{12} & S[a_{12}] + w_{13} & S[a_{13}] + w_{14} & S[a_{14}] + w_{15} \end{pmatrix} \tag{4.19}$$

The attacker is given this final output C.

To mount the attack, consider two consecutive entries in the output matrix C, say $c_0 = S[a_0]+w_0$ and $c_1 = S[a_1]+w_1$. Subtracting one equation from the other we see that when $a_0 = a_1$ the following relation holds:

$$c_0 - c_1 = w_0 - w_1 \ .$$

Therefore, with $\Delta := w_0 - w_1$ we have that $c_0 - c_1 = \Delta$ whenever $a_0 = a_1$. Moreover, when $a_0 \neq a_1$ the structure of the S table ensures that $c_0 - c_1 \neq \Delta$.

The key insight is that whenever $a_0 = a_1$, reading $S[a_0]$ loads the a_0 entry of S into the L1 cache so that the second access to this entry via $S[a_1]$ is much faster. However, when $a_0 \neq a_1$ it is possible that both reads miss the L1 cache so that both are slow. Therefore, when $a_0 = a_1$ the expected running time of the entire AES cipher is slightly less than when $a_0 \neq a_1$.

The attacker's plan now is to run the victim AES implementation on many random input blocks and measure the running time. For each value of $\Delta \in \{0,1\}^8$ the attacker creates a list L_Δ of all output ciphertexts where $c_0 - c_1 = \Delta$. For each Δ-value it computes the average running time among all ciphertexts in L_Δ. Given enough samples, the lowest average running time is obtained for the Δ-value satisfying $\Delta = w_0 - w_1$. Hence, timing information reveals one linear relation about the last round key: $w_0 - w_1 = \Delta$.

Suppose the implementation evaluates the terms of (4.19) in some sequential order. Repeating the timing procedure above for different consecutive pairs c_i and c_{i+1} in C reveals the difference in $\mathrm{GF}(2^8)$ between every two consecutive bytes of the last round key. Then if the first byte of the last round key is known, all remaining bytes of the last round key can be computed from the known differences. Moreover, since key expansion in AES-128 is invertible, it is a simple matter to reconstruct the AES-128 secret key from the last round key.

To complete the attack, the attacker simply tries all 256 possible values for the first byte of last round key. For each candidate value the attacker obtains a candidate AES-128 key. This key can be tested by trying it out on a few known plaintext/ciphertext pairs. Once a correct AES-128 key is found, the attacker has obtained the desired key.

This attack, due to Bonneau and Mironov [36], works quite well in practice. Their experiments on a Pentium IV Xeon successfully recovered the AES secret key using about 2^{20} timing measurements of the encryption algorithm. The attack only takes a few minutes to run. We note that the Pentium IV Xeon uses 32-byte cache lines so that the S table is split across eight lines.

Mitigations. The simplest approach to defeat timing attacks on AES is to use the AES-NI instructions that implement AES in hardware. These instructions are faster than a software implementation and always take the same amount of time, independent of the key or input data.

On processors that do not have built-in AES instructions one is forced to use a software implementation. One approach to mitigate cache-timing attacks is to use a table-free implementation of AES. Several such implementations of AES using a technique called **bit-slicing** provide reasonable performance in software and are supposedly resistant to timing attacks.

Another approach is to pre-load the tables T_0, T_1, T_2, T_3 and S into $L1$ cache before every invocation of AES. This prevents the cache-based timing attack, but only if the tables are not evicted from $L1$ cache while AES is executing. Ensuring that the tables stay in $L1$ cache is non-trivial on a modern processor. Interrupts during AES execution can evict cache lines. Similarly, *hyperthreading* allows for multiple threads to execute concurrently on the same core. While one thread pre-loads the AES tables into $L1$ cache another thread executing concurrently can inadvertently evict them.

Yet another approach is to pad AES execution to the maximum possible time to prevent timing attacks, but this has a non-negligible impact on performance.

To conclude, we emphasize that the following mitigation does not work: adding a random number of instructions at the end of every AES execution to randomly pad the running time does not prevent the attack. The attacker can overcome this by simply obtaining more samples and averaging out the noise.

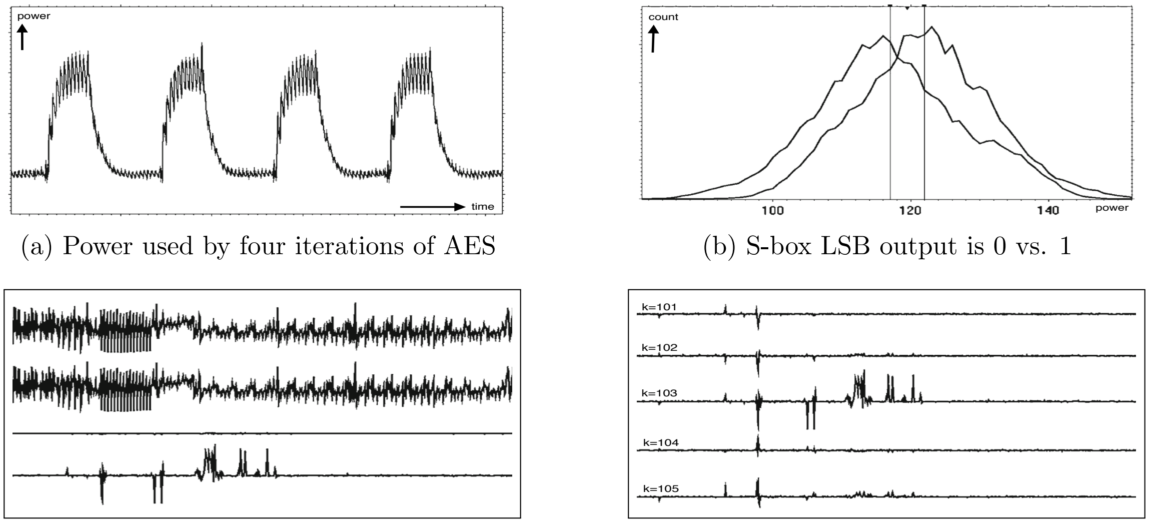
4.3.2.2 Power attacks on AES implementations

The amount of power consumed by a device as it operates can leak information about the inner-workings of the device, including secret keys stored on the device. Let us see how an attacker can use power measurements to quickly extract secret keys from a physical device.

As an example, consider a credit-card with an embedded chip where the chip contains a secret AES key. To make a purchase the user plugs the credit-card into a point-of-sale terminal. The terminal provides the card with the transaction details and the card authorizes the transaction using the secret embedded AES key. We leave the exact details for how this works to a later chapter.

Since the embedded chip must draw power from the terminal (it has no internal power source), it is quite easy for the terminal to measure the amount of power consumed by the chip at any given time. In particular, an attacker can measure the amount of power consumed as the AES algorithm is evaluated. Fig. 4.12a shows a test device's power consumption as it evaluates the AES-128 algorithm four times (the x-axis is time and y-axis is power). Each hump is one run of AES and within each hump the ten rounds of AES-128 are clearly visible.

Simple power analysis. Suppose an implementation contains a branch instruction that depends on a bit of the secret key. Say, the branch is taken when the least significant bit of the key is '1' and not taken otherwise. Since taking a branch requires more power than not taking it, the power trace will show a spike at the branch point when the key bit is one and no spike otherwise. An attacker



(a) Power used by four iterations of AES

(b) S-box LSB output is 0 vs. 1

(c) Power differential

(d) Differential for keys $k = 101, \ldots, 105$

Figure 4.12: AES differential power analysis (source: Kocher et al. [99])

can simply look for a spike at the appropriate point in the power trace and learn that bit of the key. With multiple key-dependent branch instructions the entire secret key can be extracted. This works quite well against simple implementations of certain cryptosystems (such as RSA, which is covered in a later chapter).

The attack of the previous paragraph, called **simple power analysis** (SPA), will not work on AES: during encryption the secret AES round keys are simply XORed into the cipher state. The power used by the XOR instruction only marginally depends on its operands and therefore the power used by the XOR reveals no useful information about the secret key. This resistance to simple power analysis was an attractive feature of AES.

Differential power analysis. Despite AES's resistance to SPA, a more sophisticated power analysis attack successfully extracts the AES secret key from simple implementations. Choose an AES key k at random and encrypt 4000 random plaintexts using the key k. For our test device the resulting 4000 power traces look quite different from each other indicating that the power trace is input dependent, the input being the random plaintext.

Next, consider the output of the first S-box in the first round. Call this output T. We hypothesize that the power consumed by the S-box lookup depends on the index being looked up. That is, we guess that the value of T is correlated with the power consumed by the table lookup instruction.

To test the hypothesis, let us split the 4000 traces into two piles according to the least significant bit of T: pile 1 contains traces where the LSB of T is 1 and pile 0 contains traces where the bit is 0. Consider the power consumed by traces in each pile at the moment in time when the card computes the output of the first S-box:

pile 1 (LSB = 1): mean power 116.9 units, standard deviation 10.7
pile 0 (LSB = 0): mean power 121.9 units, standard deviation 9.7

The two power distributions are shown in Fig. 4.12b. The distributions are close, but clearly different. Hence, with enough independent samples we can distinguish one distribution from the other.

To exploit this observation, consider Fig. 4.12c. The top line shows the power trace averaged over all traces in pile 1. The second line shows the power trace averaged over all traces in pile 0. The bottom line shows the difference between the two top traces, magnified by a factor of 15. The first spike in the bottom line is exactly at the time when the card computed the output of the first S-box. The size of the spike corresponds exactly to the difference in averages shown in Fig. 4.12b. This bottom line is called the **power differential**.

To attack a target device the attacker must first experiment with a clean device: the attacker loads a chosen secret key into the device and computes the power differential curve for the device as shown in Fig. 4.12c. Next, suppose the attacker obtains a device with an unknown embedded key. It can extract the key as follows:

```
first, measure the power trace for 4000 random plaintexts
next, for each candidate first byte $k \in \{0,1\}^8$ of the key do:
    split the 4000 samples into two piles according to the first bit of $T$
        (this is done using the current guess for $k$ and the 4000 known plaintexts)
    if the resulting power differential curve matches the pre-computed curve:
        output $k$ as the first byte of the key and stop
```

Fig. 4.12d shows this attack in action. When using the correct value for the first byte of the key ($k = 103$) we obtain the correct power differential curve. When the wrong guess is used ($k = 101, 102, 104, 105$) the power differential does not match the expected curve.

Iterating this procedure for all 16 bytes of the AES-128 key recovers the entire key.

Mitigations. A common defense against power analysis uses hardware tweaks. Conceptually, prior to executing AES the hardware draws a fixed amount of power to charge a capacitor and then runs the entire AES algorithm using power in the capacitor. Once AES is done the excess power left in the capacitor is discarded. The next application of AES again charges the capacitor and so on. This conceptual design (which takes some effort to implement correctly in practice) ensures that the device's power consumption is independent of secret keys embedded in the device.

Another mitigation approach concedes that some limited information about the secret key leaks every time the decryption algorithm runs. The goal is to then preemptively re-randomize the secret key after each invocation of the algorithm so that the attacker cannot combine the bits of information he learns from each execution. This approach is studied in an area called **leakage-resilient cryptography**.

4.3.3 Fault injection attacks on AES

Another class of implementation attacks, called **fault injection attacks**, attempt to deliberately cause the hardware to introduce errors while running the cryptosystem. An attacker can exploit the malformed output to learn information about the secret key. Injecting faults can be done by over-clocking the target hardware, by heating it using a laser, or by directing electromagnetic interference at the target chip [96].

Fault injection attacks have been used to break vulnerable implementations of AES by causing the AES engine to malfunction during encryption of a plaintext block. The resulting malformed ciphertext can reveal information about the secret key [96]. Fault attacks are easiest to describe in the context of public-key systems and we will come back and discuss them in detail in Section 16.4.3 where we show how they result in a complete break of some implementations of RSA.

One defense against fault injection attacks is to always check the result of the computation. For example, an AES engine could check that the computed AES ciphertext correctly decrypts to the given input plaintext. If the check fails, the hardware outputs an error and discards the computed ciphertext. Unfortunately this slows down AES performance by a factor of two and is hardly done in practice.

4.3.4 Quantum exhaustive search attacks

All the attacks described so far work on classical computers available today. Our physical world, however, is governed by the laws of quantum mechanics. In theory, computers can be built to use these laws to solve problems in much less time than would be required on a classical computer. Although no one has yet succeeded in building quantum computers, it could be just be a matter of time before the first quantum computer is built.

Quantum computers have significant implications to cryptography because they can be used to speed up certain attacks and even completely break some systems. Consider again a block cipher (E, D) with key space $\mathcal{K}$. Recall that in a classical exhaustive search the attacker is given a few plaintext/ciphertext pairs created with some key $k \in \mathcal{K}$ and the attacker tries all keys until he finds a key that maps the given plaintexts to the given ciphertexts. On a classical computer this takes time proportional to $|\mathcal{K}|$.

Quantum exhaustive search. Surprisingly, on a quantum computer the same exhaustive search problem can be solved in time proportional to only $\sqrt{|\mathcal{K}|}$. For block ciphers like AES-128 this means that exhaustive search will only require about $\sqrt{2^{128}} = 2^{64}$ steps. Computations involving 2^{64} steps can already be done in a reasonable amount of time on a classical computer. If a quantum computer is built that could handle a computation of that size, then AES-128 will no longer be secure.

The above discussion suggests that for a block cipher to resist a quantum exhaustive search attack, its key space $|\mathcal{K}|$ must have at least 2^{256} keys so that the time for quantum exhaustive search is on the order of 2^{128}. This threat of a quantum computer is one reason why AES supports a 256-bits key. Of course, we have no guarantees that there is not a faster quantum algorithm for breaking the AES-256 block cipher, but at least quantum exhaustive search is out of the question for that size key.

Grover's algorithm. The algorithm for quantum exhaustive search is a special case of a more general result in quantum computing due to Lov Grover [80]. The result says the following: suppose we are given a function $f : \mathcal{K} \to \{0, 1\}$ defined as follows

$$f(k) = \begin{cases} 1 & \text{if } k = k_0 \\ 0 & \text{otherwise} \end{cases} \tag{4.20}$$

for some $k_0 \in \mathcal{K}$. The goal is to find k_0 given only "black-box" access to f, namely by only querying f at different inputs. On a classical computer it is clear that the best algorithm is to try all possible $k \in \mathcal{K}$ and this takes $|\mathcal{K}|$ queries to f in the worst case.

Grover's algorithm shows that k_0 can be found on a quantum computer in only $O\big(\sqrt{|\mathcal{K}|}\cdot\text{time}(f)\big)$ steps, where time(f) is the time to evaluate $f(x)$. This is a very general result that holds for all functions f of the form shown in (4.20). This can be used to speed-up general hard optimization problems and is the "killer app" for quantum computers.

To break a block cipher like AES-128 given a few plaintext/ciphertext pairs, we would define the function:

$$f_{\text{AES}}(k) = \begin{cases} 1 & \text{if } AES(k, \overline{m}) = \overline{c} \\ 0 & \text{otherwise} \end{cases}$$

where $\overline{m} = (m_0, \ldots, m_Q)$ and $\overline{c} = (c_0, \ldots, c_Q)$ are the given ciphertext blocks. Assuming enough blocks are given, there is a unique key $k_0 \in \mathcal{K}$ that satisfies $AES(k, \overline{m}) = \overline{c}$ and this key can be found in time proportional to $\sqrt{|\mathcal{K}|}$ using Grover's algorithm.

4.4 Pseudo-random functions: basic definitions and properties

While secure block ciphers are the building block of many cryptographic systems, a closely related concept, called a pseudo-random function (or PRF), turns out to be the right tool in many applications. PRFs are conceptually simpler objects than block ciphers and, as we shall see, they have a broad range of applications. PRFs and block ciphers are so closely related that we can use secure block ciphers as a stand in for secure pseudo-random functions (under certain assumptions). This is quite nice, because as we saw in the previous section, we have available to us a number of very practical, and plausibly secure block ciphers.

4.4.1 Definitions

A **pseudo-random function (PRF)** F is a deterministic algorithm that has two inputs: a key k and an **input data block** x; its output $y := F(k, x)$ is called an **output data block**. As usual, there are associated, finite spaces: the key space $\mathcal{K}$, in which k lies, the input space $\mathcal{X}$, in which x lies, and the output space $\mathcal{Y}$, in which y lies. We say that F is **defined over** $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$.

Intuitively, our notion of security for a pseudo-random function says that for a randomly chosen key k, the function $F(k, \cdot)$ should — for all practical purposes — "look like" a random function from $\mathcal{X}$ to $\mathcal{Y}$. To make this idea more precise, let us first introduce some notation:

$$\text{Funs}[\mathcal{X}, \mathcal{Y}]$$

denotes the set of *all* functions $f : \mathcal{X} \to \mathcal{Y}$. This is a very big set:

$$|\text{Funs}[\mathcal{X}, \mathcal{Y}]| = |\mathcal{Y}|^{|\mathcal{X}|}.$$

We also introduce an attack game:

Attack Game 4.2 (PRF). For a given PRF F, defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define:

Experiment b:

- The challenger selects $f \in \text{Funs}[\mathcal{X}, \mathcal{Y}]$ as follows:

  if $b = 0$: $k \xleftarrow{\text{R}} \mathcal{K}$, $f \leftarrow F(k, \cdot)$;
  if $b = 1$: $f \xleftarrow{\text{R}} \text{Funs}[\mathcal{X}, \mathcal{Y}]$.

- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots,$ the ith query is an input data block $x_i \in \mathcal{X}$.

  The challenger computes $y_i \leftarrow f(x_i) \in \mathcal{Y}$, and gives y_i to the adversary.

- The adversary computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to F as

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \tag{4.21}$$

Finally, we say that $\mathcal{A}$ is a **Q-query PRF adversary** if $\mathcal{A}$ issues at most Q queries. □

Definition 4.2 (secure PRF). *A PRF F is **secure** if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F]$ *is negligible.*

Again, we stress that the queries made by the adversary in Attack Game 4.2 are allowed to be *adaptive*: the adversary is allowed to concoct each query in a way that depends on the previous responses from the challenger (see Exercise 4.6).

As discussed in Section 2.2.5, Attack Game 4.2 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\mathrm{PRF}\mathsf{adv}^*[\mathcal{A}, F]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.2.5 (namely, (2.11)) applies here as well:

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F] = 2 \cdot \mathrm{PRF}\mathsf{adv}^*[\mathcal{A}, F]. \tag{4.22}$$

Weakly secure PRFs. For certain constructions that use PRFs it suffices that the PRF satisfy a weaker security property than Definition 4.2. We say that a PRF is *weakly secure* if no efficient adversary can distinguish the PRF from a random function when its queries are severely restricted: it can only query the function at *random* points in the domain. Restricting the adversary's queries to random inputs makes it potentially easier to build weakly secure PRFs. In Exercise 4.2 we examine natural PRF constructions that are weakly secure, but not fully secure.

We define weakly secure PRFs by slightly modifying Attack Game 4.2. Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$. We modify the way in which an adversary $\mathcal{A}$ interacts with the challenger: whenever the adversary queries the function, the challenger chooses a random $x \in \mathcal{X}$ and sends both x and $f(x)$ to the adversary. In other words, the adversary sees evaluations of the function f at *random* points in $\mathcal{X}$ and needs to decide whether the function is truly random or pseudorandom. We define the adversary's advantage in this game, denoted $\mathrm{wPRF}\mathsf{adv}[\mathcal{A}, F]$, as in (4.21).

Definition 4.3 (weakly secure PRF). *A PRF F is **weakly secure** if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{wPRF}\mathsf{adv}[\mathcal{A}, F]$ *is negligible.*

4.4.2 Efficient implementation of random functions

Just as in Section 4.1.2, we can implement the random function chosen from $\mathrm{Funs}[\mathcal{X}, \mathcal{Y}]$ used by the challenger in Experiment 1 of Attack Game 4.2 by a **faithful gnome**. Just as in the block cipher case, the challenger keeps track of input/output pairs (x_i, y_i). When the challenger receives the ith query x_i, he tests whether $x_i = x_j$ for some $j < i$; if so, he sets $y_i \leftarrow y_j$ (this ensures that the challenger implements a function); otherwise, he chooses y_i at random from the set $\mathcal{Y}$; finally, he sends y_i to the adversary. We can write the logic of this implementation of the challenger as follows:

```
upon receiving the $i$th query $x_i \in \mathcal{X}$ from $\mathcal{A}$ do:
    if $x_i = x_j$ for some $j < i$
        then $y_i \leftarrow y_j$
        else $y_i \xleftarrow{R} \mathcal{Y}$
    send $y_i$ to $\mathcal{A}$.
```

4.4.3 When is a secure block cipher a secure PRF?

In this section, we ask the question: when is a secure block cipher a secure PRF? In answering this question, we introduce a proof technique that is used heavily throughout cryptography.

Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$, and let $N := |\mathcal{X}|$. We may naturally view E as a PRF, defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$. Now suppose that $\mathcal{E}$ is a secure block cipher; that is, no efficient adversary can effectively distinguish E from a random permutation. Does this imply that E is also a secure PRF? That is, does this imply that no efficient adversary can effectively distinguish E from a random function?

The answer to this question is "yes," provided N is super-poly. Before arguing this, let us argue that the answer is "no" when N is small.

Consider a PRF adversary playing Attack Game 4.2 with respect to E. Let f be the function chosen by the challenger: in Experiment 0, $f = E(k, \cdot)$ for random $k \in \mathcal{K}$, while in Experiment 1, f is randomly chosen from Funs$[\mathcal{X}, \mathcal{X}]$. Suppose that N is so small that an efficient adversary can afford to obtain the value of $f(x)$ for all $x \in \mathcal{X}$. Moreover, our adversary $\mathcal{A}$ outputs 1 if it sees that $f(x) = f(x')$ for two distinct values $x, x' \in \mathcal{X}$, and outputs 0 otherwise. Clearly, in Experiment 0, $\mathcal{A}$ outputs 1 with probability 0, since $E(k, \cdot)$ is a permutation. However, in Experiment 1, $\mathcal{A}$ outputs 1 with probability $1 - N!/N^N \geq 1/2$. Thus, PRF$\mathsf{adv}[\mathcal{A}, E] \geq 1/2$, and so E is not a secure PRF.

The above argument can be refined using the Birthday Paradox (see Section B.1). For any poly-bounded Q, we can define an efficient PRF adversary $\mathcal{A}$ that plays Attack Game 4.2 with respect to E, as follows. Adversary $\mathcal{A}$ simply makes Q distinct queries to its challenger, and outputs 1 iff it sees that $f(x) = f(x')$ for two distinct values $x, x' \in \mathcal{X}$ (from among the Q values given to the challenger). Again, in Experiment 0, $\mathcal{A}$ outputs 1 with probability 0; however, by Theorem B.1, in Experiment 1, $\mathcal{A}$ outputs 1 with probability at least $\min\big\{Q(Q-1)\big/4N, 0.63\big\}$. Thus, by making just $O(N^{1/2})$ queries, an adversary can easily see that a permutation does not behave like a random function.

It turns out that the "birthday attack" is about the best that any adversary can do, and when N is super-poly, this attack becomes infeasible:

Theorem 4.4 (PRF Switching Lemma). *Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$, and let $N := |\mathcal{X}|$. Let $\mathcal{A}$ be an adversary that makes at most Q queries to its challenger. Then*

$$\Big|\text{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] - \text{PRF}\mathsf{adv}[\mathcal{A}, E]\Big| \leq Q^2/2N.$$

Before proving this theorem, we derive the following simple corollary:

Corollary 4.5. *Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$, and assume that $N := |\mathcal{X}|$ is super-poly. Then $\mathcal{E}$ is a secure block cipher if and only if E is a secure PRF.*

Proof. By definition, if $\mathcal{A}$ is an efficient adversary, the maximum number of queries Q it makes to its challenger is poly-bounded. Therefore, by Theorem 4.4, we have

$$\Big|\mathrm{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] - \mathrm{PRF}\mathsf{adv}[\mathcal{A}, E]\Big| \le Q^2/2N$$

Since N is super-poly and Q is poly-bounded, the value $Q^2/2N$ is negligible (see Fact 2.6). It follows that $\mathrm{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible if and only if $\mathrm{PRF}\mathsf{adv}[\mathcal{A}, E]$ is negligible. □

Actually, the proof of Theorem 4.4 has nothing to do with block ciphers and PRFs — it is really an argument concerning random permutations and random functions. Let us define a new attack game that tests an adversary's ability to distinguish a random permutation from a random function.

Attack Game 4.3 (permutation vs. function). For a given finite set $\mathcal{X}$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define:

Experiment b:

- The challenger selects $f \in \mathrm{Funs}[\mathcal{X}, \mathcal{X}]$ as follows:

  if $b = 0$: $f \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$;
  if $b = 1$: $f \xleftarrow{R} \mathrm{Funs}[\mathcal{X}, \mathcal{X}]$.

- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots$, the ith query is an input data block $x_i \in \mathcal{X}$.

  The challenger computes $y_i \leftarrow f(x_i) \in \mathcal{Y}$, and gives y_i to the adversary.

- The adversary computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to $\mathcal{X}$ as

$$\mathrm{PF}\mathsf{adv}[\mathcal{A}, \mathcal{X}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Theorem 4.6. *Let $\mathcal{X}$ be a finite set of size N. Let $\mathcal{A}$ be an adversary that makes at most Q queries to its challenger. Then*

$$\mathrm{PF}\mathsf{adv}[\mathcal{A}, \mathcal{X}] \le Q^2/2N.$$

We first show that the above theorem easily implies Theorem 4.4:

Proof of Theorem 4.4. Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$. Let $\mathcal{A}$ be an adversary that makes at most Q queries to its challenger. We define Games 0, 1, and 2, played between $\mathcal{A}$ and a challenger. For $j = 0, 1, 2$, we define p_j to be the probability that $\mathcal{A}$ outputs 1 in Game j. In each game, the challenger chooses a function $f : \mathcal{X} \to \mathcal{X}$ according to a particular distribution, and responds to each query $x \in \mathcal{X}$ made by $\mathcal{A}$ with the value $f(x)$.

Game 0: The challenger in this game chooses $f := E(k, \cdot)$, where $k \in \mathcal{K}$ is chosen at random.

Game 1: The challenger in this game chooses $f \in \text{Perms}[\mathcal{X}]$ at random.

Game 2: The challenger in this game chooses $f \in \text{Funs}[\mathcal{X}, \mathcal{X}]$ at random.

Observe that by definition,

$$|p_1 - p_0| = \text{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}],$$
$$|p_2 - p_0| = \text{PRF}\mathsf{adv}[\mathcal{A}, E],$$

and that by Theorem 4.6,

$$|p_2 - p_1| = \text{PF}\mathsf{adv}[\mathcal{A}, \mathcal{X}] \leq Q^2/2N.$$

Putting these together, we get

$$\big|\text{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] - \text{PRF}\mathsf{adv}[\mathcal{A}, E]\big| = \big||p_1 - p_0| - |p_2 - p_0|\big| \leq |p_2 - p_1| \leq Q^2/2N,$$

which proves the theorem. □

So it remains to prove Theorem 4.6. Before doing so, we state and prove a simple, but extremely useful fact:

Theorem 4.7 (Difference Lemma). *Let Z, W_0, W_1 be events defined over some probability space, and let $\bar{Z}$ denote the complement of the event Z. Suppose that $W_0 \wedge \bar{Z}$ occurs if and only if $W_1 \wedge \bar{Z}$ occurs. Then we have*

$$\big|\Pr[W_0] - \Pr[W_1]\big| \leq \Pr[Z].$$

Proof. This is a simple calculation. We have

$$\begin{aligned}\big|\Pr[W_0] - \Pr[W_1]\big| &= \big|\Pr[W_0 \wedge Z] + \Pr[W_0 \wedge \bar{Z}] - \Pr[W_1 \wedge Z] - \Pr[W_1 \wedge \bar{Z}]\big| \\ &= \big|\Pr[W_0 \wedge Z] - \Pr[W_1 \wedge Z]\big| \\ &\leq \Pr[Z].\end{aligned}$$

The second equality follows from the assumption that $W_0 \wedge \bar{Z} \iff W_1 \wedge \bar{Z}$, and so in particular, $\Pr[W_0 \wedge \bar{Z}] = \Pr[W_1 \wedge \bar{Z}]$. The final inequality follows from the fact that both $\Pr[W_0 \wedge Z]$ and $\Pr[W_1 \wedge Z]$ are numbers between 0 and $\Pr[Z]$. □

In most of our applications of the Difference Lemma, W_0 will represent the event that a given adversary outputs 1 in some game against a certain challenger, while W_1 will be the event that the same adversary outputs 1 in a game played against a different challenger. To apply the Difference Lemma, we define these two games so that they both operate on the same underlying probability space. This means that we view the random choices made by both the adversary and the challenger as the same in both games — all that differs between the two games is the rule used by the challenger to compute its responses to the adversary's queries.

Proof of Theorem 4.6. Consider an adversary $\mathcal{A}$ that plays Attack Game 4.3 with respect to $\mathcal{X}$, where $N := |\mathcal{X}|$, and assume that $\mathcal{A}$ makes at most Q queries to the challenger. Consider Experiment 0 of this attack game. Using the "faithful gnome" idea discussed in Section 4.4.2, we can implement Experiment 0 by keeping track of input/output pairs (x_i, y_i); moreover, it will be convenient to choose initial "default" values z_i for y_i, where the values $z_1, \ldots, z_Q$ are chosen uniformly and independently at random from $\mathcal{X}$; these "default" values are over-ridden, if necessary, to ensure the challenger defines a random permutation. Here are the details:

```
    $z_1, \ldots, z_Q \xleftarrow{R} \mathcal{X}$
    upon receiving the $i$th query $x_i$ from $\mathcal{A}$ do:
        if $x_i = x_j$ for some $j < i$ then
            $y_i \leftarrow y_j$
        else
            $y_i \leftarrow z_i$
(*)         if $y_i \in \{y_1, \ldots, y_{i-1}\}$ then $y_i \xleftarrow{R} \mathcal{X} \setminus \{y_1, \ldots, y_{i-1}\}$
        send $y_i$ to $\mathcal{A}$.
```

The line marked $(*)$ tests if the default value z_i needs to be over-ridden to ensure that no output is for two distinct inputs.

Let W_0 be the event that $\mathcal{A}$ outputs 1 in this game, which we call Game 0.

We now obtain a different game by modifying the above implementation of the challenger:

```
    $z_1, \ldots, z_Q \xleftarrow{R} \mathcal{X}$
    upon receiving the $i$th query $x_i$ from $\mathcal{A}$ do:
        if $x_i = x_j$ for some $j < i$ then
            $y_i \leftarrow y_j$
        else
            $y_i \leftarrow z_i$
        send $y_i$ to $\mathcal{A}$.
```

All we have done is dropped line marked $(*)$ in the original challenger: our "faithful gnome" becomes a "forgetful gnome," and simply forgets to make the output consistency check.

Let W_1 be the event that $\mathcal{A}$ outputs 1 in the game played against this modified challenger, which we call Game 1.

Observe that Game 1 is equivalent to Experiment 1 of Attack Game 4.3; in particular, $\Pr[W_1]$ is equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 1 of Attack Game 4.3. Therefore, we have

$$\mathsf{PFadv}[\mathcal{A}, \mathcal{X}] = |\Pr[W_0] - \Pr[W_1]|.$$

We now want to apply the Difference Lemma. To do this, both games are understood to operate on the *same* underlying probability space. All of the random choices made by the adversary and challenger are the same in both games — all that differs is the rule used by the challenger to compute its responses. In particular, this means that the random choices made by $\mathcal{A}$, as well as the values $z_1, \ldots, z_Q$ chosen by the challenger, not only have identical distributions, but are *literally the same values* in both games.

Define Z to be the event that $z_i = z_j$ for some $i \neq j$. Now suppose we run Game 0 and Game 1, and event Z does not occur. This means that the z_i values are all distinct. Now, since the adversary's random choices are the same in both games, its first query in both games is the same, and therefore the challenger's response is the same in both games. The adversary's second query (which is a function of its random choices and the challenger's first response) is the same in both games. By the assumption that Z does not occur, the challenger's response is the same in both games. Continuing this argument, one sees that each of the adversary's queries and each of the challenger's responses are the same in both games, and therefore the adversary's output is the same in both games. Thus, if Z does not occur and the adversary outputs 1 in Game 0, then the adversary also outputs 1 in Game 1. Likewise, if Z does not occur and the adversary outputs 1 in Game 1, then the adversary outputs 1 in Game 0. More succinctly, we have $W_0 \wedge \bar{Z}$ occurs if and only if $W_1 \wedge \bar{Z}$ occurs. So the Difference Lemma applies, and we obtain

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z].$$

It remains to bound $\Pr[Z]$. However, this follows from the union bound: for each pair (i, j) of distinct indices, $\Pr[z_i = z_j] = 1/N$, and as there are less than $Q^2/2$ such pairs, we have

$$\Pr[Z] \leq Q^2/2N.$$

That proves the theorem. □

While there are other strategies one might use to prove the previous theorem (see Exercise 4.24), the **forgetful gnome** technique that we used in the above proof is very useful and we will see it again many times in the sequel.

4.4.4 Constructing PRGs from PRFs

It is easy to construct a PRG from a PRF. Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, let $\ell \geq 1$ be a poly-bounded value, and let $x_1, \ldots, x_\ell$ be any fixed, distinct elements of $\mathcal{X}$ (this requires that $|\mathcal{X}| \geq \ell$). We define a PRG G with seed space $\mathcal{K}$ and output space $\mathcal{Y}^\ell$, as follows: for $k \in \mathcal{K}$,

$$G(k) := (F(k,\ x_1), \ldots, F(k,\ x_\ell)).$$

Theorem 4.8. *If F is a secure PRF, then the PRG G described above is a secure PRG.*

> *In particular, for every PRG adversary $\mathcal{A}$ that plays Attack Game 3.1 with respect to G, there is a PRF adversary $\mathcal{B}$ that plays Attack Game 4.2 with respect to F, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{PRG}\mathsf{adv}[\mathcal{A}, G] = \text{PRF}\mathsf{adv}[\mathcal{B}, F].$$

Proof. Let $\mathcal{A}$ be an efficient PRG adversary that plays Attack Game 3.1 with respect to G. We describe a corresponding PRF adversary $\mathcal{B}$ that plays Attack Game 4.2 with respect to F. Adversary $\mathcal{B}$ works as follows:

> $\mathcal{B}$ queries its challenger at $x_1, \ldots, x_\ell$, obtaining responses $y_1, \ldots, y_\ell$. Adversary $\mathcal{B}$ then plays the role of challenger to $\mathcal{A}$, giving $\mathcal{A}$ the value $(y_1, \ldots, y_\ell)$. Adversary $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

It is obvious from the construction that for $b = 0, 1$, the probability that $\mathcal{B}$ outputs 1 in Experiment b of Attack Game 4.2 with respect to F is precisely equal to the probability that $\mathcal{A}$ outputs 1 in Experiment b of Attack Game 3.1 with respect to G. The theorem then follows immediately. □

4.4.4.1 Deterministic counter mode

The above construction gives us another way to build a semantically secure cipher out of a secure block cipher. Suppose $\mathcal{E} = (E, D)$ is a block cipher defined over $(\mathcal{K}, \mathcal{X})$, where $\mathcal{X} = \{0,1\}^n$. Let $N := |\mathcal{X}| = 2^n$. Assume that N is super-poly and that $\mathcal{E}$ is a secure block cipher. Then by Theorem 4.4, the encryption function E is a secure PRF (defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$). We can then apply Theorem 4.8 to E to obtain a secure PRG, and finally apply Theorem 3.1 to this PRG to obtain a semantically secure stream cipher.

Let us consider this stream cipher in detail. This cipher $\mathcal{E}' = (E', D')$ has key space $\mathcal{K}$, and message and ciphertext space $\mathcal{X}^{\leq \ell}$, where ℓ is a poly-bounded value, and in particular, $\ell \leq N$. We can define $x_1, \ldots, x_\ell$ to be any convenient elements of $\mathcal{X}$; in particular, we can define x_i to be the n-bit binary encoding of $i-1$, which we denote $\langle i-1 \rangle_n$. Encryption and decryption for $\mathcal{E}'$ work as follows.

- For $k \in \mathcal{K}$ and $m \in \mathcal{X}^{\leq \ell}$, with $v := |m|$, we define
$$E'(k, m) := \big(E(k, \langle 0 \rangle_n) \oplus m[0], \ldots, E(k, \langle v-1 \rangle_n) \oplus m[v-1]\big).$$
- For $k \in \mathcal{K}$ and $c \in \mathcal{X}^{\leq \ell}$, with $v := |c|$, we define
$$D'(k, c) := \big(E(k, \langle 0 \rangle_n) \oplus c[0], \ldots, E(k, \langle v-1 \rangle_n) \oplus c[v-1]\big).$$

This mode of operation of a block cipher is called **deterministic counter mode**. It is illustrated in Fig. 4.13. Notice that unlike ECB mode, the decryption algorithm D is never used. Putting together Theorems 4.4, 4.8, and 3.1, we see that cipher $\mathcal{E}'$ is semantically secure; in particular, for any efficient SS adversary $\mathcal{A}$, there exists an efficient BC adversary $\mathcal{B}$ such that

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}'] \leq 2 \cdot \mathsf{BCadv}[\mathcal{B}, \mathcal{E}] + \ell^2/N. \tag{4.23}$$

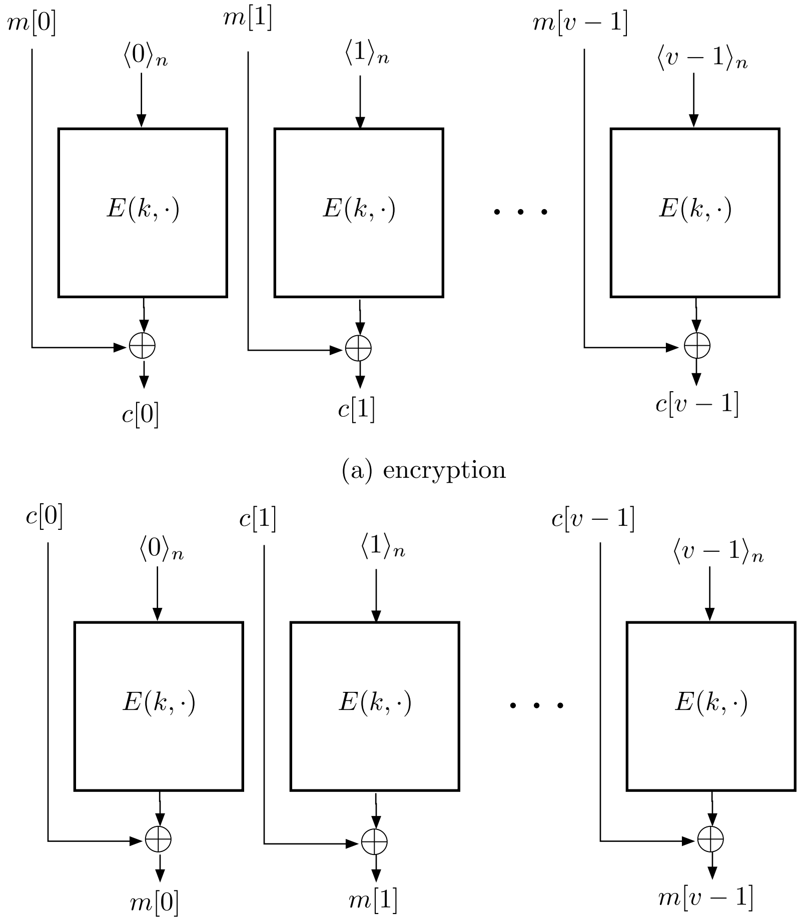
An advantage of deterministic counter mode over ECB mode is that it is semantically secure without making any restrictions on the message space. The only disadvantage is that security might degrade significantly for very long messages, because of the ℓ^2/N term in (4.23). Indeed, it is essential that $\ell^2/2N$ is very small. Consider the following attack on $\mathcal{E}'$. Set m_0 to be the message consisting of ℓ zero blocks, and set m_1 to be a message consisting of ℓ random blocks. If the challenger in Attack Game 2.1 encrypts m_0 using E', then the ciphertext will not contain any duplicate blocks. However, by the birthday paradox (see Theorem B.1), if the challenger encrypts m_1, the ciphertext will contain duplicate blocks with probability at least $\min\big\{\ell(\ell-1)/4N, 0.63\big\}$. So the adversary $\mathcal{A}$ that constructs m_0 and m_1 in this way, and outputs 1 if and only if the ciphertext contains duplicate blocks, has an advantage that grows quadratically in ℓ, and is non-negligible for $\ell \approx N^{1/2}$.

4.4.5 Mathematical details

As usual, we give a more mathematically precise definition of a PRF, using the terminology defined in Section 2.3.

Definition 4.4 (pseudo-random function). *A **pseudo-random function** consists of an algorithm F, along with three families of spaces with system parameterization P:*

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{X} = \{\mathcal{X}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \textit{and} \quad \mathbf{Y} = \{\mathcal{Y}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$



(a) encryption

(b) decryption

Figure 4.13: Encryption and decryption for deterministic counter mode

such that

1. **K***, X, and* **Y** *are efficiently recognizable.*

2. **K** *and* **Y** *are efficiently sampleable.*

3. *Algorithm F is a deterministic algorithm that on input $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, and $x \in \mathcal{X}_{\lambda,\Lambda}$, runs in time bounded by a polynomial in λ, and outputs an element of $\mathcal{Y}_{\lambda,\Lambda}$.*

As usual, in defining security, the attack game is parameterized by security and system parameters, and the advantage is a function of the security parameter.

4.5 Constructing block ciphers from PRFs

In this section, we show how to construct a secure block cipher from any secure PRF whose output space and input space is $\{0,1\}^n$, where 2^n is super-poly. The construction is called the **Luby-Rackoff construction**. The result itself is mainly of theoretical interest, as block ciphers that are used in practice have a more ad hoc design; however, the result is sometimes seen as a justification for the design of practical block ciphers that have a Feistel network structure (see Section 4.2.1).

Let F be a PRF, defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, where $\mathcal{X} = \{0,1\}^n$. We describe a block cipher $\mathcal{E} = (E, D)$ whose key space is $\mathcal{K}^3$, and whose data block space is $\mathcal{X}^2$.

Given a key $(k_1, k_2, k_3) \in \mathcal{K}^3$ and a data block $(u, v) \in \mathcal{X}^2$, the encryption algorithm E runs as follows:

```
$w \leftarrow u \oplus F(k_1,\ v)$
$x \leftarrow v \oplus F(k_2,\ w)$
$y \leftarrow w \oplus F(k_3,\ x)$
output $(x, y)$.
```

Given a key $(k_1, k_2, k_3) \in \mathcal{K}^3$ and a data block $(x, y) \in \mathcal{X}^2$, the decryption algorithm D runs as follows:

```
$w \leftarrow y \oplus F(k_3,\ x)$
$v \leftarrow x \oplus F(k_2,\ w)$
$u \leftarrow w \oplus F(k_1,\ v)$
output $(u, v)$.
```

See Fig. 4.14 for an illustration of $\mathcal{E}$. Notice that the cipher is a three round Feistel network.

It is easy to see that $\mathcal{E}$ is a block cipher. It is useful to see algorithm E as consisting of 3 "rounds." For $k \in \mathcal{K}$, let us define the "round function"

$$\begin{aligned} \phi_k : \quad & \mathcal{X}^2 \to \mathcal{X}^2 \\ & (a, b) \mapsto (b, a \oplus F(k, b)). \end{aligned}$$

It is easy to see that for any fixed k, the function ϕ_k is a permutation on $\mathcal{X}^2$; indeed, if $\sigma(a, b) := (b, a)$, then

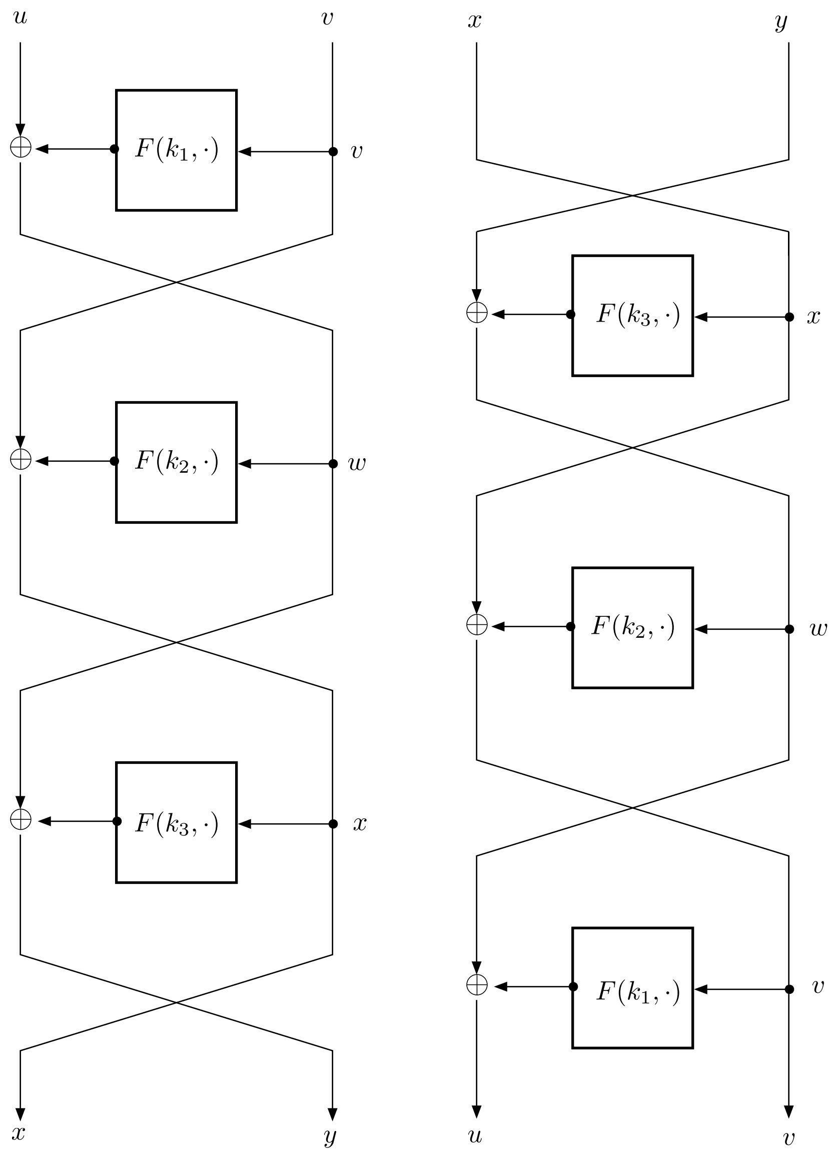
$$\phi_k^{-1} = \sigma \circ \phi_k \circ \sigma.$$

Moreover, we see that

$$E((k_1, k_2, k_3), \cdot) = \phi_{k_3} \circ \phi_{k_2} \circ \phi_{k_1}$$

and

$$D((k_1, k_2, k_3), \cdot) = \phi_{k_1}^{-1} \circ \phi_{k_2}^{-1} \circ \phi_{k_3}^{-1} = \sigma \circ \phi_{k_1} \circ \phi_{k_2} \circ \phi_{k_3} \circ \sigma.$$



(a) Encryption

(b) Decryption

Figure 4.14: Encryption and decryption with Luby-Rackoff

Theorem 4.9. *If F is a secure PRF and $N := |\mathcal{X}| = 2^n$ is super-poly, then the Luby-Rackoff cipher $\mathcal{E} = (E, D)$ constructed from F is a secure block cipher.*

> *In particular, for every Q-query BC adversary $\mathcal{A}$ that attacks $\mathcal{E}$ as in Attack Game 4.1, there exists a PRF adversary $\mathcal{B}$ that plays Attack Game 4.2 with respect to F, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq 3 \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{Q^2}{N} + \frac{Q^2}{2N^2}.$$

Proof idea. By Corollary 4.5, and the assumption that N is super-poly, it suffices to show that E is a secure PRF. So we want to show that if an adversary is playing in Experiment 0 of Attack Game 4.2 with respect to E, the challenger's responses effectively "look like" completely random bit strings. We may assume that the adversary never makes the same query twice. Moreover, as F is a PRF, we can replace $F(k_1, \cdot)$, $F(k_2, \cdot)$, and $F(k_3, \cdot)$ by truly random functions, f_1, f_2, and f_3, and the adversary should hardly notice the difference.

So now, given a query (u_i, v_i), the challenger computes its response (x_i, y_i) as follows:

$$\begin{aligned}
&w_i \leftarrow u_i \oplus f_1(v_i)\\
&x_i \leftarrow v_i \oplus f_2(w_i)\\
&y_i \leftarrow w_i \oplus f_3(x_i).
\end{aligned}$$

A rough, intuitive argument goes like this. Suppose that no two w_i values are the same. Then all of the outputs of f_2 will be random and independent. From this, we can argue that the x_i's are also random and independent. Then from this, it will follow that except with negligible probability, the inputs to f_3 will be distinct. From this, we can conclude that the y_i's are essentially random and independent.

So we will be in good shape if we can show that all of the w_i's are distinct. But the w_i's are obtained indirectly from the random function f_1, and so with some care, one can indeed argue that the w_i will be distinct, except with negligible probability. □

Proof. Let $\mathcal{A}$ be an efficient BC adversary that plays Attack Game 4.1 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger. We want to show that $\mathrm{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible. To do this, we first show that $\mathrm{PRF}\mathsf{adv}[\mathcal{A}, E]$ is negligible, and the result will then follow from the PRF Switching Lemma (i.e., Theorem 4.4) and the assumption that N is super-poly.

To simplify things a bit, we replace $\mathcal{A}$ with an adversary $\mathcal{A}_0$ with the following properties:

- $\mathcal{A}_0$ always makes exactly Q queries to its challenger;
- $\mathcal{A}_0$ never makes the same query more than once;
- $\mathcal{A}_0$ is just as efficient as $\mathcal{A}$ (more precisely, $\mathcal{A}_0$ is an elementary wrapper around $\mathcal{A}$);
- $\mathrm{PRF}\mathsf{adv}[\mathcal{A}_0, E] = \mathrm{PRF}\mathsf{adv}[\mathcal{A}, E]$.

Adversary $\mathcal{A}_0$ simply runs the same protocol as $\mathcal{A}$; however, it keeps a table of query/response pairs so as to avoid making duplicate queries; moreover, it "pads" the execution of $\mathcal{A}$ if necessary, so as to make exactly Q queries.

The overall strategy of the proof is as follows. First, we define Game 0 to be the game played between $\mathcal{A}_0$ and the challenger of Experiment 0 of Attack Game 4.2 with respect to E. We then define several more games: Game 1, Game 2, and Game 3. Each of these games is played between $\mathcal{A}_0$ and a different challenger; moreover, the challenger in Game 3 is equivalent to the challenger of Experiment 1 of Attack Game 4.2. Also, for $j = 0, \ldots, 3$, we define W_j to be the event that $\mathcal{A}_0$ outputs 1 in Game j. We will show that for $j = 1, \ldots, 3$ that the value $|\Pr[W_j] - \Pr[W_{j-1}]|$ is negligible, from which it will follow that

$$|\Pr[W_3] - \Pr[W_0]| = \text{PRF}\mathsf{adv}[\mathcal{A}_0, E]$$

is also negligible.

Game 0. Let us begin by giving a detailed description of the challenger in Game 0 that is convenient for our purposes:

```
$k_1, k_2, k_3 \xleftarrow{R} \mathcal{K}$
upon receiving the $i$th query $(u_i, v_i) \in \mathcal{X}^2$ (for $i = 1, \ldots, Q$) do:
    $w_i \leftarrow u_i \oplus F(k_1,\ v_i)$
    $x_i \leftarrow v_i \oplus F(k_2,\ w_i)$
    $y_i \leftarrow w_i \oplus F(k_3,\ x_i)$
    send $(x_i, y_i)$ to the adversary.
```

Recall that the adversary $\mathcal{A}_0$ is guaranteed to always make Q distinct queries $(u_1, v_1), \ldots, (u_Q, v_Q)$; that is, the (u_i, v_i) values are distinct *as pairs*, so that for $i \neq j$, we may have $u_i = u_j$ or $v_i = v_j$, but not both.

Game 1. We next play the "PRF card," replacing the three functions $F(k_1, \cdot), F(k_2, \cdot), F(k_3, \cdot)$ by truly random functions f_1, f_2, f_3. Intuitively, since F is a secure PRF, the adversary $\mathcal{A}_0$ should not notice the difference. Our challenger in Game 1 thus works as follows:

```
$f_1, f_2, f_3 \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{X}]$
upon receiving the $i$th query $(u_i, v_i) \in \mathcal{X}^2$ (for $i = 1, \ldots, Q$) do:
    $w_i \leftarrow u_i \oplus f_1(v_i)$
    $x_i \leftarrow v_i \oplus f_2(w_i)$
    $y_i \leftarrow w_i \oplus f_3(x_i)$
    send $(x_i, y_i)$ to the adversary.
```

As discussed in Exercise 4.26, we can model the three PRFs $F(k_1, \cdot), F(k_2, \cdot), F(k_3, \cdot)$ as a single PRF F', called the 3-wise parallel composition of F: the PRF F' is defined over $(\mathcal{K}^3, \{1, 2, 3\} \times \mathcal{X}, \mathcal{X})$, and $F'((k_1, k_2, k_3), (s, x)) := F(k_s, x)$. We can easily construct an adversary $\mathcal{B}'$, just as efficient as $\mathcal{A}_0$, such that

$$|\Pr[W_1] - \Pr[W_0]| = \text{PRF}\mathsf{adv}[\mathcal{B}', F']. \tag{4.24}$$

Adversary $\mathcal{B}'$ simply runs $\mathcal{A}_0$ and outputs whatever $\mathcal{A}_0$ outputs; when $\mathcal{A}_0$ queries its challenger with a pair (u_i, v_i), adversary $\mathcal{B}'$ computes the response (x_i, y_i) for $\mathcal{A}_0$ by computing

```
$w_i \leftarrow u_i \oplus f'(1, v_i)$
$x_i \leftarrow v_i \oplus f'(2, w_i)$
$y_i \leftarrow w_i \oplus f'(3, x_i)$.
```

Here, the f' denotes the function chosen by $\mathcal{B}'$'s challenger in Attack Game 4.2 with respect to F'. It is clear that $\mathcal{B}'$ outputs 1 with probability $\Pr[W_0]$ in Experiment 0 of that attack game, while it outputs 1 with probability $\Pr[W_1]$ in Experiment 1, from which (4.24) follows.

By Exercise 4.26, there exists an adversary $\mathcal{B}$, just as efficient as $\mathcal{B}'$, such that

$$\mathrm{PRF}\mathsf{adv}[\mathcal{B}', F'] = 3 \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F]. \tag{4.25}$$

Game 2. We next make a purely conceptual change: we implement the random functions f_2 and f_3 using the "faithful gnome" idea discussed in Section 4.4.2. This is not done for efficiency, but rather, to set us up so as to be able to make (and easily analyze) a more substantive modification later, in Game 3. Our challenger in this game works as follows:

```
$f_1 \overset{\mathrm{R}}{\leftarrow} \text{Funs}[\mathcal{X}, \mathcal{X}]$
$X_1, \ldots, X_Q \overset{\mathrm{R}}{\leftarrow} \mathcal{X}$
$Y_1, \ldots, Y_Q \overset{\mathrm{R}}{\leftarrow} \mathcal{X}$
upon receiving the $i$th query $(u_i, v_i) \in \mathcal{X}^2$ (for $i = 1, \ldots, Q$) do:
    $w_i \leftarrow u_i \oplus f_1(v_i)$
    $x'_i \leftarrow X_i$; $\boxed{\text{if } w_i = w_j \text{ for some } j < i \text{ then } x'_i \leftarrow x'_j;}$ $x_i \leftarrow v_i \oplus x'_i$
    $y'_i \leftarrow Y_i$; $\boxed{\text{if } x_i = x_j \text{ for some } j < i \text{ then } y'_i \leftarrow y'_j;}$ $y_i \leftarrow w_i \oplus y'_i$
    send $(x_i, y_i)$ to the adversary.
```

The idea is that the value x'_i represents $f_2(w_i)$. By default, x'_i is equal to the random value X_i; however, the boxed code over-rides this default value if w_i is the same as w_j for some $j < i$. Similarly, the value y'_i represents $f_3(x_i)$. By default, y'_i is equal to the random value Y_i, and the boxed code over-rides the default if necessary.

Since the challenger in Game 2 is completely equivalent to that of Game 1, we have

$$\Pr[W_2] = \Pr[W_1]. \tag{4.26}$$

Game 3. We now employ the "forgetful gnome" technique, which we already saw in the proof of Theorem 4.6. The idea is to simply eliminate the consistency checks made by the challenger in Game 2. Here is the logic of the challenger in Game 3:

```
$f_1 \overset{\mathrm{R}}{\leftarrow} \text{Funs}[\mathcal{X}, \mathcal{X}]$
$X_1, \ldots, X_Q \overset{\mathrm{R}}{\leftarrow} \mathcal{X}$
$Y_1, \ldots, Y_Q \overset{\mathrm{R}}{\leftarrow} \mathcal{X}$
upon receiving the $i$th query $(u_i, v_i) \in \mathcal{X}^2$ (for $i = 1, \ldots, Q$) do:
    $w_i \leftarrow u_i \oplus f_1(v_i)$
    $x'_i \leftarrow X_i$; $x_i \leftarrow v_i \oplus x'_i$
    $y'_i \leftarrow Y_i$; $y_i \leftarrow w_i \oplus y'_i$
    send $(x_i, y_i)$ to the adversary.
```

Note that this description is literally the same as the description of the challenger in Game 2, except that we have simply erased the underlined code in the latter.

For the purposes of analysis, we view Games 2 and 3 as operating on the same underlying probability space. This probability space is determined by

- the random choices made by the adversary, which we denote by *Coins*, and
- the random choices made by the challenger, namely, f_1, $X_1, \ldots, X_Q$, and $Y_1, \ldots, Y_Q$.

What differs between the two games is the rule that the challenger uses to compute its responses to the queries made by the adversary.

Claim 1: *in Game 3, the random variables Coins,* $f_1, x_1, y_1, \ldots, x_Q, y_Q$ *are mutually independent.* To prove this claim, observe that by construction, the random variables

$$\mathit{Coins}, \quad f_1, \quad X_1, \ldots, X_Q, \quad Y_1, \ldots, Y_Q$$

are mutually independent. Now condition on any fixed values of *Coins* and f_1. The first query (u_1, v_1) is now fixed, and hence so is w_1; however, in this conditional probability space, X_1 and Y_1 are still uniformly and independently distributed over $\mathcal{X}$, and so x_1 and y_1 are also uniformly and independently distributed. One continues the argument, conditioning on fixed values of x_1, y_1 (in addition to fixed values of *Coins* and f_1), observing that now u_2, v_2, and w_2 are also fixed, and that x_2 and y_2 are uniformly and independently distributed. It should be clear how the claim follows by induction.

Let Z_1 be the event that $w_i = w_j$ for some $i \neq j$ in Game 3. Let Z_2 be the event that $x_i = x_j$ for some $i \neq j$ in Game 3. Let $Z := Z_1 \vee Z_2$. Note that the event Z is defined in terms of the variables w_i and x_i values in Game 3. Indeed, the variables w_i and z_i may not be computed in the same way in Games 2 and 3, and so we have explicitly defined the event Z in terms of their values in Game 3. Nevertheless, it is straightforward to see that Games 2 and 3 proceed identically if Z does not occur. In particular:

Claim 2: *the event* $W_2 \wedge \bar{Z}$ *occurs if and only if the event* $W_3 \wedge \bar{Z}$ *occurs.* To prove this claim, consider any fixed values of the variables

$$\mathit{Coins}, \quad f_1, \quad X_1, \ldots, X_Q, \quad Y_1, \ldots, Y_Q$$

for which Z does not occur. It will suffice to show that the output of $\mathcal{A}_0$ is the same in both Games 2 and 3. Since the query (u_1, v_1) depends only on *Coins*, we see that the variables u_1, v_1, and hence also w_1, x_1, y_1 have the same values in both games. Since the query (u_2, v_2) depends only on *Coins* and (x_1, y_1), it follows that the variables u_2, v_2 and hence w_2 have the same values in both games; since Z does not occur, we see $w_2 \neq w_1$ and hence the variable x_2 has the same value in both games; again, since Z does not occur, it follows that $x_2 \neq x_1$, and hence the variable y_2 has the same value in both games. Continuing this argument, we see that for $i = 1, \ldots, Q$, the variables u_i, v_i, w_i, x_i, y_i have the same values in both games. Since the output of $\mathcal{A}_0$ is a function of these variables and *Coins*, the output is the same in both games. That proves the claim.

Claim 2, together with the Difference Lemma (i.e., Theorem 4.7) and the Union Bound, implies

$$|\Pr[W_3] - \Pr[W_2]| \leq \Pr[Z] \leq \Pr[Z_1] + \Pr[Z_2]. \tag{4.27}$$

By the fact that $x_1, \ldots, x_Q$ are mutually independent (see Claim 1), it is obvious that

$$\Pr[Z_2] \leq \frac{Q^2}{2} \cdot \frac{1}{N}, \tag{4.28}$$

since Z_2 is the union of less than $Q^2/2$ events, each of which occurs with probability $1/N$.

Let us now analyze the event Z_1. We claim that

$$\Pr[Z_1] \leq \frac{Q^2}{2} \cdot \frac{1}{N}. \tag{4.29}$$

To prove this, it suffices to prove it conditioned on any fixed values of $\mathit{Coins}, x_1, y_1, \ldots, x_Q, y_Q$. If these values are fixed, then so are $u_1, v_1, \ldots, u_Q, v_Q$. However, by independence (see Claim 1), the variable f_1 is still uniformly distributed over $\text{Funs}[\mathcal{X}, \mathcal{X}]$ in this conditional probability space. Now consider any fixed pair of indices i, j, with $i \neq j$. Suppose first that $v_i = v_j$. Then since $\mathcal{A}_0$ never makes the same query twice, we must have $u_i \neq u_j$, and it is easy to see that $w_i \neq w_j$ for any choice of f_1. Next suppose that $v_i \neq v_j$. Then the values $f_1(v_i)$ and $f_1(v_j)$ are uniformly and independently distributed over $\mathcal{X}$ in this conditional probability space, and

$$\Pr[f_1(v_i) \oplus f_1(v_j) = u_i \oplus u_j] = \frac{1}{N}$$

in this conditional probability space.

Thus, we have shown that in Game 3, for all pairs i, j with $i \neq j$,

$$\Pr[w_i = w_j] \leq \frac{1}{N}$$

The inequality (4.29) follows from the Union Bound.

As another consequence of Claim 1, we observe that Game 3 is equivalent to Experiment 1 of Attack Game 4.2 with respect to E. From this, together with (4.24), (4.25), (4.26), (4.27), (4.28), and (4.29), we conclude that

$$\text{PRF}\mathsf{adv}[\mathcal{A}_0, E] \leq 3 \cdot \text{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{Q^2}{N}.$$

Finally, applying Theorem 4.4 to the cipher $\mathcal{E}$, whose data block space has size N^2, we have

$$\text{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq 3 \cdot \text{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{Q^2}{N} + \frac{Q^2}{2N^2}.$$

That concludes the proof of the theorem. □

4.6 The tree construction: from PRGs to PRFs

It turns out that given a suitable, secure PRG, one can construct a secure PRF with a technique called the **tree construction**. Combining this result with the Luby-Rackoff construction in Section 4.5, we see that from any secure PRG, we can construct a secure block cipher. While this result is of some theoretical interest, the construction is not very efficient, and is not really used in practice. However, we note that a simple generalization of this construction plays an important role in practical schemes for message authentication; we shall discuss this in Section 6.4.2.

Our starting point is a PRG G defined over $(\mathcal{S}, \mathcal{S}^2)$; that is, the seed space is a set $\mathcal{S}$, and the output space is the set $\mathcal{S}^2$ of all seed pairs. For example, G might stretch n-bit strings to $2n$-bit strings.[2] It will be convenient to write $G(s) = (G_0(s), G_1(s))$; that is, $G_0(s) \in \mathcal{S}$ denotes the first component of $G(s)$ and $G_1(s)$ denotes the second component of $G(s)$. From G, we shall build a PRF F with key space $\mathcal{S}$, input space $\{0,1\}^\ell$ (where ℓ is an arbitrary, poly-bounded value), and output space $\mathcal{S}$.

Let us first define the algorithm G^*, that takes as input $s \in \mathcal{S}$ and $x = (a_1, \ldots, a_n) \in \{0,1\}^n$, where $a_i \in \{0,1\}$ for $i = 1, \ldots, n$, and outputs an element $t \in \mathcal{S}$, computed as follows:

[2] Indeed, we could even start with a PRG that stretches n bit strings to $(n+1)$-bit strings, and then apply the n-wise sequential construction analyzed in Theorem 3.3 to obtain a suitable G.

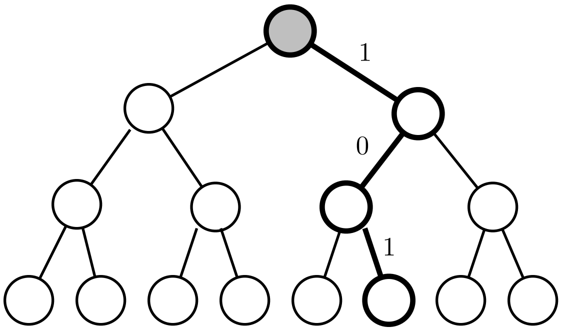


Figure 4.15: Evaluation tree for $\ell = 3$. The highlighted path corresponds to the input $x = 101$. The root is shaded to indicate it is assigned a random label. All other nodes are assigned derived labels.

```
$t \leftarrow s$
for $i \leftarrow 1$ to $n$ do
        $t \leftarrow G_{a_i}(t)$
output $t$.
```

For $s \in \mathcal{S}$ and $x \in \{0,1\}^\ell$, we define

$$F(s,x) := G^*(s,x).$$

We shall call the PRF F derived from G in this way the **tree construction**.

It is useful to envision the bits of an input $x \in \{0,1\}^\ell$ as tracing out a path through a complete binary tree of height ℓ and with 2^ℓ leaves, which we call the **evaluation tree**: a bit value of 0 means branch left and a bit value of 1 means branch right. In this way, any node in the tree can be uniquely addressed by a bit string of length at most ℓ; strings of length $j \leq \ell$ address nodes at level j in the tree: the empty string addresses the root (which is at level 0), strings of length 1 address the children of the root (which are at level 1), etc. The nodes in the evaluation tree are labeled with elements of $\mathcal{S}$, using the following rule:

- the root of the tree is labeled with s;
- the label of any other node is **derived** from the label t of its parent as follows: if the node is a left child, its label is $G_0(t)$, and if the node is a right child, its label is $G_1(t)$.

The value of $F(s,x)$ is then the label on the leaf addressed by x. See Fig. 4.15.

Theorem 4.10. *If G is a secure PRG, then the PRF F obtained from G using the tree construction is a secure PRF.*

> *In particular, for every PRF adversary $\mathcal{A}$ that plays Attack Game 4.2 with respect to F, and which makes at most Q queries to its challenger, there exists a PRG adversary $\mathcal{B}$ that plays Attack Game 3.1 with respect to G, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{PRF}\mathsf{adv}[\mathcal{A},F] = \ell Q \cdot \mathrm{PRG}\mathsf{adv}[\mathcal{B},G].$$

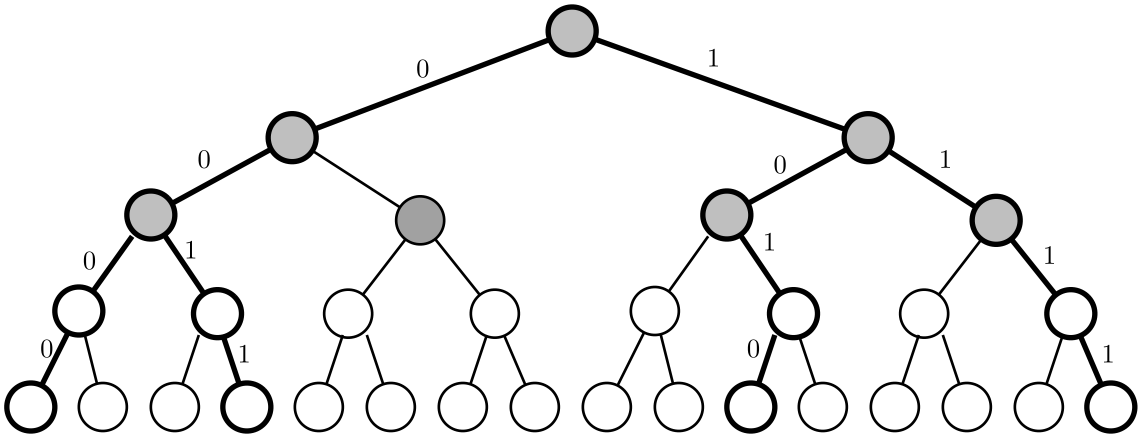


Figure 4.16: Evaluation tree for Hybrid 2 with $\ell = 4$. The shaded nodes are assigned random labels, while the unshaded nodes are assigned derived labels. The highlighted paths correspond to inputs 0000, 0011, 1010, and 1111.

Proof idea. The basic idea of the proof is a hybrid argument. We build a sequence of games, Hybrid 0, ..., Hybrid ℓ. Each of these games is played between a given PRF adversary, attacking F, and a challenger whose behavior is slightly different in each game. In Hybrid j, the challenger builds an evaluation tree whose nodes are labeled as follows:

- nodes at levels 0 through j are assigned random labels;
- the nodes at levels $j + 1$ through ℓ are assigned derived labels.

In response to a query $x \in \{0, 1\}^\ell$ in Hybrid j, the challenger sends to the adversary the label of the leaf addressed by x. See Fig. 4.16

Clearly, Hybrid 0 is equivalent to Experiment 0 of Attack Game 4.2, while Hybrid ℓ is equivalent to Experiment 1. Intuitively, under the assumption that G is a secure PRG, the adversary should not be able to tell the difference between Hybrids j and $j + 1$ for $j = 0, \ldots, \ell - 1$. In making this intuition rigorous, we have to be a bit careful: the evaluation tree is huge, and to build an efficient PRG adversary that attacks G, we cannot afford to write down the entire tree (or even one level of the tree). Instead, we use the fact that if the PRF adversary makes at most Q queries to its challenger (which is a poly-bounded value), then at any level j in the evaluation tree, the paths traced out by these Q queries touch at most Q nodes at level j (in Fig. 4.16, these would be the first, third, and fourth nodes at level 2 for the given inputs). The PRG adversary we construct will use a variation of the faithful gnome idea to effectively maintain the relevant random labels at level j, as needed. □

Proof. Let $\mathcal{A}$ be an efficient adversary that plays Attack Game 4.2 with respect to F. Let us assume that $\mathcal{A}$ makes at most a poly-bounded number Q of queries to the challenger.

As discussed above, we define $\ell + 1$ hybrid games, Hybrid 0, ..., Hybrid ℓ, each played between $\mathcal{A}$ and a challenger. In Hybrid j, the challenger works as follows:

```
$f \xleftarrow{R} \text{Funs}[\{0,1\}^j, \mathcal{S}]$
upon receiving a query $x = (a_1, \ldots, a_\ell) \in \{0,1\}^\ell$ from $\mathcal{A}$ do:
        $u \leftarrow (a_1, \ldots, a_j)$, $v \leftarrow (a_{j+1}, \ldots, a_\ell)$
        $y \leftarrow G^*(f(u), v)$
        send $y$ to $\mathcal{A}$.
```

Intuitively, for $u \in \{0,1\}^j$, $f(u)$ represents the random label at the node at level j addressed by u. Thus, each node at level j is assigned a random label, while nodes at levels $j+1$ through ℓ are assigned derived labels. Note that in our description of this game, we do not explicitly assign labels to nodes at levels 0 through $j-1$, as these labels do not affect any outputs.

For $j = 0, \ldots, \ell$, let p_j be the probability that $\mathcal{A}$ outputs 1 in Hybrid j. As Hybrid 0 is equivalent to Experiment 0 of Attack Game 4.2, and Hybrid ℓ is equivalent to Experiment 1, we have:

$$\text{PRF}\mathsf{adv}[\mathcal{A}, F] = |p_\ell - p_0|. \tag{4.30}$$

Let G' denote the Q-wise parallel composition of G, which we discussed in Section 3.4.1. G' takes as input $(s_1, \ldots, s_Q) \in \mathcal{S}^Q$ and outputs $(G(s_1), \ldots, G(s_Q)) \in (\mathcal{S}^2)^Q$. By Theorem 3.2, if G is a secure PRG, then so is G'.

We now build an efficient PRG adversary $\mathcal{B}'$ that attacks G', such that

$$\text{PRG}\mathsf{adv}[\mathcal{B}', G'] = \frac{1}{\ell} \cdot |p_\ell - p_0|. \tag{4.31}$$

We first give an overview of how $\mathcal{B}'$ works. In playing Attack Game 3.1 with respect to G', the challenger presents to $\mathcal{B}'$ a vector

$$\vec{r} = ((r_{10}, r_{11}), \ldots, (r_{Q0}, r_{Q1})) \in (\mathcal{S}^2)^Q. \tag{4.32}$$

In Experiment 0 of the attack game, $\vec{r} = G(\vec{s})$ for random $\vec{s} \in \mathcal{S}^Q$, while in Experiment 1, $\vec{r}$ is randomly chosen from $(\mathcal{S}^2)^Q$. To distinguish these two experiments, $\mathcal{B}'$ plays the role of challenger to $\mathcal{A}$ by choosing $\omega \in \{1, \ldots, \ell\}$ at random, and uses the elements of $\vec{r}$ to label nodes at level ω of the evaluation tree in a consistent fashion. To do this, $\mathcal{B}'$ maintains a lookup table, which allows it to associate with each prefix $u \in \{0,1\}^{\omega-1}$ of some query $x \in \{0,1\}^\ell$ an index p, so that the children of the node addressed by u are labeled by the seed pair (r_{p0}, r_{p1}). Finally, when $\mathcal{A}$ terminates and outputs a bit, $\mathcal{B}'$ outputs the same bit. As will be evident from the details of the construction of $\mathcal{B}'$, conditioned on $\omega = j$ for any fixed $j = 1, \ldots, \ell$, the probability that $\mathcal{B}'$ outputs 1 is:

- p_{j-1}, if $\mathcal{B}'$ is in Experiment 0 of its attack game, and
- p_j, if $\mathcal{B}'$ is in Experiment 1 of its attack game.

Then by the usual telescoping sum calculation, we get (4.31).

Now the details. We implement our lookup table as an associative array $\mathit{Map} : \{0,1\}^* \to \mathbb{Z}_{>0}$. Here is the logic for $\mathcal{B}'$:

```
upon receiving $\vec{r}$ as in (4.32) from its challenger, $\mathcal{B}'$ plays the role of challenger to $\mathcal{A}$, as
follows:
```

```
$\omega \xleftarrow{\text{R}} \{1, \ldots, \ell\}$
initialize an empty associative array $Map : \{0,1\}^* \to \mathbb{Z}_{>0}$
$ctr \leftarrow 0$
upon receiving a query $x = (a_1, \ldots, a_\ell) \in \{0,1\}^\ell$ from $\mathcal{A}$ do:
    $u \leftarrow (a_1, \ldots, a_{\omega-1})$, $d \leftarrow a_\omega$, $v \leftarrow (a_{\omega+1}, \ldots, a_\ell)$
    if $u \notin \text{Domain}(Map)$ then
        $ctr \leftarrow ctr + 1$, $Map[u] \leftarrow ctr$
    $p \leftarrow Map[u]$, $y \leftarrow G^*(r_{pd}, v)$
    send $y$ to $\mathcal{A}$.
```

Finally, $\mathcal{B}'$ outputs whatever $\mathcal{A}$ outputs.

For $b = 0, 1$, let W_b be the event that $\mathcal{B}'$ outputs 1 in Experiment b of Attack Game 3.1 with respect to G'. We claim that for any fixed $j = 1, \ldots, \ell$, we have

$$\Pr[W_0 \mid \omega = j] = p_{j-1} \quad \text{and} \quad \Pr[W_1 \mid \omega = j] = p_j.$$

Indeed, condition on $\omega = j$ for fixed j, and consider how $\mathcal{B}'$ labels nodes in the evaluation tree. On the one hand, when $\mathcal{B}'$ is in Experiment 1 of its attack game, it effectively assigns random labels to nodes at level j, and the lookup table ensures that this is done consistently. On the other hand, when $\mathcal{B}'$ is in Experiment 0 of its attack game, it effectively assigns pseudo-random labels to nodes at level j, which is the same as assigning random labels to the parents of these nodes at level $j-1$, and assigning derived labels at level j; again, the lookup table ensures a consistent labeling.

From the above claim, equation (4.31) now follows by a familiar, telescoping sum calculation:

$$\begin{aligned}
\text{PRG}\mathsf{adv}[\mathcal{B}', G'] &= \big|\Pr[W_1] - \Pr[W_0]\big| \\
&= \Big|\sum_{j=1}^{\ell} \Pr[W_1 \mid \omega = j] \cdot \Pr[\omega = j] - \sum_{j=1}^{\ell} \Pr[W_0 \mid \omega = j] \cdot \Pr[\omega = j]\Big| \\
&= \frac{1}{\ell} \cdot \Big|\sum_{j=1}^{\ell} \Pr[W_1 \mid \omega = j] - \sum_{j=1}^{\ell} \Pr[W_0 \mid \omega = j]\Big| \\
&= \frac{1}{\ell} \cdot \Big|\sum_{j=1}^{\ell} p_j - \sum_{j=1}^{\ell} p_{j-1}\Big| \\
&= \frac{1}{\ell} \cdot |p_\ell - p_0|.
\end{aligned}$$

Finally, by Theorem 3.2, there exists an efficient PRG adversary $\mathcal{B}$ such that

$$\text{PRG}\mathsf{adv}[\mathcal{B}', G'] = Q \cdot \text{PRG}\mathsf{adv}[\mathcal{B}, G]. \tag{4.33}$$

The theorem now follows by combining equations (4.30), (4.31), and (4.33). □

4.6.1 Variable length tree construction

It is natural to consider how the tree construction works on *variable length* inputs. Again, let G be a PRG defined over $(\mathcal{S}, \mathcal{S}^2)$, and let G^* be as defined above. For any poly-bounded value ℓ we define the PRF $\tilde{F}$, with key space $\mathcal{S}$, input space $\{0,1\}^{\leq \ell}$, and output space $\mathcal{S}$, as follows: for $s \in \mathcal{S}$ and $x \in \{0,1\}^{\leq \ell}$, we define

$$\tilde{F}(s,x) = G^*(s,x).$$

Unfortunately, $\tilde{F}$ is not a secure PRF. The reason is that there is a trivial **extension attack**. Suppose $u, v \in \{0,1\}^{\leq \ell}$ such that u is a proper prefix of v; that is, $v = u \parallel w$ for some non-empty string w. Then given u and v, along with $y := \tilde{F}(s,u)$, we can easily compute $F(s,v)$ as $G^*(y,w)$. Of course, for a truly random function, we could not predict its value at v, given its value at u, and so it is easy to distinguish $\tilde{F}(s,\cdot)$ from a random function.

Even though $\tilde{F}$ is not a secure PRF, we can still say something interesting about it. We show that $\tilde{F}$ is a PRF against restricted set of adversaries called **prefix-free adversaries**.

Definition 4.5. *Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$. We say that a PRF adversary $\mathcal{A}$ playing Attack Game 4.2 with respect to F is a **prefix-free adversary** if all of its queries are non-empty strings over $\mathcal{X}$ of length at most ℓ, no one of which is a proper prefix of another.*[3] *We denote $\mathcal{A}$'s advantage in winning the game by $\mathrm{PRF^{pf}}\mathsf{adv}[\mathcal{A}, F]$. Further, let us say that F is a **prefix-free secure PRF** if $\mathrm{PRF^{pf}}\mathsf{adv}[\mathcal{A}, F]$ is negligible for all efficient, prefix-free adversaries $\mathcal{A}$.*

For example, if a prefix-free adversary issues a query for the sequence (a_1, a_2, a_3) then it cannot issue queries for (a_1) or for (a_1, a_2).

Theorem 4.11. *If G is a secure PRG, then the variable length tree construction $\tilde{F}$ derived from G is a prefix-free secure PRF.*

> *In particular, for every prefix-free adversary $\mathcal{A}$ that plays Attack Game 4.2 with respect to $\tilde{F}$, and which makes at most Q queries to its challenger, there exists a PRG adversary $\mathcal{B}$ that plays Attack Game 3.1 with respect to G, where $\mathcal{B}$ is an elementary wrapper $\mathcal{A}$, such that*
>
> $$\mathrm{PRF^{pf}}\mathsf{adv}[\mathcal{A}, \tilde{F}] = \ell Q \cdot \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G].$$

Proof. The basic idea of the proof is exactly the same as that of Theorem 4.10. We sketch here the main ideas, highlighting the differences from that proof.

Let $\mathcal{A}$ be an efficient, prefix-free adversary that plays Attack Game 4.2 with respect to $\tilde{F}$. Assume that $\mathcal{A}$ makes at most Q queries to its challenger. Moreover, it will be convenient to assume that $\mathcal{A}$ never makes the same query twice. Thus, we are assuming that $\mathcal{A}$ never makes two queries, one of which is equal to, or is a prefix of, another. The challenger in Attack Game 4.2 will not enforce this assumption — we simply assume that $\mathcal{A}$ is playing by the rules.

As before, we view the evaluation of $\tilde{F}(s,\cdot)$ in terms of an evaluation tree: the root is labeled by s, and the labels on all other nodes are assigned derived labels. The only difference now is that inputs to $\tilde{F}(s,\cdot)$ may address *internal* nodes of the evaluation tree. However, the prefix-freeness restriction means that no input can address a node that is an ancestor of a node addressed by a different input.

We again define hybrid games, Hybrid 0, ..., Hybrid ℓ. In these games, the challenger uses an evaluation tree labeled in exactly the same way as in the proof of Theorem 4.10: in Hybrid j, nodes at levels 0 through j are assigned random labels, and nodes at other levels are assigned derived labels. The challenger responds to a query x by returning the label of the node in the tree addressed by x, which need not be a leaf. More formally, the challenger in Hybrid j works as follows:

[3] For sequences $x = (a_1 \dots a_s)$ and $y = (b_1 \dots b_t)$, if $s \leq t$ and $a_i = b_i$ for $i = 1, \dots, s$, then we say that x is a **prefix** of y; moreover, if $s < t$, then we say x is a **proper prefix** of y.

```
$f \xleftarrow{\text{R}} \text{Funs}[\{0,1\}^{\leq j}, \mathcal{S}]$
upon receiving a query $x = (a_1, \ldots, a_n) \in \{0,1\}^{\leq \ell}$ from $\mathcal{A}$ do:
    if $n < j$
        then $y \leftarrow f(x)$
        else  $u \leftarrow (a_1, \ldots, a_j)$, $v \leftarrow (a_{j+1}, \ldots, a_n)$, $y \leftarrow G^*(f(u), v)$
    send $y$ to $\mathcal{A}$.
```

For $j = 0, \ldots, \ell$, define p_j to be the probability that $\mathcal{A}$ outputs 1 in Hybrid j. As the reader may easily verify, we have

$$\text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{A}, \tilde{F}] = |p_\ell - p_0|.$$

Next, we define an efficient PRG adversary $\mathcal{B}'$ that attacks the Q-wise parallel composition G' of G, such that

$$\text{PRG}\mathsf{adv}[\mathcal{B}', G'] = \frac{1}{\ell} \cdot |p_\ell - p_0|.$$

Adversary $\mathcal{B}'$ runs as follows:

upon receiving $\vec{r}$ as in (4.32) from its challenger, $\mathcal{B}'$ plays the role of challenger to $\mathcal{A}$, as follows:

```
        $\omega \xleftarrow{\text{R}} \{1, \ldots, \ell\}$
        initialize an empty associative array $Map : \{0,1\}^* \to \mathbb{Z}_{>0}$
        $ctr \leftarrow 0$
        upon receiving a query $x = (a_1, \ldots, a_n) \in \{0,1\}^{\leq \ell}$ from $\mathcal{A}$ do:
            if $n < \omega$ then
(∗)             $y \xleftarrow{\text{R}} \mathcal{S}$
            else
                $u \leftarrow (a_1, \ldots, a_{\omega-1})$, $d \leftarrow a_\omega$, $v \leftarrow (a_{\omega+1}, \ldots, n)$
                if $u \notin \text{Domain}(Map)$ then
                    $ctr \leftarrow ctr + 1$, $Map[u] \leftarrow ctr$
                $p \leftarrow Map[u]$, $y \leftarrow G^*(r_{pd}, v)$
            send $y$ to $\mathcal{A}$.
```

Finally, $\mathcal{B}'$ outputs whatever $\mathcal{A}$ outputs.

For $b = 0, 1$, let W_b be the event that $\mathcal{B}'$ outputs 1 in Experiment b of Attack Game 4.2 with respect to G'. It is not too hard to see that for any fixed $j = 1, \ldots, \ell$, we have

$$\Pr[W_0 \mid \omega = j] = p_{j-1} \quad \text{and} \quad \Pr[W_1 \mid \omega = j] = p_j.$$

Indeed, condition on $\omega = j$ for fixed j, and consider how $\mathcal{B}'$ labels nodes in the evaluation tree. At the line marked (∗), $\mathcal{B}'$ assigns random labels to all nodes in the evaluation tree at levels 0 through $j - 1$, and the assumption that $\mathcal{A}$ never makes the same query twice guarantees that these labels are consistent (the same node does not receive two different labels at different times). Now, on the one hand, when $\mathcal{B}'$ is in Experiment 1 of its attack game, it effectively assigns random labels to nodes at level j as well, and the lookup table ensures that this is done consistently. On the other hand, when $\mathcal{B}'$ is in Experiment 0 of its attack game, it effectively assigns pseudo-random labels to nodes at level j, which is the same as assigning random labels to the parents of these nodes at level

$j-1$; the prefix-freeness assumption ensures that none of these parent nodes are inconsistently assigned random labels at the line marked $(*)$.

The rest of the proof goes through as in the proof of Theorem 4.10. □

4.7 The ideal cipher model

Block ciphers are used in a variety of cryptographic constructions. Sometimes it is impossible or difficult to prove a security theorem for some of these constructions under standard security assumptions. In these situations, a heuristic technique — called the **ideal cipher model** — is sometimes employed. Roughly speaking, in this model, the security analysis is done by treating the block cipher *as if* it were a family of random permutations. If $\mathcal{E} = (E, D)$ is a block cipher defined over $(\mathcal{K}, \mathcal{X})$, then the family of random permutations is $\{\Pi_{k}\}_{k \in \mathcal{K}}$, where each Π_{k} is a truly random permutation on $\mathcal{X}$, and the Π_{k}'s collectively are mutually independent. These random permutations are much too large to write down and cannot be used in a real construction. Rather, they are used to *model* a construction based on a real block cipher, to obtain a *heuristic* security argument for a given construction. We stress the heuristic nature of the ideal cipher model: while a proof of security in this model is better than nothing, it does not rule out an attack by an adversary that exploits the design of a particular block cipher, even one that is secure in the sense of Definition 4.1.

4.7.1 Formal definitions

Suppose we have some type of cryptographic scheme $\mathcal{S}$ whose implementation makes use of a block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{X})$. Moreover, suppose the scheme $\mathcal{S}$ evaluates E at various inputs $(k, a) \in \mathcal{K} \times \mathcal{X}$, and D at various inputs $(k, b) \in \mathcal{K} \times \mathcal{X}$, but does not look at the internal implementation of $\mathcal{E}$. In this case, we say that $\mathcal{S}$ **uses** $\mathcal{E}$ **as an oracle**.

We wish to analyze the security of $\mathcal{S}$. Let us assume that whatever security property we are interested in, say "property X," is modeled (as usual) as a game between a challenger (specific to property X) and an arbitrary adversary $\mathcal{A}$. Presumably, in responding to certain queries, the challenger computes various functions associated with the scheme $\mathcal{S}$, and these functions may in turn require the evaluation of E and/or D at certain points. This game defines an advantage $\mathsf{Xadv}[\mathcal{A}, \mathcal{S}]$, and security with respect to property X means that this advantage should be negligible for all efficient adversaries $\mathcal{A}$.

If we wish to analyze $\mathcal{S}$ in the ideal cipher model, then the attack game defining security is modified so that $\mathcal{E}$ is effectively replaced by a family of random permutations $\{\Pi_{k}\}_{k \in \mathcal{K}}$, as described above, to which both the adversary and the challenger have oracle access. More precisely, the game is modified as follows.

- At the beginning of the game, the challenger chooses $\Pi_{k} \in \text{Perms}[\mathcal{K}]$ at random, for each $k \in \mathcal{K}$.
- In addition to its standard queries, the adversary $\mathcal{A}$ may submit *ideal cipher queries.* There are two types of queries: Π-queries and Π^{-1}-queries.
  - For a Π-query, the adversary submits a pair $(k, a) \in \mathcal{K} \times \mathcal{X}$, to which the challenger responds with $\Pi_{k}(a)$.

- For a Π^{-1}-query, the adversary submits a pair $(\mathscr{k}, \mathscr{b}) \in \mathcal{K} \times \mathcal{X}$, to which the challenger responds with $\Pi_{\mathscr{k}}^{-1}(\mathscr{b})$.

  The adversary may make any number of ideal cipher queries, arbitrarily interleaved with standard queries.

- In processing standard queries, the challenger performs its computations using $\Pi_{\mathscr{k}}(\mathscr{a})$ in place of $E(\mathscr{k}, \mathscr{a})$ and $\Pi_{\mathscr{k}}^{-1}(\mathscr{b})$ in place of $D(\mathscr{k}, \mathscr{b})$.

The adversary's advantage is defined using the same rule as before, but is denoted $\mathrm{X}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ to emphasize that this is an advantage *in the ideal cipher model.* Security in the ideal cipher model means that $\mathrm{X}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ should be negligible for all efficient adversaries $\mathcal{A}$.

It is important to understand the role of the ideal cipher queries. Essentially, they model the ability of an adversary to make "offline" evaluations of E and D.

Ideal permutation model. Some constructions, like Even-Mansour (discussed below), make use of a permutation $\pi : \mathcal{X} \to \mathcal{X}$, rather than a block cipher. In the security analysis, one might heuristically model π as a random permutation Π, to which all parties in the attack game have oracle access to Π and Π^{-1}. We call this the **ideal permutation model**. One can view this as a special case of the ideal cipher model by simply defining $\Pi = \Pi_{\mathscr{k}_0}$ for some fixed, publicly available key $\mathscr{k}_0 \in \mathcal{K}$.

4.7.2 Exhaustive search in the ideal cipher model

Let (E, D) be a block cipher defined over $(\mathcal{K}, \mathcal{X})$ and let k be some random secret key in $\mathcal{K}$. Suppose an adversary is able to intercept a small number of input/output pairs (x_i, y_i) generated using k:

$$y_i = E(k, x_i) \quad \text{for all } i = 1, \ldots, Q.$$

The adversary can now recover k by trying all possible keys in $\mathscr{k} \in \mathcal{K}$ until a key $\mathscr{k}$ satisfying $y_i = E(\mathscr{k}, x_i)$ for all $i = 1, \ldots, Q$ is found. For block ciphers used in practice it is likely that this $\mathscr{k}$ is equal to the secret key k used to generate the given pairs. This **exhaustive search** over the key space recovers the block-cipher secret-key in time $O(|\mathcal{K}|)$ using a small number of input/output pairs. We analyze the number of input/output pairs needed to mount a successful attack in Theorem 4.12 below.

Exhaustive search is the simplest example of a key-recovery attack. Since we will present a number of key-recovery attacks, let us first define the key-recovery attack game in more detail. We will primarily use the key-recovery game as means of presenting attacks.

Attack Game 4.4 (key-recovery). For a given block cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{X})$, and for a given adversary $\mathcal{A}$, define the following game:

- The challenger picks a random $k \xleftarrow{\text{R}} \mathcal{K}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots,$ the ith query consists of a message $x_i \in \mathcal{M}$. The challenger, given x_i, computes $y_i \xleftarrow{\text{R}} E(k, x_i)$, and gives y_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate key $\mathscr{k} \in \mathcal{K}$.

We say that $\mathcal{A}$ wins the game if $\mathscr{k} = k$. We let $\mathrm{KR}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ denote the probability that $\mathcal{A}$ wins the game. □

The key-recovery game extends naturally to the ideal cipher model, where $E(\mathscr{k}, \mathscr{a}) = \Pi_{\mathscr{k}}(\mathscr{a})$ and $D(\mathscr{k}, \mathscr{b}) = \Pi_{\mathscr{k}}^{-1}(\mathscr{b})$, and $\{\Pi_{\mathscr{k}}\}_{\mathscr{k} \in \mathcal{K}}$ is a family of independent random permutations. In this model, we allow the adversary to make arbitrary Π- and Π^{-1}-queries, in addition to its standard queries to $E(k, \cdot)$. We let $\mathrm{KR}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ denote the adversary's key-recovery advantage when $\mathcal{E}$ is an ideal cipher.

It is worth noting that security against key-recovery attacks does not imply security in the sense of indistinguishability (Definition 4.1). The simplest example is the constant block cipher $E(k, x) = x$ for which key-recovery is not possible (the adversary obtains no information about k), but the block cipher is easily distinguished from a random permutation.

Exhaustive search. The following theorem bounds the number of input/output pairs needed for exhaustive search, assuming the cipher is an ideal cipher. For real-world parameters, taking $Q = 3$ in the theorem is often sufficient to ensure success.

Theorem 4.12. *Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$. Then there exists an adversary $\mathcal{A}_{\mathrm{EX}}$ that plays Attack Game 4.4 with respect to $\mathcal{E}$, modeled as an ideal cipher, making Q standard queries and $Q|\mathcal{K}|$ ideal cipher queries, such that*

$$\mathrm{KR}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}_{\mathrm{EX}}, \mathcal{E}] \geq (1 - \epsilon) \quad \textit{where} \quad \epsilon := \frac{|\mathcal{K}|}{(|\mathcal{X}| - Q)^Q} \tag{4.34}$$

Proof. In the ideal cipher model, we are modeling the block cipher $\mathcal{E} = (E, D)$ as a family $\{\Pi_{\mathscr{k}}\}_{\mathscr{k} \in \mathcal{K}}$ of random permutations on $\mathcal{X}$. In Attack Game 4.4, the challenger chooses $k \in \mathcal{K}$ at random. An adversary may make standard queries to obtain the value $E(k, x) = \Pi_k(x)$ at points $x \in \mathcal{X}$ of his choosing. An adversary may also make ideal cipher queries, obtaining the values $\Pi_{\mathscr{k}}(\mathscr{a})$ and $\Pi_{\mathscr{k}}^{-1}(\mathscr{b})$ for points $\mathscr{k} \in \mathcal{K}$ and $\mathscr{a}, \mathscr{b} \in \mathcal{X}$ of his choosing. These ideal cipher queries correspond to "offline" evaluations of E and D.

Our adversary $\mathcal{A}_{\mathrm{EX}}$ works as follows:

```
let $\{x_1, \ldots, x_Q\}$ be an arbitrary set of distinct messages in $\mathcal{X}$
for $i = 1, \ldots, Q$ do:
    make a standard query to obtain $y_i := E(k, x_i) = \Pi_k(x_i)$
for each $\mathscr{k} \in \mathcal{K}$ do:
    for $i = 1, \ldots, Q$ do:
        make an ideal cipher query to obtain $\mathscr{b}_i := \Pi_{\mathscr{k}}(x_i)$
    if $y_i = \mathscr{b}_i$ for all $i = 1, \ldots, Q$ then
        output $\mathscr{k}$ and terminate
```

Let k be the challenger's secret-key. We show that $\mathcal{A}_{\mathrm{EX}}$ outputs k with probability at least $1 - \epsilon$, with ϵ defined as in (4.34). Since $\mathcal{A}_{\mathrm{EX}}$ tries all keys, this amounts to showing that the probability that there is more than one key consistent with the given (x_i, y_i) pairs is at most ϵ. We shall show that this holds for every possible choice of k, so for the remainder of the proof, we shall view k as fixed. We shall also view $x_1, \ldots, x_Q$ as fixed, so all the probabilities are with respect to the random permutations $\Pi_{\mathscr{k}}$ for $\mathscr{k} \in \mathcal{K}$.

For each $\underset{\sim}{k} \in \mathcal{K}$, let $W_{\underset{\sim}{k}}$ be the event that $y_i = \Pi_{\underset{\sim}{k}}(x_i)$ for all $i = 1, \ldots, Q$. Note that by definition, W_k occurs with probability 1. Let W be the event that $W_{\underset{\sim}{k}}$ occurs for some $\underset{\sim}{k} \neq k$. We want to show that $\Pr[W] \leq \epsilon$.

Fix $\underset{\sim}{k} \neq k$. Since the permutation Π_k is chosen independently of the permutation $\Pi_{\underset{\sim}{k}}$, we know that

$$\Pr[W_{\underset{\sim}{k}}] = \frac{1}{|\mathcal{X}|} \cdot \frac{1}{|\mathcal{X}|-1} \cdots \frac{1}{|\mathcal{X}|-Q+1} \leq \left(\frac{1}{|\mathcal{X}|-Q}\right)^Q$$

As this holds for all $\underset{\sim}{k} \neq k$, the result follows from the union bound. □

4.7.2.1 Security of the $3\mathcal{E}$ construction

The attack presented in Theorem 4.2 works equally well against the $3\mathcal{E}$ construction. The size of the key space is $|\mathcal{K}|^3$, but one obtains a "meet in the middle" key recovery algorithm that runs in time $O\big(|\mathcal{K}|^2 \cdot Q\big)$. For Triple-DES this algorithm requires more than $2^{2\times 56}$ evaluations of Triple-DES, which is far beyond our computing power.

One wonders whether better attacks against $3\mathcal{E}$ exist. When $\mathcal{E}$ is an ideal cipher we can prove a lower bound on the amount of work needed to distinguish $3\mathcal{E}$ from a random permutation.

Theorem 4.13. *Let $\mathcal{E} = (E, D)$ be an ideal block cipher defined over $(\mathcal{K}, \mathcal{X})$, and consider an attack against the $3\mathcal{E}$ construction in the ideal cipher model. If $\mathcal{A}$ is an adversary that makes at most Q queries (including both standard and ideal cipher queries) in the ideal cipher variant of Attack Game 4.1, then*

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, 3\mathcal{E}] \leq C_1 L \frac{Q^2}{|\mathcal{K}|^3} + C_2 \frac{Q^{2/3}}{|\mathcal{K}|^{2/3}|\mathcal{X}|^{1/3}} + C_3 \frac{1}{|\mathcal{K}|},$$

where $L := \max(|\mathcal{K}|/|\mathcal{X}|, \log_2|\mathcal{X}|)$, and C_1, C_2, C_3 are constants (that do not depend on $\mathcal{A}$ or $\mathcal{E}$).

The statement of the theorem is easier to understand if we assume that $|\mathcal{K}| \leq |\mathcal{X}|$, as is the case with DES. In this case, the bound can be restated as

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, 3\mathcal{E}] \leq C \log_2 |\mathcal{X}| \frac{Q^2}{|\mathcal{K}|^3},$$

for a constant C. Ignoring the $\log \mathcal{X}$ term, this says that an adversary must make roughly $|\mathcal{K}|^{1.5}$ queries to obtain a significant advantage (say, 1/4). Compare this to the meet-in-the-middle attack. To achieve a significant advantage, that adversary must make roughly $|\mathcal{K}|^2$ queries. Thus, meet-in-the-middle attack may not be the most powerful attack.

To conclude our discussion of Triple-DES, we note that the $3\mathcal{E}$ construction does not always strengthen the cipher. For example, if $\mathcal{E} = (E, D)$ is such that the set of $|\mathcal{K}|$ permutations $\{E(\underset{\sim}{k}, \cdot) : \underset{\sim}{k} \in \mathcal{K}\}$ is a group, then $3\mathcal{E}$ would be no more secure than $\mathcal{E}$. Indeed, in this case $\pi := E_3((k_1, k_2, k_3), \cdot)$ is identical to $E(k, \cdot)$ for some $k \in \mathcal{K}$. Consequently, distinguishing $3\mathcal{E}$ from a random permutation is no harder than doing so for $\mathcal{E}$. Of course, block ciphers used in practice are not groups (as far as we know).

4.7.3 The Even-Mansour block cipher and the $\mathcal{E}X$ construction

Let $\mathcal{X} = \{0,1\}^n$. Let $\pi : \mathcal{X} \to \mathcal{X}$ be a permutation and let π^{-1} be its inverse function. Even and Mansour defined the following simple block cipher $\mathcal{E}_{EM} = (E, D)$ defined over $(\mathcal{X}^2, \mathcal{X})$:

$$E\big((P_1, P_2),\ x\big) := \pi(x \oplus P_1) \oplus P_2 \qquad \text{and} \qquad D\big((P_1, P_2),\ y\big) := \pi^{-1}(y \oplus P_2) \oplus P_1 \tag{4.35}$$

How do we analyze the security of this block cipher? Clearly for some π's this construction is insecure, for example when π is the identity function. For what π is $\mathcal{E}_{EM}$ a secure block cipher?

The only way we know to analyze security of $\mathcal{E}_{EM}$ is by modeling π as a random permutation Π on the set $\mathcal{X}$ (i.e., in the ideal cipher model using a fixed key). We show in Theorem 4.14 below that in the ideal cipher model, for all adversaries $\mathcal{A}$:

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{E}_{EM}, \mathcal{A}] \le \frac{2Q_{\mathrm{s}}Q_{\mathrm{ic}}}{|\mathcal{X}|} \tag{4.36}$$

where Q_{s} is the number of queries $\mathcal{A}$ makes to $\mathcal{E}_{EM}$ and Q_{ic} is the number of queries $\mathcal{A}$ makes to Π and Π^{-1}. Hence, the Even-Mansour block cipher is secure (in the ideal cipher model) whenever $|\mathcal{X}|$ is sufficiently large. Exercise 4.21 shows that the bound (4.36) is tight. We discuss more attacks on Even-Mansour in Exercise 18.20.

The Even-Mansour security theorem (Theorem 4.14) does not require the keys P_1 and P_2 to be independent. In fact, the bounds in (4.36) remain unchanged if we set $P_1 = P_2$ so that the key for $\mathcal{E}_{EM}$ is a single element of $\mathcal{X}$. However, we note that if one leaves out either of P_1 or P_2, the construction is completely insecure (see Exercise 4.20).

Iterated Even-Mansour and AES. Looking back at our description of AES (Fig. 4.11) one observes that the Even-Mansour cipher looks a lot like one round of AES where the round function Π_{AES} plays the role of π. Of course one round of AES is not a secure block cipher: the bound in (4.36) does not imply security because Π_{AES} is not a random permutation.

Suppose one replaces each occurrence of Π_{AES} in Fig. 4.11 by a different permutation: one function for each round of AES. The resulting structure, called **iterated Even-Mansour**, can be analyzed in the ideal cipher model and the resulting security bounds are better than those stated in (4.36).

These results suggest a theoretical justification for the AES structure in the ideal cipher model.

The $\mathcal{E}X$ construction and DESX. If we apply the Even-Mansour construction to a full-fledged block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{X})$, we obtain a new block cipher called $\mathcal{E}X = (EX, DX)$ where

$$EX\big((k, P_1, P_2),\ x\big) := E(k,\ x \oplus P_1) \oplus P_2\ , \qquad DX\big((k, P_1, P_2),\ y\big) := D(k,\ y \oplus P_2) \oplus P_1. \tag{4.37}$$

This new cipher $\mathcal{E}X$ has a key space $\mathcal{K} \times \mathcal{X}^2$ which can be much larger than the key space for the underlying cipher $\mathcal{E}$.

Theorem 4.14 below shows that — in the ideal cipher model — this larger key space translates to better security: the maximum advantage against $\mathcal{E}X$ is much smaller than the maximum advantage against $\mathcal{E}$, whenever $|\mathcal{X}|$ is sufficiently large.

Applying $\mathcal{E}X$ to the DES block cipher gives an efficient method to immunize DES against exhaustive search attacks. With $P_1 = P_2$ we obtain a block cipher called **DESX** whose key size is $56 + 64 = 120$ bits: enough to resist exhaustive search. Theorem 4.14 shows that attacks in the ideal cipher model on the resulting cipher are impractical. Since evaluating DESX requires only one call to DES, the DESX block cipher is three times faster than the Triple-DES block cipher and this makes it seem as if DESX is the preferred way to strengthen DES. However, non black-box attacks like differential and linear cryptanalysis still apply to DESX whereas they are ineffective against Triple-DES. Consequently, DESX should not be used in practice.

4.7.4 Proof of the Even-Mansour and $\mathcal{E}X$ theorems

We shall prove security of the Even-Mansour block cipher (4.35) in the ideal permutation model and of the $\mathcal{E}X$ construction (4.37) in the ideal cipher model.

We prove their security in a single theorem below. Taking a single-key block cipher (i.e., $|\mathcal{K}| = 1$) proves security of Even-Mansour in the ideal permutation model. Taking a block cipher with a larger key space proves security of $\mathcal{E}X$. Note that the pads P_1 and P_2 need not be independent and the theorem holds if we set $P_2 = P_1$.

Theorem 4.14. *Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{X})$. Let $\mathcal{E}X = (EX, DX)$ be the block cipher derived from $\mathcal{E}$ as in construction (4.37), where P_1 and P_2 are each uniformly distributed over a subset $\mathcal{X}'$ of $\mathcal{X}$. If we model $\mathcal{E}$ as an ideal cipher, and if $\mathcal{A}$ is an adversary in Attack Game 4.1 for $\mathcal{E}X$ that makes at most Q_{s} standard queries (i.e., EX-queries) and Q_{ic} ideal cipher queries (i.e., Π- or Π^{-1}-queries), then we have*

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{E}X] \le \frac{2Q_{\mathrm{s}}Q_{\mathrm{ic}}}{|\mathcal{K}||\mathcal{X}'|}. \quad \Box \tag{4.38}$$

To understand the security benefit of the $\mathcal{E}X$ construction consider the following: modeling $\mathcal{E}$ as an ideal cipher gives $\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \le Q_{\mathrm{ic}}/|\mathcal{K}|$ for all $\mathcal{A}$. Hence, Theorem 4.14 shows that, in the ideal cipher model, applying $\mathcal{E}X$ to $\mathcal{E}$ shrinks the maximum advantage by a factor of $2Q_{\mathrm{s}}/|\mathcal{X}'|$.

The bounds in Theorem 4.14 are tight: there is an adversary $\mathcal{A}$ that achieves the advantage shown in (4.38); see Exercise 4.21. The advantage of this $\mathcal{A}$ is unchanged even when P_1 and P_2 are chosen independently. Therefore, we might as well always choose $P_2 = P_1$.

We also note that it is actually no harder to prove that $\mathcal{E}X$ is a *strongly secure* block cipher (see Section 4.1.3) in the ideal cipher model, with exactly the same security bounds as in Theorem 4.14.

Proof idea. The basic idea is to show that the ideal cipher queries and the standard queries do not interact with each other, except with probability as bounded in (4.38). Indeed, to make the two types of queries interact with each other, the adversary has to make

$$(\mathscr{k} = k \text{ and } \mathscr{a} = x \oplus P_1) \ \text{ or } \ (\mathscr{k} = k \text{ and } \mathscr{b} = y \oplus P_2)$$

for some input/output pair (x, y) corresponding to a standard query and some input/output triple $(\mathscr{k}, \mathscr{a}, \mathscr{b})$ corresponding to an ideal cipher query. Essentially, the adversary will have to simultaneously guess the random key k as well as one of the random pads P_1 or P_2.

Assuming there are no such interactions, we can effectively realize all of the standard queries as $\Pi(x \oplus P_1) \oplus P_2$ using a random permutation Π that is independent of the random permutations used to realize the ideal cipher queries. But $\Pi'(x) := \Pi(x \oplus P_1) \oplus P_2$ is just a random permutation.

Before giving a rigorous proof of Theorem 4.14, we present a technical lemma, called the **Domain Separation Lemma**, that will greatly simplify the proof, and is useful in analyzing other constructions.

To motivate the lemma, consider the following two experiments. In the one experiment, called the "split experiment", an adversary has oracle access to two random permutations Π_1, Π_2 on a set $\mathcal{X}$. The adversary can make a series of queries, each of the form (μ, d, z), where $\mu \in \{1, 2\}$ specifies which of the two permutations to evaluate, $d \in \{\pm 1\}$ specifies the direction to evaluate the permutation, and $z \in \mathcal{X}$ the input to the permutation. On such a query, the challenger responds with $z' := \Pi_\mu^d(z)$. Another experiment, called the "coalesced experiment", is exactly the same as the split experiment, except that there is only a single permutation Π, and the challenger answers the query (μ, d, z) with $z' := \Pi^d(z)$, ignoring completely the index μ. The question is: under what condition can the adversary distinguish between these two experiments?

Obviously, if the adversary can submit a query $(1, +1, a)$ and a query $(2, +1, a)$, then in the split experiment, the results will almost certainly be different, while in the coalesced experiment, they will surely be the same. Another type of attack is possible as well: the adversary could make a query $(1, +1, a)$ obtaining b, and then submit the query $(2, -1, b)$, obtaining a'. In the split experiment, a and a' will almost certainly be different, while in the coalesced experiment, they will surely be the same. Besides these two examples, one could get two more examples which reverse the direction of all the queries. The Domain Separation Lemma will basically say that unless the adversary makes queries of one of these four types, he cannot distinguish between these two experiments.

Of course, the Domain Separation Lemma is only useful in contexts where the adversary is somehow constrained so that he cannot freely make queries of his choice. Indeed, we will only use it inside of the proof of a security theorem where the "adversary" in the Domain Separation Lemma comprises components of a challenger and an adversary in a more interesting attack game.

In the more general statement of the lemma, we replace Π_1 and Π_2 by a family of permutations of permutations $\{\Pi_\mu\}_{\mu \in U}$, and we replace Π by a family $\{\overline{\Pi}_\nu\}_{\nu \in V}$. We also introduce a function $f : U \to V$ that specifies how several permutations in the split experiment are collapsed into one permutation in the coalesced experiment: for each $\nu \in V$, all the permutations Π_μ in the split experiment for which $f(\mu) = \nu$ are collapsed into the single permutation $\overline{\Pi}_\nu$ in the coalesced experiment.

In the generalized version of the distinguishing game, if the adversary makes a query (μ, d, z), then in the split experiment, the challenger responds with $z' := \Pi_\mu^d(z)$, while in the coalesced experiment, the challenger responds with $z' := \Pi_{f(\mu)}^d(z)$. In the split experiment, we also keep track of the subset of the domains and ranges of the permutations that correspond to actual queries made by the adversary in the split experiment. That is, we build up sets $\mathrm{Dom}_\mu^{(d)}$ for each $\mu \in U$ and $d \in \pm 1$, so that $a \in \mathrm{Dom}_\mu^{(+1)}$ if and only if the adversary issues a query of the form $(\mu, +1, a)$ or a query of the form $(\mu, -1, b)$ that yields a. Similarly, $b \in \mathrm{Dom}_\mu^{(-1)}$ if and only if the adversary issues a query of the form $(\mu, -1, b)$ or a query of the form $(\mu, +1, a)$ that yields b. We call $\mathrm{Dom}_\mu^{(+1)}$ the **sampled domain** of Π_μ and $\mathrm{Dom}_\mu^{(-1)}$ the **sampled range** of Π_μ.

Attack Game 4.5 (domain separation). Let $U, V, \mathcal{X}$ be finite, nonempty sets, and let $f : U \to V$ be a function. For a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define:

Experiment b:

- For each $\mu \in U$, and each $\nu \in V$ the challenger sets $\Pi_\mu \xleftarrow{\text{R}} \mathrm{Perms}[\mathcal{X}]$ and $\overline{\Pi}_\nu \xleftarrow{\text{R}} \mathrm{Perms}[\mathcal{X}]$ Also, for each $\mu \in U$ and $d \in \{\pm 1\}$ the challenger sets $\mathrm{Dom}_\mu^{(d)} \leftarrow \emptyset$.

- The adversary submits a sequence of queries to the challenger.

For $i = 1, 2, \ldots$, the ith query is $(\mu_i, d_i, z_i) \in U \times \{\pm 1\} \times \mathcal{X}$.

If $b = 0$: the challenger sets $z'_i \leftarrow \overline{\Pi}^{d_i}_{f(\mu_i)}(z_i)$.

If $b = 1$: the challenger sets $z'_i \leftarrow \Pi^{d_i}_{\mu_i}(z_i)$; the challenger also adds the value z_i to the set $\mathrm{Dom}^{(d_i)}_{\mu_i}$, and adds the value z'_i to the set $\mathrm{Dom}^{(-d_i)}_{\mu_i}$.

In either case, the challenger then sends z'_i to the adversary.

- Finally, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **domain separation distinguishing advantage** as $|\Pr[W_0] - \Pr[W_1]|$. We also define the **domain separation failure event** Z to be the event that *in Experiment 1*, at the end of the game we have $\mathrm{Dom}^{(d)}_{\mu} \cap \mathrm{Dom}^{(d)}_{\mu'} \neq \emptyset$ for some $d \in \{\pm 1\}$ and some pair of *distinct* indices $\mu, \mu' \in U$ with $f(\mu) = f(\mu')$. Finally, we define the **domain separation failure probability** to be $\Pr[Z]$. □

Experiment 1 in the above game is the split experiment and Experiment 0 is the coalesced experiment.

Theorem 4.15 (Domain Separation Lemma). *In Attack Game 4.5, an adversary's domain separation distinguishing advantage is bounded by the domain separation failure probability.*

In applying the Domain Separation Lemma, we will typically analyze some attack game in which permutations start out as coalesced, and then force them to be separated. We can bound the impact of this change on the outcome of the attack by analyzing the domain separation failure probability in the attack game with the split permutations.

Before proving the Domain Separation Lemma, it is perhaps more instructive to see how it is used in the proof of Theorem 4.14.

Proof of Theorem 4.14. Let $\mathcal{A}$ be an adversary as in the statement of the theorem. For $b = 0, 1$ let p_b be the probability that $\mathcal{A}$ outputs 1 in Experiment b of the block cipher attack game in the ideal cipher model (Attack Game 4.1). So by definition we have

$$\mathrm{BC^{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{E}X] = |p_0 - p_1|. \tag{4.39}$$

We shall prove the theorem using a sequence of two games, applying the Domain Separation Lemma.

Game 0. We begin by describing Game 0, which corresponds to Experiment 0 of the block cipher attack game in the ideal cipher model. Recall that in this model, we have a family of random permutations, and the encryption function is implemented in terms of this family. Also recall that in addition to standard queries that probe the function $E_k(\cdot)$, the adversary may also probe the random permutations.

```
Initialize:
    for each $\underset{\sim}{k} \in \mathcal{K}$, set $\Pi_{\underset{\sim}{k}} \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$
    $k \xleftarrow{R} \mathcal{K}$, choose $P_1, P_2$
```

```
standard EX-query $x$:
    $\mathfrak{a} \leftarrow x \oplus P_1$
    $\mathfrak{b} \leftarrow \Pi_k(\mathfrak{a})$
    $y \leftarrow \mathfrak{b} \oplus P_2$
    return $y$

ideal cipher $\Pi$-query $\mathfrak{k}, \mathfrak{a}$:
    $\mathfrak{b} \leftarrow \Pi_{\mathfrak{k}}(\mathfrak{a})$
    return $\mathfrak{b}$

ideal cipher $\Pi^{-1}$-query $\mathfrak{k}, \mathfrak{b}$:
    $\mathfrak{a} \leftarrow \Pi_{\mathfrak{k}}^{-1}(\mathfrak{b})$
    return $\mathfrak{a}$
```

Let W_0 be the event that $\mathcal{A}$ outputs 1 at the end of Game 0. It should be clear from construction that

$$\Pr[W_0] = p_0. \tag{4.40}$$

Game 1. In this game, we apply the Domain Separation Lemma. The basic idea is that we will declare "by fiat" that the random permutations used in processing the standard queries are independent of the random permutations used in processing ideal cipher queries. Effectively, each permutation $\Pi_{\mathfrak{k}}$ gets split into two independent permutations: $\Pi_{\text{std},\mathfrak{k}}$, which is used by the challenger in responding to standard *EX*-queries, and $\Pi_{\text{ic},\mathfrak{k}}$, which is used in responding to ideal cipher queries. In detail (changes from Game 0 are highlighted):

```
Initialize:
    for each $\mathfrak{k} \in \mathcal{K}$, set $\Pi_{\text{std},\mathfrak{k}} \xleftarrow{R} \text{Perms}[\mathcal{X}]$ and $\Pi_{\text{ic},\mathfrak{k}} \xleftarrow{R} \text{Perms}[\mathcal{X}]$
    $k \xleftarrow{R} \mathcal{K}$, choose $P_1, P_2$

standard EX-query $x$:
    $\mathfrak{a} \leftarrow x \oplus P_1$
    $\mathfrak{b} \leftarrow \Pi_{\text{std},k}(\mathfrak{a})$    // add $\mathfrak{a}$ to sampled domain of $\Pi_{\text{std},k}$,  add $\mathfrak{b}$ to sampled range of $\Pi_{\text{std},k}$
    $y \leftarrow \mathfrak{b} \oplus P_2$
    return $y$

ideal cipher $\Pi$-query $\mathfrak{k}, \mathfrak{a}$:
    $\mathfrak{b} \leftarrow \Pi_{\text{ic},\mathfrak{k}}(\mathfrak{a})$    // add $\mathfrak{a}$ to sampled domain of $\Pi_{\text{ic},\mathfrak{k}}$,  add $\mathfrak{b}$ to sampled range of $\Pi_{\text{ic},\mathfrak{k}}$
    return $\mathfrak{b}$

ideal cipher $\Pi^{-1}$-query $\mathfrak{k}, \mathfrak{b}$:
    $\mathfrak{a} \leftarrow \Pi_{\text{ic},\mathfrak{k}}^{-1}(\mathfrak{b})$    // add $\mathfrak{a}$ to sampled domain of $\Pi_{\text{ic},\mathfrak{k}}$,  add $\mathfrak{b}$ to sampled range of $\Pi_{\text{ic},\mathfrak{k}}$
    return $\mathfrak{a}$
```

Let W_1 be the event that $\mathcal{A}$ outputs 1 at the end of Game 1. Let Z be the event that in Game 1 there exists $\mathfrak{k} \in \mathcal{K}$, such that the sampled domains of $\Pi_{\text{ic},\mathfrak{k}}$ and $\Pi_{\text{std},\mathfrak{k}}$ overlap or the sampled ranges of $\Pi_{\text{ic},\mathpzc{k}}$ and $\Pi_{\text{std},\mathpzc{k}}$ overlap. The Domain Separation Lemma says that

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z]. \tag{4.41}$$

In applying the Domain Separation Lemma, the "coalescing function" f maps from $\{\text{std}, \text{ic}\} \times \mathcal{K}$ to $\mathcal{K}$, sending the pair $(\cdot, \mathpzc{k})$ to $\mathpzc{k}$. Observe that the challenger only makes queries to Π_k, where k is the secret key, and so such an overlap can occur only at $\mathpzc{k} = k$. Also observe that in Game 1, the random variables k, P_1, and P_2 are completely independent of the adversary's view.

So the event Z occurs if and only if for some input/output triple $(\mathpzc{k}, \mathpzc{a}, \mathpzc{b})$ triple arising from a Π- or Π^{-1}-query, and for some input/output pair (x, y) arising from an EX-query, we have

$$(\mathpzc{k} = k \text{ and } \mathpzc{a} = x \oplus P_1) \text{ or } (\mathpzc{k} = k \text{ and } \mathpzc{b} = y \oplus P_2). \tag{4.42}$$

Using the union bound, we can therefore bound $\Pr[Z]$ as a sum of probabilities of $2Q_{\text{s}}Q_{\text{ic}}$ events, each of the form $\mathpzc{k} = k$ and $\mathpzc{a} = x \oplus P_1$, or of the form $\mathpzc{k} = k$ and $\mathpzc{b} = y \oplus P_2$. By independence, since k is uniformly distributed over a set of size $|\mathcal{K}|$, and each of P_1 and P_2 is uniformly distributed over a set of size $|\mathcal{X}'|$, each such event occurs with probability at most $1/(|\mathcal{K}||\mathcal{X}'|)$. It follows that

$$\Pr[Z] \leq \frac{2Q_{\text{s}}Q_{\text{ic}}}{|\mathcal{K}||\mathcal{X}'|}. \tag{4.43}$$

Finally, observe that Game 1 is equivalent to Experiment 1 of the block cipher attack game in the ideal cipher model: the EX-queries present to the adversary the random permutation $\Pi'(x) := \Pi_{\text{std},k}(x \oplus P_1) \oplus P_2$ and this permutation is independent of the random permutations used in the Π- and Π^{-1}-queries. Thus,

$$\Pr[W_1] = p_1. \tag{4.44}$$

The bound (4.38) now follows from (4.39), (4.40), (4.41), (4.43), and (4.44). This completes the proof of the theorem. □

Finally, we turn to the proof of the Domain Separation Lemma, which is a simple (if tedious) application of the Difference Lemma and the "forgetful gnome" technique.

Proof of Theorem 4.15. We define a sequence of games.

Game 0. This game will be equivalent to the coalesced experiment in Attack Game 4.5, but designed in a way that will facilitate the analysis.

In this game, the challenger maintains various sets Π of pairs $(\mathpzc{a}, \mathpzc{b})$. Each set Π represents a function that can be extended to a permutation on $\mathcal{X}$ that sends $\mathpzc{a}$ to $\mathpzc{b}$ for every $(\mathpzc{a}, \mathpzc{b})$ in Π. We call such a set Π a *partial permutation on $\mathcal{X}$*. Define

$$\begin{aligned}\text{Domain}(\Pi) &= \{\mathpzc{a} \in \mathcal{X} : (\mathpzc{a}, \mathpzc{b}) \in \Pi \text{ for some } \mathpzc{b} \in \mathcal{X}\}\ ,\\ \text{Range}(\Pi) &= \{\mathpzc{b} \in \mathcal{X} : (\mathpzc{a}, \mathpzc{b}) \in \Pi \text{ for some } \mathpzc{a} \in \mathcal{X}\}\ .\end{aligned}$$

Also, for $\mathpzc{a} \in \text{Domain}(\Pi)$, define $\Pi(\mathpzc{a})$ to be the unique $\mathpzc{b}$ such that $(\mathpzc{a}, \mathpzc{b}) \in \Pi$. Likewise, for $\mathpzc{b} \in \text{Range}(\Pi)$, define $\Pi^{-1}(\mathpzc{b})$ to be the unique $\mathpzc{a}$ such that $(\mathpzc{a}, \mathpzc{b}) \in \Pi$.

Here is the logic of the challenger in Game 0:

```
Initialize:
    for each $\nu \in V$, initialize the partial permutation $\overline{\Pi}_\nu \leftarrow \emptyset$
```

```
Process query $(\mu, +1, a)$:
1.    if $a \in \text{Domain}(\overline{\Pi}_{f(\mu)})$ then $b \leftarrow \overline{\Pi}_{f(\mu)}(a)$, return $b$
2.    $b \stackrel{\text{R}}{\leftarrow} \mathcal{X} \setminus \text{Range}(\overline{\Pi}_{f(\mu)})$
3.    add $(a, b)$ to $\overline{\Pi}_{f(\mu)}$
4.    return $b$
```

```
Process query $(\mu, -1, b)$:
1.    if $b \in \text{Range}(\overline{\Pi}_{f(\mu)})$ then $a \leftarrow \overline{\Pi}_{f(\mu)}^{-1}(b)$, return $a$
2.    $a \stackrel{\text{R}}{\leftarrow} \mathcal{X} \setminus \text{Domain}(\overline{\Pi}_{f(\mu)})$
3.    add $(a, b)$ to $\overline{\Pi}_{f(\mu)}$
4.    return $a$
```

This game is clearly equivalent to the coalesced experiment in Attack Game 4.5. Let W_0 be the event that the adversary outputs 1 in this game.

Game 1. Now we modify this game to get an equivalent game, but it will facilitate the application of the Difference Lemma in moving to the next game. For $\mu, \mu' \in U$, let us write $\mu \sim \mu'$ if $f(\mu) = f(\mu')$. This is an equivalence relation on U, and we write $[\mu]$ for the equivalence class containing μ.

Here is the logic of the challenger in Game 1:

```
Initialize:
      for each $\mu \in U$, initialize the partial permutation $\Pi_\mu \leftarrow \emptyset$
```

```
Process query $(\mu, +1, a)$:
  1a.   if $a \in \text{Domain}(\Pi_\mu)$ then $b \leftarrow \Pi_\mu(a)$, return $b$
* 1b.   if $a \in \text{Domain}(\Pi_{\mu'})$ for some $\mu' \in [\mu]$ then $b \leftarrow \Pi_{\mu'}(a)$, return $b$
  2a.   $b \stackrel{\text{R}}{\leftarrow} \mathcal{X} \setminus \text{Range}(\Pi_\mu)$
* 2b.   if $b \in \bigcup_{\mu' \in [\mu]} \text{Range}(\Pi_{\mu'})$ then $b \stackrel{\text{R}}{\leftarrow} \mathcal{X} \setminus \bigcup_{\mu' \in [\mu]} \text{Range}(\Pi_{\mu'})$
  3.    add $(a, b)$ to $\Pi_\mu$
  4.    return $b$
```

```
Process query $(\mu, -1, b)$:
  1a.   if $b \in \text{Range}(\Pi_\mu)$ then $a \leftarrow \Pi_\mu^{-1}(b)$, return $a$
* 1b.   if $b \in \text{Range}(\Pi_{\mu'})$ for some $\mu' \in [\mu]$ then $a \leftarrow \Pi_{\mu'}^{-1}(b)$, return $a$
  2a.   $a \stackrel{\text{R}}{\leftarrow} \mathcal{X} \setminus \text{Domain}(\Pi_\mu)$
* 2b.   if $a \in \bigcup_{\mu' \in [\mu]} \text{Domain}(\Pi_{\mu'})$ then $a \stackrel{\text{R}}{\leftarrow} \mathcal{X} \setminus \bigcup_{\mu' \in [\mu]} \text{Domain}(\Pi_{\mu'})$
  3.    add $(a, b)$ to $\Pi_\mu$
  4.    return $a$
```

Let W_1 be the event that the adversary outputs 1 in this game.

It is not hard to see that the challenger's behavior in this game is equivalent to that in Game 0, and so $\Pr[W_0] = \Pr[W_1]$. The idea is that for every $\nu \in f(U) \subseteq V$, the partial permutation $\overline{\Pi}_\nu$ in Game 0 is partitioned into a family of disjoint partial permutations $\{\Pi_\mu\}_{\mu \in f^{-1}(\nu)}$, so that

$$\overline{\Pi}_\nu = \bigcup_{\mu \in f^{-1}(\nu)} \Pi_\mu,$$

and

$$\begin{aligned}&\mathrm{Domain}(\Pi_\mu) \cap \mathrm{Domain}(\Pi_{\mu'}) = \emptyset \quad \text{and} \quad \mathrm{Range}(\Pi_\mu) \cap \mathrm{Range}(\Pi_{\mu'}) = \emptyset \\ &\text{for all } \mu, \mu' \in f^{-1}(\nu) \text{ with } \mu \neq \mu'.\end{aligned} \tag{4.45}$$

Game 2. Now we simply delete the lines marked with a "*" in Game 1. Let W_2 be the event that the adversary outputs 1 in this game.

It is clear that this game is equivalent to the split experiment in Attack Game 4.5, and so $|\Pr[W_2] - \Pr[W_1]|$ is equal to the adversary's advantage in Attack Game 4.5. We want to use the Difference Lemma to bound $|\Pr[W_2] - \Pr[W_1]|$. To make this entirely rigorous, one models both games as operating on the same underlying probability space: we define a collection of random variables representing the coins of the adversary, as well as the various random samples from different subsets of $\mathcal{X}$ made by the challenger. These random variables completely describe both Games 1 and 2: the only difference between the two games are the deterministic computation rules that determine the outcomes. Define Z be to be the event that at the end of Game 2, the condition (4.45) does not hold. One can verify that Games 1 and 2 proceed identically unless Z holds, so by the Difference Lemma, we have $|\Pr[W_2] - \Pr[W_1]| \leq \Pr[Z]$. Moreover, it is clear that $\Pr[Z]$ is precisely the failure probability in Attack Game 4.5. □

4.8 A fun application: comparing information without revealing it

In this section we describe an important application for PRFs called **sub-key derivation**. Alice and Bob have a shared key k for a PRF. They wish to generate a sequence of shared keys $k_1, k_2, \ldots$ so that key number i can be computed without having to compute all earlier keys. Naturally, they set $k_i := F(k, i)$ where F is a secure PRF whose input space is $\{1, 2, \ldots, B\}$ for some bound B. The generated sequence of keys is indistinguishable from random keys.

As a fun application of this, consider the following problem: Alice is on vacation at the Squaw valley ski resort and wants to know if her friend Bob is also there. If he is they could ski together. Alice could call Bob and ask him if he is on the slopes, but this would reveal to Bob where she is and Alice would rather not do that. Similarly, Bob values his privacy and does not want to tell Alice where he is, unless Alice happens to be close by.

Abstractly, this problem can be phrased as follows: Alice has a number $a \in \mathbb{Z}_p$ and Bob has a number $b \in \mathbb{Z}_p$ for some prime p. These numbers indicate their approximate positions on earth. Think of dividing the surface of the earth into p squares and the numbers a and b indicate what square Alice and Bob are currently at. If Bob is at the resort then $a = b$, otherwise $a \neq b$.

Alice wants to learn if $a = b$; however, if $a \neq b$ then Alice should learn nothing else about b. Bob should learn nothing at all about a.

In a later chapter we will see how to solve this exact problem. Here, we make the problem easier by allowing Alice and Bob to interact with a server, Sam, that will help Alice learn if $a = b$, but will itself learn nothing at all. The only assumption about Sam is that it does not collude with Alice or Bob, that is, it does not reveal private data that Alice or Bob send to it. Clearly, Alice and Bob could send a and b to Sam and he will tell Alice if $a = b$, but then Sam would learn both a and b. Our goal is that Sam learns nothing, not even if $a = b$.

To describe the basic protocol, suppose Alice and Bob have a shared secret key $(k_0, k_1) \in \mathbb{Z}_p^2$. Moreover, Alice and Bob each have a private channel to Sam. The protocol for comparing a and b

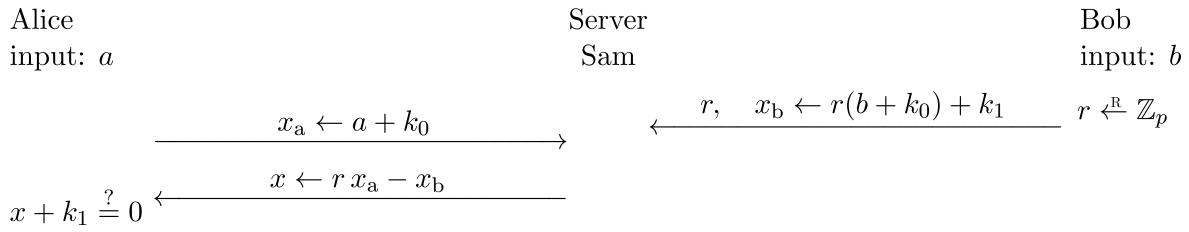

Alice
input: a
Server
Sam
Bob
input: b
$r, \quad x_{\text{b}} \leftarrow r(b + k_0) + k_1$
$r \xleftarrow{\text{R}} \mathbb{Z}_p$
$x_{\text{a}} \leftarrow a + k_0$
$x \leftarrow r\, x_{\text{a}} - x_{\text{b}}$
$x + k_1 \overset{?}{=} 0$

Figure 4.17: Comparing a and b without revealing them

is shown in Fig. 4.17. It begins with Bob choosing a random r in $\mathbb{Z}_p$ and sending (r, x_{b}) to Sam. Bob can do this whenever he wants, even before Alice initiates the protocol. When Alice wants to test equality, she sends x_{a} to Sam. Sam computes $x \leftarrow r\, x_{\text{a}} - x_{\text{b}}$ and sends x back to Alice. Now, observe that

$$x + k_1 = r(a - b)$$

so that $x + k_1 = 0$ when $a = b$ and $x + k_1$ is very likely to be non-zero otherwise (assuming p is sufficiently large so that $r \neq 0$ with high probability). This lets Alice learn if $a = b$.

What is revealed by this protocol? Clearly Bob learns nothing. Alice learns $r(a - b)$, but if $a \neq b$ this quantity is uniformly distributed in $\mathbb{Z}_p$. Therefore, when $a \neq b$ Alice just obtains a uniform element in $\mathbb{Z}_p$ and this reveals nothing beyond the fact that $a \neq b$. Sam sees $r, x_{\text{a}}, x_{\text{b}}$, but all three values are independent of a and b: x_{a} and x_{b} are one-time pad encryptions under keys k_0 and k_1, respectively. Therefore, Sam learns nothing. Notice that the only privacy assumption about Sam is that it does not reveal (r, x_{b}) to Alice or x_{a} to Bob.

The trouble, much like with the one-time pad, is that the shared key (k_0, k_1) can only be used for a *single* equality test, otherwise the protocol becomes insecure. If (k_0, k_1) is used to test if $a = b$ and later the same key (k_0, k_1) is used to test if $a' = b'$ then Alice and Sam learn information they are not supposed to. For example, Sam learns $a - a'$. Moreover, Alice can deduce $(a - b)/(a' - b')$ which reveals information about b and b' (e.g., if $a = a' = 0$ then Alice learns the ratio of b and b').

Sub-key derivation. What if Alice wants to repeatedly test proximity to Bob? The solution is to generate a new independent key (k_0, k_1) for each invocation of the protocol. We do so by deriving instance-specific sub-keys using a secure PRF.

Let F be a secure PRF defined over $(\mathcal{K},\ \{1, \ldots, B\},\ \mathbb{Z}_p^2)$ and suppose that Alice and Bob share a long term key $k \in \mathcal{K}$. Bob maintains a counter cnt_{b} that is initially set to 0. Every time Bob sends his encrypted location (r, x_{b}) to Sam he increments cnt_{b} and derives sub-keys (k_0, k_1) from the long-term key k as:

$$(k_0, k_1) \leftarrow F(k,\ cnt_{\text{b}}). \tag{4.46}$$

He sends $(r, x_{\text{b}}, cnt_{\text{b}})$ to Sam. Bob can do this whenever he wants, say every few minutes, or every time he moves to a new location.

Whenever Alice wants to test proximity to Bob she first asks Sam to send her the value of the counter in the latest message from Bob. She makes sure the counter value is larger than the previous value Sam sent her (to prevent a mischievous Sam or Bob from tricking Alice into re-using an old counter value). Alice then computes (k_0, k_1) herself using (4.46) and carries out the protocol with Sam in Fig. 4.17 using these keys.

Because F is a secure PRF, the sequence of derived sub-keys is indistinguishable from random independently sampled keys. This ensures that the repeated protocol reveals nothing about the tested values beyond equality. By using a PRF, Alice is able to quickly compute (k_0, k_1) for the latest value of cnt_{b}.

4.9 Notes

Citations to the literature to be added.

4.10 Exercises

4.1 (Exercising the definition of a secure PRF). Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{K} = \mathcal{X} = \mathcal{Y} = \{0,1\}^n$.

(a) Show that $F_1(k, x) = F(k, x) \parallel 0$ is not a secure PRF.

(b) Show that $F_2\big(k,\ (x, y)\big) := F(k, x) \oplus F(k, y)$ is insecure.

(c) Prove that $F_3(k, x) := F(k, x) \oplus x$ is a secure PRF.

(d) Prove that $F_4\big((k_1, k_2), x\big) := F(k_1, x) \oplus F(k_2, x)$ is a secure PRF.

(e) Show that $F_5\big((k_1, k_2), (x_1, x_2)\big) := F(k_1, x_1) \oplus F(k_2, x_2)$ is insecure.

(f) Show that $F_6(k, x) := F(k, x) \parallel F(k, x \oplus 1^n)$ is insecure.

(g) Prove that $F_7(k, x) := F(F(k, 0^n), x)$ is a secure PRF.

(h) Show that $F_8(k, x) := F(F(k, 0^n), x) \parallel F(k, x)$ is insecure.

(i) Show that $F_9(k, x) := F(k, x) \parallel F\big(k, F(k, x)\big)$ is insecure.

4.2 (Weak PRFs). Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ where $\mathcal{Y} := \{0,1\}^n$ and $|\mathcal{X}|$ is super-poly. Define

$$F_5\big((k_1, k_2), (x_1, x_2)\big) := F(k_1, x_1) \oplus F(k_2, x_2).$$

We showed in Exercise 4.1 part (e) that F_5 is not a secure PRF.

(a) Show that F_5 is a weakly secure PRF (as in Definition 4.3), assuming F is weakly secure. In particular, for any Q-query weak PRF adversary $\mathcal{A}$ attacking F_5 (i.e., an adversary that only queries the function at random points in $\mathcal{X}$) there is a weak PRF adversary $\mathcal{B}$ attacking F, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{wPRF}\mathsf{adv}[\mathcal{A}, F_5] \leq \text{wPRF}\mathsf{adv}[\mathcal{B}, F] + (Q/|\mathcal{X}|)^4.$$

(b) Suppose F is a secure PRF. Show that F_5 is weakly secure even if we modify the weak PRF attack game and allow the adversary $\mathcal{A}$ to query F_5 at one chosen point in addition to the Q random points. A PRF that is secure in this sense is sufficient for a popular data integrity mechanism discussed in Section 7.4.

(c) Show that F_5 is no longer secure if we modify the weak PRF attack game and allow the adversary $\mathcal{A}$ to query F_5 at two chosen points in addition to the Q random points.

4.3 (Format preserving encryption). Suppose we are given a block cipher (E, D) operating on domain $\mathcal{X}$. We want a block cipher (E', D') that operates on a smaller domain $\mathcal{X}' \subseteq \mathcal{X}$. Define (E', D') as follows:

```
$E'(k, x) :=$   $y \leftarrow E(k, x)$
                while $y \notin \mathcal{X}'$ do: $y \leftarrow E(k, y)$
                output $y$
```

$D'(k, y)$ is defined analogously, applying $D(k, \cdot)$ until the result falls in $\mathcal{X}'$. Clearly (E', D') are defined on domain $\mathcal{X}'$.

(a) With $t := |\mathcal{X}|/|\mathcal{X}'|$, how many evaluations of E are needed in expectation to evaluate $E'(k, x)$ as a function of t? You answer shows that when t is small (e.g., $t \leq 2$) evaluating $E'(k, x)$ can be done efficiently.

(b) Show that if (E, D) is a secure block cipher with domain $\mathcal{X}$ then (E', D') is a secure block cipher with domain $\mathcal{X}'$. Try proving security by induction on $|\mathcal{X}| - |\mathcal{X}'|$.

Discussion: This exercise is used in the context of encrypted 16-digit credit card numbers where the ciphertext also must be a 16-digit number. This type of encryption, called **format preserving encryption**, amounts to constructing a block cipher whose domain size is exactly 10^{16}. This exercise shows that it suffices to construct a block cipher (E, D) with domain size 2^{54} which is the smallest power of 2 larger than 10^{16}. The procedure in the exercise can then be used to shrink the domain to size 10^{16}.

4.4 (Truncating PRFs). Let F be a PRF whose range is $\mathcal{Y} = \{0,1\}^n$. For some $\ell < n$ consider the PRF F' with a range $\mathcal{Y}' = \{0,1\}^\ell$ defined as: $F'(k, x) := F(k, x)[0 \ldots \ell - 1]$. That is, we truncate the output of $F(k, x)$ to the first ℓ bits. Show that if F is a secure PRF then so is F'.

4.5 (Two-key Triple-DES). Consider the following variant of the $3\mathcal{E}$ construction that uses only two keys: for a block cipher (E, D) with key space $\mathcal{K}$ define $3\mathcal{E}'$ as $E((k_1, k_2), m) := E(k_1, E(k_2, E(k_1, m)))$. Show that this block cipher can be defeated by a meet in the middle attack using $O(|\mathcal{K}|)$ evaluation of E and D and using $O(|\mathcal{K}|)$ encryption queries to the block cipher challenger. Further attacks on this method are discussed in [112, 105].

4.6 (adaptive vs non-adaptive security). This exercise develops an argument that shows that a PRF may be secure against every adversary that makes its queries non-adaptively, (i.e., all at once) but is insecure against adaptive adversaries (i.e., the kind allowed in Attack Game 4.2).

To be a bit more precise, we define the non-adaptive version of Attack Game 4.2 as follows. The adversary submits all at once the query $(x_1, \ldots, x_Q)$ to the challenger, who responds with $(y_1, \ldots, y_Q)$, where $y := f(x_i)$. The rest of the attack game is the same: in Experiment 0, $k \xleftarrow{\text{R}} \mathcal{K}$ and $f \xleftarrow{\text{R}} F(k, \cdot)$, while in Experiment 1, $f \xleftarrow{\text{R}} \text{Funs}[\mathcal{X}, \mathcal{Y}]$. Security against non-adaptive adversaries means that all efficient adversaries have only negligible advantage; advantage is defined as usual: $|\Pr[W_0] - \Pr[W_1]|$, where W_b is the event that the adversary outputs 1 in Experiment b.

Suppose F is a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, where $N := |\mathcal{X}|$ is super-poly. We proceed to "sabotage" F, constructing a new PRF $\tilde{F}$ as follows. Let x' be some fixed element of $\mathcal{X}$. For $x = F(k, x')$ define $\tilde{F}(k, x) := x'$, and for all other x define $\tilde{F}(k, x) := F(k, x)$.

(a) Show that $\tilde{F}$ is not a secure PRF against adaptive adversaries.

(b) Show that $\tilde{F}$ is a secure PRF against non-adaptive adversaries.

(c) Show that a similar construction is possible for block ciphers: given a secure block cipher (E, D) defined over $(\mathcal{K}, \mathcal{X})$ where $|\mathcal{X}|$ is super-poly, construct a new, "sabotaged" block cipher $(\tilde{E}, \tilde{D})$ that is secure against non-adaptive adversaries, but insecure against adaptive adversaries.

4.7 (PRF security definition). This exercise develops an alternative characterization of PRF security for a PRF F defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$. As usual, we need to define an attack game between an adversary $\mathcal{A}$ and a challenger. Initially, the challenger generates

$$b \xleftarrow{\text{R}} \{0,1\},\ k \xleftarrow{\text{R}} \mathcal{K},\ \tilde{y}_1 \xleftarrow{\text{R}} \mathcal{Y}$$

Then $\mathcal{A}$ makes a series of queries to the challenger. There are two types of queries:

Function: In an *function query*, $\mathcal{A}$ submits an $x \in \mathcal{X}$ to the challenger, who responds with $y \leftarrow F(k, x)$. The adversary may make any (poly-bounded) number of function queries.

Test: In a *test query*, $\mathcal{A}$ submits an $\tilde{x} \in \mathcal{X}$ to the challenger, who computes $\tilde{y}_0 \leftarrow F(k, \tilde{x})$ and responds with $\tilde{y}_b$. The adversary is allowed to make only a *single* test query (with any number of function queries before and after the test query). The test point $\tilde{x}$ is not allowed to among the function the queries x.

At the end of the game, $\mathcal{A}$ outputs a bit $\hat{b} \in \{0,1\}$. As usual, we define $\mathcal{A}$'s advantage in the above attack game to be $|\Pr[\hat{b} = b] - 1/2|$. We say that F is Alt-PRF secure if this advantage is negligible for all efficient adversaries. Show that F is a secure PRF if and only if F is Alt-PRF secure.

Discussion: This characterization shows that the value of a secure PRF at a point $\tilde{x}$ in $\mathcal{X}$ looks like a random element of $\mathcal{Y}$, even after seeing the value of the PRF at many other points of $\mathcal{X}$.

4.8 (Key malleable PRFs). Let F be a PRF defined over $(\{0,1\}^n, \{0,1\}^n, \mathcal{Y})$.

(a) We say that F is **XOR-malleable** if $F(k,\ x \oplus c) = F(k, x) \oplus c$ for all k, x, c in $\{0,1\}^n$.

(b) We say that F is **key XOR-malleable** if $F(k \oplus c,\ x) = F(k, x) \oplus c$ for all k, x, c in $\{0,1\}^n$.

Clearly an XOR-malleable PRF cannot be secure: malleability lets an attacker distinguish the PRF from a random function. Show that the same holds for a key XOR-malleable PRF.

Remark: In contrast, we note that there are secure PRFs where $F(k_1 \oplus k_2,\ x) = F(k_1, x) \oplus F(k_2, x)$. See Exercise 11.2 for an example, where the xor on the left is replaced by addition, and the xor on the right is replaced by multiplication.

4.9 (Strongly secure block ciphers). In Section 4.1.3 we sketched out the notion of a strongly secure block cipher.

(a) Write out the complete definition of a strongly secure block cipher as a game between a challenger and an adversary.

(b) Consider the following cipher $\mathcal{E}' = (E', D')$ built from a block cipher (E, D) defined over $(\mathcal{K}, \{0,1\}^n)$:

$$E'(k, m) := D(k,\ t \oplus E(k, m)\ ) \quad \text{and} \quad D'(k, c) := D(k,\ t \oplus E(k, c)\ )$$

where $t \in \{0,1\}^n$ is a fixed constant. For what values of t is this cipher $\mathcal{E}'$ semantically secure? Prove semantic security assuming the underlying block cipher is strongly secure.

4.10 (Meet-in-the-middle attacks). Let us study the security of the $4\mathcal{E}$ construction where a block cipher (E, D) is iterated four times using four different keys: $E_4(\ (k_1, k_2, k_3, k_4),\ m) := E\big(k_4,\ E(k_3,\ E(k_2,\ E(k_1, m)))\big)$ where (E, D) is a block cipher defined over $(\mathcal{K}, \mathcal{X})$.

(a) Show that there is a meet in the middle attack on $4\mathcal{E}$ that recovers the secret key in time $|\mathcal{K}|^2$ and memory space $|\mathcal{K}|^2$.

(b) Suppose $|\mathcal{K}| = |\mathcal{X}|$. Show that there is a meet in the middle attack on $4\mathcal{E}$ that recovers the secret key in time $|\mathcal{K}|^2$, but only uses memory space $|\mathcal{K}|$. If you get stuck see [53].

4.11 (Tweakable block ciphers). A tweakable block cipher is a block cipher whose encryption and decryption algorithm take an additional input t, called a "tweak", which is drawn from a "tweak space" $\mathcal{T}$. As usual, keys come from a key space $\mathcal{K}$, and data blocks from a data block space $\mathcal{X}$. The encryption and decryption functions operate as follows: for $k \in \mathcal{K}, x \in \mathcal{X}, t \in \mathcal{T}$, we have $y = E(k, x, t) \in \mathcal{X}$ and $x = D(k, y, t)$. So for each $k \in \mathcal{K}$ and $t \in \mathcal{T}$, $E(k, \cdot, t)$ defines a permutation on $\mathcal{X}$ and $D(k, \cdot, t)$ defines the inverse permutation. Unlike keys, tweaks are typically publicly known, and may even be adversarially chosen.

Security is defined by a game with two experiments. In both experiments, the challenger defines a family of permutations $\{\Pi_t\}_{t \in \mathcal{T}}$, where each Π_t is a permutation on $\mathcal{X}$. In Experiment 0, the challenger sets $k \xleftarrow{R} \mathcal{K}$, and

$$\Pi_t := E(k, \cdot, t) \text{ for all } t \in \mathcal{T}.$$

In Experiment 1, the challenger sets

$$\Pi_t \xleftarrow{R} \text{Perms}[\mathcal{X}] \text{ for all } t \in \mathcal{T}.$$

Both experiments then proceed identically. The adversary issues a series of queries. Each query is one of two types:

forward query: the adversary sends $(x, t) \in \mathcal{X} \times \mathcal{T}$, and the challenger responds with $y := \Pi_t(x)$;

inverse queries: the adversary sends $(y, t) \in \mathcal{X} \times \mathcal{T}$, and the challenger responds with $x := \Pi_t^{-1}(y)$.

At the end of the game, the adversary outputs a bit. If p_b is the probability that the adversary outputs 1 in Experiment b, the adversary's advantage is defined to be $|p_0 - p_1|$. We say that (E, D) is a *secure* tweakable block cipher if every efficient adversary has negligible advantage.

This definition of security generalizes the notion of a *strongly* secure block cipher (see Section 4.1.3 and Exercise 4.9). In applications of tweakable block ciphers, this strong security notion is more appropriate (e.g., see Exercise 9.17).

(a) Prove security of the construction $\tilde{E}(k, m, t) := E(E(k, t), m)$ where (E, D) is a strongly secure block cipher defined over $(\mathcal{K}, \mathcal{K})$.

(b) Show that there is an attack on the construction from part (a) that achieves advantage $\geq 1/2$ and which makes $Q \approx \sqrt{|\mathcal{K}|}$ queries.

Hint: In addition to the $\approx \sqrt{|\mathcal{K}|}$ queries, your adversary should make an additional $\approx \sqrt{|\mathcal{K}|}$ "offline" evaluations of the cipher (E, D).

(c) Prove security of the construction

$$E'\big((k_0, k_1), m, t\big) := \big\{p \leftarrow F(k_0, t); \quad \text{output } p \oplus E(k_1,\ m \oplus p)\big\},$$

where (E, D) is a *strongly* secure block cipher and F is a secure PRF. In Exercise 7.10 we will see a more efficient variant of this construction.

Hint: Use the assumption that (E, D) is a strongly secure block cipher to replace $E(k_1, \cdot)$ in the challenger by a truly random permutation $\widetilde{\Pi}$; then, use the Domain Separation Lemma (see Theorem 4.15) to replace $\widetilde{\Pi}$ by a family of independent permutations $\{\widetilde{\Pi}_t\}_{t \in \mathcal{T}}$, and analyze the corresponding domain separation failure probability.

Discussion: Tweakable block ciphers are used in disk sector encryption where encryption must not expand the data: the ciphertext size is required to have the same size as the input. The sector number is used as the tweak to ensure that even if two sectors contain the same data, the resulting encrypted sectors are different. The construction in part (c) is usually more efficient than that in part (a), as the latter uses a different block cipher key with every evaluation, which can incur extra costs. See further discussion in Exercise 7.10.

4.12 (PRF combiners). We want to build a PRF F using two PRFs F_1 and F_2, so that if at some future time one of F_1 or F_2 is broken (but not both) then F is still secure. Put another way, we want to construct F from F_1 and F_2 such that F is secure if either F_1 or F_2 is secure.

Suppose F_1 and F_2 both have output spaces $\{0, 1\}^n$, and both have a common input space. Define

$$F(\ (k_1, k_2),\ x) := F_1(k_1, x) \oplus F_2(k_2, x).$$

Show that F is secure if either F_1 or F_2 is secure.

4.13 (Block cipher combiners). Continuing with Exercise 4.12, we want to build a block cipher $\mathcal{E} = (E, D)$ from two block ciphers $\mathcal{E}_1 = (E_1, D_1)$ and $\mathcal{E}_2 = (E_2, D_2)$ so that if at some future time one of $\mathcal{E}_1$ or $\mathcal{E}_2$ is broken (but not both) then $\mathcal{E}$ is still secure. Suppose both $\mathcal{E}_1$ and $\mathcal{E}_2$ are defined over $(\mathcal{K}, \mathcal{X})$. Define $\mathcal{E}$ as:

$$E(\ (k_1, k_2),\ x) := E_1\big(k_1,\ E_2(k_2, x)\big) \qquad \text{and} \qquad D(\ (k_1, k_2),\ y) := D_2\big(k_2,\ D_1(k_1, y)\big).$$

(a) Show that $\mathcal{E}$ is secure if either $\mathcal{E}_1$ or $\mathcal{E}_2$ is secure.

(b) Show that this is not a secure combiner for PRFs. That is, $F(\ (k_1, k_2),\ x) := F_1\big(k_1,\ F_2(k_2, x)\big)$ need not be a secure PRF even if one of F_1 or F_2 is.

4.14 (Key leakage). Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{K} = \mathcal{X} = \mathcal{Y} = \{0, 1\}^n$.

(a) Let $\mathcal{K}_1 = \{0, 1\}^{n+1}$. Construct a new PRF F_1, defined over $(\mathcal{K}_1, \mathcal{X}, \mathcal{Y})$, with the following property: the PRF F_1 is secure; however, if the adversary learns the last bit of the key then the PRF is no longer secure. This shows that leaking even a *single* bit of the secret key can completely destroy the PRF security property.

Hint: Let $k_1 = k \parallel b$ where $k \in \{0, 1\}^n$ and $b \in \{0, 1\}$. Set $F_1(k_1, x)$ to be the same as $F(k, x)$ for all $x \neq 0^n$. Define $F_1(k_1, 0^n)$ so that F_1 is a secure PRF, but becomes easily distinguishable from a random function if the last bit of the secret key k_1 is known to the adversary.

(b) Construct a new PRF F_2, defined over $(\mathcal{K} \times \mathcal{K}, \mathcal{X}, \mathcal{Y})$, that remains secure if the attacker learns any *single* bit of the key. Your function F_2 may only call F once.

4.15 (Variants of Luby-Rackoff). Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$.

(a) Show that two-round Luby-Rackoff is not a secure block cipher.

(b) Show that three-round Luby-Rackoff is not a strongly secure block cipher.

4.16 (Insecure tree construction). In the tree construction for building a PRF from a PRG (Section 4.6), the secret key is used at the root of the tree and the input is used to trace a path through the tree. Show that a construction that does the opposite is not a secure PRF. That is, using the input as the root and using the key to trace through the tree is not a secure PRF.

4.17 (Truncated tree construction). Suppose we cut off the tree construction from Section 4.6 after only three levels of the tree, so that there are only eight leaves, as in Fig. 4.15. Give a direct proof, using a sequence of seven hybrids, that outputting the values at all eight leaves gives a secure PRG defined over $(\mathcal{S}, \mathcal{S}^8)$, assuming the underlying PRG is secure.

4.18 (Augmented tree construction). Suppose we are given a PRG G defined over $(\mathcal{K} \times \mathcal{S}, \mathcal{S}^2)$. Write $G(k,s) = (G_0(k,s), G_1(k,s))$. Let us define the PRF G^* with key space $\mathcal{K}^n \times \mathcal{S}$ and input space $\{0,1\}^n$ as follows:

```
$G^*\big((k_0, \ldots, k_{n-1}, s),\ x \in \{0,1\}^n\big) :=$
        $t \leftarrow s$
        for $i \leftarrow 0$ to $n-1$ do
                $b \leftarrow x[i]$
                $t \leftarrow G_b(k_i, t)$
        output $t$.
```

(a) Given an example secure PRG G for which G^* is insecure as a PRF.

(b) Show that G^* is a secure PRF if for every poly-bounded Q the following PRG is secure:

$$G'(k, s_0, \ldots, s_{Q-1}) := (G(k, s_0), \ldots, G(k, s_{Q-1}))\ .$$

4.19 (Synthesizers and parallel PRFs). For a secure PRG G defined over $(\mathcal{S}, \mathcal{R})$ we showed that $G^n(s_1, \ldots, s_n) := \big(G(s_1), \ldots, G(s_n)\big)$ is a secure PRG over $(\mathcal{S}^n, \mathcal{R}^n)$. The proof requires that the components $s_1, \ldots, s_n$ of the seed be chosen uniformly and independently over $\mathcal{S}^n$. A secure synthesizer is a PRG for which this holds even if $s_1, \ldots, s_n$ are not independent of one another. Specifically, a **synthesizer** is an efficient function $S : \mathcal{X}^2 \to \mathcal{X}$. The synthesizer is said to be *n-way secure* if

$$S^n(x_1, y_1, \ldots, x_n, y_n) := \big(\ S(x_i, y_j)\ \big)_{i,j=1,\ldots,n}\ \in \mathcal{X}^{(n^2)}$$

is a secure PRG defined over $(\mathcal{X}^{2n}, \mathcal{X}^{(n^2)})$. Here S is being evaluated at n^2 inputs that are not independent of one another and yet S^n is a secure PRG.

(a) Not every secure PRG is a secure synthesizer. Let G be a secure PRG over $(\mathcal{S}, \mathcal{R})$. Show that $S(x,y) := \big(G(x), y\big)$ is a secure PRG defined over $(\mathcal{S}^2,\ \mathcal{R} \times \mathcal{S})$, but is an insecure 2-way synthesizer.

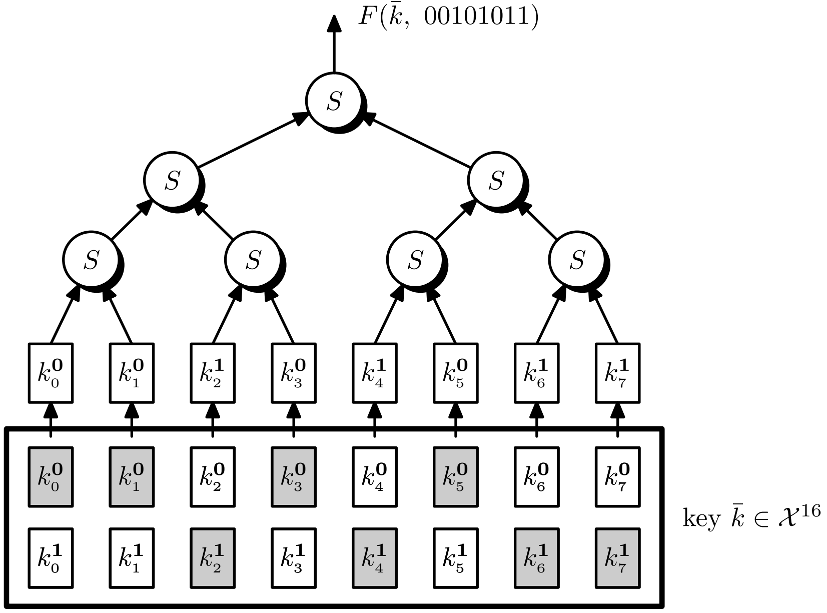


Figure 4.18: A PRF built from a synthesizer S. The PRF input in $\{0,1\}^n$ is used to select n components from the key $\bar{k} \in \mathcal{X}^{2n}$. The selected components, shown as shaded squares, are used as shown in the figure.

(b) A secure synthesizer lets us build a large domain PRF that can be evaluated quickly on a parallel computer. Show that if $S : \mathcal{X}^2 \to \mathcal{X}$ is a Q-way secure synthesizer, for poly-bounded Q, then the PRF in Fig. 4.18 is a secure PRF defined over $(\mathcal{X}^{2n},\ \{0,1\}^n,\ \mathcal{X})$. For simplicity, assume that n is a power of 2. Observe that the PRF can be evaluated in only $\log_2 n$ steps on a parallel computer.

4.20 (Insecure variants of Even-Mansour). In Section 4.7.3 we discussed the Even-Mansour block cipher (E, D) built from a permutation $\pi : \mathcal{X} \to \mathcal{X}$ where $\mathcal{X} = \{0,1\}^n$. Recall that $E\big((P_0, P_1),\ m\big) := \pi(m \oplus P_0) \oplus P_1$.

(a) Show that $E_1(P_0, m) := \pi(m \oplus P_0)$ is not a secure block cipher.

(b) Show that $E_2(P_1, m) := \pi(m) \oplus P_1$ is not a secure block cipher.

4.21 (Birthday attack on Even-Mansour). Let's show that the bounds in the Even-Mansour security theorem (Theorem 4.14) are tight. For $\mathcal{X} := \{0,1\}^n$, recall that the Even-Mansour block cipher (E, D), built from a permutation $\pi : \mathcal{X} \to \mathcal{X}$, is defined as: $E\big((k_0, k_1),\ m\big) := \pi(m \oplus k_0) \oplus k_1$. We show how to break this block cipher in time approximately $2^{n/2}$.

(a) Show that for all $a, m, \Delta \in \mathcal{X}$ and $\bar{k} := (k_0, k_1) \in \mathcal{X}^2$, whenever $a = m \oplus k_0$, we have

$$E\big(\bar{k},\ m\big) \oplus E\big(\bar{k},\ m \oplus \Delta\big) = \pi(a) \oplus \pi(a \oplus \Delta)$$

(b) Use part (a) to construct an adversary $\mathcal{A}$ that wins the block cipher security game against (E, D) with advantage close to 1, in the ideal cipher model. With $q := 2^{n/2}$ and some non-zero

$\Delta \in \mathcal{X}$, the adversary $\mathcal{A}$ queries the cipher at $2q$ random points $m_i,\ m_i \oplus \Delta \in \mathcal{X}$ and queries the permutation π at $2q$ random points $a_i,\ a_i \oplus \Delta \in \mathcal{X}$, for $i = 1, \ldots, q$.

4.22 (A variant of the Even-Mansour cipher). Let $\mathcal{M} := \{0,1\}^m$, $\mathcal{K} := \{0,1\}^n$, and $\mathcal{X} := \{0,1\}^{n+m}$. Consider the following cipher (E, D) defined over $(\mathcal{K}, \mathcal{M}, \mathcal{X})$ built from a permutation $\pi : \mathcal{X} \to \mathcal{X}$:

$$E(k, x) := (k \parallel 0^m) \oplus \pi(k \parallel x) \tag{4.47}$$

$D(k, c)$ is defined analogously. Show that if we model π as an ideal permutation Π, then for every block cipher adversary $\mathcal{A}$ attacking (E, D) we have

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, E] \le \frac{2Q_{\mathrm{ic}}}{|\mathcal{K}|} \ . \tag{4.48}$$

Here Q_{ic} is the number of queries $\mathcal{A}$ makes to Π- and Π^{-1}-oracles.

4.23 (Analysis of Salsa and ChaCha). In this exercise we analyze the Salsa and ChaCha stream ciphers from Section 3.6 in the ideal permutation model. Let $\pi : \mathcal{X} \to \mathcal{X}$ be a permutation, where $\mathcal{X} = \{0,1\}^{n+m}$. Let $\mathcal{K} := \{0,1\}^n$ and define the PRF F, which is defined over $(\mathcal{K}, \{0,1\}^m, \mathcal{X})$, as

$$F(k, x) := (k \parallel x) \oplus \pi(k \parallel x) \ . \tag{4.49}$$

This PRF is an abstraction of the PRF underlying the Salsa and ChaCha stream ciphers. Use Exercise 4.22 to show that if we model π as an ideal permutation Π, then for every PRF adversary $\mathcal{A}$ attacking F we have

$$\mathrm{PRF}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, F] \le \frac{2Q_{\mathrm{ic}}}{|\mathcal{K}|} + \frac{Q_{\mathrm{F}}^2}{2|\mathcal{X}|} \tag{4.50}$$

where Q_{F} is the number of queries that $\mathcal{A}$ makes to an $F(k, \cdot)$ oracle and Q_{ic} is the number of queries $\mathcal{A}$ makes to Π- and Π^{-1}-oracles. In Salsa and ChaCha, Q_{F} is at most $|\mathcal{X}|^{1/4}$ so that $\frac{Q_{\mathrm{F}}^2}{2|\mathcal{X}|}$ is "negligible."

Discussion: The specific permutation π used in the Salsa and ChaCha stream ciphers is not quite an ideal permutation. For example, $\pi(0^{n+m}) = 0^{n+m}$. Hence, your analysis applies to the general framework, but not specifically to Salsa and ChaCha.

4.24 (Alternative proof of Theorem 4.6). Let X and Y be random variables as defined in Exercise 3.14. Consider an adversary $\mathcal{A}$ in Attack Game 4.3 that makes at most Q queries to its challenger. Show that $\mathrm{PF}\mathsf{adv}[\mathcal{A}, \mathcal{X}] \le \Delta[\mathsf{X}, \mathsf{Y}] \le Q^2/2N$.

4.25 (A one-sided switching lemma). Following up on the previous exercise, one can use part (b) of Exercise 3.14 to get a "one sided" version of Theorem 4.6, which can be useful in some settings. Consider an adversary $\mathcal{A}$ in Attack Game 4.3 that makes at most Q queries to its challenger. Let W_0 and W_1 be as defined in that game: W_0 is the event that $\mathcal{A}$ outputs 1 when probing a random permutation, and W_1 is the event that $\mathcal{A}$ outputs 1 when probing a random function. Assume $Q^2 < N$. Show that $\Pr[W_0] \le \rho[\mathsf{X}, \mathsf{Y}] \cdot \Pr[W_1] \le 2 \Pr[W_1]$.

4.26 (Parallel composition of PRFs). Just as we can compose PRGs in parallel, while maintaining security (see Section 3.4.1), we can also compose PRFs in parallel, while maintaining security.

Suppose we have a PRF F, defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$. We want to model the situation where an adversary is given n black boxes (where $n \geq 1$ is poly-bounded): the boxes either contain $F(k_1, \cdot), \ldots, F(k_n, \cdot)$, where the k_i are random (and independent) keys, or they contain $f_1, \ldots, f_n$, where the f_i are random elements of $\text{Funs}[\mathcal{X}, \mathcal{Y}]$, and the adversary should not be able to tell the difference.

A convenient way to model this situation is to consider the **n-wise parallel composition of** F, which is a PRF F' whose key space is $\mathcal{K}^n$, whose input space is $\{1, \ldots, n\} \times \mathcal{X}$, and whose output space is $\mathcal{Y}$. Given a key $k' = (k_1, \ldots, k_n)$, and an input $x' = (s, x)$, with $s \in \{1, \ldots, n\}$ and $x \in \mathcal{X}$, we define $F'(k', x') := F(k_s, x)$.

Show that if F is a secure PRF, then so is F'. In particular, show that for every PRF adversary $\mathcal{A}$, then exist a PRF adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that $\text{PRF}\mathsf{adv}[\mathcal{A}, F'] = n \cdot \text{PRF}\mathsf{adv}[\mathcal{B}, F]$.

4.27 (A universal attacker on PRFs). Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, and let Q be some integer where $0 < Q \leq |\mathcal{K}|$.

(a) Suppose that $|\mathcal{Y}| \geq |\mathcal{K}|$. Show that there is a PRF adversary $\mathcal{A}$ that runs in time linear in Q, makes one query to the PRF challenger, and has advantage

$$\text{PRF}\mathsf{adv}[\mathcal{A}, F] \geq \frac{Q}{|\mathcal{K}|} - \frac{Q}{|\mathcal{Y}|}. \tag{4.51}$$

Hint: Try to first devise your PRF adversary $\mathcal{A}$ when $Q = |\mathcal{K}|$.

(b) When $|\mathcal{K}| = |\mathcal{Y}|$ the bound in (4.51) is meaningless. To obtain a meaningful bound, use part (a) to show that when $|\mathcal{Y}| \geq |\mathcal{K}|^{1/2}$ there is a PRF adversary $\mathcal{A}$ that makes *two* queries to the PRF challenger, runs in time linear in Q, and has advantage

$$\text{PRF}\mathsf{adv}[\mathcal{A}, F] \geq \frac{Q}{|\mathcal{K}|} - \frac{Q}{|\mathcal{Y}|^2}. \tag{4.52}$$

4.28 (Distributed PRFs). Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ where $\mathcal{Y} := \{0,1\}^n$. In Exercise 4.1 part (d) we showed that if F is secure then so is

$$F'\big((k_1, k_2), x) := F(k_1, x) \oplus F(k_2, x).$$

This F' has a useful property: the PRF key (k_1, k_2) can be split into two shares, k_1 and k_2. If Alice is given one share and Bob the other share, then both Alice and Bob are needed to evaluate the PRF, and neither can evaluate the PRF on its own. Moreover, the PRF can be evaluated distributively, that is, without re-constituting the key (k_1, k_2): to evaluate the PRF at a point x_0, Alice simply sends $F(k_1, x_0)$ to Bob.

(a) To show that Alice cannot evaluate F' by herself, show that F' is a secure PRF even if the adversary is given k_1. Argue that the same holds for k_2.

(b) Construct a PRF where the key can be split into three shares s_1, s_2, s_3 so that any *two* shares can be used evaluate the PRF distributively, but no single share is sufficient to evaluate the PRF on its own.

Hint: Consider the PRF $F''\big((k_1, k_2, k_3), x) := F(k_1, x) \oplus F(k_2, x) \oplus F(k_3, x)$ and show how to construct the shares s_1, s_2, s_3 from the keys k_1, k_2, k_3. Make sure to prove that the F'' is a secure PRF when the adversary is given a single share, namely s_i for some $i \in \{1, 2, 3\}$.

(c) Generalize the construction from part (b) to construct a PRF F''' supporting three-out-of-five sharing of the key: any three shares can be used to evaluate the PRF distributively, but no two shares can.

Hint: The key space for F''' is $\mathcal{K}^{10}$.

Chapter 5

Chosen Plaintext Attack

This chapter focuses on the problem of securely encrypting several messages in the presence of an adversary who eavesdrops, and who may even influence the choice of some messages in order to glean information about other messages. This leads us to the notion of semantic security against a *chosen plaintext attack*.

5.1 Introduction

In Chapter 2, we focused on the problem of encrypting a single message. Now we consider the problem of encrypting several messages. To make things more concrete, suppose Alice wants to use a cipher to encrypt her files on some file server, while keeping her secret keys for the cipher stored securely on her USB memory stick.

One possible approach is for Alice to encrypt each individual file using a different key. This entails that for each file, she stores an encryption of that file on the file server, as well as a corresponding secret key on her memory stick. As we will explore in detail in Section 5.2, this approach will provide Alice with reasonable security, provided she uses a semantically secure cipher. Now, although a file may be several megabytes long, a key for any practical cipher is just a few bytes long. However, if Alice has many thousands of files to encrypt, she must store many thousands of keys on her memory stick, which may not have sufficient storage for all these keys.

As we see, the above approach, while secure, is not very space efficient, as it requires one key per file. Faced with this problem, Alice may simply decide to encrypt all her files with the same key. While more efficient, this approach may be insecure. Indeed, if Alice uses a cipher that provides only semantic security (as in Definition 2.2), this may not provide Alice with any meaningful security guarantee, and may very well expose her to a realistic attack.

For example, suppose Alice uses the stream cipher $\mathcal{E}$ discussed in Section 3.2. Here, Alice's key is a seed s for a PRG G, and viewing a file m as a bit string, Alice encrypts m by computing the ciphertext $c := m \oplus \Delta$, where Δ consists of the first $|m|$ bits of the "key stream" $G(s)$. But if Alice uses this same seed s to encrypt many files, an adversary can easily mount an attack. For example, if an adversary knows some of the bits of one file, he can directly compute the corresponding bits of the key stream, and hence obtain the corresponding bits of *any* file. How might an adversary know some bits of a given file? Well, certain files, like email messages, contain standard header information (see Example 2.6), and so if the adversary knows that a given ciphertext is an encryption of an email, he can get the bits of the key stream that correspond to the location of the bits in this standard header. To mount an even more devastating attack, the adversary may try something even more devious: he could simply send Alice a large email, say one megabyte in length; assuming that Alice's software automatically stores an encryption of this email on her server, when the adversary snoops her file server, he can recover a corresponding one megabyte chunk of the key stream, and now he can decrypt any one megabyte file stored on Alice's server! This email may even be caught in Alice's spam filter, and never actually be seen by Alice, although her encryption software may very well diligently encrypt this email along with everything else. This type of an attack is called a *chosen plaintext attack*, because the adversary forces Alice to give him the encryption of one or more plaintexts of his choice during his attack on the system.

Clearly, the stream cipher above is inadequate for the job. In fact, the stream cipher, as well as *any other deterministic cipher*, should not be used to encrypt multiple files with the same key. Why? Any deterministic cipher that is used to encrypt several files with the same key will suffer from an inherent weakness: an adversary will always be able to tell when two files are identical or not. Indeed, with a deterministic cipher, if the same key is used to encrypt the same message, the resulting ciphertext will always be the same (and conversely, for *any* cipher, if the same key is used to encrypt two different messages, the resulting ciphertexts must be different). While this type of attack is certainly not as dramatic as those discussed above, in which the adversary can read Alice's files almost at will, it is still a serious vulnerability. For example, while the discussion in Section 4.1.4 about ECB mode was technically about encrypting a single message consisting of many data blocks, it applies equally well to the problem of encrypting many single-block messages under the same key.

In fact, it *is* possible for Alice to use a cipher to securely encrypt all of her files under a single, short key, but she will need to use a cipher that is better suited to this task. In particular, because of the above inherent weakness of any deterministic cipher, she will have to use a *probabilistic* cipher, that is, a cipher that uses a probabilistic encryption algorithm, so that different encryptions of the same plaintext under the same key will (generally) produce different encryptions. For her task, she will want a cipher that achieves a level of security stronger than semantic security. The appropriate notion of security is called *semantic security against chosen plaintext attack.* In Section 5.3 and the sections following, we formally define this concept, look at some constructions based on semantically secure ciphers, PRFs, and block ciphers, and look at a few case studies of "real world" systems.

While the above discussion motivated the topics in this chapter using the example of the "file encryption" problem, one can also motivate these topics by considering the "secure network communication" problem. In this setting, one considers the situation where Alice and Bob share a secret key (or keys), and Alice wants to secretly transmit several messages to Bob over an insecure network. Now, if Alice can conveniently concatenate all of her messages into one long message, then she can just use a stream cipher to encrypt the whole lot, and be done with it. However, for a variety of technical reasons, this may not be feasible: if she wants to be able to transmit the messages in an arbitrary order and at arbitrary times, then she is faced with a problem very similar to that of the "file encryption" problem. Again, if Alice and Bob want to use a single, short key, the right tool for the job is a cipher semantically secure against chosen plaintext attack.

We stress again that just like in Chapter 2, the techniques covered in this chapter *do not* provide any *data integrity*, nor do they address the problem of how two parties come to share a secret key to begin with. These issues are dealt with in coming chapters.

5.2 Security against multi-key attacks

Consider again the "file encryption" problem discussed in the introduction to this chapter. Suppose Alice chooses to encrypt each of her files under different, independently generated keys using a semantically secure cipher. Does semantic security imply a corresponding security property in this "multi-key" setting?

The answer to this question is "yes." We begin by stating the natural security property corresponding to semantic security in the multi-key setting.

Attack Game 5.1 (multi-key semantic security). For a given cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define

Experiment b:

- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots$, the ith query is a pair of messages, $m_{i0}, m_{i1} \in \mathcal{M}$, of the same length.

  The challenger computes $k_i \overset{\text{R}}{\leftarrow} \mathcal{K}$, $c_i \overset{\text{R}}{\leftarrow} E(k_i, m_{ib})$, and sends c_i to the adversary.

- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{MSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - \Pr[W_1]|. \quad \Box$$

We stress that in the above attack game, the adversary's queries are *adaptively chosen*, in the sense that for each $i = 1, 2, \ldots$, the message pair (m_{i0}, m_{i1}) may be computed by the adversary in some way that depends somehow on the previous encryptions $c_1, \ldots, c_{i-1}$ output by the challenger.

Definition 5.1 (Multi-key semantic security). *A cipher $\mathcal{E}$ is called **multi-key semantically secure** if for all efficient adversaries $\mathcal{A}$, the value* $\text{MSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

As discussed in Section 2.2.5, Attack Game 5.1 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\text{MSS}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]$ as $|\Pr[\hat{b} = b] - 1/2|$, and as usual (by (2.11)), we have

$$\text{MSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \text{MSS}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]. \tag{5.1}$$

As the next theorem shows, semantic security implies multi-key semantic security.

Theorem 5.1. *If a cipher $\mathcal{E}$ is semantically secure, it is also multi-key semantically secure.*

> *In particular, for every MSS adversary $\mathcal{A}$ that attacks $\mathcal{E}$ as in Attack Game 5.1, and which makes at most Q queries to its challenger, there exists an SS adversary $\mathcal{B}$ that attacks $\mathcal{E}$ as in Attack Game 2.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{MSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = Q \cdot \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}].$$

Proof idea. The proof is a straightforward hybrid argument, which is a proof technique we introduced in the proofs of Theorem 3.2 and 3.3 (the reader is advised to review those proofs, if necessary). In Experiment 0 of the MSS attack game, the challenger is encrypting $m_{10}, m_{20}, \ldots, m_{Q0}$. Intuitively, since the key k_1 is only used to encrypt the first message, and $\mathcal{E}$ is semantically secure, if we modify the challenger so that it encrypts m_{11} instead of m_{10}, the adversary should not behave significantly differently. Similarly, we may modify the challenger so that it encrypts m_{21} instead of m_{20}, and the adversary should not notice the difference. If we continue in this way, making a total of Q modifications to the challenger, we end up in Experiment 1 of the MSS game, and the adversary should not notice the difference. □

Proof. Suppose $\mathcal{E} = (E, D)$ is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$. Let $\mathcal{A}$ be an MSS adversary that plays Attack Game 5.1 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger in that game.

First, we introduce $Q + 1$ hybrid games, Hybrid 0, ..., Hybrid Q, played between a challenger and $\mathcal{A}$. For $j = 0, 1, \ldots, Q$, when $\mathcal{A}$ makes its ith query (m_{i0}, m_{i1}), the challenger in Hybrid j computes its response c_i as follows:

```
$k_i \stackrel{R}{\leftarrow} \mathcal{K}$
if $i > j$ then   $c_i \stackrel{R}{\leftarrow} E(k_i, m_{i0})$   else   $c_i \stackrel{R}{\leftarrow} E(k_i, m_{i1})$.
```

Put another way, the challenger in Hybrid j encrypts

$$m_{11}, \ldots, m_{j1}, \quad m_{(j+1)0}, \ldots, m_{Q0},$$

generating different keys for each of these encryptions.

For $j = 0, 1, \ldots, Q$, let p_j denote the probability that $\mathcal{A}$ outputs 1 in Hybrid j. Observe that p_0 is equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 0 of Attack Game 5.1 with respect to $\mathcal{E}$, while p_Q is equal to the probability that $\mathcal{A}$ outputs 1 in Experiment 1 of Attack Game 5.1 with respect to $\mathcal{E}$. Therefore, we have

$$\text{MSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = |p_Q - p_0|. \tag{5.2}$$

We next devise an SS adversary $\mathcal{B}$ that plays Attack Game 2.1 with respect to $\mathcal{E}$, as follows:

First, $\mathcal{B}$ chooses $\omega \in \{1, \ldots, Q\}$ at random.

Then, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$ — when $\mathcal{A}$ makes its ith query (m_{i0}, m_{i1}), $\mathcal{B}$ computes its response c_i as follows:

```
if $i > \omega$ then
    $k_i \stackrel{R}{\leftarrow} \mathcal{K}$,  $c_i \stackrel{R}{\leftarrow} E(k_i, m_{i0})$
else if $i = \omega$ then
    $\mathcal{B}$ submits $(m_{i0}, m_{i1})$ to its own challenger
    $c_i$ is set to the challenger's response
else   //   $i < \omega$
    $k_i \stackrel{R}{\leftarrow} \mathcal{K}$,  $c_i \stackrel{R}{\leftarrow} E(k_i, m_{i1})$.
```

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Put another way, adversary $\mathcal{B}$ encrypts

$$m_{11}, \ldots, m_{(\omega-1)1},$$

generating its own keys for this purpose, submits $(m_{\omega 0}, m_{\omega 1})$ to its own encryption oracle, and encrypts

$$m_{(\omega+1)0}, \ldots, m_{Q0},$$

again, generating its own keys.

We claim that

$$\text{MSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = Q \cdot \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{5.3}$$

To prove this claim, for $b = 0, 1$, let W_b be the event that $\mathcal{B}$ outputs 1 in Experiment b of its attack game. If ω denotes the random number chosen by $\mathcal{B}$, then the key observation is that for $j = 1, \ldots, Q$, we have:

$$\Pr[W_0 \mid \omega = j] = p_{j-1} \quad \text{and} \quad \Pr[W_1 \mid \omega = j] = p_j.$$

Equation (5.3) now follows from this observation, together with (5.2), via the usual telescoping sum calculation:

$$\begin{aligned} \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}] &= |\Pr[W_1] - \Pr[W_0]| \\ &= \frac{1}{Q} \cdot \left| \sum_{j=1}^{Q} \Pr[W_1 \mid \omega = j] - \sum_{j=1}^{Q} \Pr[W_0 \mid \omega = j] \right| \\ &= \frac{1}{Q} \cdot |p_Q - p_0| \\ &= \frac{1}{Q} \cdot \text{MSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}], \end{aligned}$$

and the claim, and hence the theorem, is proved. □

Let us return now to the “file encryption” problem discussed in the introduction to this chapter. What this theorem says is that if Alice uses independent keys to encrypt each of her files with a semantically secure cipher, then an adversary who sees the ciphertexts stored on the file server will effectively learn nothing about Alice's files (except possibly some information about their lengths). Notice that this holds even if the adversary plays an active role in determining the contents of some of the files (e.g., by sending Alice an email, as discussed in the introduction).

5.3 Semantic security against chosen plaintext attack

Now we consider the problem that Alice faced in the introduction of this chapter, where she wants to encrypt all of her files on her system using a single, and hopefully short, secret key. The right notion of security for this task is **semantic security against chosen plaintext attack**, or **CPA security** for short.

Attack Game 5.2 (CPA security). For a given cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define

Experiment b:

- The challenger selects $k \overset{\text{R}}{\leftarrow} \mathcal{K}$.

- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots,$ the ith query is a pair of messages, $m_{i0}, m_{i1} \in \mathcal{M}$, of the same length.

  The challenger computes $c_i \xleftarrow{\text{R}} E(k, m_{ib})$, and sends c_i to the adversary.

- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - \Pr[W_1]|. \quad \Box$$

The only difference between the CPA attack game and the MSS Attack Game 5.1 is that in the CPA game, the same key is used for all encryptions, whereas in the MSS attack game, a different key is chosen for each encryption. In particular, the adversary's queries may be adaptively chosen in the CPA game, just as in the MSS game.

Definition 5.2 (CPA security). *A cipher $\mathcal{E}$ is called* ***semantically secure against chosen plaintext attack****, or simply* ***CPA secure****, if for all efficient adversaries $\mathcal{A}$, the value* $\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

As in Section 2.2.5, Attack Game 5.2 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$; we define $\mathcal{A}$'s *bit-guessing advantage* as $\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] := |\Pr[\hat{b} = b] - 1/2|$, and as usual (by (2.11)), we have

$$\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]. \tag{5.4}$$

Again, we return to the "file encryption" problem discussed in the introduction to this chapter. What this definition says is that if Alice uses just a single key to encrypt each of her files with a CPA secure cipher, then an adversary who sees the ciphertexts stored on the file server will effectively learn nothing about Alice's files (except possibly some information about their lengths). Again, notice that this holds even if the adversary plays an active role in determining the contents of some of the files.

Example 5.1. Just to exercise the definition a bit, let us show that no deterministic cipher can possibly satisfy the definition of CPA security. Suppose that $\mathcal{E} = (E, D)$ is a deterministic cipher. We construct a CPA adversary $\mathcal{A}$ as follows. Let m, m' be any two, distinct messages in the message space of $\mathcal{E}$. The adversary $\mathcal{A}$ makes two queries to its challenger: the first is (m, m'), and the second is (m, m). Suppose c_1 is the challenger's response to the first query and c_2 is the challenger's response to the second query. Adversary $\mathcal{A}$ outputs 1 if $c_1 = c_2$, and 0 otherwise.

Let us calculate $\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$. On the one hand, in Experiment 0 of Attack Game 5.2, the challenger encrypts m in responding to both queries, and so $c_1 = c_2$; hence, $\mathcal{A}$ outputs 1 with probability 1 in this experiment (this is precisely where we need the assumption that $\mathcal{E}$ is deterministic). On the other hand, in Experiment 1, the challenger encrypts m' and m, and so $c_1 \neq c_2$; hence, $\mathcal{A}$ outputs 1 with probability 0 in this experiment. It follows that $\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 1$.

The attack in this example can be generalized to show that not only must a CPA-secure cipher be probabilistic, but it must be very unlikely that two encryptions of the same message yield the same ciphertext — see Exercise 5.12. $\Box$

Remark 5.1. Analogous to Theorem 5.1, it is straightforward to show that if a cipher is CPA-secure, it is also CPA-secure in the multi-key setting. See Exercise 5.2. $\Box$

5.4 Building CPA secure ciphers

In this section, we describe a number of ways of building ciphers that are semantically secure against chosen plaintext attack. As we have already discussed in Example 5.1, any such cipher must be probabilistic. We begin in Section 5.4.1 with a generic construction that combines any semantically secure cipher with a pseudo-random function (PRF). The PRF is used to generate "one time" keys. Next, in Section 5.4.2, we develop a probabilistic variant of the *counter mode* cipher discussed in Section 4.4.4. While this scheme can be based on any PRF, in practice, the PRF is usually instantiated with a block cipher. Finally, in Section 5.4.3, we present a cipher that is constructed from a block cipher using a method called *cipher block chaining (CBC) mode.*

These last two constructions, counter mode and CBC mode, are called *modes of operation of a block cipher.* Another mode of operation we have already seen in Section 4.1.4 is *electronic codebook (ECB) mode.* However, because of the lack of security provided by this mode of operation, it is seldom used. There are other modes of operations that provide CPA security, which we develop in the exercises.

5.4.1 A generic hybrid construction

In this section, we show how to turn any semantically secure cipher $\mathcal{E} = (E, D)$ into a CPA secure cipher $\mathcal{E}'$ using an appropriate PRF F.

The basic idea is this. A key for $\mathcal{E}'$ is a key k' for F. To encrypt a single message m, a random input x for F is chosen, and a key k for $\mathcal{E}$ is derived by computing $k \leftarrow F(k', x)$. Then m is encrypted using this key k: $c \xleftarrow{R} E(k, m)$. The ciphertext is $c' := (x, c)$. Note that we need to include x as part of c' so that we can decrypt: the decryption algorithm first derives the key k by computing $k \leftarrow F(k', x)$, and then recovers m by computing $m \leftarrow D(k, c)$.

For all of this to work, the output space of F must match the key space of $\mathcal{E}$. Also, the input space of F must be super-poly, so that the chances of accidentally generating the same x value twice is negligible.

Now the details. Let $\mathcal{E} = (E, D)$ be a cipher, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Let F be a PRF defined over $(\mathcal{K}', \mathcal{X}, \mathcal{K})$; that is, the output space of F should be equal to the key space of $\mathcal{E}$. We define a new cipher $\mathcal{E}' = (E', D')$, defined over $(\mathcal{K}', \mathcal{M}, \mathcal{X} \times \mathcal{C})$, as follows:

- for $k' \in \mathcal{K}'$ and $m \in \mathcal{M}$, we define

$$\begin{aligned} E'(k', m) := \quad & x \xleftarrow{R} \mathcal{X}, \quad k \leftarrow F(k', x), \quad c \xleftarrow{R} E(k, m) \\ & \text{output } (x, c); \end{aligned}$$

- for $k' \in \mathcal{K}'$ and $c' = (x, c) \in \mathcal{X} \times \mathcal{C}$, we define

$$\begin{aligned} D'(k', c') := \quad & k \leftarrow F(k', x), \quad m \leftarrow D(k, c) \\ & \text{output } m. \end{aligned}$$

It is easy to verify that $\mathcal{E}'$ is indeed a cipher, and is our first example of a *probabilistic* cipher.

Example 5.2. Before proving CPA security of $\mathcal{E}'$ let us first see the construction in action. Suppose $\mathcal{E}$ is the one-time pad, namely $E(k, m) := k \oplus m$ where $\mathcal{K} = \mathcal{M} = \mathcal{C} = \{0, 1\}^L$. Applying the generic hybrid construction above to the one-time pad results in the following popular cipher $\mathcal{E}_0 = (E_0, D_0)$:

- for $k' \in \mathcal{K}'$ and $m \in \mathcal{M}$, define

$$E_0(k', m) := \quad x \xleftarrow{\text{R}} \mathcal{X}, \;\; \text{output } (x, \; F(k', x) \oplus m)$$

- for $k' \in \mathcal{K}'$ and $c' = (x, c) \in \mathcal{X} \times \mathcal{C}$, define

$$D_0(k', c') := \text{ output } F(k', x) \oplus c$$

CPA security of this cipher follows from the CPA security of the generic hybrid construction $\mathcal{E}'$ which is proved in Theorem 5.2 below. □

Theorem 5.2. *If F is a secure PRF, $\mathcal{E}$ is a semantically secure cipher, and $N := |\mathcal{X}|$ is super-poly, then the cipher $\mathcal{E}'$ described above is a CPA secure cipher.*

> *In particular, for every CPA adversary $\mathcal{A}$ that attacks $\mathcal{E}'$ as in the bit-guessing version of Attack Game 5.2, and which makes at most Q queries to its challenger, there exists a PRF adversary $\mathcal{B}_F$ that attacks F as in Attack Game 4.2, and an SS adversary $\mathcal{B}_\mathcal{E}$ that attacks $\mathcal{E}$ as in the bit-guessing version of Attack Game 2.1, where both $\mathcal{B}_F$ and $\mathcal{B}_\mathcal{E}$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}'] \leq \frac{Q^2}{N} + 2 \cdot \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + Q \cdot \text{SS}\mathsf{adv}[\mathcal{B}_\mathcal{E}, \mathcal{E}]. \tag{5.5}$$

Proof idea. First, using the assumption that F is a PRF, we can effectively replace F by a truly random function. Second, using the assumption that N is super-poly, we argue that except with negligible probability, no two x-values are ever the same. But in this scenario, the challenger's keys are now all independently generated, and so the challenger is really playing the same role as the challenger in the Attack Game 5.1. The result then follows from Theorem 5.1. □

Proof. Let $\mathcal{A}$ be an efficient CPA adversary that attacks $\mathcal{E}'$ as in Attack Game 5.2. Assume that $\mathcal{A}$ makes at most Q queries to its challenger. Our goal is to show that CPA$\mathsf{adv}[\mathcal{A}, \mathcal{E}']$ is negligible, assuming that F is a secure PRF, that N is super-poly, and that $\mathcal{E}$ is semantically secure.

It is convenient to use the bit-guessing versions of the CPA and semantic security attack games. We prove:

$$\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] \leq \frac{Q^2}{2N} + \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + Q \cdot \text{SS}\mathsf{adv}^*[\mathcal{B}_\mathcal{E}, \mathcal{E}] \tag{5.6}$$

for efficient adversaries $\mathcal{B}_F$ and $\mathcal{B}_\mathcal{E}$. Then (5.5) follows from (5.4) and Theorem 2.10.

The basic strategy of the proof is as follows. First, we define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the bit-guessing version of Attack Game 5.2 with respect to $\mathcal{E}'$. We then define several more games: Game 1, Game 2, and Game 3. Each of these games are played between $\mathcal{A}$ and a different challenger; moreover, as we shall see, Game 3 is equivalent to the bit-guessing version of Attack Game 5.1 with respect to $\mathcal{E}$. In each of these games, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, \ldots, 3$, we define W_j to be the event that $\hat{b} = b$ in Game j. We will show that for $j = 1, \ldots, 3$, the value $|\Pr[W_j] - \Pr[W_{j-1}]|$ is negligible; moreover, from the assumption that $\mathcal{E}$ is semantically secure, and from Theorem 5.1, it will follow that $|\Pr[W_3] - 1/2|$ is negligible; from this, it follows that CPA$\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] := |\Pr[W_0] - 1/2|$ is negligible.

Game 0. Let us begin by giving a detailed description of the challenger in Game 0 that is convenient for our purposes:

```
$b \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}$
$k' \stackrel{\mathrm{R}}{\leftarrow} \mathcal{K}'$
for $i \leftarrow 1$ to $Q$ do
    $x_i \stackrel{\mathrm{R}}{\leftarrow} \mathcal{X}$
    $k_i \leftarrow F(k', x_i)$
upon receiving the $i$th query $(m_{i0}, m_{i1}) \in \mathcal{M}^2$:
    $c_i \stackrel{\mathrm{R}}{\leftarrow} E(k_i, m_{ib})$
    send $(x_i, c_i)$ to the adversary.
```

By construction, we have

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] = \big|\Pr[W_0] - 1/2\big|. \tag{5.7}$$

Game 1. Next, we play our "PRF card," replacing $F(k', \cdot)$ by a truly random function $f \in \mathrm{Funs}[\mathcal{X}, \mathcal{K}]$. The challenger in this game looks like this:

```
$b \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}$
$f \stackrel{\mathrm{R}}{\leftarrow} \mathrm{Funs}[\mathcal{X}, \mathcal{K}]$
for $i \leftarrow 1$ to $Q$ do
    $x_i \stackrel{\mathrm{R}}{\leftarrow} \mathcal{X}$
    $k_i \leftarrow f(x_i)$
upon receiving the $i$th query $(m_{i0}, m_{i1}) \in \mathcal{M}^2$:
    $c_i \stackrel{\mathrm{R}}{\leftarrow} E(k_i, m_{ib})$
    send $(x_i, c_i)$ to the adversary.
```

We claim that

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F], \tag{5.8}$$

where $\mathcal{B}_F$ is an efficient PRF adversary; moreover, since we are assuming that F is a secure PRF, it must be the case that $\mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F]$ is negligible.

The design of $\mathcal{B}_F$ is naturally suggested by the syntax of Games 0 and 1. If $f \in \mathrm{Funs}[\mathcal{X}, \mathcal{K}]$ denotes the function chosen by its challenger in Attack Game 4.2 with respect to F, adversary $\mathcal{B}_F$ runs as follows:

First, $\mathcal{B}_F$ makes the following computations:

```
$b \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}$
for $i \leftarrow 1$ to $Q$ do
    $x_i \stackrel{\mathrm{R}}{\leftarrow} \mathcal{X}$
    $k_i \stackrel{\mathrm{R}}{\leftarrow} f(x_i)$.
```

Here, $\mathcal{B}_F$ obtains the value $f(x_i)$ by querying its own challenger with x_i.

Next, adversary $\mathcal{B}_F$ plays the role of challenger to $\mathcal{A}$; specifically, when $\mathcal{A}$ makes its ith query (m_{i0}, m_{i1}), adversary $\mathcal{B}_F$ computes

$$c_i \stackrel{\mathrm{R}}{\leftarrow} E(k_i, m_{ib})$$

and sends (x_i, c_i) to $\mathcal{A}$.

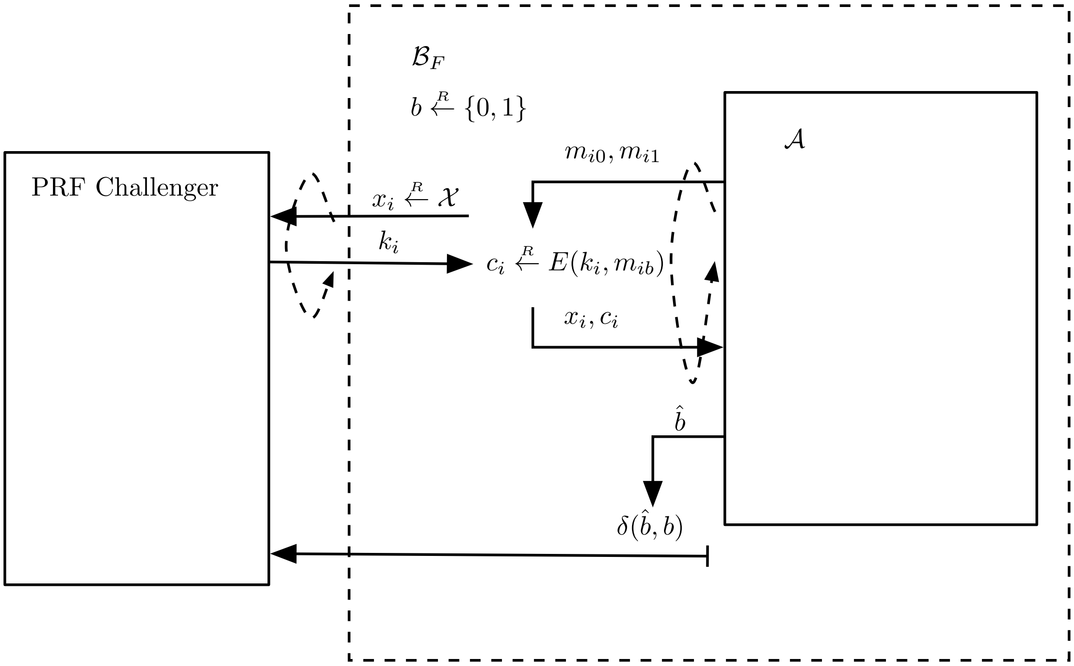


Figure 5.1: Adversary $\mathcal{B}_F$ in the proof of Theorem 5.2

> Eventually, $\mathcal{A}$ halts and outputs a bit $\hat{b}$, at which time adversary $\mathcal{B}_F$ halts and outputs 1 if $\hat{b} = b$, and outputs 0 otherwise.

See Fig. 5.1 for a picture of adversary $\mathcal{B}_F$. As usual, $\delta(x, y)$ is defined to be 1 if $x = y$, and 0 otherwise.

Game 2. Next, we use our "faithful gnome" idea (see Section 4.4.2) to implement the random function f. Our "gnome" has to keep track of the inputs to f, and detect if the same input is used twice. In the following logic, our gnome uses a truly random key as the "default" value for k_i, but over-rides this default value if necessary, as indicated in the line marked $(*)$:

```
      $b \xleftarrow{R} \{0,1\}$
      for $i \leftarrow 1$ to $Q$ do
          $x_i \xleftarrow{R} \mathcal{X}$
          $k_i \xleftarrow{R} \mathcal{K}$
($*$)     if $x_i = x_j$ for some $j < i$ then $k_i \leftarrow k_j$
      upon receiving the $i$th query $(m_{i0}, m_{i1}) \in \mathcal{M}^2$:
          $c_i \xleftarrow{R} E(k_i, m_{ib})$
          send $(x_i, c_i)$ to the adversary.
```

As this is a faithful implementation of the random function f, we have

$$\Pr[W_2] = \Pr[W_1]. \tag{5.9}$$

Game 3. Next, we make our gnome "forgetful," simply dropping the line marked $(*)$ in the previous game:

```
$b \xleftarrow{R} \{0,1\}$
for $i \leftarrow 1$ to $Q$ do
    $x_i \xleftarrow{R} \mathcal{X}$
    $k_i \xleftarrow{R} \mathcal{K}$
upon receiving the $i$th query $(m_{i0}, m_{i1}) \in \mathcal{M}^2$:
    $c_i \xleftarrow{R} E(k_i, m_{ib})$
    send $(x_i, c_i)$ to the adversary.
```

To analyze the quantity $|\Pr[W_3] - \Pr[W_2]|$, we use the Difference Lemma (Theorem 4.7). To this end, we view Games 2 and 3 as operating on the same underlying probability space: the random choices made by the adversary and the challenger are identical in both games — all that differs is the rule used by the challenger to compute its responses. In particular, the variables x_i are identical in both games. Define Z to be the event that $x_i = x_j$ for some $i \neq j$. Clearly, Games 2 and 3 proceed identically unless Z occurs; in particular, $W_2 \wedge \bar{Z}$ occurs if and only if $W_3 \wedge \bar{Z}$ occurs. Applying the Difference Lemma, we therefore have

$$\big|\Pr[W_3] - \Pr[W_2]\big| \leq \Pr[Z]. \tag{5.10}$$

Moreover, it is easy to see that

$$\Pr[Z] \leq \frac{Q^2}{2N}, \tag{5.11}$$

since Z is the union of less than $Q^2/2$ events, each of which occurs with probability $1/N$.

Observe that in Game 3, independent encryption keys k_i are used to encrypt each message. So next, we play our "semantic security card," claiming that

$$|\Pr[W_3] - 1/2| = \mathrm{MSS}\mathsf{adv}^*[\bar{\mathcal{B}}_{\mathcal{E}}, \mathcal{E}], \tag{5.12}$$

where $\bar{\mathcal{B}}_{\mathcal{E}}$ is an efficient adversary that plays the bit-guessing version of Attack Game 5.1 with respect to $\mathcal{E}$, making at most Q queries to its challenger in that game.

The design of $\bar{\mathcal{B}}_{\mathcal{E}}$ is naturally suggested by the syntactic form of Game 3. It works as follows:

> Playing the role of challenger to $\mathcal{A}$, upon receiving the ith query (m_{i0}, m_{i1}) from $\mathcal{A}$, adversary $\bar{\mathcal{B}}_{\mathcal{E}}$ submits (m_{i0}, m_{i1}) to its own challenger, obtaining a ciphertext $c_i \in \mathcal{C}$; then $\bar{\mathcal{B}}_{\mathcal{E}}$ selects x_i at random from $\mathcal{X}$, and sends (x_i, c_i) to $\mathcal{A}$ in response to the latter's query.
>
> When $\mathcal{A}$ finally outputs a bit $\hat{b}$, $\bar{\mathcal{B}}_{\mathcal{E}}$ outputs this same bit.

See Fig. 5.2 for a picture of adversary $\bar{\mathcal{B}}_{\mathcal{E}}$.

It is evident from the construction (and (2.11)) that (5.12) holds. Moreover, by Theorem 5.1 and (5.1), we have

$$\mathrm{MSS}\mathsf{adv}^*[\bar{\mathcal{B}}_{\mathcal{E}}, \mathcal{E}] = Q \cdot \mathrm{SS}\mathsf{adv}^*[\mathcal{B}_{\mathcal{E}}, \mathcal{E}], \tag{5.13}$$

where $\mathcal{B}_{\mathcal{E}}$ is an efficient adversary playing the bit-guessing version of Attack Game 2.1 with respect to $\mathcal{E}$.

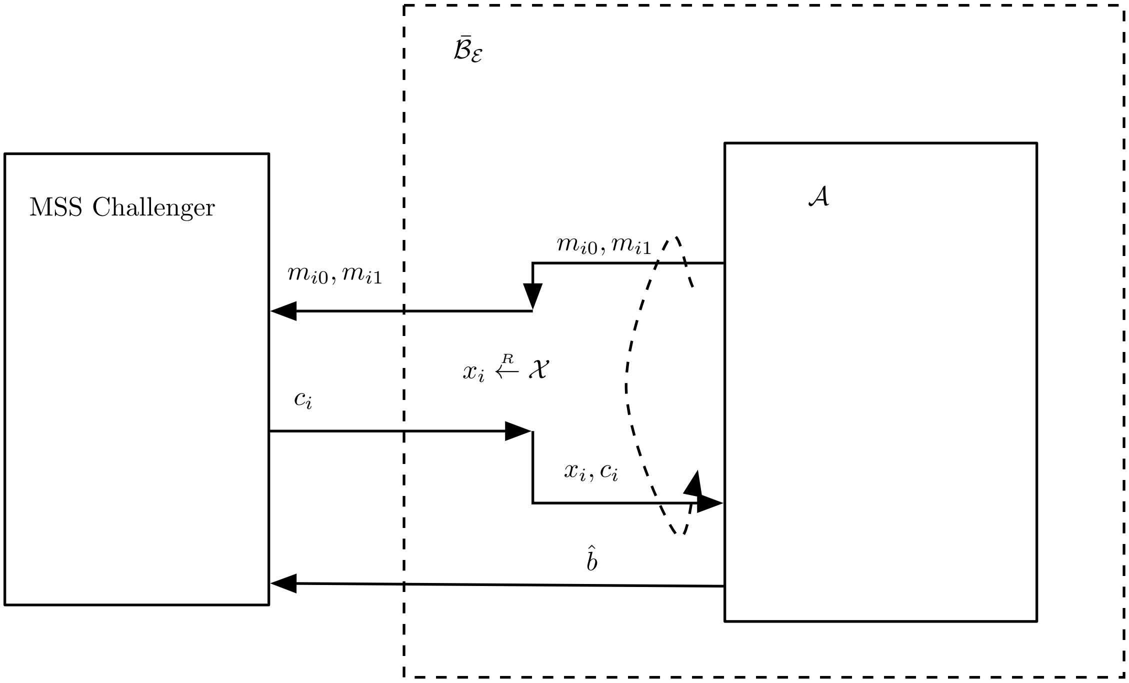


Figure 5.2: Adversary $\bar{\mathcal{B}}_{\mathcal{E}}$ in the proof of Theorem 5.2

Putting together (5.7) through (5.13), we obtain (5.6). Also, one can check that the running times of both $\mathcal{B}_F$ and $\mathcal{B}_\mathcal{E}$ are roughly the same as that of $\mathcal{A}$; indeed, they are elementary wrappers around $\mathcal{A}$, and (5.5) holds regardless of whether $\mathcal{A}$ is efficient. □

While the above proof was a bit long, we hope the reader agrees that it was in fact quite natural, and that all of the steps were fairly easy to follow. Also, this proof illustrates how one typically employs more than one security assumption in devising a security proof as a sequence of games.

Remark 5.2. We briefly mention that the hybrid construction $\mathcal{E}'$ in Theorem 5.2 is CPA secure even if the PRF F used in the construction is only weakly secure (as in Definition 4.3). To prove Theorem 5.2 under this weaker assumption observe that in both Games 0 and 1 the challenger only evaluates the PRF at *random* points in $\mathcal{X}$. Therefore, the adversary's advantage in distinguishing Games 0 and 1 is negligible even if F is only weakly secure. □

5.4.2 Randomized counter mode

We can build a CPA secure cipher directly out of a secure PRF, as follows. Suppose F is a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$. We shall assume that $\mathcal{X} = \{0, \dots, N-1\}$, and that $\mathcal{Y} = \{0,1\}^n$.

For any poly-bounded $\ell \geq 1$, we define a cipher $\mathcal{E} = (E, D)$, with key space $\mathcal{K}$, message space $\mathcal{Y}^{\leq \ell}$, and ciphertext space $\mathcal{X} \times \mathcal{Y}^{\leq \ell}$, as follows:

- for $k \in \mathcal{K}$ and $m \in \mathcal{Y}^{\leq \ell}$, with $v := |m|$, we define

```
$E(k, m) :=$
    $x \xleftarrow{R} \mathcal{X}$
    compute $c \in \mathcal{Y}^v$ as follows:
        for $j \leftarrow 0$ to $v-1$ do
            $c[j] \leftarrow F(k, x + j \bmod N) \oplus m[j]$
    output $(x, c)$;
```

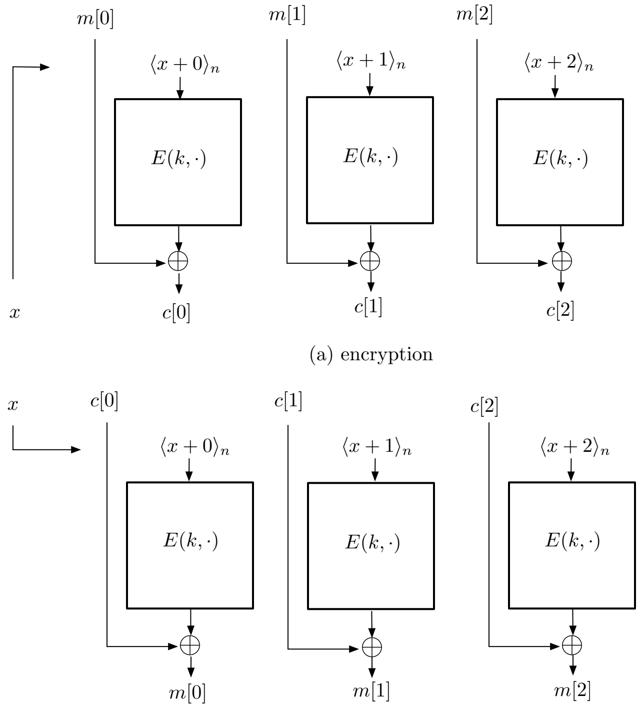
- for $k \in \mathcal{K}$ and $c' = (x, c) \in \mathcal{X} \times \mathcal{Y}^{\leq \ell}$, with $v := |c|$, we define

```
$D(k, c') :=$
    compute $m \in \mathcal{Y}^v$ as follows:
        for $j \leftarrow 0$ to $v-1$ do
            $m[j] \leftarrow F(k, x + j \bmod N) \oplus c[j]$
    output $m$.
```

This cipher is much like the stream cipher one would get by building a PRG out of F using the construction in Section 4.4.4. The difference is that instead of using a fixed sequence of inputs to F to derive a key stream, we use a random starting point, which we then increment to obtain successive inputs to F. The x component of the ciphertext is typically called an **initial value**, or **IV** for short.

In practice, F is typically implemented using the encryption function of a block cipher, and $\mathcal{X} = \mathcal{Y} = \{0,1\}^n$, where we naturally view n-bit strings as numbers in the range $0, \dots, 2^n - 1$. As it happens, the decryption function of the block cipher is not needed at all in this construction. See Fig. 5.3 for an illustration of this mode.

It is easy to verify that $\mathcal{E}$ is indeed a (probabilistic) cipher. Also, note that the message space of $\mathcal{E}$ is variable length, and that for the purposes of defining CPA security using Attack Game 5.2, the length of a message $m \in \mathcal{Y}^{\leq \ell}$ is its natural length $|m|$.



(a) encryption

(b) decryption

Figure 5.3: Randomized counter mode ($v = 3$)

Theorem 5.3. *If F is a secure PRF and N is super-poly, then for any poly-bounded $\ell \geq 1$, the cipher $\mathcal{E}$ described above is a CPA secure cipher.*

> *In particular, for every CPA adversary $\mathcal{A}$ that attacks $\mathcal{E}$ as in Attack Game 5.2, and which makes at most Q queries to its challenger, there exists a PRF adversary $\mathcal{B}$ that attacks F as in Attack Game 4.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq \frac{2Q^2\ell}{N} + 2 \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F]. \tag{5.14}$$

Proof idea. Suppose we start with an adversary that plays the CPA attack game with respect to $\mathcal{E}$. First, using the assumption that F is a PRF, we can effectively replace F by a truly random function f. Second, using the assumption that N is super-poly, and the fact that each IV is chosen at random, we can argue that except with negligible probability, the challenger never evaluates f at the same point twice. But in this case, the challenger is effectively encrypting each message using an independent one-time pad, and so we can conclude that the adversary's advantage in the original CPA attack game is negligible. □

Proof. Let $\mathcal{A}$ be an efficient adversary that plays Attack Game 5.2 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger in that game. We want to show that $\mathrm{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible, assuming that F is a secure PRF and that N is super-poly.

It is convenient to use the bit-guessing version of the CPA attack game. We prove:

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] \leq \frac{Q^2\ell}{N} + \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] \tag{5.15}$$

for an efficient adversary $\mathcal{B}$. Then (5.14) follows from (5.4).

The basic strategy of the proof is as follows. First, we define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the bit-guessing version of Attack Game 5.2 with respect to $\mathcal{E}$. We then define several more games: Game 1, Game 2, and Game 3. Each of these games is played between $\mathcal{A}$ and a different challenger. In each of these games, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, \ldots, 3$, we define W_j to be the event that $\hat{b} = b$ in Game j. We will show that for $j = 1, \ldots, 3$, the value $|\Pr[W_j] - \Pr[W_{j-1}]|$ is negligible; moreover, it will be evident that $\Pr[W_3] = 1/2$, from which it will follow that $\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - 1/2|$ is negligible.

Game 0. We may describe the challenger in Game 0 as follows:

```
$b \overset{R}{\leftarrow} \{0,1\}$
$k \overset{R}{\leftarrow} \mathcal{K}$
for $i \leftarrow 1$ to $Q$ do
    $x_i \overset{R}{\leftarrow} \mathcal{X}$
    for $j \leftarrow 0$ to $\ell - 1$ do
        $x'_{ij} \leftarrow x_i + j \bmod N$
        $y_{ij} \leftarrow F(k, x'_{ij})$
upon receiving the $i$th query $(m_{i0}, m_{i1})$, with $v_i := |m_{i0}| = |m_{i1}|$:
    compute $c_i \in \mathcal{Y}^{v_i}$ as follows:
        for $j \leftarrow 0$ to $v_i - 1$ do: $c_i[j] \leftarrow y_{ij} \oplus m_{ib}[j]$
    send $(x_i, c_i)$ to the adversary.
```

By construction, we have

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}] = \big|\Pr[W_0] - 1/2\big|. \tag{5.16}$$

Game 1. Next, we play our "PRF card," replacing $F(k,\cdot)$ by a truly random function $f \in \mathrm{Funs}[\mathcal{X},\mathcal{Y}]$. The challenger in this game looks like this:

```
$b \xleftarrow{R} \{0,1\}$
$f \xleftarrow{R} \mathrm{Funs}[\mathcal{X},\mathcal{Y}]$
for $i \leftarrow 1$ to $Q$ do
    $x_i \xleftarrow{R} \mathcal{X}$
    for $j \leftarrow 0$ to $\ell - 1$ do
        $x'_{ij} \leftarrow x_i + j \bmod N$
        $y_{ij} \leftarrow f(x'_{ij})$
...
```

We have left out part of the code for the challenger, as it will not change in any of our games. We claim that

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \mathrm{PRF}\mathsf{adv}[\mathcal{B},F], \tag{5.17}$$

where $\mathcal{B}$ is an efficient adversary; moreover, since we are assuming that F is a secure PRF, it must be the case that $\mathrm{PRF}\mathsf{adv}[\mathcal{B},F]$ is negligible. This is hopefully (by now) a routine argument, and we leave the details of this to the reader.

Game 2. Next, we use our "faithful gnome" idea to implement the random function f. In describing the logic of our challenger in this game, we use the standard lexicographic ordering on pairs of indices (i,j); that is, $(i',j') < (i,j)$ if and only if

$$i' < i \quad \text{or} \quad i' = i \text{ and } j' < j.$$

In the following logic, our "gnome" uses a truly random value as the "default" value for each y_{ij}, but over-rides this default value if necessary, as indicated in the line marked $(*)$:

```
    $b \xleftarrow{R} \{0,1\}$
    for $i \leftarrow 1$ to $Q$ do
        $x_i \xleftarrow{R} \mathcal{X}$
        for $j \leftarrow 0$ to $\ell - 1$ do
            $x'_{ij} \leftarrow x_i + j \bmod N$
            $y_{ij} \xleftarrow{R} \mathcal{Y}$
(*)         if $x'_{ij} = x'_{i'j'}$ for some $(i',j') < (i,j)$ then $y_{ij} \leftarrow y_{i'j'}$
        ...
```

As this is a faithful implementation of the random function f, we have

$$\Pr[W_2] = \Pr[W_1]. \tag{5.18}$$

Game 3. Now we make our gnome "forgetful," dropping the line marked $(*)$ in the previous game:

```
$b \xleftarrow{R} \{0,1\}$
for $i \leftarrow 1$ to $Q$ do
    $x_i \xleftarrow{R} \mathcal{X}$
    for $j \leftarrow 0$ to $\ell - 1$ do
        $x'_{ij} \leftarrow x_i + j \bmod N$
        $y_{ij} \xleftarrow{R} \mathcal{Y}$
    . . .
```

To analyze the quantity $|\Pr[W_3] - \Pr[W_2]|$, we use the Difference Lemma (Theorem 4.7). To this end, we view Games 2 and 3 as operating on the same underlying probability space: the random choices made by the adversary and the challenger are identical in both games — all that differs is the rule used by the challenger to compute its responses. In particular, the variables x'_{ij} are identical in both games. Define Z to be the event that $x'_{ij} = x'_{i'j'}$ for some $(i,j) \neq (i',j')$. Clearly, Games 2 and 3 proceed identically unless Z occurs; in particular, $W_2 \wedge \bar{Z}$ occurs if and only if $W_3 \wedge \bar{Z}$ occurs. Applying the Difference Lemma, we therefore have

$$\bigl|\Pr[W_3] - \Pr[W_2]\bigr| \leq \Pr[Z]. \tag{5.19}$$

We claim that

$$\Pr[Z] \leq \frac{Q^2 \ell}{N}. \tag{5.20}$$

To prove this claim, we may assume that $N \geq 2\ell$ (this should generally hold, since we are assuming that ℓ is poly-bounded and N is super-poly). Observe that Z occurs if and only if

$$\{x_i, \ldots, x_i + \ell - 1\} \cap \{x_{i'}, \ldots, x_{i'} + \ell - 1\} \neq \emptyset$$

for some pair of indices i and i' with $i \neq i'$ (and arithmetic is done mod N). Consider any fixed such pair of indices. Conditioned on any fixed value of x_i, the value $x_{i'}$ is uniformly distributed over $\{0, \ldots, N-1\}$, and the intervals overlap if and only if

$$x_{i'} \in \{x_i + j : -\ell + 1 \leq j \leq \ell - 1\},$$

which happens with probability $(2\ell - 1)/N$. The inequality (5.20) now follows, as there are $Q(Q-1)/2$ ways to choose i and i'.

Finally, observe that in Game 3 the y_{ij} values are uniformly and independently distributed over $\mathcal{Y}$, and thus the challenger is essentially using independent one-time pads to encrypt. In particular, it is easy to see that the adversary's output in this game is independent of b. Therefore,

$$\Pr[W_3] = 1/2. \tag{5.21}$$

Putting together (5.16) through (5.21), we obtain (5.15), and the theorem follows. □

Remark 5.3. One can also view randomized counter mode as a special case of the generic hybrid construction in Section 5.4.1. See Exercise 5.5. □

5.4.2.1 Case study: AES counter mode

The IPsec protocol uses a particular variant of AES counter mode, as specified in RFC 3686. Recall that AES uses a 128 bit block. Rather than picking a random 128-bit IV for every message, RFC 3686 picks the IV as follows:

- The most significant 32 bits are chosen at random at the time that the secret key is generated and are fixed for the life of the key. The same 32 bit value is used for all messages encrypted using this key.
- The next 64 bits are chosen at random in $\{0,1\}^{64}$.
- The least significant 32 bits are set to the number 1.

This resulting 128-bit IV is used as the initial value of the counter. When encrypting a message, the least significant 32 bits are incremented by one for every block of the message. Consequently, the maximum message length that can be encrypted is 2^{32} AES blocks or 2^{36} bytes.

With this choice of IV the decryptor knows the 32 most significant bits of the IV as well as the 32 least significant bits. Hence, only 64 bits of the IV need to be sent with the ciphertext.

The proof of Theorem 5.3 can be adapted to show that this method of choosing IVs is secure. The slight advantage of this method over picking a random 128-bit IV is that the resulting ciphertext is a little shorter. A random IV forces the encryptor to include all 128 bits in the ciphertext. With the method of RFC 3686 only 64 bits are needed, thus shrinking the ciphertext by 8 bytes.

5.4.3 CBC mode

A historically important encryption method is to use a block cipher in cipher block chaining (CBC) mode. This method is used in older versions of the TLS protocol (e.g., TLS 1.0). It is inferior to counter mode encryption as discussed in the next section.

Suppose $\mathcal{E} = (E, D)$ is a block cipher defined over $(\mathcal{K}, \mathcal{X})$, where $\mathcal{X} = \{0,1\}^n$. Let $N := |\mathcal{X}| = 2^n$. For any poly-bounded $\ell \geq 1$, we define a cipher $\mathcal{E}' = (E', D')$, with key space $\mathcal{K}$, message space $\mathcal{X}^{\leq \ell}$, and ciphertext space $\mathcal{X}^{\leq \ell+1} \setminus \mathcal{X}^0$; that is, the ciphertext space consists of all nonempty sequences of at most $\ell + 1$ data blocks. Encryption and decryption are defined as follows:

- for $k \in \mathcal{K}$ and $m \in \mathcal{X}^{\leq \ell}$, with $v := |m|$, we define

```
$E'(k, m) :=$
    compute $c \in \mathcal{X}^{v+1}$ as follows:
        $c[0] \xleftarrow{R} \mathcal{X}$
        for $j \leftarrow 0$ to $v - 1$ do
            $c[j+1] \leftarrow E(k,\ c[j] \oplus m[j])$
    output $c$;
```

- for $k \in \mathcal{K}$ and $c \in \mathcal{X}^{\leq \ell+1} \setminus \mathcal{X}^0$, with $v := |c| - 1$, we define

```
$D'(k, c) :=$
    compute $m \in \mathcal{X}^{v}$ as follows:
        for $j \leftarrow 0$ to $v - 1$ do
            $m[j] \leftarrow D(k,\ c[j+1]) \oplus c[j]$
    output $m$.
```

See Fig. 5.4 for an illustration of the encryption and decryption algorithm in the case $|m| = 3$. Here, the first component $c[0]$ of the ciphertext is also called an initial value, or IV. Note that unlike the counter mode construction in Section 5.4.2, in CBC mode, we must use a block cipher, as we actually need to use the decryption algorithm of the block cipher.

It is easy to verify that $\mathcal{E}'$ is indeed a (probabilistic) cipher. Also, note that the message space of $\mathcal{E}$ is variable length, and that for the purposes of defining CPA security using Attack Game 5.2, the length of a message $m \in \mathcal{X}^{\leq \ell}$ is its natural length $|m|$.

Theorem 5.4. *If $\mathcal{E} = (E, D)$ is a secure block cipher defined over $(\mathcal{K}, \mathcal{X})$, and $N := |\mathcal{X}|$ is super-poly, then for any poly-bounded $\ell \geq 1$, the cipher $\mathcal{E}'$ described above is a CPA secure cipher.*

> *In particular, for every CPA adversary $\mathcal{A}$ that attacks $\mathcal{E}'$ as in the bit-guessing version of Attack Game 5.2, and which makes at most Q queries to its challenger, there exists BC adversary $\mathcal{B}$ that attacks $\mathcal{E}$ as in Attack Game 4.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}'] \leq \frac{2Q^2\ell^2}{N} + 2 \cdot \text{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{5.22}$$

Proof idea. The basic idea of the proof is very similar to that of Theorem 5.3. We start with an adversary that plays the CPA attack game with respect to $\mathcal{E}'$. We then replace E by a truly random function f. Then we argue that except with negligible probability, the challenger never evaluates f at the same point twice. But then what the adversary sees is nothing but a bunch of random bits, and so learns nothing at all about the message being encrypted. □

Proof. Let $\mathcal{A}$ be an efficient CPA adversary that attacks $\mathcal{E}'$ as in Attack Game 5.2. Assume that $\mathcal{A}$ makes at most Q queries to its challenger in that game. We want to show that $\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}']$ is negligible, assuming that $\mathcal{E}$ is a secure block cipher and that N is super-poly. Under these assumptions, by Corollary 4.5, the encryption function E is a secure PRF, defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$.

It is convenient to use the bit-guessing version of the CPA attack game, We prove:

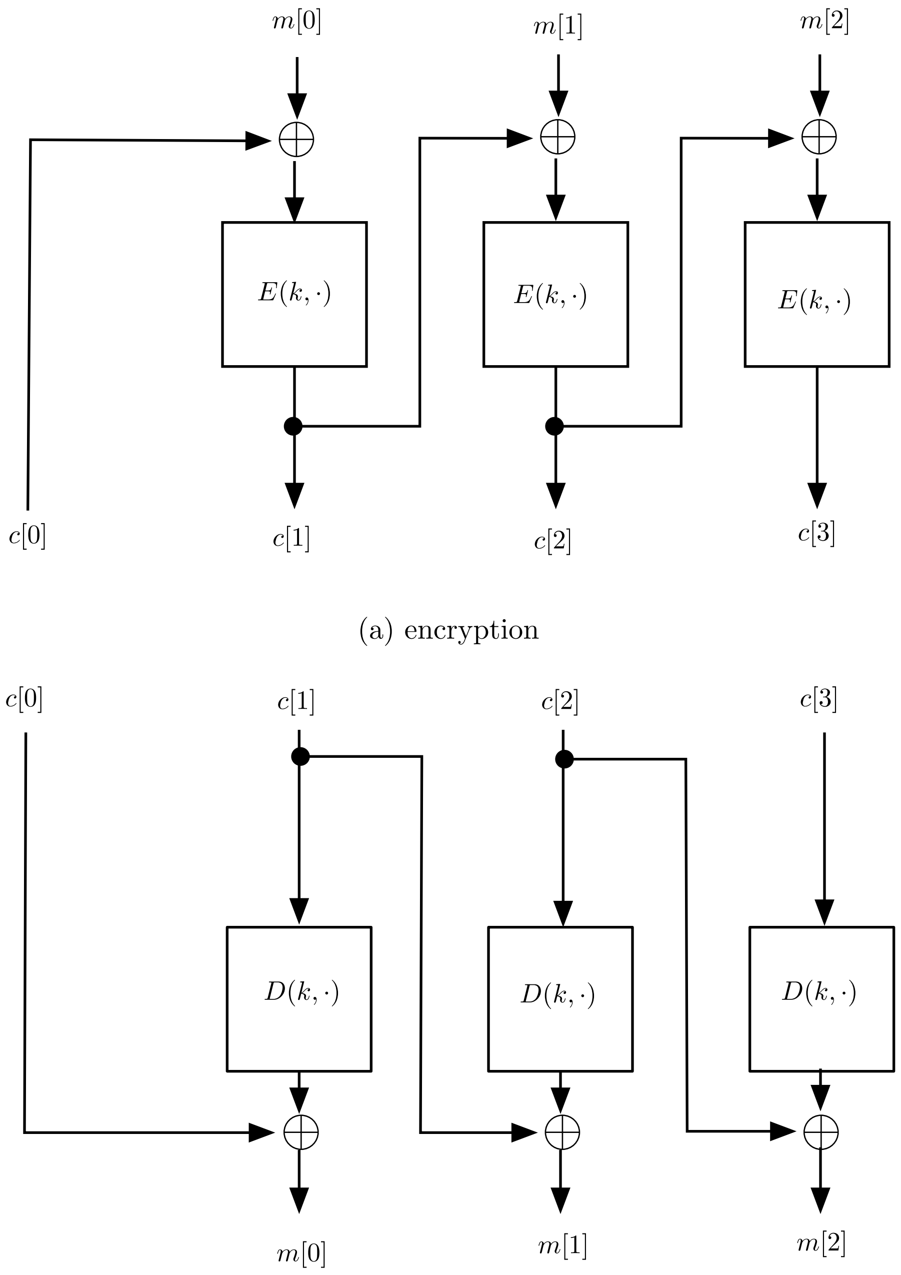
$$\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] \leq \frac{Q^2\ell^2}{N} + \text{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}] \tag{5.23}$$

for an efficient adversary $\mathcal{B}$. Then (5.22) follows from (5.4).

As usual, we define a sequence of games: Game 0, Game 1, Game 2, Game 3. Each of these games are played between $\mathcal{A}$ and a challenger. The challenger in Game 0 is the one from the bit-guessing version of Attack Game 5.2 with respect to $\mathcal{E}'$. In each of these games, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, \ldots, 3$, we define W_j to be the event that $\hat{b} = b$ in Game j. We will show that for $j = 1, \ldots, 3$, the value $|\Pr[W_j] - \Pr[W_{j-1}]|$ is negligible; moreover, it will be evident that $\Pr[W_3] = 1/2$, from which it will follow that $|\Pr[W_0] - 1/2|$ is negligible.

Here we go!

Game 0. We may describe the challenger in Game 0 as follows:



(a) encryption

(b) decryption

Figure 5.4: Encryption and decryption for CBC mode with $\ell = 3$

```
$b \xleftarrow{R} \{0,1\}$, $k \xleftarrow{R} \mathcal{K}$
upon receiving the $i$th query $(m_{i0}, m_{i1})$, with $v_i := |m_{i0}| = |m_{i1}|$:
    compute $c_i \in \mathcal{X}^{v_i+1}$ as follows:
        $c_i[0] \xleftarrow{R} \mathcal{X}$
        for $j \leftarrow 0$ to $v_i - 1$ do
            $x_{ij} \leftarrow c_i[j] \oplus m_{ib}[j]$
            $c_i[j+1] \leftarrow E(k, x_{ij})$
    send $c_i$ to the adversary.
```

By construction, we have

$$\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] = \big|\Pr[W_0] - 1/2\big|. \tag{5.24}$$

Game 1. We now play the "PRF card," replacing $E(k, \cdot)$ by a truly random function $f \in \text{Funs}[\mathcal{X}, \mathcal{X}]$. Our challenger in this game looks like this:

```
$b \xleftarrow{R} \{0,1\}$, $f \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{X}]$
upon receiving the $i$th query $(m_{i0}, m_{i1})$, with $v_i := |m_{i0}| = |m_{i1}|$:
    compute $c_i \in \mathcal{X}^{v_i+1}$ as follows:
        $c_i[0] \xleftarrow{R} \mathcal{X}$
        for $j \leftarrow 0$ to $v_i - 1$ do
            $x_{ij} \leftarrow c_i[j] \oplus m_{ib}[j]$
            $c_i[j+1] \leftarrow f(x_{ij})$
    send $c_i$ to the adversary.
```

We claim that

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \text{PRF}\mathsf{adv}[\mathcal{B}, E], \tag{5.25}$$

where $\mathcal{B}$ is an efficient adversary; moreover, since we are assuming that $\mathcal{E}$ is a secure block cipher, and that N is super-poly, it must be the case that $\text{PRF}\mathsf{adv}[\mathcal{B}, E]$ is negligible. This is hopefully (by now) a routine argument, and we leave the details of this to the reader.

Game 2. The next step in this dance should by now be familiar: we implement f using a faithful gnome. We do so by introducing random variables y_{ij} which represent the "default" values for $c_i[j]$, which get over-ridden if necessary in the line marked $(*)$ below:

```
$b \xleftarrow{R} \{0,1\}$
set $y_{ij} \xleftarrow{R} \mathcal{X}$ for $i = 1, \ldots, Q$ and $j = 0, \ldots, \ell$
upon receiving the $i$th query $(m_{i0}, m_{i1})$, with $v_i := |m_{i0}| = |m_{i1}|$:
    compute $c_i \in \mathcal{X}^{v_i+1}$ as follows:
        $c_i[0] \leftarrow y_{i0}$
        for $j \leftarrow 0$ to $v_i - 1$ do
            $x_{ij} \leftarrow c_i[j] \oplus m_{ib}[j]$
            $c_i[j+1] \leftarrow y_{i(j+1)}$
$(*)$       if $x_{ij} = x_{i'j'}$ for some $(i', j') < (i, j)$ then $c_i[j+1] \leftarrow c_{i'}[j'+1]$
    send $c_i$ to the adversary.
```

We clearly have

$$\Pr[W_2] = \Pr[W_1]. \tag{5.26}$$

Game 3. Now we make the gnome forgetful, removing the check in the line marked $(*)$:

```
$b \stackrel{\mathrm{R}}{\leftarrow} \{0,1\}$
set $y_{ij} \stackrel{\mathrm{R}}{\leftarrow} \mathcal{X}$ for $i = 1, \ldots, Q$ and $j = 0, \ldots, \ell$

upon receiving the $i$th query $(m_{i0}, m_{i1})$, with $v_i := |m_{i0}| = |m_{i1}|$:
    compute $c_i \in \mathcal{X}^{v_i+1}$ as follows:
        $c_i[0] \leftarrow y_{i0}$
        for $j \leftarrow 0$ to $v_i - 1$ do
            $x_{ij} \leftarrow c_i[j] \oplus m_{ib}[j]$
            $c_i[j+1] \leftarrow y_{i(j+1)}$
    send $c_i$ to the adversary.
```

To analyze the quantity $|\Pr[W_3] - \Pr[W_2]|$, we use the Difference Lemma (Theorem 4.7). To this end, we view Games 2 and 3 as operating on the same underlying probability space: the random choices made by the adversary and the challenger are identical in both games — all that differs is the rule used by the challenger to compute its responses.

We define Z to be the event that $x_{ij} = x_{i'j'}$ in Game 3. Note that the event Z is defined in terms of the x_{ij} values in Game 3. Indeed, the x_{ij} values may not be computed in the same way in Games 2 and 3, and so we have explicitly defined the event Z in terms of their values in Game 3. Nevertheless, it is clear that Games 2 and 3 proceed identically unless Z occurs; in particular, $W_2 \wedge \bar{Z}$ occurs if and only if $W_3 \wedge \bar{Z}$ occurs. Applying the Difference Lemma, we therefore have

$$\big|\Pr[W_3] - \Pr[W_2]\big| \leq \Pr[Z]. \tag{5.27}$$

We claim that

$$\Pr[Z] \leq \frac{Q^2\ell^2}{2N}. \tag{5.28}$$

To prove this, let *Coins* denote the random choices made by $\mathcal{A}$. Observe that in Game 3, the values

$$\mathit{Coins}, \quad b, \quad y_{ij} \ (i = 1, \ldots Q, \ j = 0, \ldots, \ell)$$

are independently distributed.

Consider any fixed index $i = 1, \ldots, Q$. Let us condition on any fixed values of *Coins*, b, and $y_{i'j}$ for $i' = 1, \ldots, i-1$ and $j = 0, \ldots, \ell$. In this conditional probability space, the values of m_{i0}, m_{i1}, and v_i are completely determined, as are the values $v_{i'}$ and $x_{i'j}$ for $i' = 1, \ldots, i-1$ and $j = 0, \ldots, v_{i'} - 1$; however, the values of $y_{i0}, \ldots, y_{i\ell}$ are still uniformly and independently distributed over $\mathcal{X}$. Moreover, as $x_{ij} = y_{ij} \oplus m_{ib}[j]$ for $j = 0, \ldots, v_i - 1$, it follows that these x_{ij} values are also uniformly and independently distributed over $\mathcal{X}$. Thus, for any fixed index $j = 0, \ldots, v_i - 1$, and any fixed indices i' and j', with $(i', j') < (i, j)$, the probability that $x_{ij} = x_{i'j'}$ in this conditional probability space is $1/N$. The bound (5.28) now follows from an easy calculation.

Finally, we claim that

$$\Pr[W_3] = 1/2. \tag{5.29}$$

This follows from the fact that

$$\mathit{Coins}, \quad b, \quad y_{ij} \ (i = 1, \ldots Q, \ j = 0, \ldots, \ell)$$

are independently distributed, and the fact that the adversary's output $\hat{b}$ is a function of

$$\mathit{Coins}, \quad y_{ij} \ (i = 1, \ldots Q, \ j = 0, \ldots, \ell).$$

From this, we see that $\hat{b}$ and b are independent, and so (5.29) follows immediately.

Putting together (5.24) through (5.29), we have

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}'] \le \frac{Q^2\ell^2}{2N} + \mathrm{PRF}\mathsf{adv}[\mathcal{B},E].$$

By Theorem 4.4, we have

$$\big|\mathrm{BC}\mathsf{adv}[\mathcal{B},\mathcal{E}] - \mathrm{PRF}\mathsf{adv}[\mathcal{B},E]\big| \le \frac{Q^2\ell^2}{2N},$$

and (5.23) follows, which proves the theorem. □

5.4.4 Case study: CBC padding in TLS 1.0

Let $\mathcal{E} = (E, D)$ be a block cipher with domain $\mathcal{X}$. Our description of CBC mode encryption using $\mathcal{E}$ assumes that messages to be encrypted are elements of $\mathcal{X}^{\le\ell}$. When the domain is $\mathcal{X} = \{0,1\}^{128}$, as in the case of AES, this implies that we can only encrypt messages whose length is a multiple of 16 bytes. But what if the message length is not a multiple of the block size?

Suppose we wish to encrypt a v-byte message m using AES in CBC mode when v is not necessarily a multiple of 16. The first thing that comes to mind is to pad the message m so that its length in bytes is a multiple of 16. Clearly the padding function must be invertible so that the padding can be removed during decryption.

The TLS 1.0 protocol defines the following padding function for encrypting a v-byte message with AES in CBC mode: let $p := 16 - (v \bmod 16)$, then append p bytes to the message m, where each byte has value $p - 1$. For example, consider the following two cases:

- if m is 29 bytes long then $p = 3$ and the pad consists of the three bytes "222" so that the padded message is 32 bytes long which is exactly two AES blocks.
- if the length of m is a multiple of the block size, say 32 bytes, then $p = 16$ and the pad consists of 16 bytes. The padded message is then 48 bytes long which is three AES blocks.

It may seem odd that when the message is a multiple of the block size we add a full dummy block at the end. This is necessary so that the decryption procedure can properly remove the pad. Indeed, it should be clear that this padding method is invertible for all input message lengths.

It is an easy fact to prove that every invertible padding scheme for CBC mode encryption built from a secure block cipher gives a CPA secure cipher for messages of arbitrary length.

Padding in CBC mode can be avoided using a method called **ciphertext stealing** as long as the plaintext is longer than a single block. The ciphertext stealing variant of CBC is the topic of Exercise 5.16. When encrypting messages whose length is less than a block, say single byte messages, there is still a need to pad.

5.4.5 Concrete parameters and a comparison of counter and CBC modes

We conclude this section with a comparison of the counter and CBC mode constructions. We assume that counter mode is implemented with a PRF F that maps n-bit blocks to n-bit blocks, and that CBC is implemented with an n-bit block cipher. In each case, the message space consists of sequences of at most ℓ n-bit data blocks. With the security theorems proved in this section, we have the following bounds:

$$\begin{aligned} \text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{ctr}}] &\leq \frac{4Q^2\ell}{2^n} + 2 \cdot \text{PRF}\mathsf{adv}[\mathcal{B}_F, F], \\ \text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{cbc}}] &\leq \frac{2Q^2\ell^2}{2^n} + 2 \cdot \text{BC}\mathsf{adv}[\mathcal{B}_{\mathcal{E}}, \mathcal{E}]. \end{aligned}$$

Here, $\mathcal{A}$ is any CPA adversary making at most Q queries to its challenger, ℓ is the maximum length (in data blocks) of any one message. For the purposes of this discussion, let us simply ignore the terms $\text{PRF}\mathsf{adv}[\mathcal{B}_F, F]$ and $\text{BC}\mathsf{adv}[\mathcal{B}_{\mathcal{E}}, \mathcal{E}]$.

One can immediately see that counter mode has a quantitative security advantage. To make things more concrete, suppose the block size is $n = 128$, and that each message is 1MB (2^{23} bits) so that $\ell = 2^{16}$ blocks. If we want to keep the adversary's advantage below 2^{-32}, then for counter mode, we can encrypt up to $Q = 2^{39.5}$ messages, while for CBC we can encrypt only up to 2^{32} messages. Once Q messages are encrypted with a given key, a fresh key must be generated and used for subsequent messages. Therefore, with counter mode a single key can be used to securely encrypt many more messages than with CBC.

We should warn that this quantitative advantage disappears if we use a block cipher to implement F, as we get the same quadratic dependence on ℓ due to the error term in the PRF switching lemma (Theorem 4.4). However, for a custom built PRF, this quantitative advantage applies.

Counter mode has several other advantages over CBC:

- *Parallelism and pipelining.* Encryption and decryption for counter mode is trivial to parallelize, whereas encryption in CBC mode is inherently sequential (decryption in CBC mode is parallelizable). Modes that support parallelism greatly improve performance when the underlying hardware can execute many instructions in parallel as is often the case in modern processors. More importantly, consider a hardware implementation of a single block cipher round that supports pipelining, as in Intel's implementation of AES-128 (page 121). Pipelining enables multiple encryption instructions to execute at the same time. A parallel mode such as counter mode keeps the pipeline busy, whereas in CBC encryption the pipeline is mostly unused due to the sequential nature of this mode. As a result, counter mode encryption on Intel's Haswell processors is about seven times faster than CBC mode encryption, assuming the plaintext data is already loaded into L1 cache.

- *Shorter ciphertext length.* For very short messages, counter mode ciphertexts are significantly shorter than CBC mode ciphertexts. Consider, for example, a one-byte plaintext (which arises naturally when encrypting individual key strokes as in SSH). A counter mode ciphertext need only be one block plus one byte: one block for the random IV plus one byte for the encrypted plaintext. In contrast, a CBC ciphertext is two full blocks. This results in 15 redundant bytes per CBC ciphertext assuming 128-bit blocks.

- *Encryption only.* CBC mode uses both algorithms E and D of the block cipher whereas counter mode uses only algorithm E. This can reduce an implementation code size.

Remark 5.4. Both randomized counter mode and CBC require a random IV. Some crypto libraries actually leave it to the higher-level application to supply the IV. This can lead to problems if the higher-level applications do not take pains to ensure the IVs are sufficiently random. For example, for counter mode, it is necessary that the IVs are sufficiently spread out, so that the corresponding intervals do not overlap. In fact, this property is sufficient as well. In contrast, for CBC mode, more is required: it is essential that IVs be unpredictable — see Exercise 5.13.

Leaving it to the higher-level application to supply the IV is actually an example of *nonce-based encryption*, which we will explore in detail next, in Section 5.5. □

5.5 Nonce-based encryption

All of the CPA-secure encryption schemes we have seen so far suffer from *ciphertext expansion*: ciphertexts are longer than plaintexts. For example, the generic hybrid construction in Section 5.4.1 generates ciphertexts (x, c), where x belongs to the input space of some PRF and c encrypts the actual message; the counter mode construction in Section 5.4.2 generates ciphertexts of the essentially same form (x, c); similarly, the CBC mode construction in Section 5.4.3 includes the IV as a part of the ciphertext.

For very long messages, the expansion is not too bad. For example, with AES and counter mode or CBC mode, a 1MB message results is a ciphertext that is just 16 bytes longer, which may be a perfectly acceptable expansion rate. However, for messages of 16 bytes or less, ciphertexts are at least twice as long as plaintexts.

The bad news is, some amount of ciphertext expansion is inevitable for any CPA-secure encryption scheme (see Exercise 5.10). The good news is, in certain settings, one can get by without any ciphertext expansion. For example, suppose Alice and Bob are fully synchronized, so that Alice first sends an encryption of m_1, then an encryption of m_2, and so on, while Bob first decrypts the encryption of m_1, then decrypts the encryption of m_2, and so on. For concreteness, assume Alice and Bob are using the generic hybrid construction of Section 5.4.1. Recall that the encryption of message m_i is (x_i, c_i), where $c_i := E(k_i, m_i)$ and $k_i := F(k', x_i)$. The essential property of the x_i's needed to ensure security was simply that they are distinct. When Alice and Bob are fully synchronized (i.e., ciphertexts sent by Alice reach Bob in-order), they simply have to agree on a fixed sequence $x_1, x_2, \ldots,$ of distinct elements in the input space of the PRF F. For example, x_i might simply be the binary encoding of i.

This mode of operation of an encryption scheme does not really fit into our definitional framework. Historically, there are two ways to modify the framework to allow for this type of operation. One approach is to allow for *stateful* encryption schemes, where both the encryption and decryption algorithms maintain some internal state that evolves with each application of the algorithm. In the example of the previous paragraph, the state would just consist of a counter that is incremented with each application of the algorithm. This approach requires encryptor and decryptor to be fully synchronized, which limits its applicability, and we shall not discuss it further.

The second, and more popular, approach is called *nonce-based encryption*. Instead of maintaining internal states, both the encryption and decryption algorithms take an additional input $\mathscr{N}$, called a *nonce*. The syntax for nonce-based encryption becomes

$$c = E(k, m, \mathscr{N}),$$

where $c \in \mathcal{C}$ is the ciphertext, $k \in \mathcal{K}$ is the key, $m \in \mathcal{M}$ is the message, and $\mathscr{N} \in \mathcal{N}$ is the nonce. Moreover, the encryption algorithm E is required to be deterministic. Likewise, the decryption syntax becomes

$$m = D(k, c, \mathcal{N}).$$

The intention is that a message encrypted with a particular nonce should be decrypted with the same nonce — it is up to the application using the encryption scheme to enforce this. More formally, the correctness requirement is that

$$D(k,\ E(k, m, \mathcal{N}),\ \mathcal{N}) = m$$

for all $k \in \mathcal{K}$, $m \in \mathcal{M}$, and $\mathcal{N} \in \mathcal{N}$. We say that such a nonce-based cipher $\mathcal{E} = (E, D)$ is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C}, \mathcal{N})$.

Intuitively, a nonce-based encryption scheme is CPA secure if it does not leak any useful information to an eavesdropper, assuming that *no nonce is used more than once* in the encryption process — again, it is up to the application using the scheme to enforce this. Note that this requirement on how nonces are used is very weak, much weaker than requiring that they are unpredictable, let alone randomly chosen. Of course, since nonces are not meant to be secret, their value must not reveal anything about the plaintext.

We can readily formalize this notion of security by slightly tweaking our original definition of CPA security.

Attack Game 5.3 (nonce-based CPA security). For a given cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C}, \mathcal{N})$, and for a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define

Experiment b:

- The challenger selects $k \xleftarrow{\text{R}} \mathcal{K}$.
- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots,$ the ith query is a pair of messages, $m_{i0}, m_{i1} \in \mathcal{M}$, of the same length, and a nonce $\mathcal{N}_i \in \mathcal{N} \setminus \{\mathcal{N}_1, \ldots, \mathcal{N}_{i-1}\}$.

  The challenger computes $c_i \leftarrow E(k, m_{ib}, \mathcal{N}_i)$, and sends c_i to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{nCPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - \Pr[W_1]|. \quad \Box$$

Note that in the above game, the nonces are completely under the adversary's control, subject only to the constraint that they are unique.

Definition 5.3 (nonce-based CPA security). *A nonce-based cipher $\mathcal{E}$ is called* ***semantically secure against chosen plaintext attack****, or simply* ***CPA secure****, if for all efficient adversaries $\mathcal{A}$, the value* nCPA$\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

As usual, as in Section 2.2.5, Attack Game 5.3 can be recast as a "bit guessing" game, and we have

$$\text{nCPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \text{nCPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}], \tag{5.30}$$

where nCPA$\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] := |\Pr[\hat{b} = b] - 1/2|$ in a version of Attack Game 5.3 where the challenger just chooses b at random.

5.5.1 Nonce-based generic hybrid encryption

Let us recast the generic hybrid construction in Section 5.4.1 as a nonce-based encryption scheme. As in that section, $\mathcal{E}$ is a cipher, which we shall now insist is deterministic, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and F is a PRF defined over $(\mathcal{K}', \mathcal{X}, \mathcal{K})$. We define the nonce-based cipher $\mathcal{E}'$, which is defined over $(\mathcal{K}', \mathcal{M}, \mathcal{C}, \mathcal{X})$, as follows:

- for $k' \in \mathcal{K}'$, $m \in \mathcal{M}$, and $x \in \mathcal{X}$, we define $E'(k', m, x) := E(k, m)$, where $k := F(k', x)$;
- for $k' \in \mathcal{K}'$, $c \in \mathcal{C}$, $x \in \mathcal{X}$, we define $D'(k', c, x) := D(k, c)$, where $k := F(k', x)$.

All we have done is to treat the value $x \in \mathcal{X}$ as a nonce; otherwise, the scheme is exactly the same as that defined in Section 5.4.1.

One can easily verify the correctness requirement for $\mathcal{E}'$. Moreover, one can easily adapt the proof of Theorem 5.2 to prove the following:

Theorem 5.5. *If F is a secure PRF and $\mathcal{E}$ is a semantically secure cipher, then the cipher $\mathcal{E}'$ described above is a CPA secure cipher.*

In particular, for every nCPA adversary $\mathcal{A}$ that attacks $\mathcal{E}'$ as in the bit-guessing version of Attack Game 5.3, and which makes at most Q queries to its challenger, there exists a PRF adversary $\mathcal{B}_F$ that attacks F as in Attack Game 4.2, and an SS adversary $\mathcal{B}_{\mathcal{E}}$ that attacks $\mathcal{E}$ as in the bit-guessing version of Attack Game 2.1, where both $\mathcal{B}_F$ and $\mathcal{B}_{\mathcal{E}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\text{nCPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}'] \leq 2 \cdot \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + Q \cdot \text{SS}\mathsf{adv}[\mathcal{B}_{\mathcal{E}}, \mathcal{E}]. \tag{5.31}$$

We leave the proof as an exercise for the reader. Note that the term $\frac{Q^2}{N}$ in (5.5), which represent the probability of a collision on the input to F, is missing from (5.31), simply because by definition, no collisions can occur.

5.5.2 Nonce-based Counter mode

Next, we recast the counter-mode cipher from Section 5.4.2 to the nonce-based encryption setting. Let us make a first attempt, by simply treating the value $x \in \mathcal{X}$ in that construction as a nonce.

Unfortunately, this scheme cannot satisfy the definition of nonce-based CPA security. The problem is, an attacker could choose two distinct nonces $x_1, x_2 \in \mathcal{X}$, such that the intervals $\{x_1, \ldots, x_1 + \ell - 1\}$ and $\{x_2, \ldots, x_2 + \ell - 1\}$ overlap (again, arithmetic is done mod N). In this case, the security proof will break down; indeed, it is easy to mount a quite devastating attack, as discussed in Section 5.1, since that attacker can essentially force the encryptor to re-use some of the same bits of the "key stream".

Fortunately, the fix is easy. Let us assume that ℓ divides N (in practice, both ℓ and N will be powers of 2, so this is not an issue). Then we use as the nonce space $\{0, \ldots, N/\ell - 1\}$, and translate the nonce $\mathcal{N}$ to the PRF input $x := \mathcal{N}\ell$. It is easy to see that for any two distinct nonces $\mathcal{N}_1$ and $\mathcal{N}_2$, for $x_1 := \mathcal{N}_1\ell$ and $x_2 := \mathcal{N}_2\ell$, the intervals $\{x_1, \ldots, x_1 + \ell - 1\}$ and $\{x_2, \ldots, x_2 + \ell - 1\}$ do not overlap.

With $\mathcal{E}$ modified in this way, we can easily adapt the proof of Theorem 5.3 to prove the following:

Theorem 5.6. *If F is a secure PRF, then the nonce-based cipher $\mathcal{E}$ described above is CPA secure.*

In particular, for every nCPA adversary $\mathcal{A}$ that attacks $\mathcal{E}$ as in Attack Game 5.3, there exists a PRF adversary $\mathcal{B}$ that attacks F as in Attack Game 4.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{nCPA}\mathsf{adv}[\mathcal{A},\mathcal{E}] \leq 2 \cdot \text{PRF}\mathsf{adv}[\mathcal{B},F]. \tag{5.32}$$

We again leave the proof as an exercise for the reader.

5.5.3 Nonce-based CBC mode

Finally, we consider how to recast the CBC-mode encryption scheme in Section 5.4.3 as a nonce-based encryption scheme. As a first attempt, one might simply try to view the IV $c[0]$ as a nonce. Unfortunately, this does not yield a CPA secure nonce-based encryption scheme. In the nCPA attack game, the adversary could make two queries:

$(m_{10}, m_{11}, \mathcal{N}_1),$
$(m_{20}, m_{21}, \mathcal{N}_2),$

where

$$m_{10} = \mathcal{N}_1 \neq \mathcal{N}_2 = m_{20},\ m_{11} = m_{21}.$$

Here, all messages are one-block messages. In Experiment 0 of the attack game, the resulting ciphertexts will be the same, whereas in Experiment 1, they will be different. Thus, we can perfectly distinguish between the two experiments.

Again, the fix is fairly straightforward. The idea is to map nonces to pseudo-random IV's by passing them through a PRF. So let us assume that we have a PRF F defined over $(\mathcal{K}', \mathcal{N}, \mathcal{X})$. Here, the key space $\mathcal{K}'$ and input space $\mathcal{N}$ of F may be arbitrary sets, but the output space $\mathcal{X}$ of F must match the block space of the underlying block cipher $\mathcal{E} = (E, D)$, which is defined over $(\mathcal{K}, \mathcal{X})$. In the nonce-based CBC scheme $\mathcal{E}'$, the key space is $\mathcal{K} \times \mathcal{K}'$, and in the encryption and decryption algorithms, the IV is computed from the nonce $\mathcal{N}$ and key k' as $c[0] := F(k', \mathcal{N})$.

With these modifications, we can now prove the following variant of Theorem 5.4:

Theorem 5.7. *If $\mathcal{E} = (E, D)$ is a secure block cipher defined over $(\mathcal{K}, \mathcal{X})$, and $N := |\mathcal{X}|$ is super-poly, and F is a secure PRF defined over $(\mathcal{K}', \mathcal{N}, \mathcal{X})$, then for any poly-bounded $\ell \geq 1$, the nonce-based cipher $\mathcal{E}'$ described above is CPA secure.*

In particular, for every nCPA adversary $\mathcal{A}$ that attacks $\mathcal{E}'$ as in the bit-guessing version of Attack Game 5.3, and which makes at most Q queries to its challenger, there exists BC adversary $\mathcal{B}$ that attacks $\mathcal{E}$ as in Attack Game 4.1, and a PRF adversary $\mathcal{B}_F$ that attacks F as in Attack Game 4.2, where $\mathcal{B}$ and $\mathcal{B}_F$ are elementary wrappers around $\mathcal{A}$, such that

$$\text{nCPA}\mathsf{adv}[\mathcal{A},\mathcal{E}'] \leq \frac{2Q^2\ell^2}{N} + 2 \cdot \text{PRF}\mathsf{adv}[\mathcal{B}_F,F] + 2 \cdot \text{BC}\mathsf{adv}[\mathcal{B},\mathcal{E}]. \tag{5.33}$$

Again, we leave the proof as an exercise for the reader. Note that in the above construction, we may use the underlying block cipher $\mathcal{E}$ for the PRF F; however, it is essential that independent keys k and k' are used (see Exercise 5.14).

Nonce-based CBC mode as described above is called `CBC-ESSIV` (encrypted salt-sector IV) and is used in full disk encryption systems, such as `dm-crypt`. When encrypting a sector on disk, the sector number is used as the nonce. Other full disk encryption systems use tweakable encryption, as discussed in Exercise 4.11).

5.6 A fun application: revocable broadcast encryption

Movie studios spend a lot of effort making blockbuster movies, and then sell the movies (on DVDs) to millions of customers who purchase them to watch at home. A customer should be able to watch movies on a stateless standalone movie player, that has no network connection.

The studios are worried about piracy, and do not want to send copyrighted digital content in the clear to millions of users. A simple solution could work as follows. Every authorized manufacturer is given a **device key** $k_{\mathrm{d}} \in \mathcal{K}$, and it embeds this key in every device that it sells. If there are a hundred authorized device manufacturers, then there are a hundred device keys $k_{\mathrm{d}}^{(1)}, \ldots, k_{\mathrm{d}}^{(100)}$. A movie m is encrypted as:

$$c_{\mathrm{m}} := \left\{ \begin{array}{l} k \xleftarrow{\mathrm{R}} \mathcal{K} \\ \text{for } i = 1, \ldots, 100 : \ c_i \xleftarrow{\mathrm{R}} E(k_{\mathrm{d}}^{(i)},\ k) \\ c \xleftarrow{\mathrm{R}} E'(k, m) \\ \text{output } (c_1, \ldots, c_{100},\ c) \end{array} \right\}$$

where (E, D) is a CPA secure cipher, and (E', D') is semantically secure with key space $\mathcal{K}$. We analyze this construction in Exercise 5.4, where we show that it is CPA secure. We refer to $(c_1, \ldots, c_{100})$ as the ciphertext header, and refer to c as the body.

Now, every authorized device can decrypt the movie using its embedded device key. First, decrypt the appropriate ciphertext in the header, and then use the obtained key k to decrypt the body. This mechanism forms the basis of the **content scrambling system** (CSS) used to encrypt DVDs. We previously encountered CSS in Section 3.8.

The trouble with this scheme is that once a single device is compromised, and its device key k_{d} is extracted and published, then anyone can use this k_{d} to decrypt every movie ever published. There is no way to revoke k_{d} without breaking many consumer devices in the field. In fact, this is exactly how CSS was broken: the device key was extracted from an authorized player, and then used in a system called **DeCSS** to decrypt encrypted DVDs.

The lesson from CSS is that global unrevocable device keys are a bad idea. Once a single key is leaked, all security is lost. When the DVD format was updated to a new format called Blu-ray, the industry got a second chance to design the encryption scheme. In the new scheme, called the **Advanced Access Content System** (AACS), every device gets a random device key unique to that device. The system is designed to support billions of devices, each with its own key.

The goals of the system are twofold. First, every authorized device should be able to decrypt every Blu-ray disk. Second, whenever a device key is extracted and published, it should be possible to revoke that key, so that this device key cannot be used to decrypt future Blu-ray disks, but without impacting any other devices in the field.

A revocable broadcast system. Suppose there are n devices in the system, where for simplicity, let us assume n is a power of two. We treat these n devices as the leaves of a complete binary tree, as shown in Fig. 5.5. Every node in the tree is assigned a random key in the key space $\mathcal{K}$. The keys embedded in device number $i \in \{1, \ldots, n\}$ is the set of keys on the path from leaf number i to the root. This way, every device is given exactly $\log_2 n$ keys in $\mathcal{K}$.

When the system is first launched, and no device keys are yet revoked, all content is encrypted using the key at the root (key number 15 in Fig. 5.5). More precisely, we encrypt a movie m as:

$$c_{\mathrm{m}} := \left\{\ k \xleftarrow{\mathrm{R}} \mathcal{K},\ c_1 \xleftarrow{\mathrm{R}} E(k_{\mathrm{root}},\ k),\ c \xleftarrow{\mathrm{R}} E'(k, m),\ \text{output } (c_1, c)\ \right\}$$

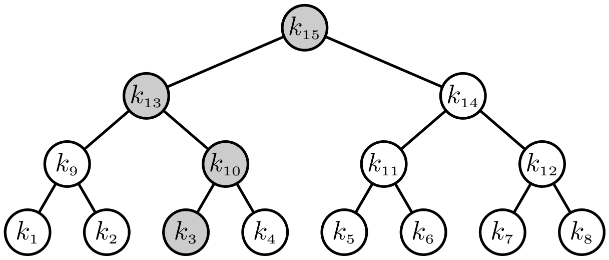


Figure 5.5: The tree of keys for $n = 8$ devices; shaded nodes are the keys embedded in device 3.

Because all devices have the root key k_{root}, all devices can decrypt.

Revoking devices. Now, suppose device number i is attacked, and all the keys stored on it are published. Then all future content will be encrypted using the keys associated with the siblings of the $\log_2 n$ nodes on the path from leaf i to the root. For example, when device number 3 in Fig. 5.5 is revoked, all future content is encrypted using the three keys k_4, k_9, k_{14} as

$$c_{\text{m}} := \left\{ \begin{array}{l} k \xleftarrow{\text{R}} \mathcal{K} \\ c_1 \xleftarrow{\text{R}} E(k_4,\ k), \quad c_2 \xleftarrow{\text{R}} E(k_9,\ k), \quad c_3 \xleftarrow{\text{R}} E(k_{14},\ k) \\ c \xleftarrow{\text{R}} E'(k, m) \\ \text{output } (c_1, c_2, c_3, c) \end{array} \right\} \tag{5.34}$$

Again, (c_1, c_2, c_3) is the ciphertext header, and c is the ciphertext body. Observe that device number 3 cannot decrypt c_{m}, because it cannot decrypt any of the ciphertexts in the header. However, every other device can easily decrypt using one of the keys at its disposal. For example device number 6 can use k_{14} to decrypt c_3. In effect, changing the encryption scheme to encrypt as in (5.34) revokes device number 3, without impacting any other device. The cost to this is that the ciphertext header now contains $\log_2 n$ blocks, as opposed to a single block before the device was revoked.

More generally, suppose r devices have been compromised and need to be revoked. Let $S \subseteq \{1, \ldots, n\}$ be the set of non-compromised devices, so that that $|S| = n - r$. New content will be encrypted using keys in the tree so that devices in S can decrypt, but all devices outside of S cannot. The set of keys that makes this possible is characterized by the following definition:

Definition 5.4. *Let T be a complete binary tree with n leaves, where n is a power of two. Let $S \subseteq \{1, \ldots, n\}$ be a set of leaves. We say that a set of nodes $W \subseteq \{1, \ldots, 2n-1\}$ **covers** the set S if every leaf in S is a descendant of some node in W, and leaves outside of S are not. We use* cover(S) *to denote the smallest set of nodes that covers S.*

Fig. 5.6 gives an example of a cover of the set of leaves $\{1, 2, 4, 5, 6\}$. The figure captures a setting where devices number 3, 7, and 8 are revoked. It should be clear that if we use keys in cover(S) to encrypt a movie m, then devices in S can decrypt, but devices outside of S cannot. In particular, we encrypt m as follows:

$$c_{\mathrm{m}} := \left\{ \begin{array}{l} k \overset{\mathrm{R}}{\leftarrow} \mathcal{K} \\ \text{for } u \in \mathrm{cover}(S) : c_u \overset{\mathrm{R}}{\leftarrow} E(k_u,\ k) \\ c \overset{\mathrm{R}}{\leftarrow} E'(k, m) \\ \text{output } (\{c_u\}_{u \in \mathrm{cover}(S)},\ c) \end{array} \right\}. \tag{5.35}$$

Security of this scheme is discussed in Exercise 5.21.

The more devices are revoked, the larger the header of c_{m} becomes. The following theorem shows how big the header gets in the worst case. The proof is an induction argument that also suggests an efficient recursive algorithm to compute an optimal cover.

Theorem 5.8. *Let T be a complete binary tree with n leaves, where n is a power of two. For every $1 \leq r \leq n$, and every set S of $n - r$ leaves, we have*

$$|\mathrm{cover}(S)| \leq r \cdot \log_2(n/r)$$

Proof. We prove the theorem by induction on $\log_2 n$. For $n = 1$ the theorem is trivial. Now, assume the theorem holds for a tree with $n/2$ leaves, and let us prove it for a tree T with n leaves. The tree T is made up of a root node, and two disjoint sub-trees, T_1 and T_2, each with $n/2$ leaves. Let us split the set $S \subseteq \{1, \ldots, n\}$ in two: $S = S_1 \cup S_2$, where S_1 is contained in $\{1, \ldots, n/2\}$, and S_2 is contained in $\{n/2 + 1, \ldots, n\}$. That is, S_1 are the elements of S that are leaves in T_1, and S_2 are the elements of S that are leaves in T_2. Let $r_1 := (n/2) - |S_1|$ and $r_2 := (n/2) - |S_2|$. Then clearly $r = r_1 + r_2$.

First, suppose both r_1 and r_2 are greater than zero. By the induction hypothesis, we know that for $i = 1, 2$ we have $|\mathrm{cover}(S_i)| \leq r_i \log_2(n/2r_i)$. Therefore,

$$\begin{aligned} |\mathrm{cover}(S)| = |\mathrm{cover}(S_1)| + |\mathrm{cover}(S_2)| &\leq r_1 \log_2(n/2r_1) + r_2 \log_2(n/2r_2) \\ &= r \log_2(n/r) + \big(r \log_2 r - r_1 \log_2(2r_1) - r_2 \log_2(2r_2)\big) \leq r \log_2(n/r), \end{aligned}$$

which is what we had to prove in the induction step. The last inequality follows from a simple fact about logarithms, namely that for all numbers $r_1 \geq 1$ and $r_2 \geq 1$, we have

$$(r_1 + r_2) \log_2(r_1 + r_2) \leq r_1 \log_2(2r_1) + r_2 \log_2(2r_2).$$

Second, if $r_1 = 0$ then $r_2 = r \geq 1$. The induction step now follows from:

$$|\mathrm{cover}(S)| = |\mathrm{cover}(S_2)| \leq r \log_2(n/2r) = r \log_2(n/r) - r \leq r \log_2(n/r),$$

as required. The case $r_2 = 0$ follows similarly. This completes the induction step, and the proof. □

Theorem 5.8 shows that r devices can be revoked at the cost of increasing the ciphertext header size to $r \log_2(n/r)$ blocks. For moderate values of r this is not too big. Nevertheless, this general approach can be improved [121, 83, 77]. The best system using this approach embeds $O(\log n)$ keys in every device, same as here, but the header size is only $O(r)$ blocks. The AACS system uses the subset-tree difference method [121], which has a worst case header of size $2r - 1$ blocks, but stores $\frac{1}{2} \log^2 n$ keys per device.

While AACS is a far better designed than CSS, it too has been attacked. In particular, the process of a revoking an AACS key is fairly involved and can take several months. Hackers showed that they can extract new device keys from unrevoked players faster than the industry can revoke them.

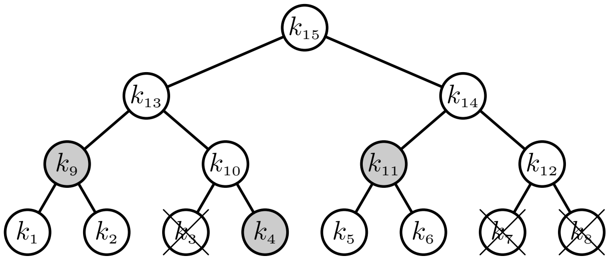


Figure 5.6: The three shaded nodes are the minimal cover for leaves $\{1, 2, 4, 5, 6\}$.

5.7 Notes

Citations to the literature to be added.

5.8 Exercises

5.1 (Double encryption). Let $\mathcal{E} = (E, D)$ be a cipher. Consider the cipher $\mathcal{E}_2 = (E_2, D_2)$, where $E_2(k, m) = E(k, E(k, m))$. One would expect that if encrypting a message once with E is secure then encrypting it twice as in E_2 should be no less secure. However, that is not always true.

(a) Show that there is a semantically secure cipher $\mathcal{E}$ such that $\mathcal{E}_2$ is not semantically secure.

(b) Prove that for every CPA secure ciphers $\mathcal{E}$, the cipher $\mathcal{E}_2$ is also CPA secure. That is, show that for every CPA adversary $\mathcal{A}$ attacking $\mathcal{E}_2$ there is a CPA adversary $\mathcal{B}$ attacking $\mathcal{E}$ with about the same advantage and running time.

5.2 (Multi-key CPA security). Generalize the definition of CPA security to the multi-key setting, analogous to Definition 5.1. In this attack game, the adversary gets to obtain encryptions of many messages under many keys. The game begins with the adversary outputting a number Q indicating the number of keys it wants to attack. The challenger chooses Q random keys. In every subsequent encryption query, the adversary submits a pair of messages and specifies under which of the Q keys it wants to encrypt; the challenger responds with an encryption of either the first or second message under the specified key (depending on whether the challenger is running Experiment 0 or 1). Flesh out all the details of this attack game, and prove, using a hybrid argument, that (single-key) CPA security implies multi-key CPA security. You should show that security degrades linearly in Q. That is, the advantage of any adversary $\mathcal{A}$ in breaking the multi-key CPA security of a scheme is at most $Q \cdot \epsilon$, where ϵ is the advantage of an adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) in attacking the scheme's (single-key) CPA security.

5.3 (An alternate definition of CPA security). This exercise develops an alternative characterization of CPA security for a cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. As usual, we need to define an attack game between an adversary $\mathcal{A}$ and a challenger. Initially, the challenger generates

$$b \xleftarrow{\text{R}} \{0, 1\}, \; k \xleftarrow{\text{R}} \mathcal{K}.$$

Then $\mathcal{A}$ makes a series of queries to the challenger. There are two types of queries:

Encryption: In an *encryption query*, $\mathcal{A}$ submits a message $m \in \mathcal{M}$ to the challenger, who responds with a ciphertext $c \xleftarrow{\text{R}} E(k, m)$. The adversary may make any (poly-bounded) number of encryption queries.

Test: In a *test query*, $\mathcal{A}$ submits a pair of messages $m_0, m_1 \in \mathcal{M}$ to the challenger, who responds with a ciphertext $c \xleftarrow{\text{R}} E(k, m_b)$. The adversary is allowed to make only a *single* test query (with any number of encryption queries before and after the test query).

At the end of the game, $\mathcal{A}$ outputs a bit $\hat{b} \in \{0, 1\}$.

As usual, we define $\mathcal{A}$'s advantage in the above attack game to be $|\Pr[\hat{b} = b] - 1/2|$. We say that $\mathcal{E}$ is Alt-CPA secure if this advantage is negligible for all efficient adversaries.

Show that $\mathcal{E}$ is CPA secure if and only if $\mathcal{E}$ is Alt-CPA secure.

5.4 (Hybrid CPA construction). Let (E_0, D_0) be a semantically secure cipher defined over $(\mathcal{K}_0, \mathcal{M}, \mathcal{C}_0)$, and let (E_1, D_1) be a CPA secure cipher defined over $(\mathcal{K}, \mathcal{K}_0, \mathcal{C}_1)$.

(a) Define the following hybrid cipher (E, D) as:

$$E(k, m) := \big\{k_0 \xleftarrow{\text{R}} \mathcal{K}_0,\ c_1 \xleftarrow{\text{R}} E_1(k, k_0),\ c_0 \xleftarrow{\text{R}} E_0(k_0, m),\ \text{output } (c_1, c_0)\big\}$$
$$D\big(k,\ (c_1, c_0)\big) := \big\{k_0 \leftarrow D_1(k, c_1),\ m \leftarrow D_0(k_0, c_0),\ \text{output } m\big\}$$

Here c_1 is called the ciphertext header, and c_0 is called the ciphertext body. Prove that (E, D) is CPA secure.

(b) Suppose m is some large copyrighted content. A nice feature of (E, D) is that the content owner can make the long ciphertext body c_0 public for anyone to download at their leisure. Suppose both Alice and Bob take the time to download c_0. When later Alice, who has key k_{a}, pays for access to the content, the content owner can quickly grant her access by sending her the short ciphertext header $c_{\text{a}} \xleftarrow{\text{R}} E_1(k_{\text{a}}, k_0)$. Similarly, when Bob, who has key k_{b}, pays for access, the content owner grants him access by sending him the short header $c_{\text{b}} \xleftarrow{\text{R}} E_1(k_{\text{b}}, k_0)$. Now, an eavesdropper gets to see

$$E'\big((k_{\text{a}}, k_{\text{b}}),\ m\big) := (c_{\text{a}}, c_{\text{b}}, c_0)$$

Generalize your proof from part (a) to show that this cipher is also CPA secure.

5.5 (A simple proof of randomized counter mode security). As mentioned in Remark 5.3, we can view randomized counter mode as a special case of the generic hybrid construction in Section 5.4.1. To this end, let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where $\mathcal{X} = \{0, \ldots, N - 1\}$ and $\mathcal{Y} = \{0, 1\}^n$, where N is super-poly. For poly-bounded $\ell \geq 1$, consider the PRF F' defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y}^\ell)$ as follows:

$$F'(k, x) := \Big(\ F(k, x),\ F(k, x + 1 \bmod N),\ \ldots,\ F(k, x + \ell - 1 \bmod N)\ \Big).$$

(a) Show that F' is a weakly secure PRF, as in Definition 4.3.

(b) Using part (a) and Remark 5.2, give a short proof that randomized counter mode is CPA secure.

5.6 (CPA security from a block cipher). Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{M} \times \mathcal{R})$. Consider the cipher $\mathcal{E}' = (E', D')$, where

$$E'(k, m) := \big\{ r \xleftarrow{\text{R}} \mathcal{R},\ c \xleftarrow{\text{R}} E\big(k,\ (m, r)\big),\ \text{output } c \big\}$$

$$D'(k, c) := \big\{ (m, r') \leftarrow D(k, c),\ \text{output } m \big\}$$

This cipher is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{M} \times \mathcal{R})$. Show that if $\mathcal{E}$ is a secure block cipher, and $1/|\mathcal{R}|$ is negligible, then $\mathcal{E}'$ is CPA secure.

5.7 (pseudo-random ciphertext security). In Exercise 3.4, we developed a notion of security called pseudo-random ciphertext security. This notion naturally extends to multiple ciphertexts. For a cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, we define two experiments: in Experiment 0 the challenger first picks a random key $k \xleftarrow{\text{R}} \mathcal{K}$ and then the adversary submits a sequence of queries, where the ith query is a message $m_i \in \mathcal{M}$, to which the challenger responds with $E(k, m_i)$. Experiment 1 is the same as Experiment 0 except that the challenger responds to the adversary's queries with random, independent elements of $\mathcal{C}$. We say that $\mathcal{E}$ is pseudo-random multi-ciphertext secure if no efficient adversary can distinguish between these two experiments with a non-negligible advantage.

(a) Consider the counter-mode construction in Section 5.4.2, based on a PRF F defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, but with a fixed-length plaintext space $\mathcal{Y}^{\ell}$ and a corresponding fixed-length ciphertext space $\mathcal{X} \times \mathcal{Y}^{\ell}$. Under the assumptions that F is a secure PRF, $|\mathcal{X}|$ is super-poly, and ℓ is poly-bounded, show that this cipher is pseudo-random multi-ciphertext secure.

(b) Consider the CBC construction Section 5.4.3, based on a block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{X})$, but with a fixed-length plaintext space $\mathcal{X}^{\ell}$ and corresponding fixed-length ciphertext space $\mathcal{X}^{\ell+1}$. Under the assumptions that $\mathcal{E}$ is a secure block cipher, $|\mathcal{X}|$ is super-poly, and ℓ is poly-bounded, show that this cipher is pseudo-random multi-ciphertext secure.

(c) Show that a pseudo-random multi-ciphertext secure cipher is also CPA secure.

(d) Give an example of a CPA secure cipher that is not pseudo-random multi-ciphertext secure.

5.8 (Deterministic CPA and SIV). We have seen that any cipher that is CPA secure must be probabilistic, since for a deterministic cipher, an adversary can always see if the same message is encrypted twice. We may define a relaxed notion of CPA security that says that this is the *only* thing the adversary can see. This is easily done by placing the following restriction on the adversary in Attack Game 5.2: for all indices i, j, we insist that $m_{i0} = m_{j0}$ if and only if $m_{i1} = m_{j1}$. We say that a cipher is **deterministic CPA secure** if every efficient adversary has negligible advantage in this restricted CPA attack game. In this exercise, we develop a general approach for building deterministic ciphers that are deterministic CPA secure.

Let $\mathcal{E} = (E, D)$ be a CPA-secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. We let $E(k, m; r)$ denote running algorithm $E(k, m)$ with randomness $r \xleftarrow{\text{R}} \mathcal{R}$ (for example, if $\mathcal{E}$ implements counter mode or CBC encryption then r is the random IV used by algorithm E). Let F be a secure PRF defined over $(\mathcal{K}', \mathcal{M}, \mathcal{R})$. Define the deterministic cipher $\mathcal{E}' = (E', D')$, defined over $(\mathcal{K} \times \mathcal{K}', \mathcal{M}, \mathcal{C})$ as follows:

$$\begin{aligned} E'\big((k, k'),\ m\big) &:= E(k, m;\ F(k', m)), \\ D'\big((k, k'),\ c\big) &:= D(k, c)\ . \end{aligned}$$

Show that $\mathcal{E}'$ is deterministic CPA secure. This construction is known as the **Synthetic IV** (or **SIV**) construction.

5.9 (Generic nonce-based encryption and nonce re-use resilience). In the previous exercise, we saw how we could generically convert a probabilistic CPA-secure cipher into a deterministic cipher that satisfies a somewhat weaker notion of security called deterministic CPA security.

(a) Show how to modify that construction so that we can convert any CPA-secure probabilistic cipher into a nonce-based CPA-secure cipher.

(b) Show how to combine the two approaches to get a cipher that is nonce-based CPA secure, but also satisfies the definition of deterministic CPA security if we drop the uniqueness requirement on nonces.

Discussion: This is an instance of a more general security property called **nonce re-use resilience**: the scheme provides full security if nonces are unique, and even if they are not, a weaker and still useful security guarantee is provided.

5.10 (Ciphertext expansion vs. security). Let $\mathcal{E} = (E, D)$ be an encryption scheme where messages and ciphertexts are bit strings.

(a) Suppose that for all keys and all messages m, the encryption of m is the exact same length as m. Show that (E, D) cannot be semantically secure under a chosen plaintext attack.

(b) Suppose that for all keys and all messages m, the encryption of m is exactly ℓ bits longer than the length of m. Show an attacker that can win the CPA security game using $\approx 2^{\ell/2}$ queries and advantage $\approx 1/2$. You may assume the message space contains many more than $2^{\ell/2}$ messages.

5.11 (CBC encryption with small blocks is insecure). Let's see a concrete example of the attack in the previous exercise. Let (E, D) be a CBC cipher, as in Section 5.4.3, built from a block cipher that operates on ℓ bit blocks. Construct an attacker that wins the CPA game against (E, D) that makes $\approx 2^{\ell/2}$ queries to its challenger and has advantage $\approx 1/2$. This attack was used to show that 3DES-CBC is no longer secure for Internet use due to its small 64-bit block size [19].

5.12 (Repeating ciphertexts). Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Assume that there are at least two messages in $\mathcal{M}$, that all messages have the same length, and that we can efficiently generate messages in $\mathcal{M}$ uniformly at random. Show that if $\mathcal{E}$ is CPA secure, then it is infeasible for an adversary to make an encryptor generate the same ciphertext twice. The precise attack game is as follows. The challenger chooses $k \in \mathcal{K}$ at random and the adversary makes a series of queries; the ith query is a message m_i, to which the challenger responds with $c_i \xleftarrow{\text{R}} E(k, m_i)$. The adversary wins the game if any two c_i's are the same. Show that if $\mathcal{E}$ is CPA secure, then every efficient adversary wins this game with negligible probability. In particular, show that the advantage of any adversary $\mathcal{A}$ in winning the repeated-ciphertext attack game is at most 2ϵ, where ϵ is the advantage of an adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) that breaks the scheme's CPA security.

5.13 (Predictable IVs). Let us see why in CBC mode an unpredictable IV is necessary for CPA security. Suppose a defective implementation of CBC encrypts a sequence of messages by always using the last ciphertext block of the ith message as the IV for the $(i+1)$-st message. The TLS 1.0

protocol, used to protect Web traffic, implements CBC encryption this way. Construct an efficient adversary that wins the CPA game against this implementation with advantage close to 1. We note that the Web-based BEAST attack [58] exploits this defect to completely break CBC encryption in TLS 1.0.

5.14 (An insecure nonce-based CBC mode). Consider the nonce-based CBC scheme $\mathcal{E}'$ described in Section 5.5.3. Suppose that the nonce space $\mathcal{N}$ is equal to block space $\mathcal{X}$ of the underlying block cipher $\mathcal{E} = (E, D)$, and the PRF F is just the encryption algorithm E. If the two keys k and k' in the construction are chosen independently, the scheme is secure. Your task is to show that if only one key k is chosen, and the other key k' is set to k, then the scheme is insecure.

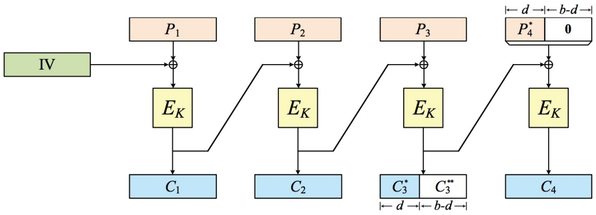
5.15 (Output feedback mode). Suppose F is a PRF defined over $(\mathcal{K}, \mathcal{X})$, and $\ell \geq 1$ is poly-bounded.

(a) Consider the following PRG $G : \mathcal{K} \to \mathcal{X}^\ell$. Let x_0 be an arbitrary, fixed element of $\mathcal{X}$. For $k \in \mathcal{K}$, let $G(k) := (x_1, \ldots, x_\ell)$, where $x_i := F(k, x_{i-1})$ for $i = 1, \ldots, \ell$. Show that G is a secure PRG, assuming F is a secure PRF and that $|\mathcal{X}|$ is super-poly.

(b) Next, assume that $\mathcal{X} = \{0,1\}^n$. We define a cipher $\mathcal{E} = (E, D)$, defined over $(\mathcal{K}, \mathcal{X}^\ell, \mathcal{X}^{\ell+1})$, as follows. Given a key $k \in \mathcal{K}$ and a message $(m_1, \ldots, m_\ell) \in \mathcal{X}^\ell$, the encryption algorithm E generates the ciphertext $(c_0, c_1, \ldots, c_\ell) \in \mathcal{X}^{\ell+1}$ as follows: it chooses $x_0 \in \mathcal{X}$ at random, and sets $c_0 = x_0$; it then computes $x_i = F(k, x_{i-1})$ and $c_i = m_i \oplus x_i$ for $i = 1, \ldots, \ell$. Describe the corresponding decryption algorithm D, and show that $\mathcal{E}$ is CPA secure, assuming F is a secure PRF and that $|\mathcal{X}|$ is super-poly.

Note: This construction is called **output feedback mode** (or **OFB**).

5.16 (CBC ciphertext stealing). One problem with CBC encryption is that messages need to be padded to a multiple of the block length and sometimes a dummy block needs to be added. The following figure describes a variant of CBC that eliminates the need to pad:



The method pads the last block with zeros if needed (a dummy block is never added), but the output ciphertext contains only the shaded parts of C_1, C_2, C_3, C_4. Note that, ignoring the IV, the ciphertext is the same length as the plaintext. This technique is called *ciphertext stealing.*

(a) Explain how decryption works.

(b) Can this method be used if the plaintext contains only one block?

5.17 (Single ciphertext block corruption in CBC mode). Let c be an ℓ block CBC-encrypted ciphertext, for some $\ell > 3$. Suppose that exactly one block of c is corrupted, and the result is decrypted using the CBC decryption algorithm. How many blocks of the decrypted plaintext are corrupted?

5.18 (The malleability of CBC mode). Let c be the CBC encryption of some message $m \in \mathcal{X}^{\ell}$, where $\mathcal{X} := \{0,1\}^n$. You do not know m. Let $\Delta \in \mathcal{X}$. Show how to modify the ciphertext c to obtain a new ciphertext c' that decrypts to m', where $m'[0] = m[0] \oplus \Delta$, and $m'[i] = m[i]$ for $i = 1, \ldots, \ell - 1$. That is, by modifying c appropriately, you can flip bits of your choice in the first block of the decryption of c, without affecting any of the other blocks.

5.19 (Online ciphers). In practice there is a strong desire to encrypt one block of plaintext at a time, outputting the corresponding block of ciphertext right away. This lets the system transmit ciphertext blocks as soon as they are ready without having to wait until the entire message is processed by the encryption algorithm.

(a) Define a CPA-like security game that captures this method of encryption. Instead of forcing the adversary to submit a complete pair of messages in every encryption query, the adversary should be allowed to issue a query indicating the beginning of a message, then repeatedly issue more queries containing message blocks, and finally issue a query indicating the end of a message. Responses to these queries will include all ciphertext blocks that can be computed given the information given.

(b) Show that randomized CBC encryption is not CPA secure in this model.

(c) Show that randomized counter mode is online CPA secure.

5.20 (Redundant bits do not harm CPA security). Let $\mathcal{E} = (E, D)$ be a CPA-secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Show that appending to a ciphertext additional data that is computed from the ciphertext does not damage CPA security. Specifically, let $g : \mathcal{C} \to \mathcal{Y}$ be some efficiently computable function. Show that the following modified cipher $\mathcal{E}' = (E', D')$ is CPA-secure:

$$E'(k, m) := \big\{c \leftarrow E(k, m),\ t \leftarrow g(c),\ \text{output } (c, t)\big\}$$
$$D'\big(k,\ (c, t)\big) := D(k, c)$$

5.21 (Broadcast encryption). In a broadcast encryption system, a sender can encrypt a message so that only a specified set of recipients can decrypt. Such a system is made up of three efficient algorithms (G, E, D): algorithm G is invoked as $G(n)$ and outputs an encryptor key ek, and n keys $k_1, \ldots, k_n$, one key for each recipient; algorithm E is invoked as $c \overset{\text{R}}{\leftarrow} E(ek, m, S)$, where m is the message and $S \subseteq \{1, \ldots, n\}$ is the intended set of recipients; algorithm D is invoked as $m \leftarrow D(k_i, c)$ for some $1 \le i \le n$, and correctly decrypts the given c whenever i is in the set S. More precisely, for all m and all subsets S of $\{1, \ldots, n\}$, we have that $D\big(k_i,\ E(ek, m, S)\big) = m$ for all $i \in S$.

(a) Describe the revocation scheme described in (5.35) in Section 5.6 as a broadcast encryption system. How do algorithms G, E, D work and what are ek and $k_1, \ldots, k_n$?

(b) A broadcast encryption scheme is secure if a set of colluding recipients B learns nothing about plaintexts encrypted for subsets of $\{1, \ldots, n\} \setminus B$, namely plaintexts that are not intended for the members of B. More precisely, CPA security of a broadcast encryption system is defined using the following two experiments, Experiment 0 and Experiment 1: In Experiment b, for $b = 0, 1$, the adversary begins by outputing a subset B of $\{1, \ldots, n\}$. The challenger then runs $G(n)$ and sends to the adversary all the keys named in B, namely $\{k_i\}_{i \in B}$. Now the adversary issues chosen plaintext queries, where query number j is a triple $(S_j, m_{j,0}, m_{j,1})$ for some set S_j in $\{1, \dots, n\} \setminus B$. The challenger sends back $c_j \xleftarrow{R} E(ek, m_{j,b}, S_j)$. The system is secure if the adversary cannot distinguish these two experiments.

Show that the scheme from part (a) is a secure broadcast encryption system, assuming the underlying header encryption scheme is CPA secure, and the body encryption scheme (E', D') is semantically secure.

Hint: Use a sequence of $2n - 1$ hybrids, one for each key in the tree of Fig. 5.5

Chapter 6

Message integrity

In previous chapters we focused on security against an eavesdropping adversary. The adversary had the ability to eavesdrop on transmitted messages, but could not change messages en-route. We showed that chosen plaintext security is the natural security property needed to defend against such attacks.

In this chapter we turn our attention to active adversaries. We start with the basic question of *message integrity*: Bob receives a message m from Alice and wants to convince himself that the message was not modified en-route. We will design a mechanism that lets Alice compute a short message integrity tag t for the message m and send the pair (m, t) to Bob, as shown in Fig. 6.1. Upon receipt, Bob checks the tag t and rejects the message if the tag fails to verify. If the tag verifies then Bob is assured that the message was not modified in transmission.

We emphasize that in this chapter the message itself need not be secret. Unlike previous chapters, our goal here is not to conceal the message. Instead, we only focus on message integrity. In Chapter 9 we will discuss the more general question of simultaneously providing message secrecy and message integrity. There are many applications where message integrity is needed, but message secrecy is not. We give two examples.

Example 6.1. Consider the problem of delivering financial news or stock quotes over the Internet. Although the news items themselves are public information, it is vital that no third party modify the data on its way to the user. Here message secrecy is irrelevant, but message integrity is critical. Our constructions will ensure that if user Bob rejects all messages with an invalid message integrity tag then an attacker cannot inject modified content that will look legitimate. One caveat is that an attacker can still change the order in which news reports reach Bob. For example, Bob might see report number 2 before seeing report number 1. In some settings this may cause the user to take an incorrect action. To defend against this, the news service may wish to include a sequence number with each report so that the user's machine can buffer reports and ensure that the user always sees news items in the correct order. □

In this chapter we are only concerned with attacks that attempt to modify data. We do not consider Denial of Service (DoS) attacks, where the attacker delays or prevents news items from reaching the user. DoS attacks are often handled by ensuring that the network contains redundant paths from the sender to the receiver so that an attacker cannot block all paths. We will not discuss these issues here.

Example 6.2. Consider an application program — such as a word processor or mail client —

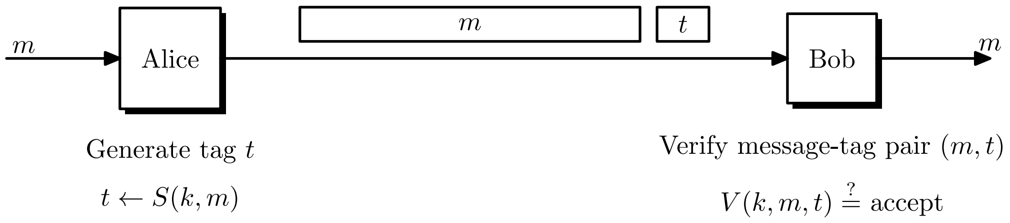


Figure 6.1: Short message integrity tag added to messages

stored on disk. Although the application code is not secret (it might even be in the public domain), its integrity is important. Before running the program the user wants to ensure that a virus did not modify the code stored on disk. To do so, when the program is first installed, the user computes a message integrity tag for the code and stores the tag on disk alongside the program. Then, every time, before starting the application the user can validate this message integrity tag. If the tag is valid, the user is assured that the code has not been modified since the tag was initially generated. Clearly a virus can overwrite both the application code and the integrity tag. Nevertheless, our constructions will ensure that no virus can fool the user into running unauthenticated code. As in our first example, the attacker can swap two authenticated programs — when the user starts application A he will instead be running application B. If both applications have a valid tag the system will not detect the swap. The standard defense against this is to include the program name in the executable file. That way, when an application is started the system can display to the user an authenticated application name. □

The question, then, is how to design a secure message integrity mechanism. We first argue the following basic principle:

> Providing message integrity between two communicating parties requires that the sending party has a secret key unknown to the adversary.

Without a secret key, ensuring message integrity is not possible: the adversary has enough information to compute tags for arbitrary messages of its choice — it knows how the message integrity algorithm works and needs no other information to compute tags. For this reason all cryptographic message integrity mechanisms require a secret key unknown to the adversary. In this chapter, we will assume that both sender and receiver will share the secret key; later in the book, this assumption will be relaxed.

We note that communication protocols not designed for security often use *keyless* integrity mechanisms. For example, the Ethernet protocol uses CRC32 as its message integrity algorithm. This algorithm, which is publicly available, outputs 32-bit tags embedded in every Ethernet frame. The TCP protocol uses a keyless 16-bit checksum which is embedded in every packet. We emphasize that these keyless integrity mechanisms are designed to detect *random* transmission errors, not malicious errors. The argument in the previous paragraph shows that an adversary can easily defeat these mechanisms and generate legitimate-looking traffic. For example, in the case of Ethernet, the adversary knows exactly how the CRC32 algorithm works and this lets him compute valid tags for arbitrary messages. He can then tamper with Ethernet traffic without being detected.

6.1 Definition of a message authentication code

We begin by defining a message integrity system based on a shared secret key. For historical reasons such systems are called Message Authentication Codes or MACs for short.

Definition 6.1. *A **MAC** system $\mathcal{I} = (S, V)$ is a pair of efficient algorithms, S and V, where S is called a **signing algorithm** and V is called a **verification algorithm**. Algorithm S is used to generate tags and algorithm V is used to verify tags.*

- *S is a probabilistic algorithm that is invoked as $t \xleftarrow{\text{R}} S(k, m)$, where k is a key, m is a message, and the output t is called a **tag**.*
- *V is a deterministic algorithm that is invoked as $r \leftarrow V(k, m, t)$, where k is a key, m is a message, t is a tag, and the output r is either* accept *or* reject*.*
- *We require that tags generated by S are always accepted by V; that is, the MAC must satisfy the following **correctness property**: for all keys k and all messages m,*

$$\Pr[V(k,\ m,\ S(k,\ m)\ ) = \mathsf{accept}] = 1.$$

*As usual, we say that keys lie in some finite **key space** $\mathcal{K}$, messages lie in a finite **message space** $\mathcal{M}$, and tags lie in some finite **tag space** $\mathcal{T}$. We say that $\mathcal{I} = (S, V)$ is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$.*

Fig. 6.1 illustrates how algorithms S and V are used for protecting network communications between two parties. Whenever algorithm V outputs accept for some message-tag pair (m, t), we say that t is a **valid tag** for m under key k, or that (m, t) is a **valid pair** under k. Naturally, we want MAC systems where tags are as short as possible so that the overhead of transmitting the tag is minimal.

We will explore a variety of MAC systems. The simplest type of system is one in which the signing algorithm S is deterministic, and the verification algorithm is defined as

$$V(k, m, t) = \begin{cases} \mathsf{accept} & \text{if } S(k, m) = t, \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

We shall call such a MAC system a **deterministic MAC system**. One property of a deterministic MAC system is that it has **unique tags**: for a given key k, and a given message m, there is a unique valid tag for m under k. Not all MAC systems we explore will have such a simple design: some have a randomized signing algorithm, so that for a given key k and message m, the output of $S(k, m)$ may be one of many possible valid tags, and the verification algorithm works some other way. As we shall see, such **randomized MAC systems** are not necessary to achieve security, but they can yield better efficiency/security trade-offs.

Secure MACs. Next, we turn to describing what it means for a MAC to be secure. To construct MACs that remain secure in a variety of applications we will insist on security in a very hostile environment. Since most real-world systems that use MACs operate in less hostile settings, our conservative security definitions will imply security for all these systems.

We first intuitively explain the definition and then motivate why this conservative definition makes sense. Suppose an adversary is attacking a MAC system $\mathcal{I} = (S, V)$. Let k be some

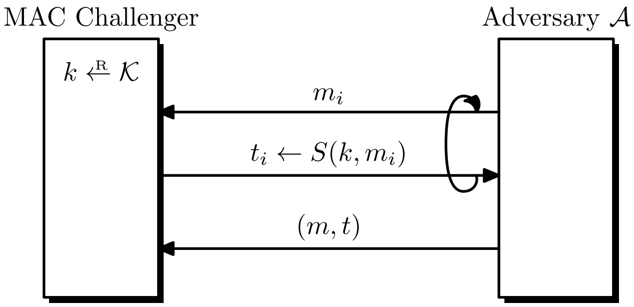


Figure 6.2: MAC attack game (Attack Game 6.1)

randomly chosen MAC key, which is unknown to the attacker. We allow the attacker to request tags $t := S(k, m)$ for arbitrary messages m of its choice. This attack, called a **chosen message attack**, enables the attacker to collect millions of valid message-tag pairs. Clearly we are giving the attacker considerable power — it is hard to imagine that a user would be foolish enough to sign arbitrary messages supplied by an attacker. Nevertheless, we will see that chosen message attacks come up in real world settings. We refer to message-tag pairs (m, t) that the adversary obtains using the chosen message attack as **signed pairs**.

Using the chosen message attack we ask the attacker to come up with an **existential MAC forgery**. That is, the attacker need only come up with some *new* valid message-tag pair (m, t). By "new", we mean a message-tag pair that is different from all of the signed pairs. The attacker is free to choose m arbitrarily; indeed, m need not have any special format or meaning and can be complete gibberish.

We say that a MAC system is secure if even an adversary who can mount a chosen message attack cannot create an existential forgery. This definition gives the adversary more power than it typically has in the real world and yet we ask it to do something that will normally be harmless; forging the MAC for a meaningless message seems to be of little use. Nevertheless, as we will see, this conservative definition is very natural and enables us to use MACs for lots of different applications.

More precisely, we define secure MACs using an attack game between a challenger and an adversary $\mathcal{A}$. The game is described below and in Fig. 6.2.

Attack Game 6.1 (MAC security). For a given MAC system $\mathcal{I} = (S, V)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger picks a random $k \xleftarrow{R} \mathcal{K}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots$, the ith *signing query* is a message $m_i \in \mathcal{M}$. Given m_i, the challenger computes a tag $t_i \xleftarrow{R} S(k, m_i)$, and then gives t_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, t) \in \mathcal{M} \times \mathcal{T}$ that is not among the signed pairs, i.e.,

$$(m, t) \notin \{(m_1, t_1), (m_2, t_2), \ldots\}.$$

We say that $\mathcal{A}$ wins the above game if (m, t) is a valid pair under k (i.e., $V(k, m, t) = \mathsf{accept}$). We define $\mathcal{A}$'s advantage with respect to $\mathcal{I}$, denoted $\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$, as the probability that $\mathcal{A}$ wins the game. Finally, we say that $\mathcal{A}$ is a **Q-query MAC adversary** if $\mathcal{A}$ issues at most Q signing queries. □

Definition 6.2. *We say that a MAC system $\mathcal{I}$ is secure if for all efficient adversaries $\mathcal{A}$, the value* $\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$ *is negligible.*

In case the adversary wins Attack Game 6.1, the pair (m, t) that it sends to the challenger is called an **existential forgery**. MAC systems that satisfy Definition 6.2 are said to be **existentially unforgeable under a chosen message attack**.

In the case of a deterministic MAC system, the only way for $\mathcal{A}$ to win Attack Game 6.1 is to produce a valid message-tag pair (m, t) for some new message $m \notin \{m_1, m_2, \ldots\}$. Indeed, security in this case just means that S is *unpredictable*, in the sense described in Section 4.1.1; that is, given $S(k, m_1), S(k, m_2), \ldots,$ it is hard to predict $S(k, m)$ for any $m \notin \{m_1, m_2, \ldots\}$.

In the case of a randomized MAC system, our security definition captures a stronger property. There may be many valid tags for a given message. Let m be some message and suppose the adversary requests one or more valid tags $t_1, t_2, \ldots$ for m. Can the adversary produce a new valid tag t' for m? (i.e. a tag satisfying $t' \notin \{t_1, t_2, \ldots\}$). Our definition says that a valid pair (m, t'), where t' is new, is a valid existential forgery. Therefore, for a MAC to be secure it must be difficult for an adversary to produce a new valid tag t' for a previously signed message m. This may seem like an odd thing to require of a MAC. If the adversary already has valid tags for m, why should we care if it can produce another one? As we will see in Chapter 9, our security definition, which prevents the adversary from producing new tags on signed messages, is necessary for the applications we have in mind.

Going back to the examples in the introduction, observe that existential unforgeability implies that an attacker cannot create a fake news report with a valid tag. Similarly, the attacker cannot tamper with a program on disk without invalidating the tag for the program. Note, however, that when using MACs to protect application code, users must provide their secret MAC key every time they want to run the application. This will quickly annoy most users. In Chapter 8 we will discuss a keyless method to protect public application code.

To exercise the definition of secure MACs let us first see a few consequences of it. Let $\mathcal{I} = (S, V)$ be a MAC defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and let k be a random key in $\mathcal{K}$.

Example 6.3. Suppose m_1 and m_2 are almost identical messages. Say m_1 is a money transfer order for \$100 and m_2 is a transfer order for \$101. Clearly, an adversary who intercepts a valid tag for m_1 should not be able to deduce from it a valid tag for m_2. A MAC system that satisfies Definition 6.2 ensures this. To see why, suppose an adversary $\mathcal{A}$ can forge the tag for m_2 given the tag for m_1. Then $\mathcal{A}$ can win Attack Game 6.1: it uses the chosen message attack to request a tag for m_1, deduces a forged tag t_2 for m_2, and outputs (m_2, t_2) as a valid existential forgery. Clearly $\mathcal{A}$ wins Attack Game 6.1. Hence, existential unforgeability captures the fact that a tag for one message m_1 gives no useful information for producing a tag for another message m_2, even when m_2 is almost identical to m_1. □

Example 6.4. Our definition of secure MACs gives the adversary the ability to obtain the tag for arbitrary messages. This may seem like giving the adversary too much power. In practice, however, there are many scenarios where chosen message attacks are feasible. The reason is that the MAC signer often does not know the source of the data being signed. For example, consider a backup system that dumps the contents of disk to backup tapes. Since backup integrity is important, the system computes an integrity tag on every disk block that it writes to tape. The tag is stored on tape along with the data block. Now, suppose an attacker writes data to a low security part of disk. The attacker's data will be backed up and the system will compute a tag over it. By examining the resulting backup tape the attacker obtains a tag on its chosen message. If the MAC system is secure against a chosen message attack then this does not help the attacker break the system. □

Remark 6.1. Just as we did for other security primitives, one can generalize the notion of a secure MAC to the multi-key setting, and prove that a secure MAC is also secure in the multi-key setting. See Exercise 6.3. □

6.1.1 Mathematical details

As usual, we give a more mathematically precise definition of a MAC, using the terminology defined in Section 2.3. This section may be safely skipped on first reading.

Definition 6.3 (MAC). *A **MAC** system is a pair of efficient algorithms, S and V, along with three families of spaces with system parameterization P:*

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \textit{and} \quad \mathbf{T} = \{\mathcal{T}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

As usual, $\lambda \in \mathbb{Z}_{\geq 1}$ is a security parameter and $\Lambda \in \mathrm{Supp}(P(\lambda))$ is a domain parameter. We require that

1. *$\mathbf{K}$, $\mathbf{M}$, and $\mathbf{T}$ are efficiently recognizable.*

2. *$\mathbf{K}$ is efficiently sampleable.*

3. *Algorithm S is an efficient probabilistic algorithm that on input λ, Λ, k, m, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, and $m \in \mathcal{M}_{\lambda,\Lambda}$, outputs an element of $\mathcal{T}_{\lambda,\Lambda}$.*

4. *Algorithm V is an efficient deterministic algorithm that on input $\lambda, \Lambda, k, m, t$, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, $m \in \mathcal{M}_{\lambda,\Lambda}$, and $t \in \mathcal{T}_{\lambda,\Lambda}$, outputs either* accept *or* reject*.*

In defining security, we parameterize Attack Game 6.1 by the security parameter λ, which is given to both the adversary and the challenger. The advantage $\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$ is then a function of λ. Definition 6.2 should be read as saying that $\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}](\lambda)$ is a negligible function.

6.2 MAC verification queries do not help the attacker

In our definition of secure MACs (Attack Game 6.1) the adversary has no way of testing whether a given message-tag pair is valid. In fact, the adversary cannot even tell if it wins the game, since only the challenger has the secret key needed to run the verification algorithm. In real life, an attacker capable of mounting a chosen message attack can probably also test whether a given message-tag pair is valid. For example, the attacker could build a packet containing the message-tag pair in question and send this packet to the victim's machine. Then, by examining the machine's behavior the attacker can tell whether the packet was accepted or dropped, indicating whether the tag was valid or not.

Consequently, it makes sense to extend Attack Game 6.1 by giving the adversary the extra power to verify message-tag pairs. Of course, we continue to allow the adversary to request tags for arbitrary messages of its choice.

Attack Game 6.2 (MAC security with verification queries). For a given MAC system $\mathcal{I} = (S, V)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger picks a random $k \xleftarrow{\text{R}} \mathcal{K}$.
- $\mathcal{A}$ queries the challenger several times. Each query can be one of two types:
  - *Signing query:* for $i = 1, 2, \ldots,$ the ith signing query consists of a message $m_i \in \mathcal{M}$. The challenger computes a tag $t_i \xleftarrow{\text{R}} S(k, m_i)$, and gives t_i to $\mathcal{A}$.
  - *Verification query:* for $j = 1, 2, \ldots,$ the jth verification query consists of a message-tag pair $(\hat{m}_j, \hat{t}_j) \in \mathcal{M} \times \mathcal{T}$ that is not among the previously signed pairs, i.e.,
  $$(\hat{m}_j, \hat{t}_j) \notin \{(m_1, t_1), (m_2, t_2), \ldots\}.$$
  The challenger responds to $\mathcal{A}$ with $V(k, \hat{m}_j, \hat{t}_j)$.

We say that $\mathcal{A}$ wins the above game if the challenger ever responds to a verification query with accept. We define $\mathcal{A}$'s advantage with respect to $\mathcal{I}$, denoted $\text{MAC}^{\text{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$, as the probability that $\mathcal{A}$ wins the game. □

The two definitions are equivalent. Attack Game 6.2 is essentially the same as the original Attack Game 6.1, except that $\mathcal{A}$ can issue MAC verification queries. We prove that this extra power does not help the adversary.

Theorem 6.1. *If $\mathcal{I}$ is a secure MAC system, then it is also secure in the presence of verification queries.*

> *In particular, for every MAC adversary $\mathcal{A}$ that attacks $\mathcal{I}$ as in Attack Game 6.2, and which makes at most Q_{v} verification queries and at most Q_{s} signing queries, there exists a Q_{s}-query MAC adversary $\mathcal{B}$ that attacks $\mathcal{I}$ as in Attack Game 6.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
> $$\text{MAC}^{\text{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \le \text{MAC}\mathsf{adv}[\mathcal{B}, \mathcal{I}] \cdot Q_{\text{v}}.$$

Proof idea. Let $\mathcal{A}$ be a MAC adversary that attacks $\mathcal{I}$ as in Attack Game 6.2, and which makes at most Q_{v} verification queries and at most Q_{s} signing queries. From adversary $\mathcal{A}$, we build an adversary $\mathcal{B}$ that attacks $\mathcal{I}$ as in Attack Game 6.1 and makes at most Q_{s} signing queries. Adversary $\mathcal{B}$ can easily answer $\mathcal{A}$'s signing queries by forwarding them to $\mathcal{B}$'s challenger and relaying the resulting tags back to $\mathcal{A}$.

The question is how to respond to $\mathcal{A}$'s verification queries. By definition, $\mathcal{A}$ only submits verification queries on message pairs that are not among the previously signed pairs. So $\mathcal{B}$ adopts a simple strategy: it responds with reject to all verification queries from $\mathcal{A}$. If $\mathcal{B}$ answers incorrectly, it has a forgery which lets it win Attack Game 6.1. Unfortunately, $\mathcal{B}$ does not know which of these verification queries is a forgery, so it simply guesses, choosing one at random. Since $\mathcal{A}$ makes at most Q_{v} verification queries, $\mathcal{B}$ will guess correctly with probability at least $1/Q_{\text{v}}$. This is the source of the Q_{v} factor in the error term. □

Proof. In more detail, adversary $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$ in Attack Game 6.2, while at the same time, it plays the role of adversary in Attack Game 6.1, interacting with the MAC challenger in that game. The logic is as follows:

```
initialization:
    $\omega \xleftarrow{R} \{1, \ldots, Q_v\}$
upon receiving a signing query $m_i \in \mathcal{M}$ from $\mathcal{A}$ do:
    forward $m_i$ to the MAC challenger, obtaining the tag $t_i$
    send $t_i$ to $\mathcal{A}$
upon receiving a verification query $(\hat{m}_j, \hat{t}_j) \in \mathcal{M} \times \mathcal{T}$ from $\mathcal{A}$ do:
    if $j = \omega$
        then output $(\hat{m}_j, \hat{t}_j)$ as a candidate forgery pair and halt
        else  send reject to $\mathcal{A}$
```

To rigorously justify the construction of adversary $\mathcal{B}$, we analyze the behavior of $\mathcal{A}$ in three closely related games.

Game 0. This is the original attack game, as played between the challenger in Attack Game 6.2 and adversary $\mathcal{A}$. Here is the logic of the challenger in this game:

```
    initialization:
        $k \xleftarrow{R} \mathcal{K}$
    upon receiving a signing query $m_i \in \mathcal{M}$ from $\mathcal{A}$ do:
        $t_i \xleftarrow{R} S(k, m_i)$
        send $t_i$ to $\mathcal{A}$
    upon receiving a verification query $(\hat{m}_j, \hat{t}_j) \in \mathcal{M} \times \mathcal{T}$ from $\mathcal{A}$ do:
        $r_j \leftarrow V(k, \hat{m}_j, \hat{t}_j)$
(*)     send $r_j$ to $\mathcal{A}$
```

Let W_0 be the event that in Game 0, $r_j = \mathsf{accept}$ for some j. Evidently,

$$\Pr[W_0] = \mathrm{MAC}^{\mathrm{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}]. \tag{6.1}$$

Game 1. This is the same as Game 0, except that the line marked $(*)$ above is changed to:

```
        send reject to $\mathcal{A}$
```

That is, when responding to a verification query, the challenger always responds to $\mathcal{A}$ with reject. We also define W_1 to be the event that in Game 1, $r_j = \mathsf{accept}$ for some j. Even though the challenger does not notify $\mathcal{A}$ that W_1 occurs, both Games 0 and 1 proceed identically until this event happens, and so events W_0 and W_1 are really the same; therefore,

$$\Pr[W_1] = \Pr[W_0]. \tag{6.2}$$

Also note that in Game 1, although the r_j values are used to define the winning condition, they are not used for any other purpose, and so do not influence the attack in any way.

Game 2. This is the same as Game 1, except that at the beginning of the game, the challenger chooses $\omega \xleftarrow{\text{R}} \{1, \ldots, Q_{\text{v}}\}$. We define W_2 to be the event that in Game 2, $r_\omega = \mathsf{accept}$. Since the choice of ω is independent of the attack itself, we have

$$\Pr[W_2] \geq \Pr[W_1]/Q_{\text{v}}. \tag{6.3}$$

Evidently, by construction, we have

$$\Pr[W_2] = \text{MAC}\mathsf{adv}[\mathcal{B}, \mathcal{I}]. \tag{6.4}$$

The theorem now follows from (6.1)–(6.3). □

In summary, we showed that Attack Game 6.2, which gives the adversary more power, is equivalent to Attack Game 6.1 used in defining secure MACs. The reduction introduces a factor of Q_{v} in the error term. Throughout the book we will make use of both attack games:

- When constructing secure MACs it is easier to use Attack Game 6.1 which restricts the adversary to signing queries only. This makes it easier to prove security since we only have to worry about one type of query. We will use this attack game throughout the chapter.
- When using secure MACs to build higher level systems (such as authenticated encryption) it is more convenient to assume that the MAC is secure with respect to the stronger adversary described in Attack Game 6.2.

We also point out that if we had used a weaker notion of security, in which the adversary only wins by presenting a valid tag on a new message (rather than new valid message-tag pair), then the analogs of Attack Game 6.1 and Attack Game 6.2 are *not* equivalent (see Exercise 6.7).

6.3 Constructing MACs from PRFs

We now turn to constructing secure MACs using the tools at our disposal. In previous chapters we used pseudo random functions (PRFs) to build various encryption systems. We gave examples of practical PRFs such as AES (while AES is a block cipher it can be viewed as a PRF thanks to the PRF switching lemma, Theorem 4.4). Here we show that any secure PRF can be directly used to build a secure MAC.

Recall that a PRF is an algorithm F that takes two inputs, a key k and an input data block x, and outputs a value $y := F(k, x)$. As usual, we say that F is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where keys are in $\mathcal{K}$, inputs are in $\mathcal{X}$, and outputs are in $\mathcal{Y}$. For a PRF F we define the **deterministic MAC system $\mathcal{I} = (S, V)$ derived from F** as:

$$S(k,\ m) := F(k,\ m);$$

$$V(k,\ m,\ t) := \begin{cases} \mathsf{accept} & \text{if } F(k, m) = t, \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

As already discussed, any PRF with a large (i.e., super-poly) output space is unpredictable (see Section 4.1.1), and therefore, as discussed in Section 6.1, the above construction yields a secure MAC. For completeness, we state this as a theorem:

Theorem 6.2. *Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, where $|\mathcal{Y}|$ is super-poly. Then the deterministic MAC system $\mathcal{I}$ derived from F is a secure MAC.*

> *In particular, for every Q-query MAC adversary $\mathcal{A}$ that attacks $\mathcal{I}$ as in Attack Game 6.1, there exists a $(Q+1)$-query PRF adversary $\mathcal{B}$ that attacks F as in Attack Game 4.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}, F] + 1/|\mathcal{Y}|$$

Proof idea. Let $\mathcal{A}$ be an efficient MAC adversary. We derive an upper bound on MAC$\mathsf{adv}[\mathcal{A}, \mathcal{I}]$ by bounding $\mathcal{A}$'s ability to generate forged message-tag pairs. As usual, replacing the underlying secure PRF F with a truly random function f in Funs$[\mathcal{X}, \mathcal{Y}]$ does not change $\mathcal{A}$'s advantage much. But now that the adversary $\mathcal{A}$ is interacting with a truly random function it is faced with a hopeless task: using the chosen message attack it obtains the value of f at a few points of its choice. He then needs to guess the value of $f(m) \in \mathcal{Y}$ at some new point m. But since f is a truly random function, $\mathcal{A}$ has no information about $f(m)$, and therefore has little chance of guessing $f(m)$ correctly. □

Proof. We make this intuition rigorous by letting $\mathcal{A}$ interact with two closely related challengers.

Game 0. As usual, we begin by reviewing the challenger in the MAC Attack Game 6.1 as it applies to $\mathcal{I}$. We implement the challenger in this game as follows:

```
(*)  $k \xleftarrow{R} \mathcal{K}$, $f \leftarrow F(k, \cdot)$
     upon receiving the $i$th signing query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) do:
         $t_i \leftarrow f(m_i)$
         send $t_i$ to the adversary
```

At the end of the game, the adversary outputs a message-tag pair (m, t). We define W_0 to be the event that the condition

$$t = f(m) \qquad \text{and} \qquad m \notin \{m_1, m_2, \ldots\} \tag{6.5}$$

holds in Game 0. Clearly, $\Pr[W_0] = \text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$.

Game 1. We next play the usual "PRF card," replacing the function $F(k, \cdot)$ by a truly random function f in Funs$[\mathcal{X}, \mathcal{Y}]$. Intuitively, since F is a secure PRF, the adversary $\mathcal{A}$ should not notice the difference. Our challenger in Game 1 is the same as in Game 0 except that we change line (*) as follows:

```
(*)  $f \xleftarrow{R}$ Funs$[\mathcal{X}, \mathcal{Y}]$
```

Let W_1 to be the event that condition (6.5) holds in Game 1. We construct a $(Q+1)$-query PRF adversary $\mathcal{B}$ such that:

$$\left|\Pr[W_1] - \Pr[W_0]\right| = \text{PRF}\mathsf{adv}[\mathcal{B}, F]. \tag{6.6}$$

Adversary $\mathcal{B}$ responds to $\mathcal{A}$'s chosen message queries by querying its own PRF challenger. Eventually $\mathcal{A}$ outputs a candidate MAC forgery (m, t) where m is not one of its chosen message queries. Now $\mathcal{B}$ queries its PRF challenger at m and gets back some $t' \in \mathcal{Y}$. If $t = t'$ then $\mathcal{B}$ outputs 0; otherwise it outputs 1. A simple argument shows that this $\mathcal{B}$ satisfies (6.6).

Next, we directly bound $\Pr[W_1]$. The adversary $\mathcal{A}$ sees the values of f at various points $m_1, m_2, \ldots$ and is then required to guess the value of f at some new point m. But since f is a truly random function, the value $f(m)$ is independent of its value at all other points. Hence, since

$m \notin \{m_1, m_2, \ldots\}$, adversary $\mathcal{A}$ will guess $f(m)$ with probability $1/|\mathcal{Y}|$. Therefore, $\Pr[W_1] \leq 1/|\mathcal{Y}|$. Putting this together with (6.6), we obtain

$$\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}] = \Pr[W_0] \leq \bigl|\Pr[W_0] - \Pr[W_1]\bigr| + \Pr[W_1] \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{1}{|\mathcal{Y}|}$$

as required. □

Concrete tag lengths. The theorem shows that to ensure $\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}] < 2^{-128}$ we need a PRF whose output space $\mathcal{Y}$ satisfies $|\mathcal{Y}| > 2^{128}$. If the output space $\mathcal{Y}$ is $\{0,1\}^n$ for some n, then the resulting tags must be at least 128 bits long.

6.4 Prefix-free PRFs for long messages

In the previous section we saw that any secure PRF is also a secure MAC. However, the concrete examples of PRFs from Chapter 4 only take short inputs and can therefore only be used to provide integrity for very short messages. For example, viewing AES as a PRF gives a MAC for 128-bit messages. Clearly, we want to build MACs for much longer messages.

All the MAC constructions in this chapter follow the same paradigm: they start from a PRF for short inputs (like AES) and produce a PRF, and therefore a MAC, for much longer inputs. Hence, our goal for the remainder of the chapter is the following:

given a secure PRF on short inputs construct a secure PRF on long inputs.

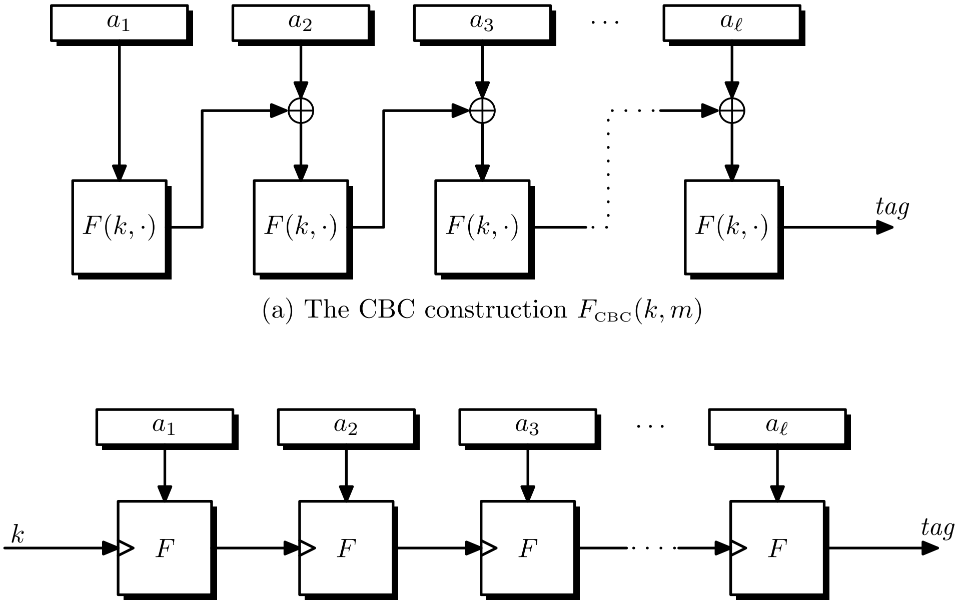
We solve this problem in three steps:

- First, in this section we construct *prefix-free secure* PRFs for long inputs. More precisely, given a secure PRF that operates on single-block (e.g., 128-bit) inputs, we construct a prefix-free secure PRF that operates on variable-length sequences of blocks. Recall that a prefix-free secure PRF (Definition 4.5) is only secure in a limited sense: we only require that *prefix-free adversaries* cannot distinguish the PRF from a random function. A prefix-free PRF adversary issues queries that are non-empty sequences of blocks, and no query can be a proper prefix of another.

- Second, in the next few sections we show how to convert prefix-free secure PRFs for long inputs into fully secure PRFs for long inputs. Thus, by the end of these sections we will have several secure PRFs, and therefore secure MACs, that operate on long inputs.

- Third, in Section 6.8 we show how to convert a PRF that operates on messages that are strings of blocks into a PRF that operates on strings of *bits*.

Prefix-free PRFs. We begin with two classic constructions for prefix-free secure PRFs. The **CBC** construction is shown in Fig. 6.3a. The **cascade** construction is shown in Fig. 6.3b. We show that when the underlying F is a secure PRF, both CBC and cascade are prefix-free secure PRFs.



(a) The CBC construction $F_{\text{CBC}}(k, m)$

(b) The cascade construction $F^*(k, m)$

Figure 6.3: Two prefix-free secure PRFs

6.4.1 The CBC prefix-free secure PRF

Let F be a PRF that maps n-bit inputs to n-bit outputs. In symbols, F is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$ where $\mathcal{X} = \{0,1\}^n$. For any poly-bounded value ℓ, we build a new PRF, denoted F_{CBC}, that maps messages in $\mathcal{X}^{\leq \ell}$ to outputs in $\mathcal{X}$. The function F_{CBC}, described in Fig. 6.3a, works as follows:

```
input: $k \in \mathcal{K}$ and $m = (a_1, \ldots, a_v) \in \mathcal{X}^{\leq \ell}$ for some $v \in \{0, \ldots, \ell\}$
output: a tag in $\mathcal{X}$
    $t \leftarrow 0^n$
    for $i \leftarrow 1$ to $v$ do:
        $t \leftarrow F(k,\ \ a_i \oplus t\ )$
    output $t$
```

F_{CBC} is similar to CBC mode encryption from Fig. 5.4, but with two important differences. First, F_{CBC} does not output any intermediate values along the CBC chain. Second, F_{CBC} uses a fixed IV, namely 0^n, where as CBC mode encryption uses a random IV per message.

The following theorem shows that F_{CBC} is a prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{X})$.

Theorem 6.3. *Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$ where $\mathcal{X} = \{0,1\}^n$ and $|\mathcal{X}| = 2^n$ is super-poly. Then for any poly-bounded value ℓ, we have that F_{CBC} is a prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{X})$.*

In particular, for every prefix-free PRF adversary $\mathcal{A}$ that attacks F_{CBC} as in Attack Game 4.2, and issues at most Q queries, there exists a PRF adversary $\mathcal{B}$ that attacks F as in Attack

Game 4.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, F_{\mathrm{CBC}}] \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{(Q\ell)^2}{2|\mathcal{X}|}. \tag{6.7}$$

Exercise 6.6 develops an attack on fixed-length F_{CBC} that demonstrates that security degrades quadratically in Q. This shows that the quadratic dependence on Q in (6.7) is necessary. A more difficult proof of security shows that security only degrades linearly in ℓ (see Section 6.13). In particular, the error term in (6.7) can be reduced to an expression dominated by $O(Q^2\ell/|\mathcal{X}|)$

Proof idea. We represent the adversary's queries in a rooted tree, where edges in the tree are labeled by message blocks (i.e., elements of $\mathcal{X}$). A query for $F_{\mathrm{CBC}}(k, m)$, where $m = (a_1, \ldots, a_v) \in \mathcal{X}^v$ and $1 \leq v \leq \ell$, defines a path in the tree, starting at the root, as follows:

$$\mathit{root} \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \cdots \xrightarrow{a_v} p_v. \tag{6.8}$$

Thus, two messages m and m' correspond to paths in the tree which both start at the root; these two paths may share a common initial subpath corresponding to the longest common prefix of m and m'.

With each node p in this tree, we associate a value $\gamma_p \in \mathcal{X}$ which represents the computed value in the CBC chain. More precisely, we define $\gamma_{\mathit{root}} := 0^n$, and for any non-root node q with parent p, if the corresponding edge in the tree is $p \overset{a}{\to} q$, then $\gamma_q := F(k, \gamma_p \oplus a)$. With these conventions, we see that if a message m traces out a path as in (6.8), then $\gamma_{p_v} = F_{\mathrm{CBC}}(k, m)$.

The crux of the proof is to argue that if F behaves like a random function, then for every pair of distinct edges in the tree, say $p \overset{a}{\longrightarrow} q$ and $p' \overset{a'}{\longrightarrow} q'$, we have $\gamma_p \oplus a \neq \gamma_{p'} \oplus a'$ with overwhelming probability. To prove that there are no collisions of this type, the prefix-freeness restriction is critical, as it guarantees that the adversary never sees γ_p and $\gamma_{p'}$, and hence a and a' are independent of these values. Once we have established that there are no collisions of these types, it will follow that all values associated with non-root nodes are random and independent, and this holds in particular for the values associated with the leaves, which represent the outputs of F_{CBC} seen by the adversary. Therefore, the adversary cannot distinguish F_{CBC} from a random function. □

Proof. We make this intuition rigorous by letting $\mathcal{A}$ interact with four closely related challengers in four games. For $j = 0, 1, 2, 3$, we let W_j be the event that $\mathcal{A}$ outputs 1 at the end of Game j.

Game 0. This is Experiment 0 of Attack Game 4.2.

Game 1. We next play the usual "PRF card," replacing the function $F(k, \cdot)$ by a truly random function f in Funs$[\mathcal{X}, \mathcal{X}]$. Clearly, we have

$$\bigl|\Pr[W_1] - \Pr[W_0]\bigr| = \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] \tag{6.9}$$

for an efficient adversary $\mathcal{B}$.

Game 2. We now make a purely conceptual change, implementing the random function f as a "faithful gnome" (as in Section 4.4.2). However, it will be convenient for us to do this in a particular way, using the "query tree" discussed above.

To this end, first let $B := Q\ell$, which represents an upper bound on the number of points at which f will be evaluated. Our challenger first prepares random values

$$\beta_i \overset{\mathrm{R}}{\leftarrow} \mathcal{X} \qquad (i = 1, \ldots, B).$$

These will be the only random values used by our challenger.

As the adversary makes queries, our challenger will dynamically build up the query tree. Initially, the tree contains only the root. Whenever the adversary makes a query, the challenger traces out the corresponding path in the existing query tree; at some point, this path will extend beyond the existing query tree, and our challenger adds the necessary nodes and edges so that the query tree grows to include the new path.

Our challenger must also compute the values γ_p associated with each node. Initially, $\gamma_{root} = 0^n$. When adding a new edge $p \xrightarrow{a} q$ to the tree, if this is the ith edge being added (for $i = 1, \ldots, B$), our challenger does the following:

$$\begin{aligned} & \gamma_q \leftarrow \beta_i \\ (*) \qquad & \text{if } \exists \text{ another edge } p' \xrightarrow{a'} q' \text{ with } \gamma_{p'} \oplus a' = \gamma_p \oplus a \text{ then } \gamma_q \leftarrow \gamma_{q'} \end{aligned}$$

The idea is that we use the next unused value in our prepared list $\beta_1, \ldots, \beta_B$ as the "default" value for γ_q. The line marked $(*)$ performs the necessary consistency check, which ensures that our gnome is indeed faithful.

Because this change is purely conceptual, we have

$$\Pr[W_2] = \Pr[W_1]. \tag{6.10}$$

Game 3. Next, we make our gnome forgetful, by removing the consistency check marked $(*)$ in the logic in Game 2.

To analyze the effect of this change, let Z be the event that *in Game 3*, for some distinct pair of edges $p \xrightarrow{a} q$ and $p' \xrightarrow{a'} q'$, we have $\gamma_{p'} \oplus a' = \gamma_p \oplus a$.

Now, the only randomly chosen values in Games 2 and 3 are the random choices of the adversary, *Coins*, and the list of values $\beta_1, \ldots, \beta_B$. Observe that for any fixed choice of values *Coins*, $\beta_1, \ldots, \beta_B$, if Z does not occur, then in fact Games 2 and 3 proceed identically. Therefore, we may apply the Difference Lemma (Theorem 4.7), obtaining

$$\big|\Pr[W_3] - \Pr[W_2]\big| \leq \Pr[Z]. \tag{6.11}$$

We next bound $\Pr[Z]$. Consider two distinct edges $p \xrightarrow{a} q$ and $p' \xrightarrow{a'} q'$. We want to bound the probability that $\gamma_{p'} \oplus a' = \gamma_p \oplus a$, which is equivalent to

$$\gamma_{p'} \oplus \gamma_p = a' \oplus a. \tag{6.12}$$

There are two cases to consider.

Case 1: $p = p'$. Since the edges are distinct, we must have $a' \neq a$, and hence (6.12) holds with probability 0.

Case 2: $p \neq p'$. The requirement that the adversary's queries are prefix free implies that in Game 3, the adversary never sees — or learns anything about — the values γ_p and $\gamma_{p'}$. One of p or p' could be the root, but not both. It follows that the value $\gamma_p \oplus \gamma_{p'}$ is uniformly distributed over $\mathcal{X}$ and is independent of $a \oplus a'$. From this, it follows that (6.12) holds with probability $1/|\mathcal{X}|$.

By the union bound, it follows that

$$\Pr[Z] \leq \frac{B^2}{2|\mathcal{X}|}. \tag{6.13}$$

Combining (6.9), (6.10), (6.11), and (6.13), we obtain

$$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, F_{\mathrm{CBC}}] = \big|\Pr[W_3] - \Pr[W_0]\big| \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{B^2}{2|\mathcal{X}|}. \tag{6.14}$$

Moreover, Game 3 corresponds exactly to Experiment 1 of Attack Game 4.2, from which the theorem follows. □

6.4.2 The cascade prefix-free secure PRF

Let F be a PRF that takes keys in $\mathcal{K}$ and produces outputs in $\mathcal{K}$. In symbols, F is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{K})$. For any poly-bounded value ℓ, we build a new PRF F^*, called the **cascade of F**, that maps messages in $\mathcal{X}^{\leq \ell}$ to outputs in $\mathcal{K}$. The function F^*, illustrated in Fig. 6.3b, works as follows:

```
input: $k \in \mathcal{K}$ and $m = (a_1, \ldots, a_v) \in \mathcal{X}^{\leq \ell}$ for some $v \in \{0, \ldots, \ell\}$
output: a tag in $\mathcal{K}$
    $t \leftarrow k$
    for $i \leftarrow 1$ to $v$ do:
        $t \leftarrow F(t,\ a_i)$
    output $t$
```

The following theorem shows that F^* is a prefix-free secure PRF.

Theorem 6.4. *Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{K})$. Then for any poly-bounded value ℓ, the cascade F^* of F is a prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{K})$.*

> *In particular, for every prefix-free PRF adversary $\mathcal{A}$ that attacks F^* as in Attack Game 4.2, and issues at most Q queries, there exists a PRF adversary $\mathcal{B}$ that attacks F as in Attack Game 4.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, F^*] \leq Q\ell \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F]. \tag{6.15}$$

Exercise 6.6 develops an attack on fixed-length F^* that demonstrates that security degrades quadratically in Q. This is disturbing as it appears to contradict the linear dependence on Q in (6.15). However, rest assured there is no contradiction here. The adversary $\mathcal{A}$ from Exercise 6.6, which uses $\ell = 3$, has advantage about $1/2$ when Q is about $\sqrt{|\mathcal{K}|}$. Plugging $\mathcal{A}$ into the proof of Theorem 6.4 we obtain a PRF adversary $\mathcal{B}$ that attacks the PRF F by making about Q queries to gain an advantage about $1/Q$. Note that $1/Q \approx Q/|\mathcal{K}|$ when Q is close to $\sqrt{|\mathcal{K}|}$. There is nothing surprising about this adversary $\mathcal{B}$: it is essentially the universal PRF attacker from Exercise 4.27. Hence, (6.15) is consistent with the attack from Exercise 6.6. Another way to view this is that the quadratic dependence on Q is already present in (6.15) because there is an implicit factor of Q hiding in the quantity $\mathrm{PRF}\mathsf{adv}[\mathcal{B}, F]$.

The proof of Theorem 6.4 is similar to the proof that the variable-length tree construction in Section 4.6 is a prefix-free secure PRF (Theorem 4.11). Let us briefly explain how to extend the proof of Theorem 4.11 to prove Theorem 6.4.

Relation to the tree construction. The cascade construction is a generalization of the variable-length tree construction of Section 4.6. Recall that the tree construction builds a secure PRF from a secure PRG that maps a seed to a pair of seeds. It is easy to see that when F is a PRF defined over $(\mathcal{K}, \{0,1\}, \mathcal{K})$ then Theorem 6.4 is an immediate corollary of Theorem 4.11: simply define the PRG G mapping $k \in \mathcal{K}$ to $G(k) := (F(k,0), F(k,1)) \in \mathcal{K}^2$, and observe that cascade applied to F is the same as the variable-length tree construction applied to G.

The proof of Theorem 4.11 generalizes easily to prove Theorem 6.4 for any PRF. For example, suppose that F is defined over $(\mathcal{K}, \{0,1,2\}, \mathcal{K})$. This corresponds to a PRG G mapping $k \in \mathcal{K}$ to $G(k) := (F(k,0), F(k,1), F(k,2)) \in \mathcal{K}^3$. The cascade construction applied to F can be viewed as a ternary tree, instead of a binary tree, and the proof of Theorem 4.11 carries over with no essential changes.

But why stop at width three? We can make the tree as wide as we wish. The cascade construction using a PRF F defined over $(\mathcal{K}, \mathcal{X}, \mathcal{K})$ corresponds to a tree of width $|\mathcal{X}|$. Again, the proof of Theorem 4.11 carries over with no essential changes. We leave the details as an exercise for the interested reader (Exercise 4.26 may be convenient here).

Comparing the CBC and cascade PRFs. Note that CBC uses a fixed key k for all applications of F while cascade uses a different key in each round. Since block ciphers are typically optimized to encrypt many blocks using the same key, the constant re-keying in cascade may result in worse performance than CBC. Hence, CBC is the more natural choice when using an off the shelf block cipher like AES.

An advantage of cascade is that there is no additive error term in Theorem 6.4. Consequently, the cascade construction remains secure even if the underlying PRF has a small domain $\mathcal{X}$. CBC, in contrast, is secure only when $\mathcal{X}$ is large. As a result, cascade can be used to convert a PRG into a PRF for large inputs while CBC cannot.

6.4.3 Extension attacks: CBC and cascade are insecure MACs

We show that the MACs derived from CBC and cascade are insecure. This will imply that CBC and cascade are not secure PRFs. All we showed in the previous section is that CBC and cascade are *prefix-free* secure PRFs.

Extension attack on cascade. Given $F^*(k, m)$ for some message m in $\mathcal{X}^{\leq \ell}$, anyone can compute

$$t' := F^*(k, \quad m \parallel m') \tag{6.16}$$

for any $m' \in \mathcal{X}^*$, without knowledge of k. Once $F^*(k, m)$ is known, anyone can continue evaluating the chain using blocks of the message m' and obtain t'. We refer to this as the **extension property** of cascade.

The extension property immediately implies that the MAC derived from F^* is terribly insecure. The forger can request the MAC on message m and then deduce the MAC on $m \parallel m'$ for any m' of its choice. It follows, by Theorem 6.2, that F^* is not a secure PRF.

An attack on CBC. We describe a simple MAC forger on the MAC derived from CBC. The forger works as follows:

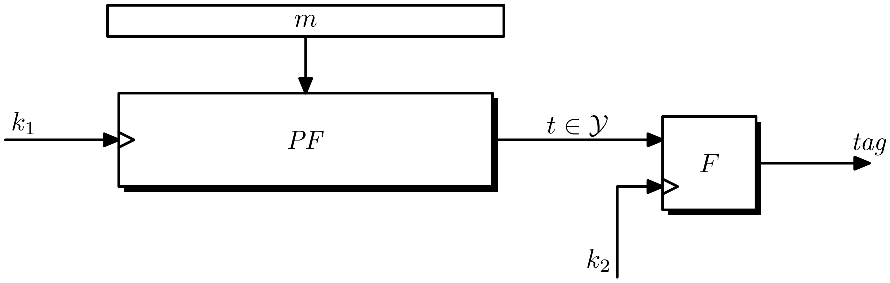


Figure 6.4: The encrypted PRF construction $EF(k, m)$

1. pick an arbitrary $a_1 \in \mathcal{X}$;
2. request the tag t on the one-block message (a_1);
3. define $a_2 := a_1 \oplus t$ and output t as a MAC forgery for the two-block message $(a_1, a_2) \in \mathcal{X}^2$.

Observe that $t = F(k, a_1)$ and $a_1 = F(k, a_1) \oplus a_2$. By definition of CBC we have:

$$F_{\text{CBC}}\big(k,\ (a_1, a_2)\ \big) = F\big(k,\ \ F(k,\ a_1) \oplus a_2\ \big) = F\big(k,\ a_1\big) = t.$$

Hence, $\big((a_1, a_2),\ t\big)$ is an existential forgery for the MAC derived from CBC. Consequently, F_{CBC} cannot be a secure PRF. Note that the attack on the cascade MAC is far more devastating than on the CBC MAC. But in any case, these attacks show that neither CBC nor cascade should be used directly as MACs.

6.5 From prefix-free secure PRF to fully secure PRF (method 1): encrypted PRF

We show how to convert the prefix-free secure PRFs F_{CBC} and F^* into secure PRFs, which will give us secure MACs for variable length inputs. More generally, we show how to convert a prefix-free secure PRF PF to a secure PRF. We present three methods:

- Encrypted PRF: encrypt the short output of PF with another PRF.
- Prefix-free encoding: encode the input to PF so that no input is a prefix of another.
- CMAC: a more efficient prefix-free encoding using randomization.

In this section we discuss the encrypted PRF method. The construction is straightforward. Let PF be a PRF mapping $\mathcal{X}^{\leq \ell}$ to $\mathcal{Y}$ and let F be a PRF mapping $\mathcal{Y}$ to $\mathcal{T}$. Define

$$EF\big((k_1, k_2),\ m\big) := F\big(k_2,\ PF(k_1,\ m)\big) \tag{6.17}$$

The construction is shown in Fig. 6.4.

We claim that when PF is either CBC or cascade then EF is a secure PRF. More generally, we show that EF is secure whenever PF is an *extendable PRF*, defined as follows:

Definition 6.4. *Let PF be a PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$. We say that PF is an* ***extendable PRF*** *if for all $k \in \mathcal{K}$, $x, y \in \mathcal{X}^{\leq \ell - 1}$, and $a \in \mathcal{X}$ we have:*

$$\text{if} \quad PF(k, x) = PF(k, y) \quad \text{then} \quad PF(k,\ x \parallel a) = PF(k,\ y \parallel a).$$

It is easy to see that both CBC and cascade are extendable PRFs. The next theorem shows that when PF is an extendable, prefix-free secure PRF then EF is a secure PRF.

Theorem 6.5. *Let PF be an extendable and prefix-free secure PRF defined over $(\mathcal{K}_1, \mathcal{X}^{\leq \ell+1}, \mathcal{Y})$, where $|\mathcal{Y}|$ is super-poly and ℓ is poly-bounded. Let F be a secure PRF defined over $(\mathcal{K}_2, \mathcal{Y}, \mathcal{T})$. Then EF, as defined in (6.17), is a secure PRF defined over $(\mathcal{K}_1 \times \mathcal{K}_2, \mathcal{X}^{\leq \ell}, \mathcal{T})$.*

In particular, for every PRF adversary $\mathcal{A}$ that attacks EF as in Attack Game 4.2, and issues at most Q queries, there exist a PRF adversary $\mathcal{B}_1$ attacking F as in Attack Game 4.2, and a prefix-free PRF adversary $\mathcal{B}_2$ attacking PF as in Attack Game 4.2, where $\mathcal{B}_1$ and $\mathcal{B}_2$ are elementary wrappers around $\mathcal{A}$, such that

$$\text{PRF}\mathsf{adv}[\mathcal{A}, EF] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_1, F] + \text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{B}_2, PF] + \frac{Q^2}{2|\mathcal{Y}|}. \tag{6.18}$$

We prove Theorem 6.5 in the next chapter (Section 7.3.1) after we develop the necessary tools. Note that to make EF a secure PRF on inputs of length up to ℓ, this theorem requires that PF is prefix-free secure on inputs of length $\ell + 1$.

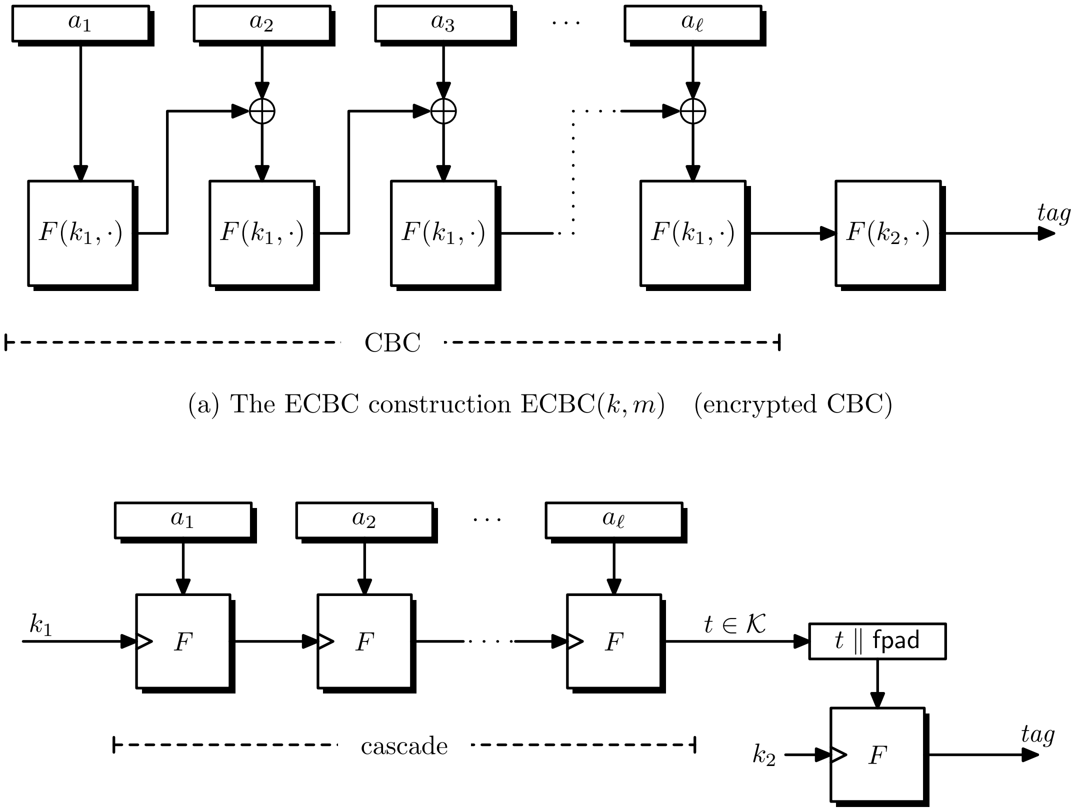
The bound in (6.18) is tight. Although not entirely necessary, let us assume that $\mathcal{Y} = \mathcal{T}$, that F is a block cipher, and that $|\mathcal{X}|$ is not too small. These assumptions will greatly simplify the argument. We exhibit an attack that breaks EF with constant probability after $Q \approx \sqrt{|\mathcal{Y}|}$ queries. Our attack will, in fact, break EF as a MAC. The adversary picks Q random inputs $x_1, \ldots, x_Q \in \mathcal{X}^2$ and queries its MAC challenger at all Q inputs to obtain $t_1, \ldots, t_Q \in \mathcal{T}$. By the birthday paradox (Corollary B.2), for any fixed key k_1, with constant probability there will be distinct indices i, j such that $x_i \neq x_j$ and $PF(k_1, x_i) = PF(k_1, x_j)$. On the one hand, if such a collision occurs, we will detect it, because $t_i = t_j$ for such a pair of indices. On the other hand, if $t_i = t_j$ for some pair of indices i, j, then our assumption that F is a block cipher guarantees that $PF(k_1, x_i) = PF(k_1, x_j)$. Now, assuming that $x_i \neq x_j$ and $PF(k_1, x_i) = PF(k_1, x_j)$, and since PF is extendable, we know that for all $a \in \mathcal{X}$, we have $PF\big(k_1, (x_i \parallel a)\big) = PF\big(k_1, (x_j \parallel a)\big)$. Therefore, our adversary can obtain the MAC tag t for $x_i \parallel a$, and this tag t will also be a valid tag for $x_j \parallel a$. This attack easily generalizes to show the necessity of the term $Q^2/(2|\mathcal{Y}|)$ in (6.18).

6.5.1 ECBC and NMAC: MACs for variable length inputs

Figures 6.5a and 6.5b show the result of applying the EF construction (6.17) to CBC and cascade.

6.5.1.1 The Encrypted-CBC PRF

Applying EF to CBC results in a classic PRF (and hence a MAC) called **encrypted-CBC** or **ECBC** for short. This MAC is standardized by ANSI (see Section 6.9) and is used in the banking industry. The ECBC PRF uses the same underlying PRF F for both CBC and the final encryption. Consequently, ECBC is defined over $(\mathcal{K}^2,\ \mathcal{X}^{\leq \ell},\ \mathcal{X})$.



(a) The ECBC construction ECBC(k, m) (encrypted CBC)

(b) The NMAC construction NMAC(k, m) (encrypted cascade)

Figure 6.5: Secure PRF constructions for variable length inputs

Theorem 6.6 (ECBC security). *Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$. Suppose $\mathcal{X}$ is super-poly, and let ℓ be a poly-bounded length parameter. Then* ECBC *is a secure PRF defined over $(\mathcal{K}^2, \mathcal{X}^{\leq \ell}, \mathcal{X})$.*

> *In particular, for every PRF adversary $\mathcal{A}$ that attacks* ECBC *as in Attack Game 4.2, and issues at most Q queries, there exist PRF adversaries $\mathcal{B}_1, \mathcal{B}_2$ that attack F as in Attack Game 4.2, and which are elementary wrappers around $\mathcal{A}$, such that*

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, \mathrm{ECBC}] \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}_1, F] + \mathrm{PRF}\mathsf{adv}[\mathcal{B}_2, F] + \frac{(Q(\ell+1))^2 + Q^2}{2|\mathcal{X}|}. \tag{6.19}$$

Proof. CBC is clearly extendable and is a prefix-free secure PRF by Theorem 6.3. Hence, if the underlying PRF F is secure, then ECBC is a secure PRF by Theorem 6.5. □

The argument given after Theorem 6.5 shows that there is an attacker that after $Q \approx \sqrt{|\mathcal{X}|}$ queries breaks this PRF with constant advantage. Recall that for 3DES we have $\mathcal{X} = \{0,1\}^{64}$. Hence, after about a billion queries (or more precisely, 2^{32} queries) an attacker can break the ECBC-3DES MAC with constant probability.

6.5.1.2 The NMAC PRF

Applying *EF* to cascade results in a PRF (and hence a MAC) called **Nested MAC** or **NMAC** for short. A variant of this MAC is standardized by the IETF (see Section 8.7.2) and is widely used in Internet protocols.

We wish to use the same underlying PRF F for the cascade construction and for the final encryption. Unfortunately, the output of cascade is in $\mathcal{K}$ while the message input to F is in $\mathcal{X}$. To solve this problem we need to embed the output of cascade into $\mathcal{X}$. More precisely, we assume that $|\mathcal{K}| \leq |\mathcal{X}|$ and that there is an efficiently computable one-to-one function g that maps $\mathcal{K}$ into $\mathcal{X}$. For example, suppose $\mathcal{K} := \{0,1\}^\kappa$ and $\mathcal{X} := \{0,1\}^n$ where $\kappa \leq n$. Define $g(t) := t \parallel \mathsf{fpad}$ where fpad is a fixed pad of length $n - \kappa$ bits. This fpad can be as simple as a string of 0s. With this translation, all of NMAC can be built from a single secure PRF F, as shown in Fig. 6.5b.

Theorem 6.7 (NMAC security). *Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{K})$, where $\mathcal{K}$ can be embedded into $\mathcal{X}$. Then* NMAC *is a secure PRF defined over $(\mathcal{K}^2, \mathcal{X}^{\leq \ell}, \mathcal{K})$.*

> *In particular, for every PRF adversary $\mathcal{A}$ that attacks* NMAC *as in Attack Game 4.2, and issues at most Q queries, there exist PRF adversaries $\mathcal{B}_1, \mathcal{B}_2$ that attack F as in Attack Game 4.2, and which are elementary wrappers around $\mathcal{A}$, such that*

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, \mathrm{NMAC}] \leq (Q(\ell+1)) \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}_1, F] + \mathrm{PRF}\mathsf{adv}[\mathcal{B}_2, F] + \frac{Q^2}{2|\mathcal{K}|}. \tag{6.20}$$

Proof. NMAC is clearly extendable and is a prefix-free secure PRF by Theorem 6.4. Hence, if the underlying PRF F is secure, then NMAC is a secure PRF by Theorem 6.5. □

ECBC **and** NMAC **are streaming MACs.** Both ECBC and NMAC can be used to authenticate variable size messages in $\mathcal{X}^{\leq \ell}$. Moreover, there is no need for the message length to be known ahead of time. A MAC that has this property is said to be a **streaming MAC**. This property enables applications to feed message blocks to the MAC one block at a time and at some arbitrary point decide that the message is complete. This is important for applications like streaming video, where the message length may not be known ahead of time.

In contrast, some MAC systems require that the message length be prepended to the message body (see Section 6.6). Such MACs are harder to use in practice since they require applications to determine the message length before starting the MAC calculations.

6.6 From prefix-free secure PRF to fully secure PRF (method 2): prefix-free encodings

Another approach to converting a prefix-free secure PRF into a secure PRF is to encode the input to the PRF so that no encoded input is a prefix of another. We use the following terminology:

- We say that a set $S \subseteq \mathcal{X}^{\leq \ell}$ is a **prefix-free set** if no element in S is a proper prefix of any other. For example, if (x_1, x_2, x_3) belongs to a prefix-free set S, then neither x_1 nor (x_1, x_2) are in S.
- Let $\mathcal{X}_{>0}^{\leq \ell}$ denote the set of all non-empty strings over $\mathcal{X}$ of length at most ℓ. We say that a function $pf : \mathcal{M} \to \mathcal{X}_{>0}^{\leq \ell}$ is a **prefix-free encoding** if pf is injective (i.e., one-to-one) and the image of pf is a prefix-free set.

Let PF be a prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$ and $pf : \mathcal{M} \to \mathcal{X}_{>0}^{\leq \ell}$ be a prefix-free encoding. Define the derived PRF F as

$$F(k, m) := PF(k, pf(m)).$$

Then F is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{Y})$. We obtain the following trivial theorem.

Theorem 6.8. *If PF is a prefix-free secure PRF and pf is a prefix-free encoding then F is a secure PRF.*

6.6.1 Prefix free encodings

To construct PRFs using Theorem 6.8 we describe two prefix-free encodings $pf : \mathcal{M} \to \mathcal{X}^{\leq \ell}$. We assume that $\mathcal{X} = \{0,1\}^n$ for some n.

Method 1: prepend length. Set $\mathcal{M} := \mathcal{X}^{\leq \ell - 1}$ and let $m = (a_1, \ldots, a_v) \in \mathcal{M}$. Define

$$pf(m) := (\langle v \rangle, a_1, \ldots, a_v) \quad \in \mathcal{X}_{>0}^{\leq \ell}$$

where $\langle v \rangle \in \mathcal{X}$ is the binary representation of v, the length of m. We assume that $\ell < 2^n$ so that the message length can be encoded as an n-bit binary string.

We argue that pf is a prefix-free encoding. Clearly pf is injective. To see that the image of pf is a prefix-free set let $pf(x)$ and $pf(y)$ be two elements in the image of pf. If $pf(x)$ and $pf(y)$ contain the same number of blocks, then neither is a proper prefix of the other. Otherwise, $pf(x)$ and $pf(y)$ contain a different number of blocks and must therefore differ in the first block. But then, again, neither is a proper prefix of the other. Hence, pf is a prefix-free encoding.

This prefix-free encoding is not often used in practice since the resulting MAC is not a streaming MAC: an application using this MAC must commit to the length of the message to MAC ahead of time. This is undesirable for streaming applications such as streaming video where the length of packets may not be known ahead of time.

Method 2: stop bits. Let $\bar{\mathcal{X}} := \{0,1\}^{n-1}$ and let $\mathcal{M} = \bar{\mathcal{X}}_{>0}^{\leq \ell}$. For $m = (a_1, \ldots, a_v) \in \mathcal{M}$, define

$$pf(m) := \big((a_1 \parallel 0),\ (a_2 \parallel 0),\ \ldots,\ (a_{v-1} \parallel 0),\ (a_v \parallel 1)\big) \quad \in \mathcal{X}_{>0}^{\leq \ell}$$

Clearly *pf* is injective. To see that the image of *pf* is a prefix-free set let $pf(x)$ and $pf(y)$ be two elements in the image of *pf*. Let v be the number of blocks in $pf(x)$. If $pf(y)$ contains v or fewer blocks then $pf(x)$ is not a proper prefix of $pf(y)$. If $pf(y)$ contains more than v blocks then block number v in $pf(y)$ ends in 0, but block number v in $pf(x)$ ends in 1. Hence, $pf(x)$ and $pf(y)$ differ in block v and therefore $pf(x)$ is not a proper prefix of $pf(y)$.

The MAC resulting from this prefix-free encoding is a streaming MAC. This encoding, however, increases the length of the message to MAC by v bits. When computing the MAC on a long message using either CBC or cascade, this encoding will result in additional evaluations of the underlying PRF (e.g. AES). In contrast, the encrypted PRF method of Section 6.5 only adds one additional application of the underlying PRF. For example, to MAC a megabyte message (2^{20} bytes) using ECBC-AES and *pf* one would need an additional 511 evaluations of AES beyond what is needed for the encrypted PRF method. In practice, things are even worse. Since computers prefer byte-aligned data, one would most likely need to append an entire byte to every block, rather than just a bit. Then to MAC a megabyte message using ECBC-AES and *pf* would result in 4096 additional evaluations of AES over the encrypted PRF method — an overhead of about 6%.

6.7 From prefix-free secure PRF to fully secure PRF (method 3): CMAC

Both prefix free encoding methods from the previous section are problematic. The first resulted in a non-streaming MAC. The second required more evaluations of the underlying PRF for long messages. We can do better by randomizing the prefix free encoding. We build a streaming secure PRF that introduces no overhead beyond the underlying prefix-free secure PRF. The resulting MACs, shown in Fig. 6.6, are superior to those obtained from encrypted PRFs and deterministic encodings. This approach is used in a NIST MAC standard called CMAC and described in Section 6.10.

First, we introduce some convenient notation:

Definition 6.5. *For two strings $x, y \in \mathcal{X}^{\leq \ell}$, let us write $x \sim y$ if x is a prefix of y or y is a prefix of x.*

Definition 6.6. *Let ϵ be a real number, with $0 \leq \epsilon \leq 1$. A **randomized ϵ-prefix-free** encoding is a function $rpf : \mathcal{K} \times \mathcal{M} \to \mathcal{X}_{>0}^{\leq \ell}$ such that for all $m_0, m_1 \in \mathcal{M}$ with $m_0 \neq m_1$, we have*

$$\Pr\big[rpf(k, m_0) \sim rpf(k, m_1)\big] \leq \epsilon,$$

where the probability is over the random choice of k in $\mathcal{K}$.

Note that the image of $rpf(k, \cdot)$ need not be a prefix-free set. However, without knowledge of k it is difficult to find messages $m_0, m_1 \in \mathcal{M}$ such that $rpf(k, m_0)$ is a proper prefix of $rpf(k, m_1)$ (or vice versa). The function $rpf(k, \cdot)$ need not even be injective.

A simple *rpf*. Let $\mathcal{K} := \mathcal{X}$ and $\mathcal{M} := \mathcal{X}_{>0}^{\leq \ell}$. Define

$$rpf(k,\ (a_1, \ldots, a_v)) := \big(a_1, \ldots, a_{v-1}, (a_v \oplus k)\big) \in \mathcal{X}_{>0}^{\leq \ell}$$

It is easy to see that *rpf* is a randomized $(1/|\mathcal{X}|)$-prefix-free encoding. Let $m_0, m_1 \in \mathcal{M}$ with $m_0 \neq m_1$. Suppose that $|m_0| = |m_1|$. Then it is clear that for all choices of k, $rpf(k, m_0)$ and $rpf(k, m_1)$ are distinct strings of the same length, and so neither is a prefix of the other. Next, suppose that $|m_0| < |m_1|$. If $v := |rpf(k, m_0)|$, then clearly $rpf(k, m_0)$ is a proper prefix of $rpf(k, m_1)$ if and only if block number v satisfies

$$m_{0,v} \oplus k = m_{1,v}.$$

But this holds with probability $1/|\mathcal{X}|$ over the random choice of k, as required. Finally, the case $|m_0| > |m_1|$ is handled by a symmetric argument.

Using *rpf*. Let *PF* be a prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$ and $rpf : \mathcal{K}_1 \times \mathcal{M} \to \mathcal{X}^{\leq \ell}_{>0}$ be a randomized prefix-free encoding. Define the derived PRF F as

$$F\big((k, k_1),\ m\big) := PF\big(k, rpf(k_1, m)\big). \tag{6.21}$$

Then F is defined over $(\mathcal{K} \times \mathcal{K}_1, \mathcal{M}, \mathcal{Y})$. We obtain the following theorem, which is analogous to Theorem 6.8.

Theorem 6.9. *If PF is a prefix-free secure PRF, ϵ is negligible, and rpf a randomized ϵ-prefix-free encoding, then F defined in (6.21) is a secure PRF.*

> *In particular, for every PRF adversary $\mathcal{A}$ that attacks F as in Attack Game 4.2, and issues at most Q queries, there exist prefix-free PRF adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$ that attack PF as in Attack Game 4.2, where $\mathcal{B}_1$ and $\mathcal{B}_2$ are elementary wrappers around $\mathcal{A}$, such that*

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F] \leq \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_1, PF] + \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_2, PF] + Q^2\epsilon/2. \tag{6.22}$$

Proof idea. If the adversary's set of inputs to F give rise to a prefix-free set of inputs to *PF*, then the adversary sees just some random looking outputs. Moreover, if the adversary sees random outputs, it obtains no information about the *rpf* key k_1, which ensures that the set of inputs to *PF* is indeed prefix free (with overwhelming probability). Unfortunately, this argument is circular. However, we will see in the detailed proof how to break this circularity. □

Proof. Without loss of generality, we assume that $\mathcal{A}$ never issues the same query twice. We structure the proof as a sequence of three games. For $j = 0, 1, 2$, we let W_j be the event that $\mathcal{A}$ outputs 1 at the end of Game j.

Game 0. The challenger in Experiment 0 of the PRF Attack Game 4.2 with respect to F works as follows.

```
$k \xleftarrow{R} \mathcal{K}$,    $k_1 \xleftarrow{R} \mathcal{K}_1$
upon receiving a signing query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) do:
    $x_i \leftarrow rpf(k_1, m_i) \in \mathcal{X}^{\leq \ell}_{>0}$
    $y_i \leftarrow PF(k, x_i)$
    send $y_i$ to $\mathcal{A}$
```

Game 1. We change the challenger in Game 0 to ensure that all queries to *PF* are prefix free. Recall the notation $x \sim y$, which means that x is a prefix of y or y is a prefix of x.

```
     $k \xleftarrow{\text{R}} \mathcal{K}$,    $k_1 \xleftarrow{\text{R}} \mathcal{K}_1$,    $r_1, \ldots, r_Q \xleftarrow{\text{R}} \mathcal{Y}$

     upon receiving a signing query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) do:
          $x_i \leftarrow rpf(k_1, m_i) \in \mathcal{X}_{>0}^{\le \ell}$
(1)       if $x_i \sim x_j$ for some $j < i$
               then $y_i \leftarrow r_i$
(2)            else  $y_i \leftarrow PF(k, x_i)$
          send $y_i$ to $\mathcal{A}$
```

Let Z_1 be the event that the condition on line (1) holds at some point during Game 1. Clearly, Games 1 and 2 proceed identically until event Z_1 occurs; in particular, $W_0 \wedge \bar{Z}_1$ occurs if and only if $W_1 \wedge \bar{Z}_1$ occurs. Applying the Difference Lemma (Theorem 4.7), we obtain

$$\big|\Pr[W_1] - \Pr[W_0]\big| \le \Pr[Z_1]. \tag{6.23}$$

Unfortunately, we are not quite in a position to bound $\Pr[Z_1]$ at this point. At this stage in the analysis, we cannot say that the evaluations of PF at line (2) do not leak some information about k_1 that could help $\mathcal{A}$ make Z_1 happen. This is the circularity problem we alluded to above. To overcome this problem, we will delay the analysis of Z_1 to the next game.

Game 2. Now we play the usual “PRF card,” replacing the function $PF(k, \cdot)$ by a truly random function. This is justified, since by construction, in Game 1, the set of inputs to $PF(k, \cdot)$ is prefix-free. To implement this change, we may simply replace the line marked (2) by

```
(2)            else  $y_i \leftarrow r_i$
```

After making this change, we see that y_i gets assigned the random value r_i, regardless of whether the condition on line (1) holds or not.

Now, let Z_2 be the event that the condition on line (1) holds at some point during Game 2. It is not hard to see that

$$|\Pr[Z_1] - \Pr[Z_2]| \le \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_1, F] \tag{6.24}$$

and

$$|\Pr[W_1] - \Pr[W_2]| \le \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_2, F] \tag{6.25}$$

for efficient prefix-free PRF adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$. These two adversaries are basically the same, except that $\mathcal{B}_1$ outputs 1 if the condition on line (1) holds, while $\mathcal{B}_2$ ouputs whatever $\mathcal{A}$ outputs.

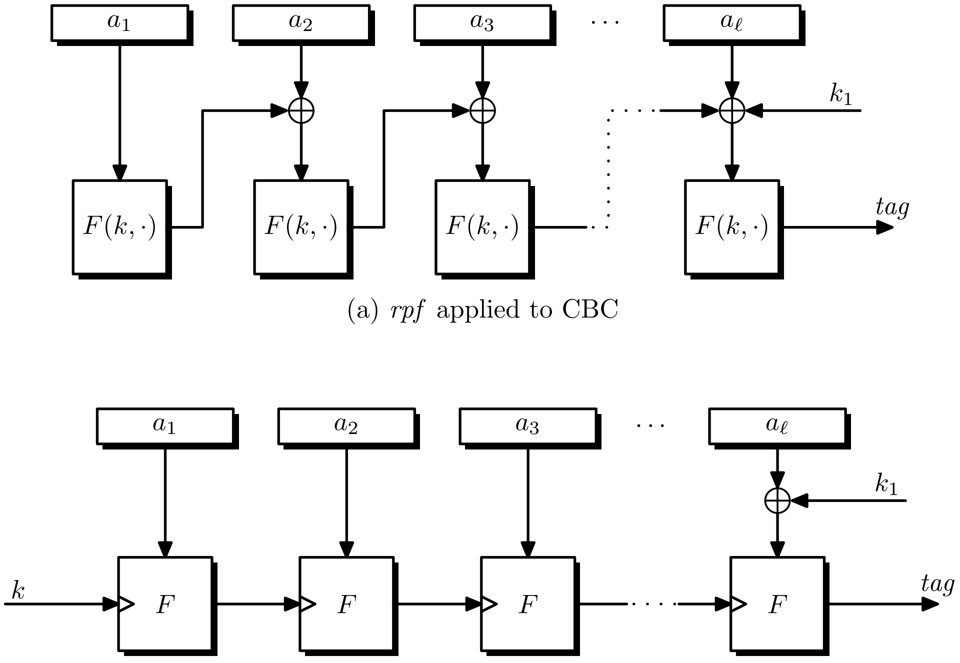
Moreover, in Game 2, the value of k_1 is clearly independent of $\mathcal{A}$'s queries, and so by making use of the ϵ-prefix-free property of rpf, and the union bound we have

$$\Pr[Z_2] \le Q^2\epsilon/2 \tag{6.26}$$

Finally, Game 2 perfectly emulates for $\mathcal{A}$ a random function in $\mathrm{Funs}[\mathcal{M}, \mathcal{Y}]$. Game 2 is therefore identical to Experiment 1 of the PRF Attack Game 4.2 with respect to F, and hence

$$|\Pr[W_0] - \Pr[W_2]| = \mathrm{PRF}\mathsf{adv}[\mathcal{A}, F]. \tag{6.27}$$

Now combining (6.23)–(6.27) proves the theorem. □



(a) *rpf* applied to CBC

(b) *rpf* applied to cascade

Figure 6.6: Secure PRFs using random prefix-free encodings

6.8 Converting a block-wise PRF to bit-wise PRF

So far we constructed a number of PRFs for variable length inputs in $\mathcal{X}^{\leq \ell}$. Typically $\mathcal{X} = \{0,1\}^n$ where n is the block size of the underlying PRF from which CBC or cascade are built (e.g., $n = 128$ for AES). All our MACs so far are designed to authenticate messages whose length is a multiple of n bits.

In this section we show how to convert these PRFs into PRFs for messages of arbitrary bit length. That is, given a PRF for messages in $\mathcal{X}^{\leq \ell}$ we construct a PRF for messages in $\{0,1\}^{\leq n\ell}$.

Let F be a PRF taking inputs in $\mathcal{X}^{\leq \ell+1}$. Let $inj : \{0,1\}^{\leq n\ell} \to \mathcal{X}^{\leq \ell+1}$ be an injective (i.e., one-to-one) function. Define the derived PRF F_{bit} as

$$F_{\text{bit}}(k, x) := F(k,\ inj(x)).$$

Then we obtain the following trivial theorem.

Theorem 6.10. *If F is a secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell+1}, \mathcal{Y})$ then F_{bit} is a secure PRF defined over $(\mathcal{K},\ \{0,1\}^{\leq n\ell},\ \mathcal{Y})$.*

An injective function. For $\mathcal{X} := \{0,1\}^n$, a standard example of an injective inj from $\{0,1\}^{\leq n\ell}$ to $\mathcal{X}^{\leq \ell+1}$ works as follows. If the input message length is not a multiple of n then inj appends $100\ldots00$ to pad the message so its length is the next multiple of n. If the given message length is a multiple of n then inj appends an entire n-bit block $(1 \parallel 0^{n-1})$. Fig. 6.7 describes this in a picture. More precisely, the function works as follows:

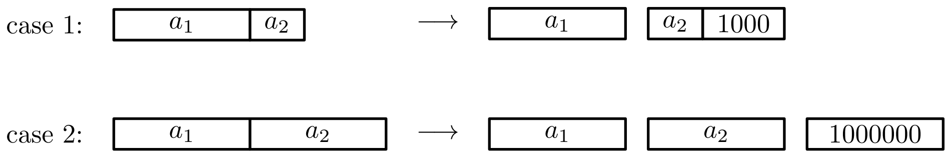


Figure 6.7: An injective function $inj : \{0,1\}^{\leq n\ell} \to \mathcal{X}^{\leq \ell+1}$

```
input: $m \in \{0,1\}^{\leq n\ell}$
$u \leftarrow |m| \bmod n, \quad m' \leftarrow m \parallel 1 \parallel 0^{n-u-1}$
output $m'$ as a sequence of $n$-bit message blocks
```

To see that *inj* is injective we show that it is invertible. Given $y \leftarrow inj(m)$ scan y from right to left and remove all the 0s until and including the first 1. The remaining string is m.

A common mistake is to pad the given message to a multiple of a block size using an all-0 pad. This pad is not injective and results in an insecure MAC: for any message m whose length is not a multiple of the block length, the MAC on m is also a valid MAC for $m \parallel 0$. Consequently, the MAC is vulnerable to existential forgery.

Injective functions must expand. When we feed an n-bit single block message into *inj*, the function adds a "dummy" block and outputs a two-block message. This is unfortunate for applications that MAC many single block messages. When using CBC or cascade, the dummy block forces the signer and verifier to evaluate the underlying PRF twice for each message, even though all messages are one block long. Consequently, *inj* forces all parties to work twice as hard as necessary.

It is natural to look for injective functions from $\{0,1\}^{\leq n\ell}$ to $\mathcal{X}^{\leq \ell}$ that never add dummy blocks. Unfortunately, there are no such functions simply because the set $\{0,1\}^{\leq n\ell}$ is larger than the set $\mathcal{X}^{\leq \ell}$. Hence, all injective functions must occasionally add a "dummy" block to the output.

The CMAC construction described in Section 6.10 provides an elegant solution to this problem. CMAC avoids adding dummy blocks by using a *randomized* injective function.

6.9 Case study: ANSI CBC-MAC

When building a MAC from a PRF, implementors often shorten the final tag by only outputting the w most significant bits of the PRF output. Exercise 4.4 shows that truncating a secure PRF has no effect on its security as a PRF. Truncation, however, affects the derived MAC. Theorem 6.2 shows that the smaller w is, the less secure the MAC becomes. In particular, the theorem adds a $1/2^w$ error in the concrete security bounds.

Two ANSI standards (ANSI X9.9 and ANSI X9.19) and two ISO standards (ISO 8731-1 and ISO/IEC 9797) specify variants of ECBC for message authentication using DES as the underlying PRF. These standards truncate the final 64-bit output of the ECBC-DES and use only the leftmost w bits of the output, where $w = 32, 48$, or 64 bits. This reduces the tag length at the cost of reduced security.

Both ANSI CBC-MAC standards specify a padding scheme to be used for messages whose length is not a multiple of the DES or AES block size. The padding scheme is identical to the

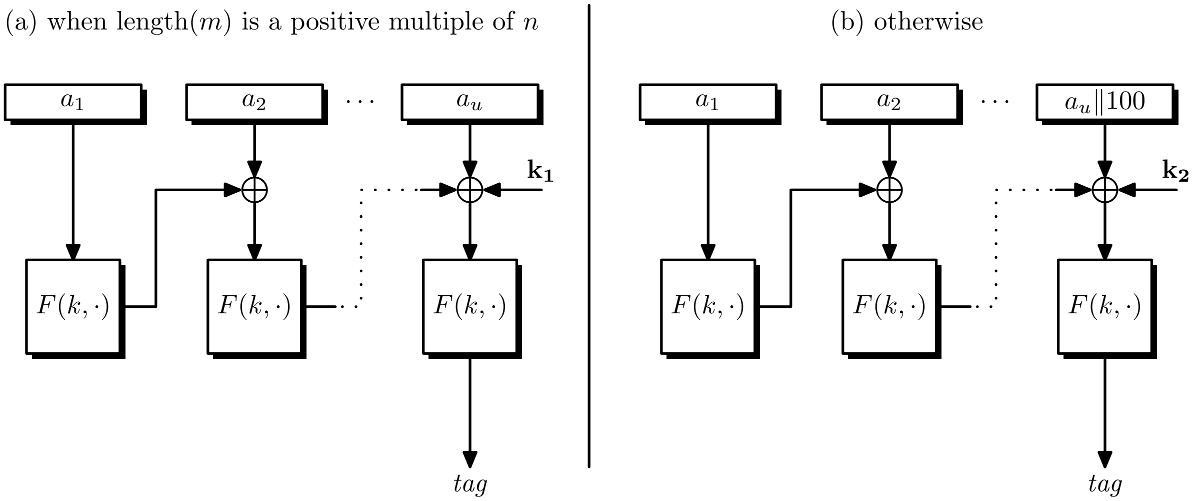


Figure 6.8: CMAC signing algorithm

function *inj* described in Section 6.8. The same padding scheme is used when signing a message and when verifying a message-tag pair.

6.10 Case study: CMAC

Cipher-based MAC — CMAC — is a variant of ECBC adopted by the National Institute of Standards (NIST) in 2005. It is based on a proposal due to Black and Rogaway and an extension due to Iwata and Kurosawa. CMAC improves over ECBC used in the ANSI standard in two ways. First, CMAC uses a randomized prefix-free encoding to convert a prefix-free secure PRF to a secure PRF. This saves the final encryption used in ECBC. Second, CMAC uses a "two key" method to avoid appending a dummy message block when the input message length is a multiple of the underlying PRF block size.

CMAC is the best approach to building a bit-wise secure PRF from the CBC prefix-free secure PRF. It should be used in place of the ANSI method. In Exercise 6.14 we show that the CMAC construction applies equally well to cascade.

The CMAC bit-wise PRF. The CMAC algorithm consists of two steps. First, a sub-key generation algorithm is used to derive three keys k_0, k_1, k_2 from the MAC key k. Then the three keys k_0, k_1, k_2 are used to compute the MAC.

Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$ where $\mathcal{X} = \{0,1\}^n$. The NIST standard uses AES as the PRF F. The CMAC signing algorithm is given in Table 6.1 and is illustrated in Fig. 6.8. The figure on the left is used when the message length is a multiple of the block size n. The figure on the right is used otherwise. The standard allows for truncating the final output to w bits by only outputting the w most significant bits of the final value t.

Security. The CMAC algorithm described in Fig. 6.8 can be analyzed using the randomized prefix-free encoding paradigm. In effect, CMAC converts the CBC prefix-free secure PRF directly

```
      input: Key $k \in \mathcal{K}$ and $m \in \{0,1\}^*$
      output: tag $t \in \{0,1\}^w$ for some $w \leq n$

      Setup:
          Run a sub-key generation algorithm
                  to generate keys $k_0, k_1, k_2 \in \mathcal{X}$ from $k \in \mathcal{K}$
          $\ell \leftarrow \text{length}(m)$
          $u \leftarrow \max(1, \lceil \ell/n \rceil)$
          Break $m$ into consecutive $n$-bit blocks so that
                  $m = a_1 \parallel a_2 \parallel \cdots \parallel a_{u-1} \parallel a_u^*$   where   $a_1, \ldots, a_{u-1} \in \{0,1\}^n$.
(*)       If $\text{length}(a_u^*) = n$
              then $a_u = k_1 \oplus a_u^*$
              else $a_u = k_2 \oplus (a_u^* \parallel 1 \parallel 0^j)$ where $j = nu - \ell - 1$

      CBC:
          $t \leftarrow 0^n$
          for $i \leftarrow 1$ to $u$ do:
              $t \leftarrow F(k_0,\ \ t \oplus a_i)$
          Output $t[0 \ldots w-1]$   // Output $w$ most significant bits of $t$.
```

Table 6.1: The CMAC signing algorithm

into a *bit-wise* secure PRF using a randomized prefix-free encoding $rpf : \mathcal{K} \times \mathcal{M} \to \mathcal{X}_{>0}^{\leq \ell}$ where $\mathcal{K} := \mathcal{X}^2$ and $\mathcal{M} := \{0,1\}^{\leq n\ell}$. The encoding *rpf* is defined as follows:

```
input: $m \in \mathcal{M}$ and $(k_1, k_2) \in \mathcal{X}^2$
if $|m|$ is not a positive multiple of $n$ then
    $u \leftarrow |m| \bmod n$
partition $m$ into a sequence of bit strings $a_1, \ldots, a_v \in \mathcal{X}$,
    so that $m = a_1 \parallel \cdots \parallel a_v$ and $a_1, \ldots, a_{v-1}$ are $n$-bit strings
if $|m|$ is a positive multiple of $n$
    then output $\big(a_1,\ \ldots,\ a_{v-1},\ (a_v \oplus k_1)\big)$
    else  output $\big(a_1,\ \ldots,\ a_{v-1},\ ((a_v \parallel 1 \parallel 0^{n-u-1}) \oplus k_2)\big)$
```

The argument that *rpf* is a randomized 2^{-n}-prefix-free encoding is similar to the one is Section 6.7. Hence, CMAC fits the randomized prefix-free encoding paradigm and its security follows from Theorem 6.9. The keys k_1, k_2 are used to resolve collisions between a message whose length is a positive multiple of n and a message that has been padded to make it a positive multiple of n. This is essential for the analysis of the CMAC *rpf*.

Sub-key generation. The sub-key generation algorithm generates the keys (k_0, k_1, k_2) from k. It uses a fixed mask string R_n that depends on the block size of F. For example, for a 128-bit block size, the standard specifies $R_{128} := 0^{120}10000111$. For a bit string X we denote by $X << 1$ the bit string that results from discarding the leftmost bit X and appending a 0-bit on the right. The sub-key generation algorithm works as follows:

```
input: key $k \in \mathcal{K}$
output: keys $k_0, k_1, k_2 \in \mathcal{X}$

        $k_0 \leftarrow k$
        $L \leftarrow F(k, 0^n)$
(1)     if $\text{msb}(L) = 0$ then $k_1 \leftarrow (L << 1)$ else $k_1 \leftarrow (L << 1) \oplus R_n$
(2)     if $\text{msb}(k_1) = 0$ then $k_2 \leftarrow (k_1 << 1)$ else $k_2 \leftarrow (k_1 << 1) \oplus R_n$
        output $k_0, k_1, k_2$.
```

where $\text{msb}(L)$ refers to the most significant bit of L. The lines marked (1) and (2) may look a bit mysterious, but in effect, they simply multiply L by X and by X^2 (respectively) in the finite field $\text{GF}(2^n)$. Here we are treating the elements of $\text{GF}(2^n)$ as polynomials in $\mathbb{F}_2[X]$ modulo a fixed polynomial $g(X)$. For a 128-bit block size, the defining polynomial $g(X)$ that corresponds to R_{128} is $g(X) := X^{128} + X^7 + X^2 + X + 1$. Exercise 6.16 explores some insecure variants of sub-key generation.

The three keys (k_0, k_1, k_2) output by the sub-key generation algorithm can be used for authenticating multiple messages. Hence, its running time is amortized across many messages.

Clearly the keys k_0, k_1, and k_2 are not independent. If they were, or if they were derived as, say, $k_i := F(k, \alpha_i)$ for constants $\alpha_0, \alpha_1, \alpha_2$, the security of CMAC would follow directly from the arguments made here and our general framework. Nevertheless, a more intricate analysis allows one to prove that CMAC is indeed secure [93].

6.11 PMAC: a parallel MAC

The MACs we developed so far, ECBC, CMAC, and NMAC, are inherently sequential: block number i cannot be processed before block number $i-1$ is finished. This makes it difficult to exploit hardware parallelism or pipelining to speed up MAC generation and verification. In this section we construct a secure MAC that is well suited for a parallel architecture. The best construction is called PMAC. We present PMAC_0 which is a little easier to describe.

Let F_1 be a PRF defined over $(\mathcal{K}_1, \mathbb{Z}_p, \mathcal{Y})$, where p is a prime and $\mathcal{Y} := \{0,1\}^n$. Let F_2 be a PRF defined over $(\mathcal{K}_2, \mathcal{Y}, \mathcal{Z})$.

We build a new PRF, called PMAC_0, which takes as input a key and a message in $\mathbb{Z}_p^{\leq \ell}$ for some ℓ. It outputs a value in $\mathcal{Z}$. The PMAC_0 construction works as follows:

```
input: $m = (a_1, \ldots, a_v) \in \mathbb{Z}_p^v$ for some $0 \leq v \leq \ell$, and
           key $\vec{k} = (k, k_1, k_2)$ where $k \in \mathbb{Z}_p$, $k_1 \in \mathcal{K}_1$, and $k_2 \in \mathcal{K}_2$
output: tag in $\mathcal{Z}$

$\text{PMAC}_0(\vec{k}, m)$:
    $t \leftarrow 0^n \in \mathcal{Y}$,   $mask \leftarrow 0 \in \mathbb{Z}_p$
    for $i \leftarrow 1$ to $v$ do:
        $mask \leftarrow mask + k$        //  $mask = i \cdot k \in \mathbb{Z}_p$
        $r \leftarrow a_i + mask$         //  $r = a_i + i \cdot k \in \mathbb{Z}_p$
        $t \leftarrow t \oplus F_1(k_1, r)$
    output $F_2(k_2,\ t)$
```

The main loop adds the masks $k, 2k, 3k, \ldots$ to message blocks prior to evaluating the PRF F_1. On a sequential machine this requires two additions modulo p per iteration. On a parallel machine

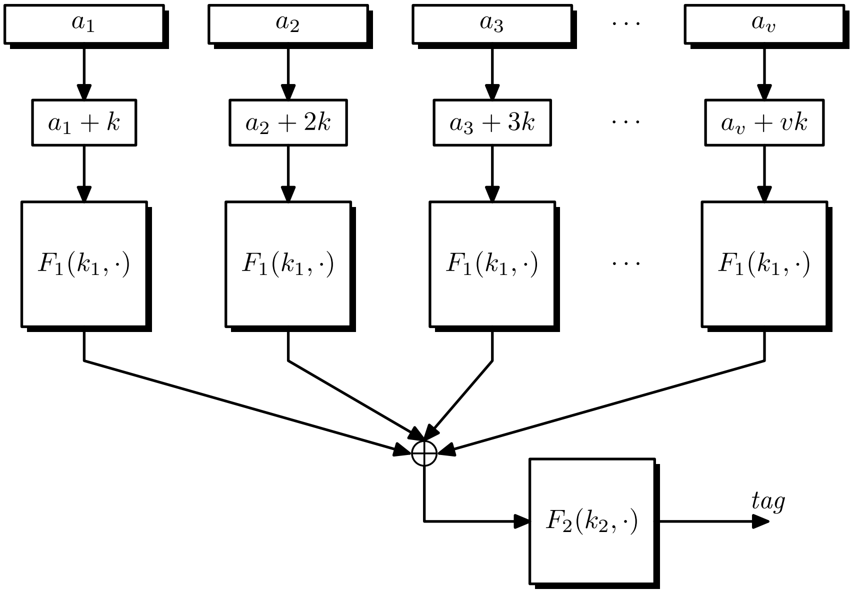


Figure 6.9: PMAC$_0$ construction

each processor can independently compute $a_i + ik$ and then apply F_1. See Fig. 6.9.

PMAC$_0$ is a secure PRF and hence gives a secure MAC for large messages. The proof will follow easily from Theorem 7.7 developed in the next chapter. For now we state the theorem and delay its proof to Section 7.3.3.

Theorem 6.11. *If F_1 and F_2 are secure PRFs, and $|\mathcal{Y}|$ and the prime p are super-poly, then* PMAC$_0$ *is a secure PRF for any poly-bounded ℓ.*

> *In particular, for every PRF adversary $\mathcal{A}$ that attacks* PMAC$_0$ *as in Attack Game 4.2, and issues at most Q queries, there exist PRF adversaries $\mathcal{B}_1$ and $\mathcal{B}_2$, which are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, \mathrm{PMAC}_0] \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}_1, F_1] + \mathrm{PRF}\mathsf{adv}[\mathcal{B}_2, F_2] + \frac{Q^2}{2|\mathcal{Y}|} + \frac{Q^2\ell^2}{2p}. \tag{6.28}$$

When using PMAC$_0$, the input message must be partitioned into blocks, where each block is an element of $\mathbb{Z}_p$. In practice, that is inconvenient. It is much easier to break the message into blocks, where each block is an n-bit string in $\{0,1\}^n$, for some n. A better parallel MAC construction, presented next, does exactly that by using the finite field GF(2^n) instead of $\mathbb{Z}_p$. This is a good illustration for why GF(2^n) is so useful in cryptography. We often need to work in a field for security reasons, but a prime finite field like $\mathbb{Z}_p$ is inconvenient to use in practice. Instead, we use GF(2^n) where arithmetic operations are much faster. GF(2^n) also lets us naturally operate on n-bit blocks.

PMAC: better than PMAC$_0$**.** Although PMAC$_0$ is well suited for a parallel architecture, there is room for improvement. Better implementations of the PMAC$_0$ approach are available. Examples include PMAC [23] and XECB [73], both of which are parallelizable. PMAC, for example, provides the following improvements over PMAC_0:

- PMAC uses arithmetic in the finite field $\text{GF}(2^n)$ instead of in $\mathbb{Z}_p$. Elements of $\text{GF}(2^n)$ can be represented as n-bit strings, and addition in $\text{GF}(2^n)$ is just a bit-wise XOR. Because of this, PMAC just uses $F_1 = F_2 = F$, where F is a PRF defined over $(\mathcal{K}, \mathcal{Y}, \mathcal{Y})$, and the input space of PMAC consists of sequences of elements of $\mathcal{Y} = \{0,1\}^n$, rather than elements of $\mathbb{Z}_p$.

- The PMAC mask for block i is defined as $\gamma_i \cdot k$ where $\gamma_1, \gamma_2, \ldots$ are fixed constants in $\text{GF}(2^n)$ and multiplication is defined in $\text{GF}(2^n)$. The γ_i's are specially chosen so that computing $\gamma_{i+1} \cdot k$ from $\gamma_i \cdot k$ is very cheap.

- PMAC derives the key k as $k \leftarrow F(k_1, 0^n)$ and sets $k_2 \leftarrow k_1$. Hence PMAC uses a shorter secret key than PMAC_0.

- PMAC uses a trick to save one application of F.

- PMAC uses a variant of the CMAC *rpf* to provide a bit-wise PRF.

The end result is that PMAC is as efficient as ECBC and NMAC on a sequential machine, but has much better performance on a parallel or pipelined architecture. PMAC is the best PRF construction in this chapter; it works well on a variety of computer architectures and is efficient for both long and short messages.

PMAC_0 **is incremental.** Suppose Bob computes the tag t for some long message m. Some time later he changes one block in m and wants to recompute the tag of this new message m'. When using CBC-MAC the tag t is of no help — Bob must recompute the tag for m' from scratch. With PMAC_0 we can do much better. Suppose the PRF F_2 used in the construction of PMAC_0 is the encryption algorithm of a block cipher such as AES, and let D be the corresponding decryption algorithm. Let m' be the result of changing block number i of m from a_i to a_i'. Then the tag $t' := \text{PMAC}_0(k, m')$ for m' can be easily derived from the tag $t := \text{PMAC}_0(k, m)$ for m as follows:

$$\begin{aligned}
&t_1 \leftarrow D(k_2, t)\\
&t_2 \leftarrow t_1 \;\oplus\; F_1(k_1,\; a_i + ik) \;\oplus\; F_1(k_1,\; a_i' + ik)\\
&t' \leftarrow F_2(k_2, t_2)
\end{aligned}$$

Hence, given the tag on some long message m (as well as the MAC secret key) it is easy to derive tags for local edits of m. MACs that have this property are said to be **incremental**. We just showed that the PMAC_0, implemented using a block cipher, is incremental.

6.12 A fun application: searching on encrypted data

To be written.

6.13 Notes

Citations to the literature to be added.

6.14 Exercises

6.1 (The 802.11b insecure MAC). Consider the following MAC (a variant of this was used for WiFi encryption in 802.11b WEP). Let F be a PRF defined over $(\mathcal{K}, \mathcal{R}, \mathcal{X})$ where $\mathcal{X} := \{0,1\}^{32}$. Let CRC32 be a simple and popular error-detecting code meant to detect random errors; CRC32 is a function that takes as input $m \in \{0,1\}^{\leq \ell}$ and outputs a 32-bit string. Define the following MAC system (S, V):

$$S(k,m) := \{ \; r \xleftarrow{\text{R}} \mathcal{R}, \; t \leftarrow F(k,r) \oplus \text{CRC32}(m), \; \text{output } (r,t) \; \}$$
$$V(k,m,(r,t)) := \{ \; \textsf{accept} \text{ if } t = F(k,r) \oplus \text{CRC32}(m) \text{ and } \textsf{reject} \text{ otherwise} \}$$

Show that this MAC system is insecure.

6.2 (Tighter bounds with verification queries). Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$, and let $\mathcal{I}$ be the MAC system derived from F, as discussed in Section 6.3. Let $\mathcal{A}$ be an adversary that attacks $\mathcal{I}$ as in Attack Game 6.2, and which makes at most Q_v verification queries and at most Q_s signing queries. Theorem 6.1 says that there exists a Q_s-query MAC adversary $\mathcal{B}$ that attacks $\mathcal{I}$ as in Attack Game 6.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that $\text{MAC}^{\text{vq}}\textsf{adv}[\mathcal{A}, \mathcal{I}] \leq \text{MAC}\textsf{adv}[\mathcal{B}, \mathcal{I}] \cdot Q_\text{v}$. Theorem 6.2 says that there exists a $(Q_\text{s}+1)$-query PRF adversary $\mathcal{B}'$ that attacks F as in Attack Game 4.2, where $\mathcal{B}'$ is an elementary wrapper around $\mathcal{B}$, such that $\text{MAC}\textsf{adv}[\mathcal{B}, \mathcal{I}] \leq \text{PRF}\textsf{adv}[\mathcal{B}', F] + 1/|\mathcal{Y}|$. Putting these two statements together, we get

$$\text{MAC}^{\text{vq}}\textsf{adv}[\mathcal{A}, \mathcal{I}] \leq (\text{PRF}\textsf{adv}[\mathcal{B}', F] + 1/|\mathcal{Y}|) \cdot Q_\text{v}$$

This bound is not the best possible. Give a direct analysis that shows that there exists a $(Q_\text{s}+Q_\text{v})$-query PRF adversary $\mathcal{B}''$, where $\mathcal{B}''$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{MAC}^{\text{vq}}\textsf{adv}[\mathcal{A}, \mathcal{I}] \leq \text{PRF}\textsf{adv}[\mathcal{B}'', F] + Q_\text{v}/|\mathcal{Y}|.$$

6.3 (Multi-key MAC security). Just as we did for semantically secure encryption in Exercise 5.2, we can extend the definition of a secure MAC from the single-key setting to the multi-key setting. In this exercise, you will show that security in the single-key setting implies security in the multi-key setting.

(a) Show how to generalize Attack Game 6.2 so that an attacker can submit both signing queries and verification queries with respect to several MAC keys $k_1, \ldots, k_Q$. At the beginning of the game the adversary outputs a number Q indicating the number of keys it wants to attack and the challenger chooses Q random keys $k_1, \ldots, k_Q$. Subsequently, every query from the attacker includes an index $j \in \{1, \ldots, Q\}$. The challenger uses the key k_j to respond to the query.

(b) Show that every efficient adversary $\mathcal{A}$ that wins your multi-key MAC attack game with probability ϵ can be transformed into an efficient adversary $\mathcal{B}$ that wins Attack Game 6.2 with probability ϵ/Q.

Hint: This is *not* done using a hybrid argument, but rather a "guessing" argument, somewhat analogous to that used in the proof of Theorem 6.1. Adversary $\mathcal{B}$ plays the role of challenger to adversary $\mathcal{A}$. Once $\mathcal{A}$ outputs a number Q, $\mathcal{B}$ chooses Q random keys $k_1, \ldots, k_Q$ and a random index $\omega \in \{1, \ldots, Q\}$. When $\mathcal{A}$ issues a query for key number $j \neq \omega$, adversary $\mathcal{B}$

uses its key k_j to answer the query. When $\mathcal{A}$ issues a query for the key k_ω, adversary $\mathcal{B}$ answers the query by querying its MAC challenger. If $\mathcal{A}$ outputs a forgery under key k_ω then $\mathcal{B}$ wins the MAC forgery game. Show that $\mathcal{B}$ wins Attack Game 6.2 with probability ϵ/Q. We call this style of argument "plug-and-pray:" $\mathcal{B}$ "plugs" the key he is challenged on at a random index ω and "prays" that $\mathcal{A}$ uses the key at index ω to form his existential forgery.

6.4 (Multicast MACs). Consider a scenario in which Alice wants to broadcast the same message to n users, $U_1, \ldots, U_n$. She wants the users to be able to authenticate that the message came from her, but she is not concerned about message secrecy. More generally, Alice may wish to broadcast a series of messages, but for this exercise, let us focus on just a single message.

(a) In the most trivial solution, Alice shares a MAC key k_i with each user U_i. When she broadcasts a message m, she appends tags $t_1, \ldots, t_n$ to the message, where t_i is a valid tag for m under key k_i. Using its shared key k_i, every user U_i can verify m's authenticity by verifying that t_i is a valid tag for m under k_i.

Assuming the MAC is secure, show that this broadcast authentication scheme is secure *even if users collude*. For example, users $U_1, \ldots, U_{n-1}$ may collude, sharing their keys $k_1, \ldots, k_{n-1}$ among each other, to try to make user U_n accept a message that is not authentic.

(b) While the above broadcast authentication scheme is secure, even in the presence of collusions, it is not very efficient; the number of keys and tags grows linearly in n.

Here is a more efficient scheme, but with a weaker security guarantee. We illustrate it with $n = 6$. The goal is to get by with $\ell < 6$ keys and tags. We will use just $\ell = 4$ keys, $k_1, \ldots, k_4$. Alice stores all four of these keys. There are $6 = \binom{4}{2}$ subsets of $\{1, \ldots, 4\}$ of size 2. Let us number these subsets $J_1, \ldots, J_6$. For each user U_i, if $J_i = \{v, w\}$, then this user stores keys k_v and k_w.

When Alice broadcasts a message m, she appends tags $t_1, \ldots, t_4$, corresponding to keys $k_1, \ldots, k_4$. User U_i verifies tags t_v and t_w, using its keys k_v, k_w, where $J_i = \{v, w\}$ as above.

Assuming the MAC is secure, show that this broadcast authentication scheme is secure *provided no two users collude*. For example, using the keys that he has, user U_1 may attempt to trick user U_6 into accepting an inauthentic message, but users U_1 and U_2 may not collude and share their keys in such an attempt.

(c) Show that the scheme presented in part (b) is completely insecure if two users are allowed to collude.

6.5 (MAC combiners). We want to build a MAC system $\mathcal{I}$ using two MAC systems $\mathcal{I}_1 = (S_1, V_1)$ and $\mathcal{I}_2 = (S_2, V_2)$, so that if at some time one of $\mathcal{I}_1$ or $\mathcal{I}_2$ is broken (but not both) then $\mathcal{I}$ is still secure. Put another way, we want to construct $\mathcal{I}$ from $\mathcal{I}_1$ and $\mathcal{I}_2$ such that $\mathcal{I}$ is secure if either $\mathcal{I}_1$ or $\mathcal{I}_2$ is secure.

(a) Define $\mathcal{I} = (S, V)$, where

$$S(\ (k_1, k_2),\ m) := (\ S_1(k_1, m),\ S_2(k_2, m)\ ),$$

and V is defined in the obvious way: on input $(k, m, (t_1, t_2))$, V accepts iff both $V_1(k_1, m, t_1)$ and $V_2(k_2, m, t_2)$ accept. Show that $\mathcal{I}$ is secure if either $\mathcal{I}_1$ or $\mathcal{I}_2$ is secure.

(b) Suppose that $\mathcal{I}_1$ and $\mathcal{I}_2$ are deterministic MAC systems (see the definition on page 217), and that both have tag space $\{0,1\}^n$. Define the deterministic MAC system $\mathcal{I} = (S, V)$, where

$$S(\ (k_1, k_2),\ m) := S_1(k_1, m) \oplus S_2(k_2, m).$$

Show that $\mathcal{I}$ is secure if either $\mathcal{I}_1$ or $\mathcal{I}_2$ is secure.

6.6 (Concrete attacks on CBC and cascade). We develop attacks on F_{CBC} and F^* as prefix-free PRFs to show that for both security degrades quadratically with number of queries Q that the attacker makes. For simplicity, let us develop the attack when inputs are exactly three blocks long.

(a) Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$ where $\mathcal{X} = \{0,1\}^n$, where $|\mathcal{X}|$ is super-poly. Consider the F_{CBC} prefix-free PRF with input space $\mathcal{X}^3$. Suppose an adversary queries the challenger at points $(x_1, y_1, z),\ (x_2, y_2, z),\ \dots (x_Q, y_Q, z)$, where the x_i's, the y_i's, and z are chosen randomly from $\mathcal{X}$. Show that if $Q \approx \sqrt{|\mathcal{X}|}$, the adversary can predict the PRF at a new point in $\mathcal{X}^3$ with probability at least 1/2.

(b) Show that a similar attack applies to the three-block cascade F^* prefix-free PRF built from a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{K})$. Assume $\mathcal{X} = \mathcal{K}$ and $|\mathcal{K}|$ is super-poly. After making $Q \approx \sqrt{|\mathcal{K}|}$ queries in $\mathcal{X}^3$, your adversary should be able to predict the PRF at a new point in $\mathcal{X}^3$ with probability at least 1/2.

6.7 (Weakly secure MACs). In Attack Game 6.1 the adversary can issue signing queries and obtain message-tag pairs (m_i, t_i) for $i = 1, \dots, Q$. It can win the game in one of two ways: (i) it can produce a new valid tag t for one of $m_1, \dots, m_Q$, or (ii) it can produce a forgery pair (m, t) for an entirely new message m, namely, an m that is not one of $m_1, \dots, m_Q$. In this exercise we consider a weaker notion of security for a MAC where the adversary can only win the game by producing a forgery of type (ii). One can modify the winning condition in Attack Games 6.1 and 6.2 to reflect this weaker security notion. In Attack Game 6.1, this means that to win, in addition to being a valid pair, the adversary's candidate forgery pair (m, t) must satisfy the constraint that m is not among the signing queries. In Attack Game 6.2, this means that the adversary wins if the challenger ever responds to a verification query $(\hat{m}_j, \hat{t}_j)$ with accept, where $\hat{m}_j$ is not among the signing queries made prior to this verification query. These two modified attack games correspond to notions of security that we call *weak security without verification queries* and *weak security with verification queries.*

(a) The analog of Theorem 6.1 does not hold for these weaker security notions. Your goal is to develop an explicit counter-example. Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{T})$ where $\mathcal{K} = \{0,1\}^n$ and $\mathcal{T}$ is super-poly. Consider the following "sabotaged" MAC system $\mathcal{I} = (S, V)$, defined over $(\mathcal{K}, \mathcal{X}, \mathcal{T}')$, where $\mathcal{T}' := \mathcal{T} \times \{0, 1, \dots, n\}$:

$$\begin{aligned} S(k, m) &:= \big(F(k, m), n\big) \\ V\big(k, m, (t, i)\big) &:= \big\{\text{output } \mathsf{accept} \text{ iff } t = F(k, m) \text{ and } k[i] = 0\big\} \end{aligned}$$

where $k[i]$ is bit i of k for $i = 0, 1, \dots, n-1$ and $k[n] := 0$. Show that $\mathcal{I}$ provides weak security without verification queries, but does not provide weak security with verification queries.

(b) Show that $\mathcal{I}$ from part (a) is insecure with respect to either Attack Games 6.1 or 6.2.

6.8 (Fixing CBC: a bad idea). We showed that CBC is a prefix-free secure PRF but not a secure PRF. We showed that prepending the length of the message makes CBC a secure PRF. Show that appending the length of the message prior to applying CBC does not make CBC a secure PRF.

6.9 (Fixing CBC: a really bad idea). To avoid extension attacks on CBC, one might be tempted to define a CBC-MAC with a *randomized* IV. This is a MAC with a probabilistic signing algorithm that on input $k \in \mathcal{K}$ and $(x_1, \ldots, x_v) \in \mathcal{X}^{\leq \ell}$, works as follows: choose $IV \in \mathcal{X}$ at random; output (IV, t), where $t := F_{\text{CBC}}(x_1 \oplus IV, x_2, \ldots, x_v)$. On input $(k, (x_1, \ldots, x_v), (IV, t))$, the verification algorithms tests if $t = F_{\text{CBC}}(x_1 \oplus IV, x_2, \ldots, x_v)$. Show that this MAC is completely insecure, and is not even a prefix-free secure PRF.

6.10 (Truncated CBC). Prove that truncating the output of CBC gives a secure PRF for variable length messages. More specifically, if CBC is instantiated with a block cipher that operates on n-bit blocks, and we truncate the output of CBC to $w < n$ bits, then this truncated version is a secure PRF on variable length inputs, provided $1/2^{n-w}$ is negligible.

Hint: Adapt the proof of Theorem 6.3.

6.11 (Truncated cascade). In the previous exercise, we saw that truncating the output of the CBC construction yields a secure PRF. In this exercise, you are to show that the same does *not* hold for the cascade construction, by giving an explicit counter-example. For your counter-example, you may assume a secure PRF F' (defined over any convenient input, output, and key spaces, of your choosing). Using F', construct another PRF F, such that (a) F is a secure PRF, but (b) the corresponding truncated version of F^* is not a secure PRF.

6.12 (Truncated cascade in the ideal cipher model). In the previous exercise, we saw that the truncated cascade may not be secure when instantiated with certain PRFs. However, in your counter-example, that PRF was constructed precisely to make cascade fail — intuitively, for "typical" PRFs, one would not expect this to happen. To substantiate this intuition, this exercise asks you to prove that in the *ideal cipher model* (see Section 4.7), the cascade construction is a secure PRF. More precisely, if we model F as the encryption function of an *ideal cipher*, then the truncated version of F^* is a secure PRF. Here, you may assume that F operates on n-bit blocks and n-bit keys, and that the output of F^* is truncated to w bits, where $1/2^{n-w}$ is negligible.

6.13 (Non-adaptive attacks on CBC and cascade). This exercise examines whether variable length CBC and cascade are secure PRFs against *non-adaptive* adversaries, i.e., adversaries that make their queries all at once (see Exercise 4.6).

(a) Show that CBC is a secure PRF against non-adaptive adversaries, assuming the underlying function F is a PRF.

Hint: Adapt the proof of Theorem 6.3.

(b) Give a non-adaptive attack that breaks the security of cascade as a PRF, regardless of the choice of F.

6.14 (Generalized CMAC).

(a) Show that the CMAC *rpf* (Section 6.10) is a randomized 2^{-n}-prefix-free encoding.

(b) Use the CMAC *rpf* to convert cascade into a bit-wise secure PRF.

6.15 (A simple randomized prefix-free encoding). Show that appending a random message block gives a randomized prefix-free encoding. That is, the following function

$$rpf(k,m) = m \parallel k$$

is a randomized $1/|\mathcal{X}|$-prefix-free encoding. Here, $m \in \mathcal{X}^{\leq \ell}$ and $k \in \mathcal{X}$.

6.16 (An insecure variant of CMAC). Show that CMAC is insecure as a PRF if the sub-key generation algorithm outputs k_0 and k_2 as in the current algorithm, but sets $k_1 \leftarrow L$.

6.17 (Domain extension). This exercise explores some simple ideas for extending the domain of a MAC system that do not work. Let $\mathcal{I} = (S, V)$ be a deterministic MAC (see the definition on page 217), defined over $(\mathcal{K}, \mathcal{M}, \{0,1\}^n)$. Each of the following are signing algorithms for deterministic MACs with message space $\mathcal{M}^2$. You are to show that each of the resulting MACs are insecure.

(a) $S_1(k, (a_1, a_2)) = S(k, a_1) \parallel S(k, a_2)$,

(b) $S_2(k, (a_1, a_2)) = S(k, a_1) \oplus S(k, a_2)$,

(c) $S_3((k_1, k_2), (a_1, a_2)) = S(k_1, a_1) \parallel S(k_2, a_2)$,

(d) $S_4((k_1, k_2), (a_1, a_2)) = S(k_1, a_1) \oplus S(k_2, a_2)$.

6.18 (Integrity for database records). Let (S, V) be a secure MAC defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. Consider a database containing records $m_1, \ldots, m_n \in \mathcal{M}$. To provide integrity for the data the data owner generates a random secret key $k \in \mathcal{K}$ and stores $t_i \leftarrow S(k, m_i)$ alongside record m_i for every $i = 1, \ldots, n$. This does not ensure integrity because an attacker can remove a record from the database or duplicte a record without being detected. To prevent addition or removal of records the data owner generates another secret key $k' \in \mathcal{K}$ and computes $t \leftarrow S\big(k',\ (t_1, \ldots, t_n)\big)$ (we are assuming that $\mathcal{T}^n \subseteq \mathcal{M}$). She stores (k, k', t) on her own machine, away from the database.

(a) Show that updating a single record in the database can be done efficiently. That is, explain what needs to be done to recompute the tag t when a single record m_j in the database is replaced by an updated record m'_j.

(b) Does this approach ensure database integrity? Suppose the MAC (S, V) is built from a secure PRF F defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ where $|\mathcal{T}|$ is super-poly. Show that the following PRF F_n is a secure PRF on message space $\mathcal{M}^n$

$$F_n\big((k, k'), (m_1, \ldots, m_n)\big) := F\big(k',\ \big(F(k, m_1), \ldots, F(k, m_n)\big)\big)\,.$$

6.19 (Timing attacks). Let (S, V) be a deterministic MAC system where tags $\mathcal{T}$ are n-bytes long. The verification algorithm $V(k, m, t)$ is implemented as follows: it first computes $t' \leftarrow S(k, m)$ and then does:

```
for $i \leftarrow 0$ to $n - 1$ do:
    if $t[i] \neq t'[i]$ output reject and exit
output accept
```

(a) Show that this implementation is vulnerable to a timing attack. An attacker who can submit arbitrary queries to algorithm V and accurately measure V's response time can forge a valid tag on every message m of its choice with at most $256 \cdot n$ queries to V.

(b) How would you implement V to prevent the timing attack from part (a)?

Chapter 7

Message integrity from universal hashing

In the previous chapter we showed how to build secure MACs from secure PRFs. In particular, we discussed the ECBC, NMAC, and PMAC constructions. We stated security theorems for these MACs, but delayed their proofs to this chapter.

In this chapter we describe a general paradigm for constructing MACs using hash functions. By a **hash function** we generally mean a function H that maps inputs in some large set $\mathcal{M}$ to short outputs in $\mathcal{T}$. Elements in $\mathcal{T}$ are often called **message digests** or just digests. Keyed hash functions, used throughout this chapter, also take as input a key k.

At a high level, MACs constructed from hash functions work in two steps. First, we use the hash function to hash the message m to a short digest t. Second, we apply a PRF to the digest t, as shown in Fig. 7.1.

As we will see, ECBC, NMAC, and PMAC$_0$ are instances of this "hash-then-PRF" paradigm. For example, for ECBC (described in Fig. 6.5a), the CBC function acts as a hash function that hashes long input messages into short digests. The final application of the PRF using the key k_2 is the final PRF step. The hash-then-PRF paradigm will enable us to directly and quite easily deduce the security of ECBC, NMAC, and PMAC$_0$.

The hash-then-PRF paradigm is very general and enables us to build new MACs out of a wide variety of hash functions. Some of these hash functions are very fast, and yield MACs that are more efficient than those discussed in the previous chapter.

7.1 Universal hash functions (UHFs)

We begin our discussion by defining a **keyed hash function** — a widely used tool in cryptography. A keyed hash function H takes two inputs: a key k and a message m. It outputs a short digest $t := H(k, m)$. The key k can be thought of as a hash function selector: for every k we obtain a specific function $H(k, \cdot)$ from messages to digests. More precisely, keyed hash functions are defined as follows:

Definition 7.1 (Keyed hash functions). *A* ***keyed hash function*** *H is a deterministic algorithm that takes two inputs, a* ***key*** *k and a* ***message*** *m; its output $t := H(k, x)$ is called a* ***digest****. As usual, there are associated spaces: the keyspace $\mathcal{K}$, in which k lies, a message space $\mathcal{M}$, in*

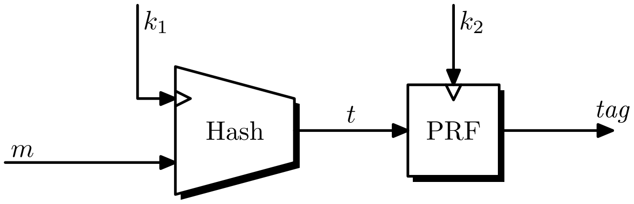


Figure 7.1: The hash-then-PRF paradigm

which m lies, and the digest space $\mathcal{T}$, in which t lies. We say that the hash function H is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$.

We note that the output digest $t \in \mathcal{T}$ can be much shorter than the input message m. Typically digests will have some fixed size, say 128 or 256 bits, independent of the input message length. A hash function $H(k, \cdot)$ can map gigabyte long messages into just 256-bit digests.

We say that two messages $m_0, m_1 \in \mathcal{M}$ form a **collision for H under key $k \in \mathcal{K}$** if

$$H(k,\ m_0) = H(k,\ m_1) \qquad \text{and} \qquad m_0 \neq m_1.$$

Since the digest space $\mathcal{T}$ is typically much smaller than the message space $\mathcal{M}$, many such collisions exist. However, a general property we shall desire in a hash function is that it is hard to actually *find* a collision. As we shall eventually see, there are a number of ways to formulate this "collision resistance" property. These formulations differ in subtle ways in how much information about the key an adversary gets in trying to find a collision. In this chapter, we focus on the weakest formulation of this collision resistance property, in which the adversary must find a collision with *no information about the key at all.* On the one hand, this property is weak enough that we can actually build very efficient hash functions that satisfy this property without making any assumptions at all on the computational power of the adversary. On the other hand, this property is strong enough to ensure that the hash-then-PRF paradigm yields a secure MAC.

Hash functions that satisfy this very weak collision resistance property are called **universal hash functions**, or **UHFs**. Universal hash functions are used in various branches of computer science, most notably for the construction of efficient hash tables. UHFs are also widely used in cryptography. Before we can analyze the security of the hash-then-PRF paradigm, we first give a more formal definition of UHFs. As usual, to make this intuitive notion more precise, we define an attack game.

Attack Game 7.1 (universal hash function). For a keyed hash function H defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger picks a random $k \xleftarrow{R} \mathcal{K}$ and keeps k to itself.
- $\mathcal{A}$ outputs two distinct messages $m_0, m_1 \in \mathcal{M}$.

We say that $\mathcal{A}$ wins the above game if $H(k, m_0) = H(k, m_1)$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\text{UHF}\mathsf{adv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. □

We now define several different notions of UHF, which depend on the power of the adversary and its advantage in the above attack game.

Definition 7.2. *Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$,*

- *We say that H is an **ϵ-bounded universal hash function**, or **ϵ-UHF**, if* $\mathrm{UHF}\mathsf{adv}[\mathcal{A}, H] \leq \epsilon$ *for all adversaries $\mathcal{A}$ (even inefficient ones).*
- *We say that H is a **statistical UHF** if it is an ϵ-UHF for some negligible ϵ.*
- *We say that H is a **computational UHF** if* $\mathrm{UHF}\mathsf{adv}[\mathcal{A}, H]$ *is negligible for all efficient adversaries $\mathcal{A}$.*

Statistical UHFs are secure against all adversaries, efficient or not: no adversary can win Attack Game 7.1 against a statistical UHF with non-negligible advantage. The main reason that we consider computationally unbounded adversaries is that we *can*: unlike most other security notions we discuss in this text, good UHFs are something we know how to build without any computational restrictions on the adversary. Note that every statistical UHF is also a computational UHF, but the converse is not true.

If H is a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, an alternative characterization of the ϵ-UHF property is the following (see Exercise 7.3):

for every pair of distinct messages $m_0, m_1 \in \mathcal{M}$ we have $\Pr[H(k, m_0) = H(k, m_1)] \leq \epsilon$, where the probability is over the random choice of $k \in \mathcal{K}$. (7.1)

7.1.1 Multi-query UHFs

It will be convenient to consider a generalization of a computational UHF. Here the adversary wins if he can output a list of distinct messages so that some pair of messages in the list is a collision for $H(k, \cdot)$. The point is that although the adversary may not know exactly which pair of messages in his list cause the collision, he still wins the game. In more detail, a multi-query UHF is defined using the following game:

Attack Game 7.2 (multi-query UHF). For a keyed hash function H over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger picks a random $k \xleftarrow{\text{R}} \mathcal{K}$ and keeps k to itself.
- $\mathcal{A}$ outputs distinct messages $m_1, \ldots, m_s \in \mathcal{M}$.

We say that $\mathcal{A}$ wins the above game if there are indices $i \neq j$ such that $H(k, m_i) = H(k, m_j)$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. We call $\mathcal{A}$ a **Q-query UHF adversary** if it always outputs a list of size $s \leq Q$. □

Definition 7.3. *We say that a hash function H over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ is a **multi-query UHF** if for all efficient adversaries $\mathcal{A}$, the quantity* $\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, H]$ *is negligible.*

Lemma 7.1 below shows that every UHF is also a multi-query UHF. However, for particular constructions, we can sometimes get better security bounds.

Lemma 7.1. *If H is a computational UHF, then it is also a multi-query UHF.*

In particular, for every Q-query UHF adversary $\mathcal{A}$, there exists a UHF adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, H] \leq (Q^2/2) \cdot \mathrm{UHF}\mathsf{adv}[\mathcal{B}, H]. \tag{7.2}$$

Proof. The UHF adversary $\mathcal{B}$ runs $\mathcal{A}$ and obtains $s \leq Q$ distinct messages $m_1, \ldots, m_s$. It chooses a random pair of distinct indices i and j from $\{1, \ldots, s\}$, and outputs m_i and m_j. The list generated by $\mathcal{A}$ contains a collision for $H(k, \cdot)$ with probability MUHF$\mathsf{adv}[\mathcal{A}, H]$ and $\mathcal{B}$ will choose a colliding pair with probability at least $2/Q^2$. Hence, UHF$\mathsf{adv}[\mathcal{B}, H]$ is at least MUHF$\mathsf{adv}[\mathcal{A}, H] \cdot (2/Q^2)$, as required. □

7.1.2 Mathematical details

As usual, we give a more mathematically precise definition of a UHF using the terminology defined in Section 2.3.

Definition 7.4 (Keyed hash functions). *A **keyed hash function** is an efficient algorithm H, along with three families of spaces with system parameterization P:*

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \textit{and} \quad \mathbf{T} = \{\mathcal{T}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

such that

1. $\mathbf{K}$*,* $\mathbf{M}$*, and* $\mathbf{T}$ *are efficiently recognizable.*

2. $\mathbf{K}$ *and* $\mathbf{T}$ *are efficiently sampleable.*

3. *Algorithm H is an efficient deterministic algorithm that on input $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, and $m \in \mathcal{M}_{\lambda,\Lambda}$, outputs an element of $\mathcal{T}_{\lambda,\Lambda}$.*

In defining UHFs we parameterize Attack Game 7.1 by the security parameter λ. The advantage UHF$\mathsf{adv}[\mathcal{A}, H]$ is then a function of λ.

The information-theoretic property (7.1) is the more traditional approach in the literature in defining ϵ-UHFs for individual hash functions with no security or system parameters; in our asymptotic setting, if property (7.1) holds for each setting of the security and system parameters, then our definition of an ϵ-UHF will certainly be satisfied.

7.2 Constructing UHFs

The challenge in constructing good universal hash functions (UHFs) is to construct a function that achieves a small collision probability using a short key. Preferably, the size of the key should not depend on the length of the message being hashed. We give three constructions. The first is an elegant construction of a *statistical* UHF using modular arithmetic and polynomials. Our second construction is based on the CBC and cascade functions defined in Section 6.4. We show that both are *computational* UHFs. The third construction is based on PMAC$_0$ from Section 6.11.

7.2.1 Construction 1: UHFs using polynomials

We start with a UHF construction using polynomials modulo a prime. Let ℓ be a (poly-bounded) length parameter and let p be a prime. We define a hash function H_{poly} that hashes a message $m \in \mathbb{Z}_p^{\leq \ell}$ to a single element $t \in \mathbb{Z}_p$. The key space is $\mathcal{K} := \mathbb{Z}_p$.

Let m be a message, so $m = (a_1, a_2, \ldots, a_v) \in \mathbb{Z}_p^{\leq \ell}$ for some $0 \leq v \leq \ell$. Let $k \in \mathbb{Z}_p$ be a key. The hash function $H_{\mathrm{poly}}(k, m)$ is defined as follows:

$$H_{\mathrm{poly}}\big(k,\ (a_1, \ldots, a_v)\big) := k^v + a_1 k^{v-1} + a_2 k^{v-2} + \cdots + a_{v-1} k + a_v \in \mathbb{Z}_p \tag{7.3}$$

That is, we use $(1, a_1, a_2, \ldots, a_v)$ as the vector of coefficients of a polynomial $f(X)$ of degree v and then evaluate $f(X)$ at a secret point k.

A very useful feature of this hash function is that it can be evaluated without knowing the length of the message ahead of time. One can feed message blocks into the hash as they become available. When the message ends we obtain the final hash. We do so using Horner's method for polynomial evaluation:

```
Input:    $m = (a_1, a_2, \ldots, a_v) \in \mathbb{Z}_p^{\leq \ell}$ and key $k \in \mathbb{Z}_p$
Output:   $t := H_{\text{poly}}(k, m)$
1.   Set $t \leftarrow 1$
2.   For $i \leftarrow 1$ to $v$:
3.        $t \leftarrow t \cdot k + a_i \ \in \mathbb{Z}_p$
4.   Output $t$
```

It is not difficult to show that this algorithm produces the same value as defined in (7.3). Observe that a long message can be processed one block at a time using little additional space. Every iteration takes one multiplication and one addition.

On a machine that has several multiplication units, say four units, we can use a 4-way parallel version of Horner's method to utilize all the available units and speed up the evaluation of H_{poly}. Assuming the length of m is a multiple of 4, simply replace lines (2) and (3) above with the following

```
2.   For $i \leftarrow 1$ to $v$ incrementing $i$ by 4 at every iteration:
3.        $t \leftarrow t \cdot k^4 + a_i \cdot k^3 + a_{i+1} \cdot k^2 + a_{i+2} \cdot k + a_{i+3} \ \in \mathbb{Z}_p$
```

One can precompute the values k^2, k^3, k^4 in $\mathbb{Z}_p$. Then at every iteration we process four blocks of the message using four multiplications that can all be done in parallel.

Security as a UHF. Next we show that H_{poly} is an (ℓ/p)-UHF. If p is super-poly, this implies that ℓ/p is negligible, which means that H_{poly} is a statistical UHF.

Lemma 7.2. *The function H_{poly} over $(\mathbb{Z}_p,\ (\mathbb{Z}_p)^{\leq \ell},\ \mathbb{Z}_p)$ defined in (7.3) is an (ℓ/p)-UHF.*

Proof. Consider two distinct messages $m_0 = (a_1, \ldots, a_u)$ and $m_1 = (b_1, \ldots, b_v)$ in $(\mathbb{Z}_p)^{\leq \ell}$. We show that $\Pr[H_{\text{poly}}(k, m_0) = H_{\text{poly}}(k, m_1)] \leq \ell/p$, where the probability is over the random choice of key k in $\mathbb{Z}_p$. Define the two polynomials:

$$\begin{aligned} f(X) &:= X^u + a_1 X^{u-1} + a_2 X^{u-2} + \cdots + a_{u-1} X + a_u \\ g(X) &:= X^v + b_1 X^{v-1} + b_2 X^{v-2} + \cdots + b_{v-1} X + b_v \end{aligned} \tag{7.4}$$

in $\mathbb{Z}_p[X]$. Then, by definition of H_{poly} we need to show that

$$\Pr[f(k) = g(k)] \leq \ell/p$$

where k is uniform in $\mathbb{Z}_p$. In other words, we need to bound the number of points $k \in \mathbb{Z}_p$ for which $f(k) - g(k) = 0$. Since the messages m_0 and m_1 are distinct we know that $f(X) - g(X)$ is a nonzero polynomial. Furthermore, its degree is at most ℓ and therefore it has at most ℓ roots in $\mathbb{Z}_p$. It follows that there are at most ℓ values of $k \in \mathbb{Z}_p$ for which $f(k) = g(k)$ and therefore, for a random $k \in \mathbb{Z}_p$ we have $\Pr[f(k) = g(k)] \leq \ell/p$ as required. □

Why the leading term k^v in $H_{\text{poly}}(k, m)$? The definition of $H_{\text{poly}}(k, m)$ in (7.3) includes a leading term k^v. This term ensures that the function is a statistical UHF for variable size inputs. If instead we defined $H_{\text{fpoly}}(k, m)$ without this term, namely

$$H_{\text{fpoly}}\big(k,\ (a_1, \ldots, a_v)\big) := a_1 k^{v-1} + a_2 k^{v-2} + \cdots + a_{v-1} k + a_v \in \mathbb{Z}_p, \tag{7.5}$$

then the result would not be a UHF for variable size inputs. For example, the two messages $m_0 = (a_1, a_2) \in \mathbb{Z}_p^2$ and $m_1 = (0, a_1, a_2) \in \mathbb{Z}_p^3$ are a collision for H_{fpoly} under all keys $k \in \mathbb{Z}_p$. Nevertheless, in Exercise 7.16 we show that H_{fpoly} is a statistical UHF if we restrict its input space to messages of fixed length, i.e., $\mathcal{M} := \mathbb{Z}_p^\ell$ for some ℓ. Specifically, H_{fpoly} is an $(\ell - 1)/p$-UHF. In contrast, the function H_{poly} defined in (7.3) is a statistical UHF for the input space $\mathbb{Z}_p^{\leq \ell}$ containing messages of varying lengths.

Remark 7.1. The function H_{poly} takes inputs in $\mathbb{Z}_p^{\leq \ell}$ and outputs values in $\mathbb{Z}_p$. This can be difficult to work with: we prefer to work with functions that operate on blocks of n-bits for some n. We can adapt the definition of H_{poly} in (7.3) so that instead of working in $\mathbb{Z}_p$, arithmetic is done in the finite field $\text{GF}(2^n)$. This version of H_{poly} is an $\ell/2^n$-UHF using the exact same analysis as in Lemma 7.2. It outputs values in $\text{GF}(2^n)$. In Exercise 7.1 we show that simply defining H_{poly} modulo 2^n (i.e., working in $\mathbb{Z}_{2^n}$) is a completely insecure UHF. □

Caution in using UHFs. UHFs can be brittle — an adversary who learns the value of the function at a few points can completely recover the secret key. For example, the value of $H_{\text{poly}}(k, \cdot)$ at a single point completely exposes the secret key $k \in \mathbb{Z}_p$. Indeed, if $m = (a_1)$, since $H_{\text{poly}}(k, m) = k + a_1$ an adversary who has both m and $H_{\text{poly}}(k, m)$ immediately obtains $k \in \mathbb{Z}_p$. Consequently, in all of our applications of UHFs, we will always hide values of the UHF from the adversary, either by encrypting them or by other means.

Mathematical details. The definition of H_{poly} requires a prime p. So far we simply assumed that p is a public value picked at the beginning of time and fixed forever. In the formal UHF framework (Section 7.1.2), the prime p is a system parameter denoted by Λ. It is generated by a *system parameter generation algorithm* P that takes the security parameter λ as input and outputs some prime p.

More precisely, let $L : \mathbb{Z} \to \mathbb{Z}$ be some function that maps the security parameter to the desired bit length of the prime. Then the formal description of H_{poly} includes a description of an algorithm P that takes the security parameter λ as input and outputs a prime p of length $L(\lambda)$ bits. Specifically, $\Lambda := p$ and

$$\mathcal{K}_{\lambda,p} = \mathbb{Z}_p, \quad \mathcal{M}_{\lambda,p} = \mathbb{Z}_p^{\leq \ell(\lambda)}, \quad \text{and} \quad \mathcal{T}_{\lambda,p} = \mathbb{Z}_p,$$

where $\ell : \mathbb{Z} \to \mathbb{Z}^{\geq 0}$ is poly-bounded. By Lemma 7.2 we know that

$$\text{UHF}\mathsf{adv}[\mathcal{A}, H_{\text{poly}}](\lambda) \leq \ell(\lambda)/2^{L(\lambda)}$$

which is a negligible function of λ provided $2^{L(\lambda)}$ is super-poly.

7.2.2 Construction 2: CBC and cascade are computational UHFs

Next we show that the CBC and cascade constructions defined in Section 6.4 are computational UHFs. More generally, we show that any prefix-free secure PRF that is also extendable is a computational UHF. Recall that a PRF F over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$ is extendable if for all $k \in \mathcal{K},\ x, y \in \mathcal{X}^{\leq \ell - 1}$, and $a \in \mathcal{X}$ we have:

$$\text{if} \quad F(k, x) = F(k, y) \quad \text{then} \quad F(k,\ x \parallel a) = F(k,\ y \parallel a).$$

In the previous chapter we showed that both CBC and cascade are prefix-free secure PRFs and that both are extendable.

Theorem 7.3. *Let PF be an extendable and prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell+1}, \mathcal{Y})$ where $|\mathcal{Y}|$ is super-poly and $|\mathcal{X}| > 1$. Then PF is a computational UHF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$.*

> *In particular, for every UHF adversary $\mathcal{A}$ that plays Attack Game 7.1 with respect to PF, there exists a prefix-free PRF adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{UHF}\mathsf{adv}[\mathcal{A}, PF] \leq \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}, PF] + \frac{1}{|\mathcal{Y}|}. \tag{7.6}$$
>
> *Moreover, $\mathcal{B}$ makes only two queries to PF.*

Proof. Let $\mathcal{A}$ be a UHF adversary attacking PF. We build a prefix-free PRF adversary $\mathcal{B}$ attacking PF. $\mathcal{B}$ plays the adversary in the PRF Attack Game 4.2. Its goal is to distinguish between Experiment 0 where it queries a function $f \leftarrow PF(k, \cdot)$ for a random $k \in \mathcal{K}$, and Experiment 1 where it queries a random function $f \xleftarrow{\text{R}} \mathrm{Funs}[\mathcal{X}^{\leq \ell+1}, \mathcal{Y}]$.

We first give some intuition as to how $\mathcal{B}$ works. $\mathcal{B}$ starts by running the UHF adversary $\mathcal{A}$ to obtain two distinct messages $m_0, m_1 \in \mathcal{X}^{\leq \ell}$. By the definition of $\mathcal{A}$, we know that in Experiment 0 we have

$$\Pr\big[f(m_0) = f(m_1)\big] = \mathrm{UHF}\mathsf{adv}[\mathcal{A}, PF]$$

while in Experiment 1, since f is a random function and $m_0 \neq m_1$, we have

$$\Pr\big[f(m_0) = f(m_1)\big] = 1/|\mathcal{Y}|.$$

Hence, if $\mathcal{B}$ could query f at m_0 and m_1 it could distinguish between the two experiments with advantage $\big|\mathrm{UHF}\mathsf{adv}[\mathcal{A}, PF] - 1/|\mathcal{Y}|\big|$, which would prove the theorem.

Unfortunately, this design for $\mathcal{B}$ does not quite work: m_0 might be a proper prefix of m_1, in which case $\mathcal{B}$ is not allowed to query f at both m_0 and m_1, because $\mathcal{B}$ is supposed to be a prefix-free adversary. However, the extendability property provides a simple solution: we extend both m_0 and m_1 by a single block $a \in \mathcal{X}$ such that $m_0 \parallel a$ is no longer a proper prefix of $m_1 \parallel a$. If $m_0 = (a_1, \ldots, a_u)$ and $m_1 = (b_1, \ldots, b_v)$, then any $a \neq b_{u+1}$ will do the trick. Moreover, by the extension property we know that

$$PF(k,\ m_0) = PF(k,\ m_1) \qquad \Longrightarrow \qquad PF(k,\ m_0 \parallel a) = PF(k,\ m_1 \parallel a).$$

Since $m_0 \parallel a$ is no longer a proper prefix of $m_1 \parallel a$, our $\mathcal{B}$ is free to query f at both inputs. $\mathcal{B}$ then obtains the desired advantage in distinguishing Experiment 0 from Experiment 1.

In more detail, adversary $\mathcal{B}$ works as follows:

```
run $\mathcal{A}$ to obtain two distinct messages $m_0, m_1$ in $\mathcal{X}^{\leq \ell}$, where
        $m_0 = (a_1, \ldots, a_u)$ and $m_1 = (b_1, \ldots, b_v)$
assume $u \leq v$ (otherwise, swap the two messages)
if $m_0$ is a proper prefix of $m_1$
        choose some $a \in \mathcal{X}$ such that $a \neq b_{u+1}$
        $m_0' \leftarrow m_0 \parallel a$  and  $m_1' \leftarrow m_1 \parallel a$
else
        $m_0' \leftarrow m_0$  and  $m_1' \leftarrow m_1$
// At this point we know that $m_0'$ is not a proper prefix of $m_1'$ nor vice versa.
query $f$ at $m_0'$ and $m_1'$ and obtain $t_0 := f(m_0')$  and  $t_1 := f(m_1')$
if $t_0 = t_1$ output 1; otherwise output 0
```

Observe that $\mathcal{B}$ is a prefix-free PRF adversary that only makes two queries to f, as required. Now, for $b = 0, 1$ let p_b be the probability that $\mathcal{B}$ outputs 1 in Experiment b. Then in Experiment 0, we know that

$$p_0 := \Pr\left[f(m_0') = f(m_1')\right] \geq \Pr\left[f(m_0) = f(m_1)\right] = \text{UHF}\mathsf{adv}[\mathcal{A}, PF]. \tag{7.7}$$

In Experiment 1, we know that

$$p_1 := \Pr\left[f(m_0') = f(m_1')\right] = 1/|\mathcal{Y}|. \tag{7.8}$$

Therefore, by (7.7) and (7.8):

$$\text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{B}, PF] = |p_0 - p_1| \geq p_0 - p_1 \geq \text{UHF}\mathsf{adv}[\mathcal{A}, PF] - 1/|\mathcal{Y}|,$$

from which (7.6) follows. □

***PF* as a multi-query UHF.** Lemma 7.1 shows that *PF* is also a multi-query UHF. However, a direct proof of this fact gives a better security bound.

Theorem 7.4. *Let PF be an extendable and prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell+1}, \mathcal{Y})$, where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are super-poly and ℓ is poly-bounded. Then PF is a multi-query UHF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$.*

In particular, if $|\mathcal{X}| > \ell Q$, then for every Q-query UHF adversary $\mathcal{A}$, there exists a Q-query prefix-free PRF adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\text{MUHF}\mathsf{adv}[\mathcal{A}, PF] \leq \text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{B}, PF] + \frac{Q^2}{2|\mathcal{Y}|}. \tag{7.9}$$

Proof. The proof is similar to the proof of Theorem 7.3. Adversary $\mathcal{B}$ begins by running the Q-query UHF adversary $\mathcal{A}$ to obtain distinct messages $m_1, \ldots, m_s$ in $\mathcal{X}^{\leq \ell}$, where $s \leq Q$. Next, $\mathcal{B}$ finds an $a \in \mathcal{X}$ such that a is not equal to any of the message blocks in $m_1, \ldots, m_s$. Since $|\mathcal{X}|$ is super-poly, we may assume it is larger than ℓQ, and therefore this a must exist. Let $m_i' := m_i \parallel a$ for $i = 1, \ldots, s$. Then, by definition of a, the set $\{m_1', \ldots, m_s'\}$ is a prefix-free set. The prefix-free adversary $\mathcal{B}$ now queries the challenger at $m_1', \ldots, m_s'$ and obtains $t_1, \ldots, t_s$ in response. $\mathcal{B}$ outputs 1 if there exist $i \neq j$ such that $t_i = t_j$, and outputs 0 otherwise.

To analyze the advantage of $\mathcal{B}$, we let p_b be the probability that $\mathcal{B}$ outputs 1 in PRF Experiment b, for $b = 0, 1$. As in (7.7), the extension property implies that

$$p_0 \geq \mathrm{MUHF}\mathsf{adv}[\mathcal{A}, PF].$$

In Experiment 1 the union bound implies that

$$p_1 \leq \frac{Q(Q-1)}{2|\mathcal{Y}|}.$$

Therefore,

$$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}, PF] = |p_0 - p_1| \geq p_0 - p_1 \geq \mathrm{MUHF}\mathsf{adv}[\mathcal{A}, PF] - \frac{Q^2}{2|\mathcal{Y}|}$$

from which (7.9) follows. □

Applications of Theorems 7.3 and 7.4. Applying Theorem 7.4 to CBC and cascade proves that both are computational UHFs. We state the resulting error bounds in the following corollary, which follows from the bounds in the CBC theorem (Theorem 6.3) and the cascade theorem (Theorem 6.4).[1]

Corollary 7.5. *Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$. Then the CBC construction F_{CBC} (assuming $\mathcal{Y} = \mathcal{X}$ is super-poly size) and the cascade construction F^* (assuming $\mathcal{Y} = \mathcal{K}$), which take inputs in $\mathcal{X}^{\leq \ell}$ for poly-bounded ℓ, are computational UHFs.*

In particular, for every Q-query UHF adversary $\mathcal{A}$, there exist prefix-free PRF adversaries $\mathcal{B}_1, \mathcal{B}_2$, which are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, F_{\text{CBC}}] \leq \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_1, F] + \frac{Q^2(\ell+1)^2 + Q^2}{2|\mathcal{Y}|} \quad \textit{and} \tag{7.10}$$

$$\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, F^*] \leq Q(\ell+1) \cdot \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_2, F] + \frac{Q^2}{2|\mathcal{Y}|}. \tag{7.11}$$

Setting $Q := 2$ in (7.10)–(7.11) gives the error bounds on F_{CBC} and F^* as UHFs.

7.2.3 Construction 3: a parallel UHF from a small PRF

The CBC and cascade constructions yield efficient UHFs from small domain PRFs, but they are inherently sequential: they cannot take advantage of hardware parallelism. Fortunately, constructing a UHF from a small domain PRF that is suitable for a parallel architecture is not difficult. An example called XOR-hash, denoted $F^{\oplus}$, is shown in Fig. 7.2. XOR-hash is defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$, where $\mathcal{Y} = \{0,1\}^n$, and is built from a PRF F defined over $(\mathcal{K}, \mathcal{X} \times \{1, \dots, \ell\}, \mathcal{Y})$. The XOR-hash works as follows:

```
input: $k \in \mathcal{K}$ and $m = (a_1, \dots, a_v) \in \mathcal{X}^{\leq \ell}$ for some $0 \leq v \leq \ell$
output: a tag in $\mathcal{Y}$
    $t \leftarrow 0^n$
    for $i = 1$ to $v$ do:
        $t \leftarrow t \oplus F(k, (a_i, i))$
    output $t$
```

[1] Note that Theorem 7.4 compels us to apply Theorems 6.3 and 6.4 using $\ell + 1$ in place of ℓ.

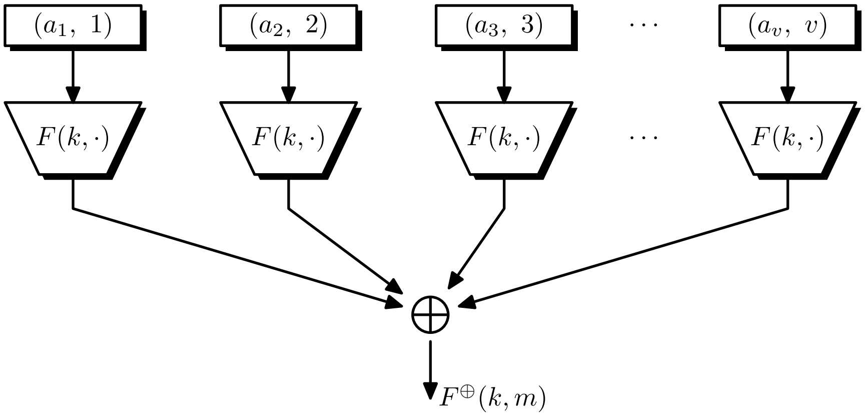


Figure 7.2: A parallel UHF from a small PRF

Evaluating $F^{\oplus}$ can easily be done in parallel. The following theorem shows that $F^{\oplus}$ is a computational UHF. Note that unlike our previous UHF constructions, security does not depend on the length of the input message. In the next section we will use $F^{\oplus}$ to construct a secure MAC suitable for parallel architectures.

Theorem 7.6. *Let F be a secure PRF and assume $|\mathcal{Y}|$ is super-poly. Then $F^{\oplus}$ is a computational UHF.*

> *In particular, for every UHF adversary $\mathcal{A}$, there exists a PRF adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that*

$$\mathrm{UHF}\mathsf{adv}[\mathcal{A}, F^{\oplus}] \le \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{1}{|\mathcal{Y}|}. \tag{7.12}$$

Proof. The proof is a sequence of two games.

Game 0. The challenger in this game computes:

$k \xleftarrow{\mathrm{R}} \mathcal{K},\ f \leftarrow F(k,\cdot)$

The adversary $\mathcal{A}$ outputs two distinct messages U, V in $\mathcal{X}^{\le \ell}$. Let $u := |U|$ and $v := |V|$. We define W_0 to be the event that the condition

$$\bigoplus_{i=0}^{u-1} f(U[i], i) = \bigoplus_{j=0}^{v-1} f(V[j], j) \tag{7.13}$$

holds in Game 0. Clearly, we have

$$\Pr[W_0] = \mathrm{UHF}\mathsf{adv}[\mathcal{A}, F^{\oplus}]. \tag{7.14}$$

Game 1. We play the "PRF card" and replace the challenger's computation by

$f \xleftarrow{\mathrm{R}} \mathrm{Funs}[\mathcal{X} \times \{1, \ldots, \ell\},\ \mathcal{Y}]$

We define W_1 to be the event that the condition (7.13) holds in Game 1.

As usual, there is a PRF adversary $\mathcal{B}$ such that

$$\big|\Pr[W_0] - \Pr[W_1]\big| \le \text{PRF}\mathsf{adv}[\mathcal{B}, F] \tag{7.15}$$

The crux of the proof is in bounding $\Pr[W_1]$, namely bounding the probability that (7.13) holds for the messages U, V. Assume $u \ge v$, swapping U and V if necessary. It is easy to see that since U and V are distinct, there must be an index i^* such that the pair $(U[i^*], i^*)$ on the left side of (7.13) does not appear among the pairs $(V[j], j)$ on the right side of (7.13): if $u > v$ then $i^* = u - 1$ does the job; otherwise, if $u = v$, then there must exist some i^* such that $U[i^*] \ne V[i^*]$, and this i^* does the job.

We can re-write (7.13) as

$$f(U[i^*], i^*) = \bigoplus_{i \ne i^*} f(U[i], i) \quad \oplus \quad \bigoplus_{j} f(V[j], j). \tag{7.16}$$

Since the left and right sides of (7.16) are independent, and the left side is uniformly distributed over $\mathcal{Y}$, equality holds with probability $1/|\mathcal{Y}|$. It follows that

$$\Pr[W_1] = 1/|\mathcal{Y}| \tag{7.17}$$

The proof of the theorem follows from (7.14), (7.15), and (7.17). □

In Exercise 7.28 we generalize Theorem 7.6 to derive bounds for $F^{\oplus}$ as a multi-query UHF.

7.3 PRF(UHF) composition: constructing MACs using UHFs

We now proceed to show that the hash-then-PRF paradigm yields a secure PRF provided the hash is a computational UHF. ECBC, NMAC, and PMAC$_0$ can all be viewed as instances of this construction and their security follows quite easily from the security of the hash-then-PRF paradigm.

Let H be a keyed hash function defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{X})$ and let F be a PRF defined over $(\mathcal{K}_F, \mathcal{X}, \mathcal{T})$. As usual, we assume $\mathcal{M}$ contains much longer messages than $\mathcal{X}$, so that H hashes long inputs to short digests. We build a new PRF, denoted F', by composing the hash function H with the PRF F, as shown in Fig. 7.3. More precisely, F' is defined as follows:

$$F'\big((k_1, k_2),\ m\big) := F(k_2,\ H(k_1,\ m)\ ) \tag{7.18}$$

We refer to F' as the **composition of** F **and** H. It takes inputs in $\mathcal{M}$ and outputs values in $\mathcal{T}$ using a key (k_1, k_2) in $\mathcal{K}_H \times \mathcal{K}_F$. Thus, we obtain a PRF with the same output space as the underlying F, but taking much longer inputs. The following theorem shows that F' is a secure PRF.

Theorem 7.7 (PRF(UHF) composition). *Suppose H is a computational UHF and F is a secure PRF. Then F' defined in (7.18) is a secure PRF.*

> *In particular, suppose $\mathcal{A}$ is a PRF adversary that plays Attack Game 4.2 with respect to F' and issues at most Q queries. Then there exist a PRF adversary $\mathcal{B}_F$ and a UHF adversary $\mathcal{B}_H$, which are elementary wrappers around $\mathcal{A}$, such that*

$$\text{PRF}\mathsf{adv}[\mathcal{A}, F'] \le \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + (Q^2/2) \cdot \text{UHF}\mathsf{adv}[\mathcal{B}_H, H]. \tag{7.19}$$

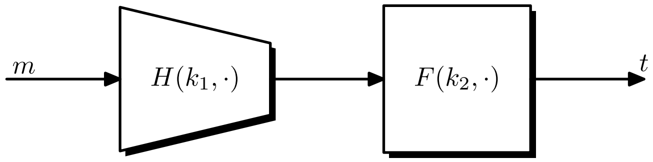


Figure 7.3: PRF(UHF) composition: MAC signing

More generally, there exists a Q-query UHF adversary $\mathcal{B}'_H$, which is an elementary wrapper around $\mathcal{A}$ such that

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F'] \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \mathrm{MUHF}\mathsf{adv}[\mathcal{B}'_H, H]. \tag{7.20}$$

To understand why H needs to be a UHF let us suppose for a minute that it is not. In particular, suppose it was easy to find distinct $m_0, m_1 \in \mathcal{M}$ such that $H(k_1, m_0) = H(k_1, m_1)$, without knowledge of k_1. This collision on H implies that $F'((k_1, k_2),\ m_0) = F'((k_1, k_2),\ m_1)$. But then F' is clearly not a secure PRF: the adversary could ask for $t_0 := F'((k_1, k_2),\ m_0)$ and $t_1 := F'((k_1, k_2),\ m_1)$ and then output 1 only if $t_0 = t_1$. When interacting with F' the adversary would always output 1, but for a random function he would most often output 0. Thus, the adversary successfully distinguishes F' from a random function. This argument shows that for F' to be a PRF it must be difficult to find collisions for H without knowledge of k_1. In other words, for F' to be a PRF the hash function H must be a UHF. Theorem 7.7 shows that this condition is sufficient.

Remark 7.2. The bound in Theorem 7.7 is tight. Consider the UHF H_{poly} discussed in Section 7.2.1. For concreteness, let us assume that $\ell = 2$, so the message space for H_{poly} is $\mathbb{Z}_p^2$, the output space is $\mathbb{Z}_p$, and the collision probability is $\epsilon = 1/p$. In Exercise 7.26, you are asked to show that for any fixed hash key k_1, among $\sqrt{p}$ random inputs to $H_{\mathrm{poly}}(k_1, \cdot)$, the probability of a collision is bounded from below by a constant; moreover, for any such collision, one can efficiently recover the key k_1. Now consider the MAC obtained from PRF(UHF) composition using H_{poly}. If the adversary ever finds two messages m_0, m_1 that cause an internal collision (i.e., a collision on H_{poly}) he can recover the secret H_{poly} key and then break the MAC. This shows that the term $(Q^2/2)\epsilon$ that appears in (7.19) cannot be substantially improved upon. □

Proof of Theorem 7.7. We now prove that the composition of F and H is a secure PRF.

Proof idea. Let $\mathcal{A}$ be an efficient PRF adversary that plays Attack Game 4.2 with respect to F'. We derive an upper bound on $\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F']$. That is, we bound $\mathcal{A}$'s ability to distinguish F' from a truly random function in $\mathrm{Funs}[\mathcal{M}, \mathcal{X}]$. As usual, we first observe that replacing the underlying secure PRF F with a truly random function f does not change $\mathcal{A}$'s advantage much. Next, we will show that, since f is a random function, the only way $\mathcal{A}$ can distinguish $F' := f(H(k_1, m))$ from a truly random function is if he can find two inputs m_0, m_1 such that $H(k_1, m_0) = H(k_1, m_1)$. But since H is a computational UHF, $\mathcal{A}$ cannot find collisions for $H(k_1, \cdot)$. Consequently, F' cannot be distinguished from a random function. □

Proof. We prove the bound in (7.20). Equation (7.19) follows from (7.20) by Lemma 7.1. We let

$\mathcal{A}$ interact with closely related challengers in three games. For $j = 0, 1, 2$, we define W_j to be the event that $\mathcal{A}$ outputs 1 at the end of Game j.

Game 0. The Game 0 challenger is identical to the challenger in Experiment 0 of the PRF Attack Game 4.2 with respect to F'. Without loss of generality we assume that $\mathcal{A}$'s queries to F' are all distinct. The challenger works as follows:

```
$k_1 \xleftarrow{R} \mathcal{K}_H$,  $k_2 \xleftarrow{R} \mathcal{K}_F$
upon receiving the $i$th query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) do:
    $x_i \leftarrow H(k_1,\ m_i)$
    $t_i \leftarrow F(k_2,\ x_i)$
    send $t_i$ to the adversary
```

Note that since $\mathcal{A}$ is guaranteed to make distinct queries, all the m_i values are distinct.

Game 1. Now we play the usual "PRF card," replacing the function $F(k_2, \cdot)$ by a truly random function f in Funs$[\mathcal{X}, \mathcal{T}]$, which we implement as a faithful gnome (as in Section 4.4.2). The Game 1 challenger works as follows:

```
      $k_1 \xleftarrow{R} \mathcal{K}_H$,   $t'_1, \ldots, t'_Q \xleftarrow{R} \mathcal{T}$
      upon receiving the $i$th query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) do:
          $x_i \leftarrow H(k_1,\ m_i)$
          $t_i \leftarrow t'_i$
(*)       if $x_i = x_j$ for some $j < i$ then $t_i \leftarrow t_j$
          send $t_i$ to the adversary
```

For $i = 1, \ldots, Q$, the value t'_i is chosen in advance to be the default, random value for $t_i = f(x_i)$. Although the messages are distinct, their hash values might not be. The line marked with a $(*)$ ensures that the challenger emulates a function in Funs$[\mathcal{X}, \mathcal{T}]$ — if two hash values collide, the challenger's response to both queries is the same. As usual, one can easily show that there is a PRF adversary $\mathcal{B}_F$ whose running time is about the same as that of $\mathcal{A}$ such that:

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F] \tag{7.21}$$

Game 2. Next, we make our gnome forgetful, by removing the line marked $(*)$.

We show that $\mathcal{A}$ cannot distinguish Games 1 and 2 using the fact that $\mathcal{A}$ cannot find collisions for H. Formally, we analyze the quantity $|\Pr[W_2] - \Pr[W_1]|$ using the Difference Lemma (Theorem 4.7). Let Z be the event that in Game 2 we have $x_i = x_j$ for some $i \neq j$. Event Z is essentially the winning condition in the multi-query UHF game (Attack Game 7.2) with respect to H. In particular, there is a Q-query UHF adversary $\mathcal{B}'_H$ that wins Attack Game 7.2 with probability equal to $\Pr[Z]$. Adversary $\mathcal{B}'_H$ simply emulates the challenger in Game 2 until $\mathcal{A}$ terminates and then outputs the queries $m_1, m_2, \ldots$ from $\mathcal{A}$ as its final list. This works, because in Game 2, the challenger does not really need the hash key k_1: it simply responds to each query with a random element of $\mathcal{T}$. Thus, adversary $\mathcal{B}'_H$ can easily emulate the challenger in Game 2 without knowledge of k_1. By definition of Z, we have $\mathrm{MUHF}\mathsf{adv}[\mathcal{B}'_H, H] = \Pr[Z]$.

Clearly, Games 1 and 2 proceed identically unless event Z occurs; in particular, $W_2 \wedge \bar{Z}$ occurs if and only if $W_1 \wedge \bar{Z}$ occurs. Applying the Difference Lemma, we obtain

$$\big|\Pr[W_2] - \Pr[W_1]\big| \leq \Pr[Z] = \mathrm{MUHF}\mathsf{adv}[\mathcal{B}'_H, H]. \tag{7.22}$$

Finishing the proof. The Game 2 challenger emulates for $\mathcal{A}$ a random function in Funs$[\mathcal{M},\mathcal{T}]$ and is therefore identical to an Experiment 1 PRF challenger with respect to F'. We obtain

$$\begin{aligned}\mathrm{PRF}\mathsf{adv}[\mathcal{A},F'] = & \bigl|\Pr[W_2]-\Pr[W_0]\bigr| \leq \\ & \bigl|\Pr[W_2]-\Pr[W_1]\bigr| + \bigl|\Pr[W_1]-\Pr[W_0]\bigr| \leq \\ & \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F,F] + \mathrm{MUHF}\mathsf{adv}[\mathcal{B}'_H,H]\end{aligned}$$

which proves (7.20), as required. □

7.3.1 Using PRF(UHF) composition: ECBC and NMAC security

Using Theorem 7.7 we can quickly prove security of many MAC constructions. It suffices to show that the MAC signing algorithm can be described as the composition of a PRF with a UHF. We begin by showing that ECBC and NMAC can be described this way and give more examples in the next two sub-sections.

Security of ECBC and NMAC follows directly from PRF(UHF) composition. The proof for both schemes runs as follows:

- First, we proved that CBC and cascade are prefix-free secure PRFs (Theorems 6.3 and 6.4). We observed that both are extendable.

- Next, we showed that any extendable prefix-free secure PRF is also a computational UHF (Theorem 7.3). In particular, CBC and cascade are computational UHFs.

- Finally, we proved that the composition of a computational UHF and a PRF is a secure PRF (Theorem 7.7). Hence, ECBC and NMAC are secure PRFs.

More generally, the encrypted PRF construction (Theorem 6.5) is an instance of PRF(UHF) composition and hence its proof follows from Theorem 7.7. The concrete bounds in the ECBC and NMAC theorems (Theorems 6.6 and 6.7) are obtained by plugging (7.10) and (7.11), respectively, into (7.20).

One can simplify the proof of ECBC and NMAC security by directly proving that CBC and cascade are computational UHFs. We proved that they are prefix-free secure PRFs, which is more than we need. However, this stronger result enabled us to construct other secure MACs such as CMAC (see Section 6.7).

7.3.2 Using PRF(UHF) composition with polynomial UHFs

Of course, one can use the PRF(UHF) construction with a polynomial-based UHF, such as H_{poly}. Depending on the underlying hardware, this construction can be much faster than either ECBC, NMAC, or PMAC$_0$ especially for very long messages.

Recall that H_{poly} hashes messages in $\mathbb{Z}_p^{\leq \ell}$ to digests in $\mathbb{Z}_p$, where p is a prime. For our PRF, we may very well want to use a block cipher, like AES, that takes as input an n-bit block.

To make this work, we have to somehow make an adjustment so that the output space of the hash is contained in the input space of the PRF. One way to do this is to choose the prime p so that it is just a little bit smaller than 2^n, so that we can encode hash digests as inputs to the PRF. This approach works; however, it has the drawback that we have to view the input to the hash as a sequence of elements of $\mathbb{Z}_p$. So, for example, with $n = 128$ as in AES, we could choose a

128-bit prime, but then the input to the hash would have to be broken up into, say, 120-bit (i.e., 15 byte) blocks. It would be even more convenient if we could also process the input to the hash directly as a sequence of n-bit blocks. Part (d) of Exercise 7.23 shows how this can be done, using a prime that is just a little bit *bigger* than 2^n. Yet another approach is that instead of basing the hash on arithmetic modulo a prime p, we instead base it on arithmetic in the finite field $\mathrm{GF}(2^n)$, as discussed in Remark 7.1.

7.3.3 Using PRF(UHF) composition: PMAC_0 security

Next we show that the PMAC_0 construction discussed in Section 6.11 is an instance of PRF(UHF) composition. Recall that PMAC_0 is built out of two PRFs, F_1, which is defined over $(\mathcal{K}_1, \mathbb{Z}_p, \mathcal{Y})$, and F_2, which is defined over $(\mathcal{K}_2, \mathcal{Y}, \mathcal{Z})$, where $\mathcal{Y} := \{0,1\}^n$.

The reader should review the PMAC_0 construction, especially Fig. 6.9. One can see that PMAC_0 is the composition of the PRF F_2 with a certain keyed hash function $\widehat{H}$, which is everything else in Fig. 6.9.

The goal now is to show that $\widehat{H}$ is a computational UHF. To do this, we observe that $\widehat{H}$ can be viewed as an instance of the XOR-hash construction in Section 7.2.3, applied to the PRF F' defined over $(\mathbb{Z}_p \times \mathcal{K}_1, \mathbb{Z}_p \times \{1, \ldots, \ell\}, \mathcal{Y})$ as follows:

$$F'((k, k_1), (a, i)) := F_1(k_1, a + i \cdot k).$$

So it suffices to show that F' is a secure PRF. But it turns out we can view F' itself as an instance of PRF(UHF) composition. Namely, it is the composition of the PRF F_1 with the keyed hash function H defined over $(\mathbb{Z}_p, \mathbb{Z}_p \times \{1, \ldots, \ell\}, \mathbb{Z}_p)$ as $H(k, (a, i)) := a + i \cdot k$. However, H is just a special case of H_{fpoly} (see Section 7.2.1). In particular, by the result of Exercise 7.16, H is a $1/p$-UHF.

The security of PMAC_0 follows from the above observations. The concrete security bound (6.28) in Theorem 6.11 follows from the concrete security bound (7.20) in Theorem 7.7 and the more refined analysis of XOR-hash in Exercise 7.28.

In the design of PMAC_0, we assumed the input space of F_1 was equal to $\mathbb{Z}_p$. While this simplifies the analysis, it makes it harder to work with in practice. Just as in Section 7.3.2 above, we would prefer to work with a PRF defined in terms of a block cipher, like AES, which takes as input an n-bit block. One can apply the same techniques discussed Section 7.3.2 to get a variant of PMAC_0 whose input space consists of sequences of n-bit blocks, rather than sequences of elements of $\mathbb{Z}_p$. For example, see Exercise 7.25.

7.4 The Carter-Wegman MAC

In this section we present a different paradigm for constructing secure MAC systems that offers different tradeoffs compared to PRF(UHF) composition.

Recall that in PRF(UHF) composition the adversary's advantage in breaking the MAC after seeing Q signed messages grows as $\epsilon \cdot Q^2/2$ when using an ϵ-UHF. Therefore to ensure security when many messages need to be signed, the ϵ-UHF must have a sufficiently small ϵ so that $\epsilon \cdot Q^2/2$ is small. This can hurt the performance of an ϵ-UHF like H_{poly} where the smaller ϵ is, the slower the hash function. As an example, suppose that after signing $Q := 2^{32}$ messages, the adversary's advantage in breaking the MAC should be no more than 2^{-64}. Then ϵ must be at most $1/2^{127}$.

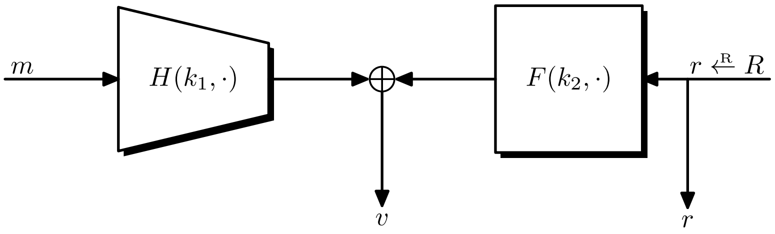


Figure 7.4: Carter-Wegman MAC signing algorithm

Our second MAC paradigm, called a Carter-Wegman MAC, maintains the same level of security as PRF(UHF) composition, but does so with a much larger value of ϵ. With the parameters in the example above, ϵ need only be $1/2^{64}$ and this can improve the speed of the hash function, especially for long messages. The downside is that the resulting tags are longer than those generated by a PRF(UHF) composition MAC of comparable security. In Exercise 7.5 we explore a different randomized MAC construction that achieves the same security as Carter-Wegman with the same ϵ, but with shorter tags.

The Carter-Wegman MAC is our first example of a randomized MAC system. The signing algorithm is randomized and there are many valid tags for every message.

To describe the Carter-Wegman MAC, first fix some large integer N and set $\mathcal{T} := \mathbb{Z}_N$, the group of size N where addition is defined "modulo N." We use a hash function H and a PRF F that output values in $\mathbb{Z}_N$:

- H is a keyed hash function defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{T})$,
- F is a PRF defined over $(\mathcal{K}_F, \mathcal{R}, \mathcal{T})$.

The Carter-Wegman MAC, denoted $\mathcal{I}_{\text{CW}}$, takes inputs in $\mathcal{M}$ and outputs tags in $\mathcal{R} \times \mathcal{T}$. It uses keys in $\mathcal{K}_H \times \mathcal{K}_F$. The **Carter-Wegman MAC derived from F and H** works as follows (see also Fig. 7.4):

- For key (k_1, k_2) and message m we define

```
$S\big(\ (k_1, k_2),\ m\ \big) :=$
    $r \xleftarrow{R} \mathcal{R}$
    $v \leftarrow H(k_1, m) + F(k_2, r) \quad \in \mathbb{Z}_N \quad //$ addition modulo $N$
    output $(r, v)$
```

- For key (k_1, k_2), message m, and tag (r, v) we define

```
$V\big(\ (k_1, k_2),\ m,\ (r, v)\ \big) :=$
    $v^* \leftarrow H(k_1, m) + F(k_2, r) \quad \in \mathbb{Z}_N \quad //$ addition modulo $N$
    if $v = v^*$ output accept; otherwise output reject
```

The Carter-Wegman signing algorithm uses a randomizer $r \in \mathcal{R}$. As we will see, the set $\mathcal{R}$ needs to be sufficiently large so that the probability that two tags use the same randomizer is negligible.

An encrypted UHF MAC. The Carter-Wegman MAC can be described as an encryption of the output of a hash function. Indeed, let $\mathcal{E} = (E, D)$ be the cipher

$$E(k, m) := \big\{ r \xleftarrow{\text{R}} \mathcal{R},\ \text{output } (r,\ m + F\big(k, r)\big) \big\} \qquad \text{and} \qquad D\big(k, (r, c)\big) := c - F(k, r)$$

where F is a PRF defined over $(\mathcal{K}_F, \mathcal{R}, \mathcal{T})$. This cipher is CPA secure when F is a secure PRF as shown in Example 5.2. Then the Carter-Wegman MAC can be written as:

$$S\big((k_1, k_2),\ m\big) := E(k_2,\ H(k_1, m))$$
$$V\big((k_1, k_2),\ m,\ t\big) := \begin{cases} \textsf{accept} & \text{if } D(k_2, t) = H(k_1, m), \\ \textsf{reject} & \text{otherwise.} \end{cases}$$

which we call the **encrypted UHF MAC system derived from $\mathcal{E}$ and H.**

Why encrypt the output of a hash function? Recall that in the PRF(UHF) composition MAC, if the adversary finds two messages m_1, m_2 that collide on the hash function (i.e., $H(k_1, m_1) = H(k_1, m_2)$) then the MAC for m_1 is the same as the MAC for m_2. Therefore, by requesting the tags for many messages the adversary can identify messages m_1 and m_2 that collide on the hash function (assuming collisions on the PRF are unlikely). A collision m_1, m_2 on the UHF can reveal information about the hash function key k_1 that may completely break the MAC. To prevent this we must use an ϵ-UHF with a sufficiently small ϵ to ensure that with high probability the adversary will never find a hash function collision. In contrast, by encrypting the output of the hash function with a CPA secure cipher, we prevent the adversary from learning when a hash function collision occurred: the tags for m_1 and m_2 are different, with high probability, even if $H(k_1, m_1) = H(k_1, m_2)$. This lets us maintain security with a much smaller ϵ.

The trouble is that the encrypted UHF MAC is not generally secure even when (E, D) is CPA secure and H is an ϵ-UHF. For example, we show in Remark 7.5 below that the Carter-Wegman MAC is insecure when the hash function H is instantiated with H_{poly}. To obtain a secure Carter-Wegman MAC, we strengthen the hash function H and require that it satisfy a stronger property called difference unpredictability defined below. Exercise 9.16 explores other aspects of the encrypted UHF MAC.

Security of the Carter-Wegman MAC. To prove security of $\mathcal{I}_{\text{CW}}$ we need the hash function H to satisfy a stronger property than universality (UHF). We refer to this stronger property as **difference unpredictability**. Roughly speaking, it means that for any two distinct messages, it is hard to predict the difference (in $\mathbb{Z}_N$) of their hashes. As usual, a game:

Attack Game 7.3 (difference unpredictability). For a keyed hash function H defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, where $\mathcal{T} = \mathbb{Z}_N$, and a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger picks a random $k \xleftarrow{\text{R}} \mathcal{K}$ and keeps k to itself.
- $\mathcal{A}$ outputs two distinct messages $m_0, m_1 \in \mathcal{M}$ and a value $\delta \in \mathcal{T}$.

We say that $\mathcal{A}$ wins the game if $H(k, m_1) - H(k, m_0) = \delta$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\text{DUF}\mathsf{adv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 7.5. *Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$,*

- *We say that H is an **ϵ-bounded difference unpredictable function**, or **ϵ-DUF**, if $\text{DUF}\mathsf{adv}[\mathcal{A}, H] \le \epsilon$ for all adversaries $\mathcal{A}$ (even inefficient ones).*

- *We say that H is a* ***statistical DUF*** *if it is an ϵ-DUF for some negligible ϵ.*
- *We say that H is a* ***computational DUF*** *if* $\text{DUF}\mathsf{adv}[\mathcal{A}, H]$ *is negligible for all efficient adversaries $\mathcal{A}$.*

Remark 7.3. Note that as we have defined a DUF, the digest space $\mathcal{T}$ must be of the form $\mathbb{Z}_N$ for some integer N. We did this to keep things simple. More generally, one can define a notion of difference unpredictability for a keyed hash function whose digest space comes equipped with an appropriate difference operator (in the language of abstract algebra, $\mathcal{T}$ should be an *abelian group*). Besides $\mathbb{Z}_N$, another popular digest space is the set of all n-bit strings, $\{0,1\}^n$, with the XOR used as the difference operator. In this setting, we use the terms **ϵ-XOR-DUF** and statistical/computational **XOR-DUF** to correspond to the terms ϵ-DUF and statistical/computational DUF. □

When H is a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, an alternative characterization of the ϵ-DUF property is the following:

> *for every pair of distinct messages $m_0, m_1 \in \mathcal{M}$, and every $\delta \in \mathcal{T}$, the following inequality holds:* $\Pr[H(k, m_1) - H(k, m_0) = \delta] \leq \epsilon$*. Here, the probability is over the random choice of $k \in \mathcal{K}$.*

Clearly if H is an ϵ-DUF then H is also an ϵ-UHF: a UHF adversary can be converted into a DUF adversary that wins with the same probability (just set $\delta = 0$).

We give a simple example of a statistical DUF that is very similar to the hash function H_{poly} defined in equation (7.3). Recall that H_{poly} is a UHF defined over $(\mathbb{Z}_p,\ (\mathbb{Z}_p)^{\leq \ell},\ \mathbb{Z}_p)$. It is clearly not a DUF: for $a \in \mathbb{Z}_p$ set $m_0 := (a)$ and $m_1 := (a+1)$ so that both m_0 and m_1 are tuples over $\mathbb{Z}_p$ of length 1. Then for every key k, we have

$$H_{\text{poly}}(k, m_1) - H_{\text{poly}}(k, m_0) = (k + a + 1) - (k + a) = 1$$

which lets the attacker win the DUF game.

A simple modification to H_{poly} yields a good DUF. For a message $m = (a_1, a_2, \ldots, a_v) \in \mathbb{Z}_p^{\leq \ell}$ and key $k \in \mathbb{Z}_p$ define a new hash function $H_{\text{xpoly}}(k, m)$ as:

$$H_{\text{xpoly}}(k, m) := k \cdot H_{\text{poly}}(k, m) = k^{v+1} + a_1 k^v + a_2 k^{v-1} + \cdots + a_v k \in \mathbb{Z}_p. \tag{7.23}$$

Lemma 7.8. *The function H_{xpoly} over $(\mathbb{Z}_p,\ (\mathbb{Z}_p)^{\leq \ell},\ \mathbb{Z}_p)$ defined in (7.23) is an $(\ell+1)/p$-DUF.*

Proof. Consider two distinct messages $m_0 = (a_1, \ldots, a_u)$ and $m_1 = (b_1, \ldots, b_v)$ in $(\mathbb{Z}_p)^{\leq \ell}$ and an arbitrary value $\delta \in \mathbb{Z}_p$. We want to show that $\Pr[H_{\text{xpoly}}(k, m_1) - H_{\text{xpoly}}(k, m_0) = \delta] \leq (\ell+1)/p$, where the probability is over the random choice of key k in $\mathbb{Z}_p$. Just as in the proof of Lemma 7.2, the inputs m_0 and m_1 define two polynomials $f(X)$ and $g(X)$ in $\mathbb{Z}_p[X]$, as in (7.4). However, $H_{\text{xpoly}}(k, m_1) - H_{\text{xpoly}}(k, m_0) = \delta$ holds if and only if k is root of the polynomial $X(g(X) - f(X)) - \delta$, which is a nonzero polynomial of degree at most $\ell + 1$, and so has at most $\ell + 1$ roots in $\mathbb{Z}_p$. Thus, the chances of choosing such a k is at most $(\ell+1)/p$. □

Remark 7.4. We can modify H_{xpoly} to operate on n-bit blocks by doing all arithmetic in the finite field $\text{GF}(2^n)$ instead of $\mathbb{Z}_p$. The exact same analysis as in Lemma 7.8 shows that the resulting hash function is an $(\ell+1)/2^n$-XOR-DUF. □

We now turn to the security analysis of the Carter-Wegman construction.

Theorem 7.9 (Carter-Wegman security). *Let F be a secure PRF defined over $(\mathcal{K}_F, \mathcal{R}, \mathcal{T})$ where $|\mathcal{R}|$ is super-poly. Let H be a computational DUF defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{T})$. Then the Carter-Wegman MAC $\mathcal{I}_{\text{CW}}$ derived from F and H is a secure MAC.*

> *In particular, for every MAC adversary $\mathcal{A}$ that attacks $\mathcal{I}_{\text{CW}}$ as in Attack Game 6.1, there exist a PRF adversary $\mathcal{B}_F$ and a DUF adversary $\mathcal{B}_H$, which are elementary wrappers around $\mathcal{A}$, such that*
> $$\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \le \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \text{DUF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{Q^2}{2|\mathcal{R}|} + \frac{1}{|\mathcal{T}|}. \tag{7.24}$$

Remark 7.5. To understand why H needs to be a DUF, let us suppose for a minute that it is not. In particular, suppose it was easy to find distinct $m_0, m_1 \in \mathcal{M}$ and $\delta \in \mathcal{T}$ such that $H(k_1, m_1) = H(k_1, m_0) + \delta$, without knowledge of k_1. The adversary could then ask for the tag on the message m_0 and obtain (r, v) where $v = H(k_1, m_0) + F(k_2, r)$. Since

$$v = H(k_1, m_0) + F(k_2, r) \quad \Longrightarrow \quad v + \delta = H(k_1, m_1) + F(k_2, r),$$

the tag $(r, v + \delta)$ is a valid tag for m_1. Therefore, $\big(m_1,\ (r, v + \delta)\big)$ is an existential forgery on $\mathcal{I}_{\text{CW}}$. This shows that the Carter-Wegman MAC is easily broken when the hash function H is instantiated with H_{poly}. □

Remark 7.6. We also note that the term $Q^2/2|R|$ in (7.24) corresponds to the probability that two signing queries generate the same randomizer. In fact, if such a collision occurs, Carter-Wegman may be completely broken for certain DUFs (including H_{xpoly}) — see Exercises 7.13 and 7.14. □

Proof idea. Let $\mathcal{A}$ be an efficient MAC adversary that plays Attack Game 6.1 with respect to $\mathcal{I}_{\text{CW}}$. We derive an upper bound on $\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}]$. As usual, we first replace the underlying secure PRF F with a truly random function $f \in \text{Funs}[\mathcal{R}, \mathcal{T}]$ and argue that this doesn't change the adversary's advantage much. We then show that only three things can happen that enable the adversary to generate a forged message-tag pair and that the probability for each of those is small:

1. The challenger might get unlucky and choose the same randomizer $r \in \mathcal{R}$ to respond to two separate signing queries. This happens with probability at most $Q^2/(2|\mathcal{R}|)$.

2. The adversary might output a MAC forgery $\big(m, (r, v)\big)$ where $r \in \mathcal{R}$ is a fresh randomizer that was never used to respond to $\mathcal{A}$'s signing queries. Then $f(r)$ is independent of $\mathcal{A}$'s view and therefore the equality $v = H(k_1, m) + f(r)$ will hold with probability at most $1/|\mathcal{T}|$.

3. Finally, the adversary could output a MAC forgery $\big(m, (r, v)\big)$ where $r = r_j$ for some uniquely determined signed message-tag pair $(m_j, (r_j, v_j))$. But then

   $$v_j = H(k_1, m_j) + f(r_j) \quad \text{and} \quad v = H(k_1, m) + f(r_j).$$

   By subtracting the right equality from the left, the $f(r_j)$ term cancels, and we obtain

   $$v_j - v = H(k_1, m_j) - H(k_1, m).$$

   But since H is an computational DUF, the adversary can find such a relation with only negligible probability. □

Proof. We make the intuitive argument above rigorous by considering $\mathcal{A}$'s behavior in three closely related games. For $j = 0, 1, 2$, we define W_j to be the event that $\mathcal{A}$ wins Game j. Game 0 will be identical to the original MAC attack game with respect to $\mathcal{I}$. We then slightly modify each game in turn and argue that the attacker will not detect these modifications. Finally, we argue that $\Pr[W_3]$ is negligible, which will prove that $\Pr[W_0]$ is negligible, as required.

Game 0. We begin by describing in detail the challenger in the MAC Attack Game 6.1 with respect to $\mathcal{I}_{\text{CW}}$. In this description, we assume that the actual number of signing queries made by the adversary in a particular execution of the attack game is s, which is at most Q.

```
Initialization:
    $k_1 \xleftarrow{R} \mathcal{K}_H,\ \ k_2 \xleftarrow{R} \mathcal{K}_F$
    $r_1, \ldots, r_Q \xleftarrow{R} \mathcal{R}$   // prepare randomizers needed for the game
upon receiving the $i$th signing query $m_i \in \mathcal{M}$ (for $i = 1, \ldots, s$) do:
    $v_i \leftarrow H(k_1, m_i) + F(k_2, r_i) \in \mathcal{T}$
    send $(r_i, v_i)$ to the adversary
upon receiving a forgery attempt $(m, (r, v)) \notin \{(m_1, (r_1, v_1)), \ldots, (m_s, (r_s, v_s))\}$ do:
    if $v = H(k_1, m) + F(k_2, r)$
        then output "win"
        else output "lose"
```

Then, by construction

$$\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] = \Pr[W_0]. \tag{7.25}$$

Game 1. We next play the usual "PRF card," replacing the function $F(k_2, \cdot)$ by a truly random function f in $\text{Funs}[\mathcal{R}, \mathcal{T}]$, which we implement as a faithful gnome (as in Section 4.4.2). Our challenger in Game 1 thus works as follows:

```
    Initialization:
        $k_1 \xleftarrow{R} \mathcal{K}_H$
        $r_1, \ldots, r_Q \xleftarrow{R} \mathcal{R}$   // prepare randomizers needed for the game
        $u'_0, u'_1, \ldots, u'_Q \xleftarrow{R} \mathcal{T}$   // prepare default $f$ outputs
    upon receiving the $i$th signing query $m_i \in \mathcal{M}$ (for $i = 1, \ldots, s$) do:
        $u_i \leftarrow u'_i$
(1)     if $r_i = r_j$ for some $j < i$ then $u_i \leftarrow u_j$
        $v_i \leftarrow H(k_1, m_i) + u_i \in \mathcal{T}$
        send $(r_i, v_i)$ to the adversary
    upon receiving a forgery attempt $(m, (r, v)) \notin \{(m_1, (r_1, v_1)), \ldots, (m_s, (r_s, v_s))\}$ do:
(2)     if $r = r_j$ for some $j = 1, \ldots, s$
            then $u \leftarrow u_j$
            else $u \leftarrow u'_0$
        if $v = H(k_1, m) + u$
            then output "win"
            else output "lose"
```

For $i = 1, \ldots, Q$, the value u'_i is chosen in advance to be the default, random value for $u_i = f(r_i)$. The tests at the lines marked (1) and (2) ensure that our gnome is faithful, i.e., that we emulate a function in Funs$[\mathcal{R}, \mathcal{T}]$. At (2), if the value $u = f(r)$ has already been defined, we use that value; otherwise, we use the fresh random value u_0' for u.

As usual, one can show that there is a PRF adversary $\mathcal{B}_F$, just as efficient as $\mathcal{A}$, such that:

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] \tag{7.26}$$

Game 2. We make our gnome forgetful. We do this by deleting the line marked (1) in the challenger. In addition, we insert the following special test before the line marked (2):

if $r_i = r_j$ for some $1 \leq i < j \leq s$ then output "lose" (and stop)

Let Z be the event that $r_i = r_j$ for some $1 \leq i < j \leq Q$. By the union bound we know that $\Pr[Z] \leq Q^2/(2|\mathcal{R}|)$. Moreover, if Z does not happen, then Games 1 and 2 proceed identically. Therefore, by the Difference Lemma (Theorem 4.7), we obtain

$$\big|\Pr[W_2] - \Pr[W_1]\big| \leq \Pr[Z] \leq Q^2/(2|\mathcal{R}|) \tag{7.27}$$

To bound $\Pr[W_2]$, we decompose W_2 into two events:

- W_2': $\mathcal{A}$ wins in Game 2 and $r = r_j$ for some $j = 1, \ldots, s$;
- W_2'': $\mathcal{A}$ wins in Game 2 and $r \neq r_j$ for all $j = 1, \ldots, s$.

Then $W_2 = W_2' \cup W_2''$, and since these events are disjoint we know that

$$\Pr[W_2] = \Pr[W_2'] + \Pr[W_2'']. \tag{7.28}$$

We analyze these two events separately. Consider W_2'' first. If this happens, then $u = u_0'$ and $v = u + H(k_1, m)$; that is, $u_0' = v - H(k_1, m)$. But since u_0' and $v - H(k_1, m)$ are independent, this happens with probability $1/|\mathcal{T}|$. So we have

$$\Pr[W_2''] \leq 1/|\mathcal{T}|. \tag{7.29}$$

Next, consider W_2'. Our goal here is to show that

$$\Pr[W_2'] \leq \text{DUF}\mathsf{adv}[\mathcal{B}_H, H] \tag{7.30}$$

for a DUF adversary $\mathcal{B}_H$ that is just as efficient as $\mathcal{A}$. To this end, consider what happens if $\mathcal{A}$ wins in Game 2 and $r = r_j$ for some $j = 1, \ldots, s$. Since $\mathcal{A}$ wins, and because of the special test that we added above the line marked (2), the values $r_1, \ldots, r_s$ are distinct, and so there can be only one such index j, and $u = u_j$. Therefore, we have the following two equalities:

$$v_j = H(k_1, m_j) + u_j \qquad \text{and} \qquad v = H(k_1, m) + u_j;$$

subtracting, we obtain

$$v_j - v = H(k_1, m_j) - H(k_1, m). \tag{7.31}$$

We claim that $m \neq m_j$. Indeed, if $m = m_j$, then (7.31) would imply $v = v_j$, which would imply $(m, (r, v)) = (m_j, (r_j, v_j))$; however, this is impossible, since we require that $\mathcal{A}$ does not submit a previously signed pair as a forgery attempt.

So, if W_2' occurs, we have $m \neq m_j$ and the equality (7.31) holds. But observe that in Game 2, the challenger's responses are completely independent of k_1, and so we can easily convert $\mathcal{A}$ into a DUF adversary $\mathcal{B}_H$ that succeeds with probability at least $\Pr[W_2']$ in Attack Game 7.3. Adversary $\mathcal{B}_H$ works as follows: it interacts with $\mathcal{A}$, simulating the challenger in Game 2 by simply responding to each signing query with a random pair $(r_i, v_i) \in \mathcal{R} \times \mathcal{T}$; when $\mathcal{A}$ outputs its forgery attempt $(m, (r, v))$, our $\mathcal{B}_H$ determines if $r = r_j$ and $m \neq m_j$ for some $j = 1, \ldots, s$; if so, $\mathcal{B}_H$ outputs the triple $(m_j, m, v_j - v)$. Now (7.31) shows that $\mathcal{B}_H$ succeeds in attacking H as a DUF whenever event W_2' happens. The bound in (7.30) now follows.

The theorem follows from (7.25)–(7.30). □

7.4.1 Using Carter-Wegman with polynomial UHFs

If we want to use the Carter-Wegman construction with a polynomial-based DUF, such as H_{xpoly}, then we have to make an adjustment so that the digest space of the hash function is equal to the *output* space of the PRF. Again, the issue is that our example H_{xpoly} has outputs in $\mathbb{Z}_p$, while for typical implementations, the PRF will have outputs that are n-bit blocks.

Similarly to what we did in Section 7.3.2, we can choose p to be a prime that is just a little bit bigger than 2^n. This also allows us to view the inputs to the hash as n-bit blocks. Part (b) of Exercise 7.23 shows how this can be done. One can also use a prime p that is a bit smaller than 2^n (see part (a) of Exercise 7.22), although this is less convenient, because inputs to the hash will have to be broken up into blocks of size less than n. Alternatively, we can use a variant of H_{xpoly} where all arithmetic is done in the finite field $\text{GF}(2^n)$, as discussed in Remark 7.4.

7.5 Nonce-based MACs

In the Carter-Wegman construction (Section 7.4), the only essential property we need for the randomizers is that they are distinct. This motivates the study of nonce-based MACs, which are the analogue of nonce-based encryption (Section 5.5). Not only can this approach reduce the size of the tag, it can also improve security.

A **nonce-based MAC** is similar to an ordinary MAC and consists of a pair of *deterministic* algorithms S and V for signing and verifying tags. However, these algorithms take an additional input $\mathcal{N}$ called a nonce that lies in a **nonce-space** $\mathcal{N}$. Algorithms S and V work as follows:

- S takes as input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a nonce $\mathcal{N} \in \mathcal{N}$. It outputs a tag $t \in \mathcal{T}$.
- V takes as input four values $k, m, t, \mathcal{N}$, where k is a key, m is a message, t is a tag, and $\mathcal{N}$ is a nonce. It outputs either accept or reject.

We say that the nonce-based MAC is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T}, \mathcal{N})$. As usual, we require that tags generated by S are always accepted by V, *as long as both are given the same nonce.* The MAC must satisfy the following **correctness property**: for all keys k, all messages m, and all nonces $\mathcal{N} \in \mathcal{N}$:

$$V\big(k,\ m,\ S(k,\ m,\ \mathcal{N}),\ \mathcal{N}\ \big) = \mathsf{accept}.$$

Just as in Section 5.5, in order to guarantee security, the sender should avoid using the same nonce twice (on the same key). If the sender can maintain state then a nonce can be implemented using a simple counter. Alternatively, nonces can be chosen at random, so long as the nonce space is large enough to ensure that the probability of generating the same nonce twice is negligible.

7.5.1 Secure nonce-based MACs

Nonce-based MACs must be existentially unforgeable under a chosen message attack when the adversary chooses the nonces. The adversary, however, must never request a tag using a previously used nonce. This captures the idea that nonces can be chosen arbitrarily, as long as they are never reused. Nonce-based MAC security is defined using the following game.

Attack Game 7.4 (nonce-based MAC security). For a given nonce-based MAC system $\mathcal{I} = (S, V)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T}, \mathcal{N})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger picks a random $k \xleftarrow{R} \mathcal{K}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots,$ the ith signing query consists of a pair $(m_i, \mathcal{N}_i)$ where $m_i \in \mathcal{M}$ and $\mathcal{N}_i \in \mathcal{N}$. We require that $\mathcal{N}_i \neq \mathcal{N}_j$ for all $j < i$. The challenger computes $t_i \xleftarrow{R} S(k, m_i, \mathcal{N}_i)$, and gives t_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ sends outputs a candidate forgery triple $(m, t, \mathcal{N}) \in \mathcal{M} \times \mathcal{T} \times \mathcal{N}$, where
$$(m, t, \mathcal{N}) \notin \{(m_1, t_1, \mathcal{N}_1), (m_2, t_1, \mathcal{N}_2), \ldots\}.$$

We say that $\mathcal{A}$ wins the game if $V(k, m, t, \mathcal{N}) = \mathsf{accept}$. We define $\mathcal{A}$'s advantage with respect to $\mathcal{I}$, denoted nMAC$\mathsf{adv}[\mathcal{A}, \mathcal{I}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 7.6. *We say that a nonce-based MAC system $\mathcal{I}$ is secure if for all efficient adversaries $\mathcal{A}$, the value* nMAC$\mathsf{adv}[\mathcal{A}, \mathcal{I}]$ *is negligible.*

Nonce-based Carter-Wegman MAC. The Carter-Wegman MAC (Section 7.4) can be recast as a nonce-based MAC: We simply view the randomizer $r \in \mathcal{R}$ as a nonce, supplied as an input to the signing algorithm, rather than a randomly generated value that is a part of the tag. Using the notation of Section 7.4, the MAC system is then

$$S\big((k_1, k_2),\ m,\ \mathcal{N}\big) := H(k_1, m) + F(k_2, \mathcal{N})$$

$$V\big((k_1, k_2),\ m,\ t,\ \mathcal{N}\big) := \begin{cases} \mathsf{accept} & \text{if } t = S\big((k_1, k_2),\ m,\ \mathcal{N}\big) \\ \mathsf{reject} & \text{otherwise} \end{cases}$$

We obtain the following security theorem, which is the nonce-based analogue of Theorem 7.9. The proof is essentially the same as the proof of Theorem 7.9.

Theorem 7.10. *With the notation of Theorem 7.9 we obtain the following bounds*

$$\text{nMAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \text{DUF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{1}{|\mathcal{T}|}.$$

This bound is much tighter than (7.24): the Q^2-term is gone. Of course, it is gone because we insist that the same nonce is never used twice. If nonces are, in fact, generated by the signer at random, then the Q^2-term returns; however, if the signer implements the nonce as a counter, then we avoid the Q^2-term — the only requirement is that the signer does not sign more than $|\mathcal{R}|$ values. See also Exercise 7.12 for a subtle point regarding the implementation of F.

Analogous to the discussion in Remark 7.6, when using nonce-based Carter-Wegman it is vital that the nonce is never re-used for different messages. If this happens, Carter-Wegman may be completely broken — see Exercises 7.13 and 7.14.

7.6 Unconditionally secure one-time MACs

In Chapter 2 we saw that the one-time pad gives unconditional security as long as the key is only used to encrypt a single message. Even algorithms that run in exponential time cannot break the semantic security of the one-time pad. Unfortunately, security is lost entirely if the key is used more than once.

In this section we ask the analogous question for MACs: can we build a "one-time MAC" that is unconditionally secure if the key is only used to provide integrity for a single message?

We can model one-time MACs using the standard MAC Attack Game 6.1 used to define MAC security. To capture the one-time nature of the MAC we allow the adversary to issue *only one* signing query. We denote the adversary's advantage in this restricted game by $\mathrm{MAC}_1\mathsf{adv}[\mathcal{A}, \mathcal{I}]$. This game captures the fact that the adversary sees only one message-tag pair and then tries to create an existential forgery using this pair.

Unconditional security means that $\mathrm{MAC}_1\mathsf{adv}[\mathcal{A}, \mathcal{I}]$ is negligible for all adversaries $\mathcal{A}$, even computationally unbounded ones. In this section, we show how to implement efficient and unconditionally secure one-time MACs using hash functions.

7.6.1 Pairwise unpredictable functions

Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. Intuitively, H is a **pairwise unpredictable function** if the following holds for a randomly chosen key $k \in \mathcal{K}$: given the value $H(k, m_0)$, it is hard to predict $H(k, m_1)$ for any $m_1 \neq m_0$. As usual, we make this definition rigorous using an attack game.

Attack Game 7.5 (pairwise unpredicability). For a keyed hash function H defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger picks a random $k \xleftarrow{\text{R}} \mathcal{K}$ and keeps k to itself.
- $\mathcal{A}$ sends a message $m_0 \in \mathcal{M}$ to the challenger, who responds with $t_0 = H(k, m_0)$.
- $\mathcal{A}$ outputs $(m_1, t_1) \in \mathcal{M} \times \mathcal{T}$, where $m_1 \neq m_0$.

We say that $\mathcal{A}$ wins the game if $t_1 = H(k, m_1)$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\mathrm{PUF}\mathsf{adv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 7.7. *We say that H is an ϵ-**bounded pairwise unpredictable function**, or ϵ-**PUF** for short, if $\mathrm{PUF}\mathsf{adv}[\mathcal{A}, H] \leq \epsilon$ for all adversaries $\mathcal{A}$ (even inefficient ones).*

It should be clear that if H is an ϵ-PUF, then H is also an ϵ-UHF; if, in addition, $\mathcal{T}$ is of the form $\mathbb{Z}_N$ (or is an abelian group as in Remark 7.3), then H is an ϵ-DUF.

7.6.2 Building unpredictable functions

So far we know that any ϵ-PUF is also an ϵ-DUF. The converse is not true (see Exercise 7.29). Nevertheless, we show that any ϵ-DUF can be tweaked so that it becomes an ϵ-PUF. This tweak increases the key size.

Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, where $\mathcal{T} = \mathbb{Z}_N$ for some N. We build a new hash function H' derived from H with the same input and output space as H. The key space, however, is $\mathcal{K} \times \mathcal{T}$. The function H' is defined as follows:

$$H'\big((k_1, k_2),\ m\big) = H(k_1, m) + k_2 \quad \in \mathcal{T} \tag{7.32}$$

Lemma 7.11. *If H is an ϵ-DUF, then H' is an ϵ-PUF.*

Proof. Let $\mathcal{A}$ attack H' as a PUF. In response to its query m_0, adversary $\mathcal{A}$ receives $t_0 := H(k_1, m_0) + k_2$. Observe that t_0 is uniformly distributed over $\mathcal{T}$, and is independent of k_1. Moreover, if $\mathcal{A}$'s prediction t_1 of $H(k_1, m_1) + k_2$ is correct, then $t_1 - t_0$ correctly predicts the difference $H(k_1, m_1) - H(k_1, m_0)$.

So we can define a DUF adversary $\mathcal{B}$ as follows: it runs $\mathcal{A}$, and when $\mathcal{A}$ submits its query m_0, $\mathcal{B}$ responds with a random $t_0 \in \mathcal{T}$; when $\mathcal{A}$ outputs (m_1, t_1), adversary $\mathcal{B}$ outputs $(m_0, m_1, t_1 - t_0)$. It is clear that

$$\mathrm{PUF}\mathsf{adv}[\mathcal{A}, H'] \le \mathrm{DUF}\mathsf{adv}[\mathcal{B}, H] \le \epsilon. \quad \square$$

Lemma 7.11 shows how to convert the function H_{xpoly}, defined in (7.23), into an $(\ell+1)/p$-PUF. We obtain the following keyed hash function defined over $(\mathbb{Z}_p^2, \mathbb{Z}_p^{\le \ell}, \mathbb{Z}_p)$:

$$H'_{\mathrm{xpoly}}\big((k_1, k_2), (a_1, \ldots, a_v)\big) := k_1^{v+1} + a_1 k_1^v + \cdots + a_v k_1 + k_2 \tag{7.33}$$

for $0 \le v \le \ell$. This function is an $(\ell+1)/p$-PUF.

7.6.3 From PUFs to unconditionally secure one-time MACs

We now return to the problem of building unconditionally secure one-time MACs. In fact, PUFs are just the right tool for the job.

Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. We can use H to define the MAC system $\mathcal{I} = (S, V)$ **derived from** H:

$$S(k,\ m) := H(k,\ m);$$

$$V(k,\ m,\ t) := \begin{cases} \mathsf{accept} & \text{if } H(k, m) = t, \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

The following theorem shows that PUFs are the MAC analogue of the one-time pad, since both provide unconditional security for one time use. The proof is immediate from the definitions.

Theorem 7.12. *Let H be an ϵ-PUF and let $\mathcal{I}$ be the MAC system derived from H. Then for all adversaries $\mathcal{A}$ (even inefficient ones), we have* $\mathrm{MAC}_1\mathsf{adv}[\mathcal{A}, \mathcal{I}] \le \epsilon$.

The PUF construction in Section 7.6.2 is very similar to the Carter-Wegman MAC. The only difference is that the PRF is replaced by a truly random pad k_2. Hence, Theorem 7.12 shows that the Carter-Wegman MAC with a truly random pad is an unconditionally secure one-time MAC.

7.7 A fun application: timing attacks

To be written.

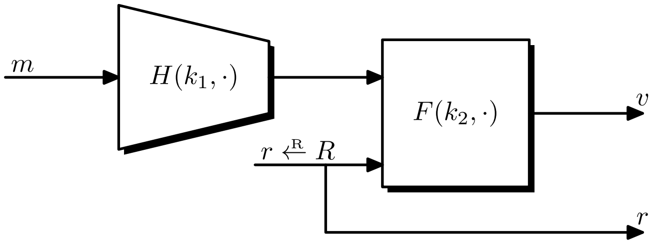


Figure 7.5: Randomized PRF(UHF) composition: MAC signing

7.8 Notes

Citations to the literature to be added.

7.9 Exercises

7.1 (Using H_{poly} with power-of-2 modulus). We can adapt the definition of H_{poly} in (7.3) so that instead of working in $\mathbb{Z}_p$ we work in $\mathbb{Z}_{2^n}$ (i.e., work modulo 2^n). Show that this version of H_{poly} is not a good UHF, and in particular an attacker can find two messages m_0, m_1 each of length two blocks that are guaranteed to collide.

7.2 (Non-adaptively secure PRFs are computational UHFs). Show that if F is a secure PRF against non-adaptive adversaries (see Exercise 4.6), and the size of the output space of F is super-poly, then F is a computational UHF.

Note: Using the result of Exercise 6.13, this gives another proof that CBC is a computational UHF.

7.3 (On the alternative characterization of the ϵ-UHF property). Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. Suppose that for some pair of distinct messages m_0 and m_1, we have $\Pr[H(k, m_0) = H(k, m_1)] > \epsilon$, where the probability is over the random choice of $k \in \mathcal{K}$. Give an adversary $\mathcal{A}$ that wins Attack Game 7.1 with probability greater than ϵ. Your adversary is not allowed to just have the values m_0 and m_1 "hardwired" into its code, but it may be *very* inefficient.

7.4 (MAC(UHF) composition is insecure). The PRF(UHF) composition shows that a UHF can extend the input domain of a specific type of MAC, namely a MAC that is itself a PRF. Show that this construction cannot be extended to arbitrary MACs. That is, exhibit a secure MAC $\mathcal{I} = (S, V)$ and a computational UHF H for which the MAC(UHF) composition $\mathcal{I}' = (S', V')$ where $S'((k_1, k_2), m) = S(k_2, H(k_1, m))$ is insecure. In your design, you may assume the existence of a secure PRF defined over any convenient spaces. Then show how to "sabotage" this PRF so that it remains a secure MAC, but the MAC(UHF) composition becomes insecure.

7.5 (Randomized PRF(UHF) composition). In this exercise we develop a randomized variant of PRF(UHF) composition that provides better security with little impact on the running time. Let H be a keyed hash function defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{X})$ and let F be a PRF defined over $(\mathcal{K}_F,\ \mathcal{R} \times$

$\mathcal{X}$, $\mathcal{T}$). Define the **randomized PRF(UHF) system** $\mathcal{I} = (S, V)$ as follows: for key (k_1, k_2) and message $m \in \mathcal{M}$ define

$$S\big((k_1,k_2),\ m\big) := \big\{r \xleftarrow{\text{R}} \mathcal{R},\ x \leftarrow H(k_1,m),\ v \leftarrow F\big(k_2,(r,x)\big),\ \text{output } (r,v)\big\} \qquad \text{(see Fig. 7.5)}$$

$$V\big((k_1,k_2),\ m,\ (r,v)\big) := \begin{cases} \textsf{accept} & \text{if } x \leftarrow H(k_1,m),\ v = F\big(k_2,(r,x)\big) \\ \textsf{reject} & \text{otherwise.} \end{cases}$$

This MAC is defined over $(\mathcal{K}_F \times \mathcal{K}_H,\ \mathcal{M},\ \mathcal{R} \times \mathcal{T})$. The tag size is a little larger than in deterministic PRF(UHF) composition, but signing and verification time is about the same.

(a) Suppose $\mathcal{A}$ is a MAC adversary that plays Attack Game 6.1 with respect to $\mathcal{I}$ and issues at most Q queries. Show that there exists a PRF adversary $\mathcal{B}_F$ and UHF adversaries $\mathcal{B}_H$ and $\mathcal{B}'_H$, which are elementary wrappers around $\mathcal{A}$, such that

$$\begin{aligned} \text{MAC}\mathsf{adv}[\mathcal{A},\mathcal{I}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_F,F] + \text{UHF}\mathsf{adv}[\mathcal{B}_H,H] + \frac{Q^2}{2|\mathcal{R}|}\text{UHF}\mathsf{adv}[\mathcal{B}'_H,H] \\ + \frac{Q^2}{2|\mathcal{R}||\mathcal{T}|} + \frac{1}{|\mathcal{T}|}. \end{aligned} \qquad (7.34)$$

Discussion: When H is an ϵ-UHF let us set $\epsilon = 1/|\mathcal{T}|$ and $|\mathcal{R}| = Q^2/2$ so that the right most four terms in (7.34) are all equal. Then (7.34) becomes simply

$$\text{MAC}\mathsf{adv}[\mathcal{A},\mathcal{I}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_F,F] + 4\epsilon. \qquad (7.35)$$

Comparing to deterministic PRF(UHF) composition, the error term $\epsilon \cdot Q^2/2$ in (7.19) is far worse than in (7.35). This means that for the same parameters, randomized PRF(UHF) composition security is preserved for far many more queries than for deterministic PRF(UHF) composition.

In the Carter-Wegman MAC to get an error bound as in (7.35) we must set $|\mathcal{R}|$ to $|Q|^2/\epsilon$ in (7.24). In randomized PRF(UHF) composition we only need $|\mathcal{R}| = |Q|^2$ and therefore tags in randomized PRF(UHF) are shorter than in Carter-Wegman for the same security and the same ϵ.

(b) Rephrase the MAC system $\mathcal{I}$ as a nonce-based MAC system (as in Section 7.5). What are the concrete security bounds for this system?

Observe that if the nonce is accidentally re-used, or even always set to the same value, then the MAC system $\mathcal{I}$ still provides some security: security degrades to the security of deterministic PRF(UHF) composition. We refer to this as **nonce re-use resistance**.

7.6 (One-key PRF(UHF) composition). This exercise analyzes a one-key variant of the PRF(UHF) construction. Let F be a PRF defined over $(\mathcal{K},\mathcal{X},\mathcal{Y})$ and let H be a keyed hash function defined over $(\mathcal{Y},\mathcal{M},\mathcal{X})$; in particular, the output space of F is equal to the key space of H, and the output space of H is equal to the input space of F. Let $x_0 \in \mathcal{X}$ be a public constant. Consider the PRF F' defined over $(\mathcal{K},\mathcal{M},\mathcal{Y})$ as follows:

$$F'(k,m) := F(k, H(k_0,m)), \quad \text{where} \quad k_0 := F(k,x_0).$$

This is the same as the usual PRF(UHF) composition, except that we use a single key k and use F to derive the key k_0 for H.

(a) Show that F' is a secure PRF assuming that F is a PRF, that H is a computational UHF, and that H satisfies a certain *preimage resistance* property, defined by the following game.

In this game, the adversary computes a message M and the challenger (independently) chooses a random hash key $k_0 \in \mathcal{K}$. The adversary wins the game if $H(k_0, M) = x_0$, where $x_0 \in \mathcal{X}$ is a constant, as above. We say that H is preimage resistant if every efficient adversary wins this game with only negligible probability.

Hint: Modify the proof of Theorem 7.7.

(b) Show that the cascade construction is preimage resistant, assuming the underlying PRF is a secure PRF.

Hint: This follows almost immediately from the fact that the cascade is a prefix-free PRF.

7.7 (XOR-DUFs). In Remark 7.3 we adapted the definition of DUF to a hash function whose digest space $\mathcal{T}$ is the set of all n-bit strings, $\{0,1\}^n$, with the XOR used as the difference operator.

(a) Show that the XOR-hash $F^{\oplus}$ defined in Section 7.2.3 is a computational XOR-DUF.

(b) Show that the CBC construction F_{CBC} defined in Section 6.4.1 is a computational XOR-DUF.

Hint: Use the fact that F_{CBC} is a prefix-free secure PRF (or, alternatively, the result of Exercise 6.13).

7.8 (Luby-Rackoff with an XOR-DUF). Show that the Luby-Rackoff construction (see Section 4.5) remains secure if the first round function $F(k_1, \cdot)$ is replaced by a computational XOR-DUF.

7.9 (Nonce-based CBC cipher with an XOR-DUF). Show that in the nonce-based CBC cipher (Section 5.5.3) the PRF that is applied to the nonce can be replaced by an XOR-DUF.

7.10 (Tweakable block ciphers). Continuing with Exercise 4.11, show that in the construction from part (c) the PRF can be replaced by an XOR-DUF. That is, prove that the following construction is a strongly secure tweakable block cipher:

$$E'\big((k_0, k_1), m, t\big) := \big\{p \leftarrow h(k_0, t);\ \ \text{output } p \oplus E(k_1,\ m \oplus p)\ \big\}$$
$$D'\big((k_0, k_1), c, t\big) := \big\{p \leftarrow h(k_0, t);\ \ \text{output } p \oplus D(k_1,\ c \oplus p)\ \big\}$$

Here (E, D) is a strongly secure block cipher defined over $(\mathcal{K}_0, \mathcal{X})$ and h is an XOR-DUF defined over $(\mathcal{K}_1, \mathcal{T}, \mathcal{X})$ where $\mathcal{X} := \{0,1\}^n$.

Discussion: **XTS mode**, used in disk encryption systems, is based on this tweakable block cipher. The tweak in XTS is a combination of i, the disk sector number, and j, the position of the block within the sector. The XOR-DUF used in XTS is defined as $h\big(k_0, (i,j)\big) := E(k_0, i) \cdot \alpha^j\ \in \text{GF}(2^n)$ where α is a fixed primitive element of $\text{GF}(2^n)$. XTS uses ciphertext stealing (Exercise 5.16) to handle sectors whose bit length is not a multiple of n.

7.11 (Carter-Wegman with verification queries: concrete security). Consider the security of the Carter-Wegman construction (Section 7.4) in an attack with verification queries (Section 6.2). Show that following concrete security result: for every MAC adversary $\mathcal{A}$ that attacks $\mathcal{I}_{\text{CW}}$ as in Attack Game 6.2, and which makes at most Q_{v} verification queries and at most Q_{s} signing queries, there exist a PRF adversary $\mathcal{B}_F$ and a DUF adversary $\mathcal{B}_H$, which are elementary wrappers around $\mathcal{A}$, such that

$$\text{MAC}^{\text{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + Q_{\text{v}} \cdot \text{DUF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{Q_{\text{s}}^2}{2|\mathcal{R}|} + \frac{Q_{\text{v}}}{|\mathcal{T}|}.$$

7.12 (Nonce-based Carter-Wegman: improved security bounds). In Section 7.5, we studied a nonce-based version of the Carter-Wegman MAC. In particular, in Theorem 7.10, we derived the security bound

$$\text{nMAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \text{DUF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{1}{|\mathcal{T}|},$$

and rejoiced in the fact that there were no Q^2-terms in this bound, where Q is a bound on the number of signing queries. Unfortunately, a common implementation of F is to use the encryption function of a block cipher $\mathcal{E}$ defined over $(\mathcal{K}, \mathcal{X})$, so $\mathcal{R} = \mathcal{X} = \mathcal{T} = \mathbb{Z}_N$. A straightforward application of the PRF switching lemma (see Theorem 4.4) gives us the security bound

$$\text{nMAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{BC}\mathsf{adv}[\mathcal{B}_{\mathcal{E}}, \mathcal{E}] + \frac{Q^2}{2N} + \text{DUF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{1}{N},$$

and a Q^2-term has returned! In particular, when $Q^2 \approx N$, this bound is entirely useless. However, one can obtain a better bound. Using the result of Exercise 4.25, show that assuming $Q^2 < N$, we have the following security bound:

$$\text{nMAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{BC}\mathsf{adv}[\mathcal{B}_{\mathcal{E}}, \mathcal{E}] + 2 \cdot \left(\text{DUF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{1}{N}\right).$$

7.13 (Carter-Wegman MAC falls apart under nonce re-use). Suppose that when using a nonce-based MAC, an implementation error causes the system to re-use a nonce more than once. Let us show that the nonce-based Carter-Wegman MAC falls apart if this ever happens.

(a) Consider the nonce-based Carter-Wegman MAC built from the hash function H_{xpoly}. Show that if the adversary obtains the tag on some one-block message m_1 using nonce $\mathcal{N}$ and the tag on a different one-block message m_2 using *the same* nonce $\mathcal{N}$, then the MAC system becomes insecure: the adversary can forge the MAC on any message of its choice with non-negligible probability.

(b) Consider the nonce-based Carter-Wegman MAC with an arbitrary hash function. Suppose that an adversary is free to re-use nonces at will. Show how to create an existential forgery.

Note: These attacks also apply to the randomized version of Carter-Wegman, if the signer is unlucky enough to generate the same randomizer $r \in \mathcal{R}$ more than once. Also, the attack in part (a) can be extended to work even if the messages are not single-block messages by using efficient algorithms for finding roots of polynomials over finite fields.

7.14 (Encrypted Carter-Wegman). Continuing with the previous exercise, we show how to make Carter-Wegman resistant to nonce re-use by encrypting the tag. To make things more concrete, suppose that H is an ϵ-DUF defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{X})$, where $\mathcal{X} = \mathbb{Z}_N$, and $\mathcal{E} = (E, D)$ is a secure block cipher defined over $(\mathcal{K}_{\mathcal{E}}, \mathcal{X})$. The encrypted Carter-Wegman nonce-based MAC system $\mathcal{I} = (S, V)$ has key space $\mathcal{K}_H \times \mathcal{K}_{\mathcal{E}}^2$, message space $\mathcal{M}$, tag space $\mathcal{X}$, nonce space $\mathcal{X}$, and is defined as follows:

- For key (k_1, k_2, k_3), message m, and nonce $\mathcal{N}$, we define

$$S((k_1, k_2, k_3), m, \mathcal{N}) := E(k_3,\ H(k_1, m) + E(k_2, \mathcal{N})\ )$$

- For key (k_1, k_2, k_3), message m, tag v, and nonce $\mathcal{N}$, we define

```
$V((k_1, k_2, k_3), m, v, \mathcal{N}) :=$
    $v^* \leftarrow E(k_3,\ H(k_1, m) + E(k_2, \mathcal{N})\ )$
    if $v = v^*$ output $\mathsf{accept}$; otherwise output $\mathsf{reject}$
```

(a) Show that assuming no nonces get re-used, this scheme is just as secure as Carter-Wegman. In particular, using the result of Exercise 7.12, show that for every adversary $\mathcal{A}$ that makes at most Q signing queries, where $Q^2 < N$, the probability that $\mathcal{A}$ produces an existential forgery is at most $\mathrm{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}] + 2(\epsilon + 1/N)$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$.

(b) Now suppose an adversary can re-use nonces at will. Show that for every such adversary $\mathcal{A}$ that makes at most Q signing queries, where $Q^2 < N$, the probability that $\mathcal{A}$ produces an existential forgery is at most $\mathrm{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}] + (Q + 1)^2\epsilon + 2/N$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$. Thus, while nonce re-use degrades security, it is not catastrophic.

Hint: Theorem 7.7 and Exercises 4.25 and 7.21 may be helpful.

7.15 (Composing UHFs). Let H_1 be a keyed hash function defined over $(\mathcal{K}_1, \mathcal{X}, \mathcal{Y})$. Let H_2 be a keyed hash function defined over $(\mathcal{K}_2, \mathcal{Y}, \mathcal{Z})$. Let H be the keyed hash function defined over $(\mathcal{K}_1 \times \mathcal{K}_2, \mathcal{X}, \mathcal{Z})$ as $H((k_1, k_2), x) := H_2(k_2, H(k_1, x))$.

(a) Show that if H_1 is an ϵ_1-UHF and H_2 is an ϵ_2-UHF, then H is an $(\epsilon_1 + \epsilon_2)$-UHF.

(b) Show that if H_1 is an ϵ_1-UHF and H_2 is an ϵ_2-DUF, then H is an $(\epsilon_1 + \epsilon_2)$-DUF.

7.16 (Variations on H_{poly}). Show that if p is prime and the input space is $\mathbb{Z}_p^\ell$ for some fixed (poly-bounded) value ℓ, then

(a) the function H_{fpoly} defined in (7.5) is an $(\ell - 1)/p$-UHF.

(b) the function H_{fxpoly} defined as

$$H_{\mathrm{fxpoly}}(k, (a_1, \ldots, a_\ell)) := k \cdot H_{\mathrm{fpoly}}(k, (a_1, \ldots, a_\ell)) = a_1 k^\ell + a_2 k^{v-1} + \cdots + a_\ell k \in \mathbb{Z}_p$$

is an (ℓ/p)-DUF.

7.17 (A DUF from an ideal permutation). Let $\pi : \mathcal{X} \to \mathcal{X}$ be an permutation where $\mathcal{X} := \{0, 1\}^n$. Define $H : \mathcal{X} \times \mathcal{X}^{\leq \ell} \to \mathcal{X}$ as the following keyed hash function:

```
$H(k, (a_1, \ldots, a_v)) :=$   $h \leftarrow k$
                                for $i \leftarrow 1$ to $v$ do:   $h \leftarrow \pi(a_i \oplus h)$
                                output $h$
```

Assuming 2^n is super-poly, show that H is a computational XOR-DUF (see Remark 7.3) in the ideal permutation model, where we model π as a random permutation Π (see Section 4.7).

We outline here one possible proof approach. The first idea is to use the same strategy that was used in the analysis of CBC in the proof of Theorem 6.3; indeed, one can see that the two constructions process message blocks in a very similar way. The second idea is to use the Domain Separation Lemma (Theorem 4.15) to streamline the proof.

Consider two games:

0. The original attack game: adversary makes a series of ideal permutation queries, which evaluate Π and Π^{-1} on points of the adversary's choice. Then the adversary submits two distinct messages m_0, m_1 to the challenger, along with a value δ, and hopes that $H(k, m_0) \oplus H(k, m_1) = \delta$.

1. Use the Domain Separation Lemma to split Π into many independent permutations. One is Π_{ip}, which is used to evaluate the ideal permutation queries. The others are of the form $\Pi_{\text{std},\alpha}$ for $\alpha \in \mathcal{X}_{>0}^{\leq \ell}$. These are used to perform the evaluations $H(k, m_0)$, $H(k, m_1)$: in the evaluation of $H(k, (a_1, \ldots, a_s))$, in the ith loop iteration in the hash algorithm, we use the permutation $\Pi_{\text{std},\alpha}$, where $\alpha = (a_1, \ldots, a_i)$. Now one just has to analyze the probability of separation failure.

Note that H is certainly not a secure PRF, even if we restrict ourselves to non-adaptive or prefix-free adversaries: given $H(k, m)$ for any message m, we can efficiently compute the key k.

7.18 (Optimal collision probability with shorter hash keys). For positive integer d, let $I_d := \{0, \ldots, d-1\}$ and $I_d^* := \{1, \ldots, d-1\}$.

(a) Let p be a prime, and let $N < p$ be a positive integer. Consider the keyed hash function H defined over $(I_p \times I_p^*,\ I_p,\ I_N)$ as follows: $H((k_0, k_1), a) := ((k_0 + ak_1) \bmod p) \bmod N$. Show that H is a $1/N$-UHF.

(b) While the construction in part (a) gives a UHF with "optimal" collision probability, the key space is unfortunately larger than the message space. Using the result of part (a), along with part (a) of Exercise 7.15, and the result of Exercise 7.16, you are to design a hash function with "nearly optimal" collision probability, but with much smaller keys.

In particular, let N and ℓ be positive integers. Let α be a number with $0 < \alpha < 1$. Design a $(1 + \alpha)/N$-UHF with message space $\{0, 1\}^\ell$ and output space I_N, where keys are bit strings of length $O(\log(N\ell/\alpha))$.

7.19 (Inner product hash). Let p be a prime.

(a) Consider the keyed hash function H defined over $(\mathbb{Z}_p^\ell, \mathbb{Z}_p^\ell, \mathbb{Z}_p)$ as follows:

$$H((k_1, \ldots, k_\ell), (a_1, \ldots, a_\ell)) := a_1k_1 + \cdots + a_\ell k_\ell.$$

Show that H is a $1/p$-DUF.

(b) Since multiplications can be much more expensive than additions, the following variant of the hash function in part (a) is sometimes preferable. Assume ℓ is even, and consider the keyed hash function H' defined over $(\mathbb{Z}_p^\ell, \mathbb{Z}_p^\ell, \mathbb{Z}_p)$ as follows:

$$H'((k_1, \ldots, k_\ell), (a_1, \ldots, a_\ell)) := \sum_{i=1}^{\ell/2} (a_{2i-1} + k_{2i-1})(a_{2i} + k_{2i}).$$

Show that H' is also a $1/p$-DUF.

(c) Although both H and H' are ϵ-DUFs with "optimal" ϵ values, the keys are unfortunately very large. Using a similar approach to part (b) of the previous exercise, design a $(1+\alpha)/p$-DUF with message space $\{0,1\}^{\ell}$ and output space $\mathbb{Z}_p$, where keys bit strings of length $O(\log(p\ell/\alpha))$.

7.20 (Division-free hash). This exercise develops a hash function that does not require any division or mod operations, which can be expensive. It can be implemented just using shifts and adds. For positive integer d, let $I_d := \{0, \dots, d-1\}$. Let n be a positive integer and set $N := 2^n$.

(a) Consider the keyed hash function H defined over $(I_{N^2}^{\ell}, I_N^{\ell}, \mathbb{Z}_N)$ as follows:

$$H((k_1, \dots, k_\ell), (a_1, \dots, a_\ell)) := [t]_N \in \mathbb{Z}_N, \quad \text{where } \; t := \big\lfloor \big(\; \big(\sum_i a_i k_i\big) \bmod N^2 \; \big) \; / \; N \big\rfloor.$$

Show that H is a $2/N$-DUF. Below in Exercise 7.31 we will see a minor variant of H that satisfies a stronger property, and in particular, is a $1/N$-DUF.

(b) Analogous to part (b) in the previous exercise, assume ℓ is even, and consider the keyed hash function H defined over $(I_{N^2}^{\ell}, I_N^{\ell}, \mathbb{Z}_N)$ as follows:

$$H'((k_1, \dots, k_\ell), (a_1, \dots, a_\ell)) := [t]_N \in \mathbb{Z}_N,$$

where

$$t := \big\lfloor \big(\; \big(\sum_{i=1}^{\ell/2} (a_{2i-1} + k_{2i-1})(a_{2i} + k_{2i})\big) \bmod N^2 \; \big) \; / \; N \big\rfloor.$$

Show that H' is a $2/N$-DUF.

7.21 (DUF to UHF conversion). Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathbb{Z}_N)$. We construct a new keyed hash function H', defined over $(\mathcal{K}, \mathcal{M} \times \mathbb{Z}_N, \mathbb{Z}_N)$ as follows: $H'(k, (m, x)) := H(k, m) + x$. Show that if H is an ϵ-DUF, then H' is an ϵ-UHF.

7.22 (DUF modulus switching). We will be working with DUFs with digest spaces $\mathbb{Z}_m$ for various m, and so to make things clearer, we will work with digest spaces that are plain old sets of integers, and state explicitly the modulus m, as in "an ϵ-DUF modulo m". For positive integer d, let $I_d := \{0, \dots, d-1\}$.

Let p and N be integers greater than 1. Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, I_p)$. Let H' be the keyed hash function defined over $(\mathcal{K}, \mathcal{M}, I_N)$ as follows: $H'(k, m) := H(k, m) \bmod N$.

(a) Show that if $p \le N/2$ and H is an ϵ-DUF modulo p, then H' is an ϵ-DUF modulo N.

(b) Suppose that $p \ge N$ and H is an ϵ-DUF modulo p. Show that H' is an ϵ'-DUF modulo N for $\epsilon' = 2(p/N + 1)\epsilon$. In particular, if $\epsilon = \alpha/p$, we can take $\epsilon' = 4\alpha/N$.

7.23 (More flexible output spaces). As in the previous exercise, we work with DUFs whose digest spaces are plain old sets of integers, but we explicitly state the modulus m. Again, for positive integer d, we let $I_d := \{0, \dots, d-1\}$.

Let $1 < N \le p$, where p is prime.

(a) H^*_{fxpoly} is the keyed hash function defined over (I_p, I_N^{ℓ}, I_N) as follows:

$$H^*_{\text{fxpoly}}(k, (a_1, \dots, a_\ell)) := \Big((a_1 k^{\ell} + \cdots + a_\ell k) \bmod p \Big) \bmod N.$$

Show that H^*_{fxpoly} is a $4\ell/N$-DUF modulo N.

(b) H^*_{xpoly} is the keyed hash function defined over $(I_p, I_N^{\leq \ell}, I_N)$ as follows:

$$H^*_{\text{xpoly}}(k, (a_1, \ldots, a_v)) := \Big((k^{v+1} + a_1 k^v + \cdots + a_v k) \bmod p \Big) \bmod N.$$

Show that H^*_{xpoly} is a $4(\ell+1)/N$-DUF modulo N.

(c) H^*_{fpoly} is the keyed hash function defined over (I_p, I_N^{ℓ}, I_N) as follows:

$$H^*_{\text{fpoly}}(k, (a_1, \ldots, a_\ell)) := \Big(\big((a_1 k^{\ell-1} + \cdots + a_{\ell-1} k) \bmod p \big) + a_\ell \Big) \bmod N.$$

Show that H^*_{fpoly} is a $4(\ell-1)/N$-UHF.

(d) H^*_{poly} is the keyed hash function is defined over $(I_p, I_N^{\leq \ell}, I_N)$ as follows:

$$H^*_{\text{poly}}(k, (a_1, \ldots, a_v)) := \Big(\big((k^v + a_1 k^{v-1} + \cdots + a_{v-1} k) \bmod p \big) + a_v \Big) \bmod N.$$

for $v > 0$, and for zero-length messages, it is defined to be the constant 0. Show that H^*_{poly} is a $4\ell/N$-UHF.

Hint: All of these results follow easily from the previous two exercises, except that the analysis in part (d) requires that zero-length messages are treated separately.

7.24 (Be careful: reducing at the wrong time can be dangerous). With notation as in the previous exercise, show that if $(3/2)N \leq p < 2N$, the keyed hash function H defined over (I_p, I_N^2, I_N) as

$$H(k, (a, b)) := ((ak + b) \bmod p) \bmod N$$

is not a (1/3)-UHF. Contrast this function with that in part (c) of the previous exercise with $\ell = 2$.

7.25 (A PMAC$_0$ ***alternative).*** Again, for positive integer d, let $I_d := \{0, \ldots, d-1\}$. Let $N = 2^n$ and let p be a prime with $N/4 < p < N/2$. Let H be the hash function defined over $(I_{N/4}, I_N \times I_{N/4}, I_N)$ as follows:

$$H(k, (a, i)) := (((i \cdot k) \bmod p) + a) \bmod N.$$

(a) Show that H is a $4/N$-UHF.

Hint: Use Exercise 7.21 and part (a) of Exercise 7.22.

(b) Show how to use H to modify PMAC$_0$ so that the message space is $\mathcal{Y}^{\leq \ell}$ (where $\mathcal{Y} = \{0,1\}^n$ and $\ell < N/4$), and the PRF F_1 is defined over $(\mathcal{K}_1, \mathcal{Y}, \mathcal{Y})$. Analyze the security of your construction, giving a concrete security bound.

7.26 (Collision lower-bounds for H_{poly}***).*** Consider the function $H_{\text{poly}}(k, m)$ defined in (7.3) using a prime p and assume $\ell = 2$.

(a) Show that for all sufficiently large p, the following holds: for any fixed $k \in \mathbb{Z}_p$, among $\lfloor \sqrt{p} \rfloor$ random inputs to $H_{\text{poly}}(k, \cdot)$, the probability of a collision is bounded from below by a constant.

Hint: Use the birthday paradox (Appendix B.1).

(b) Show that given any collision for H_{poly} under key k, we can efficiently compute k. That is, give an efficient algorithm that takes two inputs $m, m' \in \mathbb{Z}_p^2$, and that outputs $\hat{k} \in \mathbb{Z}_p$, and satisfies the following property: for every $k \in \mathbb{Z}_p$, if $H(k, m) = H(k, m')$, then $\hat{k} = k$.

7.27 (H_{poly} key collisions). Consider the function $H_{\text{poly}}(k, m)$ defined in (7.3), where $k \in \mathbb{Z}_p$ and $m \in \mathbb{Z}_p^{\ell}$. Show that there is an efficient algorithm that takes $t, k_1, \ldots, k_\ell \in \mathbb{Z}_p$ as input, and outputs a message $m \in \mathbb{Z}_p^{\ell}$ such that $t = E(k_1, m) = \cdots = E(k_\ell, m)$. This message maps to the same output t under ℓ different keys. This property of H_{poly} has been used in some attacks [102].

7.28 (XOR-hash analysis). Generalize Theorem 7.6 to show that for every Q-query UHF adversary $\mathcal{A}$, there exists a PRF adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\text{MUHF}\mathsf{adv}[\mathcal{A}, F^{\oplus}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{Q^2}{2|\mathcal{Y}|}.$$

Moreover, $\mathcal{B}$ makes at most $Q\ell$ queries to F.

7.29 (H_{xpoly} is not a good PUF). Show that H_{xpoly} defined in (7.23) is not a good PUF by exhibiting an adversary that wins Attack Game 7.5 with probability 1.

7.30 (Converting a one-time MAC to a MAC). Suppose $\mathcal{I} = (S, V)$ is a (possibly randomized) MAC defined over $(\mathcal{K}_1, \mathcal{M}, \mathcal{T})$, where $\mathcal{T} = \{0,1\}^n$, that is one-time secure (see Section 7.6). Further suppose that F is a secure PRF defined over $(\mathcal{K}_2, \mathcal{R}, \mathcal{T})$, where $|\mathcal{R}|$ is super-poly. Consider the MAC $\mathcal{I}' = (S', V')$ defined over $(\mathcal{K}_1 \times \mathcal{K}_2, \mathcal{M}, \mathcal{R} \times \mathcal{T})$ as follows:

$$\begin{aligned} S'((k_1, k_2), m) &:= \big\{\ r \xleftarrow{R} \mathcal{R};\ \ t \xleftarrow{R} S(k_1, m);\ \ t' \leftarrow F(k_2, r) \oplus t;\quad \text{output } (r, t')\ \big\} \\ V'((k_1, k_2), m, (r, t')) &:= \big\{\ t \leftarrow F(k_2, r) \oplus t';\quad \text{output } V(k_1, m, t)\ \big\} \end{aligned}$$

Show that $\mathcal{I}'$ is a secure (many time) MAC.

7.31 (Pairwise independent functions). In this exercise, we develop the notion of a PRF that is unconditionally secure, provided the adversary can make at most *two* queries. We say that a PRF F defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ is an **ϵ-almost pairwise independent function**, or **ϵ-APIF**, if the following holds: for all adversaries $\mathcal{A}$ (even inefficient ones) that make at most 2 queries in Attack Game 4.2, we have $\text{PRF}\mathsf{adv}[\mathcal{A}, F] \leq \epsilon$. If $\epsilon = 0$, we call F a **pairwise independent function**, or **PIF**.

(a) Show that any ϵ-APIF is an $(\epsilon + 1/|\mathcal{Y}|)$-PUF.

Discussion: This shows that every PIF is also a PUF. The converse is false: there is an ϵ-PUF that is not an ϵ'-APIF even for $\epsilon' = 1/2$.

(b) Suppose that $|\mathcal{X}| > 1$ and that for all $x_0, x_1 \in \mathcal{X}$ with $x_0 \neq x_1$, and all $y_0, y_1 \in \mathcal{Y}$, we have

$$\Pr\big[F(k, x_0) = y_0 \ \text{ and } \ F(k, x_1) = y_1\big] = \frac{1}{|Y|^2},$$

where the probability is over the random choice of $k \in \mathcal{K}$. Show that F is a PIF.

(c) Consider the function H' built from H in (7.32). Show that if H is a $(1/N)$-DUF, then H' is a PIF.

(d) For positive integer d, let $I_d := \{0, \ldots, d-1\}$. Let n be a positive integer and set $N := 2^n$. Consider the keyed hash function H defined over $(I_{N^2}^{\ell+1}, I_N^{\ell}, I_N)$ as follows:

$$H((k_0, k_1, \ldots, k_\ell), (a_1, \ldots, a_\ell)) := \big\lfloor \big(\ \big(k_0 + \sum_i a_i k_i\big) \bmod N^2 \ \big) \ / \ N \big\rfloor.$$

Show that H is a PIF. Note: on a typical computer, if n is not too large, this can be implemented very easily with just integer multiplications, additions, and shifts.

(e) Show that in the PRF(UHF) composition, if H is an ϵ_1-UHF and F is an ϵ_2-APIF, then the composition F' is an $(\epsilon_1 + \epsilon_2)$-APIF.

7.32 (Unconditionally secure two time encryption). Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ where $\mathcal{X} = \{0,1\}^n$ and $|\mathcal{X}|$ is super-poly. Suppose that F is an ϵ-APIF for a negligible ϵ, as defined in the previous exercise. Consider the following cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ where

$$\begin{aligned} E(k, m) &:= \big\{ r \xleftarrow{\text{R}} \mathcal{X}, \ \text{output } c \leftarrow \big(r, \ F(k, r) \oplus m\big) \big\} \\ D\big(k, (c_0, c_1)\big) &:= c_1 \oplus F(k, c_0) \end{aligned}$$

Show that this cipher is unconditionally CPA secure provided the adversary can make at most two queries in Attack Game 5.2. Specifically, CPA$\mathsf{adv}[\mathcal{A}, \mathcal{E}] \le 2(\epsilon + (1/|\mathcal{X}|))$ for all 2-query adversaries $\mathcal{A}$, even inefficient ones.

Discussion: Recall that the one time pad cipher is unconditionally secure against a single query CPA adversary. The construction here is a generalization of the one time pad that is unconditionally secure against a two query CPA adversary. One can further generalize this: for every $d \ge 1$, there is a cipher $\mathcal{E}_d = (E_d, D_d)$ that is unconditionally secure against a d-query CPA adversary. The size of the key used by $\mathcal{E}_d$ grows linearly with d.

Chapter 8

Message integrity from collision resistant hashing

In the previous chapter we discussed universal hash functions (UHFs) and showed how they can be used to construct MACs. Recall that UHFs are *keyed* hash functions for which finding collisions is difficult, as long as the key is kept secret.

In this chapter we study *keyless* hash functions for which finding collisions is difficult. Informally, a keyless function is an efficiently computable function whose description is fully public. There are no secret keys and anyone can evaluate the function. Let H be a keyless hash function from some large message space $\mathcal{M}$ into a small digest space $\mathcal{T}$. As in the previous chapter, we say that two messages $m_0, m_1 \in \mathcal{M}$ are a **collision** for the function H if

$$H(m_0) = H(m_1) \quad \text{and} \quad m_0 \neq m_1.$$

Informally, we say that the function H is **collision resistant** if finding a collision for H is difficult. Since the digest space $\mathcal{T}$ is much smaller than $\mathcal{M}$, we know that many such collisions exist. Nevertheless, if H is collision resistant, actually finding a pair m_0, m_1 that collide should be difficult. We give a precise definition in the next section.

In this chapter we will construct collision resistant functions and present several applications. To give an example of a collision resistant function we mention a US federal standard called the Secure Hash Algorithm Standard or SHA for short. The SHA standard describes a number of hash functions that offer varying degrees of collision resistance. For example, **SHA256** is a function that hashes long messages into 256-bit digests. It is believed that finding collisions for SHA256 is difficult.

Collision resistant hash functions have many applications. We briefly mention two such applications here and give the details later on in the chapter. Many other applications are described throughout the book.

Extending the domain of cryptographic primitives. An important application for collision resistance is its ability to extend primitives built for short inputs to primitives for much longer inputs. We give a MAC construction as an example. Suppose we are given a MAC system $\mathcal{I} = (S, V)$ that only authenticates short messages, say messages that are 256 bits long. We want to extend the domain of the MAC so that it can authenticate much longer inputs. Collision resistant hashing gives a very simple solution. To compute a MAC for some long message m we first hash m and

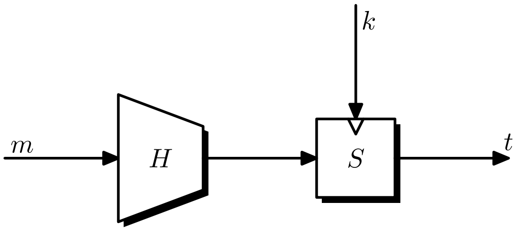


Figure 8.1: Hash-then-MAC construction

then apply S to the resulting short digest, as described in Fig. 8.1. In other words, we define a new MAC system $\mathcal{I} = (S', V')$ where $S'(k,\ m) := S(k,\ H(m))$. MAC verification works analogously by first hashing the message and then verifying the tag of the digest.

Clearly this hash-then-MAC construction would be insecure if it were easy to find collisions for H. If an adversary could find two long messages m_0 and m_1 such that $H(m_0) = H(m_1)$ then he could forge tags using a chosen message attack. Suppose m_0 is an innocuous message while m_1 is evil, say a virus infected program. The adversary would ask for the tag on the message m_0 and obtain a tag t in response. Then the pair (m_0, t) is a valid message-tag pair, but so is the pair (m_1, t). Hence, the adversary is able to forge a tag for m_1, which breaks the MAC. Even worse, the valid tag may fool a user into running the virus. This argument shows that collision resistance is necessary for this hash-then-MAC construction to be secure. Later on in the chapter we prove that collision resistance is, in fact, sufficient to prove security.

The hash-then-MAC construction looks similar to the PRF(UHF) composition discussed in the previous chapter (Section 7.3). These two methods build similar looking MACs from very different building blocks. The main difference is that a collision resistant hash can extend the input domain of any MAC. On the other hand, a UHF can only extend the domain of a very specific type of MAC, namely a PRF. This is illustrated further in Exercise 7.4. Another difference is that the secret key in the hash-then-MAC method is exactly the same as in the underlying MAC. The PRF(UHF) method, in contrast, extends the secret key of the underlying PRF by adding a UHF secret key.

The hash-then-MAC construction performs better than PRF(UHF) when we wish to compute the tag for a single message m under multiple keys $k_1, \ldots, k_n$. That is, we wish to compute $S'(k_i, m)$ for all $i = 1, \ldots, n$. This comes up, for example, when providing integrity for files on a disk that is readable by multiple users. Each file header contains one integrity tag per user, so that each user can verify integrity using its own MAC key. With the hash-then-MAC construction it suffices to compute $H(m)$ once and then quickly derive the n tags from this single hash. With a PRF(UHF) MAC, the UHF depends on the key k_i and consequently we will need to rehash the entire file n times, once for each user. See also Exercise 6.4 for more on this problem.

File integrity. Collision resistance is frequently used for file integrity. Consider a set of n critical files that change infrequently, such as a set of executables on disk. We want a method to verify that these files have not been modified by some malicious code or malware. To do so we need a small amount of read-only memory, namely memory that the malware can read, but cannot modify. Read-only memory can be implemented, for example, using a small USB disk that has a physical switch flipped to the "read-only" position. We place a hash of each of the n critical files in the

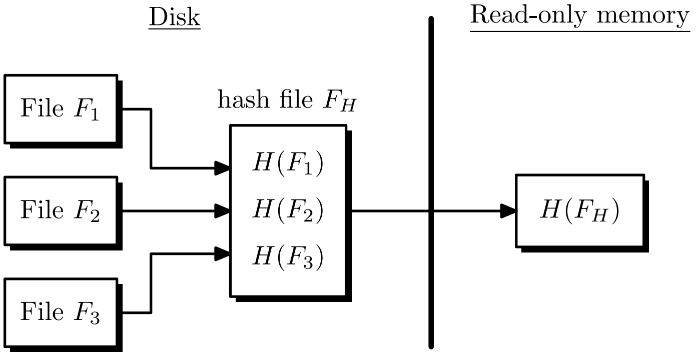


Figure 8.2: File integrity using small read-only memory

read-only memory so that this storage area only contains n short hashes. Now, we can check integrity of a file F by hashing F and comparing the resulting hash to the one stored in read-only memory. If a mismatch is found, the system declares that file F is corrupt. Operating systems use this mechanism to ensure integrity of operating system files during the boot process. Package managers also use this mechanism: a user who downloads a package from a remote server can verify that the package is authentic by hashing it, and verifying that the hash value is equal to the value stored on a trusted read-only site.

What property should the hash function H satisfy for this integrity mechanism to be secure? Let F be a file protected by this system. Since the malware cannot alter the contents of the read-only storage, its only avenue for modifying F without being detected is to find another file F' such that $H(F) = H(F')$. Replacing F by F' would not be caught by this hashing system. However, finding such an F' will be difficult if H is collision resistant. Collision resistance, thus, implies that the malware cannot change F without being detected by the hash.

This system stores all file hashes in read-only memory. The amount of read-only memory needed could become quite large when there are many files to protect. We can reduce the size of read-only memory by treating the entire list of hashes as just another file stored on disk and denoted F_H. We store the hash of F_H in read-only memory, as described in Fig. 8.2, so that now read-only memory contains just a single hash value. To verify the integrity of some file F, we first verify integrity of the file F_H by hashing the contents of F_H and comparing the result to the value in read-only memory. Then we verify integrity of F by hashing F and comparing the result with the corresponding hash stored in F_H. In Section 8.9 we will see a more efficient solution using a data structure called a *Merkle tree*.

In the introduction to Chapter 6 we proposed a MAC-based file integrity system. The system stored a tag of every file along with the file. We also needed a small amount of *secret storage* to store the user's secret MAC key. This key was used every time file integrity was verified. In comparison, when using collision resistant hashing there are no secrets and there is no need for secret storage. Instead, we need a small amount of read-only storage for storing file hashes. Generally speaking, read-only storage is much easier to build than secret storage. Hence, collision resistance seems more appropriate for this particular application. In Chapter 13 we will develop an even better solution to this problem, using digital signatures, that does not need read-only storage or online secret storage.

Security without collision resistance. By extending the input to the hash function with a few random bits we can prove security for both applications above using a weaker notion of collision resistance called **target collision resistance** or TCR for short. We show in Section 8.11.2 how to use TCR for both file integrity and for extending cryptographic primitives. The downside is that the resulting tags are longer than the ones obtained from collision resistant hashing. Hence, although in principle it is often possible to avoid relying on collision resistance, the resulting systems are not as efficient.

8.1 Definition of collision resistant hashing

A **(keyless) hash function** $H : \mathcal{M} \to \mathcal{T}$ is an efficiently computable function from some (large) message space $\mathcal{M}$ into a (small) digest space $\mathcal{T}$. We say that H is defined over $(\mathcal{M}, \mathcal{T})$. We define collision resistance of H using the following (degenerate) game:

Attack Game 8.1 (Collision Resistance). For a given hash function H defined over $(\mathcal{M}, \mathcal{T})$ and adversary $\mathcal{A}$, the adversary takes no input and outputs two messages m_0 and m_1 in $\mathcal{M}$.

We say that $\mathcal{A}$ wins the game if the pair m_0, m_1 is a collision for H, namely $m_0 \neq m_1$ and $H(m_0) = H(m_1)$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. Adversary $\mathcal{A}$ is called a **collision finder**. □

Definition 8.1. *We say that a hash function H over $(\mathcal{M}, \mathcal{T})$ is **collision resistant** if for all efficient adversaries $\mathcal{A}$, the quantity $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H]$ is negligible.*

At first glance, it may seem that collision resistant functions cannot exist. The problem is this: since $|\mathcal{M}| > |\mathcal{T}|$ there must exist inputs m_0 and m_1 in $\mathcal{M}$ that collide, namely $H(m_0) = H(m_1)$. An adversary $\mathcal{A}$ that simply prints m_0 and m_1 and exits is an efficient adversary that breaks the collision resistance of H. We may not be able to write the explicit program code for $\mathcal{A}$ (since we do not know m_0, m_1), but this $\mathcal{A}$ certainly exists. Consequently, for any hash function H defined over $(\mathcal{M}, \mathcal{T})$ there *exists* some efficient adversary $\mathcal{A}_H$ that breaks the collision resistance of H. Hence, it appears that no function H can satisfy Definition 8.1.

The way out of this is that, formally speaking, our hash functions are parameterized by a system parameter: each choice of a system parameter describes a different function H, and so we cannot simply "hardwire" a fixed collision into an adversary: an effective adversary must be able to efficiently compute a collision *as a function of the system parameter*. This is discussed in more depth in the Mathematical details section below.[1]

8.1.1 Mathematical details

As usual, we give a more mathematically precise definition of a collision resistant hash function using the terminology defined in Section 2.3.

Definition 8.2 (Keyless hash functions). *A **(keyless) hash function** is an efficient algorithm H, along with two families of spaces with system parameterization P:*

$$\mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \textit{and} \quad \mathbf{T} = \{\mathcal{T}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

[1] Some authors deal with this issue by have H take as input a randomly chosen key k, and giving k to the adversary at the beginning of this attack game. By viewing k as a system parameter, this approach is really the same as ours.

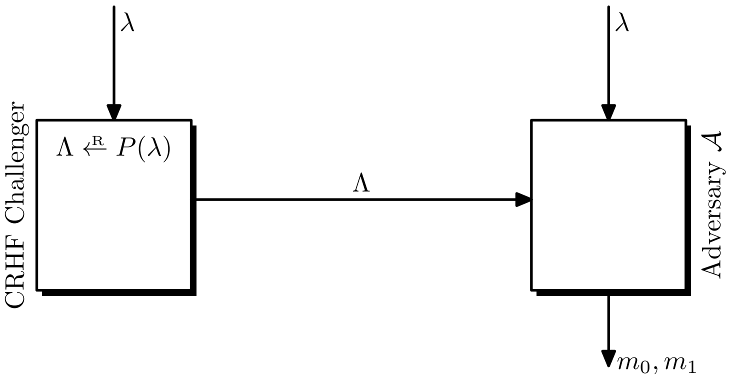


Figure 8.3: Asymptotic version of Attack Game 8.1

such that

1. $\mathbf{M}$*, and* $\mathbf{T}$ *are efficiently recognizable.*
2. *Algorithm* H *is an efficient deterministic algorithm that on input* $\lambda \in \mathbb{Z}_{\geq 1}$*,* $\Lambda \in \text{Supp}(P(\lambda))$*, and* $m \in \mathcal{M}_{\lambda,\Lambda}$*, outputs an element of* $\mathcal{T}_{\lambda,\Lambda}$*.*

In defining collision resistance, we parameterize Attack Game 8.1 by the security parameter λ. The asymptotic game is shown in Fig. 8.3. The advantage $\text{CR}\mathsf{adv}[\mathcal{A}, H]$ is then a function of λ. Definition 8.1 should be read as saying that $\text{CR}\mathsf{adv}[\mathcal{A}, H](\lambda)$ is a negligible function.

It should be noted that the security and system parameters are artifacts of the formal framework that are needed to make sense of Definition 8.1. In the real world, however, these parameters are picked when the hash function is designed, and are ignored from that point onward. SHA256, for example, does not take either a security parameter or a system parameter as input.

8.2 Building a MAC for large messages

To exercise the definition of collision resistance, we begin with an easy application described in the introduction — extending the message space of a MAC. Suppose we are given a secure MAC $\mathcal{I} = (S, V)$ for short messages. Our goal is to build a new secure MAC $\mathcal{I}'$ for much longer messages. We do so using a collision resistant hash function: $\mathcal{I}'$ computes a tag for a long message m by first hashing m to a short digest and then applying $\mathcal{I}$ to the digest, as shown in Fig. 8.1.

More precisely, let H be a hash function that hashes long messages in $\mathcal{M}$ to short digests in $\mathcal{T}_H$. Suppose $\mathcal{I}$ is defined over $(\mathcal{K}, \mathcal{T}_H, \mathcal{T})$. Define $\mathcal{I}' = (S', V')$ for long messages as follows:

$$S'(k, m) := S\big(k,\ H(m)\big) \quad \text{and} \quad V'(k, m, t) := V\big(k,\ H(m),\ t\big) \tag{8.1}$$

Then $\mathcal{I}'$ authenticates long messages in $\mathcal{M}$. The following easy theorem shows that $\mathcal{I}'$ is secure, assuming H is collision resistant.

Theorem 8.1. *Suppose the MAC system* $\mathcal{I}$ *is a secure MAC and the hash function* H *is collision resistant. Then the derived MAC system* $\mathcal{I}' = (S', V')$ *defined in (8.1) is a secure MAC.*

In particular, suppose $\mathcal{A}$ is a MAC adversary attacking $\mathcal{I}'$ (as in Attack Game 6.1). Then there exist a MAC adversary $\mathcal{B}_{\mathcal{I}}$ and an efficient collision finder $\mathcal{B}_H$, which are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}'] \le \mathrm{MAC}\mathsf{adv}[\mathcal{B}_{\mathcal{I}}, \mathcal{I}] + \mathrm{CR}\mathsf{adv}[\mathcal{B}_H, H].$$

It is clear that collision resistance of H is essential for the security of $\mathcal{I}'$. Indeed, if an adversary can find a collision m_0, m_1 on H, then he can win the MAC attack game as follows: submit m_0 to the MAC challenger for signing, obtaining a tag $t_0 := S(k, H(m_0))$, and then output the message-tag pair (m_1, t_0). Since $H(m_0) = H(m_1)$, the tag t_0 must be a valid tag on the message m_1.

Proof idea. Our goal is to show that no efficient adversary can win the MAC Attack Game 6.1 for our new MAC system $\mathcal{I}'$. An adversary $\mathcal{A}$ in this game asks the challenger to MAC a few long messages $m_1, m_2, \ldots \in \mathcal{M}$ and then tries to invent a new valid message-MAC pair (m, t). If $\mathcal{A}$ is able to produce a valid forgery (m, t) then one of two things must happen:

1. either m collides with some query m_i from $\mathcal{A}$, so that $H(m) = H(m_i)$ and $m \neq m_i$;

2. or m does not collide under H with any of $\mathcal{A}$'s queries $m_1, m_2, \ldots \in \mathcal{M}$.

It should be intuitively clear that if $\mathcal{A}$ produces forgeries of the first type then $\mathcal{A}$ can be used to break the collision resistance of H since m and m_i are a valid collision for H. On the other hand, if $\mathcal{A}$ produces forgeries of the second type then $\mathcal{A}$ can be used to break the MAC system $\mathcal{I}$: the pair $(H(m),\ t)$ is a valid MAC forgery for $\mathcal{I}$. Thus, if $\mathcal{A}$ wins the MAC attack game for $\mathcal{I}'$ we break one of our assumptions. □

Proof. We make this intuition rigorous. Let $m_1, m_2, \ldots \in \mathcal{M}$ be $\mathcal{A}$'s queries during the MAC attack game and let $(m, t) \in \mathcal{M} \times \mathcal{T}$ be the adversary's output, which we assume is not among the signed pairs. We define three events:

- Let X be the event that adversary $\mathcal{A}$ wins the MAC Attack Game 6.1 with respect to $\mathcal{I}'$.

- Let Y denote the event that some m_i collides with m under H, that is, for some i we have $H(m) = H(m_i)$ and $m \neq m_i$.

- Let Z denote the event that $\mathcal{A}$ wins Attack Game 6.1 on $\mathcal{I}'$ and event Y did not occur.

Using events Y and Z we can rewrite $\mathcal{A}$'s advantage in winning Attack Game 6.1 as follows:

$$\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}'] \;=\; \Pr[X] \;\le\; \Pr[X \wedge \neg Y] + \Pr[Y] \;=\; \Pr[Z] + \Pr[Y] \tag{8.2}$$

To prove the theorem we construct a collision finder $\mathcal{B}_H$ and a MAC adversary $\mathcal{B}_{\mathcal{I}}$ such that

$$\Pr[Y] = \mathrm{CR}\mathsf{adv}[\mathcal{B}_H, H] \quad \text{and} \quad \Pr[Z] = \mathrm{MAC}\mathsf{adv}[\mathcal{B}_{\mathcal{I}}, \mathcal{I}].$$

Both adversaries are straight-forward.

Adversary $\mathcal{B}_H$ plays the role of challenger to $\mathcal{A}$ in the MAC attack game, as follows:

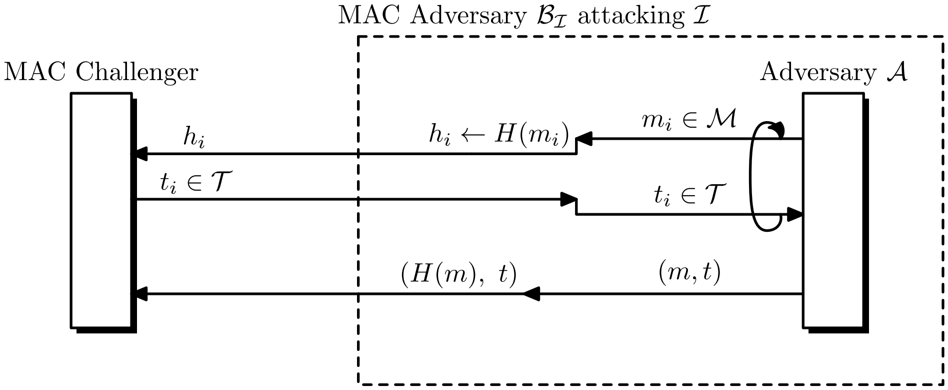


Figure 8.4: Adversary $\mathcal{B}_\mathcal{I}$ in the proof of Theorem 8.1

```
Initialization:
    $k \xleftarrow{R} \mathcal{K}$
Upon receiving a signing query $m_i \in \mathcal{M}$ from $\mathcal{A}$ do:
    $t_i \xleftarrow{R} S(k,\ H(m_i)\ )$
    Send $t_i$ to $\mathcal{A}$
Upon receiving the final message-tag pair $(m, t)$ from $\mathcal{A}$ do:
    if $H(m) = H(m_i)$ and $m \neq m_i$ for some $i$
        then output the pair $(m, m_i)$
```

Algorithm $\mathcal{B}_H$ responds to $\mathcal{A}$'s signature queries exactly as in a real MAC attack game. Therefore, event Y happens during the interaction with $\mathcal{B}_H$ with the same probability that it happens in a real MAC attack game. Clearly when event Y happens, $\mathcal{A}_H$ succeeds in finding a collision for H. Hence, $\text{CR}\mathsf{adv}[\mathcal{B}_H, H] = \Pr[Y]$ as required.

MAC adversary $\mathcal{B}_\mathcal{I}$ is just as simple and is shown in Fig. 8.4. When $\mathcal{A}$ outputs the final message-tag pair (m, t), adversary $\mathcal{B}_\mathcal{I}$ outputs $(H(m), t)$. When event Z happens we know that $V'(k, m, t)$ outputs accept and the pair (m, t) is not equal to any of $(m_1, t_1),\ (m_2, t_2), \ldots \in \mathcal{M} \times \mathcal{T}$. Furthermore, since event Y does not happen, we know that $(H(m), t)$ is not equal to any of $(H(m_1), t_1),\ (H(m_2), t_2), \ldots \in \mathcal{T}_H \times \mathcal{T}$. It follows that $(H(m),\ t)$ is a valid existential forgery for $\mathcal{I}$. Hence, $\mathcal{B}_\mathcal{I}$ succeeds in creating an existential forgery with the same probability that event Z happens. In other words, $\text{MAC}\mathsf{adv}[\mathcal{B}_\mathcal{I}, \mathcal{I}] = \Pr[Z]$, as required. The proof now follows from (8.2). □

8.3 Birthday attacks on collision resistant hash functions

Cryptographic hash functions are most useful when the output digest size is small. The challenge is to design hash functions whose output is as short as possible and yet finding collisions is difficult. It should be intuitively clear that the shorter the digest, the easier it is for an attacker to find collisions. To illustrate this, consider a hash function H that outputs ℓ-bit digests for some small ℓ. Clearly, by hashing $2^\ell + 1$ distinct messages the attacker will find two messages that hash to the same digest and will thus break collision resistance of H. This brute-force attack will break the collision resistance of any hash function. Hence, for instance, hash functions that output 16-bit digests cannot be collision resistant — a collision can always be found using only $2^{16} + 1 = 65537$ evaluations of the hash.

Birthday attacks. A far more devastating attack can be built using the birthday paradox discussed in Section B.1 in the appendix. Let H be a hash function defined over $(\mathcal{M}, \mathcal{T})$ and set $N := |\mathcal{T}|$. For standard hash functions N is quite large, for example $N = 2^{256}$ for SHA256. Throughout this section we will assume that the size of $\mathcal{M}$ is at least $100N$. This basically means that messages being hashed are slightly longer than the output digest. We describe a general collision finder that finds collisions for H after an expected $O(\sqrt{N})$ evaluations of H. For comparison, the brute-force attack above took $O(N)$ evaluations. This more efficient collision finder forces us to use much larger digests.

The birthday collision finder for H works as follows: it chooses $s \approx \sqrt{N}$ random and independent messages, $m_1, \ldots, m_s \overset{\mathrm{R}}{\leftarrow} \mathcal{M}$, and looks for a collision among these s messages. We will show that the birthday paradox implies that a collision is likely to exist among these messages. More precisely, the birthday collision finder works as follows:
Algorithm `BirthdayAttack`:

```
1.  Set $s \leftarrow \lceil 2\sqrt{N}\ \rceil + 1$
2.  Generate $s$ uniform random messages $m_1, \ldots, m_s$ in $\mathcal{M}$
3.  Compute $x_i \leftarrow H(m_i)$ for all $i = 1, \ldots, s$
4.  Look for distinct $i, j \in \{1, \ldots, s\}$ such that $H(m_i) = H(m_j)$
5.  If such $i, j$ exist and $m_i \neq m_j$ then
6.      output the pair $(m_i, m_j)$
```

We argue that when the adversary picks $s := \left\lceil 2\sqrt{N} \right\rceil + 1$ random messages in $\mathcal{M}$, then with probability at least $1/2$, there will exist distinct i, j such that $H(m_i) = H(m_j)$ and $m_i \neq m_j$. This means that the algorithm will output a collision with probability at least $1/2$.

Lemma 8.2. *Let $m_1, \ldots, m_s$ be the random messages sampled in Step 2. Assume $|\mathcal{M}| \geq 100N$. Then with probability at least $1/2$ there exists i, j in $\{1, \ldots, s\}$ such that $H(m_i) = H(m_j)$ and $m_i \neq m_j$.*

Proof. For $i = 1, \ldots, s$ let $x_i := H(m_i)$. First, we argue that two of the x_i values will collide with probability at least $3/4$. If the x_i were uniformly distributed in $\mathcal{T}$ then this would follow immediately from part (i) of Theorem B.1. Indeed, if the x_i were independent and uniform in $\mathcal{T}$ a collision among the x_i will occur with probability at least $1 - e^{-s(s-1)/2N} \geq 1 - e^{-2} \geq 3/4$.

However, in reality, the function $H(\cdot)$ might bias the output distribution. Even though the m_i are sampled uniformly from $\mathcal{M}$, the resulting x_i may not be uniform in $\mathcal{T}$. As a simple example, consider a hash function $H(\cdot)$ that only outputs digests in a certain small subset of $\mathcal{T}$. The resulting x_i would certainly not be uniform in $\mathcal{T}$. Fortunately (for the attacker) Corollary B.2 shows that non-uniform x_i only increase the probability of collision. Since the x_i are independent and identically distributed the corollary implies that a collision among the x_i will occur with probability at least $1 - e^{-s(s-1)/2N} \geq 3/4$ as required.

Next, we argue that a collision among the x_i is very likely to lead to a collision on $H(\cdot)$. Suppose $x_i = x_j$ for some distinct i, j in $\{1, \ldots, s\}$. Since $x_i = H(m_i)$ and $x_j = H(m_j)$, the pair m_i, m_j is a candidate for a collision on $H(\cdot)$. We just need to argue that $m_i \neq m_j$. We do so by arguing that all the $m_1, \ldots, m_s$ are distinct with probability at least 4/5. This follows directly from part (ii) of Theorem B.1. Recall that $\mathcal{M}$ is greater than $100N$. Since $m_1, m_2, \ldots$ are uniform and independent in $\mathcal{M}$, and $s < |\mathcal{M}|/2$, part (ii) of Theorem B.1 implies that the probability of collision among these m_i is at most $1 - e^{-s(s-1)/100N} \leq 1/5$. Therefore, the probability that no collision occurs is at least 4/5.

In summary, for the algorithm to discover a collision for $H(\cdot)$ it is sufficient that both a collision occurs on the x_i values and no collision occurs on the m_i values. This happens with probability at least $3/4 - 1/5 > 1/2$, as required. □

Variations. Algorithm `BirthdayAttack` requires $O(\sqrt{N})$ memory space, which can be quite large: larger than the size of commercially available disk farms. However, a modified birthday collision finder, described in Exercise 8.8, will find a collision with an expected $4\sqrt{N}$ evaluations of the hash function and *constant* memory space.

The birthday attack is likely to fail if one makes fewer than $\sqrt{N}$ queries to $H(\cdot)$. Suppose we only make $s = \epsilon\sqrt{N}$ queries to $H(\cdot)$, for some small $\epsilon \in [0,1]$. For simplicity we assume that $H(\cdot)$ outputs digests distributed uniformly in $\mathcal{T}$. Then part (ii) of Theorem B.1 shows that the probability of finding a collision degrades exponentially to approximately $1 - e^{-(\epsilon^2)} \approx \epsilon^2$.

Put differently, if after evaluating the hash function s times an adversary should obtain a collision with probability at most δ, then we need the digest space $\mathcal{T}$ to satisfy $|\mathcal{T}| \geq s^2/\delta$. For example, if after 2^{80} evaluations of H a collision should be found with probability at most 2^{-80} then the digest size must be at least 240 bits. Cryptographic hash functions such as SHA256 output a 256-bit digest. Other hash functions, such as SHA384 and SHA512, output even longer digests, namely, 384 and 512 bits respectively.

8.4 The Merkle-Damgård paradigm

We now turn to constructing collision resistant hash functions. Many practical constructions follow the Merkle-Damgård paradigm: start from a collision resistant hash function that hashes short messages and build from it a collision resistant hash function that hashes much longer messages. This paradigm reduces the problem of constructing collision resistant hashing to the problem of constructing collision resistance for short messages, which we address in the next section.

Let $h : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$ be a hash function. We shall assume that $\mathcal{Y}$ is of the form $\{0,1\}^\ell$ for some ℓ. While it is not necessary, typically $\mathcal{X}$ is of the form $\{0,1\}^n$ for some n. The **Merkle-Damgård function derived from** h, denoted H_{MD} and shown in Fig. 8.5, is a hash function defined over $(\{0,1\}^{\leq L},\ \mathcal{X})$ that works as follows (the pad PB is defined below):

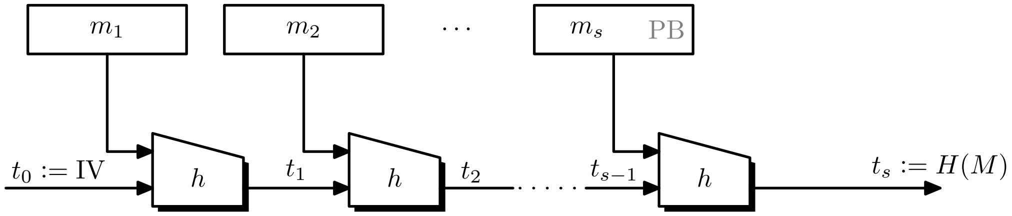


Figure 8.5: The Merkle-Damgård iterated hash function

```
input: $M \in \{0,1\}^{\leq L}$
output: a tag in $\mathcal{X}$
$\hat{M} \leftarrow M \parallel \text{PB}$    // pad with PB to ensure that the length of M is a multiple of ℓ bits
partition $\hat{M}$ into consecutive ℓ-bit blocks so that
            $\hat{M} = m_1 \parallel m_2 \parallel \cdots \parallel m_s$    where    $m_1, \ldots, m_s \in \{0,1\}^\ell$
$t_0 \leftarrow \text{IV} \in \mathcal{X}$
for $i = 1$ to $s$ do:
    $t_i \leftarrow h(t_{i-1}, m_i)$
output $t_s$
```

The function SHA256 is a Merkle-Damgård function where $\ell = 512$ and $n = 256$.

Before proving collision resistance of H_{MD}, let us first introduce some terminology for the various elements in Fig. 8.5:

- The hash function h is called the **compression function** of H.
- The constant IV is called the **initial value** and is fixed to some pre-specified value. One could take IV $= 0^n$, but usually the IV is set to some complicated string. For example, SHA256 uses a 256-bit IV whose value in hex is

$$\text{IV} := \text{6A09E667 BB67AE85 3C6EF372 A54FF53A 510E527F 9B05688C 1F83D9AB 5BE0CD19.}$$

- The variables $m_1, \ldots, m_s$ are called message blocks.
- The variables $t_0, t_1, \ldots, t_s \in \mathcal{X}$ are called **chaining variables**.
- The string PB is called the **padding block**. It is appended to the message to ensure that the message length is a multiple of ℓ bits.

The padding block PB must contain an encoding of the input message length. We will use this in the proof of security below. A standard format for PB is as follows:

$$\text{PB} := \boxed{100\ldots00 \parallel \langle s \rangle}$$

where $\langle s \rangle$ is a fixed-length bit string that encodes, in binary, the number of ℓ-bit blocks in M. Typically this field is 64-bits which means that messages to be hashed are less than 2^{64} blocks long. The '100 . . . 00' string is a variable length pad used to ensure that the total message length, including PB, is a multiple of ℓ. The variable length string '100 . . . 00' starts with a '1' to identify the position where the pad ends and the message begins. If the message length is such that there is no space for PB in the last block (for example, if the message length happens to be a multiple of ℓ), then an additional block is added just for the padding block.

Security of Merkle-Damgård. Next we prove that the Merkle-Damgård function is collision resistant, assuming the compression function is.

Theorem 8.3 (Merkle-Damgård). *Let L be a poly-bounded length parameter and let h be a collision resistant hash function defined over $(\mathcal{X} \times \mathcal{Y},\ \mathcal{X})$. Then the Merkle-Damgård hash function H_{MD} derived from h, defined over $(\{0,1\}^{\le L}, \mathcal{X})$, is collision resistant.*

> *In particular, for every collision finder $\mathcal{A}$ attacking H_{MD} (as in Attack Game 8.1) there exists a collision finder $\mathcal{B}$ attacking h, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathsf{CRadv}[\mathcal{A}, H_{\text{MD}}] = \mathsf{CRadv}[\mathcal{B}, h].$$

Proof. The collision finder $\mathcal{B}$ for finding h-collisions works as follows: it first runs $\mathcal{A}$ to obtain two distinct messages M and M' in $\{0,1\}^{\le L}$ such that $H_{\text{MD}}(M) = H_{\text{MD}}(M')$. We show that $\mathcal{B}$ can use M and M' to find an h-collision. To do so, $\mathcal{B}$ scans M and M' starting from the last block and works its way backwards. To simplify the notation, we assume that M and M' already contain the appropriate padding block PB in their last block.

Let $M = m_1 m_2 \ldots m_u$ be the u blocks of M and let $M' = m'_1 m'_2 \ldots m'_v$ be the v blocks of M'. We let $t_0, t_1, \ldots, t_u \in \mathcal{X}$ be the chaining values for M and $t'_0, t'_1, \ldots, t'_v \in \mathcal{X}$ be the chaining values for M'. The very last application of h gives the final output digest and since $H_{\text{MD}}(M) = H_{\text{MD}}(M')$ we know that

$$h(t_{u-1}, m_u) = h(t'_{v-1}, m'_v).$$

If either $t_{u-1} \ne t'_{v-1}$ or $m_u \ne m'_v$ then the pair of inputs (t_{u-1}, m_u) and (t'_{v-1}, m'_v) is an h-collision. $\mathcal{B}$ outputs this collision and terminates.

Otherwise, $t_{u-1} = t'_{v-1}$ and $m_u = m'_v$. Recall that the padding blocks are contained in m_u and m'_v and these padding blocks contain an encoding of u and v. Therefore, since $m_u = m'_v$ we deduce that $u = v$ so that M and M' must contain the same number of blocks.

At this point we know that $u = v$, $m_u = m'_u$, and $t_{u-1} = t'_{u-1}$. We now consider the second-to-last block. Since $t_{u-1} = t'_{u-1}$ we know that

$$h(t_{u-2}, m_{u-1}) = h(t'_{u-2}, m'_{u-1}).$$

As before, if either $t_{u-2} \ne t'_{u-2}$ or $m_{u-1} \ne m'_{u-1}$ then $\mathcal{B}$ just found an h-collision. It outputs this collision and terminates.

Otherwise, we know that $t_{u-2} = t'_{u-2}$ and $m_{u-1} = m'_{u-1}$ and $m_u = m'_u$. We now consider the third block from the end. As before, we either find an h-collision or deduce that $m_{u-2} = m'_{u-2}$ and $t_{u-3} = t'_{u-3}$. We keep iterating this process moving from right to left one block at a time. At the ith block one of two things happens. Either the pair of messages (t_{i-1}, m_i) and (t'_{i-1}, m'_i) is an h-collision, in which case $\mathcal{B}$ outputs this collision and terminates. Or we deduce that $t_{i-1} = t'_{i-1}$ and $m_j = m'_j$ for all $j = i, i+1, \ldots, u$.

Suppose this process continues all the way to the first block and we still did not find an h-collision. Then at this point we know that $m_i = m'_i$ for $i = 1, \ldots, u$. But this implies that $M = M'$ contradicting the fact that M and M' were a collision for H_{MD}. Hence, since $M \neq M'$, the process of scanning blocks of M and M' from right to left must produce an h-collision. We conclude that $\mathcal{B}$ breaks the collision resistance of h as required.

In summary, we showed that whenever $\mathcal{A}$ outputs an H_{MD}-collision, $\mathcal{B}$ outputs an h-collision. Hence, $\text{CR}\mathsf{adv}[\mathcal{A}, H_{\text{MD}}] = \text{CR}\mathsf{adv}[\mathcal{B}, h]$ as required. □

Variations. Note that the Merkle-Damgård construction is inherently sequential — the ith block cannot be hashed before hashing all previous blocks. This makes it difficult to take advantage of hardware parallelism when available. In Exercise 8.9 we investigate a different hash construction that is better suited for a multi-processor machine.

The Merkle-Damgård theorem (Theorem 8.3) shows that collision resistance of the compression function is sufficient to ensure collision resistance of the iterated function. This condition, however, is not necessary. Black, Rogaway, and Shrimpton [24] give several examples of compression functions that are clearly not collision resistant, and yet the resulting iterated Merkle-Damgård functions are collision resistant.

8.4.1 Joux's attack

We briefly describe a cute attack that applies specifically to Merkle-Damgård hash functions. Let H_1 and H_2 be Merkle-Damgård hash functions that output tags in $\mathcal{X} := \{0,1\}^n$. Define $H_{12}(M) := H_1(M) \parallel H_2(M) \in \{0,1\}^{2n}$. One would expect that finding a collision for H_{12} should take time at least $\Omega(2^n)$. Indeed, this would be the case if H_1 and H_2 were independent random functions.

We show that when H_1 and H_2 are Merkle-Damgård functions we can find collisions for H in time approximately $n2^{n/2}$ which is far less than 2^n. This attack illustrates that our intuition about random functions may lead to incorrect conclusions when applied to a Merkle-Damgård function.

We say that an s-collision for a hash function H is a set of messages $M_1, \ldots, M_s \in \mathcal{M}$ such that $H(M_1) = \ldots = H(M_s)$. Joux showed how to find an s-collision for a Merkle-Damgård function in time $O((\log_2 s)|\mathcal{X}|^{1/2})$. Using Joux's method we can find a $2^{n/2}$-collision $M_1, \ldots, M_{2^{n/2}}$ for H_1 in time $O(n2^{n/2})$. Then, by the birthday paradox it is likely that two of these messages, say M_i, M_j, are also a collision for H_2. This pair M_i, M_j is a collision for both H_1 and H_2 and therefore a collision for H_{12}. It was found in time $O(n2^{n/2})$, as promised.

Finding s-collisions. To find an s-collision, let H be a Merkle-Damgård function over $(\mathcal{M}, \mathcal{X})$ built from a compression function h. We find an s-collision $M_1, \ldots, M_s \in \mathcal{M}$ where each message M_i contains $\log_2 s$ blocks. For simplicity, assume that s is a power of 2 so that $\log_2 s$ is an integer. As usual, we let t_0 denote the Initial Value (IV) used in the Merkle-Damgård construction.

The plan is to use the birthday attack $\log_2 s$ times on the compression function h. We first spend time $2^{n/2}$ to find two distinct blocks m_0, m'_0 such that (t_0, m_0) and (t_0, m'_0) collide under h. Let $t_1 := h(t_0, m_0)$. Next we spend another $2^{n/2}$ time to find two distinct blocks m_1, m'_1 such that (t_1, m_1) and (t_1, m'_1) collide under h. Again, we let $t_2 := h(t_1, m_1)$ and repeat. We iterate this process $b := \log_2 s$ times until we have b pairs of blocks:

$$(m_i, m'_i) \quad \text{for } i = 0, 1, \ldots b-1 \quad \text{that satisfy} \quad h(t_i, m_i) = h(t_i, m'_i).$$

Now, consider the message $M = m_0 m_1 \dots m_{b-1}$. The main point is that replacing any block m_i in this message by m'_i will not change the chaining value t_{i+1} and therefore the value of $H(M)$ will not change. Consequently, we can replace any subset of $m_0, \dots, m_{b-1}$ by the corresponding blocks in $m'_0, \dots, m'_{b-1}$ without changing $H(M)$. As a result we obtain $s = 2^b$ messages

$$\begin{array}{c} m_0 m_1 \dots m_{b-1} \\ m'_0 m_1 \dots m_{b-1} \\ m_0 m'_1 \dots m_{b-1} \\ m'_0 m'_1 \dots m_{b-1} \\ \vdots \\ m'_0 m'_1 \dots m'_{b-1} \end{array}$$

that all hash to the same value under H. In summary, we found a 2^b-collision in time $O(b2^{n/2})$. As explained above, this lets us find collisions for $H(M) := H_1(M) \parallel H_2(M)$ in time $O(n2^{n/2})$.

8.5 Building Compression Functions

The Merkle-Damgård paradigm shows that to construct a collision resistant hash function for long messages, it suffices to construct a collision resistant compression function h for short blocks. In this section we describe a few candidate compression functions. These constructions fall into two categories:

- Compression functions built from a block cipher. The most widely used method is called Davies-Meyer. The SHA family of cryptographic hash functions all use Davies-Meyer.
- Compression functions using number theoretic primitives. These are elegant constructions with clean proofs of security. Unfortunately, they are generally far less efficient than the first method.

8.5.1 A simple but inefficient compression function

We start with a compression function built using modular arithmetic. Let p be a large prime such that $q := (p-1)/2$ is also prime. Let x and y be suitably chosen integers in the range $[1, q]$. Consider the following simple compression function that takes as input two integers in $[1, q]$ and outputs an integer in $[1, q]$:

$$H(a, b) = \text{abs}(x^a y^b \bmod p), \quad \text{where} \quad \text{abs}(z) := \begin{cases} z & \text{if } z \le q, \\ p - z & \text{if } z > q. \end{cases} \tag{8.3}$$

We will show later in Exercise 10.22 that this function is collision resistant assuming a certain standard number theoretic problem is hard. Applying the Merkle-Damgård paradigm to this function gives a collision resistant hash function for arbitrary size inputs. Although this is an elegant collision resistant hash with a clean security proof, it is far less efficient than functions derived from the Davies-Meyer construction and, as a result, is hardly ever used in practice.

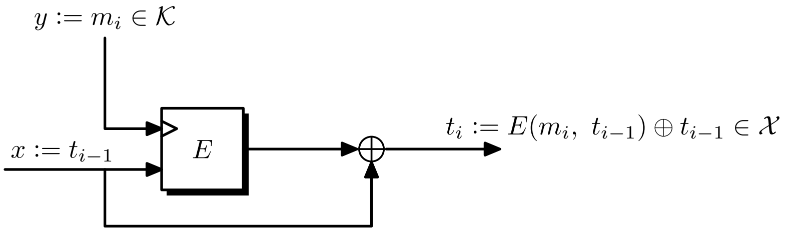

$y := m_i \in \mathcal{K}$
$x := t_{i-1}$
E
$t_i := E(m_i,\ t_{i-1}) \oplus t_{i-1} \in \mathcal{X}$

Figure 8.6: The Davies-Meyer compression function

8.5.2 Davies-Meyer compression functions

In this section we show how to construct a fast compression function from a secure block cipher $\mathcal{E} = (E, D)$. One method, called Davies-Meyer, enables us to do just that. To prove security of the resulting compression function we need to model the block cipher $\mathcal{E}$ as an *ideal cipher*.

Let $\mathcal{E} = (E, D)$ be a block cipher over $(\mathcal{K}, \mathcal{X})$ where $\mathcal{X} = \{0,1\}^n$. The **Davies-Meyer compression function derived from** E maps inputs in $\mathcal{X} \times \mathcal{K}$ to outputs in $\mathcal{X}$. The function is defined as follows:

$$h_{\text{DM}}(x, y) := E(y,\ x) \oplus x$$

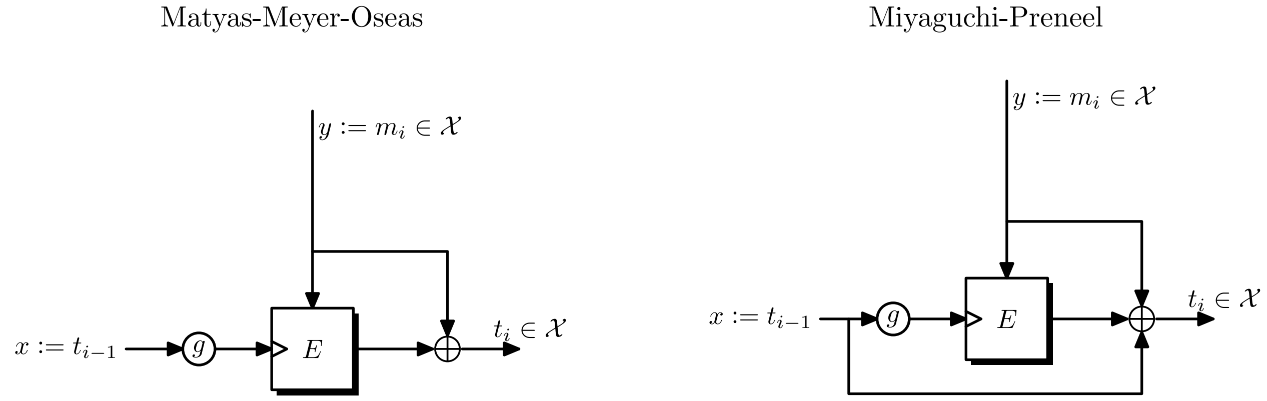
and is illustrated in Fig. 8.6. In symbols, h_{DM} is defined over $(\mathcal{X} \times \mathcal{K},\ \mathcal{X})$.

When plugging this compression function into the Merkle-Damgård paradigm, the inputs are a chaining variable $x := t_{i-1} \in \mathcal{X}$ and a message block $y := m_i \in \mathcal{K}$. The output is the next chaining variable $t_i := E(m_i,\ t_{i-1}) \oplus t_{i-1} \in \mathcal{X}$. Note that the message block is used as the block cipher key which seems a bit odd since the adversary has full control over the message. Nevertheless, we will show that h_{DM} is collision resistant and therefore the resulting Merkle-Damgård function is collision resistant.

When using h_{DM} in Merkle-Damgård the block cipher key (m_i) changes from one message block to the next, which is an unusual way of using a block cipher. Common block ciphers are optimized to encrypt long messages with a fixed key; changing the block cipher key on every block can slow down the cipher. Consequently, using Davies-Meyer with an off-the-shelf block cipher such as AES will result in a relatively slow hash function. Instead, one uses a custom block cipher specifically designed for rapid key changes.

Another reason to not use an off-the-shelf block cipher in Davies-Meyer is that the block size may be too short, for example 128 bits for AES. An AES-based compression function would produce a 128-bit output which is much too short for collision resistance: a collision could be found with only 2^{64} evaluations of the function. In addition, off-the-shelf block ciphers use relatively short keys, say 128 bits long. This would result in Merkle-Damgård processing only 128 message bits per round. Typical ciphers used in Merkle-Damgård hash functions use longer keys (typically, 512-bits or even 1024-bits long) so that many more message bits are processed in every round.

Davies-Meyer variants. The Davies-Meyer construction is not unique. Many other similar methods can convert a block cipher into a collision resistant compression function. For example,


Matyas-Meyer-Oseas

Miyaguchi-Preneel

Figure 8.7: Other block cipher compression functions

one could use

$$\begin{array}{ll} \text{Matyas-Meyer-Oseas:} & h_1(x,y) := E(x,\ y) \oplus y \\ \text{Miyaguchi-Preneel:} & h_2(x,y) := E(x,\ y) \oplus y \oplus x \\ \text{Or even:} & h_3(x,y) := E(x \oplus y,\ y) \oplus y \end{array}$$

or many other such variants. Preneel et al. [132] give twelve different variants that can be shown to be collision resistant.

The Matyas-Meyer-Oseas function h_1 is similar to Davies-Meyer, but reverses the roles of the chaining variable and the message block — in h_1 the chaining variable is used as the block cipher key. The function h_1 maps elements in $(\mathcal{K} \times \mathcal{X})$ to $\mathcal{X}$. Therefore, to use h_1 in Merkle-Damgård we need an auxiliary encoding function $g : \mathcal{X} \to \mathcal{K}$ that maps the chaining variable $t_{i-1} \in \mathcal{X}$ to an element in $\mathcal{K}$, as shown in Fig. 8.7. The same is true for the Miyaguchi-Preneel function h_2. The Davies-Meyer function does not need such an encoding function. We note that the Miyaguchi-Preneel function has a minor security advantage over Davies-Meyer, as discussed in Exercise 8.15.

Many other natural variants of Davies-Meyer are totally insecure. For example, for the following functions

$$h_4(x,y) := E(y,\ x) \oplus y$$
$$h_5(x,y) := E(x,\ x \oplus y) \oplus x$$

we can find collisions in constant time (see Exercise 8.11).

8.5.3 Collision resistance of Davies-Meyer

We cannot prove that Davies-Meyer is collision resistant by assuming a standard complexity assumption about the block cipher. Simply assuming that $\mathcal{E} = (E, D)$ is a secure block cipher is insufficient for proving that h_{DM} is collision resistant. Instead, we have to model the block cipher as an *ideal cipher*.

We introduced the ideal cipher model back in Section 4.7. Recall that this is a heuristic technique in which we treat the block cipher as if it were a family of random permutations. If $\mathcal{E} = (E, D)$ is a block cipher with key space $\mathcal{K}$ and data block space $\mathcal{X}$, then the family of random permutations is $\{\Pi_{k}\}_{k\in\mathcal{K}}$, where each Π_{k} is a truly random permutation on $\mathcal{X}$, and the Π_{k}'s collectively are mutually independent.

Attack Game 8.1 can be adapted to the ideal cipher model, so that before the adversary outputs a collision, it may make a series of Π-queries and Π^{-1}-queries to its challenger.

- For a Π-query, the adversary submits a pair $(k, a) \in \mathcal{K} \times \mathcal{X}$, to which the challenger responds with $b := \Pi_{k}(a)$.
- For a Π^{-1}-query, the adversary submits a pair $(k, b) \in \mathcal{K}\times\mathcal{X}$, to which the challenger responds with $a := \Pi_{k}^{-1}(b)$.

After making these queries, the adversary attempts to output a collision, which in the case of Davies-Meyer, means $(x, y) \neq (x', y')$ such that

$$\Pi_y(x) \oplus x = \Pi_{y'}(x') \oplus x'.$$

The adversary $\mathcal{A}$'s advantage in finding a collision for h_{DM} in the ideal cipher model is denoted $\text{CR}^{\text{ic}}\mathsf{adv}[\mathcal{A}, h_{\text{DM}}]$, and security in the ideal cipher model means that this advantage is negligible for all efficient adversaries $\mathcal{A}$.

Theorem 8.4 (Davies-Meyer). *Let h_{DM} be the Davies-Meyer hash function derived from a block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{X})$, where $|\mathcal{X}|$ is large. Then h_{DM} is collision resistant in the ideal cipher model.*

In particular, every collision finding adversary $\mathcal{A}$ that issues at most q ideal-cipher queries will satisfy

$$\text{CR}^{\text{ic}}\mathsf{adv}[\mathcal{A}, h_{\text{DM}}] \leq (q + 1)(q + 2)/|\mathcal{X}|.$$

The theorem shows that Davies-Meyer is an optimal compression function: the adversary must issue $q = \Omega(\sqrt{|\mathcal{X}|})$ queries (and hence must run for at least that amount of time) if he is to find a collision for h_{DM} with constant probability. No compression function can have higher security due to the birthday attack.

Proof. Let $\mathcal{A}$ be a collision finder for h_{DM} that makes at most a total of q ideal cipher queries. We shall assume that $\mathcal{A}$ is "reasonable" meaning that before $\mathcal{A}$ outputs its collision attempt $(x, y), (x', y')$, it makes corresponding ideal cipher queries: for (x, y), either a Π-query on (y, x) or a Π^{-1}-query on $(y, \cdot)$ that yields x, and similarly for (x', y'). If $\mathcal{A}$ is not already reasonable, we can make it so by increasing total number of queries to at most $q' := q + 2$. So from now on we will assume $\mathcal{A}$ is reasonable and makes at most q' ideal cipher queries.

For $i = 1, \ldots, q'$, the ith ideal cipher query defines a triple (k_i, a_i, b_i): for a Π-query (k_i, a_i), we set $b_i := \Pi_{k_i}(a_i)$, and for a Π^{-1}-query (k_i, b_i), we set $a_i := \Pi_{k_i}^{-1}(b_i)$. We assume that $\mathcal{A}$ makes no extraneous queries, so that no triples repeat.

If the adversary outputs a collision, then by our reasonableness assumption, for some distinct pair of indices $i, j = 1, \ldots, q'$, we have $a_i \oplus b_i = a_j \oplus b_j$. Let us call this event Z. So we have

$$\text{CR}^{\text{ic}}\mathsf{adv}[\mathcal{A}, h_{\text{DM}}] \leq \Pr[Z].$$

Our goal is to show

$$\Pr[Z] \leq \frac{q'(q' - 1)}{2^n}, \tag{8.4}$$

where $|\mathcal{X}| = 2^n$.

Consider any fixed indices $i < j$. Conditioned on any fixed values of the adversary's coins and the first $j-1$ triples, one of a_j and b_j is completely fixed, while the other is uniformly distributed over a set of size at least $|\mathcal{X}| - j + 1$. Therefore,

$$\Pr[a_i \oplus b_i = a_j \oplus b_j] \leq \frac{1}{2^n - j + 1}.$$

So by the union bound, we have

$$\Pr[Z] \leq \sum_{j=1}^{q'} \sum_{i=1}^{j-1} \Pr[a_i \oplus b_i = a_j \oplus b_j] \leq \sum_{j=1}^{q'} \frac{j-1}{2^n - j + 1} \leq \sum_{j=1}^{q'} \frac{j-1}{2^n - q'} = \frac{q'(q'-1)}{2(2^n - q')}. \tag{8.5}$$

For $q' \leq 2^{n-1}$ this bound simplifies to $\Pr[Z] \leq q'(q'-1)/2^n$. For $q' > 2^{n-1}$ the bound holds trivially. Therefore, (8.4) holds for all q'. □

8.6 Case study: SHA256

The Secure Hash Algorithm (SHA) was published by NIST in 1993 [FIPS 180] as part of the design specification of the Digital Signature Standard (DSS). This hash function, often called **SHA0**, outputs 160-bit digests. Two years later, in 1995, NIST updated the standard [FIPS 180-1] by adding one extra instruction to the compression function. The resulting function is called **SHA1**. NIST gave no explanation for this change, but it was later found that this extra instruction is crucial for collision resistance. SHA1 became the de-facto standard for collision resistant hashing and was widely deployed.

The birthday attack can find collisions for SHA1 using an expected 2^{80} evaluations of the function. In 2002 NIST added [FIPS 180-2] two new hash functions to the SHA family: **SHA256** and **SHA512**. They output larger digests (256 and 512-bit digests respectively) and therefore provide better protection against the birthday attack. NIST also approved SHA224 and SHA384 which are obtained from SHA256 and SHA512 respectively by truncating the output to 224 and 384 bits. These and a few other proposed hash functions are summarized in Table 8.1.

The years 2004–5 were bad years for collision resistant hash functions. A number of new attacks showed how to find collisions for several hash functions. In particular, Wang, Yao, and Yao [155] presented a collision finder for SHA1 that uses 2^{63} evaluations of the function — far less than the birthday attack. The first collision for SHA1, using an improved algorithm, was found in 2017. As a result SHA1 is no longer considered collision resistant, and should not be used. The current recommended practice is to use SHA256 which we describe here.

The SHA256 function. SHA256 is a Merkle-Damgård hash function using a Davies-Meyer compression function h. This h takes as input a 256-bit chaining variable t and a 512-bit message block m. It outputs a 256-bit chaining variable.

We first describe the SHA256 Merkle-Damgård chain. Recall that the padding block PB in our description of Merkle-Damgård contained a 64-bit encoding of the number of *blocks* in the message being hashed. The same is true for SHA256 with the minor difference that PB encodes the number

[2] Performance numbers were provided by Wei Dai using the Crypto++ 5.6.0 benchmarks running on a 1.83 GHz Intel Core 2 processor. Higher numbers are better.

		digest	message	Speed[2]	best known
Name	year	size	block size	MB/sec	attack time
SHA0	1993	160	512		2^{39}
SHA1	1995	160	512	153	2^{63}
SHA224	2004	224	512		
SHA256	2002	256	512	111	
SHA384	2002	384	1024		
SHA512	2002	512	1024	99	
MD4	1990	128	512		2^{1}
MD5	1992	128	512	255	2^{16}
Whirlpool	2000	512	512	57	

Table 8.1: Merkle-Damgård collision resistant hash functions

of *bits* in the message. Hence, SHA256 can hash messages that are at most $2^{64} - 1$ bits long. The Merkle-Damgård Initial Value (IV) in SHA256 is set to:

$$\text{IV} := \text{6A09E667 BB67AE85 3C6EF372 A54FF53A 510E527F 9B05688C 1F83D9AB 5BE0CD19} \in \{0,1\}^{256}$$

written in base 16.

Clearly the output of SHA256 can be truncated to obtain shorter digests at the cost of reduced security. This is, in fact, how the SHA224 hash function works — it is identical to SHA256 with two exceptions: (1) SHA224 uses a different initialization vector IV, and (2) SHA224 truncates the output of SHA256 to its left most 224 bits.

Next, we describe the SHA256 Davies-Meyer compression function h. It is built from a block cipher which we denote by E_{SHA256}. However, instead of using XOR as in Davies-Meyer, SHA256 uses addition modulo 2^{32}. That is, let

$$x_0, x_1, \ldots, x_7 \in \{0,1\}^{32} \qquad \text{and} \qquad y_0, y_1, \ldots, y_7 \in \{0,1\}^{32}$$

and set

$$x := x_0 \parallel \cdots \parallel x_7 \ \in \{0,1\}^{256} \qquad \text{and} \qquad y := y_0 \parallel \cdots \parallel y_7 \ \in \{0,1\}^{256}.$$

Define: $\quad x \boxplus y := (x_0 + y_0) \parallel \cdots \parallel (x_7 + y_7) \ \in \{0,1\}^{256}$ where all additions are modulo 2^{32}. Then the SHA256 compression function h is defined as:

$$h(t, m) \ := \ E_{\text{SHA256}}(m, t) \ \boxplus \ t \quad \in \{0,1\}^{256}.$$

Our ideal cipher analysis of Davies-Meyer (Theorem 8.4) applies equally well to this modified function.

The SHA256 block cipher. To complete the description of SHA256 it remains to describe the block cipher E_{SHA256}. The algorithm makes use of a few auxiliary functions defined in Table 8.2. Here, SHR and ROTR denote the standard shift-right and rotate-right functions.

For x, y, z in $\{0,1\}^{32}$ define:

$$\begin{array}{rcll}
\mathrm{SHR}^n(x) & := & (x >> n) & \text{(Shift Right)} \\
\mathrm{ROTR}^n(x) & := & (x >> n) \vee (x << 32 - n) & \text{(Rotate Right)} \\[4pt]
\mathrm{Ch}(x,y,z) & := & (x \wedge y) \oplus (\neg x \wedge z) & \\
\mathrm{Maj}(x,y,z) & := & (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & \\[4pt]
\Sigma_0(x) & := & \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x) & \\
\Sigma_1(x) & := & \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x) & \\
\sigma_0(x) & := & \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) & \\
\sigma_1(x) & := & \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x) &
\end{array}$$

Table 8.2: Functions used in the SHA256 block cipher

The cipher E_{SHA256} takes as input a 512-bit key k and a 256-bit message t. We first break both the key and the message into 32-bit words. That is, write:

$$k := k_0 \parallel k_1 \parallel \cdots \parallel k_{15} \quad \in \{0,1\}^{512}$$
$$t := t_0 \parallel t_1 \parallel \cdots \parallel t_7 \quad \in \{0,1\}^{256}$$

where each k_i and t_i is in $\{0,1\}^{32}$.

The code for E_{SHA256} is shown in Table 8.3. It iterates the same round function 64 times. In each round the cipher uses a round key $W_i \in \{0,1\}^{32}$ defined recursively during the key setup step. One cipher round, shown in Fig. 8.8, looks like two adjoined Feistel rounds. The cipher uses 64 fixed constants $K_0, K_1, \ldots, K_{63} \in \{0,1\}^{32}$ whose values are specified in the SHA256 standard. For example, $K_0 := 428A2F98$ and $K_1 := 71374491$, written base 16.

Interestingly, NIST never gave the block cipher E_{SHA256} an official name. The cipher was given the unofficial name **SHACAL-2** by Handschuh and Naccache (submission to NESSIE, 2000). Similarly, the block cipher underlying SHA1 is called SHACAL-1. The SHACAL-2 block cipher is identical to E_{SHA256} with the only difference that it can encrypt using keys shorter than 512 bits. Given a key $k \in \{0,1\}^{\leq 512}$ the SHACAL-2 cipher appends zeros to the key to get a 512-bit key. It then applies E_{SHA256} to the given 256-bit message block. Decryption in SHACAL-2 is similar to encryption. This cipher is well suited for applications where SHA256 is already implemented, thus reducing the overall size of the crypto code.

8.6.1 Other Merkle-Damgård hash functions

MD4 and MD5. Both cryptographic hash functions were designed by Ron Rivest in 1990–1 [134, 135]. Both are Merkle-Damgård hash functions that output a 128-bit digest. They are quite similar, although MD5 uses a stronger compression function than MD4. Collisions for both hash functions can be found efficiently as described in Table 8.1. Consequently, these hash functions are no longer used.

Whirlpool. Whirlpool was designed by Barreto and Rijmen in 2000 and was adopted as an ISO/IEC standard in 2004. Whirlpool is a Merkle-Damgård hash function. Its compression function

Input: plaintext $t = t_0 \parallel \cdots \parallel t_7 \in \{0,1\}^{256}$ and
key $k = k_0 \parallel k_1 \parallel \cdots \parallel k_{15} \in \{0,1\}^{512}$

Output: ciphertext in $\{0,1\}^{256}$.

// *Here all additions are modulo* 2^{32}.
// *The algorithm uses constants* $K_0, K_1, \ldots, K_{63} \in \{0,1\}^{32}$

Key setup: Construct 64 round keys $W_0, \ldots, W_{63} \in \{0,1\}^{32}$:

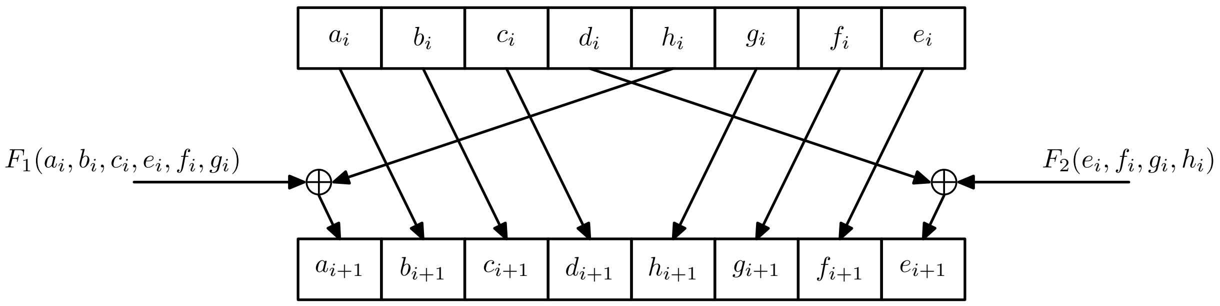
$$\begin{cases} \text{for } i = 0,1,\ldots,15 & \text{set} \quad W_i \leftarrow k_i, \\ \text{for } i = 16,17,\ldots,63 & \text{set} \quad W_i \leftarrow \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \end{cases}$$

64 Rounds:

```
$(a_0,\ b_0,\ c_0,\ d_0,\ e_0,\ f_0,\ g_0,\ h_0) \ \leftarrow\ (t_0,\ t_1,\ t_2,\ t_3,\ t_4,\ t_5,\ t_6,\ t_7)$
for $i = 0$ to 63 do:
    $T_1 \leftarrow h_i + \Sigma_1(e_i) + \mathrm{Ch}(e_i, f_i, g_i) + K_i + W_i$
    $T_2 \leftarrow \Sigma_0(a_i) + \mathrm{Maj}(a_i, b_i, c_i)$
    $(a_{i+1},\ b_{i+1},\ c_{i+1},\ d_{i+1},\ e_{i+1},\ f_{i+1},\ g_{i+1},\ h_{i+1}) \ \leftarrow$
                        $(T_1 + T_2,\ a_i,\ b_i,\ c_i,\ d_i + T_1,\ e_i,\ f_i,\ g_i)$
```

Output: $a_{64} \parallel b_{64} \parallel c_{64} \parallel d_{64} \parallel e_{64} \parallel f_{64} \parallel g_{64} \parallel h_{64} \in \{0,1\}^{256}$

Table 8.3: The SHA256 block cipher



$$F_1(a,b,c,e,f,g) := \Sigma_1(e) + \mathrm{Ch}(e,f,g) + \Sigma_0(a) + \mathrm{Maj}(a,b,c) + K_i + W_i$$

$$F_2(e,f,g,h) := h + \Sigma_1(e) + \mathrm{Ch}(e,f,g) + K_i + W_i$$

Figure 8.8: One round of the SHA256 block cipher

uses the Miyaguchi-Preneel method (Fig. 8.7) with a block cipher called W. This block cipher is very similar to AES, but has a 512-bit block size. The resulting hash output is 512-bits.

Others. Many other Merkle-Damgård hash functions were proposed in the literature. Two examples are Tiger/192 [20] and RIPEMD-160.

8.7 Case study: HMAC

In this section, we return to our problem of building a secure MAC that works on long messages. Merkle-Damgård hash functions such as SHA256 are widely deployed. Most Crypto libraries include an implementation of multiple Merkle-Damgård functions. Furthermore, these implementations are very fast: one can typically hash a very long message with SHA256 much faster than one can apply, say, CBC-MAC with AES to the same message.

Of course, one might use the hash-then-MAC construction analyzed in Section 8.2. Recall that in this construction, we combine a secure MAC system $\mathcal{I} = (S, V)$ and a collision resistant hash function H, so that the resulting signing algorithm signs a message m by first hashing m using H to get a short digest $H(m)$, and then signs $H(m)$ using S to obtain the MAC tag $t = S(k, H(m))$. As we saw in Theorem 8.1 the resulting construction is secure. However, this construction is not very widely deployed. Why?

First of all, as discussed after the statement of Theorem 8.1, if one can find collisions in H, then the hash-then-MAC construction is completely broken. A collision-finding attack, such as a birthday attack (Section 8.3), or a more sophisticated attack, can be carried out entirely *offline*, that is, without the need to interact with any users of the system. In contrast, *online* attacks require many interactions between the adversary and honest users of the system. In general, offline attacks are considered especially dangerous since an adversary can invest huge computing resources over an extended period of time: in an attack on hash-then-MAC, an attacker could spend months quietly computing on many machines to find a collision on H, without arousing any suspicions.

Another reason not to use the hash-then-MAC construction directly is that we need both a hash function H and a MAC system $\mathcal{I}$. So an implementation might need software and/or hardware to execute both, say, SHA256 for the hash and CBC-MAC with AES for the MAC. All other things being equal, it would be nice to simply use one algorithm as the basis for a MAC.

This leads us to the following problem: how to take a *keyless* Merkle-Damgård hash function, such as SHA256, and use it somehow to implement a *keyed* function that is a secure MAC, or even better, a secure PRF. Moreover, we would like to be able to prove the security of this construction under an assumption that is (qualitatively, at least) weaker than collision resistance; in particular, the construction should not be susceptible to an offline collision-finding attack on the underlying compression function.

Assume that H is a Merkle-Damgård hash built from a compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. A few simple approaches come to mind.

Prepend the key: $F_{\text{pre}}(k, M) := H(k \parallel M)$. This is completely insecure, because of the following *extension attack*: given $F_{\text{pre}}(k, M)$, one can easily compute $F_{\text{pre}}(k, M \parallel \text{PB} \parallel M')$ for any M'. Here, PB is the Merkle-Damgård padding block for the message $k \parallel M$. Aside from this extension attack, the construction is secure, under reasonable assumptions (see Exercise 8.18).

Append the key: $F_{\text{post}}(k, M) := H(M \parallel k)$. This is somewhat similar to the hash-then-MAC construction, and relies on the collision resistance of h. Indeed, it is vulnerable to an offline collision-finding attack: assuming we find two distinct ℓ-bit strings M_0 and M_1 such that $h(\text{IV}, M_0) = h(\text{IV}, M_1)$, then we have $F_{\text{post}}(k, M_0) = F_{\text{post}}(k, M_1)$. For these reasons, this construction does not solve our problem. However, under the right assumptions (including the collision resistance of h, of course), we can still get a security proof (see Exercise 8.19).

Envelope method: $F_{\text{env}}(k, M) := H(k \parallel M \parallel k)$. Under reasonable pseudorandomness assumptions on h, and certain formatting assumptions (that k is an ℓ-bit string and M is padded out to a bit string whose length is a multiple of ℓ), this can be proven to be a secure PRF. See Exercise 8.17.

Two-key nest: $F_{\text{nest}}((k_1, k_2), M) := H(k_2 \parallel H(k_1 \parallel M))$. Under reasonable pseudorandomness assumptions on h, and certain formatting assumptions (that k_1 and k_2 are ℓ-bit strings), this can also be proven to be a secure PRF.

The two-key nest is very closely related to a classic MAC construction known as **HMAC**. HMAC is the most widely deployed MAC on the Internet. It is used in SSL, TLS, IPsec, SSH, and a host of other security protocols. TLS and IPsec also use HMAC as a means for deriving session keys during session setup. We will give a security analysis of the two-key nest, and then discuss its relation to HMAC.

8.7.1 Security of two-key nest

We will now show that the two-key nest is indeed a secure PRF, under appropriate pseudorandomness assumptions on h. Let us start by "opening up" the definition of $F_{\text{nest}}((k_1, k_2), M)$, using the fact that H is a Merkle-Damgård hash built from h. See Fig. 8.9. The reader should study this figure carefully. We are assuming that the keys k_1 and k_2 are ℓ-bit strings, so they each occupy one full message block. The input to the inner evaluation of H is the padded string $k_1 \parallel M \parallel \text{PB}_{\text{in}}$, which is broken into ℓ-bit blocks as shown. The output of the inner evaluation of H is the n-bit string t. The input to the outer evaluation of H is the padded string $k_2 \parallel t \parallel \text{PB}_{\text{out}}$. We shall assume that n is significantly smaller than ℓ, so that $t \parallel \text{PB}_{\text{out}}$ is a single ℓ-bit block, as shown in the figure.

We now state the pseudorandomness assumptions we need. We define the following two PRFs h_{bot} and h_{top} derived from h:

$$h_{\text{bot}}(k, m) := h(k, m) \qquad \text{and} \qquad h_{\text{top}}(k, m) := h(m, k). \tag{8.6}$$

For the PRF h_{bot}, the PRF key k is viewed as the first input to h, i.e., the n-bit chaining variable input, which is the *bottom* input to the h-boxes in Fig. 8.9. For the PRF h_{top}, the PRF key k is viewed as the second input to h, i.e., the ℓ-bit message block input, which is the *top* input to the h-boxes in the figure. To make the figure easier to understand, we have decorated the h-box inputs with a $>$ symbol, which indicates which input is to be viewed as a PRF key. Indeed, the reader will observe that we will treat the two evaluations of h that appear within the dotted boxes as evaluations of the PRF h_{top}, so that the values labeled k_1' and k_2' in the figure are computed as $k_1' \leftarrow h_{\text{top}}(k_1, \text{IV})$ and $k_2' \leftarrow h_{\text{top}}(k_2, \text{IV})$. All of the other evaluations of h in the figure will be treated as evaluations of h_{bot}.

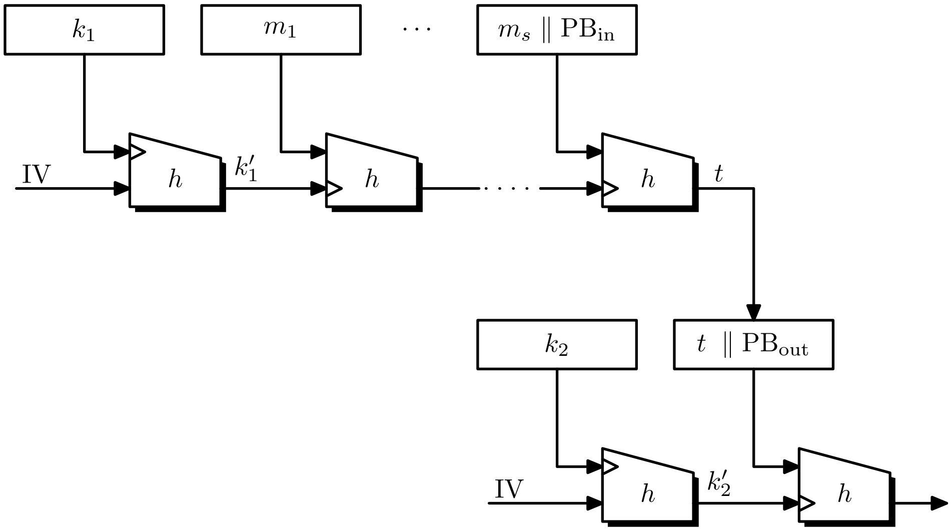

k_1
m_1
$\cdots$
$m_s \parallel \mathrm{PB_{in}}$
IV
h
k'_1
h
h
t
k_2
$t \parallel \mathrm{PB_{out}}$
IV
h
k'_2
h

Figure 8.9: The two-key nest

Our assumption will be that h_{bot} and h_{top} are both secure PRFs. Later, we will use the ideal cipher model to justify this assumption for the Davies-Meyer compression function (see Section 8.7.3).

We will now sketch a proof of the following result:

> *If* h_{bot} *and* h_{top} *are secure PRFs, then so is the two-key nest.*

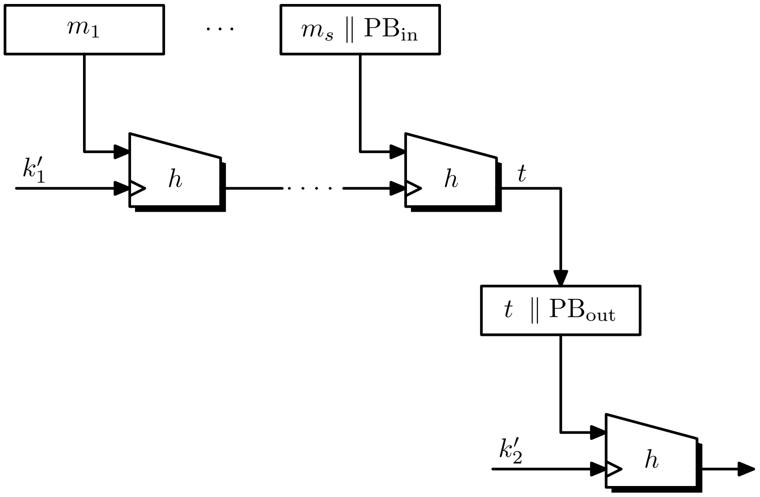
The first observation is that the keys k_1 and k_2 are only used to derive k'_1 and k'_2 as $k'_1 = h_{\mathrm{top}}(k_1, \mathrm{IV})$ and $k'_2 = h_{\mathrm{top}}(k_2, \mathrm{IV})$. The assumption that h_{top} is a secure PRF means that in the PRF attack game, we can effectively replace k'_1 and k'_2 by truly random n-bit strings. The resulting construction is drawn in Fig. 8.10. All we have done here is to throw away all of the elements in Fig. 8.9 that are within the dotted boxes. The function in this new construction takes as input the two keys k'_1 and k'_2 and a message M. By the above observations, it suffices to prove that the construction in Fig. 8.10 is a secure PRF.

Hopefully (without reading the caption), the reader will recognize the construction in Fig. 8.10 as none other than NMAC applied to h_{bot}, which we introduced in Section 6.5.1 (in particular, take a look at Fig. 6.5b). Actually, the construction in Fig. 8.10 is a bit-wise version of NMAC, obtained from the block-wise version via padding (as discussed in Section 6.8). Thus, security for the two-key nest now follows directly from the NMAC security theorem (Theorem 6.7) and the assumption that h_{bot} is a secure PRF.

8.7.2 The HMAC standard

The HMAC standard is exactly the same as the two-key nest (Fig. 8.9), but with one important difference: the keys k_1 and k_2 are not independent, but rather, are derived in a somewhat *ad hoc* way from a single key k.

To describe this in more detail, we first observe that HMAC itself is somewhat byte oriented, so all strings are byte strings. Message blocks for the underlying Merkle-Damgård hash are assumed to be B bytes (rather than ℓ bits). A key k for HMAC is a byte string of arbitrary length. To



m_1
$\cdots$
$m_s \parallel \mathrm{PB}_{\mathrm{in}}$
k'_1
h
h
t
$t \parallel \mathrm{PB}_{\mathrm{out}}$
k'_2
h

Figure 8.10: A bit-wise version of NMAC

derive the keys k_1 and k_2, which are byte strings of length B, we first make k exactly B bytes long: if the length of k is less than or equal to B, we pad it out with zero bytes; otherwise, we replace it with $H(k)$ padded with zero bytes. Then we compute

$$k_1 \leftarrow k \oplus \mathsf{ipad} \quad \text{and} \quad k_2 \leftarrow k \oplus \mathsf{opad},$$

where ipad and opad ("i" and "o" stand for "inner" and "outer") are B-byte constant strings, defined as follows:

$$\begin{aligned} \mathsf{ipad} &= \text{the byte 0x36 repeated } B \text{ times} \\ \mathsf{opad} &= \text{the byte 0x5C repeated } B \text{ times} \end{aligned}$$

HMAC implemented using a hash function H is denoted HMAC-H. The most common HMACs used in practice are HMAC-SHA1 and HMAC-SHA256. The HMAC standard also allows the output of HMAC to be truncated. For example, when truncating the output of SHA1 to 80 bits, the HMAC function is denoted HMAC-SHA1-80. Implementations of TLS 1.0, for example, are required to support HMAC-SHA1-96.

Security of HMAC. Since the keys k'_1, k'_2 are related — their XOR is equal to $\mathsf{opad} \oplus \mathsf{ipad}$ — the security proof we gave for the two-key nest no longer applies: under the stated assumptions, we cannot justify the claim that the derived keys k'_1, k'_2 are indistinguishable from random. One solution is to make a stronger assumption about the compression function h – one needs to assume that h_{top} remains a PRF under a related key attack (as defined by Bellare and Kohno [12]). If h is itself a Davies-Meyer compression function, then this stronger assumption can be justified in the ideal cipher model.

8.7.3 Davies-Meyer is a secure PRF in the ideal cipher model

It remains to justify our assumption that the PRFs h_{bot} and h_{top} derived from h in (8.6) are secure. Suppose the compression function h is a Davies-Meyer function, that is $h(x, y) := E(y,\ x) \oplus x$ for some block cipher $\mathcal{E} = (E, D)$. Then

- $h_{\text{bot}}(k, m) := h(k, m) = E(m, k) \oplus k$ is a PRF defined over$(\mathcal{X}, \mathcal{K}, \mathcal{X})$, and
- $h_{\text{top}}(k, m) := h(m, k) = E(k, m) \oplus m$ is a PRF defined over$(\mathcal{K}, \mathcal{X}, \mathcal{X})$

When $\mathcal{E}$ is a secure block cipher, the fact that h_{top} is a secure PRF is trivial (see Exercise 4.1 part (c)). The fact that h_{bot} is a secure PRF is a bit surprising — the message m given as input to h_{bot} is used as the key for E. But m is chosen by the adversary and hence E is evaluated with a key that is completely under the control of the adversary. As a result, even though E is a secure block cipher, there is no security guarantee for h_{bot}. Nevertheless, we can prove that h_{bot} is a secure PRF, but this requires the ideal cipher model. Just assuming that $\mathcal{E}$ is a secure block cipher is insufficient.

If necessary, the reader should review the basic concepts regarding the ideal cipher model, which was introduced in Section 4.7. We also used the ideal cipher model earlier in this chapter (see Section 8.5.3).

In the ideal cipher model, we heuristically model a block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{X})$ as a family of random permutations $\{\Pi_{\mathscr{k}}\}_{\mathscr{k} \in \mathcal{K}}$. We adapt the PRF Attack Game 4.2 to work in the ideal cipher model. The challenger, in addition to answering standard queries, also answers Π-queries and Π^{-1}-queries: a Π-query is a pair $(\mathscr{k}, \mathscr{a})$ to which the challenger responds with $\mathscr{b} := \Pi_{\mathscr{k}}(\mathscr{a})$; a Π^{-1}-query is a pair $(\mathscr{k}, \mathscr{b})$ to which is the challenger responds with $\mathscr{a} := \Pi_{\mathscr{k}}^{-1}(\mathscr{b})$. For a standard query m, the challenger responds with $v := f(m)$: in Experiment 0 of the attack game, f is $F(k, \cdot)$, where F is a PRF and k is a randomly chosen key; in Experiment 1, f is a truly random function. Moreover, in Experiment 0, F is evaluated using the random permutations in the role of E and D used in the construction of F. For our PRF $h_{\text{bot}}(k, m) = E(m, k) \oplus k = \Pi_m(k) \oplus k$.

For an adversary $\mathcal{A}$, we define $\text{PRF}^{\text{ic}}\mathsf{adv}[\mathcal{A}, F]$ to be the advantage in the modified PRF attack game, and security in the ideal cipher model means that this advantage is negligible for all efficient adversaries.

Theorem 8.5 (Security of h_{bot}). *Let $\mathcal{E} = (E, D)$ be a block cipher over $(\mathcal{K}, \mathcal{X})$, where $|\mathcal{X}|$ is large. Then $h_{\text{bot}}(k, m) := E(m, k) \oplus k$ is a secure PRF in the ideal cipher model.*

> *In particular, for every PRF adversary $\mathcal{A}$ attacking h_{bot} and making at most a total of Q_{ic} ideal cipher queries, we have*
>
> $$\text{PRF}^{\text{ic}}\mathsf{adv}[\mathcal{A}, h_{\text{bot}}] \leq \frac{2Q_{\text{ic}}}{|\mathcal{X}|}.$$

The bound in the theorem is fairly tight, as brute-force key search gets very close to this bound.

Proof. The proof will mirror the analysis of the Evan-Mansour/$\mathcal{E}X$ constructions (see Theorem 4.14 in Section 4.7.4), and in particular, will make use of the Domain Separation Lemma (see Theorem 4.15, also in Section 4.7.4).

Let $\mathcal{A}$ be an adversary as in the statement of the theorem. Let p_b be the probability that $\mathcal{A}$ outputs 1 in Experiment b of Attack Game 4.2, for $b = 0, 1$. So by definition we have

$$\text{PRF}^{\text{ic}}\mathsf{adv}[\mathcal{A}, h_{\text{bot}}] = |p_0 - p_1|. \tag{8.7}$$

We shall prove the theorem using a sequence of two games, applying the Domain Separation Lemma.

Game 0. The game will correspond to Experiment 0 of the PRF attack game in the ideal cipher model. We can write the logic of the challenger as follows:

```
Initialize:
    for each $\mathscr{k} \in \mathcal{K}$, set $\Pi_{\mathscr{k}} \overset{R}{\leftarrow} \text{Perms}[\mathcal{X}]$
    $k \overset{R}{\leftarrow} \mathcal{X}$

standard $h_{\text{bot}}$-query $m$:
    $c \leftarrow \Pi_m(k)$
    $v \leftarrow c \oplus k$
    return $v$
```

The challenger in Game 0 processes ideal cipher queries *exactly as in Game 0 of the proof of Theorem 4.14*:

```
ideal cipher $\Pi$-query $\mathscr{k}, \mathscr{a}$:
    $\mathscr{b} \leftarrow \Pi_{\mathscr{k}}(\mathscr{a})$
    return $\mathscr{b}$

ideal cipher $\Pi^{-1}$-query $\mathscr{k}, \mathscr{b}$:
    $\mathscr{a} \leftarrow \Pi_{\mathscr{k}}^{-1}(\mathscr{b})$
    return $\mathscr{a}$
```

Let W_0 be the event that $\mathcal{A}$ outputs 1 at the end of Game 0. It should be clear from construction that

$$\Pr[W_0] = p_0. \tag{8.8}$$

Game 1. Just as in the proof of Theorem 4.14, we declare "by fiat" that standard queries and ideal cipher queries are processed using independent random permutations. In detail (changes from Game 0 are highlighted):

```
Initialize:
    for each $\mathscr{k} \in \mathcal{K}$, set $\Pi_{\text{std},\mathscr{k}} \overset{R}{\leftarrow} \text{Perms}[\mathcal{X}]$ and $\Pi_{\text{ic},\mathscr{k}} \overset{R}{\leftarrow} \text{Perms}[\mathcal{X}]$
    $k \overset{R}{\leftarrow} \mathcal{X}$

standard $h_{\text{bot}}$-query $m$:
    $c \leftarrow \Pi_{\text{std},m}(k)$    // add $k$ to sampled domain of $\Pi_{\text{std},m}$, add $c$ to sampled range of $\Pi_{\text{std},m}$
    $v \leftarrow c \oplus k$
    return $v$
```

The challenger in Game 1 processes ideal cipher queries *exactly as in Game 1 of the proof of Theorem 4.14*:

```
ideal cipher $\Pi$-query $\mathscr{k}, \mathscr{a}$:
    $\mathscr{b} \leftarrow \Pi_{\text{ic},\mathscr{k}}(\mathscr{a})$    // add $\mathscr{a}$ to sampled domain of $\Pi_{\text{ic},\mathscr{k}}$, add $\mathscr{b}$ to sampled range of $\Pi_{\text{ic},\mathscr{k}}$
    return $\mathscr{b}$
```

```
ideal cipher $\Pi^{-1}$-query $\mathscr{k}, \mathscr{b}$:
1.    $\mathscr{a} \leftarrow \Pi^{-1}_{\mathrm{ic},\mathscr{k}}(\mathscr{b})$    // add $\mathscr{a}$ to sampled domain of $\Pi_{\mathrm{ic},\mathscr{k}}$,  add $\mathscr{b}$ to sampled range of $\Pi_{\mathrm{ic},\mathscr{k}}$
2.    return $\mathscr{a}$
```

Let W_1 be the event that $\mathcal{A}$ outputs 1 at the end of Game 1. Consider an input/output pair (m, v) for a standard query in Game 1. Observe that k is the only item ever added to the sampled domain of $\Pi_{\mathrm{std},m}(k)$, and $c = v \oplus k$ is the only item ever added to the sampled range of $\Pi_{\mathrm{std},m}(k)$. In particular, c is generated at random and k remains perfectly hidden (i.e., is independent of the adversary's view).

Thus, from the adversary's point of view, the standard queries behave identically to a random function, and the ideal cipher queries behave like ideal cipher queries for an *independent* ideal cipher. In particular, we have

$$\Pr[W_1] = p_1. \tag{8.9}$$

Finally, we use the Domain Separation Lemma to analyze $|\Pr[W_0] - \Pr[W_1]|$. The domain separation failure event Z is the event that in Game 1, the sampled domain of one of the $\Pi_{\mathrm{std},m}$'s overlaps with the sampled domain of one of the $\Pi_{\mathrm{ic},\mathscr{k}}$'s, or the sampled range of one of the $\Pi_{\mathrm{std},m}$'s overlaps with the sampled range of one of the $\Pi_{\mathrm{ic},\mathscr{k}}$'s. The Domain Separation Lemma tells us that

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z]. \tag{8.10}$$

If Z occurs, then for some input/output triple $(\mathscr{k}, \mathscr{a}, \mathscr{b})$ corresponding to an ideal cipher query, $\mathscr{k} = m$ was the input to a standard query with output v, and either

(i) $\mathscr{a} = k$, or

(ii) $\mathscr{b} = v \oplus k$.

For any fixed triple $(\mathscr{k}, \mathscr{a}, \mathscr{b})$, by the independence of k, conditions (i) and (ii) each hold with probability $1/|\mathcal{X}|$, and so by the union bound

$$\Pr[Z] \leq \frac{2Q_{\mathrm{ic}}}{|\mathcal{X}|}. \tag{8.11}$$

The theorem now follows from (8.7)–(8.11). □

8.8 The Sponge Construction and SHA3

For many years, essentially all collision resistant hash functions were based on the Merkle-Damgård paradigm. Recently, however, an alternative paradigm has emerged, called the **sponge construction**. Like Merkle-Damgård, it is a simple iterative construction built from a more primitive function; however, instead of a compression function $h : \{0,1\}^{n+\ell} \to \{0,1\}^n$, a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$ is used. We stress that unlike a block cipher, the function π has no key. There are two other high-level differences between the sponge and Merkle-Damgård that we should point out:

- On the negative side, it is not known how to reduce the collision resistance of the sponge to a concrete security property of π. The only known analysis of the sponge is in the ideal permutation model, where we (heuristically) model π as a truly random permutation Π.

- On the positive side, the sponge is designed to be used flexibly and securely in a variety of applications where collision resistance is not the main property we need. For example, in Section 8.7, we looked at several possible ways to convert a hash function H into a PRF F. We saw, in particular, that the intuitive idea of simply prepending the key, defining $F_{\text{pre}}(k, M) := H(k \parallel M)$, does not work when H is instantiated with a Merkle-Damgård hash. The sponge avoids these problems: it allows one to hash variable length inputs to variable length outputs, and if we model π as a random permutation, then one can argue that for all intents and purposes, the sponge is a random function (we will discuss this in more detail in Section 8.10). In particular, the construction F_{pre} is secure when H is instantiated with a sponge hash.

A new hash standard, called SHA3, is based on the sponge construction. After giving a description and analysis of the general sponge construction, we discuss some of the particulars of SHA3.

8.8.1 The sponge construction

We now describe the sponge construction. In addition to specifying a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$, we need to specify two positive integers r and c such that $n = r + c$. The number r is called the **rate** of the sponge: larger rate values lead to faster evaluation. The number c is called the **capacity** of the sponge: larger capacity values lead to better security bounds. Thus, different choices of r and c lead to different speed/security trade-offs.

The sponge allows variable length inputs. To hash a long message $M \in \{0,1\}^{\leq L}$, we first append a padding string to M to make its length a multiple of r, and then break the padded M into a sequence of r-bit blocks $m_1, \ldots, m_s$. The requirements of the padding procedure are minimal: it just needs to be injective. Just adding a string of the form 10^* suffices, although in SHA3 a pad of the form 10^*1 is used: this latter padding has the effect of encoding the rate in the last block and helps to analyze security in applications that use the same sponge with different rates; however, we will not explore these use cases here. Note that an entire dummy block may need to be added if the length of M is already at or near a multiple of r.

The sponge allows variable length outputs. So in addition to a message $M \in \{0,1\}^{\leq L}$ as above, it takes as input a positive integer v, which specifies the number of output bits.

Here is how the sponge works:

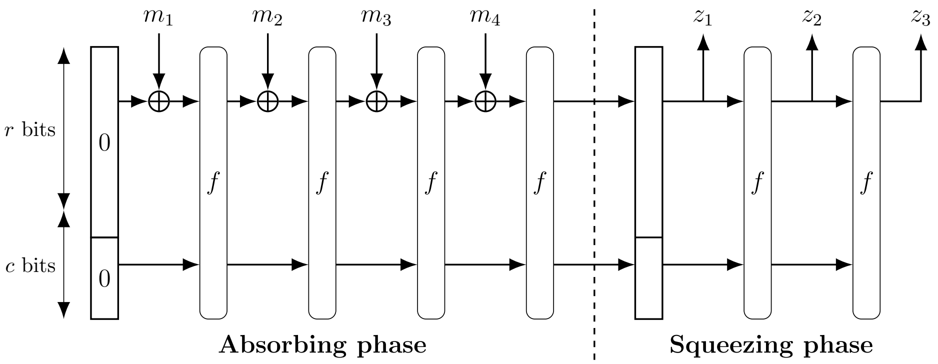


Figure 8.11: The sponge construction

```
Input: message $M \in \{0,1\}^{\leq L}$ and desired output length $v > 0$
Output: a tag $h \in \{0,1\}^v$
        // Absorbing stage
        pad $M$ and break into $r$-bit blocks $m_1, \ldots, m_s \in \{0,1\}^r$
        $h \leftarrow 0^n$
        for $i \leftarrow 1$ to $s$ do
                $m'_i \leftarrow m_i \parallel 0^c \ \in \{0,1\}^n$
                $h \leftarrow \pi(h \oplus m'_i)$
        // Squeezing stage
        $z_1 \leftarrow h[0 \mathinner{..} r-1]$
        for $i \leftarrow 2$ to $\lceil v/r \rceil$ do
                $h \leftarrow \pi(h)$
                $z_i \leftarrow h[0 \mathinner{..} r-1]$
        output $\big(z_1 \parallel \cdots \parallel z_{\lceil v/r \rceil}\big)[0 \mathinner{..} v-1]$
```

The diagram in Fig. 8.11 may help to clarify the algorithm. The sponge runs in two stages: the "absorbing stage" where the message blocks get "mixed in" to a chaining variable h, and a "squeezing stage" where the output is "pulled out" of the chaining variable. Note that input blocks and output blocks are r-bit strings, so that the remaining c bits of the chaining variable cannot be directly tampered with or seen by an attacker. This is what gives the sponge its security, and is the reason why c must be large. Indeed, if the sponge has small capacity, it is easy to find collisions (see Exercise 8.21).

In the SHA3 standard, the sponge construction is intended to be used as a collision resistant hash, and the output length is fixed to a value $v \leq r$, and so the squeezing stage simply outputs the first v bits of the output h of the absorbing stage. We will now prove that this version of the sponge is collision resistant in the ideal permutation model, assuming 2^c and 2^v are both super-poly.

Theorem 8.6. *Let H be the hash function obtained from a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$, with capacity c, rate r (so $n = r + c$), and output length $v \leq r$. In the ideal permutation model, where π is modeled as a random permutation Π, the hash function H is collision resistant, assuming 2^v and 2^c are super-poly.*

In particular, for every collision finding adversary $\mathcal{A}$, if the number of ideal-permutation queries plus the number of r-bit blocks in the output messages of $\mathcal{A}$ is bounded by q, then

$$\mathrm{CR}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, H] \leq \frac{q(q-1)}{2^v} + \frac{q(q+1)}{2^c}.$$

Proof. As in the proof of Theorem 8.4, we assume our collision-finding adversary is "reasonable", in the sense that it makes ideal permutation queries corresponding to its output. We can easily convert an arbitrary adversary into a reasonable one by forcing the adversary to evaluate the hash function on its output messages if it has not done so already. As we have defined it, q will be an upper bound on the total number of ideal permutation queries made by our reasonable adversary. So from now on, we assume a reasonable adversary $\mathcal{A}$ that makes at most q queries, and we bound the probability that such $\mathcal{A}$ finds anything during its queries that can be "assembled" into a collision (we make this more precise below).

We also assume that *no queries are redundant.* This means that if the adversary makes a Π-query on a yielding $b = \Pi(a)$, then the adversary never makes a Π^{-1}-query on b, and never makes another Π-query on a; similarly, if the adversary makes a Π^{-1}-query on b yielding $a = \Pi^{-1}(b)$, then the adversary never makes a Π-query on a, and never makes another Π^{-1}-query on b. Of course, there is no need for the adversary to make such redundant queries, which is why we exclude them; moreover, doing so greatly simplifies the "bookkeeping" in the proof.

It helps to visualize the adversary's attack as building up a directed graph G. The nodes in G consist of the set of all 2^n bit strings of length n. The graph G starts out with no edges, and every query that $\mathcal{A}$ makes adds an edge to the graph: an edge $a \to b$ is added if $\mathcal{A}$ makes a Π-query on a that yields b or a Π^{-1}-query on b that yields a. Notice that if we have an edge $a \to b$, then $\Pi(a) = b$, regardless of whether that edge was added via a Π-query or a Π^{-1}-query. We say that an edge added via a Π-query is a **forward edge**, and one added via a Π^{-1}-query is a **back edge**.

Note that the assumption that the adversary makes no redundant queries means that an edge gets added only once to the graph, and its classification is uniquely determined by the type of query that added the edge.

We next define a special type of path in the graph that corresponds to sponge evaluation. For an n-bit string z, let $R(z)$ be the first r bits of z and $C(z)$ be the last c bits of z. We refer to $R(z)$ as the R**-part of** z and $C(z)$ as the C**-part of** z. For $s \geq 1$, a C**-path of length** s is a sequence of $2s$ nodes

$$a_0, b_1, a_1, b_2, a_2, \ldots, b_{s-1}, a_{s-1}, b_s,$$

where

- $C(a_0) = 0^c$ and for $i = 1, \ldots, s-1$, we have $C(b_i) = C(a_i)$, and
- G contains edges $a_{i-1} \to b_i$ for $i = 1, \ldots, s$.

For such a path p, the **message** of p is defined as $(m_0, \ldots, m_{s-1})$, where

$$m_0 := R(a_0) \quad \text{and} \quad m_i := R(b_i) \oplus R(a_i) \ \text{ for } \ i = 1, \ldots, s-1.$$

and the **result** of p is defined to be $m_s := R(b_s)$. Such a C-path p corresponds to evaluating the sponge at the message $(m_0, \ldots, m_{s-1})$ and obtaining the (untruncated) output m_s. Let us write such a path as

$$m_0|a_0 \longrightarrow b_1|m_1|a_1 \longrightarrow \cdots \longrightarrow b_{s-2}|m_{s-2}|a_{s-2} \longrightarrow b_{s-1}|m_{s-1}|a_{s-1} \longrightarrow b_s|m_s. \tag{8.12}$$

The following diagram illustrates a C-path of length 3.

$$\begin{tikzcd}
a_0 \arrow[r] & b_1 & & \\
m_0 = R(a_0) & a_1 \arrow[r] & b_2 & \\
0^c = C(a_0) & m_1 = R(b_1) \oplus R(a_1) & a_2 \arrow[r] & b_3 \\
& C(b_1) = C(a_1) & m_2 = R(b_2) \oplus R(a_2) & m_3 = R(b_3) \\
& & C(b_2) = C(a_2) &
\end{tikzcd}$$

The path has message (m_0, m_1, m_2) and result m_3. Using the notation in (8.12), we write this path as

$$m_0|a_0 \longrightarrow b_1|m_1|a_1 \longrightarrow b_2|m_2|a_2 \longrightarrow b_3|m_3.$$

We can now state what a collision looks like in terms of the graph G. It is a pair of C-paths on different messages but whose results agree on their first v bits (recall $v \leq r$). Let us call such a pair of paths *colliding.*

To analyze the probability of finding a pair of colliding paths, it will be convenient to define another notion. Let p and p' be two C-paths on different messages whose final edges are $a_{s-1} \to b_s$ and $a'_{t-1} \to b'_t$. Let us call such a pair of paths *problematic* if

(i) $a_{s-1} = a'_{t-1}$, or

(ii) one of the edges in p or p' are back edges.

Let W be the event that $\mathcal{A}$ finds a pair of colliding paths. Let Z be the event that $\mathcal{A}$ finds a pair of problematic paths. Then we have

$$\Pr[W] \leq \Pr[Z] + \Pr[W \text{ and not } Z]. \tag{8.13}$$

First, we bound $\Pr[W \text{ and not } Z]$. For an n-bit string z, let $V(z)$ be the first v bits of z, and we refer to $V(z)$ as the **V-part of z**. Suppose $\mathcal{A}$ is able to find a pair of colliding paths that is not problematic. By definition, the final edges on these two paths correspond to Π-queries on distinct inputs that yield outputs whose V-parts agree. That is, if W and not Z occurs, then it must be the case that at some point $\mathcal{A}$ issued two Π-queries on distinct inputs a and a', yielding outputs b and b' such that $V(b) = V(b')$. We can use the union bound: for each pair of indices $i < j$, let X_{ij} be the event that the ith query is a Π-query on some value, say a, yielding $b = \Pi(a)$, and the j-th query is also a Π-query on some other value $a' \neq a$, yielding $b' = \Pi(a')$ such that $V(b) = V(b')$. If we fix i and j, fix the coins of $\mathcal{A}$, and fix the outputs of all queries made prior to the jth query, then the values a, b, and a' are all fixed, but the value b' is uniformly distributed over a set of size at least $2^n - j + 1$. To get $V(b) = V(b')$, the value of b' must be equal to one of the 2^{n-v} strings whose first v bits agree with that of b, and so we have

$$\Pr[X_{ij}] \leq \frac{2^{n-v}}{2^n - j + 1}.$$

A simple calculation like that done in (8.5) in the proof of Theorem 8.4 yields

$$\Pr[W \text{ and not } Z] \leq \frac{q(q-1)}{2^v}. \tag{8.14}$$

Second, we bound $\Pr[Z]$, the probability that $\mathcal{A}$ finds a pair of problematic paths. The technical heart of the analysis is the following:

Main Claim: *If Z occurs, then one of the following occurs:*

(E1) some query yields an output whose C-part is 0^c, or

(E2) two different queries yield outputs whose C-parts are equal.

Just to be clear, (E1) means $\mathcal{A}$ made a query of the form:

> (i) a Π^{-1}-query on some value $\mathscr{b}$ such that $C(\Pi^{-1}(\mathscr{b})) = 0^c$, or (ii) a Π-query on some value $\mathscr{a}$ such that $C(\Pi(\mathscr{a})) = 0^c$,

and (E2) means $\mathcal{A}$ made pair of queries of the form:

> (i) a Π-query on some value $\mathscr{a}$ and a Π^{-1}-query on some value $\mathscr{b}$, such that $C(\Pi(\mathscr{a})) = C(\Pi^{-1}(\mathscr{b}))$, or (ii) Π-queries on two distinct values $\mathscr{a}$ and $\mathscr{a}'$ such that $C(\Pi(\mathscr{a})) = C(\Pi(\mathscr{a}'))$.

First, suppose $\mathcal{A}$ is able to find a problematic pair of paths, and one of the paths contain a back edge. So at the end of the execution, there exists a C-path containing one or more back edges. Let p be such a path of shortest length, and write it as in (8.12). We observe that the last edge in p is a back edge, and all other edges (if any) in p are forward edges. Indeed, if this is not the case, then we can delete this edge from p, obtaining a shorter C-path containing a back edge, contradicting the assumption that p is a shortest path of this type. From this observation, we see that either:

- $s = 1$ and (E1) occurs with the Π^{-1}-query on $\mathscr{b}_1$, or
- $s > 1$ and (E2) occurs with the Π^{-1}-query on $\mathscr{b}_s$ and the Π-query on $\mathscr{a}_{s-2}$.

Second, suppose $\mathcal{A}$ is able to find a problematic pair of paths, neither of which contains any back edges. Let us call these paths p and p'. The argument in this case somewhat resembles the "backwards walk" in the Merkle-Damgård analysis. Write p as in (8.12) and write p' as

$$m'_0|\mathscr{a}'_0 \longrightarrow \mathscr{b}'_1|m'_1|\mathscr{a}'_1 \longrightarrow \cdots \longrightarrow \mathscr{b}'_{t-2}|m'_{t-2}|\mathscr{a}'_{t-2} \longrightarrow \mathscr{b}'_{t-1}|m'_{t-1}|\mathscr{a}'_{t-1} \longrightarrow \mathscr{b}'_t|m'_t.$$

We are assuming that $(m_0, \ldots, m_{s-1}) \neq (m'_0, \ldots, m'_{t-1})$ but $\mathscr{a}_{s-1} = \mathscr{a}'_{t-1}$, and that none of these edges are back edges. Let us also assume that we choose the paths so that they are shortest, in the sense that $s+t$ is minimal among all C-paths of this type. Also, let us assume that $s \leq t$ (swapping if necessary). There are a few cases:

1. $s = 1$ and $t = 1$. This case is impossible, since in this case the paths are just $m_0|\mathscr{a}_0 \to \mathscr{b}_1|m_1$ and $m'_0|\mathscr{a}'_0 \to \mathscr{b}'_1|m'_1$, and we cannot have both $m_0 \neq m'_0$ and $\mathscr{a}_0 = \mathscr{a}'_0$.
2. $s = 1$ and $t \geq 2$. In this case, we have $\mathscr{a}_0 = \mathscr{b}'_{t-1}$, and so (E1) occurs on the Π-query on $\mathscr{a}'_{t-2}$.
3. $s \geq 2$ and $t \geq 2$. Consider the penultimate edges, which are forward edges:

   $$\mathscr{a}_{s-2} \to \mathscr{b}_{s-1}|m_{s-1}|\mathscr{a}_{s-1}$$

   and

   $$\mathscr{a}'_{t-2} \to \mathscr{b}'_{t-1}|m'_{t-1}|\mathscr{a}'_{t-1}.$$

   We are assuming $\mathscr{a}_{s-1} = \mathscr{a}'_{t-1}$. Therefore, the C-parts of $\mathscr{b}_{s-1}$ and $\mathscr{b}'_{t-1}$ are equal and their R-parts differ by $m_{s-1} \oplus m'_{t-1}$. There are two subcases:

(a) $m_{s-1} = m'_{t-1}$. We argue that this case is impossible. Indeed, in this case, we have $\mathfrak{b}_{s-1} = \mathfrak{b}'_{t-1}$, and therefore $\mathfrak{a}_{s-2} = \mathfrak{a}'_{t-2}$, while the truncated messages $(m_0, \ldots, m_{s-2})$ and $(m'_0, \ldots, m'_{t-2})$ differ. Thus, we can simply throw away the last edge in each of the two paths, obtaining a shorter pair of paths that contradicts the minimality of $s + t$.

(b) $m_{s-1} \neq m'_{t-1}$. In this case, we know: the C-parts of $\mathfrak{b}_{s-1}$ and $\mathfrak{b}'_{t-1}$ are the same, but their R-parts differ, and therefore, $\mathfrak{a}_{s-2} \neq \mathfrak{a}'_{t-2}$. Thus, (E2) occurs on the Π-queries on $\mathfrak{a}_{s-2}$ and $\mathfrak{a}'_{t-2}$.

That proves the Main Claim. We can now turn to the problem of bounding the probability that either (E1) or (E2) occurs. This is really just the same type of calculation we did at least twice already, once above in obtaining (8.14), and earlier in the proof of Theorem 8.4. The only difference from (8.14) is that we are now counting collisions on the C-parts, and we have a new type of "collision" to count, namely, "hitting 0^c" as in (E1). We leave it to the reader to verify:

$$\Pr[Z] \leq \frac{q(q+1)}{2^c}. \tag{8.15}$$

The theorem now follows from (8.13)–(8.15). □

8.8.2 Case study: SHA3, SHAKE128, and SHAKE256

The NIST standard for SHA3 specifies a family of sponge-based hash functions. At the heart of these hash functions is a permutation called Keccak, which maps 1600-bit strings to 1600-bit strings. We denote by Keccak$[c]$ the sponge derived from Keccak with capacity c, and using the 10*1 padding rule. This is a function that takes two inputs: a message m and output length v. Here, the input m is an arbitrary bit string and the output of Keccak$[c](m, v)$ is a v-bit string.

We will not describe the internal workings of the Keccak permutation; they can be found in the SHA3 standard. We just describe the different parameter choices that are standardized. The standard specifies four hash functions whose output lengths are fixed, and two hash functions with variable length outputs.

Here are the four fixed-length output hash functions:

- SHA3-224(m) = Keccak$[448](m \parallel 01, 224)$;
- SHA3-256(m) = Keccak$[512](m \parallel 01, 256)$;
- SHA3-384(m) = Keccak$[768](m \parallel 01, 384)$;
- SHA3-512(m) = Keccak$[1024](m \parallel 01, 512)$.

Note the two extra padding bits that are appended to the message. Note that in each case, the capacity c is equal to twice the output length v. Thus, as the output length grows, the security provided by the capacity grows as well, and the rate — and, therefore, the hashing speed — decreases.

Here are the two variable-length output hash functions:

- SHAKE128(m, v) = Keccak$[256](m \parallel 1111, v)$;
- SHAKE256(m, v) = Keccak$[512](m \parallel 1111, v)$.

Note the four extra padding bits that are appended to the message. The only difference between these two is the capacity size, which affects the speed and security. The various padding bits and the 10*1 padding rule ensure that these six functions behave independently.

8.9 Merkle trees: proving properties of a hashed list

Now that we understand how to construct collision resistant functions, let's see more of their applications to data integrity. Consider a large executable file, stored on disk as a list of short ℓ-bit blocks $x_1, \ldots, x_n$. Before the operating system loads and runs this executable, it needs to verify that its contents have not been altered. At the beginning of the chapter we discussed how one can store a short hash of the entire file in read-only storage[3]. Every time the file is run, the system first recomputes the file hash, and verifies that it matches the value in storage. We explained that a collision resistant hash ensures that the adversary cannot tamper with the file without being detected. The problem is that for a large file, computing the hash of the entire file can take quite a while, and this will greatly increase the time to launch the executable.

Can we do better? To start running the executable, the system only needs to verify the first block x_1. When execution moves to some other block, the system only needs to verify that block, and so on. In other words, instead of verifying the entire file all at once, it would be much better if the system could verify each block independently, just before that block is loaded. One option is to compute the hash of every block $x_1, \ldots, x_n$, and store the resulting n hashes in read-only storage. This makes it easy to verify every block by itself, but also takes up a lot of read-only space to store the n hashes. Fortunately, there is a much better solution.

Merkle trees. To restate the problem, we have a vector of n items $T := (x_1, \ldots, x_n) \in \mathcal{X}^n$, and we wish to compute a short hash y of this vector. Later, we are presented with the hash y and a pair (i, x) where $1 \leq i \leq n$. We need to validate that x is the item at position i in T.

A solution to this problem makes use of a clever data structure called a **Merkle tree**, shown in Fig. 8.12. The resulting hash function H is called a **Merkle tree hash**.

The Merkle tree hash uses a collision resistant hash function h, such as SHA256, that outputs values in a set $\mathcal{Y}$. The input to h is either a single element in $\mathcal{X}$, or a pair of elements in $\mathcal{Y}$. The Merkle tree hash H, derived from h, is defined over $(\mathcal{X}^n, \mathcal{Y})$. For simplicity, let's assume that n is a power of two (if not, one can pad with dummy elements to the closest power of two). The Merkle tree hash works as in Fig. 8.12: to hash $(x_1, \ldots, x_n) \in \mathcal{X}^n$, first apply h to each of the n input elements to get $(y_1, \ldots, y_n) \in \mathcal{Y}^n$. Then build a hash tree from these elements, as shown in the figure. More precisely, the hash function H is defined as follows:

```
input:   $x_1, \ldots, x_n \in \mathcal{X}$, where $n$ is a power of 2
output:  $y \in \mathcal{Y}$

for $i = 1$ to $n$:       $y_i \leftarrow h(x_i)$                  // initialize $y_1, \ldots, y_n$
for $i = 1$ to $n - 1$:   $y_{i+n} \leftarrow h(y_{2i-1},\ y_{2i})$     // compute tree nodes $y_{n+1}, \ldots, y_{2n-1}$
output $y_{2n-1} \in \mathcal{Y}$
```

In Exercise 8.9 we show that a closely related hash function, designed for variable length inputs, is collision resistant, assuming h is collision resistant.

Proving membership. The remarkable thing about the Merkle tree hash is that given a hash value $y := H(x_1, \ldots, x_n)$, it is quite easy to prove that an $x \in \mathcal{X}$ is the element at a particular

[3] Recall that read-only storage can be read, but not modified, by an adversary. It can be implemented as a separate system that provides the data to anyone who asks for it. Or, more simply, it can be implemented by signing the data using a digital signature scheme, as discussed in Chapter 13, and storing the signing key offline.

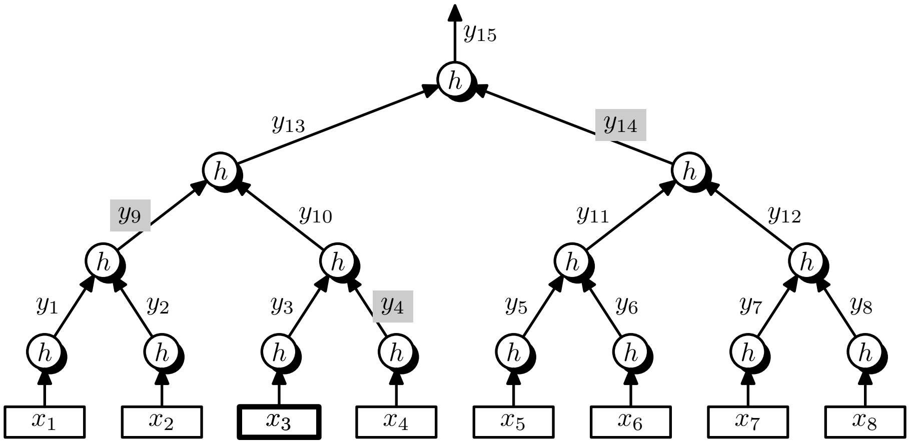


Figure 8.12: A Merkle tree with eight leaves. The values y_4, y_9, y_{14} prove authenticity of x_3.

position in $T := (x_1, \ldots, x_n)$. For example, to prove that $x = x_3$ in Fig. 8.12, one provides the intermediate hashes $\pi := (\mathbf{y_4},\ \mathbf{y_9},\ \mathbf{y_{14}})$, shaded in the figure. The verifier can then compute

$$\hat{y}_3 \leftarrow h(x), \quad \hat{y}_{10} \leftarrow h(\hat{y}_3, \mathbf{y_4}), \quad \hat{y}_{13} \leftarrow h(\mathbf{y_9}, \hat{y}_{10}), \quad \hat{y}_{15} \leftarrow h(\hat{y}_{13}, \mathbf{y_{14}}), \tag{8.16}$$

and accept that $x = x_3$ if $y = \hat{y}_{15}$. This π is called a **Merkle proof** that x is in position 3 of T.

More generally, to prove that an element x is the element in position i of $T := (x_1, \ldots, x_n)$, one outputs as the proof π all the intermediate hashes that are the siblings of nodes on the path from the leaf number i to the root of the tree. This proof π contains exactly $\log_2 n$ elements in $\mathcal{Y}$. The verifier can use the quantities provided in π to re-derive the Merkle hash of T. It does so by computing hashes, starting at leaf number i, and working its way up to the root, as in (8.16). It accepts that x as authentic (i.e., that $x = x_i$) if the final computed Merkle hash matches the hash value y stored in read-only memory.

We will show in Theorem 8.8 below that, if h is collision resistant, an adversary cannot exhibit an x and an i, along with a proof π', that incorrectly convinces the verifier that x is in position i of T.

Consider again our executable stored on disk as a sequence of blocks $x_1, \ldots, x_n$, and suppose that the system has $y := H(x_1, \ldots, x_n)$ in read-only storage. We can store the $2n - 1$ hash values in the Merkle tree, denoted $y_1, \ldots, y_{2n-1}$, along with the executable. Then, to validate a block, the system will quickly locate the $\log_2 n$ hash values that make up the Merkle proof for that block, compute the Merkle hash by computing $\log_2 n$ hashes, and compare the result to the stored value y. In practice, suppose blocks are 4KB each. Then even for an executable of 2^{16} blocks, we are adding at most two hash values per block ($2n - 1$ hash values in total), which is only 64 bytes per block. Validating a block is done by computing 16 hashes.

There are other solutions to this problem. For example, the system could store a MAC tag next to every block, and verify the tag before executing the block. However, this would require the system to manage the secret MAC key, and ensure that it is never read by the adversary. While this may be reasonable in some settings, the Merkle tree approach provides an efficient solution that requires no online secret keys.

Proving membership of multiple elements. Suppose again that $y := H(x_1, \ldots, x_n)$ is stored in read-only storage, and let $T := (x_1, \ldots, x_n)$. Let $L \subseteq \mathcal{X}$ be a set of elements. We wish to convince the verifier that all the elements in L are in T. We could provide a Merkle proof for every element in L, giving a total proof size of $|L| \log_2 n$ elements in $\mathcal{Y}$. However, many of these Merkle proofs overlap, and we can shrink the overall proof by removing repeated elements. The following theorem bounds the worst-case proof size. We write $L \subseteq T$ to denote the fact that all the elements in L are contained in T.

Theorem 8.7. *Let $T \subseteq \mathcal{X}$ be a set of size n, where n is a power of two. For every $1 \leq r \leq n$, and a set $L \subseteq T$ of size r, the Merkle proof that all the elements of L are in T contains at most $r \cdot \log_2(n/r)$ elements in $\mathcal{Y}$.*

Proof. The theorem is a direct corollary of Theorem 5.8. Let $S := T \setminus L$, so that $|S| = n - r$. It is not difficult to see that the set of hash values in the Merkle proof for L are precisely those that correspond to nodes in cover(S). The bound on $|\text{cover}(S)|$ provided in Theorem 5.8 proves the theorem. □

Proving non-membership. Let's look at another application for Merkle trees. Consider a credit card processing center that maintains a list T of revoked credit card numbers $T := (x_1, \ldots, x_n) \in \mathcal{X}^n$. The list T is sent to untrusted cache servers all over the world, and every merchant is sent the short Merkle tree hash $y := H(x_1, \ldots, x_n)$. This hash y is assumed to be computed correctly by the center. When a merchant needs to process a customer's credit card x, it sends x to the closest cache server to test if x is revoked (i.e., test if x is in T). If so, the cache server responds with a Merkle proof that x is in T, and this convinces the merchant to reject the transaction. Security of the Merkle tree scheme implies that a malicious cache server cannot fool the merchant into believing that an active credit card is revoked. More generally, Merkle trees let us replicate a data set T across untrusted cache servers, so that no cache server can lie about membership in the set.

For the credit card application, proving membership in T is not enough. The cache server must also be able to convince the merchant that a credit card x is not in T (i.e., not revoked). Surprisingly, a Merkle tree can also be used to prove set non-membership, but to do so we must first slightly modify the Merkle tree construction.

Suppose that the elements in T are integers, so that $\mathcal{X} \subseteq \mathbb{Z}$. In the modified tree hash we first sort the leaves of the tree, so that $x_1 < x_2 < \cdots < x_n$, as shown in Fig. 8.13. We then compute the tree hash $y := H(x_1, \ldots, x_n)$ as before. We call this the **sorted Merkle tree hash**.

Now, given some $x \notin T$, we wish to produce a proof that x is not in T. The verifier only has the sorted tree hash y. To produce the proof, the prover first locates the two adjacent leaves x_i and x_{i+1} in T that bracket x, namely $x_i < x < x_{i+1}$. For simplicity, let's assume that $x_1 < x < x_n$, so that the required x_i and x_{i+1} always exist. Next, the prover provides a Merkle proof that x_i is in position i in T, and that x_{i+1} is in position $i + 1$ in T. The verifier can check that these two leaves are adjacent, and that $x_i < x < x_{i+1}$, and this proves that x is not in T. Indeed, if x were in T, it must occupy a leaf between x_i and x_{i+1}, but because x_i and x_{i+1} are adjacent leaves, this is not possible.

Fig. 8.13 gives an example proof that a value x in the interval (x_4, x_5) is not in T. The proof is the set of hashes $(y_3,\ y_6,\ y_9,\ y_{12})$ along with the data items x_4, x_5. The verifies checks the Merkle proofs to convince itself that x_4 and x_5 are in T, and that they are adjacent in the tree. It then checks that $x_4 < x < x_5$, and this proves that x is not in the tree. We see that in the worst case, a

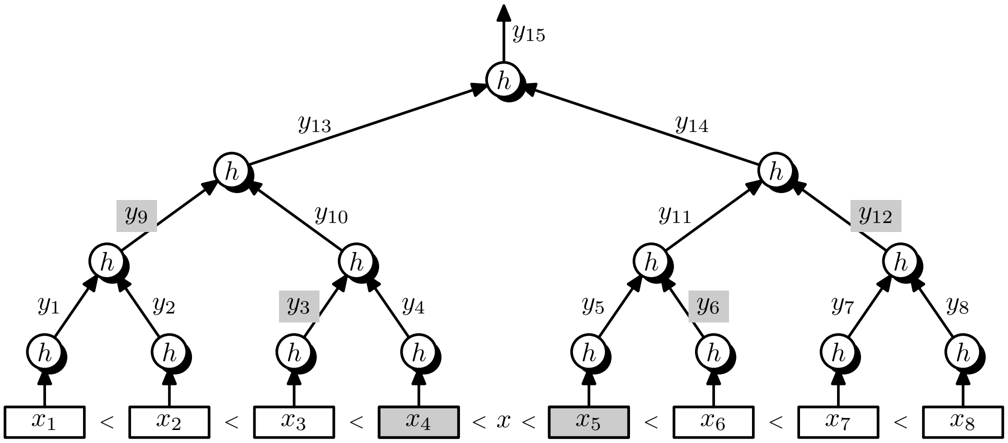


Figure 8.13: The sorted tree hash. The shaded elements prove non-membership of x.

proof of non-membership contains $2\log_2(n/2)$ elements in $\mathcal{Y}$, plus two data items in $\mathcal{X}$.

Security of the scheme is discussed in the next section. It shows that, when the underlying hash function h is collision resistant, an adversary cannot convince the verifier that an $x \in T$ is not a member of T. In our example application, a malicious cache server cannot convince a merchant that a revoked credit card is active.

8.9.1 Authenticated data structures

A Merkle tree is an example of a more abstract concept called an authenticated data structure. An authenticated data structure is used to compute a short hash of a sequence $T := (x_1, \ldots, x_n)$, so that later one can prove properties of T with respect to this hash. Merkle trees let us prove membership and non-membership. Other authenticated data structures support additional operations, such as efficient insertions and deletions, as discussed below.

We begin by defining an authenticated data structure for set membership, and its security property.

Definition 8.3. *An* ***authenticated data structure scheme*** $\mathcal{D} = (H, P, V)$ *defined over* $(\mathcal{X}^n, \mathcal{Y})$ *is a tuple of three efficient deterministic algorithms:*

- *H is an algorithm that is invoked as $y \leftarrow H(T)$, where $T := (x_1, \ldots, x_n) \in \mathcal{X}^n$ and $y \in \mathcal{Y}$.*
- *P is an algorithm that is invoked as $\pi \leftarrow P(i, x, T)$, where $x \in \mathcal{X}$ and $1 \leq i \leq n$. The algorithm outputs a proof π that $x = x_i$, where $T := (x_1, \ldots, x_n)$.*
- *V is an algorithm that is invoked as $V(i, x, y, \pi)$ and outputs* accept *or* reject*.*
- *We require that for all $T := (x_1, \ldots, x_n) \in \mathcal{X}^n$, and all $1 \leq i \leq n$, we have that*

$$V\big(i,\ x_i,\ H(T),\ P(i, x_i, T)\big) = \textsf{accept}$$

The Merkle tree scheme from the previous section can be easily cast as these three algorithms (H, P, V).

We next define security. We say that an adversary defeats the scheme if it can output a hash value $y \in \mathcal{Y}$ and then fool the verifier into accepting two different elements x and x' in $\mathcal{X}$ at some position i.

Attack Game 8.2 (authenticated data structure security). For an authenticated data structure scheme $\mathcal{D} = (H, P, V)$ defined over $(\mathcal{X}^n, \mathcal{Y})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

> The adversary $\mathcal{A}$ outputs a $y \in \mathcal{Y}$, a position $i \in \{1, \ldots, n\}$, and two pairs (x, π) and (x', π') where $x, x' \in \mathcal{X}$.

We say that $\mathcal{A}$ wins the game if $x \neq x'$ and $V\big(i,\ x,\ y,\ \pi\big) = V\big(i,\ x',\ y,\ \pi'\big) = \mathsf{accept}$. Define $\mathcal{A}$'s advantage with respect to $\mathcal{D}$, denoted $\mathrm{ADS}\mathsf{adv}[\mathcal{A}, \mathcal{D}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 8.4. *We say that an authenticated data structure scheme $\mathcal{D}$ is secure if for all efficient adversaries $\mathcal{A}$, the value $\mathrm{ADS}\mathsf{adv}[\mathcal{A}, \mathcal{D}]$ is negligible.*

Theorem 8.8. *The Merkle hash tree scheme is a secure authenticated data structure scheme, assuming the underlying hash function h is collision resistant.*

Proof. The proof is essentially the same as the proof of Exercise 8.9. □

One can similarly formulate a security definition for proving non-membership in a hashed data set. We leave it as an instructive exercise to state the security definition, and prove that the sorted Merkle tree is a secure scheme for proving non-membership, assuming that the underlying hash function is collision resistant.

Updatable Merkle data structures. Let T be a data set of size n. One downside of sorted Merkle hash trees is that if even a single element in the data set T is changed, that element may need to move to a different leaf, and the entire hash tree will need to be recomputed from scratch. This can take $O(n)$ hash computations. Other data structures provide the same functionality as Merkle trees, but also support an efficient update, requiring at most $O(\log n)$ hash calculations to update an element. One example is a scheme based on a 2-3 tree [126], and another is a scheme based on skip lists [76]. A common authenticated data structure, used in certain crypto currency systems, is a hash tree based on the *Patricia tree* data structure.

8.10 Key derivation and the random oracle model

Although hash functions like SHA256 were initially designed to provide collision resistance, we have already seen in Section 8.7 that practitioners are often tempted to use them to solve other problems. Intuitively, hash functions like SHA256 are designed to "thoroughly scramble" their inputs, and so this approach seems to make some sense. Indeed, in Section 8.7, we looked at the problem of taking an unkeyed hash function and turning it into a keyed function that is a secure PRF, and found that it was indeed possible to give a security analysis under reasonable assumptions.

In this section, we study another problem, called **key derivation**. Roughly speaking, the problem is this: we start with some secret data, and we want to convert it into an n-bit string that we can use as the key to some cryptographic primitive, like AES. Now, the secret data may be random in some sense — at the very least, somewhat hard to guess — but it may not look anything at all like a uniformly distributed, random, n-bit string. So how do we get from such a secret s to a cryptographic key t? Hashing, of course. In practice, one takes a hash function H, such as SHA256 (or, as we will ultimately recommend, some function built out of SHA256), and computes $t \leftarrow H(s)$.

Along the way, we will also introduce the *random oracle model*, which is a heuristic tool that is useful not only for analyzing the key derivation problem, but a host of other problems as well.

8.10.1 The key derivation problem

Let us look at the key derivation problem in more detail. Again, at a high level, the problem is to convert some discreet data that is hard to guess into an n-bit string we can use directly as a key to some standard cryptographic primitive, such as AES. The solution in all cases will be to hash the secret to obtain the key. We begin with some motivating examples.

- The secret might be a password. While such a password might be somewhat hard to guess, it could be dangerous to use such a password directly as an AES key. Even if the password were uniformly distributed over a large dictionary (already a suspect assumption), the distribution of its encoding as a bit string is certainly not. It could very well be that a significant fraction of passwords correspond to "weak keys" for AES that make it vulnerable to attack. Recall that AES was designed to be used with a random bit string as the key, so how it behaves on passwords is another matter entirely.

- The secret could be the log of various types of system events on a running computer (e.g., the time of various interrupts such as those caused by key presses or mouse movements). Again, it might be difficult for an attacker who is outside the computer system to accurately predict the contents of such a log. However, using the log directly as an AES key is problematic: it is likely far too long, and far from uniformly distributed.

- The secret could be a cryptographic key which has been partially compromised. Imagine that a user has a 128-bit key, but that 64 of the bits have been leaked to the adversary. The key is still fairly difficult to guess, but it is still not uniformly distributed from the adversary's point of view, and so should not be used directly as an AES key.

- Later, we will see examples of number-theoretic transformations that are widely used in public-key cryptography. Looking ahead a bit, we will see that for a large, composite modulus N, if x is chosen at random modulo N, and an adversary is given $y := x^3 \bmod N$, it is hard to compute x. We can view x as the secret, and similarly to the previous example, we can view y as information that is leaked to the adversary. Even though the value of y completely determines x in an information-theoretic sense, it is still widely believed to be hard to compute. Therefore, we might want to treat x as secret data in exactly the same way as in the previous examples. Many of the same issues arise here, not the least of which is that x is typically much longer (typically, thousands of bits long) than an AES key.

As already mentioned, the solution that is adopted in practice is simply to hash the secret s using a hash function H to obtain the key $t \leftarrow H(s)$.

Let us now give a formal definition of the security property we are after.

We assume the secret s is sampled according to some fixed (and publicly known) probability distribution P. We assume any such secret data can be encoded as an element of some finite set $\mathcal{S}$.

Further, we model the fact that some partial information about s could be leaked by introducing a function I, so that an adversary trying to guess s knows the side information $I(s)$.

Attack Game 8.3 (Guessing advantage). Let P be a probability distribution defined on a finite set $\mathcal{S}$ and let I be a function defined in $\mathcal{S}$. For a given adversary $\mathcal{A}$, the attack game runs as follows:

- the challenger chooses s at random according to P and sends $I(s)$ to $\mathcal{A}$;
- the adversary outputs a guess $\hat{s}$ for s, and wins the game if $\hat{s} = s$.

The probability that $\mathcal{A}$ wins this game is called its **guessing advantage**, and is denoted Guess$\mathsf{adv}[\mathcal{A}, P, I]$. □

In the first example above, we might simplistically model s as being a password that is uniformly distributed over (the encodings of) some dictionary D of words. In this case, there is no side information given to the adversary, and the guessing advantage is $1/|D|$, regardless of the computational power of the adversary.

In the second example above, it seems very hard to give a meaningful and reliable estimate of the guessing advantage.

In the third example above, s is uniformly distributed over $\{0,1\}^{128}$, and $I(s)$ is (say) the first 64-bits of s. Clearly, any adversary, no matter how powerful, has guessing advantage no greater than 2^{-64}.

In the fourth example above, s is the number x and $I(s)$ is the number y. Since y completely determines x, it is possible to recover s from $I(s)$ by brute-force search. There are smarter and faster algorithms as well, but there is no known efficient algorithm to do this. So for all *efficient* adversaries, the guessing advantage appears to be *negligible*.

Now suppose we use a hash function $H : \mathcal{S} \to \mathcal{T}$ to derive the key t from s. Intuitively, we want t to "look random". To formalize this intuitive notion, we use the concept of computational indistinguishability from Section 3.11. So formally, the property that we want is that if s is sampled according to P and t is chosen at random from $\mathcal{T}$, the two distributions $(I(s), H(s))$ and $(I(s), t)$ are computationally indistinguishable. For an adversary $\mathcal{A}$, let Dist$\mathsf{adv}[\mathcal{A}, P, I, H]$ be the adversary's advantage in Attack Game 3.3 for these two distributions.

The type of theorem we would like to be able to prove would say, roughly speaking, if H satisfies some specific property, and perhaps some constraints are placed on P and I, then for every adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) such that Dist$\mathsf{adv}[\mathcal{A}, P, I, H]$ is not too much larger than Guess$\mathsf{adv}[\mathcal{B}, P, I]$. In fact, in certain situations it *is* possible to prove such a theorem. We will discuss this result later, in Section 8.10.4 — for now, we will simply say that this rigorous approach is not widely used in practice, for a number of reasons. Instead, we will examine in greater detail the heuristic approach of using an "off the shelf" hash function like SHA256 to derive keys.

Sub-key derivation. Before moving on, we consider the following, related problem: what to do with the key t derived from s. In some applications, we might use t directly as, say, an AES key. In other applications, however, we might need several keys: for example, an encryption key and a MAC key, or two different encryption keys for bi-directional secure communications (so Alice has one key for sending encrypted messages to Bob, and Bob uses a different key for sending encrypted messages to Alice). So once we have derived a single key t that "for all intents and purposes" behaves like a random bit string, we wish to derive several sub-keys. We call this the **sub-key derivation problem** to distinguish it from the key derivation problem. For the sub-key derivation problem, we assume that we start with a truly random key t — it is not, but when t is computationally indistinguishable from a truly random key, this assumption is justified.

Fortunately, for sub-key derivation, we already have all the tools we need at our disposal. Indeed, we can derive sub-keys from t using either a PRG or a PRF. For example, in the above example, if Alice and Bob have a shared key t, derived from a secret s, they can use a PRF F as follows:

- derive a MAC key $k_{\text{mac}} \xleftarrow{\text{R}} F(t, \texttt{"MAC-KEY"})$;
- derive an Alice-to-Bob encryption key $k_{\text{AB}} \xleftarrow{\text{R}} F(t, \texttt{"AB-KEY"})$;
- derive a Bob-to-Alice encryption key $k_{\text{BA}} \xleftarrow{\text{R}} F(t, \texttt{"BA-KEY"})$.

Assuming F is a secure PRF, then the keys k_{mac}, k_{AB}, and k_{BA} behave, for all intents and purposes, as independent random keys. To implement F, we can even use a hash-based PRF, like HMAC, so we can do everything we need — key derivation and sub-key derivation — using a single "off the shelf" hash function like SHA256.

So once we have solved the key derivation problem, we can use well-established tools to solve the sub-key derivation problem. Unfortunately, the practice of using "off the shelf" hash functions for key derivation is not very well understood or analyzed. Nevertheless, there are some useful heuristic models to explore.

8.10.2 Random oracles: a useful heuristic

We now introduce a heuristic that we can use to model the use of hash functions in a variety of applications, including key derivation. As we will see later in the text, this has become a popular heuristic that is used to justify numerous cryptographic constructions.

The idea is that we simply model a hash function H *as if* it were a truly random function $\mathcal{O}$. If H maps $\mathcal{M}$ to $\mathcal{T}$, then $\mathcal{O}$ is chosen uniformly at random from the set $\text{Funs}[\mathcal{M}, \mathcal{T}]$. We can translate any attack game into its random oracle version: the challenger uses $\mathcal{O}$ in place of H for all its computations, and in addition, the adversary is allowed to obtain the value of $\mathcal{O}$ at arbitrary input points of his choosing. The function $\mathcal{O}$ is called a **random oracle** and security in this setting is said to hold in the **random oracle model**. The function $\mathcal{O}$ is too large to write down and cannot be used in a real construction. Instead, we only use $\mathcal{O}$ as a means for carrying out a heuristic security analysis of the proposed system that actually uses H.

This approach to analyzing constructions using hash function is analogous to the *ideal cipher model* introduced in Section 4.7, where we replace a block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{X})$ by a family of random permutations $\{\Pi_{k}\}_{k \in \mathcal{K}}$.

As we said, the random oracle model is used quite a bit in modern cryptography, and it would be nice to be able to use an "off the shelf" hash function H, and model it as a random oracle. However, if we want a truly general purpose tool, we have to be a bit careful, especially if we want to model H as a random oracle taking *variable length inputs.* The basic rule of thumb is that Merkle-Damgård hashes should not be used *directly* as general purpose random oracles. We will discuss in Section 8.10.3 how to safely (but again, heuristically) use Merkle-Damgård hashes as general purpose random oracles, and we will also see that the sponge construction (see Section 8.8) can be used directly "as is".

We stress that even though security results in the random oracle are rigorous, mathematical theorems, they are still only heuristic results that do not guarantee any security for systems built with any specific hash function. They do, however, rule out "generic attacks" on systems that *would* work if the hash function *were* a random oracle. So, while such results do not rule out all attacks, they do rule out generic attacks, which is better than saying nothing at all about the security of the system. Indeed, in the real world, given a choice between two systems, $\mathcal{S}_1$ and $\mathcal{S}_2$, where $\mathcal{S}_1$ comes with a security proof in the random oracle model, and $\mathcal{S}_2$ comes with a real security proof but is twice as slow as $\mathcal{S}_1$, most practitioners would (quite reasonably) choose $\mathcal{S}_1$ over $\mathcal{S}_2$.

Defining security in the random oracle model. Suppose we have some type of cryptographic scheme $\mathcal{S}$ whose implementation makes use of a subroutine for computing a hash function H defined over $(\mathcal{M}, \mathcal{T})$. The scheme $\mathcal{S}$ evaluates H at arbitrary points of its choice, but does not look at the internal implementation of H. We say that $\mathcal{S}$ **uses** H **as an oracle**. For example, $F_{\text{pre}}(k, x) := H(k \parallel x)$, which we briefly considered in Section 8.7, is a PRF that uses the hash function H as an oracle.

We wish to analyze the security of $\mathcal{S}$. Let us assume that whatever security property we are interested in, say "property X," is modeled (as usual) as a game between a challenger (specific to property X) and an arbitrary adversary $\mathcal{A}$. Presumably, in responding to certain queries, the challenger computes various functions associated with the scheme $\mathcal{S}$, and these functions may in turn require the evaluation of H at certain points. This game defines an advantage $\text{X}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, and security with respect to property X means that this advantage should be negligible for all efficient adversaries $\mathcal{A}$.

If we wish to analyze $\mathcal{S}$ in the random oracle model, then the attack game defining security is modified so that H is effectively replaced by a *random function* $\mathcal{O} \in \text{Funs}[\mathcal{M}, \mathcal{T}]$, to which both the adversary and the challenger have oracle access. More precisely, the game is modified as follows.

- At the beginning of the game, the challenger chooses $\mathcal{O} \in \text{Funs}[\mathcal{M}, \mathcal{T}]$ at random.
- In addition to its standard queries, the adversary $\mathcal{A}$ may submit *random oracle queries*: it gives $m \in \mathcal{M}$ to the challenger, who responds with $t = \mathcal{O}(m)$. The adversary may make any number of random oracle queries, arbitrarily interleaved with standard queries.
- In processing standard queries, the challenger performs its computations using $\mathcal{O}$ in place of H.

The adversary's advantage is defined using the same rule as before, but is denoted $\text{X}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ to emphasize that this is an advantage *in the random oracle model*. Security *in the random oracle model* means that $\text{X}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ should be negligible for all efficient adversaries $\mathcal{A}$.

A simple example: PRFs in the random oracle model. We illustrate how to apply the random oracle framework to construct secure PRFs. In particular, we will show that F_{pre} is a secure PRF in the random oracle model. We first adapt the standard PRF security game to obtain a PRF security game in the random oracle model. To make things a bit clearer, if we have a PRF F that uses a hash function H as an oracle, we denote by $F^{\mathcal{O}}$ the function that uses the random oracle $\mathcal{O}$ in place of H.

Attack Game 8.4 (PRF in the random oracle model). Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ that uses a hash function H defined over $(\mathcal{M}, \mathcal{T})$ as an oracle. For a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define:

Experiment b:

- $\mathcal{O} \overset{\text{R}}{\leftarrow} \text{Funs}[\mathcal{M}, \mathcal{T}]$.
- The challenger selects $f \in \text{Funs}[\mathcal{X}, \mathcal{Y}]$ as follows:

  if $b = 0$: $k \overset{\text{R}}{\leftarrow} \mathcal{K}$, $f \leftarrow F^{\mathcal{O}}(k, \cdot)$;
  if $b = 1$: $f \overset{\text{R}}{\leftarrow} \text{Funs}[\mathcal{X}, \mathcal{Y}]$.

- The adversary submits a sequence of queries to the challenger.
  - F-query: respond to a query $x \in \mathcal{X}$ with $y = f(x) \in \mathcal{Y}$.
  - $\mathcal{O}$-query: respond to a query $m \in \mathcal{M}$ with $t = \mathcal{O}(m) \in \mathcal{T}$.
- The adversary computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to F as

$$\text{PRF}^{\text{ro}}\mathsf{adv}[\mathcal{A}, F] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Definition 8.5. *We say that a PRF F is secure in the random oracle model if for all efficient adversaries $\mathcal{A}$, the value* $\text{PRF}^{\text{ro}}\mathsf{adv}[\mathcal{A}, F]$ *is negligible.*

Consider again the PRF $F_{\text{pre}}(k, x) := H(k \parallel x)$. Let us assume that F_{pre} is defined over $(\mathcal{K}, \mathcal{X}, \mathcal{T})$, where $\mathcal{K} = \{0,1\}^{\kappa}$ and $\mathcal{X} = \{0,1\}^{\leq L}$, and that H is defined over $(\mathcal{M}, \mathcal{T})$, where M includes all bit strings of length at most $\kappa + L$.

We will show that this is a secure PRF in the random oracle model. But wait! We already argued in Section 8.7 that F_{pre} is completely insecure when H is a Merkle-Damgård hash. This seems to be a contradiction. The problem is that, as already mentioned, it is not safe to use a Merkle-Damgård hash directly as a random oracle. We will see how to fix this problem in Section 8.10.3.

Theorem 8.9. *If $\mathcal{K}$ is large then F_{pre} is a secure PRF when H is modeled as a random oracle.*

> *In particular, if $\mathcal{A}$ is a random oracle PRF adversary, as in Attack Game 8.4, that makes at most Q_{ro} oracle queries, then*
>
> $$\text{PRF}^{\text{ro}}\mathsf{adv}[\mathcal{A}, F_{\text{pre}}] \leq Q_{\text{ro}}/|\mathcal{K}|$$

Note that Theorem 8.9 is unconditional, in the sense that the only constraint on $\mathcal{A}$ is on the number of oracle queries: it does not depend on any complexity assumptions.

Proof idea. Once H is replaced with $\mathcal{O}$, the adversary has to distinguish $\mathcal{O}(k \parallel \cdot)$ from a random function in $\text{Funs}[\mathcal{X}, \mathcal{T}]$, without the key k. Since $\mathcal{O}(k \parallel \cdot)$ is a random function in $\text{Funs}[\mathcal{X}, \mathcal{T}]$, the only hope the adversary has is to somehow use the information returned from queries to $\mathcal{O}$. We say that an $\mathcal{O}$-query $k' \parallel x'$ is relevant if $k' = k$. It should be clear that queries to $\mathcal{O}$ that are not relevant cannot help distinguish $\mathcal{O}(k \parallel \cdot)$ from random since the returned values are independent of the function $\mathcal{O}(k \parallel \cdot)$. Moreover, the probability that after Q_{ro} queries the adversary succeeds in issuing a relevant query is at most $Q_{\text{ro}}/|\mathcal{K}|$. □

Proof. To make this proof idea rigorous we let $\mathcal{A}$ interact with two PRF challengers. For $j = 0, 1$, let W_j to be the event that $\mathcal{A}$ outputs 1 in Game j.

Game 0. We write the challenger in Game 0 so that it is equivalent to Experiment 0 of Attack Game 8.4, but will be more convenient for us to analyze. We assume the adversary never makes the same F_{pre}-query twice. Also, we use an associative array $Map : \mathcal{M} \to \mathcal{T}$ to build up the random oracle on the fly, using the "faithful gnome" idea we have used so often. Here is our challenger:

```
      Initialization:
            initialize the empty associative array $Map : \mathcal{M} \to \mathcal{T}$
            $k \xleftarrow{R} \mathcal{K}$
      Upon receiving an $F_{\text{pre}}$-query on $x \in \{0,1\}^{\leq L}$ do:
            $t \xleftarrow{R} \mathcal{T}$
(1)         if $(k \parallel x) \in \text{Domain}(Map)$ then $t \leftarrow Map[k \parallel x]$
(2)         $Map[k \parallel x] \leftarrow t$
            send $t$ to $\mathcal{A}$
      Upon receiving an $\mathcal{O}$-query $m \in \mathcal{M}$ do:
            $t \xleftarrow{R} \mathcal{T}$
            if $m \in \text{Domain}(Map)$ then $t \leftarrow Map[m]$
            $Map[m] \leftarrow t$
            send $t$ to $\mathcal{A}$
```

It should be clear that this challenger is equivalent to that in Experiment 0 of Attack Game 8.4. In Game 0, whenever the challenger needs to sample the random oracle at some input (in processing either an F_{pre}-query or an $\mathcal{O}$-query), it generates a random "default output", overriding that default if it turns out the oracle has already been sampled at that input; in either case, the associative array records the input/output pair.

Game 1. We make our gnome "forgetful": we modify Game 0 by deleting the lines marked (1) and (2) in that game. Observe now that in Game 1, the challenger does not use Map or k in responding to F_{pre}-queries: it just returns a random value. So it is clear (by the assumption that $\mathcal{A}$ never makes the same F_{pre}-query twice) that Game 1 is equivalent to Experiment 1 of Attack Game 8.4, and hence

$$\text{PRF}^{\text{ro}}\mathsf{adv}[\mathcal{A}, F_{\text{pre}}] = |\Pr[W_1] - \Pr[W_0]|.$$

Let Z be the event that *in Game 1*, the adversary makes an $\mathcal{O}$-query at a point $m = (k \parallel \hat{x})$. It is clear that both games result in the same outcome unless Z occurs, so by the by Difference Lemma, we have

$$|\Pr[W_1] - \Pr[W_0]| \leq \Pr[Z].$$

Since the key k is completely independent of $\mathcal{A}$'s view in Game 1, each $\mathcal{O}$-query hits the key with probability $1/|\mathcal{K}|$, and so a simple application of the union bound yields

$$\Pr[Z] \leq Q_{\text{ro}}/|\mathcal{K}|.$$

That completes the proof. □

Key derivation in the random oracle model. Let us now return to the key derivation problem introduced in Section 8.10.1. Again, we have a secret s sampled from some distribution P, and information $I(s)$ is leaked to the adversary. We want to argue that if H is modeled as a random oracle, then the adversary's advantage in distinguishing $(I(s), H(s))$ from $(I(s), t)$, where t is truly random, is not too much more than the adversary's advantage in guessing the secret s with only $I(s)$ (and not $H(s)$).

To model H as a random oracle $\mathcal{O}$, we convert the computational indistinguishability Attack Game 3.3 to the random oracle model, so that the attacker is now trying to distinguish $(I(s), \mathcal{O}(s))$ from $(I(s), t)$, given oracle access to $\mathcal{O}$. The corresponding advantage is denoted $\text{Dist}^{\text{ro}}\mathsf{adv}[\mathcal{A}, P, I, H]$.

Before stating our security theorem, it is convenient to generalize Attack Game 8.3 to allow the adversary to output a list of guesses $\hat{s}_1, \ldots, \hat{s}_Q$, where the adversary is said to win the game if $\hat{s}_i = s$ for some $i = 1, \ldots, Q$. An adversary $\mathcal{A}$'s probability of winning in this game is called his **list guessing advantage**, denoted $\text{ListGuess}\mathsf{adv}[\mathcal{A}, P, I]$.

Clearly, if an adversary $\mathcal{A}$ can win the above list guessing game with probability ϵ, we can convert him into an adversary that wins the singleton guessing game with probability ϵ/Q: we simply run $\mathcal{A}$ to obtain a list $\hat{s}_1, \ldots, \hat{s}_Q$, choose $i = 1, \ldots, Q$ at random, and output $\hat{s}_i$. However, sometimes we can do better than this: using the partial information $I(s)$ may allow us to rule out some of the $\hat{s}_i$'s, and in some situations, we may be able to identify the correct $\hat{s}_i$ uniquely. This depends on the application.

Theorem 8.10. *If H is modeled as a random oracle, then for every distinguishing adversary $\mathcal{A}$ that makes at most Q_{ro} random oracle queries, there exists a list guessing adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that*

$$\text{Dist}^{\text{ro}}\mathsf{adv}[\mathcal{A}, P, I, H] \leq \text{ListGuess}\mathsf{adv}[\mathcal{B}, P, I]$$

and $\mathcal{B}$ outputs a list of size at most Q_{ro}. In particular, there exists a guessing adversary $\mathcal{B}'$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\text{Dist}^{\text{ro}}\mathsf{adv}[\mathcal{A}, P, I, H] \leq Q_{\text{ro}} \cdot \text{Guess}\mathsf{adv}[\mathcal{B}', P, I].$$

Proof. The proof is almost identical to that of Theorem 8.9. We define two games, and for $j = 0, 1$, let W_j to be the event that $\mathcal{A}$ outputs 1 in Game j.

Game 0. We write the challenger in Game 0 so that it is equivalent to Experiment 0 of the $(I(s), H(s))$ vs $(H(s), t)$ distinguishing game. We build up the random oracle on the fly with an associative array $Map : \mathcal{S} \to \mathcal{T}$. Here is our challenger:

```
      Initialization:
            initialize the empty associative array $Map : \mathcal{S} \to \mathcal{T}$
            generate $s$ according to $P$
            $t \xleftarrow{R} \mathcal{T}$
(*)         $Map[s] \leftarrow t$
            send $(I(s), t)$ to $\mathcal{A}$
      Upon receiving an $\mathcal{O}$-query $\hat{s} \in \mathcal{S}$ do:
            $\hat{t} \xleftarrow{R} \mathcal{T}$
            if $\hat{s} \in$ Domain($Map$) then $\hat{t} \leftarrow Map[\hat{s}]$
            $Map[\hat{s}] \leftarrow \hat{t}$
            send $\hat{t}$ to $\mathcal{A}$
```

Game 1. We delete the line marked $(*)$. This game is equivalent to Experiment 1 of this distinguishing game, as the value t is now truly independent of the random oracle. Moreover, both games result in the same outcome unless the adversary $\mathcal{A}$ in Game 1 makes an $\mathcal{O}$-query at the point s. So our list guessing adversary $\mathcal{B}$ simply takes the value $I(s)$ that it receives from its own challenger, and plays the role of challenger to $\mathcal{A}$ as in Game 1. At the end of the game, $\mathcal{B}$ simply outputs Domain(Map) — the list of points at which $\mathcal{A}$ made $\mathcal{O}$-queries. The essential points are: our $\mathcal{B}$ can play this role with no knowledge of s besides $I(s)$, and it records all of the $\mathcal{O}$-queries made by $\mathcal{A}$. So by the Difference Lemma, we have

$$\text{Dist}^{\text{ro}}\mathsf{adv}[\mathcal{A}] = |\Pr[W_0] - \Pr[W_1]| \le \text{ListGuess}\mathsf{adv}[\mathcal{B}]. \quad \Box$$

8.10.3 Random oracles: safe modes of operation

We have already seen that $F_{\text{pre}}(k,x) := H(k \parallel x)$ is secure in the random oracle model, and yet we know that it is completely insecure if H is a Merkle-Damgård hash. The problem is that a Merkle-Damgård construction has a very simple, iterative structure which exposes it to "extension attacks". While this structure is not a problem from the point of view of collision resistance, it shows that grabbing a hash function "off the shelf" and using it as if it were a random oracle is a dangerous move.

In this section, we discuss how to safely use a Merkle-Damgård hash as a random oracle. We will also see that the sponge construction (see Section 8.8) is already safe to use "as is"; in fact, the sponge was designed exactly for this purpose: to provide a variable-length input and variable-length output hash function that could be used directly as a random oracle.

Suppose H is a Merkle-Damgård hash built from a compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. One recommended mode of operation is to use HMAC with a zero key:

$$\text{HMAC}_0(m) := \text{HMAC}(0^\ell, m) = H(\mathsf{opad} \parallel H(\mathsf{ipad} \parallel m)).$$

While this construction foils the obvious extension attacks, why should we have any confidence that HMAC_0 is safe to use as a general purpose random oracle? We can only give heuristic evidence. Essentially, what we want to argue is that there are no inherent structural weaknesses in HMAC_0 that give rise to a generic attack that treats the underlying compression function itself as a random oracle — or perhaps, more realistically, as a Davies-Meyer construction based on an ideal cipher.

So basically, we want to show that using certain modes of operation, we can build a "big" random oracle out of a "small" random oracle — or out of an ideal cipher or even out of an ideal permutation.

The mathematical tool used to carry out such a task is called **indifferentiability**. We shall present a somewhat simplified version of this notion here. Suppose we are trying to build a "big" random oracle $\mathcal{O}$ out of a smaller primitive ρ, where ρ could be a random oracle on a small domain, or an ideal cipher, or an ideal permutation. Let us denote by $F[\rho]$ a particular construction for a random oracle based on the ideal primitive ρ.

Now consider a generic attack game defined by some challenger $\mathbf{C}$ and adversary $\mathcal{A}$. Let us write the interaction between $\mathbf{C}$ and $\mathcal{A}$ as $\langle \mathbf{C}, \mathcal{A} \rangle$. We assume that the interaction results in an output bit. All of our security definitions are modeled in terms of games of this form.

In the random oracle version of the attack game, with the big random oracle $\mathcal{O}$, we would give both the challenger and adversary oracle access to the random function $\mathcal{O}$, and we denote the interaction $\langle \mathbf{C}^{\mathcal{O}}, \mathcal{A}^{\mathcal{O}} \rangle$. However, if we are using the construction $F[\rho]$ to implement the big random oracle, then while the challenger accesses ρ only via the construction F, the adversary is allowed to directly query ρ. We denote this interaction as $\langle \mathbf{C}^{F[\rho]}, \mathcal{A}^{\rho} \rangle$.

For example, in the HMAC_0 construction, the compression function h is modeled as a random oracle ρ, or if h itself is built via Davies-Meyer, then the underlying block cipher is modeled as an ideal cipher ρ. In either case, $F[\rho]$ corresponds to the HMAC_0 construction itself. Note the asymmetry: in any attack game, the challenger only accesses ρ indirectly via $F[\rho]$ (HMAC_0 in this case), while the adversary can access ρ itself (the compression function h or the underlying block cipher).

We say that $F[\rho]$ is **indifferentiable** from $\mathcal{O}$ if the following holds:

> *for every efficient challenger* $\mathbf{C}$ *and efficient adversary* $\mathcal{A}$*, there exists an efficient adversary* $\mathcal{B}$*, which is an elementary wrapper around* $\mathcal{A}$*, such that*
>
> $$\left|\Pr[\langle \mathbf{C}^{F[\rho]}, \mathcal{A}^{\rho} \rangle \text{ outputs } 1] - \Pr[\langle \mathbf{C}^{\mathcal{O}}, \mathcal{B}^{\mathcal{O}} \rangle \text{ outputs } 1]\right|$$
>
> *is negligible.*

It should be clear from the definition that if we prove security of any cryptographic scheme in the random oracle model for the big random oracle $\mathcal{O}$, the scheme remains secure if we implement $\mathcal{O}$ using $F[\rho]$: if an adversary $\mathcal{A}$ could break the scheme with $F[\rho]$, then the adversary $\mathcal{B}$ above would break the scheme with $\mathcal{O}$.

Some safe modes. The HMAC_0 construction can be proven to be indifferentiable from a random oracle on variable length inputs, if we either model the compression function h itself as a random oracle, or if h is built via Davies-Meyer and we model the underlying block cipher as an ideal cipher.

One problem with using HMAC_0 as a random oracle is that its *output* is fairly short. Fortunately, it is fairly easy to use HMAC_0 to get a random oracle with longer outputs. Here is how. Suppose HMAC_0 has an n-bit output, and we need a random oracle with, say, $N > n$ bits of output. Set $q := \lceil N/n \rceil$. Let $e_0, e_1, \ldots, e_q$ be fixed-length encodings of the integers $0, 1, \ldots, q$. Our new hash function H' works as follows. On input m, we compute $t \leftarrow \text{HMAC}_0(e_0 \parallel m)$. Then, for $i = 1, \ldots, q$, we compute $t_i \leftarrow \text{HMAC}_0(e_i \parallel t)$. Finally, we output the first N bits of $t_1 \parallel t_2 \parallel \cdots \parallel t_q$. One can show that H' is indifferentiable from a random oracle with N-bit outputs. This result holds if we replace HMAC_0 with any hash function that is itself indifferentiable from a random oracle with n-bit outputs. Also note that when applied to long inputs, H' is quite efficient: it only needs to evaluate HMAC_0 once on a long input.

The sponge construction has been proven to be indifferentiable from a random oracle on variable length inputs, if we model the underlying permutation as an ideal permutation (assuming 2^c is super-poly, where c is the capacity). This includes the standardized implementations SHA3 (for fixed length outputs) and the SHAKE variants (for variable length outputs), discussed in Section 8.8.2. The special padding rules used in the SHA3 and SHAKE specifications ensure that all of the variants act as independent random oracles.

Sometimes, we need random oracles whose output should be uniformly distributed over some specialized set. For example, we may want the output to be uniformly distributed over the set $S = \{0, \ldots, d-1\}$ for some positive integer d. To realize this, we can use a hash function H with an n-bit output, which we can view as an n-bit binary encoding of a number, and define $H'(m) := H(m) \bmod d$. If H is indifferentiable from a random oracle with n-bit outputs, and $2^n/d$ is super-poly, then the hash function H' is indifferentiable from a random oracle with outputs in S.

8.10.4 The leftover hash lemma

We now return to the key derivation problem. Under the right circumstances, we can solve the key derivation problem with no heuristics and no computational assumptions whatsoever. Moreover, the solution is a surprising and elegant application of universal hash functions (see Section 7.1). The result, known as the **leftover hash lemma**, says that if we use an ϵ-UHF to hash a secret that can be guessed with probability at most γ, then provided ϵ and γ are sufficiently small, the output of the hash is statistically indistinguishable from a truly random value. Recall that a UHF has a key, which we normally think of as a secret key; however, in this result, the key may be made public — indeed, it could be viewed as a public, system parameter that is generated once and for all, and used over and over again.

Our goal here is to simply state the result, and to indicate when and where it can (and cannot) be used. To state the result, we will need to use the notion of the statistical distance between two random variables, which we introduced in Section 3.11. Also, if s is a random variable taking values in a set $\mathcal{S}$, we define the **guessing probability of** s to be $\max_{x \in \mathcal{S}} \Pr[\mathsf{s} = x]$.

Theorem 8.11 (Leftover Hash Lemma). *Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{S}, \mathcal{T})$. Assume that H is a $(1+\alpha)/N$-UHF, where $N := |\mathcal{T}|$. Let $\mathsf{k}, \mathsf{s}_1, \ldots, \mathsf{s}_m$ be mutually independent random variables, where k is uniformly distributed over $\mathcal{K}$, and each s_i has guessing probability at most γ. Let δ be the statistical difference between*

$$(\mathsf{k}, H(\mathsf{k}, \mathsf{s}_1), \ldots, H(\mathsf{k}, \mathsf{s}_m))$$

and the uniform distribution on $\mathcal{K} \times \mathcal{T}^m$. Then we have

$$\delta \leq \frac{1}{2} m \sqrt{N\gamma + \alpha}.$$

Let us look at what the lemma says when $m = 1$. We have a secret s that can be guessed with probability at most γ, given whatever side information $I(s)$ is known about s. To apply the lemma, the bound γ on the guessing probability must hold for all adversaries, even computationally unbounded ones. We then hash s using a random hash key k. It is essential that s (given $I(s)$) and k are independent — although we have not discussed the possibility here, there are potential use cases where the distribution of s or the function I can be somehow biased by an adversary in a way that depends on k, which is assumed public and known to the adversary. Therefore, to apply the lemma, we must ensure that s (given $I(s)$) and k are truly independent. If all of these conditions are met, then the lemma says that for any adversary $\mathcal{A}$, even a computationally unbounded one, its advantage in distinguishing $(k, I(s), H(k, s))$ from $(k, I(s), t)$, where t is a truly random element of $\mathcal{T}$, is bounded by δ, as in the lemma.

Now let us plug in some realistic numbers. If we want the output to be used as an AES key, we need $N = 2^{128}$. We know how to build $(1/N)$-UHFs, so we can take $\alpha = 0$ (see Exercise 7.18 — with α non-zero, but still quite small, one can get by with significantly shorter hash keys). If we want $\delta \leq 2^{-64}$, we will need the guessing probability γ to be about 2^{-256}.

So in addition to all the conditions listed above, we really need an extremely small guessing probability for the lemma to be applicable. None of the examples discussed in Section 8.10.1 meet these requirements: the guessing probabilities are either not small enough, or do not hold unconditionally against unbounded adversaries, or can only be heuristically estimated. So the practical applicability to the Leftover Hash Lemma is limited — but when it does apply, it can be a very powerful tool. Also, we remark that by using the lemma with $m > 1$, under the right conditions, we can model the situation where the same hash key is used to derive many keys from many independent secrets with small guessing probability. The distinguishing probability grows linearly with the number of derivations, which is not surprising.

Because of these practical limitations, it is more typical to use cryptographic hash functions, modeled as random oracles, for key derivation, rather than UHFs. Indeed, if one uses a UHF and any of the assumptions discussed above turns out to be wrong, this could easily lead to a catastrophic security breach. Cryptographic hash functions, while only heuristically secure for key derivation, are also more forgiving.

8.10.5 Case study: HKDF

HKDF is a key derivation function specified in RFC 5869, and is deployed in many standards.

HKDF is specified in terms of the HMAC construction (see Section 8.7). So it uses the function $\text{HMAC}(k, m)$, where k and m are variable length byte strings, which itself is implemented in terms of a Merkle-Damgård hash H, such as SHA256.

The input to HKDF consists of a secret s, an optional salt value *salt* (discussed below), an optional *info* field (also discussed below), and an output length parameter L. The parameters s, *salt*, and *info* are variable length byte strings.

The execution of HKDF consists of two stages, called *extract* (which corresponds to what we called key derivation), and *expand* (which corresponds to what we called sub-key derivation).

In the extract stage, HKDF uses *salt* and s to compute

$$t \leftarrow \text{HMAC}(\mathit{salt}, s).$$

Using the intermediate key t, along with *info*, the expand (or sub-key derivation) stage computes L bytes of output data, as follows:

```
$q \leftarrow \lceil L/\mathit{HashLen} \rceil$   // $\mathit{HashLen}$ is the output length (in bytes) of $H$
initialize $z_0$ to the empty string
for $i \leftarrow 1$ to $q$ do:
        $z_i \leftarrow \text{HMAC}(t, z_{i-1} \parallel \mathit{info} \parallel \mathit{Octet}(i))$   // $\mathit{Octet}(i)$ is a single byte whose value is $i$
output the first $L$ octets of $z_1 \parallel \ldots \parallel z_q$
```

When *salt* is empty, the extract stage of HKDF is the same as what we called HMAC_0 in Section 8.10.3. As discussed there, HMAC_0 can heuristically be viewed as a random oracle, and so we can use the analysis in Section 8.10.2 to show that this is a secure key derivation procedure in the random oracle model. Thus, if s is hard to guess, then t is indistinguishable from random.

Users of HKDF have the option of providing a non-zero salt. The salt plays a role akin to the random hash key used in the Leftover Hash Lemma (see Section 8.10.4); in particular, it need not be secret, and may be reused. However, it is important that the salt value is independent of the secret s and cannot be manipulated by an adversary. The idea is that under these circumstances, the output of the extract stage of HKDF seems more likely to be indistinguishable from random, without relying on the full power of the random oracle model. Unfortunately, the known security proofs apply to limited settings, so in the general case, this is still somewhat heuristic.

The expand stage is just a simple application of HMAC as a PRF to derive sub-keys, as we discussed at the end of Section 8.10.1. The *info* parameter may be used to "name" the derived sub-keys, ensuring the independence of keys used for different purposes. Since the output length of the underlying hash is fixed, a simple iterative scheme is used to generate longer outputs. This stage can be analyzed rigorously under the assumption that the intermediate key t is indistinguishable from random, and that HMAC is a secure PRF — and we already know that HMAC is a secure PRF, under reasonable assumptions about the compression function of H.

8.11 Security without collision resistance

Theorem 8.1 shows how to extend the domain of a MAC using a collision resistant hash. It is natural to ask whether MAC domain extension is possible without relying on collision resistant functions. In this section we show that a weaker property called second preimage resistance is sufficient.

8.11.1 Second preimage resistance

We start by defining two classic security properties for non-keyed hash functions: one-wayness and 2nd-preimage resistance. Let H be a hash function defined over $(\mathcal{M}, \mathcal{T})$.

- We say that H is **one-way** if given $t := H(m)$ as input, for a random $m \in \mathcal{M}$, it is difficult to find an $m' \in \mathcal{M}$ such that $H(m') = t$. Such an m' is called an inverse of t. In other words, H is one-way if it is easy to compute but difficult to invert.

- We say that H is **2nd-preimage resistant** if given a random $m \in \mathcal{M}$ as input, it is difficult to find a different $m' \in \mathcal{M}$ such that $H(m) = H(m')$. In other words, it is difficult to find an m' that collides with a given m.

- For completeness, recall that a hash function is collision resistant if it is difficult to find two distinct messages $m, m' \in \mathcal{M}$ such that $H(m) = H(m')$.

The following defines these concepts more precisely.

Definition 8.6. *Let H be a hash function defined over $(\mathcal{M}, \mathcal{T})$.*

- *We define the advantage $\mathrm{OW}\mathsf{adv}[\mathcal{A}, H]$ of an adversary $\mathcal{A}$ in defeating the one-wayness of H as the probability of winning the following game:*

  - *the challenger chooses $m \in \mathcal{M}$ at random and sends $t := H(m)$ to $\mathcal{A}$;*
  - *the adversary $\mathcal{A}$ outputs $m' \in \mathcal{M}$, and wins if $H(m') = t$.*

  *H is **one-way** if $\mathrm{OW}\mathsf{adv}[\mathcal{A}, H]$ is negligible for every efficient adversary $\mathcal{A}$.*

- *We define the advantage $\mathrm{SPR}\mathsf{adv}[\mathcal{A}, H]$ of an adversary $\mathcal{A}$ in defeating the 2nd-preimage resistance of H as the probability of winning the following game:*

  - *the challenger chooses $m \in \mathcal{M}$ at random and sends m to $\mathcal{A}$;*
  - *the adversary $\mathcal{A}$ outputs $m' \in \mathcal{M}$, and wins if $H(m') = H(m)$ and $m' \neq m$.*

  *H is **2nd-preimage resistant** if $\mathrm{SPR}\mathsf{adv}[\mathcal{A}, H]$ is negligible for every efficient adversary $\mathcal{A}$.*

Let's first look at some easy relations between these security notions. Suppose H is compressing, meaning that the domain of H is bigger than its range. Specifically, suppose $|\mathcal{M}| \geq s \cdot |\mathcal{T}|$ for some compression factor $s > 1$. If s is super-poly, then we have the following implications:

$$H \text{ is collision resistant} \quad \Rightarrow \quad H \text{ is 2nd-preimage resistant} \quad \Rightarrow \quad H \text{ is one-way}, \tag{8.17}$$

as shown in Exercise 8.23. In fact, the implication on the left holds with no restriction on s.

The converse of (8.17) is not true. A hash function can be 2nd-preimage resistant, but not collision resistant. For example, SHA1 is believed to be 2nd-preimage resistant even though SHA1 is not collision resistant. Similarly, a hash function can be one-way, but not be 2nd-preimage resistant. For example, let h be a one-way function defined over $(\mathcal{M}, \mathcal{T})$. Let h' be a function over $(\mathcal{M} \times \{0,1\},\ \mathcal{T})$ defined as $h'(m, b) := h(m)$. Then clearly h' is one-way, but is not 2nd-preimage resistant: given $(m, b) \in \mathcal{M} \times \{0,1\}$, the input $(m,\ 1-b)$ is a second preimage because $h'(m, b) = h'(m,\ 1-b)$.

Our goal for this section is to show that 2nd-preimage resistance is sufficient for extending the domain of a MAC and for providing file integrity. To give some intuition, consider the file integrity problem (which we discussed at the very beginning of this chapter). Our goal is to ensure that malware cannot modify a file without being detected. Recall that we hash all critical files on disk using a hash function H and store the resulting hashes in read-only memory. For a file F it should be difficult for the malware to find an F' such that $H(F') = H(F)$. Clearly, if H is collision resistant then finding such an F' is difficult. It would seem, however, that 2nd-preimage resistance of H is sufficient. To see why, consider malware trying to modify a specific file F without being detected. The malware is given F as input and must come up with a 2nd-preimage of F, namely an F' such that $H(F') = H(F)$. If H is 2nd-preimage resistant the malware cannot find such an F' and it would seem that 2nd-preimage resistance is sufficient for file integrity. Unfortunately, this argument doesn't quite work. Our definition of 2nd-preimage resistance says that finding a 2nd-preimage for a *random* F in $\mathcal{M}$ is difficult. But files on disk are not random bit strings — it may be difficult to find a 2nd-preimage for a random file, but it may be quite easy to find a 2nd-preimage for a specific file on disk.

The solution is to randomize the data before hashing it. To do so we first convert the hash function to a *keyed* hash function. We then require that the resulting keyed function satisfy a property called *target collision resistance* which we now define.

8.11.2 Randomized hash functions: target collision resistance

At the beginning of the chapter we mentioned two applications for collision resistance: extending the domain of a MAC and protecting file integrity. In this section we describe solutions to these problems that rely on a weaker security property than collision resistance. The resulting systems, although more likely to be secure, are not as efficient as the ones obtained from collision resistance.

Target collision resistance. Let H be a *keyed* hash function. We define what it means for H to be **target collision resistant**, or TCR for short, using the following attack game, also shown in Fig. 8.14.

Attack Game 8.5 (Target collision resistance). For a given keyed hash function H over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ and adversary $\mathcal{A}$, the attack game runs as follows:

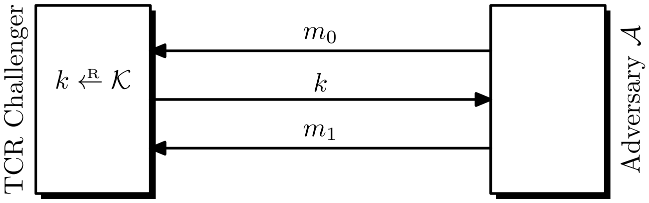


Figure 8.14: TCR Attack Game

- $\mathcal{A}$ sends a message $m_0 \in \mathcal{M}$ to the challenger.
- The challenger picks a random $k \xleftarrow{R} \mathcal{K}$ and sends k to $\mathcal{A}$.
- $\mathcal{A}$ sends a second message $m_1 \in \mathcal{M}$ to the challenger.

The adversary is said to win the game if $m_0 \neq m_1$ and $H(k, m_0) = H(k, m_1)$. We define $\mathcal{A}$'s advantage with respect to H, denoted $\text{TCR}\mathsf{adv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 8.7. *We say that a keyed hash function H over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ is target collision resistant if* $\text{TCR}\mathsf{adv}[\mathcal{A}, H]$ *is negligible.*

Casting the definition in our formal mathematical framework is done exactly as for universal hash functions (Section 7.1.2).

We note that one can view a collision resistant hash H over $(\mathcal{M}, \mathcal{T})$ as a TCR function with an empty key. More precisely, let $\mathcal{K}$ be a set of size one containing only the empty word. We can define a keyed hash function H' over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ as $H'(k, m) := H(m)$. It is not difficult to see that if H is collision resistant then H' is TCR. Thus, a collision resistant function can be viewed as the ultimate TCR hash — its key is the shortest possible.

8.11.3 TCR from 2nd-preimage resistance

We show how to build a keyed TCR hash function from a keyless 2nd-preimage resistant function such as SHA1. Let H, defined over $(\mathcal{M}, \mathcal{T})$, be a 2nd-preimage resistant function. We construct a keyed TCR function H_{tcr} defined over $(\mathcal{M}, \mathcal{M}, \mathcal{T})$ as follows:

$$H_{\text{tcr}}(k, m) = H(k \oplus m) \tag{8.18}$$

Note that the length of the key k is equal to the length of the message being hashed. This is a problem for the applications we have in mind. As a result, we will only use this construction as a TCR hash for short messages. First we prove that the construction is secure.

Theorem 8.12. *Suppose H is 2nd-preimage resistant then H_{tcr} is TCR.*

In particular, for every TCR adversary $\mathcal{A}$ attacking H_{tcr} as in Attack Game 8.5, there exists a 2nd-preimage finder $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\text{TCR}\mathsf{adv}[\mathcal{A}, H_{tcr}] \leq \text{SPR}\mathsf{adv}[\mathcal{B}, H].$$

Proof. The proof is a simple direct reduction. Adversary $\mathcal{B}$ emulates the challenger in Attack Game 8.5 and works as follows:

```
Input: Random $m \in \mathcal{M}$
Output: $m' \in \mathcal{M}$ such that $m \neq m'$ and $H(m) = H(m')$
1.  Run $\mathcal{A}$ and obtain an $m_0 \in \mathcal{M}$ from $\mathcal{A}$
2.  $k \leftarrow m \oplus m_0$
3.  Send $k$ as the hash key to $\mathcal{A}$
4.  $\mathcal{A}$ responds with an $m_1 \in \mathcal{M}$
5.  Output $m' := m_1 \oplus k$
```

We show that $\text{SPR}\mathsf{adv}[\mathcal{B}, H] = \text{TCR}\mathsf{adv}[\mathcal{A}, H_{\text{tcr}}]$. First, denote by W the event that in step (4) the messages m_0, m_1 output by $\mathcal{A}$ are distinct and $H_{\text{tcr}}(k, m_0) = H_{\text{tcr}}(k, m_1)$.

The input m given to $\mathcal{B}$ is uniformly distributed in $\mathcal{M}$. Therefore, the key k given to $\mathcal{A}$ in step (2) is uniformly distributed in $\mathcal{M}$ and independent of $\mathcal{A}$'s current view, as required in Attack Game 8.5. It follows that $\mathcal{B}$ perfectly emulates the challenger in Attack Game 8.5 and consequently $\Pr[W] = \text{TCR}\mathsf{adv}[\mathcal{A}, H_{\text{tcr}}]$.

By definition of H_{tcr}, we also have the following:

$$\begin{aligned} H_{\text{tcr}}(k, m_0) &= H((m \oplus m_0) \oplus m_0) = H(m) \\ H_{\text{tcr}}(k, m_1) &= H(m_1 \oplus k) = H(m') \end{aligned} \tag{8.19}$$

Now, suppose event W happens. Then $H_{\text{tcr}}(k, m_0) = H_{\text{tcr}}(k, m_1)$ and therefore, by (8.19), we know that $H(m) = H(m')$. Second, we deduce that $m \neq m'$ which follows since $m_0 \neq m_1$ and $m' = m \oplus (m_1 \oplus m_0)$. Hence, when event W occurs, $\mathcal{B}$ outputs a 2nd-preimage of m. It now follows that:

$$\text{SPR}\mathsf{adv}[\mathcal{B}, H] \geq \Pr[W] = \text{TCR}\mathsf{adv}[\mathcal{A}, H_{\text{tcr}}]$$

as required. □

Target collision resistance for long inputs. The function H_{tcr} in (8.18) shows that a 2nd-preimage resistant function directly gives a TCR function. If we assume that the SHA256 compression function h is 2nd-preimage resistant (a weaker assumption than assuming that h is collision resistant) then, by Theorem 8.12 we obtain a TCR hash for inputs of length $512 + 256 = 768$ bits. The length of the required key is also 768 bits.

We will often need TCR functions for much longer inputs. Using the SHA256 compression function we already know how to build a TCR hash for short inputs using a short key. Thus, let us assume that we have a TCR function h defined over $(\mathcal{K},\ \mathcal{T} \times \mathcal{M},\ \mathcal{T})$ where $\mathcal{M} := \{0,1\}^{\ell}$ for some small ℓ, say $\ell = 512$. We build a new TCR hash for much larger inputs. Let $L \in \mathbb{Z}^{>0}$ be a power of 2. We build a **derived TCR hash** H that hashes messages in $\{0,1\}^{\leq \ell L}$ using keys in $(\mathcal{K} \times \mathcal{T}^{1+\log_2 L})$. Note that the length of the keys is *logarithmic* in the length of the message, which is much better than (8.18).

To describe the function H we need an auxiliary function $\nu : \mathbb{Z}^{>0} \to \mathbb{Z}^{>0}$ defined as:

$$\nu(x) := \text{largest } n \in \mathbb{Z}^{>0} \text{ such that } 2^n \text{ divides } x.$$

Thus, $\nu(x)$ counts the number of least significant bits of x that are zero. For example, $\nu(x) = 0$ if x is odd and $\nu(x) = n$ if $x = 2^n$. Note that $\nu(x) \leq 7$ for more than 99% of the integers.

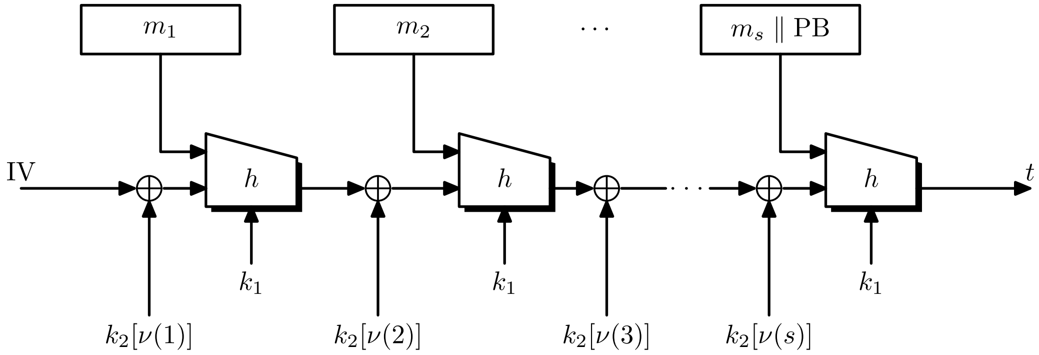


Figure 8.15: Extending the domain of a TCR hash

The derived TCR hash H is similar to Merkle-Damgård. It uses the same padding block PB as in Merkle-Damgård and a fixed initial value IV. The derived TCR hash H is defined as follows (see Fig. 8.15):

```
Input: Message $M \in \{0,1\}^{\leq \ell L}$ and key $(k_1, k_2) \in \mathcal{K} \times \mathcal{T}^{1+\log_2 L}$
Output: $t \in \mathcal{T}$
$M \leftarrow M \parallel \text{PB}$
Break $M$ into consecutive $\ell$-bit blocks so that
          $M = m_1 \parallel m_2 \parallel \cdots \parallel m_s$   where   $m_1, \ldots, m_s \in \{0,1\}^\ell$
$t_0 \leftarrow \text{IV}$
for $i = 1$ to $s$ do:
    $u \leftarrow k_2[\nu(i)] \oplus t_{i-1} \quad \in \mathcal{T}$
    $t_i \leftarrow h(k_1,\ (u, m_i)\ ) \quad \in \mathcal{T}$
Output $t_s$
```

We note that directly using Merkle-Damgård to extend the domain of a TCR hash does not work. Plugging $h(k_1, \cdot)$ directly into Merkle-Damgård can fail to give a TCR hash.

Security of the derived hash. The following theorem shows that the derived hash H is TCR assuming the underlying hash h is TCR. We refer to [146, 114] for the proof of this theorem.

Theorem 8.13. *Suppose h is a TCR hash function that hashes messages in $(\mathcal{T} \times \{0,1\}^\ell)$. Then, for any bounded L, the derived function H is a TCR hash for messages in $\{0,1\}^{\leq \ell L}$.*

In particular, suppose $\mathcal{A}$ is a TCR adversary attacking H (as in Attack Game 8.5). Then there exists a TCR adversary $\mathcal{B}$ (whose running times are about the same as that of $\mathcal{A}$) such that

$$\text{TCR}\mathsf{adv}[\mathcal{A}, H] \leq L \cdot \text{TCR}\mathsf{adv}[\mathcal{B}, h].$$

As in Merkle-Damgård this construction is inherently sequential. A tree-based construction similar to Exercise 8.9 gives a TCR hash using logarithmic size keys that is more suitable for a parallel machine. We refer to [14] for the details.

8.11.4 Using target collision resistance

We now know how to build a TCR function for large inputs from a small 2nd-preimage resistant function. We show how to use such TCR functions to extend the domain for a MAC and to ensure file integrity. We start with file integrity.

8.11.4.1 File integrity

Let H be a TCR hash defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. We use H to protect integrity of files $F_1, F_2, \ldots \in \mathcal{M}$ using a small amount of read-only memory. The idea is to pick a random key r_i in $\mathcal{K}$ for every file F_i and then store the pair $(r_i,\ H(r_i, F_i)\ )$ in read-only memory. Note that we are using a little more read-only memory than in the system based on collision resistance. To verify integrity of file F_i we simply recompute $H(r_i, F_i)$ and compare to the hash stored in read-only memory.

Why is this mechanism secure? Consider malware targeting a specific file F. We store in read-only memory the key r and $t := H(r, F)$. To modify F without being detected the malware must come up with a new file F' such that $t = H(r, F')$. In other words, the malware is given as input the file F along with a random key $r \in \mathcal{K}$ and must produce a new F' such that $H(r, F) = H(r, F')$. The adversary (the malware writer in this case) chooses which file F to attack. But this is precisely the TCR Attack Game 8.5 — the adversary chooses an F, gets a random key r, and must output a new F' that collides with F under r. Hence, if H is TCR the malware cannot modify F without being detected.

In summary, we can provide file integrity using a small amount of read-only memory and by relying only on 2nd-preimage resistance. The cost, in comparison to the system based on collision resistance, is that we need a little more read-only memory to store the key r. In particular, using the TCR construction from the previous section, the amount of additional read-only memory needed is logarithmic in the size of the files being protected. Using a recursive construction (see Exercise 8.25) we can reduce the additional read-only memory used to a small constant, but still non-zero.

8.11.4.2 Extending the domain of a MAC

Let H be a TCR hash defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{T})$. Let $\mathcal{I} = (S, V)$ be a MAC for authenticating short messages in $\mathcal{K}_H \times \mathcal{T}$ using keys in $\mathcal{K}$. We assume that $\mathcal{M}$ is much larger than $\mathcal{T}$. We build a new MAC $\mathcal{I}' = (S', V')$ for authenticating messages in $\mathcal{M}$ using keys in $\mathcal{K}$ as follows:

```
$S'(k,\ m) :=$                          $V'\big(k,\ m,\ (t, r)\big) :=$
        $r \xleftarrow{R} \mathcal{K}_H$                        $h \leftarrow H(r, m)$                    (8.20)
        $h \leftarrow H(r, m)$                        Output $V(k,\ (r, h),\ t)$
        $t \leftarrow S\big(k,\ (r, h)\big)$
        Output $(t, r)$
```

Note the MAC signing is randomized — we pick a random TCR key r, include r in the input to the signing algorithm S, and output r as part of the final tag. As a result, tags produced by this MAC are longer than tags produced from extending MACs using a collision resistance hash (as in Section 8.2). Using the construction from the previous section, the length of r is logarithmic in the size of the message being authenticated. This extra logarithmic size key is included in every tag. On the plus side, this construction only relies on H being TCR which is a much weaker property than collision resistance and hence much more likely to hold for H.

The following theorem proves security of the construction in (8.20) above. The theorem is the analog of Theorem 8.1 and its proof is similar. Note however, that the error bounds are not as tight as the bounds in Theorem 8.1.

Theorem 8.14. *Suppose the MAC system $\mathcal{I}$ is a secure MAC and the hash function H is TCR. Then the derived MAC system $\mathcal{I}' = (S', V')$ defined in (8.20) is a secure MAC.*

> *In particular, for every MAC adversary $\mathcal{A}$ attacking $\mathcal{I}'$ (as in Attack Game 6.1) that issues at most Q signing queries, there exist an efficient MAC adversary $\mathcal{B}_{\mathcal{I}}$ and an efficient TCR adversary $\mathcal{B}_H$, which are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}'] \leq \mathrm{MAC}\mathsf{adv}[\mathcal{B}_{\mathcal{I}}, \mathcal{I}] + Q \cdot \mathrm{TCR}\mathsf{adv}[\mathcal{B}_H, H].$$

Proof idea. Our goal is to show that no efficient MAC adversary can successfully attack $\mathcal{I}'$. Such an adversary $\mathcal{A}$ asks the challenger to sign a few long messages $m_1, m_2, \ldots \in \mathcal{M}$ and gets back tags (t_i, r_i) for $i = 1, 2, \ldots$. It then tries to invent a new valid message-MAC pair $(m,\ (t, r))$. If $\mathcal{A}$ is able to produce a valid forgery $(m,\ (t, r))$ then one of two things must happen:

1. either $(r, H(r, m))$ is equal to $(r_i, H(r_i, m_i))$ for some i;

2. or not.

It is not difficult to see that forgeries of the second type can be used to attack the underlying MAC $\mathcal{I}$. We show that forgeries of the first type can be used to break the target collision resistance of H. Indeed, if $(r, H(r, m)) = (r_i, H(r_i, m_i))$ then $r = r_i$ and therefore $H(r, m) = H(r, m_i)$. Thus m_i and m collide under the random key r. We will show that this lets us build an adversary $\mathcal{B}_H$ that wins the TCR game when attacking H. Unfortunately, $\mathcal{B}_H$ must guess ahead of time which of $\mathcal{A}$'s queries to use as m_i. Since there are Q queries to choose from, $\mathcal{B}_H$ will guess correctly with probability $1/Q$. This is the reason for the extra factor of Q in the error term. □

Proof. Let X be the event that adversary $\mathcal{A}$ wins the MAC Attack Game 6.1 with respect to $\mathcal{I}'$. Let $m_1, m_2, \ldots \in \mathcal{M}$ be $\mathcal{A}$'s queries during the game and let $(t_1, r_1),\ (t_2, r_2), \ldots$ be the challenger's responses. Furthermore, let $(m,\ (t, r))$ be the adversary's final output. We define two additional events:

- Let Y denote the event that for some $i = 1, 2, \ldots$ we have that $(r, H(r, m)) = (r_i, H(r, m_i))$ and $m \neq m_i$.
- Let Z denote the event that $\mathcal{A}$ wins Attack Game 6.1 on $\mathcal{I}'$ and event Y did not occur.

Then

$$\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}'] \ = \ \Pr[X] \ \leq \ \Pr[X \wedge \neg Y] + \Pr[Y] \ = \ \Pr[Z] + \Pr[Y] \tag{8.21}$$

To prove the theorem we construct a TCR adversary $\mathcal{B}_H$ and a MAC adversary $\mathcal{B}_{\mathcal{I}}$ such that

$$\Pr[Y] \leq Q \cdot \mathrm{TCR}\mathsf{adv}[\mathcal{B}_H, H] \quad \text{and} \quad \Pr[Z] = \mathrm{MAC}\mathsf{adv}[\mathcal{B}_{\mathcal{I}}, \mathcal{I}].$$

Adversary $\mathcal{B}_{\mathcal{I}}$ is essentially the same as in the proof of Theorem 8.1. Here we only describe the TCR adversary $\mathcal{B}_H$, which emulates a MAC challenger for $\mathcal{A}$ as follows:

```
$k \xleftarrow{R} \mathcal{K}$
$u \xleftarrow{R} \{1, 2, \ldots, Q\}$
Run algorithm $\mathcal{A}$
Upon receiving the $i$th signing query $m_i \in \mathcal{M}$ from $\mathcal{A}$ do:
    If $i \neq u$ then
        $r_i \xleftarrow{R} \mathcal{K}_H$
    Else  //  $i = u$: for query number $u$ get $r_i$ from the TCR challenger
        $\mathcal{B}_H$ sends $\hat{m}_0 := m_i$ to its TCR challenger
        $\mathcal{B}_H$ receives a random key $\hat{r} \in \mathcal{K}$ from its challenger
        $r_i \leftarrow \hat{r}$
    $h \leftarrow H(r_i, m_i)$
    $t \leftarrow S(k,\ (r_i, h)\ )$
    Send $(t, r)$ to $\mathcal{A}$
Upon receiving the final message-tag pair $(m,\ (t, r)\ )$ from $\mathcal{A}$ do:
    $\mathcal{B}_H$ sends $\hat{m}_1 := m$ to its challenger
```

Algorithm $\mathcal{B}_H$ responds to $\mathcal{A}$'s signature queries exactly as in a real MAC attack game. Therefore, event Y happens during the interaction with $\mathcal{B}_H$ with the same probability that it happens in a real MAC attack game. Now, when event Y happens there exists a $j \in \{1, 2, \ldots\}$ such that $(r, H(r, m)) = (r_j, H(r_j, m_j))$ and $m \neq m_j$. Suppose that furthermore $j = u$. Then $r = r_j = \hat{r}$ and therefore $H(\hat{r}, m) = H(\hat{r}, m_u)$. Hence, if event Y happens and $j = u$ then $\mathcal{B}_H$ wins the TCR attack game. In symbols,

$$\mathrm{TCR}\mathsf{adv}[\mathcal{B}_H, H] = \Pr[Y \wedge (j = u)].$$

Notice that u is independent of $\mathcal{A}$'s view — it is only used for choosing which random key r_i is from $\mathcal{B}_H$'s challenger, but no matter what u is, the key r_i given to $\mathcal{A}$ is always uniformly random. Hence, event Y is independent of the event $j = u$. For the same reason, if the adversary makes a total of w queries then $\Pr[j = u] = 1/w \geq 1/Q$. In summary,

$$\mathrm{TCR}\mathsf{adv}[\mathcal{B}_H, H] = \Pr[Y \wedge (j = u)] = \Pr[Y] \cdot \Pr[j = u] \geq \Pr[Y]/Q$$

as required. □

8.12 A fun application: commitments and auctions

Alice and Bob decide to participate in an auction for a rare porcelain vase, along with other bidders. The auction house that runs the auction uses a *Vickrey sealed bid auction*, a commonly used auction mechanism, that works as follows: each participant submits a secret sealed bid for the vase. Once all the bids are in, the party who submitted the highest bid gets the vase, and the price paid is the second-highest bid. This auction mechanism can be shown to have good game theoretic properties, assuming the participants do not collude, and do not know each other's bids until the auction is finished. Hence, the need for a sealed bid auction.

When all the participants are in the same room, the auction house can implement a Vickrey auction by having each participant submit their bid in a sealed envelope. After collecting all the envelopes, the auctioneer opens them and announces the results. The participants can inspect the envelopes and the papers in them to verify that the auction was administered correctly. There is no need to trust the auction house.

Let's see how to implement a sealed bid auction when the participants are remote and communicate with the auction house over the Internet. We do so using a cryptographic commitment, an important cryptographic primitive that has many applications. We have previously encountered commitments in Section 3.12, where we used them for a coin-flipping protocol between two parties.

Recall that a commitment scheme $\mathcal{C}$ lets one party, Alice, commit to a message $m \in \mathcal{M}$ by publishing a commitment string c. Later, Alice can open the commitment and convince some other party, Bob, that the committed message was m. More precisely, a **commitment scheme** for a finite message space $\mathcal{M}$, is a pair of efficient algorithms $\mathcal{C} = (C, V)$ where:

- Algorithm C is invoked as $(c, o) \xleftarrow{R} C(m)$, where $m \in \mathcal{M}$ is the message to be committed, c is the commitment string, and o is an opening string.

- Algorithm V is a deterministic algorithm invoked as $V(m, c, o)$ and outputs accept or reject.

- Correctness property: for all $m \in \mathcal{M}$, if we compute $(c, o) \xleftarrow{R} C(m)$, then $V(m, c, o) = \mathsf{accept}$ with probability 1.

Alice commits to a message $m \in \mathcal{M}$ by computing $(c, o) \xleftarrow{R} C(m)$. She sends the commitment c to Bob, and keeps o to herself. Later, when Alice wants to open the commitment, she sends m and o to Bob, and Bob verifies that the commitment was opened correctly by running $V(m, c, o)$.

A commitment scheme $\mathcal{C}$ is intended to be the digital analogue of a sealed envelope. As such, it needs to satisfy two properties:

- **Binding:** Once a commitment c is generated, Alice can only open it to a single message. In particular, for every efficient adversary $\mathcal{A}$ that outputs a 5-tuple (c, m_1, o_1, m_2, o_2) the advantage

  $$\mathrm{BIND}\mathsf{adv}[\mathcal{A}, \mathcal{C}] := \Pr\big[m_1 \neq m_2 \;\text{ and }\; V(m_1, c, o_1) = V(m_2, c, o_2) = \mathsf{accept}\big]$$

  is negligible.

- **Hiding:** The commitment string c should reveal no information about the committed message $m \in \mathcal{M}$. We capture this using a semantic security definition. Specifically, we define two experiments, Experiment 0 and Experiment 1, between an adversary $\mathcal{A}$ and a challenger. For $b = 0, 1$, in Experiment b adversary $\mathcal{A}$ begins by sending $m_0, m_1 \in \mathcal{M}$ to the challenger, who computes $(c, o) \xleftarrow{R} C(m_b)$ and sends c to $\mathcal{A}$; finally, $\mathcal{A}$ outputs a guess $\hat{b} \in \{0, 1\}$. For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We require that for all efficient adversaries $\mathcal{A}$ the advantage

  $$\mathrm{HID}\mathsf{adv}[\mathcal{A}, \mathcal{C}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|$$

  is negligible.

Definition 8.8. *A commitment scheme $\mathcal{C} = (C, V)$ is secure if it is both hiding and binding.*

Remark 8.1 (Encryption can be non-binding). One might be tempted to implement a commitment scheme using encryption. Let (E, D) be a semantically secure cipher with key space $\mathcal{K}$

and message space $\mathcal{M}$. The derived commitment scheme (C, V) works as follows: $C(m)$ chooses a random key $k \xleftarrow{\text{R}} \mathcal{K}$, computes $c \xleftarrow{\text{R}} E(k, m)$, and outputs (c, k) as the commitment and opening strings. Algorithm $V(m, c, k)$ accepts if $D(k, c) = m$.

At first glance, this may seem like a fine commitment scheme. However, this construction may be completely insecure, and this is a common source of real-world implementation errors. Let's see why. The commitment scheme is clearly hiding, because the cipher is semantically secure, so this is not where the problem lies. However, the scheme may not be binding, and this breaks security. The problem is that it may be feasible to find a commitment string c and two keys k_1 and k_2 such that $D(k_1, c) \neq D(k_2, c)$ and both decryptions are in $\mathcal{M}$. This lets an attacker open the commitment c to two different messages, which breaks the binding property. This shows that encryption and commitments, while related, are quite different objects.

For example, suppose $\mathcal{M} = \mathcal{K} = \{0, 1\}^n$ and (E, D) is the one-time pad. Then the commitment to a message $m \in \mathcal{M}$ is simply $c = m \oplus k$, where $k \xleftarrow{\text{R}} \mathcal{K}$. Now, it is quite easy for the committer to open this commitment c as any messages m' in $\mathcal{M}$ of its choice. Simply compute $k' = c \oplus m'$ and send (m', k') to the verifier, who will incorrectly accept this opening. We say that the one-time pad is a **non-binding encryption scheme** because the derived commitment scheme is non-binding. Many other encryption schemes can also be shown to be non-binding. □

A simple auction. Coming back to the auction question, let's see how to use a secure commitment scheme $\mathcal{C} = (C, V)$ to implement a simple verifiable sealed bid auction that does not require trusting the auction house. Every participant posts a commitment to his or her bid on a public bulletin board (say, hosted at the auction house web site). The commitment hiding property ensures that nothing is revealed about the participant's bid. The commitment binding property ensures that the participant can no longer change the bid. Once all the bids are posted, the auction house asks all the participants to open their commitments, and the winner is determined. All the openings are posted on the bulletin board, so that the participants can audit the auction. If a participant does not open his or her commitment by a certain deadline, then their bid is discarded.

Of course, the bulletin board needs to be authenticated, so that everyone can tell that the commitment came from the participant, and not from someone who is masquerading as the participant. This can be done using digital signatures, which is the topic of Chapter 13. We also need to ensure that once a message is posted on the bulletin board it cannot be removed, so that bids cannot be maliciously deleted.

One downside of this auction scheme is that it forces everyone, even non-winners, to publicly reveal their bids. This can be fixed by using a *private* sealed bid auction scheme, where bids remain secret even after the auction is finished. We will see how to construct such a scheme in Chapter 23, after we develop a few more tools.

As it turns out, the simple auction scheme described above may be insecure, even if we use a secure commitment scheme. Let's first use collision resistance to construct two secure commitment schemes, and then take another look at this auction at the end of the section.

A commitment scheme from collision resistance. In Section 3.12 we constructed an elegant commitment scheme from a pseudorandom generator. However, that commitment scheme is difficult to use in this and other applications, because it utilizes an *interactive* commitment protocol. We can do much better using a collision resistant hash function.

Let H be a hash function defined over $(\mathcal{X}, \mathcal{Y})$ where $\mathcal{X} = \mathcal{M} \times \mathcal{R}$. Here $\mathcal{M}$ is the message space for the commitment scheme, and $\mathcal{R}$ is a finite nonce space that will be used for the hiding property. For $m \in \mathcal{M}$, the derived commitment scheme $\mathcal{C}_H = (C, V)$ is defined as:

$$\begin{aligned} C(m) &:= \big\{ o \xleftarrow{\text{R}} \mathcal{R}, \quad c \leftarrow H(m, o), \quad \text{output}(c, o) \big\}, \\ V(m, c, o) &:= \big\{ \text{output } \mathsf{accept} \text{ if } c = H(m, o) \big\}. \end{aligned}$$

To argue that this is a secure commitment scheme we need the hash function H to satisfy two properties.

- First, to ensure that $\mathcal{C}_H$ is binding, we require that H is collision resistant. It is easy to see that that collision resistance is sufficient, as an efficient adversary $\mathcal{A}$ that defeats the binding property immediately gives a collision for H. Indeed, suppose $\mathcal{A}$ outputs two pairs (m_1, o_1) and (m_2, o_2), where $m_1 \neq m_2$, but $V(m_1, c, o_1) = V(m_2, c, o_2) = \mathsf{accept}$, for some commitment string c. Then $H(m_1, o_1) = c = H(m_2, o_2)$ is a collision for H.

  Because the binding property depends on a computational assumption, we say that $\mathcal{C}_H$ is ***computationally* binding**. We note that the commitment scheme presented in Section 3.12 is ***unconditionally* binding**, meaning that the binding property holds without any computational assumptions.

- Second, to ensure that $\mathcal{C}_H$ is binding, we require that H to satisfies a certain technical property, called **input hiding**, which means that for adversarially chosen $m_1, m_2 \in \mathcal{M}$, and for a random $o \xleftarrow{\text{R}} \mathcal{R}$, the distribution $\{H(m_1, o)\}$ is computationally indistinguishable from the distribution $\{H(m_2, o)\}$ (as in Definition 3.4). The input hiding property for H clearly implies that the commitment scheme $\mathcal{C}_H$ is hiding.

  Standard collision resistant hash functions are believed to be input hiding in a stronger sense, provided the set $\mathcal{R}$ is sufficiently large relative to the size of the output space $\mathcal{Y}$ of H (for example, taking $\mathcal{R} = \{0, 1\}^{512}$ should be sufficient for SHA256). Indeed, it is believed that such hash functions are ***statistically* input hiding**, meaning that for *all* $m_1, m_2 \in \mathcal{M}$ the distribution $\{H(m_1, o)\}$ is *statistically* indistinguishable from the distribution $\{H(m_2, o)\}$ (see Definition 3.6). In this case, we say that the commitment scheme is ***unconditionally* hiding**, meaning that the committed message remains well hidden without any computational assumptions.

  We note that the commitment scheme presented in Section 3.12 is ***computationally* hiding**, meaning that the hiding property holds only under a computational assumption.

A weakly homomorphic commitment scheme. Let's construct another commitment scheme, this time with an additional property. The message space for this commitment will be $\mathbb{Z}_q$, for some integer $q > 1$. We will need two hash functions: H_1 defined over $(\mathcal{R}, \mathcal{Y})$ and H_2 defined over $(\mathcal{R}, \mathbb{Z}_q)$. For $m \in \mathbb{Z}_q$, the derived commitment scheme $\mathcal{C}_{H_1, H_2} = (C, V)$ is defined as:

$$\begin{aligned} C(m) &:= \big\{ o \xleftarrow{\text{R}} \mathcal{R}, \quad c_1 \leftarrow H_1(o), \quad c_2 \leftarrow m + H_2(o), \quad \text{output}\big((c_1, c_2), o\big) \big\} \\ V\big(m, (c_1, c_2), o\big) &:= \big\{ \text{output } \mathsf{accept} \text{ if } c_1 = H(o) \text{ and } c_2 = m + H_2(o) \big\} \end{aligned}$$

As before, to argue that this is a secure commitment scheme we need the hash functions H_1 and H_2 to satisfy two properties.

- First, to ensure that $\mathcal{C}_{H_1,H_2}$ is binding, we require that H_1 is collision resistant. To see why this is sufficient, suppose that an efficient adversary $\mathcal{A}$ defeats the binding property. Then $\mathcal{A}$ must output two pairs (m_1, o_1) and (m_2, o_2), where $m_1 \neq m_2$, but $V(m_1, c, o_1) = V(m_2, c, o_2) = \mathsf{accept}$, for some commitment string $c = (c_1, c_2)$. But then $H_1(o_1) = c_1 = H_1(o_2)$. For this to be a collision, we need to argue that $o_1 \neq o_2$, but this must be true because $m_1 + H_2(o_1) = c_2 = m_2 + H_2(o_2)$. Hence, if $o_1 = o_2$ then $m_1 = m_2$, which is not the case.

- Second, for the hiding property we again need (H_1, H_2) to satisfy a specific property. We say that (H_1, H_2) are **compatible** if for $o \xleftarrow{R} \mathcal{R}$ and $r \xleftarrow{R} \mathbb{Z}_q$, the distribution $\big(H_1(o), H_2(o)\big)$ is computationally indistinguishable from the distribution $\big(H_1(o), r\big)$. When (H_1, H_2) are compatible, then the message $m \in \mathbb{Z}_q$ is effectively hidden using a one-time pad, and therefore the scheme is hiding.

  When $q < 2^{128}$ and $\mathcal{R} = \{0,1\}^{768}$, it is believed that the functions $H_1(x) := \text{SHA256}(1 \parallel x)$ and $H_2(x) := \text{SHA256}(2 \parallel x) \bmod q$ are ***unconditionally*** **compatible**, meaning that the compatibility property holds without any computatitional assumption.

This new secure commitment scheme has an interesting property: let $c = (c_1, c_2)$ be a commitment for a message $m \in \mathbb{Z}_q$, and let $\delta \in \mathbb{Z}_q$. Then $c' := (c_1, c_2 + \delta)$ is a commitment for the message $(m+\delta) \in \mathbb{Z}_q$. In other words, a commitment for m can be transformed into a commitment for $(m+\delta)$ by anyone and without any knowledge of m. A commitment scheme that has such a property is said to be a **weakly homomorphic commitment scheme** or a **malleable commitment scheme**.

What is this good for? For some applications, a homomorphic property is a blessing, for others it is a curse. For now, let's see a negative application.

Consider again the auction scheme from the beginning of the section. In that scheme, the homomorphic property leads to a complete break of the system, even when the commitment scheme is otherwise secure. Let's see the attack.

Suppose Alice submits a commitment c for her bid $b \in \mathbb{Z}_q$ for the vase. Bob really wants the vase, and is willing to outspend Alice by a small amount. Unfortunately for Bob, he does not know what Alice bid. But Bob has a way to cheat. Once Alice's commitment c is posted on the public bulletin board, Bob can use the homomorphic property to create a new commitment c' for the value $b + 1 \in \mathbb{Z}_q$. This ensures that Bob will beat Alice in the auction (unless $b = q - 1$, but we can assume that q is sufficiently large so that this will not happen). Once Alice posts the opening for her commitment c, Bob has what he needs to post the opening for his commitment c'.

This attack is possible even though the commitment scheme $\mathcal{C}_{H_1,H_2}$ is secure according to Definition 8.8. A potential defense is to require that no two openings have the same opening string o, but the auction house needs to be aware of this attack to know to check for this.

To avoid this problem altogether, the auction house needs a commitment scheme with a stronger security property. The required property is called **non-malleability**, and is defined and studied in [56]. Interestingly, the first commitment scheme $\mathcal{C}_H$, can be shown to be non-malleable, when H is modeled as a random oracle, and its range $\mathcal{Y}$ is sufficiently large.

8.13 Notes

Citations to the literature to be added.

8.14 Exercises

8.1 (Truncating a CRHF is dangerous). Let H be a collision resistant hash function defined over $(\mathcal{M}, \{0,1\}^n)$. Use H to construct a hash function H' over $(\mathcal{M}, \{0,1\}^n)$ that is also collision resistant, but if one truncates the output of H' by one bit then H' is no longer collision resistant. That is, H' is collision resistant, but $H''(x) := H'(x)[0 \mathinner{..} n-2]$ is not.

8.2 (CRHF combiners). We want to build a CRHF H using two CRHFs H_1 and H_2, so that if at some future time one of H_1 or H_2 is broken (but not both) then H is still secure.

(a) Suppose H_1 and H_2 are defined over $(\mathcal{M}, \mathcal{T})$. Let $H(m) := \big(H_1(m), H_2(m)\big)$. Show that H is a secure CRHF if either H_1 or H_2 is secure.

(b) Show that $H'(x) = H_1(H_2(x))$ need not be a secure CRHF even if one of H_1 or H_2 is secure.

8.3 (Broken collision resistance). For $i = 1, \ldots, n$ let H_i be a function defined over $(\mathcal{M}, \mathcal{T})$ where $\mathcal{T} := \{0,1\}^b$. Define $H^{(n)}$ as the hash function

$$H^{(n)}(m_1, \ldots, m_n) := H_1(m_1) \oplus \cdots \oplus H_n(m_n)$$

defined over $(\mathcal{M}^n, \mathcal{T})$. Suppose that $H_1, \ldots, H_n$ are independent random oracles.

(a) Show that $H^{(2)}$ is collision resistant if 2^b is super-poly. In particular, show that $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H^{(2)}]$ is at most $Q^4/2^b$ for every adversary $\mathcal{A}$ that makes at most Q queries to each of H_1 and H_2.

(b) Show that $H^{(n)}$ is not collision resistant whenever $n \geq b/2$. In particular, there is an efficient adversary $\mathcal{A}$ such that $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H^{(n)}] \geq 1/4$, where $\mathcal{A}$ makes 4 queries to each of the functions $H_1, \ldots, H_n$.
Hint: use the fact that a random $2n \times 2n$ matrix over $\mathbb{Z}_2$ is full rank with probability at least 1/4.

(c) Show that $H^{(n)}$ is not one-way whenever $n \geq b$.

Discussion: In Chapter 17 we will see how to make this construction collision resistant even when $n \geq b$. We will also explain the reason for the interest in this construction.

8.4 (Extending the domain of a PRF with a CRHF). Suppose F is a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ and H is a collision resistant hash defined over $(\mathcal{M}, \mathcal{X})$. Show that $F'(k, m) = F(k, H(m))$ is a secure PRF. This shows that H can be used to extend the domain of a PRF.

8.5 (Hash-then-encrypt MAC). Let H be a collision resistant hash defined over $(\mathcal{M}, \mathcal{X})$ and let $\mathcal{E} = (E, D)$ be a secure block cipher defined over $(\mathcal{K}, \mathcal{X})$. Show that the encrypted-hash MAC system (S, V) defined by $S(k, m) := E(k, H(m))$ is a secure MAC.

Hint: Use Theorem 8.1.

8.6 (Finding many collisions). Let H be a hash function defined over $(\mathcal{M}, \mathcal{T})$ where $N := |\mathcal{T}|$ and $|\mathcal{M}| \gg N$. We showed that $O(\sqrt{N})$ evaluations of H are sufficient to find a collision for H with probability 1/2. Show that $O\left(\sqrt{sN}\right)$ evaluations of H are sufficient to find s collisions $(x_0^{(1)}, x_1^{(1)}), \ldots, (x_0^{(s)}, x_1^{(s)})$ for H with probability at least 1/2. Therefore, finding a million collisions is only about a thousand times harder than finding a single collision.

8.7 (Finding multi-collisions). Continuing with Exercise 8.6, we say that an s-collision for H is a set of s distinct points $x_1, \ldots, x_s$ in $\mathcal{M}$ such that $H(x_1) = \cdots = H(x_s)$. Show that for each constant value of s, $O\left(N^{(s-1)/s}\right)$ evaluations of H are sufficient to find an s-collision for H, with probability at least 1/2.

8.8 (Collision finding in constant space). Let H be a hash function defined over $(\mathcal{M}, \mathcal{T})$ where $N := |\mathcal{M}|$. In Section 8.3 we developed a method to find an H collision with constant probability using $O(\sqrt{N})$ evaluations of H. However, the method required $O(\sqrt{N})$ memory space. In this exercise we develop a *constant-memory* collision finding method that runs in about the same time. More precisely, the method only needs memory to store two hash values in $\mathcal{T}$. You may assume that $H : \mathcal{M} \to \mathcal{T}$ is a random function chosen uniformly from Funs$[\mathcal{M}, \mathcal{T}]$ and $\mathcal{T} \subseteq \mathcal{M}$. A collision should be produced with probability at least 1/2.

(a) Let $x_0 \xleftarrow{R} \mathcal{M}$ and define $H^{(i)}(x_0)$ to be the ith iterate of H starting at x_0. For example, $H^{(3)}(x_0) = H(H(H(x_0)))$.

  (i) Let i be the smallest positive integer satisfying $H^{(i)}(x_0) = H^{(2i)}(x_0)$.

  (ii) Let j be the smallest positive integer satisfying $H^{(j)}(x_0) = H^{(j+i)}(x_0)$. Notice that $j \leq i$.

  Show that $H^{(j-1)}(x_0)$ and $H^{(j+i-1)}(x_0)$ are an H collision with probability at least 3/4.

(b) Show that i from part (a) satisfies $i = O(\sqrt{N})$ with probability at least 3/4 and that it can be found using $O(\sqrt{N})$ evaluations of H. Once i is found, finding j takes another $O(\sqrt{N})$ evaluations, as required. The entire process only needs to store two elements in T at any given time.

8.9 (A parallel Merkle-Damgård). The Merkle-Damgård construction in Section 8.4 gives a *sequential* method for extending the domain of a secure CRHF. The tree construction in Fig. 8.16 is a parallelizable approach: all the hash functions h within a single level can be computed in parallel. Prove that the resulting hash function defined over $(\mathcal{X}^{\leq L},\ \mathcal{X})$ is collision resistant, assuming h is collision resistant. Here h is a compression function $h : \mathcal{X}^2 \to \mathcal{X}$, and we assume the message length can be encoded as an element of $\mathcal{X}$. More precisely, the hash function is defined as follows:

```
input: $m_1 \ldots m_s \in \mathcal{X}^s$ for some $1 \leq s \leq L$
output: $y \in \mathcal{X}$

let $t \in \mathbb{Z}$ be the smallest power of two such that $t \geq s$    (i.e., $t := 2^{\lceil \log_2 s \rceil}$)
for $i = s+1$ to $t$:    $m_i \leftarrow \perp$
for $i = t+1$ to $2t-1$:
    $\ell \leftarrow 2(i-t)-1,\quad r \leftarrow \ell + 1$        // indices of left and right children
    if $m_\ell = \perp$ and $m_r = \perp$:    $m_i \leftarrow \perp$    // if node has no children, set node to null
    else if $m_r = \perp$:    $m_i \leftarrow m_\ell$          // if one child, propagate child as is
    else $m_i \leftarrow h(m_\ell, m_r)$                // if two children, hash with h
output $y \leftarrow h\big(m_{2t-1},\ s\big)$                 // hash final output and message length
```

8.10 (Secure variants of Davies-Meyer). Prove that the h_1, h_2, and h_3 variants of Davies-Meyer defined on page 301 are collision resistant in the ideal cipher model.

8.11 (Insecure variants of Davies-Meyer). Show that the h_4 and h_5 variants of Davies-Meyer defined on page 301 are not collision resistant.

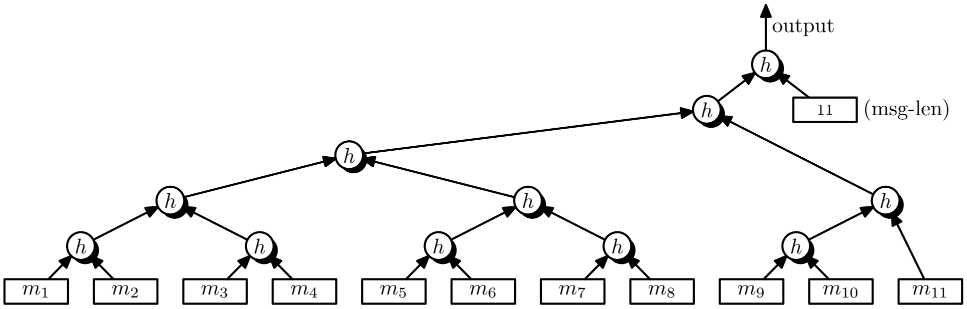


Figure 8.16: Tree-based Merkle-Damgård for a message of length $s = 11$ blocks

8.12 (An insecure instantiation of Davies-Meyer). Let's show that Davies-Meyer may not be collision resistant when instantiated with a real-world block cipher. Let (E, D) be a block cipher defined over $(\mathcal{K}, \mathcal{X})$ where $\mathcal{K} = \mathcal{X} = \{0,1\}^n$. For $y \in \mathcal{X}$ let $\overline{y}$ denote the bit-wise complement of y.

(a) Suppose that $E(\overline{k}, \overline{x}) = \overline{E(k, x)}$ for all keys $k \in \mathcal{K}$ and all $x \in \mathcal{X}$. The DES block cipher has precisely this property. Show that the Davies-Meyer construction, $h(k, x) := E(k, x) \oplus x$, is not collision resistant when instantiated with algorithm E.

(b) Suppose (E, D) is an Even-Mansour cipher, $E(k, x) := \pi(x \oplus k) \oplus k$, where $\pi : \mathcal{X} \to \mathcal{X}$ is a fixed public permutation. Show that the Davies-Meyer construction instantiated with algorithm E is not collision resistant.

Hint: Show that this Even-Mansour cipher satisfies the property from part (a).

8.13 (Merkle-Damgård without length encoding). Suppose that in the Merkle-Damgård construction, we drop the requirement that the padding block encodes the message length. Let h be the compression function, let H be the resulting hash function, and let IV be the prescribed initial value.

(a) Show that H is collision resistant, assuming h is collision resistant and that *it is hard to find a preimage of IV under h.*

(b) Show that if h is a Davies-Meyer compression function, and we model the underlying block cipher as an ideal cipher, then for any fixed IV, it is hard to find a preimage of IV under h.

8.14 (2nd-preimage resistance of Merkle-Damgård). Let H be a Merkle-Damgård hash built out of a Davies-Meyer compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. Consider the attack game characterizing 2nd-preimage resistance in Definition 8.6. Let us assume that the initial, random message in that attack game consists of s blocks. We shall model the underlying block cipher used in the Davies-Meyer construction as an ideal cipher, and adapt the attack game to work in the ideal cipher model. Show that for every adversary $\mathcal{A}$ that makes at most Q ideal-cipher queries, we have

$$\mathrm{SPR}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, H] \leq \frac{(Q+s)s}{2^{n-1}}.$$

Discussion: This bound for finding second preimages is significantly better than the bound for finding arbitrary collisions. Unfortunately, we have to resort to the ideal cipher model to prove it.

8.15 (Fixed points). We consider the Davies-Meyer and Miyaguchi-Preneel compression functions defined in Section 8.5.2.

(a) Show that for a Davies-Meyer compression function it is easy to find a pair (t, m) such that $h_{\text{DM}}(t, m) = t$. Such a pair is called a **fixed point** for h_{DM}.

(b) Show that in the ideal cipher model it is difficult to find fixed points for the Miyaguchi-Preneel compression function.

The next exercise gives an application for fixed points.

8.16 (Finding second preimages in Merkle-Damgård). In this exercise, we develop a second preimage attack on Merkle-Damgård that roughly matches the security bounds in Exercise 8.14. Let H_{MD} be a Merkle-Damgård hash built out of a Davies-Meyer compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. Recall that H_{MD} pads a given message with a padding block that encodes the message length. We will also consider the hash function H, which is the same as H_{MD}, but which uses a padding block that *does not* encode the message length. Throughout this exercise, we model the underlying block cipher in the Davies-Meyer construction as an ideal cipher. For concreteness, assume $\ell = 2n$.

(a) Let $s \approx 2^{n/2}$. You are given a message M that consists of s random ℓ-bit blocks. Show that by making $O(s)$ ideal cipher queries, with probability 1/2 you can find a message $M' \neq M$ such that $H(M') = H(M)$. Here, the probability is over the random choice of M, the random permutations defining the ideal cipher, and the random choices made by your attack.

Hint: Repeatedly choose random blocks x in $\{0,1\}^\ell$ until $h(\text{IV}, x)$ is the same as one of the s chaining variables obtained when computing $H(M)$. Use this x to construct the second preimage M'.

(b) Repeat part (a) for H_{MD}.

Hint: The attack in part (a) will likely find a second preimage M' that is shorter than M; because of length encoding, this will not be a second preimage under H_{MD}; nevertheless, show how to use fixed points (see previous exercise) to modify M' so that it has the same length as M.

Discussion: Let H be a hash function with an n-bit output. If H is a *random* function then breaking 2nd-preimage resistance takes about 2^n time. This exercise shows that for Merkle-Damgård functions, breaking 2nd-preimage resistance can be done much faster, taking only about $2^{n/2}$ time.

8.17 (The envelope method is a secure PRF). Consider the envelope method for building a PRF from a hash function discussed in Section 8.7: $F_{\text{env}}(k, M) := H(k \parallel M \parallel k)$. Here, we assume that H is a Merkle-Damgård hash built from a compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. Assume that the keys for F_{env} are ℓ-bit strings. Furthermore, assume that the message M is a bit string whose length is an even multiple of ℓ (we can always pad the message, if necessary). Under the assumption that both h_{top} and h_{bot} are secure PRFs, show that F_{env} is a secure PRF.

Hint: Use the result of Exercise 7.6; also, first consider a simplified setting where H does not append the usual Merkle-Damgård padding block to the inputs $k \parallel M \parallel k$ (this padding block does not really help in this setting, but it does not hurt either — it just complicates the analysis).

8.18 (The key-prepending method revisited). Consider the key-prepending method for building a PRF from a hash function discussed in Section 8.7: $F_{\text{pre}}(k, M) := H(k \parallel M)$. Here, we assume that H is a Merkle-Damgård hash built from a compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. Assume that the keys for F_{pre} are ℓ-bit strings. Under the assumption that both h_{top} and h_{bot} are secure PRFs, show that F_{pre} is a prefix-free secure PRF.

8.19 (The key-appending method revisited). Consider the following variant of the key-appending method for building a PRF from a hash function discussed in Section 8.7: $F'_{\text{post}}(k, M) := H(M \parallel \text{PB} \parallel k)$. Here, we assume that H is a Merkle-Damgård hash built from a compression function $h : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^n$. Also, PB is the standard Merkle-Damgård padding for M, which encodes the length of M. Assume that the keys for F'_{post} are ℓ-bit strings. Under the assumption that h is collision resistant and h_{top} is a secure PRF, show that F'_{post} is a secure PRF.

8.20 (Dual PRFs). The security analysis of HMAC assumes that the underlying compression function is a secure PRF when either input is used as the key. A PRF with this property is said to be a **dual PRF**. Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ where $\mathcal{Y} = \{0,1\}^n$ for some n. We wish to build a new PRF $\hat{F}$ that is a dual PRF. This $\hat{F}$ can be used as a building block for HMAC.

(a) Suppose $\mathcal{K} = \mathcal{X}$. Show that the most natural construction $\hat{F}(x, y) := F(x, y) \oplus F(y, x)$ is insecure: there exists a secure PRF F for which $\hat{F}$ is not a dual PRF.

Hint: Start from a secure PRF F' and then "sabotage" it to get the required F.

(b) Let G be a PRG defined over $(\mathcal{S},\ \mathcal{K} \times \mathcal{X})$. Let $G_0 : \mathcal{S} \to \mathcal{K}$ be the left output of G and let $G_1 : \mathcal{S} \to \mathcal{X}$ be the right output of G. Let $\hat{F}$ be the following PRF defined over $(\mathcal{S}, \mathcal{S}, \mathcal{Y})$:

$$\hat{F}(x, y) := F\Big(G_0(x),\ G_1(y)\Big)\ \oplus\ F\Big(G_0(y),\ G_1(x)\Big).$$

Prove that $\hat{F}$ is a dual PRF assuming G is a secure PRG and that G_1 is collision resistant.

8.21 (Sponge with low capacity is insecure). Let H be a sponge hash with rate r and capacity c, built from a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$, where $n = r + c$ (see Section 8.8). Assume $r \geq 2c$. Show how to find a collision for H with probability at least $1/2$ in time $O(2^{c/2})$. The colliding messages can be $2r$ bits each.

8.22 (Sponge as a PRF). Let H be a sponge hash with rate r and capacity c, built from a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$, where $n = r + c$ (see Section 8.8). Consider again the PRF built from H by pre-pending the key: $F_{\text{pre}}(k, M) := H(k \parallel M)$. Assume that the key is r bits and the output of F_{pre} is also r bits. Prove that in the ideal permutation model, where π is replaced by a random permutation Π, this construction yields a secure PRF, assuming 2^r and 2^c are super-poly.

Note: This follows immediately from the fact that H is indifferentiable from a random oracle (see Section 8.10.3) and Theorem 8.9. However, you are to give a *direct proof* of this fact.

Hint: Use the same domain splitting strategy as outlined in Exercise 7.17.

8.23 (Relations among hash function properties). In this exercise we explore the relation between 2nd-preimage resistance and one-wayness. Let H be a hash function defined over $(\mathcal{M}, \mathcal{T})$ where $|\mathcal{M}| \geq s \cdot |\mathcal{T}|$ for some $s > 1$.

(a) We say that $m \in \mathcal{M}$ has a second preimage if there exists an $m' \neq m$ in $\mathcal{M}$ such that $H(m') = H(m)$. Show that at most $1/s$ of the elements in $\mathcal{M}$ do not have a second preimage.

(b) Suppose s is super-poly. Show that if H is 2nd-preimage resistant then it must also be one-way. In particular, for every one-way adversary $\mathcal{A}$ there is a 2nd-preimage adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\text{OW}\mathsf{adv}[\mathcal{A}, H] \leq 2 \cdot \text{SPR}\mathsf{adv}[\mathcal{B}, H] + (1/s). \tag{8.22}$$

Hint: $\mathcal{B}$ is given a random $m \in \mathcal{M}$ and asks $\mathcal{A}$ to invert $H(m)$. Use part (a) to argue that this $\mathcal{B}$ satisfies (8.22).

(c) Show that part (b) may be false when $s > 1$ is poly-bounded, say when $s = 2$. Let H be a hash function defined over $(\mathcal{M}, \mathcal{T})$ where $\mathcal{M} := \mathcal{T} \times \{1, 2, \ldots, 2s\}$. Suppose that H is 2nd-preimage resistant. Let $\mathcal{T}' := \mathcal{T} \times \{0, 1\}$. Construct an H' defined over $(\mathcal{M}, \mathcal{T}')$ that is 2nd-preimage resistant, but not one-way. This H' is a counter-example to part (b) when s is poly-bounded. Note that $|\mathcal{M}| = s \cdot |\mathcal{T}'|$, so that H' has compression ratio s, as required. First construct H' when $s = 2$ and then generalize.

(d) Show that a collision resistant hash must be 2nd-preimage resistant, with no restriction on s.

8.24 (From TCR to 2nd-preimage resistance). Let H be a TCR hash defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. Choose a random $r \in \mathcal{M}$. Prove that $f_r(x) := H(r, x)$ is 2nd-preimage resistant, where r is treated as a system parameter.

8.25 (File integrity: reducing read-only memory). The file integrity construction in Section 8.11.4 uses additional read-only memory proportional to $\log |F|$ where $|F|$ is the size of the file F being protected.

(a) By first hashing the file F and then hashing the key r, show how to reduce the amount of additional read-only memory used to $O(\log \log |F|)$. This requires storing additional $O(\log |F|)$ bits on disk.

(b) Generalize your solution from part (a) to show how to reduce read-only overhead to constant size independent of $|F|$. The extra information stored on disk is still of size $O(\log |F|)$.

8.26 (Strong 2nd-preimage resistance). Let H be a hash function defined over $(\mathcal{X} \times \mathcal{Y},\ \mathcal{T})$ where $\mathcal{X} := \{0, 1\}^n$. We say that H is **strong 2nd-preimage resistant**, or simply **strong SPR**, if no efficient adversary, given a random x in $\mathcal{X}$ as input, can output y, x', y' such that $H(x, y) = H(x', y')$ with non-negligible probability.

(a) Show that $H_{\text{TCR}}(k, (x, y)) := H(k \oplus x,\ y)$ is a TCR hash function assuming H is a strong SPR hash function. If $\mathcal{X}$ is relatively small and $\mathcal{Y}$ is much larger, we obtain a TCR for long messages, and with short keys, that is a lot simpler than the constrution in Section 8.11.3.

(b) Let H be a strong SPR. Use H to construct a collision resistant hash function H' defined over $(\mathcal{Y},\ \mathcal{T})$.

Discussion: This result shows that when $\mathcal{Y}$ is much bigger than $\mathcal{T}$, the range $\mathcal{T}$ of a strong SPR must be as big as the range of a collision resistant hash function. This was not the case for an SPR, whose range can be smaller than that of a collision resistant function, while providing the same level of security.

(c) Let us show that a function H can be a strong SPR, but not collision resistant. For example, consider the hash function:

$$H''(0,0) := H''(0,1) := 0 \quad \text{and} \quad H''(x,y) := H(x,y) \text{ for all other inputs.}$$

Prove that if $|\mathcal{X}|$ is super-poly and H is a strong SPR then so is H''. However, H'' is clearly not collision resistant.

8.27 (Enhanced TCR). Let H be a keyed hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. We say that H is an **enhanced TCR** if no efficient adversary $\mathcal{A}$ can win the following game with non-negligible advantage: the adversary outputs $m \in \mathcal{M}$, is given random $k \in \mathcal{K}$, and outputs (k', m') such that $H(k,m) = H(k',m')$, where $(k,m) \neq (k',m')$. As usual, let $\text{eTCR}\mathsf{adv}[\mathcal{A}, H]$ denote $\mathcal{A}$'s advantage against H.

(a) Show how to use an enhanced TCR to extend the domain of a MAC. Let H be a enhanced TCR defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{X})$ and let (S, V) be a secure MAC defined over $(\mathcal{K}, \mathcal{X}, \mathcal{T})$. Show that the following is a secure MAC for messages in $\mathcal{M}$:

$$\begin{aligned} S'(k,m) &:= \{\ r \xleftarrow{\text{R}} \mathcal{K}_H,\ h \leftarrow H(r,m),\ t \leftarrow S(k,h),\ \text{output } (r,t)\ \} \\ V'\big(k,m,(r,t)\big) &:= V\big(k, H(r,m), t\big) \end{aligned}$$

Discussion: The small domain MAC (S, V) in this construction is only given h as the input message, where as when using a TCR, the small domain MAC was given (r, h) as the message. Hence, the message space of the small domain MAC can be much smaller when using an enhanced TCR.

(b) Let H be a hash function defined over $(\mathcal{K} \times \mathcal{M},\ \mathcal{T})$. Show that modeling H as a random oracle makes H an enhanced TCR defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, assuming $|\mathcal{K}|$ and $|\mathcal{T}|$ are super-poly. Specifically, for every adversary $\mathcal{A}$ that makes at most Q_{ro} queries to H, we have

$$\text{eTCR}^{(\text{ro})}\mathsf{adv}[\mathcal{A}, H] \leq \frac{Q_{\text{ro}}^2}{2|\mathcal{T}| \cdot |\mathcal{K}|} + \frac{Q_{\text{ro}}}{|\mathcal{T}|}.$$

Discussion: When $|\mathcal{K}| = |\mathcal{T}|$ this bound is less than $2Q_{\text{ro}}/|\mathcal{T}|$. This shows that there is no generic birthday attack on an enhanced TCR. Consequently, the small domain MAC (S, V) can operate on shorter messages than needed in the MAC extension construction from collision resistance, discussed in Section 8.2. This fact will be quite useful in Chapter 14.

(c) Let H be a strong SPR hash function over $(\mathcal{X} \times \mathcal{Y}, \mathcal{T})$, as defined in Exercise 8.26, where $\mathcal{X} := \{0,1\}^n$. Show that $H'(k, (x,y)) := H(k \oplus x,\ y)$ is an enhanced TCR function.

Discussion: Other constructions for enhanced TCR functions can be found in [82].

(d) Let H be a TCR defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. Show that $H'(k,m) := (H(k,m),\ k)$ is an enhanced TCR defined over $(\mathcal{K},\ \mathcal{M},\ \mathcal{T} \times \mathcal{K})$.

8.28 (Even-Mansour is not an enhanced TCR). Let $\mathcal{X} := \{0,1\}^n$ and let $\pi : \mathcal{X} \to \mathcal{X}$ be a random permutation.

- Consider the Even-Mansour function $H_1(k,m) := \pi(m \oplus k) \oplus k$ defined over $(\mathcal{X}, \mathcal{X}, \mathcal{X})$. This function is trivially a TCR because for every fixed $k \in \mathcal{X}$ it defines a permutation on $\mathcal{X}$. Show that it is not an enhanced TCR as defined in Exercise 8.27.

- Show that the function $H_2(k,m) := \pi(k \oplus m) \oplus m$ is a TCR defined over $(\mathcal{X},\mathcal{X},\mathcal{X})$.

8.29 (Weak collision resistance). Let H be a keyed hash function defined over $(\mathcal{K},\mathcal{M},\mathcal{T})$. We say that H is a **weak collision resistant (WCR)** if no efficient adversary can win the following game with non-negligible advantage: the challenger chooses a random key $k \in \mathcal{K}$ and lets the adversary query the function $H(k,\cdot)$ at any input of its choice. The adversary wins if it outputs a collision m_0, m_1 for $H(k,\cdot)$.

(a) Show that WCR is a weaker notion than a secure MAC: (1) show that every deterministic secure MAC is WCR, (2) give an example of a secure WCR that is not a secure MAC.

(b) MAC domain extension with a WCR: let (S,V) be a secure MAC and let H be a WCR. Show that the MAC system (S',V') defined by $S'\big((k_0,k_1),m\big) := S\big(k_1, H(k_0,m)\big)$ is secure.

(c) Show that Merkle-Damgård expands a compressing fixed-input length WCR to a variable input length WCR. In particular, let h be a WCR defined over $(\mathcal{K},\mathcal{X}\times\mathcal{Y},\mathcal{X})$, where $\mathcal{X} := \{0,1\}^n$ and $\mathcal{Y} := \{0,1\}^\ell$. Define H as a keyed hash function over $(\mathcal{K},\ \{0,1\}^{\le L},\ \mathcal{X})$ as follows:

$$H\big((k_1,k_2),M\big) := \left\{\begin{array}{l} \text{pad and break } M \text{ into } \ell\text{-bit blocks: } m_1,\ldots,m_s \\ t_0 \leftarrow 0^n \in \mathcal{X} \\ \text{for } i = 1 \text{ to } s \text{ do:} \\ \quad t_i \leftarrow h\big(k_1,\ (t_{i-1},m_i)\big) \\ \text{encode } s \text{ as a block } b \in \mathcal{Y} \\ t_{s+1} \leftarrow h\big(k_2,\ (t_s,b)\big) \\ \text{output } t_{s+1} \end{array}\right\}$$

Show that H is a WCR if h is.

8.30 (A simple random oracle PRF). Let $\mathcal{K} = \mathcal{X} = \{0,1\}^n$ and let H be a hash function defined over $(\mathcal{X},\mathcal{Y})$ for some $\mathcal{Y}$. Let F_{xor} be the PRF $F_{\text{xor}}(k,x) := H(k \oplus x)$ defined over $(\mathcal{K},\mathcal{X},\mathcal{Y})$.

(a) Show that if $\mathcal{K}$ is large then $F_{\text{xor}}(k,x)$ is a secure PRF when H is modeled as a random oracle. In particular, if $\mathcal{A}$ makes at most Q_{F} PRF queries and Q_{ro} random oracle queries, then $\text{PRF}^{\text{ro}}\mathsf{adv}[\mathcal{A},F_{\text{xor}}] \le Q_{\text{ro}} \cdot Q_{\text{F}}/|\mathcal{K}|$.

(b) Show an attack on F_{xor} that has advantage at least 1/2 and makes at most $2^{n/2}$ PRF queries and $2^{n/2}$ random oracle queries. You may also assume that $|\mathcal{Y}| \gg 2^n$.

8.31 (The trouble with random oracles). Let H be a hash function defined over $(\mathcal{K}\times\mathcal{X},\ \mathcal{Y})$. We showed that $H(k,x)$ is a secure PRF when H is modeled as a random oracle. In this exercise we show that this PRF can be tweaked into a new PRF F that uses H as a black-box, and that is a secure PRF when H is modeled as a random model. However, for every concrete instantiation of the hash function H, the PRF F becomes insecure.

For simplicity, assume that $\mathcal{K}$ and $\mathcal{Y}$ consist of bit strings of length n and that $\mathcal{X}$ consists of bit strings of length at most L for some poly-bounded n and L. Assume also that the program for H parses its input as a bit string of the form $k \parallel x$, where $k \in \mathcal{K}$ and $x \in \mathcal{X}$.

Consider a program $\mathit{Exec}(P,v,t)$ that takes as input three bit strings P,v,t. When $\mathit{Exec}(P,v,t)$ runs, it attempts to interpret P as a program written in some programming language (take your pick); it runs P on input v, but stops the execution after $|t|$ steps (if necessary), where $|t|$ is the bit-length of t. The output of $\mathit{Exec}(P, v, t)$ is whatever P outputs on input v, or some special default value if the time bound is exceeded. For simplicity, assume that $\mathit{Exec}(P, v, t)$ always outputs an n-bit string (padding or truncating as necessary). Even though P on input v may run in exponential time (or even fall into an infinite loop), $\mathit{Exec}(P, v, t)$ always runs in time bounded by a polynomial in its input length.

Finally, let T be some arbitrary polynomial, and define

$$F(k, x) := H(k, x) \oplus \mathit{Exec}(x,\ k \parallel x,\ 0^{T(|k|+|x|)}).$$

(a) Show that if H is any hash function that can be implemented by a program P_H whose length is at most L and whose running time on input $k \parallel x$ is at most $T(|k| + |x|)$, then the concrete instantiation of F using this H runs in polynomial time and is not a secure PRF.

Hint: Find a value of x that makes the PRF output 0^n, for all keys $k \in \mathcal{K}$.

(b) Show that F is a secure PRF if H is modeled as a random oracle.

Discussion: Although this is a contrived example, it shakes our confidence in the random oracle model. Nevertheless, the reason why the random oracle model has been so successful in practice is that typically real-world attacks treat the hash function as a black box. The attack on F clearly does not. See also the discussion in [40], which removes the strict time bound restriction on H.

Chapter 9

Authenticated Encryption

This chapter is the culmination of our symmetric encryption story. Here we construct systems that ensure both data secrecy (confidentiality) and data integrity, even against very aggressive attackers that can interact maliciously with both the sender and the receiver. Such systems are said to provide **authenticated encryption** or are simply said to be AE-secure. This chapter concludes our discussion of symmetric encryption, and shows how to correctly do secure encryption in the real-world.

Recall that in our discussion of CPA security in Chapter 5 we stressed that CPA security does not provide any integrity. An attacker can tamper with the output of a CPA-secure cipher without being detected by the decryptor. We will present many real-world settings where undetected ciphertext tampering compromises both message secrecy and message integrity. Consequently, CPA security by itself is insufficient for almost all applications. Instead, applications should almost always use authenticated encryption to ensure both message secrecy and integrity. We stress that even if secrecy is the only requirement, CPA security is insufficient.

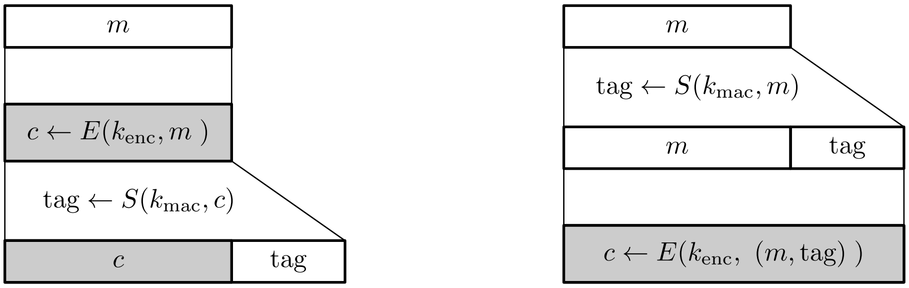
In this chapter we develop the notion of authenticated encryption and construct several AE systems. There are two general paradigms for constructing AE systems. The first, called **generic composition**, is to combine a CPA-secure cipher with a secure MAC. There are many ways to combine these two primitives and not all combinations are secure. We briefly consider two examples.

Let (E, D) be a cipher and (S, V) be a MAC. Let k_{enc} be a cipher key and k_{mac} be a MAC key. Two options for combining encryption and integrity immediately come to mind, which are shown in Fig. 9.1 and work as follows:

Encrypt-then-MAC Encrypt the message, $c \xleftarrow{\text{R}} E(k_{\text{enc}}, m)$, then MAC the ciphertext, tag $\xleftarrow{\text{R}} S(k_{\text{mac}}, c)$; the result is the ciphertext-tag pair (c, tag). This method is supported in the TLS 1.2 protocol and later versions as well as in the IPsec protocol and in a widely-used NIST standard called GCM (see Section 9.7).

MAC-then-encrypt MAC the message, tag $\xleftarrow{\text{R}} S(k_{\text{mac}}, m)$, then encrypt the message-tag pair, $c \xleftarrow{\text{R}} E\big(k_{\text{enc}},\ (m, t)\big)$; the result is the ciphertext c. This method is used in older versions of TLS (e.g., SSL 3.0 and its successor called TLS 1.0) and in the 802.11i WiFi encryption protocol.

As it turns out, only the first method is secure for every combination of CPA-secure cipher and secure MAC. The intuition is that the MAC on the ciphertext prevents any tampering with the ciphertext. We will show that the second method can be insecure — the MAC and cipher can



encrypt-then-mac

mac-then-encrypt

Figure 9.1: Two methods to combine encryption and MAC

interact badly and cause the resulting system to not be AE-secure. This has led to many attacks on widely deployed systems.

The second paradigm for constructing authenticated encryption is to build it directly from a block cipher or a PRF, without first constructing a standalone cipher or a MAC. These are called **integrated schemes**. The OCB encryption mode is the primary example in this category (see Exercise 9.17). Other examples include IAPM, XCBC, and CCFB.

Authenticated encryption standards. Cryptographic libraries such as OpenSSL often provide an interface for CPA-secure encryption (such as counter mode with a random IV) and a separate interface for computing MACs on messages. In the past, it was up to developers to correctly combine these two primitives to provide authenticated encryption. Every system did it differently and not all incarnations used in practice were secure.

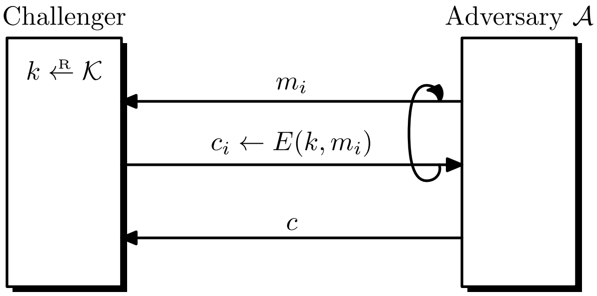
More recently, several standards have emerged for secure authenticated encryption. A popular method called Galois Counter Mode (GCM) uses encrypt-then-MAC to combine random counter mode encryption with a Carter-Wegman MAC (see Section 9.7). We will examine the details of this construction and its security later on in the chapter. Developers are encouraged to use an authenticated encryption mode provided by the underlying cryptographic library and to not implement it themselves.

9.1 Authenticated encryption: definitions

We start by defining what it means for a cipher $\mathcal{E}$ to provide authenticated encryption. It must satisfy two properties. First, $\mathcal{E}$ must be CPA-secure. Second, $\mathcal{E}$ must provide ciphertext integrity, as defined below. Ciphertext integrity is a new property that captures the fact that $\mathcal{E}$ should have properties similar to a MAC. Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. We define ciphertext integrity using the following attack game, shown in Fig. 9.2. The game is analogous to the MAC Attack Game 6.1.

Attack Game 9.1 (ciphertext integrity). For a given cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger chooses a random $k \overset{\text{R}}{\leftarrow} \mathcal{K}$.



Challenger
Adversary $\mathcal{A}$
$k \xleftarrow{\text{R}} \mathcal{K}$
m_i
$c_i \leftarrow E(k, m_i)$
c

Figure 9.2: Ciphertext integrity game (Attack Game 9.1)

- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots$, the ith query consists of a message $m_i \in \mathcal{M}$. The challenger computes $c_i \xleftarrow{\text{R}} E(k, m_i)$, and gives c_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate ciphertext $c \in \mathcal{C}$ that is not among the ciphertexts it was given, i.e.,
$$c \notin \{c_1, c_2, \ldots\}.$$

We say that $\mathcal{A}$ wins the game if c is a valid ciphertext under k, that is, $D(k, c) \neq \mathsf{reject}$. We define $\mathcal{A}$'s advantage with respect to $\mathcal{E}$, denoted $\mathrm{CI}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, as the probability that $\mathcal{A}$ wins the game. Finally, we say that $\mathcal{A}$ is a ***Q*-query adversary** if $\mathcal{A}$ issues at most Q encryption queries. □

Definition 9.1. *We say that a $\mathcal{E} = (E, D)$ provides* ***ciphertext integrity****, or* CI *for short, if for every efficient adversary $\mathcal{A}$, the value* $\mathrm{CI}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

CPA security and ciphertext integrity are the properties needed for authenticated encryption. This is captured in the following definition.

Definition 9.2. *We say that a cipher $\mathcal{E} = (E, D)$ provides* ***authenticated encryption****, or is simply* ***AE-secure****, if $\mathcal{E}$ is (1) semantically secure under a chosen plaintext attack, and (2) provides ciphertext integrity.*

Why is Definition 9.2 the right definition? In particular, why are we requiring *ciphertext* integrity, rather than some notion of *plaintext* integrity (which might seem more natural)? In Section 9.2, we will describe a very insidious class of attacks called *chosen ciphertext attacks*, and we will see that our definition of AE-security is sufficient (and, indeed, necessary) to prevent such attacks. In Section 9.3, we give a more high-level justification for the definition.

9.1.1 One-time authenticated encryption

In practice, one often uses a symmetric key to encrypt a single message. The key is never used again. For example, when sending encrypted email one often picks an ephemeral key and encrypts the email body under this ephemeral key. The ephemeral key is then encrypted and transmitted in the email header. A new ephemeral key is generated for every email.

In these settings one can use a one-time encryption scheme such as a stream cipher. The cipher must be semantically secure, but need not be CPA-secure. Similarly, it suffices that the cipher provides one-time ciphertext integrity, which is a weaker notion than ciphertext-integrity. In particular, we change Attack Game 9.1 so that the adversary can only obtain the encryption of a single message m.

Definition 9.3. *We say that $\mathcal{E} = (E, D)$ provides **one-time ciphertext integrity** if for every efficient single-query adversary $\mathcal{A}$, the value* $\mathrm{CI}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Definition 9.4. *We say that $\mathcal{E} = (E, D)$ provides **one-time authenticated encryption**, or is* 1AE-***secure*** *for short, if $\mathcal{E}$ is semantically secure and provides one-time ciphertext integrity.*

In applications that only use a symmetric key once, 1AE-security suffices. We will show that the encrypt-then-MAC construction of Fig. 9.1 using a semantically secure cipher and a one-time MAC, provides one-time authenticated encryption. Replacing the MAC by a one-time MAC can lead to efficiency improvements.

9.2 Implications of authenticated encryption

Before constructing AE-secure systems, let us first play with Definition 9.1 a bit to see what it implies. Consider a sender, Alice, and a receiver, Bob, who have a shared secret key k. Alice sends a sequence of messages to Bob over a public network. Each message is encrypted with an AE-secure cipher $\mathcal{E} = (E, D)$ using the key k.

For starters, consider an eavesdropping adversary $\mathcal{A}$. Since $\mathcal{E}$ is CPA-secure this does not help $\mathcal{A}$ learn any new information about messages sent from Alice to Bob.

Now consider a more aggressive adversary $\mathcal{A}$ that attempts to make Bob receive a message that was not sent by Alice. We claim this cannot happen. To see why, consider the following single-message example: Alice encrypts to Bob a message m and the resulting ciphertext c is intercepted by $\mathcal{A}$. The adversary's goal is to create some $\hat{c}$ such that $\hat{m} := D(k, \hat{c}) \neq \mathsf{reject}$ and $\hat{m} \neq m$. This $\hat{c}$ would fool Bob into thinking that Alice sent $\hat{m}$ rather than m. But then $\mathcal{A}$ could also win Attack Game 9.1 with respect to $\mathcal{E}$, contradicting $\mathcal{E}$'s ciphertext integrity. Consequently, $\mathcal{A}$ cannot modify c without being detected. More generally, applying the argument to multiple messages shows that $\mathcal{A}$ cannot cause Bob to receive any messages that were not sent by Alice. The more general conclusion here is that *ciphertext integrity* implies *message integrity.*

9.2.1 Chosen ciphertext attacks: a motivating example

We now consider an even more aggressive type of attack, called a **chosen ciphertext attack** for short. As we will see, an AE-secure cipher provides message secrecy and message integrity even against such a powerful attack.

To motivate chosen ciphertext attacks suppose Alice sends an email message to Bob. For simplicity let us assume that every email starts with the letters `To:` followed by the recipient's email address. So, an email to Bob starts with `To:bob@mail.com` and an email to Mel begins with `To:mel@mail.com`. The mail server decrypts every incoming email and writes it into the recipient's inbox: emails that start with `To:bob@mail.com` are written to Bob's inbox and emails that start with `To:mel@mail.com` are written to Mel's inbox.

Mel, the attacker in this story, wants to read the email that Alice sent to Bob. Unfortunately for Mel, Alice was careful and encrypted the email using a key known only to Alice and to the mail server. When the ciphertext c is received at the mail server it will be decrypted and the resulting message is placed into Bob's inbox. Mel will be unable to read it.

Nevertheless, let us show that if Alice encrypts the email with a CPA-secure cipher such as randomized counter mode or randomized CBC mode then Mel can quite easily obtain the email contents. Here is how: Mel will intercept the ciphertext c en-route to the mail server and modify it to obtain a ciphertext $\hat{c}$ so that the decryption of $\hat{c}$ starts with `To:mel@mail.com`, but is otherwise the same as the original message. Mel then forwards $\hat{c}$ to the mail server. When the mail server receives $\hat{c}$ it will decrypt it and (incorrectly) place the plaintext into Mel's inbox where Mel can easily read it.

To successfully carry out this attack, Mel must first solve the following problem: given an encryption c of some message $(u \parallel m)$ where u is a fixed known prefix (in our case $u :=$ `To:bob@mail.com`), compute a ciphertext $\hat{c}$ that will decrypt to the message $(v \parallel m)$, where v is some other prefix (in our case $v :=$ `To:mel@mail.com`).

Let us show that Mel can easily solve this problem, assuming the encryption scheme is either randomized counter mode or randomized CBC. For simplicity, we also assume that u and v are binary strings whose length is the same as the block size of the underlying block cipher. As usual $c[0]$ and $c[1]$ are the first and second blocks of c where $c[0]$ is the random IV. Mel constructs $\hat{c}$ as follows:

- randomized counter mode: define $\hat{c}$ to be the same as c except that $\hat{c}[1] := c[1] \oplus u \oplus v$.

- randomized CBC mode: define $\hat{c}$ to be the same as c except that $\hat{c}[0] := c[0] \oplus u \oplus v$.

It is not difficult to see that in either case the decryption of $\hat{c}$ starts with the prefix v (see Section 3.3.2). Mel is now able to obtain the decryption of $\hat{c}$ and read the secret message m in the clear.

What just happened? We proved that both encryption modes are CPA secure, and yet we just showed how to break them. This attack is an example of a chosen ciphertext attack — by querying for the decryption of $\hat{c}$, Mel was able to deduce the decryption of c. This attack is also another demonstration of how attackers can exploit the *malleability* of a cipher — we saw another attack based on malleability back in Section 3.3.2.

As we just saw, a CPA-secure system can become completely insecure when an attacker can decrypt certain ciphertexts, even if he cannot directly decrypt a ciphertext that interests him. Put another way, the lack of ciphertext integrity can completely compromise secrecy — even if plaintext integrity is not an explicit security requirement.

We informally argue that if Alice used an AE-secure cipher $\mathcal{E} = (E, D)$ then it would be impossible to mount the attack we just described. Suppose Mel intercepts a ciphertext $c := E(k, m)$. He tries to create another ciphertext $\hat{c}$ such that (1) $\hat{m} := D(k, \hat{c})$ starts with prefix v, and (2) the adversary can recover m from $\hat{m}$, in particular $\hat{m} \neq \mathsf{reject}$. Ciphertext integrity, and therefore AE-security, implies that the attacker cannot create this $\hat{c}$. In fact, the attacker cannot create any new valid ciphertexts and therefore an AE-secure cipher foils the attack.

In the next section, we formally define the notion of a chosen ciphertext attack, and show that if a cipher is AE-secure then it is secure even against this type of attack.

9.2.2 Chosen ciphertext attacks: definition

In this section, we formally define the notion of a *chosen ciphertext attack*. In such an attack, the adversary has all the power of an attacker in a chosen plaintext attack, but in addition, the adversary may obtain decryptions of ciphertexts of its choosing — subject to a restriction. Recall that in a chosen plaintext attack, the adversary obtains a number of ciphertexts from its challenger, in response to encryption queries. The restriction we impose is that the adversary may not ask for the decryptions of any of these ciphertexts. While such a restriction is necessary to make the attack game at all meaningful, it may also seem a bit unintuitive: if the adversary can decrypt ciphertexts of its choosing, why would it not decrypt the most important ones? We will explain later (in Section 9.3) more of the intuition behind this definition. We will show in Section 9.2.3 that if a cipher is AE-secure then it is secure against a chosen ciphertext attack.

Here is the formal attack game:

Attack Game 9.2 (CCA security). For a given cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments. For $b = 0, 1$, we define

Experiment b:

- The challenger selects $k \xleftarrow{R} \mathcal{K}$.
- $\mathcal{A}$ then makes a series of queries to the challenger. Each query can be one of two types:
  - *Encryption query:* for $i = 1, 2, \ldots$, the ith encryption query consists of a pair of messages $(m_{i0}, m_{i1}) \in \mathcal{M}^2$. The challenger computes $c_i \xleftarrow{R} E(k, m_{ib})$ and sends c_i to $\mathcal{A}$.
  - *Decryption query:* for $j = 1, 2, \ldots$, the jth decryption query consists of a ciphertext $\hat{c}_j \in \mathcal{C}$ that is not among the responses to the previous encryption queries, i.e.,

    $$\hat{c}_j \notin \{c_1, c_2, \ldots\}.$$

    The challenger computes $\hat{m}_j \leftarrow D(k, \hat{c}_j)$, and sends $\hat{m}_j$ to $\mathcal{A}$.
- At the end of the game, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b and define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \big|\Pr[W_0] - \Pr[W_1]\big|. \quad \Box$$

We stress that in the above attack game, the encryption and decryption queries may be arbitrarily interleaved with one another.

Definition 9.5 (CCA security). *A cipher $\mathcal{E}$ is called* ***semantically secure against chosen ciphertext attack****, or simply* ***CCA-secure****, if for all efficient adversaries $\mathcal{A}$, the value* $\text{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

In some settings, a new key is generated for every message so that a particular key k is only used to encrypt a single message. The system needs to be secure against chosen ciphertext attacks where the attacker fools the user into decrypting multiple ciphertexts using k. For these settings we define security against an adversary that can only issue a *single* encryption query, but many decryption queries.

Definition 9.6 (1CCA security). *In Attack Game 9.2, if the adversary $\mathcal{A}$ is restricted to making a single encryption query, we denote its advantage by* $1\mathrm{CCA}\mathsf{adv}[\mathcal{A},\mathcal{E}]$*. A cipher $\mathcal{E}$ is* ***one-time semantically secure against chosen ciphertext attack****, or simply,* ***1CCA-secure****, if for all efficient adversaries $\mathcal{A}$, the value* $1\mathrm{CCA}\mathsf{adv}[\mathcal{A},\mathcal{E}]$ *is negligible.*

As discussed in Section 2.2.5, Attack Game 9.2 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0,1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\mathrm{CCA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}]$ (and $1\mathrm{CCA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}]$) as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.2.5 (namely, (2.11)) applies here as well:

$$\mathrm{CCA}\mathsf{adv}[\mathcal{A},\mathcal{E}] = 2 \cdot \mathrm{CCA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}]. \tag{9.1}$$

And similarly, for adversaries restricted to a single encryption query, we have:

$$1\mathrm{CCA}\mathsf{adv}[\mathcal{A},\mathcal{E}] = 2 \cdot 1\mathrm{CCA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}]. \tag{9.2}$$

9.2.3 Authenticated encryption implies chosen ciphertext security

We now show that every AE-secure system is also CCA-secure. Similarly, every 1AE-secure system is 1CCA-secure.

Theorem 9.1. *Let $\mathcal{E} = (E, D)$ be a cipher. If $\mathcal{E}$ is AE-secure, then it is CCA-secure. If $\mathcal{E}$ is* 1AE*-secure, then it is 1CCA-secure.*

> *In particular, suppose $\mathcal{A}$ is a CCA-adversary for $\mathcal{E}$ that makes at most Q_{e} encryption queries and Q_{d} decryption queries. Then there exist a CPA-adversary $\mathcal{B}_{\mathrm{cpa}}$ and a* CI*-adversary $\mathcal{B}_{\mathrm{ci}}$, where $\mathcal{B}_{\mathrm{cpa}}$ and $\mathcal{B}_{\mathrm{ci}}$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{CCA}\mathsf{adv}[\mathcal{A},\mathcal{E}] \leq \mathrm{CPA}\mathsf{adv}[\mathcal{B}_{\mathrm{cpa}},\mathcal{E}] + 2Q_{\mathrm{d}} \cdot \mathrm{CI}\mathsf{adv}[\mathcal{B}_{\mathrm{ci}},\mathcal{E}]. \tag{9.3}$$
>
> *Moreover, $\mathcal{B}_{\mathrm{cpa}}$ and $\mathcal{B}_{\mathrm{ci}}$ both make at most Q_{e} encryption queries.*

Before proving this theorem, we point out a converse of sorts: if a cipher is CCA-secure and provides *plaintext* integrity, then it must be AE-secure. You are asked to prove this in Exercise 9.15. These two results together provide strong support for the claim that AE-security is the *right* notion of security for general purpose communication over an insecure network. We also note that it is possible to build a CCA-secure cipher that does not provide ciphertext (or plaintext) integrity — see Exercise 9.12 for an example.

Proof idea. A CCA-adversary $\mathcal{A}$ issues encryption and allowed decryption queries. We first argue that the response to all these decryption queries must be reject. To see why, observe that if the adversary ever issues a valid decryption query c_i whose decryption is not reject, then this c_i can be used to win the ciphertext integrity game. Hence, since all of $\mathcal{A}$'s decryption queries are rejected, the adversary learns nothing by issuing decryption queries and they may as well be discarded. After removing decryption queries we end up with a standard CPA game. The adversary cannot win this game because $\mathcal{E}$ is CPA-secure. We conclude that $\mathcal{A}$ has negligible advantage in winning the CCA game. □

Proof. Let $\mathcal{A}$ be an efficient CCA-adversary attacking $\mathcal{E}$ as in Attack Game 9.2, and which makes at most Q_{e} encryption queries and Q_{d} decryption queries. We want to show that $\mathrm{CCA}\mathsf{adv}[\mathcal{A},\mathcal{E}]$

is negligible, assuming that $\mathcal{E}$ is AE-secure. We will use the bit-guessing versions of the CCA and CPA attack games, and show that

$$\mathrm{CCA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}] \leq \mathrm{CPA}\mathsf{adv}^*[\mathcal{B}_{\mathrm{cpa}},\mathcal{E}] + Q_{\mathrm{d}} \cdot \mathrm{CI}\mathsf{adv}[\mathcal{B}_{\mathrm{ci}},\mathcal{E}]. \tag{9.4}$$

for efficient adversaries $\mathcal{B}_{\mathrm{cpa}}$ and $\mathcal{B}_{\mathrm{ci}}$. Then (9.3) follows from (9.4), along with (9.1) and (5.4). Moreover, as we shall see, the adversary $\mathcal{B}_{\mathrm{cpa}}$ makes at most Q_{e} encryption queries; therefore, if $\mathcal{E}$ is 1AE-secure, it is also 1CCA-secure.

Let us define Game 0 to be the bit-guessing version of the CCA Attack Game 9.2. The challenger in this game, called Game 0, works as follows:

```
    $b \xleftarrow{R} \{0,1\}$   // $\mathcal{A}$ will try to guess $b$
    $k \xleftarrow{R} \mathcal{K}$

    upon receiving the $i$th encryption query $(m_{i0}, m_{i1})$ from $\mathcal{A}$ do:
        send $c_i \xleftarrow{R} E(k, m_{ib})$ to $\mathcal{A}$

    upon receiving the $j$th decryption query $\hat{c}_j$ from $\mathcal{A}$ do:
(1)     send $D(k, \hat{c}_j)$ to $\mathcal{A}$
```

Eventually the adversary outputs a guess $\hat{b} \in \{0,1\}$. We say that $\mathcal{A}$ wins the game if $b = \hat{b}$ and we denote this event by W_0. By definition, the bit-guessing advantage is

$$\mathrm{CCA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}] = \big|\Pr[W_0] - 1/2\big|. \tag{9.5}$$

Game 1. We now modify line (1) in the challenger as follows:

```
(1)     send reject to $\mathcal{A}$
```

We argue that $\mathcal{A}$ cannot distinguish this challenger from the original. Let Z be the event that in Game 1, $\mathcal{A}$ issues a decryption query $\hat{c}_j$ such that $D(k,\hat{c}_j) \neq \mathsf{reject}$. Clearly, Games 0 and 1 proceed identically as long as Z does not happen. Hence, by the Difference Lemma (i.e., Theorem 4.7) it follows that $\big|\Pr[W_0] - \Pr[W_1]\big| \leq \Pr[Z]$.

Using a "guessing strategy" similar to that used in the proof of Theorem 6.1, we can use $\mathcal{A}$ to build a CI-adversary $\mathcal{B}_{\mathrm{ci}}$ that wins the CI attack game with probability at least $\Pr[Z]/Q_{\mathrm{d}}$. Note that in Game 1, the decryption algorithm is not used at all. Adversary $\mathcal{B}_{\mathrm{ci}}$'s strategy is simply to guess a random number $\omega \in \{1,\dots,Q_{\mathrm{d}}\}$, and choose a random bit b in $\{0,1\}$. It then plays the role of challenger to $\mathcal{A}$:

- when $\mathcal{A}$ makes an encryption query (m_0,m_1), our $\mathcal{B}_{\mathrm{ci}}$ forwards m_b to its own challenger, and returns the response to $\mathcal{A}$;
- when $\mathcal{A}$ makes a decryption query $\hat{c}_j$, $\mathcal{B}_{\mathrm{ci}}$ simply sends reject to $\mathcal{A}$. However, when $j = \omega$, our $\mathcal{B}_{\mathrm{ci}}$ outputs $\hat{c}_j$ and halts.

It is not hard to see that $\mathrm{CI}\mathsf{adv}[\mathcal{B}_{\mathrm{ci}},\mathcal{E}] \geq \Pr[Z]/Q_{\mathrm{d}}$, and so

$$\Big|\Pr[W_0] - \Pr[W_1]\Big| \leq \Pr[Z] \leq Q_{\mathrm{d}} \cdot \mathrm{CI}\mathsf{adv}[\mathcal{B}_{\mathrm{ci}},\mathcal{E}]. \tag{9.6}$$

Final reduction. Since all decryption queries are rejected in Game 1, this is essentially a CPA attack game. More precisely, we can construct a CPA adversary $\mathcal{B}_{\mathrm{cpa}}$ that plays the role of challenger to $\mathcal{A}$ as follows:

- when $\mathcal{A}$ makes an encryption query (m_0, m_1), our $\mathcal{B}_{\text{cpa}}$ forwards this to its own CPA challenger, and returns the response to $\mathcal{A}$;

- when $\mathcal{A}$ makes a decryption query, $\mathcal{B}_{\text{cpa}}$ simply sends reject to $\mathcal{A}$.

At the end of the game, $\mathcal{B}_{\text{cpa}}$ simply outputs the bit $\hat{b}$ that $\mathcal{A}$ outputs. Clearly,

$$|\Pr[W_1] - 1/2| = \text{CPA}\mathsf{adv}^*[\mathcal{B}_{\text{cpa}}, \mathcal{E}] \tag{9.7}$$

Putting equations (9.5)–(9.7) together gives us (9.4), which proves the theorem. □

9.3 Encryption as an abstract interface

To further motivate the definition of authenticated encryption we show that it precisely captures an intuitive notion of secure encryption as an *abstract interface.* AE-security implies that the real implementation of this interface may be replaced by an idealized implementation in which messages literally jump from sender to receiver, without going over the network at all (even in encrypted form). We now develop this idea more fully.

Suppose a sender S and receiver R are using some arbitrary Internet-based system (e.g, gambling, auctions, banking — whatever). Also, we assume that S and R have already established a shared, random encryption key k. During the protocol, S will send encryptions of messages $m_1, m_2, \ldots$ to R. The messages m_i are determined by the logic of the protocol S is using, whatever that happens to be. We can imagine S placing a message m_i in his "out-box", the precise details of how the out-box works being of no concern to S. Of course, inside S's out-box, we know what happens: an encryption c_i of m_i under k is computed, and this is sent out over the wire to R.

On the receiving end, when a ciphertext $\hat{c}$ is received at R's end of the wire, it is decrypted using k, and if the decryption is a message $\hat{m} \neq \mathsf{reject}$, the message $\hat{m}$ is placed in R's "in-box". Whenever a message appears in his in-box, R can retrieve it and processes it according to the logic of his protocol, without worrying about how the message got there.

An attacker may try to subvert communication between S and R in a number of ways.

- First, the attacker may drop, re-order, or duplicate the ciphertexts sent by S.

- Second, the attacker may modify ciphertexts sent by S, or inject ciphertexts created out of "whole cloth".

- Third, the attacker may have partial knowledge of some of the messages sent by S, or may even be able to influence the choice of some of these messages.

- Fourth, by observing R's behavior, the attacker may be able to glean partial knowledge of some of the messages processed by R. Even the knowledge of whether or not a ciphertext delivered to R was rejected could be useful.

Having described an abstract encryption interface and its implementation, we now describe an *ideal implementation* of this interface that captures in an intuitive way the guarantees ensured by authenticated encryption. When S drops m_i in its out-box, instead of encrypting m_i, the ideal implementation creates a ciphertext c_i by encrypting a dummy message dummy_i, that has nothing to do with m_i (except that it should be of the same length). Thus, c_i serves as a "handle" for m_i, but does not contain any information about m_i (other than its length). When c_i arrives at R, the corresponding message m_i is magically copied from S's out-box to R's in-box. If a ciphertext arrives at R that is not among the previously generated c_i's, the ideal implementation simply discards it.

This ideal implementation is just a thought experiment. It obviously cannot be physically realized in any efficient way (without first inventing teleportation). As we shall argue, however, if the underlying cipher $\mathcal{E}$ provides authenticated encryption, the ideal implementation is — for all practical purposes — equivalent to the real implementation. Therefore, a protocol designer need not worry about any of the details of the real implementation or the nuances of cryptographic definitions: he can simply pretend he is using the abstract encryption interface with its ideal implementation, in which ciphertexts are just handles and messages magically jump from S to R. Hopefully, analyzing the security properties of the higher-level protocol will be much easier in this setting.

Note that even in the ideal implementation, the attacker may still drop, re-order, or duplicate ciphertexts, and these will cause the corresponding messages to be dropped, re-ordered, or duplicated. Using sequence numbers and buffers, it is not hard to deal with these possibilities, but that is left to the higher-level protocol.

We now argue informally that when $\mathcal{E}$ provides authenticated encryption, the real world implementation is indistinguishable from the ideal implementation. The argument proceeds in three steps. We start with the real implementation, and in each step, we make a slight modification.

- First, we modify the real implementation of R's in-box, as follows. When a ciphertext $\hat{c}$ arrives on R's end, the list of ciphertexts $c_1, c_2, \ldots$ previously generated by S is scanned, and if $\hat{c} = c_i$, then the corresponding message m_i is magically copied from S's out-box into R's in-box, without actually running the decryption algorithm.

  The correctness property of $\mathcal{E}$ ensures that this modification behaves exactly the same as the real implementation.

- Second, we modify the implementation on R's in-box again, so that if a ciphertext $\hat{c}$ arrives on R's end that is not among the ciphertexts generated by S, the implementation simply discards $\hat{c}$.

  The only way the adversary could distinguish this modification from the first is if it could create a ciphertext that would not be rejected and was not generated by S. But this is not possible, since $\mathcal{E}$ has ciphertext integrity.

- Third, we modify the implementation of S's out-box, replacing the encryption of m_i with the encryption of $dummy_i$. The implementation of R's in-box remains as in the second modification. Note that the decryption algorithm is never used in either the second or third modifications. Therefore, an adversary who can distinguish this modification from the second can be used to directly break the CPA-security of $\mathcal{E}$. Hence, since $\mathcal{E}$ is CPA-secure, the two modifications are indistinguishable.

Since the third modification is identical to the ideal implementation, we see that the real and ideal implementations are indistinguishable from the adversary's point of view.

A technical point we have not considered is the possibility that the c_i's generated by S are not unique. Certainly, if we are going to view the c_i's as handles in the ideal implementation, uniqueness would seem to be an essential property. In fact, CPA-security implies that the c_i's generated in the ideal implementation are unique with overwhelming probability — see Exercise 5.12.

9.4 Authenticated encryption ciphers from generic composition

We now turn to constructing authenticated encryption by combining a CPA-secure cipher and a secure MAC. We show that encrypt-then-MAC is always AE-secure, but MAC-then-encrypt is not.

9.4.1 Encrypt-then-MAC

Let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}_{\rm e}, \mathcal{M}, \mathcal{C})$ and let $\mathcal{I} = (S, V)$ be a MAC defined over $(\mathcal{K}_{\rm m}, \mathcal{C}, \mathcal{T})$. The **encrypt-then-MAC** system $\mathcal{E}_{\rm EtM} = (E_{\rm EtM}, D_{\rm EtM})$, or EtM for short, is defined as follows:

$$\begin{aligned} E_{\rm EtM}(\ (k_{\rm e}, k_{\rm m}),\ m) \quad &:= \quad c \xleftarrow{R} E(k_{\rm e}, m), \quad t \xleftarrow{R} S(k_{\rm m}, c) \\ & \quad \text{Output } (c, t) \\ D_{\rm EtM}((k_{\rm e}, k_{\rm m}),\ (c, t)\ ) \quad &:= \quad \text{if } V(k_{\rm m}, c, t) = \mathsf{reject} \text{ then output } \mathsf{reject} \\ & \quad \text{otherwise, output } D(k_{\rm e}, c) \end{aligned}$$

The EtM system is defined over $(\mathcal{K}_{\rm e} \times \mathcal{K}_{\rm m},\ \mathcal{M},\ \mathcal{C} \times \mathcal{T})$. The following theorem shows that $\mathcal{E}_{\rm EtM}$ provides authenticated encryption.

Theorem 9.2. *Let $\mathcal{E} = (E, D)$ be a cipher and let $\mathcal{I} = (S, V)$ be a MAC system. Then $\mathcal{E}_{\rm EtM}$ is* AE*-secure assuming $\mathcal{E}$ is CPA-secure and $\mathcal{I}$ is a secure MAC system. Also, $\mathcal{E}_{\rm EtM}$ is* 1AE*-secure assuming $\mathcal{E}$ is semantically secure and $\mathcal{I}$ is a one-time secure MAC system.*

> *In particular, for every ciphertext integrity adversary $\mathcal{A}_{\rm ci}$ that attacks $\mathcal{E}_{\rm EtM}$ as in Attack Game 9.1 there exists a MAC adversary $\mathcal{B}_{\rm mac}$ that attacks $\mathcal{I}$ as in Attack Game 6.1, where $\mathcal{B}_{\rm mac}$ is an elementary wrapper around $\mathcal{A}_{\rm ci}$, and which makes no more signing queries than $\mathcal{A}_{\rm ci}$ makes encryption queries, such that*
>
> $$\mathrm{CI}\mathsf{adv}[\mathcal{A}_{\rm ci}, \mathcal{E}_{\rm EtM}] = \mathrm{MAC}\mathsf{adv}[\mathcal{B}_{\rm mac}, \mathcal{I}].$$
>
> *For every CPA adversary $\mathcal{A}_{\rm cpa}$ that attacks $\mathcal{E}_{\rm EtM}$ as in Attack Game 5.2 there exists a CPA adversary $\mathcal{B}_{\rm cpa}$ that attacks $\mathcal{E}$ as in Attack Game 5.2, where $\mathcal{B}_{\rm cpa}$ is an elementary wrapper around $\mathcal{A}_{\rm cpa}$, and which makes no more encryption queries than does $\mathcal{A}_{\rm cpa}$, such that*
>
> $$\mathrm{CPA}\mathsf{adv}[\mathcal{A}_{\rm cpa}, \mathcal{E}_{\rm EtM}] = \mathrm{CPA}\mathsf{adv}[\mathcal{B}_{\rm cpa}, \mathcal{E}].$$

Proof. Let us first show that $\mathcal{E}_{\rm EtM}$ provides ciphertext integrity. Suppose $\mathcal{A}_{\rm ci}$ is a ciphertext integrity adversary attacking $\mathcal{E}_{\rm EtM}$. We construct a MAC adversary $\mathcal{B}_{\rm mac}$ attacking $\mathcal{I}$.

Adversary $\mathcal{B}_{\rm mac}$ plays the role of adversary in a MAC attack game for $\mathcal{I}$. It interacts with a MAC challenger $\mathbf{C}_{\rm mac}$ that starts by picking a random $k_{\rm m} \xleftarrow{R} \mathcal{K}_{\rm m}$. Adversary $\mathcal{B}_{\rm mac}$ works by emulating a $\mathcal{E}_{\rm EtM}$ ciphertext integrity challenger for $\mathcal{A}_{\rm ci}$, as follows:

```
$k_e \xleftarrow{R} \mathcal{K}_e$
upon receiving a query $m_i \in \mathcal{M}$ from $\mathcal{A}_{ci}$ do:
        $c_i \xleftarrow{R} E(k_e, m_i)$
        Query $\mathbf{C}_{mac}$ on $c_i$ and obtain $t_i \xleftarrow{R} S(k_m, c_i)$ in response
        Send $(c_i, t_i)$ to $\mathcal{A}_{ci}$   //  then $(c_i, t_i) = E_{EtM}(\ (k_e, k_m),\ m_i)$
eventually $\mathcal{A}_{ci}$ outputs a ciphertext $(c, t) \in \mathcal{C} \times \mathcal{T}$
output the message-tag pair $(c, t)$
```

It should be clear that $\mathcal{B}_{mac}$ responds to $\mathcal{A}_{ci}$'s queries as in a real ciphertext integrity attack game. Therefore, with probability $\mathrm{CIadv}[\mathcal{A}_{ci}, \mathcal{E}_{EtM}]$ adversary $\mathcal{A}_{ci}$ outputs a ciphertext (c, t) that makes it win Attack Game 9.1 so that $(c, t) \notin \{(c_1, t_1), \ldots\}$ and $V(k_m, c, t) = \mathsf{accept}$. It follows that (c, t) is a message-tag pair that lets $\mathcal{B}_{mac}$ win the MAC attack game and therefore $\mathrm{CIadv}[\mathcal{A}_{ci}, \mathcal{E}_{EtM}] = \mathrm{MACadv}[\mathcal{B}_{mac}, \mathcal{I}]$, as required.

It remains to show that if $\mathcal{E}$ is CPA-secure then so is $\mathcal{E}_{EtM}$. This simply says that the tag included in the ciphertext, which is computed using the key k_m (and does not involve the encryption key k_e at all), does not help the attacker break CPA security of $\mathcal{E}_{EtM}$. This is straightforward and is left as an easy exercise (see Exercise 5.20). □

Recall that our definition of a secure MAC from Chapter 6 requires that given a message-tag pair (c, t) the attacker cannot come up with a new tag $t' \neq t$ such that (c, t') is a valid message-tag pair. At the time it seemed odd to require this: if the attacker already has a valid tag for c, why do we care if he finds another tag for c? Here we see that if the attacker could come with a new valid tag t' for c then he could break ciphertext integrity for EtM. From an EtM ciphertext (c, t) the attacker could construct a new valid ciphertext (c, t') and win the ciphertext integrity game. Our definition of secure MAC ensures that the attacker cannot modify an EtM ciphertext without being detected.

9.4.1.1 Common mistakes in implementing encrypt-then-MAC

A common mistake when implementing encrypt-then-MAC is to use the same key for the cipher and the MAC, i.e., setting $k_e = k_m$. The resulting system need not provide authenticated encryption and can be insecure, as shown in Exercise 9.8. In the proof of Theorem 9.2 we relied on the fact that the two keys k_e and k_m are chosen independently.

Another common mistake is to apply the MAC signing algorithm to only part of the ciphertext. We look at an example. Suppose the underlying CPA-secure cipher $\mathcal{E} = (E, D)$ is randomized CBC mode (Section 5.4.3) so that the encryption of a message m is $(r, c) \xleftarrow{R} E(k, m)$ where r is a random IV. When implementing encrypt-then-MAC $\mathcal{E}_{EtM} = (E_{EtM}, D_{EtM})$ the encryption algorithm is incorrectly defined as

$$E_{EtM}\big(\ (k_e, k_m),\ m\big) := \big\{\ (r, c) \xleftarrow{R} E(k_e, m),\ t \xleftarrow{R} S(k_m, c),\ \text{output } (r, c, t)\ \big\}.$$

Here, $E(k_e, m)$ outputs the ciphertext (r, c), but the MAC signing algorithm is only applied to c; the IV is not protected by the MAC. This mistake completely destroys ciphertext integrity: given a ciphertext (r, c, t) an attacker can create a new valid ciphertext (r', c, t) for some $r' \neq r$. The decryption algorithm will not detect this modification of the IV and will not output reject. Instead, the decryption algorithm will output $D\big(k_e,\ (r', c)\big)$. Since (r', c, t) is a valid ciphertext the adversary wins the ciphertext integrity game. Even worse, if (r, c, t) is the encryption of a message m then changing (r, c, t) to $(r \oplus \Delta, c, t)$ for any Δ causes the CBC decryption algorithm to output a message m' where $m'[0] = m[0] \oplus \Delta$. This means that the attacker can change header information in the first block of m to any value of the attacker's choosing. An early edition of the ISO 19772 standard for authenticated encryption made precisely this mistake [119]. Similarly, in 2013 it was discovered that the `RNCryptor` facility in Apple's iOS, built for data encryption, used a faulty encrypt-then-MAC where the HMAC was not applied to the encryption IV [123].

Another pitfall to watch out for in an implementation is that no plaintext data should be output before the integrity tag over the entire message is verified. See Section 9.9 for an example of this.

9.4.2 MAC-then-encrypt is not generally secure: padding oracle attacks on SSL

Next, we consider the MAC-then-encrypt generic composition of a CPA secure cipher and a secure MAC. We show that this construction need not be AE-secure, and can lead to many real world problems.

To define MAC-then-encrypt precisely, let $\mathcal{I} = (S, V)$ be a MAC defined over $(\mathcal{K}_{\rm m}, \mathcal{M}, \mathcal{T})$ and let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K}_{\rm e},\ \mathcal{M} \times \mathcal{T},\ \mathcal{C})$. The **MAC-then-encrypt** system $\mathcal{E}_{\rm MtE} = (E_{\rm MtE}, D_{\rm MtE})$, or MtE for short, is defined as follows:

$$\begin{array}{rcl} E_{\rm MtE}(\ (k_{\rm e}, k_{\rm m}),\ m) & := & t \xleftarrow{\rm R} S(k_{\rm m}, m), \quad c \xleftarrow{\rm R} E(k_{\rm e},\ (m, t)\ ) \\ & & \text{Output } c \\[1em] D_{\rm MtE}((k_{\rm e}, k_{\rm m}),\ c\ ) & := & (m, t) \leftarrow D(k_{\rm e}, c) \\ & & \text{if } V(k_{\rm m}, m, t) = \mathsf{reject} \text{ then output } \mathsf{reject} \\ & & \text{otherwise, output } m \end{array}$$

The MtE system is defined over $(\mathcal{K}_{\rm e} \times \mathcal{K}_{\rm m},\ \mathcal{M},\ \mathcal{C})$.

A badly broken MtE cipher. We show that MtE is not guaranteed to be AE-secure even if $\mathcal{E}$ is a CPA-secure cipher and $\mathcal{I}$ is a secure MAC. In fact, MtE can fail to be secure for widely-used ciphers and MACs and this has lead to many significant attacks on deployed systems.

Consider the SSL 3.0 protocol used to protect WWW traffic for over two decades (the protocol is disabled in modern browsers). SSL 3.0 uses MtE to combine randomized CBC mode encryption and a secure MAC. We showed in Chapter 5 that randomized CBC mode encryption is CPA-secure, yet this combination is badly broken: an attacker can effectively decrypt all traffic using a chosen ciphertext attack. This leads to a devastating attack on SSL 3.0 called **POODLE** [25].

Let us assume that the underlying block cipher used in CBC operates on 16 byte blocks, as in AES. Recall that CBC mode encryption pads its input to a multiple of the block length and SSL 3.0 does so as follows: if a pad of length $p > 0$ bytes is needed, the scheme pads the message with $p - 1$ arbitrary bytes and adds one additional byte whose value is set to $(p - 1)$. If the message length is already a multiple of the block length (16 bytes) then SSL 3.0 adds a dummy block of 16 bytes where the last byte is set to 15 and the first 15 bytes are arbitrary. During decryption the pad is removed by reading the last byte and removing that many more bytes.

Concretely, the cipher $\mathcal{E}_{\rm MtE} = (E_{\rm MtE}, D_{\rm MtE})$ obtained from applying MtE to randomized CBC mode encryption and a secure MAC works as follows:

- $E_{\rm MtE}(\ (k_{\rm e}, k_{\rm m}),\ m)$: First use the MAC signing algorithm to compute a fixed-length tag $t \xleftarrow{\rm R} S(k_{\rm m}, m)$ for m. Next, encrypt $m \parallel t$ with randomized CBC encryption: pad the message and then encrypt in CBC mode using key k_{e} and a random IV. Thus, the following data is encrypted to generate the ciphertext c:

$$\begin{array}{|c|c|c|}\hline \qquad\text{message } m \qquad & \text{tag } t & \text{pad } p \\ \hline\end{array} \tag{9.8}$$

Notice that the tag t does not protect the integrity of the pad. We will exploit this to break CPA security using a chosen ciphertext attack.

- $D_{\mathrm{MtE}}(\ (k_{\mathrm{e}}, k_{\mathrm{m}}),\ c)$: Run CBC decryption to obtain the plaintext data in (9.8). Next, remove the pad p by reading the last byte in (9.8) and removing that many more bytes from the data (i.e., if the last byte is 3 then that byte is removed plus 3 additional bytes). Next, verify the MAC tag and if valid return the remaining bytes as the message. Otherwise, output reject.

Both SSL 3.0 and TLS 1.0 use a defective variant of randomized CBC encryption, discussed in Exercise 5.13, but this is not relevant to our discussion here. Here we will assume that a correct implementation of randomized CBC encryption is used.

The chosen ciphertext attack. We show a chosen ciphertext attack on the system $\mathcal{E}_{\mathrm{MtE}}$ that lets the adversary decrypt any ciphertext of its choice. It follows that $\mathcal{E}_{\mathrm{MtE}}$ need not be AE-secure, even though the underlying cipher is CPA-secure. Throughout this section we let (E, D) denote the block cipher used in CBC mode encryption. It operates on 16-byte blocks.

Suppose the adversary intercepts a valid ciphertext $c := E_{\mathrm{MtE}}(\ (k_{\mathrm{e}}, k_{\mathrm{m}}),\ m)$ for some unknown message m. The length of m is such that after a MAC tag t is appended to m the length of $(m \parallel t)$ is a multiple of 16 bytes. This means that a full padding block of 16 bytes is appended during CBC encryption and the last byte of this pad is 15. Then the ciphertext c looks as follows:

$$c \quad = \quad \underbrace{\boxed{\;c[0]\;}}_{\text{IV}}\underbrace{\boxed{\;c[1]\;}\ \cdots\ }_{\text{encryption of m}}\underbrace{\boxed{c[\ell-1]}}_{\text{encrypted tag}}\underbrace{\boxed{\;c[\ell]\;}}_{\text{encrypted pad}}$$

Lets us first show that the adversary can learn something about $m[0]$ (the first 16-byte block of m). This will break semantic security of $\mathcal{E}_{\mathrm{MtE}}$. The attacker prepares a chosen ciphertext query $\hat{c}$ by replacing the last block of c with $c[1]$. That is,

$$\hat{c} \quad := \quad \boxed{\;c[0]\;}\boxed{\;c[1]\;}\ \cdots\ \boxed{c[\ell-1]}\underbrace{\boxed{\;c[1]\;}}_{\text{encrypted pad?}} \tag{9.9}$$

By definition of CBC decryption, decrypting the last block of $\hat{c}$ yields the 16-byte plaintext block

$$v := D\big(k_{\mathrm{e}}, c[1]\big) \oplus c[\ell-1] = m[0] \oplus c[0] \oplus c[\ell-1].$$

If the last byte of v is 15 then during decryption the entire last block will be treated as a padding block and removed. The remaining string is a valid message-tag pair and will decrypt properly. If the last byte of v is not 15 then most likely the response to the decryption query will be reject.

Put another way, if the response to a decryption query for $\hat{c}$ is not reject then the attacker learns that the last byte of $m[0]$ is equal to the last byte of $u := 15 \oplus c[0] \oplus c[\ell-1]$. Otherwise, the attacker learns that the last byte of $m[0]$ is not equal to the last byte of u. This directly breaks semantic security of the $\mathcal{E}_{\mathrm{MtE}}$: the attacker learned something about the plaintext m.

We leave it as an instructive exercise to recast this attack in terms of an adversary in a chosen ciphertext attack game (as in Attack Game 9.2). With a single plaintext query followed by a single ciphertext query the adversary has advantage 1/256 in winning the game. This already proves that $\mathcal{E}_{\text{MtE}}$ is insecure.

Now, suppose the attacker obtains another encryption of m, call it c', using a different IV. The attacker can use the ciphertexts c and c' to form four useful chosen ciphertext queries: it can replace the last block of either c or c' with either of $c[1]$ or $c'[1]$. By issuing these four ciphertext queries the attacker learns if the last byte of $m[0]$ is equal to the last byte of one of

$$15 \oplus c[0] \oplus c[\ell-1], \qquad 15 \oplus c[0] \oplus c'[\ell-1], \qquad 15 \oplus c'[0] \oplus c[\ell-1], \qquad 15 \oplus c'[0] \oplus c'[\ell-1].$$

If these four values are distinct they give the attacker four chances to learn the last byte of $m[0]$. Repeating this multiple times with more fresh encryptions of the message m will quickly reveal the last byte of $m[0]$. Each chosen ciphertext query reveals that byte with probability 1/256. Therefore, on average, with 256 chosen ciphertext queries the attacker learns the exact value of the last byte of $m[0]$. So, not only can the attacker break semantic security, the attacker can actually recover one byte of the plaintext. Next, suppose the adversary could request an encryption of m shifted one byte to the right to obtain a ciphertext c_1. Plugging $c_1[1]$ into the last block of the ciphertexts from the previous phase (i.e., encryptions of the unshifted m) and issuing the resulting chosen ciphertext queries reveals the second to last byte of $m[0]$. Repeating this for every byte of m eventually reveals all of m. We show next that this gives a real attack on SSL 3.0.

A complete break of SSL 3.0. Chosen ciphertext attacks may seem theoretical, but they frequently translate to devastating real-world attacks. Consider a Web browser and a victim Web server called `bank.com`. The two exchange information encrypted using SSL 3.0. The browser and server have a shared secret called a cookie and the browser embeds this cookie in every request that it sends to `bank.com`. That is, abstractly, requests from the browser to `bank.com` look like:

$$\boxed{\text{GET }\texttt{path}\quad\text{cookie: }\texttt{cookie}}$$

where `path` identifies the name of a resource being requested from `bank.com`. The browser only inserts the cookie into requests that it sends to `bank.com`

The attacker's goal is to recover the secret cookie. First it makes the browser visit `attacker.com` where it sends a Javascript program to the browser. This Javascript program makes the browser issue a request for resource "/AA" at `bank.com`. The reason for this particular path is to ensure that the length of the message and MAC is a multiple of the block size (16 bytes), as needed for the attack. Consequently, the browser sends the following request to `bank.com`

$$\boxed{\text{GET /AA}\quad\text{cookie: }\texttt{cookie}} \tag{9.10}$$

encrypted using SSL 3.0. The attacker can intercept this encrypted request c and mounts the chosen ciphertext attack on MtE to learn one byte of the cookie. That is, the attacker prepares $\hat{c}$ as in (9.9), sends $\hat{c}$ to `bank.com` and looks to see if `bank.com` responds with an SSL error message. If no error message is generated then the attacker learns one byte of the cookie. The Javascript can cause the browser to repeatedly issue the request (9.10) giving the adversary the fresh encryptions needed to eventually learn one byte of the cookie.

Once the adversary learns one byte of the cookie it can shift the cookie one byte to the right by making the Javascript program issue a request to `bank.com` for

$$\boxed{\text{GET /AAA}\quad\text{cookie: }\texttt{cookie}}$$

This gives the attacker a block of ciphertext, call it $c_1[2]$, where the cookie is shifted one byte to the right. Resending the requests from the previous phase to the server, but now with the last block replaced by $c_1[2]$, eventually reveals the second byte of the cookie. Iterating this process for every byte of the cookie eventually reveals the entire cookie.

In effect, Javascript in the browser provides the attacker with the means to mount the desired chosen plaintext attack. Intercepting packets in the network, modifying them and observing the server's response, gives the attacker the means to mount the desired chosen ciphertext attack. The combination of these two completely breaks MtE encryption in SSL 3.0.

One minor detail is that whenever `bank.com` responds with an SSL error message the SSL session shuts down. This does not pose a problem: every request that the Javascript running in the browser makes to `bank.com` initiates a new SSL session. Hence, every chosen ciphertext query is encrypted under a different session key, but that makes no difference to the attack: every query tests if one byte of the cookie is equal to one known random byte. With enough queries the attacker learns the entire cookie.

9.4.3 More padding oracle attacks.

TLS 1.0 is an updated version of SSL 3.0. It defends against the attack of the previous section by adding structure to the pad as explained in Section 5.4.4: when padding with p bytes, all bytes of the pad are set to $p-1$. Moreover, during decryption, the decryptor is required to check that all padding bytes have the correct value and reject the ciphertext if not. This makes it harder to mount the attack of the previous section. Of course our goal was merely to show that MtE is not generally secure and SSL 3.0 made that abundantly clear.

A padding oracle timing attack. Despite the defenses in TLS 1.0 a naive implementation of MtE decryption may still be vulnerable. Suppose the implementation works as follows: first it applies CBC decryption to the received ciphertext; next it checks that the pad structure is valid and if not it rejects the ciphertext; if the pad is valid it checks the integrity tag and if valid it returns the plaintext. In this implementation the integrity tag is checked only if the pad structure is valid. This means that a ciphertext with an invalid pad structure is rejected faster than a ciphertext with a valid pad structure, but an invalid tag. An attacker can measure the time that the server takes to respond to a chosen ciphertext query and if a TLS error message is generated quickly it learns that the pad structure was invalid. Otherwise, it learns that the pad structure was valid.

This timing channel is called a **padding oracle side-channel**. It is a good exercise to devise a chosen ciphertext attack based on this behavior to completely decrypt a secret cookie, as we did for SSL 3.0. To see how this might work, suppose an attacker intercepts an encrypted TLS 1.0 record c. Let m be the decryption of c. Say the attacker wishes to test if the last byte of $m[2]$ is equal to some fixed byte value b. Let B be an arbitrary 16-byte block whose last byte is b. The attacker creates a new ciphertext block $\hat{c}[1] := c[1] \oplus B$ and sends the 3-block record $\hat{c} = (c[0], \hat{c}[1], c[2])$ to the server. After CBC decryption of $\hat{c}$, the last plaintext block will be

$$\hat{m}[2] := \hat{c}[1] \oplus D(k, c[2]) = m[2] \oplus B.$$

If the last byte of $m[2]$ is equal to b then $\hat{m}[2]$ ends in zero which is a valid pad. The server will attempt to verify the integrity tag resulting in a slow response. If the last byte of $m[2]$ is not equal to b then $\hat{m}[2]$ will not end in 0 and will likely end in an invalid pad, resulting in a fast response. By measuring the response time the attacker learns if the last byte of $m[2]$ is equal to b. Repeating this with many chosen ciphertext queries, as we did for SSL 3.0, reveals the entire secret cookie.

An even more sophisticated padding oracle timing attack on MtE, as used in TLS 1.0, is called Lucky13 [5]. It is quite challenging to implement TLS 1.0 decryption in a way that hides the timing information exploited by the Lucky13 attack.

Informative error messages. To make matters worse, the TLS 1.0 specification [52] states that the server should send one type of error message (called `bad_record_mac`) when a received ciphertext is rejected because of a MAC verification error and another type of error message (`decryption_failed`) when the ciphertext is rejected because of an invalid padding block. In principle, this tells the attacker if a ciphertext was rejected because of an invalid padding block or because of a bad integrity tag. This could have enabled the chosen ciphertext attack of the previous paragraph without needing to resort to timing measurements. Fortunately, the error messages are encrypted and the attacker cannot see the error code.

Nevertheless, there is an important lesson to be learned here: when decryption fails, the system should never explain why. A generic '`decryption_failed`' code should be sent without offering any other information. This issue was recognized and addressed in TLS 1.1. Moreover, upon decryption failure, a correct implementation should always take the same amount of time to respond, no matter the failure reason.

9.4.4 Secure instances of MAC-then-encrypt

Although MtE is not generally secure when applied to a CPA-secure cipher, it can be shown to be secure for specific CPA ciphers discussed in Chapter 5. We show in Theorem 9.3 below that if $\mathcal{E}$ happens to implement randomized counter mode, then MtE is secure. In Exercise 9.9 we show that the same holds for randomized CBC, assuming there is no message padding.

Theorem 9.3 shows that MAC-then-encrypt with randomized counter mode is AE-secure even if the MAC is only one-time secure. That is, it suffices to use a weak MAC that is only secure against an adversary that makes a *single* chosen message query. Intuitively, the reason we can prove security using such a weak MAC is that the MAC value is encrypted, and consequently it is harder for the adversary to attack the MAC. Since one-time MACs are a little shorter and faster than many-time MACs, MAC-then-encrypt with randomized counter mode has a small advantage over encrypt-then-MAC. Nevertheless, the attacks on MAC-then-encrypt presented in the previous section suggest that it is difficult to implement correctly, and should not be used.

Our starting point is a randomized counter-mode cipher $\mathcal{E} = (E, D)$, as discussed in Section 5.4.2. We will assume that $\mathcal{E}$ has the general structure as presented in the case study on AES counter mode at the end of Section 5.4.2 (page 194). Namely, we use a counter-mode variant where the cipher $\mathcal{E}$ is built from a secure PRF F defined over $(\mathcal{K}_{\mathrm{e}},\ \mathcal{X} \times \mathbb{Z}_\ell,\ \mathcal{Y})$, where $\mathcal{Y} := \{0,1\}^n$. More precisely, for a message $m \in \mathcal{Y}^{\leq \ell}$ algorithm E works as follows:

$$E(k_{\rm e},\ m) := \left\{ \begin{array}{l} x \xleftarrow{\text{R}} \mathcal{X} \\ \text{for } j = 0 \text{ to } |m| - 1: \\ \qquad u[j] \leftarrow F\big(k_{\rm e},\ (x, j)\big) \oplus m[j] \\ \text{output } c := (x, u) \in \mathcal{X} \times \mathcal{Y}^{|m|} \end{array} \right\}$$

Algorithm $D(k_{\rm e}, c)$ is defined similarly. Let $\mathcal{I} = (S, V)$ be a secure one-time MAC defined over $(\mathcal{K}_{\rm m}, \mathcal{M}, \mathcal{T})$ where $\mathcal{M} := \mathcal{Y}^{\leq \ell_{\rm m}}$ and $\mathcal{T} := \mathcal{Y}^{\ell_{\rm t}}$, and where $\ell_{\rm m} + \ell_{\rm t} < \ell$.

The MAC-then-encrypt cipher $\mathcal{E}_{\rm MtE} = (E_{\rm MtE}, D_{\rm MtE})$, built from F and $\mathcal{I}$ and taking messages in $\mathcal{M}$, is defined as follows:

$$\begin{aligned} E_{\rm MtE}\big(\ (k_{\rm e}, k_{\rm m}),\ m\big) &:= \big\{\ t \xleftarrow{\text{R}} S(k_{\rm m}, m),\ \ c \xleftarrow{\text{R}} E\big(k_{\rm e},\ (m \parallel t)\big),\ \ \text{output } c\ \big\} \\ D_{\rm MtE}\big(\ (k_{\rm e}, k_{\rm m}),\ c\big) &:= \left\{ \begin{array}{l} (m \parallel t) \leftarrow D(k_{\rm e}, c) \\ \text{if } V(k_{\rm m}, m, t) = \textsf{reject} \text{ then output } \textsf{reject} \\ \text{otherwise, output } m \end{array} \right\} \end{aligned} \tag{9.11}$$

As we discussed at the end of Section 9.4.1, and in Exercise 9.8, the two keys $k_{\rm e}$ and $k_{\rm m}$ must be chosen independently. Setting $k_{\rm e} = k_{\rm m}$ will invalidate the following security theorem.

Theorem 9.3. *The cipher $\mathcal{E}_{\rm MtE} = (E_{\rm MtE}, D_{\rm MtE})$ in (9.11) built from the PRF F and MAC $\mathcal{I}$ provides authenticated encryption assuming $\mathcal{I}$ is a secure one-time MAC and F is a secure PRF where $1/|\mathcal{X}|$ is negligible.*

In particular, for every Q-query ciphertext integrity adversary $\mathcal{A}_{\rm ci}$ that attacks $\mathcal{E}_{\rm MtE}$ as in Attack Game 9.1 there exists two MAC adversaries $\mathcal{B}_{\rm mac}$ and $\mathcal{B}'_{\rm mac}$ that attack $\mathcal{I}$ as in Attack Game 6.1, and a PRF adversary $\mathcal{B}_{\rm prf}$ that attacks F as in Attack Game 4.2, each of which is an elementary wrapper around $\mathcal{A}_{\rm ci}$, such that

$$\begin{aligned} \text{CI}\mathsf{adv}[\mathcal{A}_{\rm ci}, \mathcal{E}_{\rm MtE}] \leq\ & \text{PRF}\mathsf{adv}[\mathcal{B}_{\rm prf}, F] + \\ & Q \cdot \text{MAC}_1\mathsf{adv}[\mathcal{B}_{\rm mac}, \mathcal{I}] + \text{MAC}_1\mathsf{adv}[\mathcal{B}'_{\rm mac}, \mathcal{I}] + \frac{Q^2}{2|\mathcal{X}|}. \end{aligned} \tag{9.12}$$

For every CPA adversary $\mathcal{A}_{\rm cpa}$ that attacks $\mathcal{E}_{\rm MtE}$ as in Attack Game 5.2 there exists a CPA adversary $\mathcal{B}_{\rm cpa}$ that attacks $\mathcal{E}$ as in Attack Game 5.2, which is an elementary wrapper around $\mathcal{A}_{\rm cpa}$, such that

$$\text{CPA}\mathsf{adv}[\mathcal{A}_{\rm cpa}, \mathcal{E}_{\rm MtE}] = \text{CPA}\mathsf{adv}[\mathcal{B}_{\rm cpa}, \mathcal{E}]$$

Proof idea. CPA security of the system follows immediately from CPA security of randomized counter mode. The challenge is to prove ciphertext integrity for $\mathcal{E}_{\rm MtE}$. So let $\mathcal{A}_{\rm ci}$ be a ciphertext integrity adversary. This adversary makes a series of queries, $m_1, \ldots, m_Q$. For each m_i, the CI challenger gives to $\mathcal{A}_{\rm ci}$ a ciphertext $c_i = (x_i, u_i)$, where x_i is a random IV, and u_i is a one-time pad encryption of the pair $m_i \parallel t_i$ using a pseudo-random pad r_i derived from x_i using the PRF F. Here, t_i is a MAC tag computed on m_i. At the end of the attack game, adversary $\mathcal{A}_{\rm ci}$ outputs a ciphertext $c = (x, u)$, which is not among the c_i's, and wins if c is a valid ciphertext. This means that u decrypts to $m \parallel t$ using a pseudo-random pad r derived from x, and t is a valid tag on m.

Now, using the PRF security property and the fact that the x_i's are unlikely to repeat, we can effectively replace the pseudo-random r_i's (and r) with truly random pads, without affecting $\mathcal{A}_{\rm ci}$'s advantage significantly. This is where the terms $\text{PRF}\mathsf{adv}[\mathcal{B}_{\text{prf}}, F]$ and $Q^2/2|\mathcal{X}|$ in (9.12) come from. Note that after making this modification, the t_i's are perfectly hidden from the adversary.

We then consider two different ways in which $\mathcal{A}_{\text{ci}}$ can win in this modified attack game.

- In the first way, the value x output by $\mathcal{A}_{\text{ci}}$ is not among the x_i's. But in this case, the only way for $\mathcal{A}_{\text{ci}}$ to win is to hope that a random tag on a random message is valid. This is where the term $\text{MAC}_1\mathsf{adv}[\mathcal{B}'_{\text{mac}}, \mathcal{I}]$ in (9.12) comes from.

- In the second way, the value x is equal to x_j for some $j = 1, \ldots, Q$. In this case, to win, the value u must decrypt under the pad r_j to $m \parallel t$ where t is a valid tag on m. Moreover, since $c \neq c_j$, we have $(m, t) \neq (m_j, t_j)$. To turn $\mathcal{A}_{\text{ci}}$ into a one-time MAC adversary, we have to guess the index j in advance: for all indices i different from the guessed index, we can replace the tag t_i by a dummy tag. This guessing strategy is where the term $Q \cdot \text{MAC}_1\mathsf{adv}[\mathcal{B}_{\text{mac}}, \mathcal{I}]$ in (9.12) comes from. □

Proof. To prove ciphertext integrity, we let $\mathcal{A}_{\text{ci}}$ interact with a number of closely related challengers. For $j = 0, 1, 2, 3, 4$ we define W_j to be the event that the adversary wins in Game j.

Game 0. As usual, we begin by letting $\mathcal{A}_{\text{ci}}$ interact with the standard ciphertext integrity challenger in Attack Game 9.1 as it applies to $\mathcal{E}_{\text{MtE}}$, so that $\Pr[W_0] = \text{CI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{E}_{\text{MtE}}]$.

Game 1. Now, we replace the pseudo-random pads in the counter-mode cipher by truly independent one-time pads. Since F is a secure PRF and $1/|\mathcal{X}|$ is negligible, the adversary will not notice the difference. The resulting CI challenger for $\mathcal{E}_{\text{MtE}}$ works as follows.

```
        $k_{\text{m}} \xleftarrow{\text{R}} \mathcal{K}_{\text{m}}$                         // Choose random MAC key
        $\omega \xleftarrow{\text{R}} \{1, \ldots, Q\}$                    // this $\omega$ will be used in Game 3
        upon receiving the $i$th query  $m_i \in \mathcal{Y}^{\leq \ell_{\text{m}}}$  for $i = 1, 2, \ldots$ do:
(1)         $t_i \leftarrow S(k_{\text{m}}, m_i) \in \mathcal{T}$      // compute the tag for $m_i$
(2)         $x_i \xleftarrow{\text{R}} \mathcal{X}$                     // Choose a random IV
            $r_i \xleftarrow{\text{R}} \mathcal{Y}^{|m_i| + \ell_{\text{t}}}$               // Choose a sufficiently long truly random one-time pad
            $u_i \leftarrow (m_i \parallel t_i) \oplus r_i$,    $c_i \leftarrow (x_i, u_i)$   // build ciphertext
            send $c_i$ to the adversary

        upon receiving $c = (x, u) \notin \{c_1, c_2, \ldots\}$ do:
            // decrypt ciphertext $c$
(3)         if $x = x_j$ for some $j$
                then $(m \parallel t) \leftarrow u \oplus r_j$
(4)             else  $r \xleftarrow{\text{R}} \mathcal{Y}^{|u|}$ and $(m \parallel t) \leftarrow u \oplus r$

            // check resulting message-tag pair
            if $V(k_{\text{m}}, m, t) = \mathsf{accept}$
                then output "win"
                else  output "lose"
```

Note that for specificity, in line (3) if there is more than one j for which $x = x_j$, we can take the smallest such j.

A standard argument shows that there exists an efficient PRF adversary $\mathcal{B}_{\text{prf}}$ such that:

$$|\Pr[W_1] - \Pr[W_0]| \leq \text{PRF}\mathsf{adv}[\mathcal{B}_{\text{prf}}, F] + \frac{Q^2}{2|\mathcal{X}|}. \tag{9.13}$$

Note that if we wanted to be a bit more careful, we would break this argument up into two steps. In the first step, we would play our "PRF card" to replace $F(k_\mathrm{e}, \cdot)$ be a truly random function f. This introduces the term $\mathrm{PRF}\mathsf{adv}[\mathcal{B}_\mathrm{prf}, F]$ in (9.13). In the second step, we would use the "forgetful gnome" technique to make all the outputs of f independent. Using the Difference Lemma applied to the event that all of the x_i's are distinct introduces the term $Q^2/2|\mathcal{X}|$ in (9.13).

Game 2. Now we restrict the adversary's winning condition to require that the IV used in the final ciphertext c is the same as one of the IVs given to $\mathcal{A}_\mathrm{ci}$ during the game. In particular, we replace line (4) with

```
(4)            else  output "lose" (and stop)
```

Let Z_2 be the event that in Game 2, the final ciphertext $c = (x, u)$ from $\mathcal{A}_\mathrm{ci}$ is valid despite using a previously unused $x \in \mathcal{X}$. We know that the two games proceed identically, unless event Z_2 happens. When event Z_2 happens in Game 2 then the resulting pair (m, t) is uniformly random in $\mathcal{Y}^{|u|-\ell_\mathrm{t}} \times \mathcal{Y}^{\ell_\mathrm{t}}$. Such a pair is unlikely to form a valid message-tag pair. Not only that, the challenger in Game 2 effectively encrypts all of the tags t_i generated in line (1) with a one-time pad, so these tags could be replaced by dummy tags, without affecting the probability that Z_2 occurs. Based on these observations, we can easily construct an efficient MAC adversary $\mathcal{B}'_\mathrm{mac}$ such that $\Pr[Z_2] \le \mathrm{MAC}_1\mathsf{adv}[\mathcal{B}'_\mathrm{mac}, \mathcal{I}]$. Adversary $\mathcal{B}'_\mathrm{mac}$ runs as follows. It plays the role of challenger to $\mathcal{A}_\mathrm{ci}$ as in Game 2, except that in line (1) above, it computes $t_i \leftarrow 0^{\ell_\mathrm{t}}$. When $\mathcal{A}_\mathrm{ci}$ outputs $c = (x, u)$, adversary $\mathcal{B}'_\mathrm{mac}$ outputs a random pair in $\mathcal{Y}^{|u|-\ell_\mathrm{t}} \times \mathcal{Y}^{\ell_\mathrm{t}}$. Hence, by the difference lemma, we have

$$|\Pr[W_2] - \Pr[W_1]| \le \mathrm{MAC}_1\mathsf{adv}[\mathcal{B}'_\mathrm{mac}, \mathcal{I}]. \tag{9.14}$$

Game 3. We further constrain the adversary's winning condition by requiring that the ciphertext forgery use the IV from ciphertext number ω given to $\mathcal{A}_\mathrm{ci}$. Here ω is a random number in $\{1, \dots, Q\}$ chosen by the challenger. The only change to the winning condition of Game 2 is that line (3) now becomes:

```
(3)        if $x = x_\omega$ then
```

Since ω is independent of $\mathcal{A}_\mathrm{ci}$'s view, we know that

$$\Pr[W_3] \ge (1/Q) \cdot \Pr[W_2] \tag{9.15}$$

Game 4. Finally, we change the challenger so that it only computes a valid tag for query number ω issued by $\mathcal{A}_\mathrm{ci}$. For all other queries the challenger just makes up an arbitrary (invalid) tag. Since the tags are encrypted using one-time pads the adversary cannot tell that he is given encryptions of invalid tags. In particular, the only difference from Game 3 is that we replace line (1) by the following two lines:

```
(1)        $t_i \leftarrow (0^n)^{\ell_\mathrm{t}} \in \mathcal{T}$
           if $i = \omega$ then $t_i \leftarrow S(k_\mathrm{m}, m_i) \in \mathcal{T}$   //  only compute correct tag for $m_\omega$
```

Since the adversary's view in this game is identical to its view in Game 3 we have

$$\Pr[W_4] = \Pr[W_3] \tag{9.16}$$

Final reduction. We claim that there is an efficient one-time MAC forger $\mathcal{B}_\mathrm{mac}$ so that

$$\Pr[W_4] = \mathrm{MAC}_1\mathsf{adv}[\mathcal{B}_\mathrm{mac}, \mathcal{I}] \tag{9.17}$$

Adversary $\mathcal{B}_\mathrm{mac}$ interacts with a MAC challenger $\mathbf{C}$ and works as follows:

```
$\omega \xleftarrow{R} \{1, \dots, Q\}$
upon receiving the $i$th query $m_i \in \{0,1\}^{\ell_m}$ for $i = 1, 2, \dots$ do:
        $t_i \leftarrow (0^n)^{\ell_t} \in \mathcal{T}$
        if $i = \omega$ then query $\mathbf{C}$ for the tag on $m_i$ and let $t_i \in \mathcal{T}$ be the response
        $x_i \xleftarrow{R} \mathcal{X}$            // Choose a random IV
        $r_i \xleftarrow{R} \mathcal{Y}^{|m|+\ell_t}$        // Choose a sufficiently long random one-time pad
        $u_i \leftarrow (m_i \parallel t_i) \oplus r_i$,      $c_i \leftarrow (x_i, u_i)$
        send $c_i$ to the adversary

when $\mathcal{A}_{ci}$ outputs $c = (x, u)$ do:
        if $x = x_\omega$ then
                $(m \parallel t) \leftarrow u \oplus r_\omega$
                output $(m, t)$ as the message-tag forgery
```

Since $c \neq c_\omega$ we know that $(m, t) \neq (m_\omega, t_\omega)$. Hence, whenever $\mathcal{A}_{ci}$ wins Game 4 we know that $\mathcal{B}_{mac}$ does not abort, and outputs a pair (m, t) that lets it win the one-time MAC attack game. It follows that $\Pr[W_4] = \text{MAC}_1\mathsf{adv}[\mathcal{B}_{mac}, \mathcal{I}]$ as required. In summary, putting equations (9.13)–(9.17) together proves the theorem. □

9.4.5 Encrypt-then-MAC or MAC-then-encrypt?

So far we proved the following facts about the MtE and EtM modes:

- EtM provides authenticated encryption whenever the cipher is CPA-secure and the MAC is secure. The MAC on the ciphertext prevents any tampering with the ciphertext.

- MtE is not generally secure — there are examples of CPA-secure ciphers for which the MtE system is not AE-secure. Moreover, MtE is difficult to implement correctly due to a potential timing side-channel that can lead to a chosen ciphertext attack. However, for specific ciphers, such as randomized counter mode and randomized CBC, the MtE mode is AE-secure even if the MAC is only one-time secure.

- A third mode, called encrypt-and-MAC (EaM), is discussed in Exercise 9.10. The exercise shows that EaM is secure when using randomized counter-mode cipher as long as the MAC is a secure PRF. EaM is inferior to EtM in every respect and should not be used.

These facts, and the example attacks on MtE, suggest that EtM is the better mode to use. Of course, it is critically important that the underlying cipher be CPA-secure and the underlying MAC be a secure MAC. Otherwise, EtM may provide no security at all.

Given all the past mistakes in implementing these modes it is advisable that developers not implement EtM themselves. Instead, it is best to use an encryption standard, like GCM (see Section 9.7), that uses EtM to provide authenticated encryption out of the box.

9.5 Nonce-based authenticated encryption with associated data

In this section we extend the syntax of authenticated encryption to match the way in which it is commonly used. First, as we did for encryption and for MACs, we define nonce-based authenticated encryption where we make the encryption and decryption algorithms deterministic, but let them take as input a unique nonce. This approach can reduce ciphertext size and also improve security.

Second, we extend the encryption algorithm by giving it an additional input message, called **associated data**, whose integrity is protected by the ciphertext, but its secrecy is not. The need for associated data comes up in a number of settings. For example, when encrypting packets in a networking protocol, authenticated encryption protects the packet body, but the header must be transmitted in the clear so that the network can route the packet to its intended destination. Nevertheless, we want to ensure header integrity. The header is provided as the associated data input to the encryption algorithm.

A cipher that supports associated data is called an **AD cipher**. The syntax for a nonce-based AD cipher $\mathcal{E} = (E, D)$ is as follows:

$$c = E(k, m, d, \mathcal{N}),$$

where $c \in \mathcal{C}$ is the ciphertext, $k \in \mathcal{K}$ is the key, $m \in \mathcal{M}$ is the message, $d \in \mathcal{D}$ is the associated data, and $\mathcal{N} \in \mathscr{N}$ is the nonce. Moreover, the encryption algorithm E is required to be deterministic. Likewise, the decryption syntax becomes

$$D(k, c, d, \mathcal{N})$$

which outputs a message m or reject. We say that the nonce-based AD cipher is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{D}, \mathcal{C}, \mathscr{N})$. As usual, we require that ciphertexts generated by E are correctly decrypted by D, *as long as both are given the same nonce and associated data*. That is, for all keys k, all messages m, all associated data d, and all nonces $\mathcal{N} \in \mathscr{N}$:

$$D\big(k,\ E(k,\ m,\ d,\ \mathcal{N}),\ d,\ \mathcal{N}\big) = m.$$

If the message m given as input to the encryption algorithm is the empty message then cipher (E, D) essentially becomes a MAC system for the associated data d.

CPA security. A nonce-based AD cipher is CPA-secure if it does not leak any useful information to an eavesdropper assuming that *no nonce is used more than once* in the encryption process. CPA security for a nonce-based AD cipher is defined as CPA security for a standard nonce-based cipher (Section 5.5). The only difference is in the encryption queries. Encryption queries in Experiment b, for $b = 0, 1$, are processed as follows:

> The ith encryption query is a pair of messages, $m_{i0}, m_{i1} \in \mathcal{M}$, of the same length, associated data $d_i \in \mathcal{D}$, and a unique nonce $\mathcal{N}_i \in \mathscr{N} \setminus \{\mathcal{N}_1, \ldots, \mathcal{N}_{i-1}\}$.
>
> The challenger computes $c_i \leftarrow E(k, m_{ib}, d_i, \mathcal{N}_i)$, and sends c_i to the adversary.

Nothing else changes from the definition in Section 5.5. Note that the associated data d_i is under the adversary's control, as are the nonces $\mathcal{N}_i$, subject to the nonces being unique. For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\mathrm{nCPA}_{\mathrm{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \left|\Pr[W_0] - \Pr[W_1]\right|. \quad \Box$$

Definition 9.7 (CPA security). *A nonce-based AD cipher is called* ***semantically secure against chosen plaintext attack****, or simply* ***CPA-secure****, if for all efficient adversaries $\mathcal{A}$, the quantity* $\mathrm{nCPA}_{\mathrm{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Ciphertext integrity. A nonce-based AD cipher provides ciphertext integrity if an attacker who can request encryptions under key k for messages, associated data, and nonces of his choice cannot output a new triple $(c, d, \mathcal{N})$ that is accepted by the decryption algorithm. The adversary, however, must never issue an encryption query using a previously used nonce.

More precisely, we modify the ciphertext integrity game (Attack Game 9.1) as follows:

Attack Game 9.3 (ciphertext integrity). For a given AD cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{D}, \mathcal{C}, \mathscr{N})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger chooses a random $k \overset{R}{\leftarrow} \mathcal{K}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots,$ the ith query consists of a message $m_i \in \mathcal{M}$, associated data $d_i \in \mathcal{D}$, and a previously unused nonce $\mathcal{N}_i \in \mathscr{N} \setminus \{\mathcal{N}_1, \ldots, \mathcal{N}_{i-1}\}$. The challenger computes $c_i \overset{R}{\leftarrow} E(k, m_i, d_i, \mathcal{N}_i)$, and gives c_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate triple $(c, d, \mathcal{N})$ where $c \in \mathcal{C}$, $d \in \mathcal{D}$, and $\mathcal{N} \in \mathscr{N}$ that is not among the triples it was given, i.e.,

$$(c, d, \mathcal{N}) \notin \{(c_1, d_1, \mathcal{N}_1),\ (c_2, d_2, \mathcal{N}_2),\ \ldots\}.$$

We say that $\mathcal{A}$ wins the game if $D(k, c, d, \mathcal{N}) \neq \mathsf{reject}$. We define $\mathcal{A}$'s advantage with respect to $\mathcal{E}$, denoted $\mathrm{nCI_{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 9.8. *We say that a nonce-based AD cipher $\mathcal{E} = (E, D)$ has* ***ciphertext integrity*** *if for all efficient adversaries $\mathcal{A}$, the value $\mathrm{nCI_{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible.*

Authenticated encryption. We can now define nonce-based authenticated encryption for an AD cipher. We refer to this notion as a **nonce-based AEAD cipher** which is shorthand for **authenticated encryption with associated data**.

Definition 9.9. *We say that a nonce-based AD cipher $\mathcal{E} = (E, D)$ provides authenticated encryption, or is simply a* ***nonce-based AEAD cipher****, if $\mathcal{E}$ is CPA-secure and has ciphertext integrity.*

Generic encrypt-then-MAC composition. We construct a nonce-based AEAD cipher $\mathcal{E} = (E_{\mathrm{EtM}}, D_{\mathrm{EtM}})$ by combining a nonce-based CPA-secure cipher (E, D) (as in Section 5.5) with a nonce-based secure MAC (S, V) (as in Section 7.5) as follows:

$$\begin{aligned}
E_{\mathrm{EtM}}(\ (k_{\mathrm{e}}, k_{\mathrm{m}}),\ m, d, \mathcal{N}) \quad &:= \quad c \leftarrow E(k_{\mathrm{e}}, m, \mathcal{N}), \quad t \leftarrow S(k_{\mathrm{m}}, (c, d), \mathcal{N}) \\
& \text{Output } (c, t) \\
D_{\mathrm{EtM}}((k_{\mathrm{e}}, k_{\mathrm{m}}),\ (c, t),\ d, \mathcal{N}) \quad &:= \quad \text{if } V(k_{\mathrm{m}},\ (c, d),\ t,\ \mathcal{N}) = \mathsf{reject} \text{ then output } \mathsf{reject} \\
& \text{otherwise, output } D(k_{\mathrm{e}}, c, d, \mathcal{N})
\end{aligned}$$

The EtM system is defined over $(\mathcal{K}_{\mathrm{e}} \times \mathcal{K}_{\mathrm{m}}, \mathcal{M}, \mathcal{D}, \mathcal{C} \times \mathcal{T}, \mathscr{N})$. The following theorem shows that $\mathcal{E}_{\mathrm{EtM}}$ is a secure AEAD cipher.

Theorem 9.4. *Let $\mathcal{E} = (E, D)$ be a nonce-based cipher and let $\mathcal{I} = (S, V)$ be a nonce-based MAC system. Then $\mathcal{E}_{\mathrm{EtM}}$ is a nonce-based AEAD cipher assuming $\mathcal{E}$ is CPA-secure and $\mathcal{I}$ is a secure MAC system.*

The proof of Theorem 9.4 is essentially the same as the proof of Theorem 9.2.

9.6 One more variation: CCA-secure ciphers with associated data

In Section 9.5, we introduced two new features to our ciphers: nonces and associated data. There are two variations we could consider: ciphers with nonces but without associated data, and ciphers with associated data but without nonces. We could also consider all of these variations with respect to other security notions, such as CCA security. Considering all of these variations in detail would be quite tedious. However, we consider one variation that will be important later in the text, namely CCA-secure ciphers with associated data (but without nonces).

To define this notion, we begin by defining the syntax for a cipher with associated data, or AD cipher, without nonces. For such a cipher $\mathcal{E} = (E, D)$, the encryption algorithm may be probabilistic and works as follows:

$$c \xleftarrow{R} E(k, m, d),$$

where $c \in \mathcal{C}$ is the ciphertext, $k \in \mathcal{K}$ is the key, $m \in \mathcal{M}$ is the message, and $d \in \mathcal{D}$ is the associated data. The decryption syntax is

$$D(k, c, d),$$

which outputs a message m or reject. We say that the AD cipher is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{D}, \mathcal{C})$. As usual, we require that ciphertexts generated by E are correctly decrypted by D, as long as both are given the same associated data. That is,

$$\Pr\big[D\big(k,\ E(k,\ m,\ d),\ d\big) = m\big] = 1.$$

Definition 9.10 (CCA and 1CCA security with associated data). *The definition of CCA security for ordinary ciphers carries over naturally to AD ciphers. Attack Game 9.2 is modified as follows. For encryption queries, in addition to a pair of messages (m_{i0}, m_{i1}), the adversary also submits associated data d_i, and the challenger computes $c_i \xleftarrow{R} E(k, m_{ib}, d_i)$. For decryption queries, in addition to a ciphertext $\hat{c}_j$, the adversary submits associated data $\hat{d}_j$, and the challenger computes $\hat{m}_j \leftarrow D(k, \hat{c}_j, \hat{d}_j)$. The restriction is that the* pair *$(\hat{c}_j, \hat{d}_j)$ may not be among the pairs $(c_1, d_1), (c_2, d_2), \ldots$ corresponding to previous encryption queries. An adversary $\mathcal{A}$'s advantage in this game is denoted $\mathrm{CCA_{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, and the cipher is said to be **CCA secure** if this advantage is negligible for all efficient adversaries $\mathcal{A}$. If we restrict the adversary to a single encryption query, as in Definition 9.6, the advantage is denoted $\mathrm{1CCA_{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, and the cipher is said to be **1CCA secure** if this advantage is negligible for all efficient adversaries $\mathcal{A}$.*

Generic encrypt-then-MAC composition. In later applications, the notion that we will use is 1CCA security, so for simplicity we focus on that notion for now. We construct a 1CCA-secure AD cipher $\mathcal{E} = (E_{\mathrm{EtM}}, D_{\mathrm{EtM}})$ by combining a semantically secure cipher (E, D) with a one-time MAC (S, V) as follows:

$$\begin{aligned}
E_{\mathrm{EtM}}(\ (k_{\mathrm{e}}, k_{\mathrm{m}}),\ m, d) \quad &:= \quad c \xleftarrow{R} E(k_{\mathrm{e}}, m), \quad t \xleftarrow{R} S(k_{\mathrm{m}}, (c, d)) \\
&\qquad \text{Output } (c, t) \\
D_{\mathrm{EtM}}((k_{\mathrm{e}}, k_{\mathrm{m}}),\ (c, t),\ d) \quad &:= \quad \text{if } V(k_{\mathrm{m}},\ (c, d),\ t) = \mathsf{reject} \text{ then output } \mathsf{reject} \\
&\qquad \text{otherwise, output } D(k_{\mathrm{e}}, c, d)
\end{aligned}$$

The EtM system is defined over $(\mathcal{K}_{\mathrm{e}} \times \mathcal{K}_{\mathrm{m}}, \mathcal{M}, \mathcal{D}, \mathcal{C} \times \mathcal{T})$.

Theorem 9.5. *Let $\mathcal{E} = (E, D)$ be a semantically secure cipher and let $\mathcal{I} = (S, V)$ be a one-time secure MAC system. Then $\mathcal{E}_{\mathrm{EtM}}$ is a 1CCA-secure AD cipher.*

The proof of Theorem 9.5 is straightforward, and we leave it as an exercise to the reader.

We observe that in most common implementations of the semantically secure cipher $\mathcal{E} = (E, D)$, the encryption algorithm E is deterministic. Likewise, in the most common implementations of the one-time secure MAC $\mathcal{I} = (S, V)$, the signing algorithm is deterministic. So for such implementations, the resulting 1CCA-secure AD cipher will have a deterministic encryption algorithm.

9.7 Case study: Galois counter mode (GCM)

Galois counter mode (GCM) is a popular nonce-based AEAD cipher standardized by NIST in 2007. GCM is an encrypt-then-MAC cipher combining a CPA-secure cipher and a secure MAC. The CPA secure cipher is nonce-based counter mode, usually using AES. The secure MAC is a Carter-Wegman MAC built from a keyed hash function called GHASH, a variant of the function H_{xpoly} from Section 7.4. When encrypting the empty message the cipher becomes a MAC system called **GMAC** providing integrity for the associated data.

GCM uses an underlying block cipher $\mathcal{E} = (E, D)$ such as AES defined over $(\mathcal{K}, \mathcal{X})$ where $\mathcal{X} := \{0,1\}^{128}$. The block cipher is used for both counter mode encryption and the Carter-Wegman MAC. The GHASH function is defined over $(\mathcal{X}, \mathcal{X}^{\leq \ell}, \mathcal{X})$ for $\ell := 2^{32} - 1$.

GCM can take variable size nonces, but let us first describe GCM using a 96-bit nonce $\mathcal{N}$ which is the simplest case. The GCM encryption algorithm operates as follows:

```
input: key $k \in \mathcal{K}$, message $m$, associated data $d$, and nonce $\mathcal{N} \in \{0,1\}^{96}$
$k_{\mathrm{m}} \leftarrow E(k,\ 0^{128})$   // first, generate the key for GHASH (a variant of $H_{\mathrm{xpoly}}$)
Compute the initial value of the counter in counter mode encryption:
    $x \leftarrow (\mathcal{N} \parallel 0^{31}1) \in \{0,1\}^{128}$
    $x' \leftarrow x + 1$   // initial value of counter
$c \leftarrow$ {encrypt $m$ using counter mode starting the counter at $x'$}
$d' \leftarrow$ {pad $d$ with zeros to closest multiple of 128 bits}
$c' \leftarrow$ {pad $c$ with zeros to closest multiple of 128 bits}
Compute the Carter-Wegman MAC:
(*)     $h \leftarrow \mathrm{GHASH}\Big(k_{\mathrm{m}},\ \big(d' \parallel c' \parallel \mathrm{length}(d) \parallel \mathrm{length}(c)\big)\Big) \in \{0,1\}^{128}$
        $t \leftarrow h \oplus E(k, x) \in \{0,1\}^{128}$
output $(c, t)$   // encrypt-then-MAC ciphertext
```

Each of the length fields on line $(*)$ is a 64-bit value indicating the length in bytes of the respective field. If the input nonce $\mathcal{N}$ is not 96-bits long, then $\mathcal{N}$ is padded to the closest multiple of 128 bits, yielding the padded string $\mathcal{N}'$, and the initial counter value x is computed as $x \leftarrow \mathrm{GHASH}\big(k_{\mathrm{m}},\ (\mathcal{N}' \parallel \mathrm{length}(\mathcal{N}))\big)$ which is a value in $\{0,1\}^{128}$.

As usual, the integrity tag t can be truncated to whatever length is desired. The shorter the tag t the more vulnerable the system becomes to ciphertext integrity attacks.

Messages to be encrypted must be less than 2^{32} blocks each (i.e., messages must be in $\mathcal{X}^v$ for some $v < 2^{32}$). Recommendations in the standard suggest that a single key k should not be used to encrypt more than 2^{32} messages.

The GCM decryption algorithm takes as input a key $k \in \mathcal{K}$, a ciphertext (c, t), associated data d and a nonce $\mathcal{N}$. It operates as in encrypt-then-MAC: it first derives $k_{\mathrm{m}} \leftarrow E(k, 0^{128})$ and checks the Carter-Wegman integrity tag t. If valid it outputs the counter mode decryption of c. We emphasize that decryption must be atomic: no plaintext data is output before the integrity tag is verified over the entire message.

GHASH. It remains to describe the keyed hash function GHASH defined over $(\mathcal{X}, \mathcal{X}^{\leq \ell}, \mathcal{X})$. This hash function is used in a Carter-Wegman MAC and therefore, for security, must be a DUF. In Section 7.4 we showed that the function H_{xpoly} is a DUF. Recall that $H_{\mathrm{xpoly}}(k, z)$ is defined as the evaluation of a polynomial derived from z at the point k. We described H_{xpoly} using arithmetic modulo a prime p, so that the blocks of z and the output are elements in $\mathbb{Z}_p$.

The hash function GHASH is almost the same as H_{xpoly}, except that the input message blocks and the output are elements of $\{0,1\}^{128}$. Also, the DUF property holds with respect to the XOR operator $\oplus$, rather than subtraction modulo some number. As discussed in Remark 7.4, to build an XOR-DUF we use polynomials defined over the finite field $\mathrm{GF}(2^{128})$. This is a field of 2^{128} elements called a **Galois field**, which is where GCM gets its name. This field is defined by the irreducible polynomial $g(X) := X^{128} + X^7 + X^2 + X + 1$. Elements of $\mathrm{GF}(2^{128})$ are polynomials over GF(2) of degree less than 128, with arithmetic done modulo $g(X)$. While that sounds fancy, an element of $\mathrm{GF}(2^{128})$ can be conveniently represented as a string of 128 bits (each bit encodes one of the coefficients of the polynomial). Addition in the field is just XOR. Multiplication is done by multiplying the two given polynomials and reducing the result modulo g. Modern processors provide direct hardware support for this operation.

With this notation, for $k \in \mathrm{GF}(2^{128})$ and $z \in \big(\mathrm{GF}(2^{128})\big)^v$ the function $\mathrm{GHASH}(k, z)$ is simply polynomial evaluation in $\mathrm{GF}(2^{128})$:

$$\mathrm{GHASH}(k, z) := z[0] \cdot k^v + z[1] \cdot k^{v-1} + \ldots + z[v-1] \cdot k \ \in \mathrm{GF}(2^{128}) \tag{9.18}$$

That's it. Appending the two length fields to the GHASH input on line $(*)$ ensures that the XOR-DUF property is maintained even for messages of different lengths.

Security. The AEAD security of GCM is similar to the analysis we did for generic composition of encrypt-then-MAC (Theorem 9.4), and follows from the security of the underlying block cipher as a PRF. The main difference between GCM and our generic composition is that GCM "cuts a few corners" when it comes to keys: it uses a single key k and uses $E(k, 0^n)$ as the GHASH key, and $E(k, x)$ as the pad that is used to mask the output of GHASH. This is similar to, but not exactly the sames as, what is done in Carter-Wegman. Importantly, the counter mode encryption begins with the counter value $x' := x + 1$, so that the inputs to the PRF that are used to encrypt the message are guaranteed to be distinct from the inputs used to derive the GHASH key and pad. The above discussion focused on the case where the nonce is 96 bits. The other case, where GHASH is applied to the nonce to compute x, requires a more involved analysis — see Exercise 9.14.

GCM has no nonce re-use resistance. If a nonce is accidentally re-used on two different messages then all secrecy for those messages is lost. Even worse, the GHASH secret key k_{m} is exposed (Exercise 7.13) and this can be used to break ciphertext integrity. Hence, it is vital that nonces not be re-used in GCM.

Optimizations and performance. There are many ways to optimize the implementation of GCM and GHASH. In practice, the polynomial in (9.18) is evaluated using Horner's method so that processing each block of plaintext requires only one addition and one multiplication in $\mathrm{GF}(2^{128})$.

Intel added a special instruction, called PCLMULQDQ, to their instruction set to quickly carry out binary polynomial multiplication. This instruction cannot be used directly to implement GHASH because of incompatibility with how the standard represents elements in $\mathrm{GF}(2^{128})$. Fortunately, work of Gueron shows how to overcome these difficulties and use the PCLMULQDQ instruction to speed-up GHASH on Intel platforms.

Since GHASH needs only one addition and one multiplication in $\mathrm{GF}(2^{128})$ per block, one would expect that the bulk of the time during GCM encryption and decryption is spent on computing AES in counter mode. However, due to improvements in hardware AES, especially pipelining of the AES-NI instructions, this is not always the case. On Intel's Haswell processors (introduced in 2013) GCM is about three times slower than pure counter mode due to the overhead of computing GHASH. However, improvements in the hardware implementation of PCLMULQDQ make GCM just slightly more expensive than pure AES counter mode, which is the best one can hope for.

9.8 Case study: the TLS 1.3 record protocol

The Transport Layer Security (TLS) protocol is by far the most widely deployed security protocol. Virtually every online purchase is protected by TLS. Although TLS is primarily used to protect Web traffic, it is a general protocol that can protect many types of traffic: email, messaging, and many others.

The original version of TLS was designed at Netscape where it was called the Secure Socket Layer protocol or SSL. SSL 2.0 was designed in 1994 to protect Web e-commerce traffic. SSL 3.0, designed in 1995, corrected several significant security problems in SSLv2. For example, SSL 2.0 uses the same key for both the cipher and the MAC. While this is bad practice — it invalidates the proofs of security for MtE and EtM — it also implies that if one uses a weak cipher key, say due to export restrictions, then the MAC key must also be weak. SSL 2.0 supported only a small number of algorithms and, in particular, only supported MD5-based MACs.

The Internet Engineering Task Force (IETF) created the Transport Layer Security (TLS) working group to standardize an SSL-like protocol. The working group produced a specification for the TLS 1.0 protocol in 1999 [52]. TLS 1.0 is a minor variation of SSL 3.0 and is often referred to as SSL version 3.1. Minor updates were introduced in 2006, and again in 2008, leading to TLS version 1.2. Due to several security vulnerabilities in TLS 1.2, the protocol was overhauled in 2017, resulting in a much stronger TLS version 1.3. TLS has become ubiquitous, and is used worldwide in many software systems. Here we will focus on TLS 1.3

The TLS 1.3 record protocol. Abstractly, TLS consists of two components. The first, called **TLS session setup**, negotiates the cipher suite that will be used to encrypt the session and then sets up a shared secret between the browser and server. The second, called the **TLS record protocol** uses this shared secret to securely transmit data between the two sides. TLS session setup uses public-key techniques and will be discussed later in Chapter 21. Here we focus on the TLS record protocol.

In TLS terminology, the shared secret generated during session setup is called a **master-secret**. This high entropy master secret is used to derive two keys $k_{b \to s}$ and $k_{s \to b}$. The key $k_{b \to s}$ encrypts messages from the browser to the server while $k_{s \to b}$ encrypts messages in the reverse direction. TLS derives the two keys by using the master secret and other randomness as a seed for a key derivation function called HKDF (Section 8.10.5) to derive enough pseudo-random bits for the two keys. This step is carried out by both the browser and server so that both sides have the keys $k_{b \to s}$ and $k_{s \to b}$.

The TLS record protocol sends data in records whose size is at most 2^{14} bytes. If one side needs to transmit more than 2^{14} bytes, the record protocol fragments the data into multiple records each of size at most 2^{14}. Each party maintains a 64-bit **write sequence number** that is initialized to zero and is incremented by one for every record sent by that party.

TLS 1.3 uses a nonce-based AEAD cipher (E, D) to encrypt a record. Which nonce-based AEAD cipher is used is determined by negotiation during TLS session setup. The AEAD encryption algorithm is given the following arguments:

- secret key: $k_{b \to s}$ or $k_{s \to b}$ depending on whether the browser or server is encrypting.
- plaintext data: up to 2^{14} bytes.
- associated data: empty (zero length).
- nonce (8 bytes or longer): the nonce is computed by (1) padding the encrypting party's 64-bit write sequence number on the left with zeroes to the expected nonce length and (2) XORing this padded sequence number with a random string (called `client_write_iv` or `server_write_iv`, depending on who is encrypting) that was derived from the master secret during session setup and is fixed for the life of the session. TLS 1.3 could have used an equivalent and slightly easier to comprehend method: choose the initial nonce value at random and then increment it sequentially for each record. The method used by TLS 1.3 is a little easier to implement.

The AEAD cipher outputs a ciphertext c which is then formatted into an encrypted TLS record as follows:

`type`	`version`	`length`	ciphertext c

where `type` is a 1-byte record type (handshake record or application data record), `version` is a legacy 2-byte field that is always set to 0301, `length` is a 2-byte field indicating the length of c, and c is the ciphertext. The type, version, and length fields are all sent in the clear. Notice that the nonce is not part of the encrypted TLS record. The recipient computes the nonce by itself.

Why is the initial nonce value random and not simply set to zero? In networking protocols the first message block sent over TLS is usually a fixed public value. If the nonce were set to zero then the first ciphertext would be computed as $c_0 \leftarrow E(k, m_0, d, 0)$ where the adversary knows m_0 and associated data d. This opens up the system to an exhaustive search attack for the key k using a *time-space tradeoff* discussed in Section 18.7. The attack shows that with a large amount of pre-computation and sufficient storage, an attacker can quickly recover k from c_0 with non-negligible advantage — for 128-bit keys, such attacks may be feasible in the not-too-distant future. Randomizing the initial nonce "future proofs" TLS against such attacks.

When a record is received, the receiving party runs the AEAD decryption algorithm to decrypt c. If decryption results in `reject` then the party sends a fatal `bad_record_mac` alert to its peer and shuts down the TLS session.

The length field. In TLS 1.3, as in earlier versions of TLS, the record length is sent in the clear. Several attacks based on traffic analysis exploit record lengths to deduce information about the record contents. For example, if an encrypted TLS record contains one of two images of different size then the length will reveal to an eavesdropper which image was encrypted. Chen et al. [41] show that the lengths of encrypted records can reveal considerable information about private data that a user supplies to a cloud application. They use an online tax filing system as their example. Other works show attacks of this type on many other systems. Since there is no complete solution to this problem, it is often ignored.

When encrypting a TLS record the length field is not part of the associated data and consequently has no integrity protection. The reason is that due to variable length padding, the length of c may not be known before the encryption algorithm terminates. Therefore, the length cannot be given as input to the encryption algorithm. This does not compromise security: a secure AEAD cipher will reject a ciphertext that is a result of tampering with the length field.

Replay prevention. An attacker may attempt to replay a previous record to cause the wrong action at the recipient. For example, the attacker could attempt to make the same purchase order be processed twice, by simply replaying the record containing the purchase order. TLS uses the 64-bit write sequence number to reject such replicated packets. TLS assumes in-order record delivery so that the recipient already knows what sequence number to expect without any additional information in the record. A replicated or out-of-order record will be discarded because the AEAD decryption algorithm will be given the wrong nonce as input causing it to reject the ciphertext.

The cookie cutter attack. TLS provides a streaming interface, where records are sent as soon as they are ready. While replay, re-ordering, and mid-stream deletion of records is prevented by a 64-sequence number, there is no defense against deletion of the *last* record in a stream. In particular, an active attacker can close the network connection mid-way through a session, and to the participants this will look like the conversation ended normally. This can lead to a real-world attack called **cookie cutter**. To see how this works, consider a victim web site and a victim web browser. The victim browser visits a malicious web site that directs the browser to connect to `victim.com`. Say that the encrypted response from the victim site looks as follows:

```
HTTP/1.1 302 Redirect
Location: https://victim.com/path
Set-Cookie: SID=[AuthenticationToken]; secure
Content-Length: 0   \r\n\r\n
```

The first two lines indicate the type of response. Notice that the second line includes a "path" value that is copied from the browser's request. The third line sets a cookie that will be stored on the browser. Here the "secure" attribute indicates that this cookie should only be sent to `victim.com` over an encrypted TLS session. The fourth line indicates the end of the response.

Suppose that in the original browser request, the "path" value is sufficiently long so that the server's response is split across two TLS frames:

```
frame 1:   HTTP/1.1 302 Redirect
           Location: https://victim.com/path
           Set-Cookie: SID=[AuthenticationToken]
```

```
frame 2:   ; secure
           Content-Length: 0  \r\n\r\n
```

The network attacker shuts down the connection after the first frame is sent, so that the second frame never reaches the browser. This causes the browser to mark the cookie as non-secure. Now the attacker directs the browser to the cleartext (http) version of `victim.com`, and the browser will send the SID cookie in the clear, where the attacker can easily read it.

In effect, the adversary was able to make the browser receive a message that the server did not send: the server sent both frames, but the browser only received one and accepted it as a valid message. This is despite proper use of authenticated encryption on every frame.

TLS assumes that the application layer will defend against this attack. In particular, the server's response ends with an end-of-message (EOM) mark in the form of `\r\n\r\n`. The browser should not process an incoming message until it sees the EOM. In practice, however, it is tempting to process headers as soon as they are received, resulting in the vulnerability above. Every application that uses TLS must be aware of this issue, and defend against it using an EOM or equivalent mechanism.

9.9 Case study: an attack on non-atomic decryption in SSH

SSH (secure shell) is a popular command line tool for securely exchanging information with a remote host. SSH is designed to replace (insecure) UNIX tools such as telnet, rlogin, rsh, and rcp. Here we describe a fascinating vulnerability in an older cipher suite used in SSH. This vulnerability is an example of what can go wrong when decryption is not atomic, that is, when the decryption algorithm releases fragments of a decrypted record before verifying integrity of the entire record.

First, a bit of history. The first version of SSH, called SSHv1, was made available in 1995. It was quickly pointed out that SSHv1 suffers from serious design flaws.

- Most notably, SSHv1 provides data integrity by computing a Cyclic Redundancy Check (CRC) of the plaintext and appending the resulting checksum to the ciphertext in the clear. CRC is a simple keyless, linear function — so not only does this directly leak information about the plaintext, it is also not too hard to break integrity either.
- Another issue is the incorrect use of CBC mode encryption. SSHv1 always sets the CBC initial value (IV) to 0. Consequently, an attacker can tell when two SSHv1 packets contain the same prefix. Recall that for CPA security one must choose the IV at random.
- Yet another problem, the same encryption key was used for both directions (user to server and server to user).

To correct these issues, a revised and incompatible protocol called SSHv2 was published in 1996. Session setup results in two keys $k_{u \to s}$, used to encrypt data from the user to the server, and $k_{s \to u}$, used to encrypt data in the reverse direction. Here we focus only how these keys are used for message transport in SSHv2.

SSHv2 encryption. Let us examine an older cipher suite used in SSHv2. SSHv2 combines a CPA-secure cipher with a secure MAC using encrypt-and-MAC (Exercise 9.10) in an attempt to construct a secure AEAD cipher. Specifically, SSHv2 encryption works as follows (Fig. 9.3):

Gray area is encrypted; Boxed area is authenticated by integrity tag

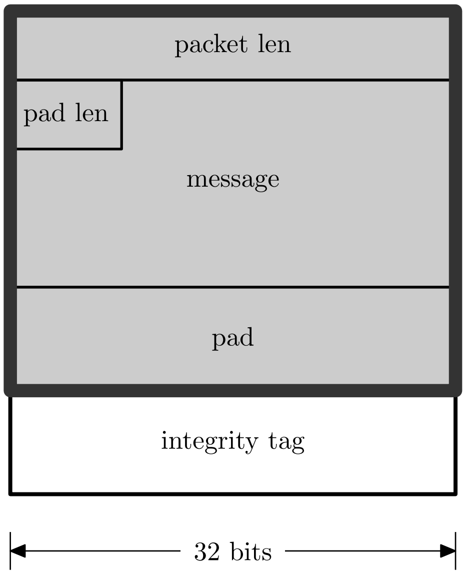


Figure 9.3: An SSHv2 packet

1. **Pad.** Pad the plaintext with *random* bytes so that the total length of

   $$\texttt{plaintext} := \texttt{packet-length} \,\|\, \texttt{pad-length} \,\|\, \texttt{message} \,\|\, \texttt{pad}$$

   is a multiple of the cipher block length (16 bytes for AES). The pad length can be anywhere from 4 bytes to 255 bytes. The packet length field measures the length of the packet in bytes, not including the integrity tag or the packet-length field itself.

2. **Encrypt.** Encrypt the gray area in Fig. 9.3 using AES in randomized CBC mode with either $k_{u \to s}$ or $k_{s \to u}$, depending on the encrypting party. SSHv2 uses a defective version of randomized CBC mode encryption described in Exercise 5.13.

3. **MAC.** A MAC is computed over a `sequence-number` and the plaintext data in the thick box in Fig. 9.3. Here `sequence-number` is a 32-bit sequence number that is initialized to zero for the first packet, and is incremented by one after every packet. SSHv2 can use one of a number of MAC algorithms, but HMAC-SHA1-160 must be supported.

When an encrypted packet is received the decryption algorithm works as follows: first it decrypts the `packet-length` field using either $k_{u \to s}$ or $k_{s \to u}$. Next, it reads that many more packets from the network plus as many additional bytes as needed for the integrity tag. Next it decrypts the rest of the ciphertext and verifies validity of the integrity tag. If valid, it removes the pad and returns the plaintext message.

Although SSH uses encrypt-and-MAC, which is not generally secure, we show in Exercise 9.10 that for certain combinations of cipher and MAC, including the required ones in SSHv2, encrypt-and-MAC provides authenticated encryption.

SSH boundary hiding via length encryption. An interesting aspect of SSHv2 is that the encryption algorithm encrypts the packet length field, as shown in Fig. 9.3. The motivation for this is to ensure that if a sequence of encrypted SSH packets are sent over an insecure network as a stream of bytes, then an eavesdropper should be unable to determine the number of packets sent or their lengths. This is intended to frustrate certain traffic analysis attacks that deduce information about the plaintext from its size.

Hiding message boundaries between consecutive encrypted messages is outside the requirements addressed by authenticated encryption. In fact, many secure AEAD modes do not provide this level of secrecy. TLS 1.0, for example, sends the length of the every record in the clear making it easy to detect boundaries between consecutive encrypted records. Enhancing authenticated encryption to ensure boundary hiding has been formalized by Boldyreva, Degabriele, Paterson, and Stam [27], proposing a number of constructions satisfying the definitions.

An attack on non-atomic decryption. Notice that CBC decryption is done in two steps: first the 32-bit `packet-length` field is decrypted and used to decide how many more bytes to read from the network. Next, the rest of the CBC ciphertext is decrypted.

Generally speaking, AEAD ciphers are not designed to be used this way: plaintext data should not be used until the entire ciphertext decryption process is finished; however, in SSHv2 the decrypted length field is used before its integrity has been verified.

Can this be used to attack SSHv2? A beautiful attack [3] shows how this non-atomic decryption can completely compromise secrecy. Here we only describe the high-level idea, ignoring many details. Suppose an attacker intercepts a 16-byte ciphertext block c and it wants to learn the first four bytes of the decryption of c. It does so by abusing the decryption process as follows: first, it sends the ciphertext block c to the server *as if* it were the first block of a new encrypted packet. The server decrypts c and interprets the first four bytes as a length field ℓ. The server now expects to read ℓ bytes of data from the network before checking the integrity tag. The attacker can slowly send to the server arbitrary bytes, one byte at a time, waiting after each byte to see if the server responds. Once the server reads ℓ bytes it attempts to verify the integrity tag on the bytes it received and this most likely fails causing the server to send back an error message. Thus, once ℓ bytes are read the attacker receives an error message. This tells the attacker the value of ℓ which is what it wanted.

In practice, there are many complications in mounting an attack like this. Nevertheless, it shows the danger of using decrypted data — the length field in this case — before its integrity has been verified. As mentioned above, we refer to [27] for encryption methods that securely hide packet lengths.

A clever traffic analysis attack on SSH. SSHv2 operates by sending one network packet for every user keystroke. This gives rise to an interesting traffic analysis attack reported in [149]. Suppose a network eavesdropper knows that the user is entering a password at his or her keyboard. By measuring timing differences between consecutive packets, the eavesdropper obtains timing information between consecutive keystrokes. This exposes information about the user's password: a large timing gap between consecutive keystrokes reveals information about the keyboard position of the relevant keys. The authors show that this information can significantly speed up an offline password dictionary attack. To make matters worse, password packets are easily identified since applications typically turn off echo during password entry so that password packets do not generate

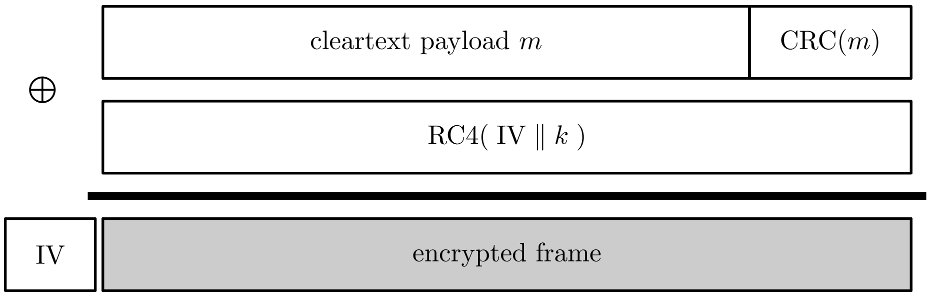


Figure 9.4: WEP Encryption

an echo packet from the server.

Some SSH implementations defend against this problem by injecting randomly timed "dummy" messages to make traffic analysis more difficult. Dummy messages are identified by setting the first message byte to `SSH_MSG_IGNORE` and are ignored by the receiver. The eavesdropper cannot distinguish dummy records from real ones thanks to encryption.

9.10 Case study: 802.11b WEP, a badly broken system

The IEEE 802.11b standard ratified in 1999 defines a protocol for short range wireless communication (WiFi). Security is provided by a Wired Equivalent Privacy (WEP) encapsulation of 802.11b data frames. The design goal of WEP is to provide data privacy at the level of a wired network. WEP, however, completely fails on this front and gives us an excellent case study illustrating how a weak design can lead to disastrous results.

When WEP is enabled, all members of the wireless network share a long term secret key k. The standard supports either 40-bit keys or 128-bit keys. The 40-bit version complies with US export restrictions that were in effect at the time the standard was drafted. We will use the following notation to describe WEP:

- WEP encryption uses the RC4 stream cipher. We let RC4(s) denote the pseudo random sequence generated by RC4 given the seed s.
- We let CRC(m) denote the 32-bit CRC checksum of a message $m \in \{0,1\}^*$. The details of CRC are irrelevant for our discussion and it suffices to view CRC as some fixed function from bit strings to $\{0,1\}^{32}$.

Let m be an 802.11b cleartext frame. The first few bits of m encode the length of m. To encrypt an 802.11b frame m the sender picks a 24-bit IV and computes:

$$c \leftarrow \big(m \parallel \mathrm{CRC}(m)\big) \;\oplus\; \mathrm{RC4}(\mathrm{IV} \parallel k)$$
$$c_{\text{full}} \leftarrow (\mathrm{IV},\; c)$$

The WEP encryption process is shown in Fig. 9.4. The receiver decrypts by first computing $c \oplus \mathrm{RC4}(\mathrm{IV} \parallel k)$ to obtain a pair (m, s). The receiver accepts the frame if $s = \mathrm{CRC}(m)$ and rejects it otherwise.

Attack 1: IV collisions. The designers of WEP understood that a stream cipher key should never be reused. Consequently, they used the 24-bit IV to derive a per-frame key $k_{\rm f} := \text{IV} \parallel k$. The standard, however, does not specify how to choose the IVs and many implementations do so poorly. We say that an IV collision occurs whenever a wireless station happens to send two frames, say frame number i and frame number j, encrypted using the same IV. Since IVs are sent in the clear, an eavesdropper can easily detect IV collisions. Moreover, once an IV collision occurs the attacker can use the two-time pad attack discussed in Section 3.3.1 to decrypt both frames i and j.

So, how likely is an IV collision? By the birthday paradox, an implementation that chooses a random IV for each frame will cause an IV collision after only an expected $\sqrt{2^{24}} = 2^{12} = 4096$ frames. Since each frame body is at most 1156 bytes, a collision will occur after transmitting about 4MB on average.

Alternatively, an implementation could generate the IV using a counter. The implementation will exhaust the entire IV space after 2^{24} frames are sent, which will take about a day for a wireless access point working at full capacity. Even worse, several wireless cards that use the counter method reset the counter to 0 during power-up. As a result, these cards will frequently reuse low value IVs, making the traffic highly vulnerable to a two-time pad attack.

Attack 2: related keys. A far more devastating attack on WEP encryption results from the use of related RC4 keys. In Chapter 3 we explained that a new and *random* stream cipher key must be chosen for every encrypted message. WEP, however, uses keys $1 \parallel k$, $2 \parallel k, \dots$ which are all closely related — they all have the same suffix k. RC4 was never designed for such use, and indeed, is completely insecure in these settings. Fluhrer, Mantin, and Shamir [63] showed that after about a million WEP frames are sent, an eavesdropper can recover the entire long term secret key k. The attack was implemented by Stubblefield, Ioannidis, and Rubin [151] and is now available in a variety of hacking tools such as WepCrack and AirSnort.

Generating per frame keys should have been done using a PRF, for example, setting the key for frame i to $k_i := F(k, \text{IV})$ — the resulting keys would be indistinguishable from random, independent keys. Of course, while this approach would have prevented the related keys problem, it would not solve the IV collision problem discussed above, or the malleability problem discussed next.

Attack 3: malleability. Recall that WEP attempts to provide authenticated encryption by using a CRC checksum for integrity. In a sense, WEP uses the MAC-then-encrypt method, but it uses CRC instead of a MAC. We show that despite the encryption step, this construction utterly fails to provide ciphertext integrity.

The attack uses the linearity of CRC. That is, given $\text{CRC}(m)$ for some message m, it is easy to compute $\text{CRC}(m \oplus \Delta)$ for any Δ. More precisely, there is a public function L such that for any m and $\Delta \in \{0,1\}^\ell$ we have that

$$\text{CRC}(m \oplus \Delta) = \text{CRC}(m) \oplus L(\Delta)$$

This property enables an attacker to make arbitrary modifications to a WEP ciphertext without ever being detected by the receiver. Let c be a WEP ciphertext, namely

$$c = \big(m, \text{CRC}(m)\big) \;\oplus\; \text{RC4}(\text{IV} \parallel k)$$

For any $\Delta \in \{0,1\}^{\ell}$, an attacker can create a new ciphertext $c' \leftarrow c \oplus \big(\Delta, L(\Delta)\big)$, which satisfies

$$\begin{aligned} c' \;&=\; \mathrm{RC4}(\mathrm{IV} \,\|\, k) \;\oplus\; \big(m,\ \mathrm{CRC}(m)\big) \;\oplus\; \big(\Delta,\ L(\Delta)\big) = \\ &\; \mathrm{RC4}(\mathrm{IV} \,\|\, k) \;\oplus\; \big(m \oplus \Delta,\ \ \mathrm{CRC}(m) \oplus L(\Delta)\big) = \\ &\; \mathrm{RC4}(\mathrm{IV} \,\|\, k) \;\oplus\; \big(m \oplus \Delta,\ \ \mathrm{CRC}(m \oplus \Delta)\big) \end{aligned}$$

Hence, c' decrypts without errors to $m \oplus \Delta$. We see that given the encryption of m, an attacker can create a valid encryption of $m \oplus \Delta$ for any Δ of his choice. We explained in Section 3.3.2 that this can lead to serious attacks.

Attack 4: Chosen ciphertext attack. The protocol is vulnerable to a chosen ciphertext attack called **chop-chop** that lets the attacker decrypt an encrypted frame of its choice. We describe a simple version of this attack in Exercise 9.5.

Attack 5: Denial of Service. We briefly mention that 802.11b suffers from a number of serious Denial of Service (DoS) attacks. For example, in 802.11b a wireless client sends a “disassociate” message to the wireless station once the client is done using the network. This allows the station to free memory resources allocated to that client. Unfortunately, the “disassociate” message is unauthenticated, allowing anyone to send a disassociate message on behalf of someone else. Once disassociated, the victim will take a few seconds to re-establish the connection to the base station. As a result, by sending a single “disassociate” message every few seconds, an attacker can prevent a computer of their choice from connecting to the wireless network. These attacks are implemented in 802.11b tools such as `Void11`.

802.11i. Following the failures of the 802.11b WEP protocol, a new standard called 802.11i was ratified in 2004. 802.11i provides authenticated encryption using a MAC-then-encrypt mode called CCM. In particular, CCM uses (raw) CBC-MAC for the MAC and counter mode for encryption. Both are implemented in 802.11i using AES as the underlying PRF. CCM was adopted by NIST as a federal standard [128].

9.11 A fun application: private information retrieval

To be written.

9.12 Notes

Citations to the literature to be added.

9.13 Exercises

9.1 (AE-security: simple examples). Let (E, D) be an AE-secure cipher. Consider the following derived ciphers:

(a) $E_1(k,m) := \big(E(k,m),\ E(k,m)\big);\quad D_1\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } D(k,c_1) = D(k,c_2) \\ \textsf{reject} & \text{otherwise} \end{cases}$

(b) $E_2(k,m) := \big\{c \leftarrow E(k,m),\ \text{output } (c,c)\big\};\quad D_2\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } c_1 = c_2 \\ \textsf{reject} & \text{otherwise} \end{cases}$

Show that part (b) is AE-secure, but part (a) is not.

9.2 (AE-security: some insecure constructions). Let (E,D) be a CPA-secure cipher defined over $(\mathcal{K},\mathcal{M},\mathcal{C})$ and let $H_1 : \mathcal{M} \to \mathcal{T}$ and $H_2 : \mathcal{C} \to \mathcal{T}$ be collision resistant hash functions. Define the following two ciphers:

(a) $E_1(k,m) := \big\{c \xleftarrow{\text{R}} E(k,m),\ \text{output } \big(c, H_1(m)\big)\big\};$

$$D_1\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } H_1(D(k,c_1)) = c_2 \\ \textsf{reject} & \text{otherwise} \end{cases}$$

(b) $E_2(k,m) := \big\{c \xleftarrow{\text{R}} E(k,m),\ \text{output } \big(c, H_2(c)\big)\big\};$

$$D_2\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } H_2(c_1) = c_2 \\ \textsf{reject} & \text{otherwise} \end{cases}$$

Show that both ciphers are not AE-secure.

9.3 (An Android Keystore Attack). Let (E,D) be a secure block cipher defined over $(\mathcal{K},\mathcal{X})$, and let $(E_{\text{cbc}}, D_{\text{cbc}})$ be the cipher derived from (E,D) using randomized CBC mode, as in Section 5.4.3. Let $H : \mathcal{X}^{\leq L} \to \mathcal{X}$ be a collision resistant hash function. Consider the following attempt at building an AE-secure cipher defined over $(\mathcal{K},\ \mathcal{X}^{\leq L},\ \mathcal{X}^{\leq L+2})$:

$$E'(k,m) := E_{\text{cbc}}\big(k,\ (H(m),m)\big)\ ; \qquad D'(k,c) := \left\{ \begin{array}{l} (t,m) \leftarrow D_{\text{cbc}}(k,c) \\ \text{if } t = H(m) \text{ output } m, \text{ otherwise } \textsf{reject} \end{array} \right\}$$

Show that (E',D') is not AE-secure by giving a chosen-ciphertext attack on it. This construction was used to protect secret keys in the Android KeyStore. The chosen-ciphertext attack resulted in a compromise of the key store [138].

9.4 (Redundant message encoding does not give AE***).*** The attack in the previous exercise can be generalized if instead of using CBC encryption as the underlying cipher, we use randomized counter mode, as in Section 5.4.2. Let $(E_{\text{ctr}}, D_{\text{ctr}})$ be such a counter-mode cipher, and assume that its message space is $\{0,1\}^{\ell'}$. Let $f : \{0,1\}^{\ell} \to \{0,1\}^{\ell'}$ be a one-to-one function, so that $\ell' \geq \ell$. Let $g : \{0,1\}^{\ell'} \to \{0,1\}^{\ell} \cup \{\bot\}$ be the inverse of f in the sense that $g(m') = m$ whenever $m' = f(m)$ for some m, and $g(m') = \bot$ if m' is not in the image of f. Intuitively, f represents an “error detecting code”: a message $m \in \{0,1\}^{\ell}$ is “encoded” as $m' = f(m)$. If m' gets modified into a value $\tilde{m}'$, this modification will be detected if $g(\tilde{m}') = \bot$. Now define a new cipher (E_2, D_2) with message space $\{0,1\}^{\ell}$ as follows:

$$E_2(k,m) := E_{\text{ctr}}\big(k,\ f(m)\big)\ ; \qquad D_1(k,c) := \left\{ \begin{array}{l} m' \leftarrow D_{\text{ctr}}(k,c) \\ \text{if } g(m') \neq \bot \text{ output } g(m'), \text{ otherwise } \textsf{reject} \end{array} \right\}$$

Show that (E_2, D_2) is not AE-secure by giving a chosen-ciphertext attack on it.

9.5 (Chop-chop attack). The parity bit b for a message $m \in \{0,1\}^*$ is just the XOR of all the bits in m. After appending the parity bit, the message $m' = m \parallel b$ has the property that the XOR of all the bits is zero. Parity bits are sometimes used as a very simple form of error detection. They are meant to provide a little protection against low-probability, random errors: if a single bit of m' gets flipped, this can be detected, since the XOR of the bits of the corrupted m' will now be one.

Consider a cipher where messages are variable length bit strings, and encryption is done using randomized counter mode without any padding. No MAC is used, but before the plaintext is encrypted, the sender appends a parity bit to the end of the plaintext. After the receiver decrypts, it checks the parity bit and returns either the plaintext (with the parity bit removed) or reject.

Design a chosen-ciphertext attack that recovers the complete plaintext of every encrypted message. Your attack should work even if the adversary learns only one bit for every chosen-ciphertext query c; it only learns if the decryption of c succeeded or resulted in reject, and learns nothing else about c.

Hint: Use the fact that the system encrypts variable length messages.

Remark: A variant of this attack, called **chopchop**, was used successfully against encryption in the 802.11b protocol. The name is a hint for how the attack works. Note that the previous exercise already tells us that this scheme is not CCA-secure, but the attack in this exercise is much more devastating.

9.6 (Nested encryption). Let (E, D) be an AE-secure cipher. Consider the following derived cipher (E', D'):

$$E'\big((k_1, k_2), m\big) := E\big(k_2, E(k_1, m)\big); \qquad D'\big((k_1, k_2),\ c\big) := \begin{cases} D\big(k_1, D(k_2, c)\big) & \text{if } D(k_2, c) \neq \mathsf{reject} \\ \mathsf{reject} & \text{otherwise} \end{cases}$$

(a) Show that (E', D') is AE-secure even if the adversary knows k_1, but not k_2.

(b) Show that (E', D') is not AE-secure if the adversary knows k_2 but not k_1.

(c) Design a cipher built from (E, D) where keys are pairs $(k_1, k_2) \in \mathcal{K}^2$ and the cipher remains AE-secure even if the adversary knows one of the keys, but not the other.

9.7 (A format oracle attack). Let $\mathcal{E}$ be an arbitrary CPA-secure cipher, and assume that the key space for $\mathcal{E}$ is $\{0, 1\}^n$. Show how to "sabotage" $\mathcal{E}$ to obtain another cipher $\mathcal{E}'$ such that $\mathcal{E}'$ is still CPA secure, but $\mathcal{E}'$ is insecure against chosen ciphertext attack, in the following sense. In the attack, the adversary is allowed to make several decryption queries, such that in each query, the adversary only learns whether the result of the decryption was reject or not. Design an adversary that makes a series of decryption queries as above, and then outputs the secret key in its entirety.

9.8 (Choose independent keys). Let us see an example of a CPA-secure cipher and a secure MAC that are insecure when used in encrypt-then-MAC when the same secret key k is used for both the cipher and the MAC. Let (E, D) be a block cipher defined over $(\mathcal{K}, \mathcal{X})$ where $\mathcal{X} = \{0, 1\}^n$ and $|\mathcal{X}|$ is super-poly. Consider randomized CBC mode encryption built from (E, D) as the CPA-secure cipher for single block messages: an encryption of $m \in \mathcal{X}$ is the pair $c := (r,\ E(k,\ r \oplus m))$ where r is the random IV. Use F_{CBC} (from Fig. 6.3a) built from (E, D) as the secure MAC. This MAC is secure in this context because it is only being applied to fixed length messages (messages in $\mathcal{X}^2$): the tag on a ciphertext $c \in \mathcal{X}^2$ is $t := E\big(k,\ E(k, c[0]) \oplus c[1]\big)$. Show that using the same key k for both the cipher and the MAC in encrypt-then-MAC results in a cipher that does not provide authenticated encryption. Both CPA security and ciphertext integrity can be defeated.

9.9 (MAC-then-encrypt). Prove that MAC-then-encrypt provides authenticated encryption when the underlying cipher is randomized CBC mode encryption and the MAC is a secure MAC. Specifically, assume that the underlying cipher works on blocks of a fixed size, a message m is a sequence of full blocks, and the tag t for the MAC is one full block, so that the message that is CBC-encrypted is the block sequence $m \parallel t$, and no CBC padding is needed.

9.10 (An AEAD from encrypt-and-MAC). Let (E, D) be randomized counter mode encryption defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where the underlying secure PRF has domain $\mathcal{X}$. We let $E(k, m; r)$ denote the encryption of message m with key k using $r \in \mathcal{X}$ as the IV. Let F be a secure PRF defined over $(\mathcal{K},\ (\mathcal{M} \times \mathcal{D} \times \mathcal{N}),\ \mathcal{X})$. Show that the following cipher (E_1, D_1) is a secure nonce-based AEAD cipher assuming $|\mathcal{X}|$ is super-poly.

$$E_1\big((k_{\mathrm{e}}, k_{\mathrm{m}}),\ m,\ d,\ \mathcal{N}\,\big) := \big\{t \leftarrow F\big(k_{\mathrm{m}},\ (m, d, \mathcal{N})\big),\ c \overset{\mathrm{R}}{\leftarrow} E(k_{\mathrm{c}}, m; t),\ \text{output } (c, t)\ \big\}$$

$$D_1\big((k_{\mathrm{e}}, k_{\mathrm{m}}),\ (c, t),\ d,\ \mathcal{N}\,\big) := \left\{\begin{array}{l} m \leftarrow D(k_{\mathrm{e}}, c; t) \\ \text{if } F\big(k_{\mathrm{m}},\ (m, d, \mathcal{N})\big) \neq t \text{ output } \mathsf{reject}, \text{ otherwise output } m \end{array}\right\}$$

This method is loosely called encrypt-and-MAC because the message m is both encrypted by the cipher and is the input to the MAC signing algorithm, which here is a PRF.

Discussion: This construction is related to the authenticated SIV cipher (Exercise 9.11) and offers similar **nonce re-use resistance**. One down-side of this system is that the tag t cannot be truncated as one often does with a PRF-based MAC.

9.11 (Authenticated SIV). We discuss a modification of the SIV construction, introduced in Exercise 5.8, that provides ciphertext integrity without enlarging the ciphertext any further. We call this the **authenticated SIV** construction. With $\mathcal{E} = (E, D)$, F, and $\mathcal{E}' = (E', D')$ as in Exercise 5.8, we define $\mathcal{E}'' = (E', D'')$, where

$$D''\big((k, k'),\ c\big) := \left\{\begin{array}{l} m \leftarrow D(k, c) \\ \text{if } E'((k, k'), m) = c \text{ output } m, \text{ otherwise output } \mathsf{reject} \end{array}\right\}$$

Assume that $|\mathcal{R}|$ is super-poly and that for every fixed key $k \in \mathcal{K}$ and $m \in \mathcal{M}$, the function $E(k, m; \cdot) : \mathcal{R} \to \mathcal{C}$ is one to one (which holds for counter and CBC mode encryption). Show that $\mathcal{E}''$ provides ciphertext integrity.

Note: Since the encryption algorithm of $\mathcal{E}''$ is the same as that of $\mathcal{E}'$ we know that $\mathcal{E}''$ is deterministic CPA-secure, assuming that $\mathcal{E}$ is CPA-secure (as was shown in Exercise 5.8).

9.12 (Constructions based on strongly secure block ciphers). Let (E, D) be a block cipher defined over $(\mathcal{K},\ \mathcal{M} \times \mathcal{R})$.

(a) As in Exercise 5.6, let (E', D') be defined as

$$E'(k, m) := \big\{r \overset{\mathrm{R}}{\leftarrow} \mathcal{R},\ c \overset{\mathrm{R}}{\leftarrow} E\big(k,\ (m, r)\big),\ \text{output } c\big\}$$

$$D'(k, c) := \big\{(m, r') \leftarrow D(k, c),\ \text{output } m\big\}$$

Show that (E', D') is CCA-secure provided (E, D) is a *strongly secure* block cipher and $1/|\mathcal{R}|$ is negligible. This is an example of a CCA-secure cipher that clearly does not provide ciphertext integrity.

(b) Let (E'', D'') be defined as

$$E''(k, m) := \big\{ r \xleftarrow{R} \mathcal{R},\ c \xleftarrow{R} E\big(k,\ (m, r)\big),\ \text{output } (c, r) \big\}$$

$$D''\big(k, (c, r)\big) := \left\{ \begin{array}{l} (m, r') \leftarrow D(k, c) \\ \text{if } r = r' \text{ output } m, \text{ otherwise output } \mathsf{reject} \end{array} \right\}$$

This cipher is defined over $\big(\mathcal{K},\ \mathcal{M},\ (\mathcal{M}\times\mathcal{R})\times\mathcal{R}\big)$. Show that (E'', D'') is AE-secure provided (E, D) is a strongly secure block cipher and $1/|\mathcal{R}|$ is negligible.

(c) Suppose that $0 \in \mathcal{R}$ and we modify algorithms E'' and D'' to work as follows:

$$\tilde{E}''(k, m) := \big\{ r \leftarrow 0,\ c \xleftarrow{R} E\big(k,\ (m, r)\big),\ \text{output } c \big\}$$

$$\tilde{D}''\big(k, c\big) := \left\{ \begin{array}{l} (m, r') \leftarrow D(k, c) \\ \text{if } r' = 0 \text{ output } m, \text{ otherwise output } \mathsf{reject} \end{array} \right\}$$

Show that $(\tilde{E}'', \tilde{D}'')$ is *one-time* AE-secure provided (E, D) is a strongly secure block cipher, and $1/|\mathcal{R}|$ is negligible.

9.13 (MAC from encryption). Let (E, D) be a cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. Define the following MAC system (S, V) also defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$:

$$S(k, m) := E(k, m); \qquad V(k, m, t) := \begin{cases} \mathsf{accept} & \text{if } D(k, t) = m \\ \mathsf{reject} & \text{otherwise} \end{cases}$$

Show that if (E, D) has ciphertext integrity then (S, V) is a secure MAC system.

9.14 (GCM analysis). Give a complete security analysis of GCM (see Section 9.7). Show that it is nonce-based AEAD secure assuming the security of the underlying block cipher as a PRF and that GHASH is an XOR-DUF. Start out with the easy case when the nonce is 96-bits. Then proceed to the more general case where GHASH may be applied to the nonce to compute x.

9.15 (Plaintext integrity). Consider a weaker notion of integrity called **plaintext integrity**, or simply PI. The PI game is identical to the CI game except that the winning condition is relaxed to:

- $D(k, c) \neq \mathsf{reject}$, and
- $D(k, c) \notin \{m_1, m_2, \ldots\}$

Prove that the following holds:

(a) Show that MAC-then-Encrypt is both CPA and PI secure.

Note: The MAC-then-Encrypt counter-example (Section 9.4.2) shows that a system that is CPA and PI secure is not CCA-secure (and, therefore, not AE-secure).

(b) Prove that a system that is CCA- and PI-secure is also AE-secure. The proof only needs a weak version of CCA, namely where the adversary issues a single decryption query and is told whether the ciphertext is accepted or rejected. Also, you may assume a super-poly-sized message space.

9.16 (Encrypted UHF MAC). Let H be a hash function defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{X})$ and (E, D) be a cipher defined over $(\mathcal{K}_E, \mathcal{X}, \mathcal{C})$. Define the **encrypted UHF MAC system** $\mathcal{I} = (S, V)$ as follows: for key (k_1, k_2) and message $m \in \mathcal{M}$ define

$$S\big((k_1, k_2),\ m\big) := E\big(k_1,\ H(k_2, m)\big)$$
$$V\big((k_1, k_2),\ m,\ c\big) := \begin{cases} \textsf{accept} & \text{if } H(k_2, m) = D(k_1, c), \\ \textsf{reject} & \text{otherwise.} \end{cases}$$

Show that $\mathcal{I}$ is a secure MAC system assuming H is a computational UHF and (E, D) provides authenticated encryption. Recall from Section 7.4 that CPA security of (E, D) is insufficient for this MAC system to be secure.

9.17 (Simplified OCB mode). OCB is an elegant and efficient AE cipher built from a tweakable block cipher (as defined in Exercise 4.11). Let (E, D) be a tweakable block cipher defined over $(\mathcal{K}, \mathcal{X}, \mathcal{T})$ where $\mathcal{X} := \{0,1\}^n$ and the tweak set is $\mathcal{T} := \mathcal{N} \times \{-\ell, \dots, \ell\}$. Consider the following nonce-based cipher (E', D') with key space $\mathcal{K}$, message space $\mathcal{X}^{\leq \ell}$, ciphertext space $\mathcal{X}^{\ell+1}$, and nonce space $\mathcal{N}$. For simplicity, the cipher does not support associated data.

```
$E'(k, m, \mathcal{N}) :=$
    create (uninitialized) $c \in \mathcal{X}^{|m|}$
    checksum $\leftarrow 0^n$
    for $i = 0, \dots, |m| - 1$ :
        $c[i] \leftarrow E\big(k,\ m[i],\ (\mathcal{N}, i+1)\big)$
        checksum $\leftarrow$ checksum $\oplus\ m[i]$
    $t \leftarrow E\big(k,$ checksum, $(\mathcal{N}, -|m|)\big)$
    output $(c, t)$
```

```
$D'(k, (c, t), \mathcal{N}) :=$
    create (uninitialized) $m \in \mathcal{X}^{|c|}$
    checksum $\leftarrow 0^n$
    for $i = 0, \dots, |c| - 1$ :
        $m[i] \leftarrow D\big(k,\ c[i],\ (\mathcal{N}, i+1)\big)$
        checksum $\leftarrow$ checksum $\oplus\ m[i]$
    $t' \leftarrow E\big(k,$ checksum, $(\mathcal{N}, -|c|)\big)$
    if $t = t'$ output $m$, else reject
```

(a) Prove that (E', D') is a nonce-based AE-secure cipher assuming (E, D) is a strongly secure tweakable block cipher and $|\mathcal{X}|$ is super-poly.

(b) Show that if t were computed as $t \leftarrow E\big(k,\ \text{checksum},\ (\mathcal{N}, 0)\big)$ then the scheme would be insecure: it would have no ciphertext integrity.

9.18 (Non-committing encryption). Let (E, D) be a cipher. We say that the cipher is **non-committing** if an adversary can find a ciphertext c and two keys k_0, k_1 such that c decrypts successfully under both k_0 and k_1 and the resulting plaintexts are different. The non-committing property means that the adversary can transmit c, but if he or she are later required to reveal the decryption key, say for an internal audit, the adversary can "open" the ciphertext in two different ways.

(a) Let (E, D) be an encrypt-then-MAC AE-secure cipher where the underlying encryption is randomized counter mode built using a secure PRF. Show that (E, D) is non-committing.

(b) Show that GCM mode encryption is non-committing.

(c) Describe a simple way in which the ciphers from parts (a) and (b) can be made committing.

9.19 (Middlebox encryption). In this exercise we develop a mode of encryption that lets a middlebox placed between the sender and recipient inspect all traffic in the clear, but prevents the middlebox from modifying traffic en-route. This is often needed in enterprise settings where a middlebox ensures that no sensitive information is accidentally sent out. Towards this goal let us define a middlebox cipher as a tuple of four algorithms (E, D, D', K) where $E(k, m)$ and $D(k, c)$ are the usual encryption and decryption algorithms used by the end-points, K is an algorithm that derives a sub-key k' from the primary key k (i.e., $k' \xleftarrow{\text{R}} K(k)$), and $D'(k', c)$ is the decryption algorithm used by the middlebox with the sub-key k'. We require the usual correctness properties: $D(k, c)$ and $D'(k', c)$ output m whenever $c \xleftarrow{\text{R}} E(k, m)$ and $k' \xleftarrow{\text{R}} K(k)$.

(a) Security for a middlebox cipher (E, D, D', K) captures our desired confidentiality and integrity requirements. In particular, we say that a middlebox cipher is secure if the following three properties hold:

  (i) the cipher is secure against a chosen plaintext attack (CPA security) when the adversary knows nothing about k,

  (ii) the cipher provides ciphertext integrity with respect to the decryption algorithm $D'(k', \cdot)$, when the adversary knows nothing about k, and

  (iii) the cipher provides ciphertext integrity with respect to the decryption algorithm $D(k, \cdot)$, when the adversary is given a sub-key $k' \xleftarrow{\text{R}} K(k)$, but again knows nothing about k.

  The second requirement says that the middlebox will only decrypt authentic ciphertexts. The third requirement says that the receiving end-point will only decrypt authentic ciphertexts, even if the middlebox is corrupt.

  Formalize these requirements as attack games.

(b) Give a construction that satisfies your definition from part (a). You can use an AE secure cipher and a secure MAC as building blocks.

Part II

Public key cryptography

Chapter 10

Public key tools

We begin our discussion of public-key cryptography by introducing several basic tools that will be used in the remainder of the book. The main applications for these tools will emerge in the next few chapters where we use them for public-key encryption, digital signatures, and key exchange. Since we use some basic algebra and number theory in this chapter, the reader is advised to first briefly scan through Appendix A.

We start with a simple toy problem: generating a shared secret key between two parties so that a passive eavesdropping adversary cannot feasibly guess their shared key. The adversary can listen in on network traffic, but cannot modify messages en-route or inject his own messages. In a later chapter we develop the full machinery needed for key exchange in the presence of an active attacker who may tamper with network traffic.

At the onset we emphasize that security against eavesdropping is typically not sufficient for real world-applications, since an attacker capable of listening to network traffic is often also able to tamper with it; nevertheless, this toy eavesdropping model is a good way to introduce the new public-key tools.

10.1 A toy problem: anonymous key exchange

Two users, Alice and Bob, who have never met before talk on the phone. They are worried that an eavesdropper is listening to their conversation and hence they wish to encrypt the session. Since Alice and Bob have never met before they have no shared secret key with which to encrypt the session. Thus, their initial goal is to generate a shared secret unknown to the adversary. They may later use this secret as a session-key for secure communication. To do so, Alice and Bob execute a protocol where they take turns in sending messages to each other. The eavesdropping adversary can hear all these messages, but cannot change them or inject his own messages. At the end of the protocol Alice and Bob should have a secret that is unknown to the adversary. The protocol itself provides no assurance to Alice that she is really talking to Bob, and no assurance to Bob that he is talking to Alice — in this sense, the protocol is “anonymous.”

More precisely, we model Alice and Bob as communicating machines. A **key exchange protocol** P is a pair of probabilistic machines (A, B) that take turns in sending messages to each other. At the end of the protocol, when both machines terminate, they both obtain the same value k. A **protocol transcript** T_P is the sequence of messages exchanged between the parties in one execution of the protocol. Since A and B are probabilistic machines, we obtain a different transcript every time we run the protocol. Formally, the transcript T_P of protocol P is a random variable, which is a function of the random bits generated by A and B. The eavesdropping adversary $\mathcal{A}$ sees the entire transcript T_P and its goal is to figure out the secret k. We define security of a key exchange protocol using the following game.

Attack Game 10.1 (Anonymous key exchange). For a key exchange protocol $P = (A, B)$ and a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger runs the protocol between A and B to generate a shared key k and transcript T_P. It gives T_P to $\mathcal{A}$.
- $\mathcal{A}$ outputs a guess $\hat{k}$ for k.

We define $\mathcal{A}$'s advantage, denoted $\mathrm{AnonKE}\mathsf{adv}[\mathcal{A}, P]$, as the probability that $\hat{k} = k$. □

Definition 10.1. *We say that an anonymous key exchange protocol P is secure against an eavesdropper if for all efficient adversaries $\mathcal{A}$, the quantity* $\mathrm{AnonKE}\mathsf{adv}[\mathcal{A}, P]$ *is negligible.*

This definition of security is extremely weak, for three reasons. First, we assume the adversary is unable to tamper with messages. Second, we only guarantee that the adversary cannot guess k in its entirety. This does not rule out the possibility that the adversary can guess, say, half the bits of k. If we are to use k as a secret session key, the property we would really like is that k is indistinguishable from a truly random key. Third, the protocol provides no assurance of the identities of the participants. We will strengthen Definition 10.1 to meet these stronger requirements in Chapter 21.

Given all the tools we developed in Part 1, it is natural to ask if anonymous key exchange can be done using an arbitrary secure symmetric cipher. The answer is yes, it can be done as we show in Section 10.8, but the resulting protocol is highly inefficient. To develop efficient protocols we must first introduce a few new tools.

10.2 One-way trapdoor functions

In this section, we introduce a tool that will allow us to build an efficient and secure key exchange protocol. In Section 8.11, we introduced the notion of a one-way function. This is a function $F : \mathcal{X} \to \mathcal{Y}$ that is easy to compute, but hard to invert. As we saw in Section 8.11, there are a number of very efficient functions that are plausibly one-way. One-way functions, however, are not sufficient for our purposes. We need one-way functions with a special feature, called a **trapdoor**. A trapdoor is a secret that allows one to efficiently invert the function; however, without knowledge of the trapdoor, the function remains hard to invert.

Let us make this notion more precise.

Definition 10.2 (Trapdoor function scheme). *Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets. A **trapdoor function scheme** $\mathcal{T}$, defined over $(\mathcal{X}, \mathcal{Y})$, is a triple of algorithms (G, F, I), where*

- *G is a probabilistic key generation algorithm that is invoked as $(pk, sk) \xleftarrow{\mathrm{R}} G()$, where pk is called a **public key** and sk is called a **secret key**.*
- *F is a deterministic algorithm that is invoked as $y \leftarrow F(pk, x)$, where pk is a public key (as output by G) and x lies in $\mathcal{X}$. The output y is an element of $\mathcal{Y}$.*

- *I is a deterministic algorithm that is invoked as $x \leftarrow I(sk, y)$, where sk is a secret key (as output by G) and y lies in $\mathcal{Y}$. The output x is an element of $\mathcal{X}$.*

Moreover, the following ***correctness property*** *should be satisfied: for all possible outputs (pk, sk) of $G()$, and for all $x \in \mathcal{X}$, we have $I(sk,\ F(pk, x)\ ) = x$.*

Observe that for every pk, the function $F(pk, \cdot)$ is a function from $\mathcal{X}$ to $\mathcal{Y}$. The correctness property says that sk is the trapdoor for inverting this function; note that this property also implies that the function $F(pk, \cdot)$ is one-to-one. Note that we do not insist that $F(pk, \cdot)$ maps $\mathcal{X}$ onto $\mathcal{Y}$. That is, there may be elements $y \in \mathcal{Y}$ that do not have any preimage under $F(pk, \cdot)$. For such y, we make no requirements on algorithm I — it can return some arbitrary element $x \in \mathcal{X}$ (one might consider returning a special reject symbol in this case, but it simplifies things a bit not to do this).

In the special case where $\mathcal{X} = \mathcal{Y}$, then $F(pk, \cdot)$ is not only one-to-one, but onto. That is, $F(pk, \cdot)$ is a *permutation on the set $\mathcal{X}$*. In this case, we may refer to (G, F, I) as a **trapdoor permutation scheme** defined over $\mathcal{X}$.

The basic security property we want from a trapdoor permutation scheme is a one-wayness property, which basically says that given pk and $F(pk, x)$ for random $x \in \mathcal{X}$, it is hard to compute x without knowledge of the trapdoor sk. This is formalized in the following game.

Attack Game 10.2 (One-way trapdoor function scheme). For a given trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger computes

  $$(pk, sk) \xleftarrow{R} G(), \quad x \xleftarrow{R} \mathcal{X}, \quad y \leftarrow F(pk, x)$$

  and sends (pk, y) to the adversary.

- The adversary outputs $\hat{x} \in \mathcal{X}$.

We define the adversary's advantage in inverting $\mathcal{T}$, denoted $\mathrm{OW}\mathsf{adv}[\mathcal{A}, \mathcal{T}]$, to be the probability that $\hat{x} = x$. □

Definition 10.3. *We say that a trapdoor function scheme $\mathcal{T}$ is* ***one way*** *if for all efficient adversaries $\mathcal{A}$, the quantity $\mathrm{OW}\mathsf{adv}[\mathcal{A}, \mathcal{T}]$ is negligible.*

Note that in Attack Game 10.2, since the value x is uniformly distributed over $\mathcal{X}$ and $F(pk, \cdot)$ is one-to-one, it follows that the value $y := F(pk, x)$ is uniformly distributed over the image of $F(pk, \cdot)$. In the case of a trapdoor permutation scheme, where $\mathcal{X} = \mathcal{Y}$, the value of y is uniformly distributed over $\mathcal{X}$.

10.2.1 Key exchange using a one-way trapdoor function scheme

We now show how to use a one-way trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$, to build a secure anonymous key exchange protocol. The protocol runs as follows, as shown in Fig. 10.1:

- Alice computes $(pk, sk) \xleftarrow{R} G()$, and sends pk to Bob.

- Upon receiving pk from Alice, Bob computes $x \xleftarrow{R} \mathcal{X}, y \leftarrow F(pk, x)$, and sends y to Alice.

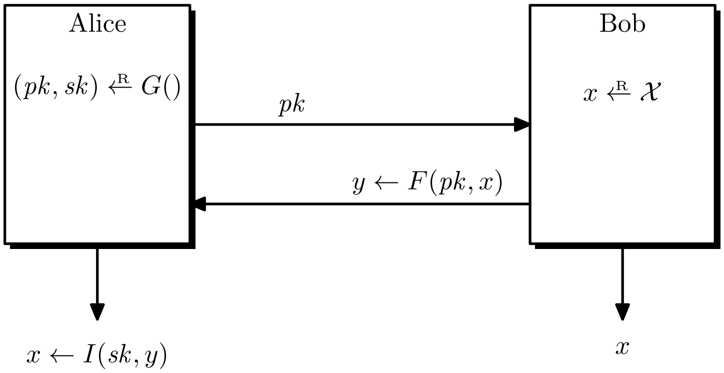


Figure 10.1: Key exchange using a trapdoor function scheme

- Upon receiving y from Bob, Alice computes $x \leftarrow I(sk, y)$.

The correctness property of the trapdoor function scheme guarantees that at the end of the protocol, Alice and Bob have the same value x — this is their shared, secret key. Now consider the security of this protocol, in the sense of Definition 10.1. In Attack Game 10.1, the adversary sees the transcript consisting of the two messages pk and y. If the adversary could compute the secret x from this transcript with some advantage, then this very same adversary could be used directly to break the trapdoor function scheme, as in Attack Game 10.2, with exactly the same advantage.

10.2.2 Mathematical details

We give a more mathematically precise definition of a trapdoor function scheme, using the terminology defined in Section 2.3.

Definition 10.4 (Trapdoor function scheme). *A* ***trapdoor function scheme*** *is a triple of efficient algorithms (G, F, I) along with families of spaces with system parameterization P:*

$$\mathbf{X} = \{\mathcal{X}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \mathbf{Y} = \{\mathcal{Y}_{\lambda,\Lambda}\}_{\lambda,\Lambda}.$$

As usual, $\lambda \in \mathbb{Z}_{\geq 1}$ is a security parameter and $\Lambda \in \text{Supp}(P(\lambda))$ is a domain parameter. We require that

1. *$\mathbf{X}$ is efficiently recognizable and sampleable.*
2. *$\mathbf{Y}$ is efficiently recognizable.*
3. *G is an efficient probabilistic algorithm that on input λ, Λ, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \text{Supp}(P(\lambda))$, outputs a pair (pk, sk), where pk and sk are bit strings whose lengths are always bounded by a polynomial in λ.*
4. *F is an efficient* deterministic *algorithm that on input λ, Λ, pk, x, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \text{Supp}(P(\lambda))$, $(pk, sk) \in \text{Supp}(G(\lambda, \Lambda))$ for some sk, and $x \in \mathcal{X}_{\lambda,\Lambda}$, outputs an element of $\mathcal{Y}_{\lambda,\Lambda}$.*

5. *I is an efficient* deterministic *algorithm that on input λ, Λ, sk, y, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in$* $\text{Supp}(P(\lambda))$*, $(pk, sk) \in$* $\text{Supp}(G(\lambda, \Lambda))$ *for some pk, and $y \in \mathcal{Y}_{\lambda,\Lambda}$, outputs an element of $\mathcal{X}_{\lambda,\Lambda}$.*

6. *For all $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in$* $\text{Supp}(P(\lambda))$*, $(pk, sk) \in$* $\text{Supp}(G(\lambda, \Lambda))$*, and $x \in \mathcal{X}_{\lambda,\Lambda}$, we have $I(\lambda, \Lambda; sk, F(\lambda, \Lambda; pk, x)) = x$.*

As usual, in defining the one-wayness security property, we parameterize Attack Game 10.2 by the security parameter λ, and the advantage $\text{OW}\mathsf{adv}[\mathcal{A}, \mathcal{T}]$ is actually a function of λ. Definition 10.3 should be read as saying that $\text{OW}\mathsf{adv}[\mathcal{A}, \mathcal{T}](\lambda)$ is a negligible function.

10.3 A trapdoor permutation scheme based on RSA

We now describe a trapdoor permutation scheme that is plausibly one-way. It is called RSA after its inventors, Rivest, Shamir, and Adleman. Recall that a trapdoor permutation is a special case of a trapdoor function, where the domain and range are the same set. This means that for every public-key, the function is a permutation of its domain, which is why we call it a trapdoor permutation. Despite many years of study, RSA is essentially the only known reasonable candidate trapdoor permutation scheme (there are a few others, but they are all very closely related to the RSA scheme).

Here is how RSA works. First, we describe a probabilistic algorithm RSAGen that takes as input an integer $\ell > 2$, and an odd integer $e > 2$.

```
RSAGen($\ell, e$) :=
    generate a random $\ell$-bit prime $p$ such that $\gcd(e, p-1) = 1$
    generate a random $\ell$-bit prime $q$ such that $\gcd(e, q-1) = 1$ and $q \neq p$
    $n \leftarrow pq$
    $d \leftarrow e^{-1} \bmod (p-1)(q-1)$
    output $(n, d)$.
```

To efficiently implement the above algorithm, we need an efficient algorithm to generate random ℓ-bit primes. This is discussed in Appendix A. Also, we use the extended Euclidean algorithm (Appendix A) to compute $e^{-1} \bmod (p-1)(q-1)$. Note that since $\gcd(e, p-1) = \gcd(e, q-1) = 1$, it follows that $\gcd(e, (p-1)(q-1)) = 1$, and hence e has a multiplicative inverse modulo $(p-1)(q-1)$.

Now we describe the RSA trapdoor permutation scheme $\mathcal{T}_{\text{RSA}} = (G, F, I)$. It is parameterized by fixed values of ℓ and e.

- Key generation runs as follows:

$$\begin{aligned} G() := \quad & (n, d) \xleftarrow{\text{R}} \text{RSAGen}(\ell, e), \quad pk \leftarrow (n, e), \quad sk \leftarrow (n, d) \\ & \text{output } (pk, sk). \end{aligned}$$

- For a given public key $pk = (n, e)$, and $x \in \mathbb{Z}_n$, we define $F(pk, x) := x^e \in \mathbb{Z}_n$.

- For a given secret key $sk = (n, d)$, and $y \in \mathbb{Z}_n$, we define $I(sk, y) := y^d \in \mathbb{Z}_n$.

Note that although the encryption exponent e is considered to be a fixed system parameter, we also include it as part of the public key pk.

A technicality. For each fixed $pk = (n, e)$, the function $F(pk, \cdot)$ maps $\mathbb{Z}_n$ into $\mathbb{Z}_n$; thus, the domain and range of this function actually vary with pk. However, in our definition of a trapdoor permutation scheme, the domain and range of the function are not allowed to vary with the public key. So in fact, this scheme does not quite satisfy the formal syntactic requirements of a trapdoor permutation scheme. One could easily generalize the definition of a trapdoor permutation scheme, to allow for this. However, we shall not do this; rather, we shall state and analyze various schemes based on a trapdoor permutation scheme as we have defined it, and then show how to instantiate these schemes using RSA. Exercise 10.27 explores an idea that builds a proper trapdoor permutation scheme based on RSA.

Ignoring this technical issue for the moment, let us first verify that $\mathcal{T}_{\text{RSA}}$ satisfies the correctness requirement of a trapdoor permutation scheme. This is implied by the following:

Theorem 10.1. *Let $n = pq$ where p and q are distinct primes. Let e and d be integers such that $ed \equiv 1 \pmod{(p-1)(q-1)}$. Then for all $x \in \mathbb{Z}$, we have $x^{ed} \equiv x \pmod{n}$.*

Proof. The hypothesis that $ed \equiv 1 \pmod{(p-1)(q-1)}$ just means that $ed = 1 + k(p-1)(q-1)$ for some integer k. Certainly, if $x \equiv 0 \pmod{p}$, then $x^{ed} \equiv 0 \equiv x \pmod{p}$; otherwise, if $x \not\equiv 0 \pmod{p}$, then by Fermat's little theorem (Appendix A), we have

$$x^{p-1} \equiv 1 \pmod{p},$$

and so

$$x^{ed} \equiv x^{1+k(p-1)(q-1)} \equiv x \cdot \left(x^{(p-1)}\right)^{k(q-1)} \equiv x \cdot 1^{k(q-1)} \equiv x \pmod{p}.$$

Therefore,

$$x^{ed} \equiv x \pmod{p}.$$

By a symmetric argument, we have

$$x^{ed} \equiv x \pmod{q}.$$

Thus, $x^{ed} - x$ is divisible by the distinct primes p and q, and must therefore be divisible by their product n, which means

$$x^{ed} \equiv x \pmod{n}. \quad \square$$

So now we know that $\mathcal{T}_{\text{RSA}}$ satisfies the correctness property of a trapdoor permutation scheme. However, it is not clear that it is one-way. For $\mathcal{T}_{\text{RSA}}$, one-wayness means that there is no efficient algorithm that given n and x^e, where $x \in \mathbb{Z}_n$ is chosen at random, can effectively compute x. It is clear that if $\mathcal{T}_{\text{RSA}}$ is one-way, then it must be hard to factor n; indeed, if it were easy to factor n, then one could compute d in exactly the same way as is done in algorithm RSAGen, and then use d to compute $x = y^d$.

It is widely believed that factoring n is hard, provided ℓ is sufficiently large — typically, ℓ is chosen to be between 1000 and 1500. Moreover, the only known efficient algorithm to invert $\mathcal{T}_{\text{RSA}}$ is to first factor n and then compute d as above. However, there is no known *proof* that the assumption that factoring n is hard implies that $\mathcal{T}_{\text{RSA}}$ is one-way. Nevertheless, based on current evidence, it seems reasonable to conjecture that $\mathcal{T}_{\text{RSA}}$ is indeed one-way. We state this conjecture now as an explicit assumption. As usual, this is done using an attack game.

Attack Game 10.3 (RSA). For given integers $\ell > 2$ and odd $e > 2$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger and the adversary $\mathcal{A}$ take (ℓ, e) as input.
- The challenger computes

$$(n, d) \overset{\text{R}}{\leftarrow} \text{RSAGen}(\ell, e), \quad x \overset{\text{R}}{\leftarrow} \mathbb{Z}_n, \quad y \leftarrow x^e \in \mathbb{Z}_n$$

  and sends (n, y) to the adversary.
- The adversary outputs $\hat{x} \in \mathbb{Z}_n$.

We define the adversary's advantage in breaking RSA, denoted $\text{RSA}\mathsf{adv}[\mathcal{A}, \ell, e]$, as the probability that $\hat{x} = x$. □

Definition 10.5 (RSA assumption). *We say that the RSA assumption holds for* (ℓ, e) *if for all efficient adversaries* $\mathcal{A}$*, the quantity* $\text{RSA}\mathsf{adv}[\mathcal{A}, \ell, e]$ *is negligible.*

We analyze the RSA assumption and present several known attacks on it later on in Chapter 16.

We next introduce some terminology that will be useful later. Suppose (n, d) is an output of $\text{RSAGen}(\ell, e)$, and suppose that $x \in \mathbb{Z}_n$ and let $y := x^e$. The number n is called an **RSA modulus**, the number e is called an **encryption exponent**, and the number d is called a **decryption exponent**. We call (n, y) an **instance** of the **RSA problem**, and we call x a **solution** to this instance of the RSA problem. The RSA assumption asserts that there is no efficient algorithm that can effectively solve the RSA problem.

10.3.1 Key exchange based on the RSA assumption

Consider now what happens when we instantiate the key exchange protocol in Section 10.2.1 with $\mathcal{T}_{\text{RSA}}$. The protocol runs as follows:

- Alice computes $(n, d) \overset{\text{R}}{\leftarrow} \text{RSAGen}(\ell, e)$, and sends (n, e) to Bob.
- Upon receiving (n, e) from Alice, Bob computes $x \overset{\text{R}}{\leftarrow} \mathbb{Z}_n,\ y \leftarrow x^e$, and sends y to Alice.
- Upon receiving y from Bob, Alice computes $x \leftarrow y^d$.

The secret shared by Alice and Bob is x. The message flow is the same as in Fig. 10.1. Under the RSA assumption, this is a secure anonymous key exchange protocol.

10.3.2 Mathematical details

We give a more mathematically precise definition of the RSA assumption, using the terminology defined in Section 2.3.

In Attack Game 10.3, the parameters ℓ and e are actually poly-bounded and efficiently computable functions of a security parameter λ. Likewise, $\text{RSA}\mathsf{adv}[\mathcal{A}, \ell, e]$ is a function of λ. As usual, Definition 10.5 should be read as saying that $\text{RSA}\mathsf{adv}[\mathcal{A}, \ell, e](\lambda)$ is a negligible function.

There are a couple of further wrinkles we should point out. First, as already mentioned above, the RSA scheme does not quite fit our definition of a trapdoor permutation scheme, as the definition of the latter does not allow the set $\mathcal{X}$ to vary with the public key. It would not be too difficult to modify our definition of a trapdoor permutation scheme to accommodate this generalization. Second, the specification of RSAGen requires that we generate random prime numbers of a given bit length. In theory, it is possible to do this in (expected) polynomial time; however, the most practical algorithms (see Appendix A) may — with negligible probability — output a number that is not a prime. If that should happen, then it may be the case that the basic correctness requirement — namely, that $I(sk, F(pk, x)) = x$ for all pk, sk, x — is no longer satisfied. It would also not be too difficult to modify our definition of a trapdoor permutation scheme to accommodate this type of generalization as well. For example, we could recast this requirement as an attack game (in which any efficient adversary wins with negligible probability): in this game, the challenger generates $(pk, sk) \xleftarrow{R} G()$ and sends (pk, sk) to the adversary; the adversary wins the game if he can output $x \in \mathcal{X}$ such that $I(sk, F(pk, x)) \neq x$. While this would be a perfectly reasonable definition, using it would require us to modify security definitions for higher-level constructs. For example, if we used this relaxed correctness requirement in the context of key exchange, we would have to allow for the possibility that the two parties end up with different keys with some negligible probability.

10.4 Diffie-Hellman key exchange

In this section, we explore another approach to constructing secure key exchange protocols, which was invented by Diffie and Hellman. Just as with the protocol based on RSA, this protocol will require a bit of algebra and number theory. However, before getting in to the details, we provide a bit of motivation and intuition.

Consider the following "generic" key exchange protocol that makes use of two functions E and F. Alice chooses a random secret α, computes $E(\alpha)$, and sends $E(\alpha)$ to Bob over an insecure channel. Likewise, Bob chooses a random secret β, computes $E(\beta)$, and sends $E(\beta)$ to Alice over an insecure channel. Alice and Bob both somehow compute a shared key $F(\alpha, \beta)$. In this high-level description, E and F are some functions that should satisfy the following properties:

1. E should be easy to compute;

2. given α and $E(\beta)$, it should be easy to compute $F(\alpha, \beta)$;

3. given $E(\alpha)$ and β, it should be easy to compute $F(\alpha, \beta)$;

4. given $E(\alpha)$ and $E(\beta)$, it should be hard to compute $F(\alpha, \beta)$.

Properties 1–3 ensure that Alice and Bob can efficiently implement the protocol: Alice computes the shared key $F(\alpha, \beta)$ using the algorithm from Property 2 and her given data α and $E(\beta)$. Bob computes the same key $F(\alpha, \beta)$ using the algorithm from Property 3 and his given data $E(\alpha)$ and β. Property 4 ensures that the protocol is secure: an eavesdropper who sees $E(\alpha)$ and $E(\beta)$ should not be able to compute the shared key $F(\alpha, \beta)$.

Note that properties 1–4 together imply that E is hard to invert; indeed, if we could compute efficiently α from $E(\alpha)$, then by Property 2, we could efficiently compute $F(\alpha, \beta)$ from $E(\alpha), E(\beta)$, which would contradict Property 4.

To make this generic approach work, we have to come up with appropriate functions E and F. To a first approximation, the basic idea is to implement E in terms of exponentiation to some fixed base g, defining $E(\alpha) := g^{\alpha}$ and $F(\alpha, \beta) := g^{\alpha\beta}$. Notice then that

$$E(\alpha)^{\beta} = (g^{\alpha})^{\beta} = F(\alpha, \beta) = (g^{\beta})^{\alpha} = E(\beta)^{\alpha}.$$

Hence, provided exponentiation is efficient, Properties 1–3 are satisfied. Moreover, if Property 4 is to be satisfied, then at the very least, we require that taking logarithms (i.e., inverting E) is hard.

To turn this into a practical and plausibly secure scheme, we cannot simply perform exponentiation on ordinary integers since the numbers would become too large. Instead, we have to work in an appropriate finite algebraic domain, which we introduce next.

10.4.1 The key exchange protocol

Suppose p is a large prime and that q is a large prime dividing $p-1$ (think of p as being very large random prime, say 2048 bits long, and think of q as being about 256 bits long).

We will be doing arithmetic mod p, that is, working in $\mathbb{Z}_p$. Recall that $\mathbb{Z}_p^*$ is the set of nonzero elements of $\mathbb{Z}_p$. An essential fact is that since q divides $p-1$, $\mathbb{Z}_p^*$ has an element g of order q (see Appendix A). This means that $g^q = 1$ and that all of the powers g^a, for $a = 0, \ldots, q-1$, are distinct. Let $\mathbb{G} := \{g^a : a = 0, \ldots, q-1\}$, so that $\mathbb{G}$ is a subset of $\mathbb{Z}_p^*$ of cardinality q. It is not hard to see that $\mathbb{G}$ is closed under multiplication and inversion; that is, for all $u, v \in \mathbb{G}$, we have $uv \in \mathbb{G}$ and $u^{-1} \in \mathbb{G}$. Indeed, $g^a \cdot g^b = g^{a+b} = g^c$ with $c := (a+b) \bmod q$, and $(g^a)^{-1} = g^d$ with $d := (-a) \bmod q$. In the language of algebra, $\mathbb{G}$ is called a subgroup of the group $\mathbb{Z}_p^*$.

For every $u \in \mathbb{G}$ and integers a and b, it is easy to see that $u^a = u^b$ if $a \equiv b \bmod q$. Thus, the value of u^a depends only on the residue class of a modulo q. Therefore, if $\alpha = [a]_q \in \mathbb{Z}_q$ is the residue class of a modulo q, we can define $u^\alpha := u^a$ and this definition is unambiguous. From here on we will frequently use elements of $\mathbb{Z}_q$ as exponents applied to elements of $\mathbb{G}$.

So now we have everything we need to describe the Diffie-Hellman key exchange protocol. We assume that the description of $\mathbb{G}$, including $g \in \mathbb{G}$ and q, is a system parameter that is generated once and for all at system setup time and shared by all parties involved. The protocol runs as follows, as shown in Fig. 10.2:

1. Alice computes $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$, $u \leftarrow g^\alpha$, and sends u to Bob.

2. Bob computes $\beta \xleftarrow{\text{R}} \mathbb{Z}_q, v \leftarrow g^\beta$ and sends v to Alice.

3. Upon receiving v from Bob, Alice computes $w \leftarrow v^\alpha$

4. Upon receiving u from Alice, Bob computes $w \leftarrow u^\beta$

The secret shared by Alice and Bob is

$$w = v^\alpha = g^{\alpha\beta} = u^\beta.$$

10.4.2 Security of Diffie-Hellman key exchange

For a fixed element $g \in \mathbb{G}$, different from 1, the function from $\mathbb{Z}_q$ to $\mathbb{G}$ that sends $\alpha \in \mathbb{Z}_q$ to $g^\alpha \in \mathbb{G}$ is called the **discrete exponentiation function**. This function is one-to-one and onto, and its inverse function is called the **discrete logarithm function**, and is usually denoted Dlog_g; thus, for $u \in \mathbb{G}$, $\mathsf{Dlog}_g(u)$ is the unique $\alpha \in \mathbb{Z}_q$ such that $u = g^\alpha$. The value g is called the **base** of the discrete logarithm.

If the Diffie-Hellman protocol has any hope of being secure, it must be hard to compute α from g^α for a random α; in other words, it must be hard to compute the discrete logarithm function.

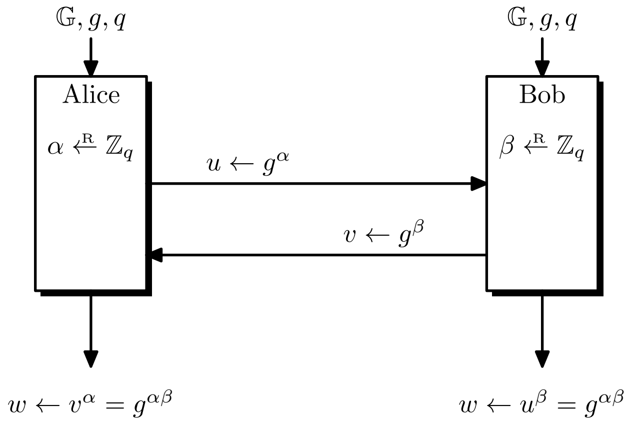

$\mathbb{G}, g, q$
$\mathbb{G}, g, q$
Alice
$\alpha \xleftarrow{R} \mathbb{Z}_q$
Bob
$\beta \xleftarrow{R} \mathbb{Z}_q$
$u \leftarrow g^{\alpha}$
$v \leftarrow g^{\beta}$
$w \leftarrow v^{\alpha} = g^{\alpha\beta}$
$w \leftarrow u^{\beta} = g^{\alpha\beta}$

Figure 10.2: Diffie-Hellman key exchange

There are a number of candidate group families $\mathbb{G}$ where the discrete logarithm function is believed to be hard to compute. For example, when p and q are sufficiently large, suitably chosen primes, the discrete logarithm function in the order q subgroup of $\mathbb{Z}_p^*$ is believed to be hard to compute (p should be at least 2048-bits, and q should be at least 256-bits). This assumption is called the **discrete logarithm assumption** and is defined in the next section.

Unfortunately, the discrete logarithm assumption by itself is not enough to ensure that the Diffie-Hellman protocol is secure. Observe that the protocol is secure if and only if the following holds:

given $g^{\alpha}, g^{\beta} \in \mathbb{G}$, where $\alpha \xleftarrow{R} \mathbb{Z}_q$ and $\beta \xleftarrow{R} \mathbb{Z}_q$, it is hard to compute $g^{\alpha\beta} \in \mathbb{G}$.

This security property is called the **computational Diffie-Hellman assumption**. Although the computational Diffie-Hellman assumption is stronger than the discrete logarithm assumption, all evidence still suggests that this is a reasonable assumption in groups where the discrete logarithm assumption holds.

10.5 Discrete logarithm and related assumptions

In this section, we state the discrete logarithm and related assumptions more precisely and in somewhat more generality, and explore in greater detail relationships among them.

The subset $\mathbb{G}$ of $\mathbb{Z}_p^*$ that we defined above in Section 10.4 is a specific instance of a general type of mathematical object known as a *cyclic group*. There are in fact other cyclic groups that are very useful in cryptography, most notably, groups based on *elliptic curves* — we shall study elliptic curve cryptography in Chapter 15. From now on, we shall state assumptions and algorithms in terms of an abstract cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. In general, such groups may be selected by a randomized process, and again, the description of $\mathbb{G}$, including $g \in \mathbb{G}$ and q, is a system parameter that is generated once and for all at system setup time and shared by all parties involved.

We shall use just a bit of terminology from group theory. The reader who is unfamiliar with the concept of a group may wish to refer to Appendix A; alternatively, for the time being, the reader may simply ignore this abstraction entirely:

- Whenever we refer to a "cyclic group," the reader may safely assume that this means the specific set $\mathbb{G}$ defined above as a subgroup of $\mathbb{Z}_p^*$.
- The "order of $\mathbb{G}$" is just a fancy name for the size of the set $\mathbb{G}$, which is q.
- A "generator of $\mathbb{G}$" is an element $g \in \mathbb{G}$ with the property that every element of $\mathbb{G}$ can be expressed as a power of g.
- By convention, we assume that the description of $\mathbb{G}$ includes its order q and some generator $g \in \mathbb{G}$.

We begin with a formal statement of the discrete logarithm assumption, stated in our more general language. As usual, we need an attack game.

Attack Game 10.4 (Discrete logarithm). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$, define the following attack game:

- The challenger and the adversary $\mathcal{A}$ take a description of $\mathbb{G}$ as input. The description includes the order q and a generator $g \in \mathbb{G}$.
- The challenger computes
  $$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g^\alpha,$$
  and sends $u \in \mathbb{G}$ to the adversary.
- The adversary outputs some $\hat{\alpha} \in \mathbb{Z}_q$.

We define $\mathcal{A}$'s **advantage in solving the discrete logarithm problem for** $\mathbb{G}$, denoted $\mathsf{DLadv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{\alpha} = \alpha$. □

Definition 10.6 (Discrete logarithm assumption). *We say that the* ***discrete logarithm (DL) assumption*** *holds for* $\mathbb{G}$ *if for all efficient adversaries* $\mathcal{A}$ *the quantity* $\mathsf{DLadv}[\mathcal{A}, \mathbb{G}]$ *is negligible.*

We say that g^α is an **instance** of the **discrete logarithm (DL) problem (for** $\mathbb{G}$**)**, and that α is a solution to this problem instance. The DL assumption asserts that there is no efficient algorithm that can effectively solve the DL problem for $\mathbb{G}$.

Note that the DL assumption is defined in terms of a group $\mathbb{G}$ and generator $g \in \mathbb{G}$. As already mentioned, the group $\mathbb{G}$ and generator g are chosen and fixed at system setup time via a process that may be randomized. Also note that all elements of $\mathbb{G} \setminus \{1\}$ are in fact generators for $\mathbb{G}$, but we do not insist that g is chosen uniformly among these (but see Exercise 10.19). Different methods for selecting groups and generators give rise to different DL assumptions (and the same applies to the CDH and DDH assumptions, defined below).

Now we state the computational Diffie-Hellman assumption.

Attack Game 10.5 (Computational Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger and the adversary $\mathcal{A}$ take a description of $\mathbb{G}$ as input. The description includes the order q and a generator $g \in \mathbb{G}$.

- The challenger computes

$$\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha}, \quad v \leftarrow g^{\beta}, \quad w \leftarrow g^{\alpha\beta}$$

  and sends the pair (u, v) to the adversary.

- The adversary outputs some $\hat{w} \in \mathbb{G}$.

We define $\mathcal{A}$'s **advantage in solving the computational Diffie-Hellman problem for** $\mathbb{G}$, denoted $\text{CDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{w} = w$. □

Definition 10.7 (Computational Diffie-Hellman assumption). *We say that the* ***computational Diffie-Hellman (CDH)*** *assumption holds for* $\mathbb{G}$ *if for all efficient adversaries* $\mathcal{A}$ *the quantity* $\text{CDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ *is negligible.*

We say that (g^{α}, g^{β}) is an **instance** of the **computational Diffie-Hellman (CDH) problem**, and that $g^{\alpha\beta}$ is a solution to this problem instance. The CDH assumption asserts that there is no efficient algorithm that can effectively solve the CDH problem for $\mathbb{G}$.

An interesting property of the CDH problem is that there is no general and efficient algorithm to even *recognize* correct solutions to the CDH problem, that is, given an instance (u, v) of the CDH problem, and a group element $\hat{w}$, to determine if $\hat{w}$ is a solution to the given problem instance. This is in contrast to the RSA problem: given an instance (n, e, y) of the RSA problem, and an element $\hat{x}$ of $\mathbb{Z}_n^*$, we can efficiently test if $\hat{x}$ is a solution to the given problem instance simply by testing if $\hat{x}^e = y$. In certain cryptographic applications, this lack of an efficient algorithm to recognize solutions to the CDH problem can lead to technical difficulties. However, this apparent limitation is also an *opportunity*: if we assume not only that solving the CDH problem is hard, but also that recognizing solutions to the CDH problem is hard, then we can sometimes prove stronger security properties for certain cryptographic schemes.

We shall now formalize the assumption that recognizing solutions to the CDH problem is hard. In fact, we shall state a stronger assumption, namely, that even distinguishing solutions from random group elements is hard. It turns out that this stronger assumption is equivalent to the weaker one (see Exercise 10.10).

Attack Game 10.6 (Decisional Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$, we define two experiments.

Experiment b **(**$b = 0, 1$**):**

- The challenger and the adversary $\mathcal{A}$ take a description of $\mathbb{G}$ as input.

- The challenger computes

$$\alpha, \beta, \gamma \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha}, \quad v \leftarrow g^{\beta}, \quad w_0 \leftarrow g^{\alpha\beta}, \quad w_1 \leftarrow g^{\gamma},$$

  and sends the triple (u, v, w_b) to the adversary.

- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage in solving the decisional Diffie-Hellman problem for** $\mathbb{G}$ as

$$\text{DDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Definition 10.8 (Decisional Diffie-Hellman assumption). *We say that the* ***decisional Diffie-Hellman (DDH)*** *assumption holds for* $\mathbb{G}$ *if for all efficient adversaries* $\mathcal{A}$ *the quantity* $\text{DDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ *is negligible.*

For $\alpha, \beta, \gamma \in \mathbb{Z}_q$, we call $(g^\alpha, g^\beta, g^\gamma)$ a **DH-triple** if $\gamma = \alpha\beta$; otherwise, we call it a **non-DH-triple**. The DDH assumption says that there is no efficient algorithm that can effectively distinguish between random DH-triples and random triples. More precisely, in the language of Section 3.11, the DDH assumptions says that the uniform distribution over DH-triples and the uniform distribution over $\mathbb{G}^3$ are computationally indistinguishable. It is not hard to show that the DDH assumption implies that it is hard to distinguish between random DH-triples and random non-DH-triples (see Exercise 10.7).

Clearly, the DDH assumption implies the CDH assumption: if we could effectively solve the CDH problem, then we could easily determine if a given triple $(u, v, \hat{w})$ is a DH-triple by first computing a correct solution w to the instance (u, v) of the CDH problem, and then testing if $w = \hat{w}$.

In defining the DL, CDH, and DDH assumptions, we have restricted our attention to prime order groups. This is convenient for a number of technical reasons. See, for example Section 16.1.3 where we show that the DDH assumption for groups of even order is simply false.

10.5.1 Random self-reducibility

An important property of the discrete-log function in a group $\mathbb{G}$ is that it is either hard almost everywhere in $\mathbb{G}$ or easy everywhere in $\mathbb{G}$. A middle ground where discrete-log is easy for some inputs and hard for others is not possible. We prove this by showing that the discrete-log function has a *random self reduction*.

Consider a specific cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. Suppose $\mathcal{A}$ is an efficient algorithm with the following property: if $u \in \mathbb{G}$ is chosen at random, then $\Pr[\mathcal{A}(u) = \mathsf{Dlog}_g(u)] = \epsilon$. That is, on a random input u, algorithm $\mathcal{A}$ computes the discrete logarithm of u with probability ϵ. Here, the probability is over the random choice of u, as well as any random choices made by $\mathcal{A}$ itself.[1] Suppose $\epsilon = 0.1$. Then the group $\mathbb{G}$ is of little use in cryptography since an eavesdropper can use $\mathcal{A}$ to break 10% of all Diffie-Hellman key exchanges. However, this does not mean that $\mathcal{A}$ is able to compute $\mathsf{Dlog}_g(u)$ with non-zero probability for all $u \in \mathbb{G}$. It could be the case that for 10% of the inputs $u \in \mathbb{G}$, algorithm $\mathcal{A}$ always computes $\mathsf{Dlog}_g(u)$, while for the remaining 90%, it never computes $\mathsf{Dlog}_g(u)$.

We show how to convert $\mathcal{A}$ into an efficient algorithm $\mathcal{B}$ with the following property: for all $u \in \mathbb{G}$, algorithm $\mathcal{B}$ on input u successfully computes $\mathsf{Dlog}_g(u)$ with probability ϵ. Here, the probability is only over the random choices made by $\mathcal{B}$. We do so using a reduction that maps a given discrete-log instance to a random discrete-log instance. Such a reduction is called a **random self reduction**.

Theorem 10.2. *Consider a specific cyclic group* $\mathbb{G}$ *of prime order* q *generated by* $g \in \mathbb{G}$*. Suppose* $\mathcal{A}$ *is an efficient algorithm with the following property: if* $u \in \mathbb{G}$ *is chosen at random, then* $\Pr[\mathcal{A}(u) = \mathsf{Dlog}_g(u)] = \epsilon$*, where the probability is over the random choice of* u *and the random choices made by* $\mathcal{A}$*. Then there is an efficient algorithm* $\mathcal{B}$ *with the following property: for all* $u \in \mathbb{G}$*, algorithm* $\mathcal{B}$

[1] Technical note: the probability ϵ is not quite the same as $\text{DL}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$, as the latter is also with respect to the random choice of group/generator made at system setup time; here, we are viewing these as truly fixed.

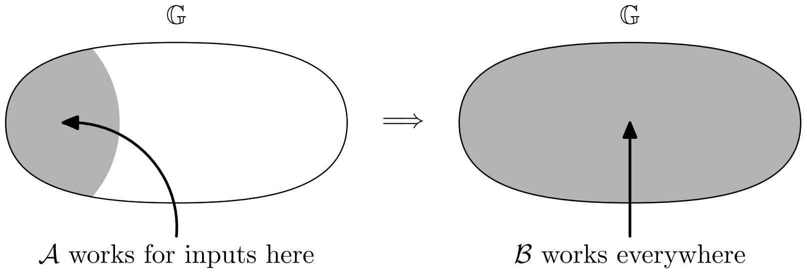

$\mathbb{G}$
$\mathbb{G}$
$\Longrightarrow$
$\mathcal{A}$ works for inputs here
$\mathcal{B}$ works everywhere

Figure 10.3: The effect of a random self reduction

either outputs fail *or* $\mathsf{Dlog}_g(u)$*, and it outputs the latter with probability* ϵ*, where now the probability is only over the random choices made by* $\mathcal{B}$*.*

Theorem 10.2 implements the transformation shown in Fig. 10.3. The point is that, unlike $\mathcal{A}$, algorithm $\mathcal{B}$ works for *all* inputs. To compute discrete-log of a particular $u \in \mathbb{G}$ one can iterate $\mathcal{B}$ on the same input u several times, say $n\lceil 1/\epsilon \rceil$ times for some n. Using the handy inequality $1 + x \leq \exp(x)$ (which holds for all x), this iteration will produce the discrete-log with probability $1-(1-\epsilon)^{n\lceil 1/\epsilon \rceil} \geq 1-\exp(-n)$. In particular, if $1/\epsilon$ is poly-bounded, we can efficiently compute the discrete logarithm of any group element with negligible failure probability. In contrast, iterating $\mathcal{A}$ on the same input u many times may never produce a correct answer. Consequently, if discrete-log is easy for a non-negligible fraction of instances, then it will be easy for *all* instances.

Proof of Theorem 10.2. Algorithm $\mathcal{B}$ works as follows:

```
Input: $u \in \mathbb{G}$
Output: $\mathsf{Dlog}_g(u)$ or fail
$\sigma \xleftarrow{R} \mathbb{Z}_q$
$u_1 \leftarrow u \cdot g^{\sigma} \in \mathbb{G}$
$\alpha_1 \leftarrow \mathcal{A}(u_1)$
if $g^{\alpha_1} \neq u_1$
    then output fail
    else  output $\alpha \leftarrow \alpha_1 - \sigma$
```

Suppose that $u = g^{\alpha}$. Observe that $u_1 = g^{\alpha+\sigma}$. Since σ is uniformly distributed over $\mathbb{Z}_q$, the group element u_1 is uniformly distributed over $\mathbb{G}$. Therefore, on input u_1, adversary $\mathcal{A}$ will output $\alpha_1 = \alpha + \sigma$ with probability ϵ. When this happens, $\mathcal{B}$ will output $\alpha_1 - \sigma = \alpha$, and otherwise, $\mathcal{B}$ will output fail. □

Why random self reducibility is important. Any hard problem can potentially form the basis of a cryptosystem. For example, an NP-hard problem known as subset sum has attracted attention for many years. Unfortunately, many hard problems, including subset sum, are only hard in the worst case. Generally speaking, such problems are of little use in cryptography, where we need problems that are not just hard in the worst case, but hard *on average* (i.e., for randomly chosen inputs). For a problem with a random self-reduction, if it is hard in the worst case, then it must be hard on average. This implication makes such problems attractive for cryptography.

One can also give random self reductions for both the CDH and DDH problems, as well as for the RSA problem (in a more limited sense). These ideas are developed in the chapter exercises.

10.5.2 Mathematical details

As in previous sections, we give the mathematical details pertaining to the DL, CDH, and DDH assumptions. We use the terminology introduced in Section 2.3. This section may be safely skipped on first reading with very little loss in understanding.

To state the assumptions asymptotically we introduce a security parameter λ that identifies the group in which the DL, CDH, and DDH games are played. We will require that the adversary's advantage in breaking the assumption is a negligible function of λ. As lambda increases the adversary's advantage in breaking discrete-log in the group defined by λ should quickly go to zero.

To make sense of the security parameter λ we need a family of groups that increase in size as λ increases. As in Section 2.3, this family of groups is parameterized by both λ and an additional system parameter Λ. The idea is that once λ is chosen, a system parameter Λ is generated by a **system parameterization algorithm** P. The pair (λ, Λ) then fully identifies the group $\mathbb{G}_{\lambda,\Lambda}$ where the DL, CDH, and DDH games are played. Occasionally we will refer to Λ as a **group description**. This Λ is a triple

$$\Lambda := (\ \Lambda_1,\ q,\ g\ )$$

where Λ_1 is an arbitrary string, q is a prime number that represents the order of the group $\mathbb{G}_{\lambda,\Lambda}$, and g is a generator of $\mathbb{G}_{\lambda,\Lambda}$.

Definition 10.9 (group family). *A **group family** $\mathbb{G}$ consists of an algorithm* Mul *along with a family of spaces:*

$$\mathbf{G} = \{\mathbb{G}_{\lambda,\Lambda}\}_{\lambda,\Lambda}$$

with system parameterization algorithm P, such that

1. $\mathbf{G}$ *is efficiently recognizable.*

2. *Algorithm* Mul *is an efficient deterministic algorithm that on input $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $u, v \in \mathbb{G}_{\lambda,\Lambda}$, outputs $w \in \mathbb{G}_{\lambda,\Lambda}$.*

3. *For all $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda = (\Lambda_1, q, g) \in \mathrm{Supp}(P(\lambda))$, algorithm* Mul *is a multiplication operation on $\mathbb{G}_{\lambda,\Lambda}$ that defines a cyclic group of prime order q generated by g.*

The definition implies that all the spaces $\mathbb{G}_{\lambda,\Lambda}$ are efficiently sampleable. Since $\Lambda = (\Lambda_1, q, g)$ we can randomly sample a random element u of $\mathbb{G}_{\lambda,\Lambda}$ by picking a random $\alpha \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$ and setting $u \leftarrow g^{\alpha}$. Specific group families may allow for a more efficient method that generates a random group element. The group identity element may always be obtained by raising g to the power q, although for specific group families, there are most likely simpler and faster ways to do this.

An example. We define the asymptotic version of a subgroup of prime order q within $\mathbb{Z}_p^*$, where q is a prime dividing $p-1$, and p itself is prime. Here the system parameterization algorithm P takes λ as input and outputs a group description $\Lambda := (p, q, g)$ where p is a random $\ell(\lambda)$-bit prime (for some poly-bounded length function ℓ) and g is an element of $\mathbb{Z}_p^*$ of order q. The group $\mathbb{G}_{\lambda,\Lambda}$ is the subgroup of $\mathbb{Z}_p^*$ generated by g. Elements of $\mathbb{G}_{\lambda,\Lambda}$ may be efficiently recognized as follows: first, one can check that a given bit string properly encodes an element u of $\mathbb{Z}_p^*$; second, one can check that $u^q = 1$.

Armed with the concept of a group family, we now parameterize the DL Attack Game 10.4 by the security parameter λ. In that game, the adversary is given the security parameter λ and a group description $\Lambda = (\Lambda_1, q, g)$, where g is a generator for the group $\mathbb{G}_{\lambda,\Lambda}$. It is also given a random $u \in \mathbb{G}_{\lambda,\Lambda}$, and it wins the game if it computes $\mathsf{Dlog}_g(u)$. Its advantage $\mathrm{DL}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ is now a function of λ, and for each λ, this advantage is a probability that depends on the random choice of group and generator, as well as the random choices made by the the challenger and the adversary. Definition 10.6 should be read as saying that $\mathrm{DL}\mathsf{adv}[\mathcal{A}, \mathbb{G}](\lambda)$ is a negligible function.

We use the same approach to define the asymptotic CDH and DDH assumptions.

10.6 Collision resistant hash functions from number-theoretic primitives

It turns out that the RSA and DL assumptions are extremely versatile, and can be used in many cryptographic applications. As an example, in this section, we show how to build collision-resistant hash functions based on the RSA and DL assumptions.

Recall from Section 8.1 that a hash function H defined over $(\mathcal{M}, \mathcal{T})$ is an efficiently computable function from $\mathcal{M}$ to $\mathcal{T}$. In most applications, we want the message space $\mathcal{M}$ to be much larger than the digest space $\mathcal{T}$. We also defined a notion of collision resistance, which says that for every efficient adversary $\mathcal{A}$, its collision-finding advantage $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H]$ is negligible. Here, $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H]$ is defined to be the probability that $\mathcal{A}$ can produce a collision, i.e., a pair $m_0, m_1 \in \mathcal{M}$ such that $m_0 \neq m_1$ but $H(m_0) = H(m_1)$.

10.6.1 Collision resistance based on DL

Before presenting our DL-based hash function, we introduce a simple but surprisingly useful concept. Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Suppose $h \in \mathbb{G}$ is an arbitrary group element.

For $u \in \mathbb{G}$, a **representation (relative to g and h) of u** is a pair $(\alpha, \beta) \in \mathbb{Z}_q^2$ such that $g^\alpha h^\beta = u$. For a given $u \in \mathbb{G}$, there are many representations. In fact, there are precisely q of them: for every $\beta \in \mathbb{Z}_q$, there exists a unique $\alpha \in \mathbb{Z}_q$ such that $g^\alpha = uh^{-\beta}$.

The key to our hash function design is the following fact: *given two different representations of the same group element, we can efficiently compute* $\mathsf{Dlog}_g h$. Indeed, suppose (α, β) and (α', β') are two different representations of the same group element. This means

$$g^\alpha h^\beta = g^{\alpha'} h^{\beta'} \quad \text{and} \quad (\alpha, \beta) \neq (\alpha', \beta').$$

This implies

$$g^{\alpha - \alpha'} = h^{\beta' - \beta}. \tag{10.1}$$

Moreover, we must have $\beta' - \beta \neq 0$, as otherwise, (10.1) (and the fact that g is a generator) would imply $\alpha - \alpha' = 0$, contradicting the assumption that $(\alpha, \beta) \neq (\alpha', \beta')$. It follows that $\beta - \beta'$ has a multiplicative inverse in $\mathbb{Z}_q$, which we can in fact efficiently compute (see Appendix A). Raising both sides of (10.1) to the power $1/(\beta' - \beta)$, we obtain

$$g^{(\alpha - \alpha')/(\beta' - \beta)} = h.$$

In other words, $\mathsf{Dlog}_g h = (\alpha - \alpha')/(\beta' - \beta)$.

To summarize:

Fact 10.3 (Computing DL from two representations). *Suppose we are given (α, β) and (α', β'), which are two different representations (relative to g and h) of the same group element. Then we can efficiently compute $\mathsf{Dlog}_g h$ as follows:*

$$\mathsf{Dlog}_g h = (\alpha - \alpha')/(\beta' - \beta).$$

This fact suggests the following hash function H_{dl}, which is defined over $(\mathbb{Z}_q \times \mathbb{Z}_q, \mathbb{G})$. This hash function is parameterized by the group $\mathbb{G}$ and the generator g, along with a randomly chosen $h \in \mathbb{G}$. Thus, the group $\mathbb{G}$, along with the group elements g and h, are chosen once and for all; together, these system parameters define the hash function H_{dl}. For $\alpha, \beta \in \mathbb{Z}_q$, we define

$$H_{\text{dl}}(\alpha, \beta) := g^\alpha h^\beta.$$

The essential observation is that a collision on H_{dl} is a pair of distinct representations of the same group element, and so from any collision, we can use Fact 10.3 to compute $\mathsf{Dlog}_g h$.

Theorem 10.4. *The hash function H_{dl} is collision resistant under the DL assumption.*

In particular, for every collision-finding adversary $\mathcal{A}$, there exists a DL adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\text{CR}\mathsf{adv}[\mathcal{A}, H_{\text{dl}}] = \text{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}]. \tag{10.2}$$

Proof. We use the given collision-finding adversary $\mathcal{A}$ to build a DL adversary $\mathcal{B}$ as follows. When $\mathcal{B}$ receives its challenge $h \in \mathbb{G}$ from its DL-challenger, $\mathcal{B}$ runs $\mathcal{A}$ using H_{dl}, which is defined using $\mathbb{G}$, g, and the given h. Suppose $\mathcal{A}$ finds a collision. This is a pair of distinct inputs $(\alpha, \beta) \neq (\alpha', \beta')$ such that

$$g^\alpha h^\beta = g^{\alpha'} h^{\beta'}.$$

In other words, (α, β) and (α', β') are distinct representations (relative to g and h) of the same group element. From these, $\mathcal{B}$ can compute $\mathsf{Dlog}_g h$ as in Fact 10.3. □

The function $H_{\text{dl}} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ maps from a message space of size q^2 to a digest space of size q. The good news is that the message space is larger than the digest space, and so the hash function actually compresses. The bad news is that the set of *encodings* of $\mathbb{G}$ may be much larger than the set $\mathbb{G}$ itself. Indeed, if $\mathbb{G}$ is constructed as recommended in Section 10.4 as a subset of $\mathbb{Z}_p^*$, then elements of $\mathbb{G}$ are encoded as 2048-bit strings, even though the group $\mathbb{G}$ itself has order $\approx 2^{256}$. So if we replace the set $\mathbb{G}$ by the set of encodings, the hash function H_{dl} is not compressing at all. This problem can be avoided by using other types of groups with more compact encodings, such as elliptic curve groups (see Chapter 15). See also Exercise 10.21 and Exercise 10.22.

10.6.2 Collision resistance based on RSA

We shall work with an RSA encryption exponent e that is a prime. For this application, the bigger e is, the more compression we get. Let $I_e := \{0, \dots, e-1\}$. Let n be an RSA modulus, generated as in Section 10.3 using an appropriate length parameter ℓ. We also choose a random $y \in \mathbb{Z}_n^*$. The values e, n, and y are chosen once and for all, and together they determine a hash function H_{rsa} defined over $(\mathbb{Z}_n^* \times I_e,\ \mathbb{Z}_n^*)$ as follows: for $a \in \mathbb{Z}_n^*$ and $b \in I_e$, we define

$$H_{\text{rsa}}(a, b) := a^e y^b.$$

We will show that H_{rsa} is collision resistant under the RSA assumption. Note that H_{rsa} can be used directly as a compression function in the Merkle-Damgård paradigm (see Section 8.4) to build a collision-resistant hash function for arbitrarily large message spaces. In applying Theorem 8.3, we would take $\mathcal{X} = \mathbb{Z}_n^*$ and $\mathcal{Y} = \{0,1\}^{\lfloor \log_2 e \rfloor}$.

To analyze H_{rsa}, we will need a couple of technical results. The first result simply says that in the RSA attack game, it is no easier to compute an eth root of a random element of $\mathbb{Z}_n^*$ than it is to compute an eth root of a random element of $\mathbb{Z}_n$. To make this precise, suppose that we modify Attack Game 10.3 so that the challenger chooses $x \xleftarrow{\text{R}} \mathbb{Z}_n^*$, and keep everything else the same. Note that since x is uniformly distributed over $\mathbb{Z}_n^*$, the value $y := x^e$ is also uniformly distributed over $\mathbb{Z}_n^*$. Denote by uRSA$\mathsf{adv}[\mathcal{A}, \ell, e]$ the adversary $\mathcal{A}$'s advantage in this modified attack game.

Theorem 10.5. *Let $\ell > 2$ and odd $e > 2$ be integers. For every adversary $\mathcal{A}$, there exists an adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that* uRSA$\mathsf{adv}[\mathcal{A}, \ell, e] \le$ RSA$\mathsf{adv}[\mathcal{B}, \ell, e]$.

Proof. Let $\mathcal{A}$ be a given adversary. Here is how $\mathcal{B}$ works. Adversary $\mathcal{B}$ receives a random element $y \in \mathbb{Z}_n$. If $y \in \mathbb{Z}_n^*$, then $\mathcal{B}$ gives y to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Otherwise, $\mathcal{B}$ computes an eth root x of y as follows. If $y = 0$, $\mathcal{B}$ sets $x := 0$; otherwise, by computing the GCD of y and n, $\mathcal{B}$ can factor n, compute the RSA decryption exponent d, and then compute $x := y^d$.

Let W be the event that $\mathcal{B}$ succeeds. We have

$$\Pr[W] = \Pr[W \mid y \in \mathbb{Z}_n^*] \Pr[y \in \mathbb{Z}_n^*] + \Pr[W \mid y \notin \mathbb{Z}_n^*] \Pr[y \notin \mathbb{Z}_n^*].$$

The result follows from the observations that

$$\Pr[W \mid y \in \mathbb{Z}_n^*] = \text{uRSA}\mathsf{adv}[\mathcal{A}, \ell, e]$$

and

$$\Pr[W \mid y \notin \mathbb{Z}_n^*] = 1 \ge \text{uRSA}\mathsf{adv}[\mathcal{A}, \ell, e]. \quad \Box$$

The above theorem shows that the standard RSA assumption implies a variant RSA assumption, where the preimage is chosen at random from $\mathbb{Z}_n^*$, rather than $\mathbb{Z}_n$. In Exercise 10.26, you are to show the converse, that is, that this variant RSA assumption implies the standard RSA assumption.

We also need the following technical result, which says that given $y \in \mathbb{Z}_n^*$, along with an integer f that is relatively prime to e, and an eth root of y^f, we can easily compute an eth root of y itself.

Just to get a feeling for the result, suppose $e = 3$ and $f = 2$. We have $w \in \mathbb{Z}_n^*$ such that $w^3 = y^2$. We want to compute $x \in \mathbb{Z}_n^*$ such that $x^3 = y$. If we set $x := (y/w)$, then we have

$$x^3 = y^3/w^3 = y^3/y^2 = y.$$

Theorem 10.6 (Shamir's trick). *There is an efficient algorithm that takes as input n, e, f, w, y, where n is a positive integer, e and f are relatively prime integers, and w and y are elements of $\mathbb{Z}_n^*$ that satisfy $w^e = y^f$, and outputs $x \in \mathbb{Z}_n^*$ such that $x^e = y$.*

Proof. Using the extended Euclidean algorithm (Appendix A), we compute integers s and t such that $es + ft = \gcd(e, f)$, and output $x := y^s w^t$. If $\gcd(e, f) = 1$ and $w^e = y^f$, then

$$x^e = (y^s w^t)^e = y^{es} w^{et} = y^{es} y^{ft} = y^{es+ft} = y^1 = y. \quad \Box$$

Theorem 10.7. *The hash function H_{rsa} is collision resistant under the RSA assumption.*

In particular, for every collision-finding adversary $\mathcal{A}$, there exists an RSA adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{CR}\mathsf{adv}[\mathcal{A}, H_{\mathrm{rsa}}] \leq \mathrm{RSA}\mathsf{adv}[\mathcal{B}, \ell, e]. \tag{10.3}$$

Proof. We construct an adversary $\mathcal{B}'$ that plays the alternative RSA attack game considered in Theorem 10.5. We will show that $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H_{\mathrm{rsa}}] = \mathrm{uRSA}\mathsf{adv}[\mathcal{B}', \ell, e]$, and the theorem will the follow from Theorem 10.5.

Our RSA adversary $\mathcal{B}'$ runs as follows. It receives (n, y) from its challenger, where n is an RSA modulus and y is a random element of $\mathbb{Z}_n^*$. The values e, n, y define the hash function H_{rsa}, and adversary $\mathcal{B}'$ runs adversary $\mathcal{A}$ with this hash function. Suppose that $\mathcal{A}$ finds a collision. This is a pair of inputs $(a, b) \neq (a', b')$ such that

$$a^e y^b = (a')^e y^{b'},$$

which we may rewrite as

$$(a/a')^e = y^{b'-b}.$$

Using this collision, $\mathcal{B}'$ will compute an eth root of y.

Observe that $b' - b \neq 0$, since otherwise we would have $(a/a') = 1$ and hence $a = a'$. Also observe that since $|b - b'| < e$ and e is prime, we must have $\gcd(e, b - b') = 1$. So now we simply apply Theorem 10.6 with n, e, and y as given, and $w := a/a'$ and $f := b' - b$. □

10.7 Attacks on the anonymous Diffie-Hellman protocol

The Diffie-Hellman key exchange is secure against a passive eavesdropper. Usually, however, an attacker capable of eavesdropping on traffic is also able to inject its own messages. The protocol completely falls apart in the presence of an active adversary who controls the network. The main reason is the lack of authentication. Alice sets up a shared secret, but she has no idea with whom the secret is shared. The same holds for Bob. An active attacker can abuse this to expose all traffic between Alice and Bob. The attack, called a **man in the middle attack**, works against any key exchange protocol that does not include authentication. It works as follows (see Fig. 10.4):

- Alice sends (g, g^{α}) to Bob. The attacker blocks this message from reaching Bob. He picks a random $\alpha' \xleftarrow{\mathrm{R}} \mathbb{Z}_n$ and sends $(g, g^{\alpha'})$ to Bob.

- Bob responds with g^{β}. The attacker blocks this message from reaching Alice. He picks a random $\beta' \xleftarrow{\mathrm{R}} \mathbb{Z}_n$ and sends $g^{\beta'}$ to Alice.

- Now Alice computes the key $k_{\mathrm{A}} := g^{\alpha\beta'}$ and Bob computes $k_{\mathrm{B}} := g^{\alpha'\beta}$. The attacker knows both k_{A} and k_{B}.

At this point Alice thinks k_{A} is a secret key shared with Bob and will use k_{A} to encrypt messages to him. Similarly for Bob with his key k_{B}. The attacker can act as a proxy between the two. He intercepts each message $c_i := E(k_{\mathrm{A}}, m_i)$ from Alice, re-encrypts it as $c_i' \leftarrow E(k_{\mathrm{B}}, m_i)$ and forwards c_i' to Bob. He also re-encrypts messages from Bob to Alice. The communication channel works properly for both parties and they have no idea that this proxying is taking place. The attacker, however, sees all plaintexts in the clear.

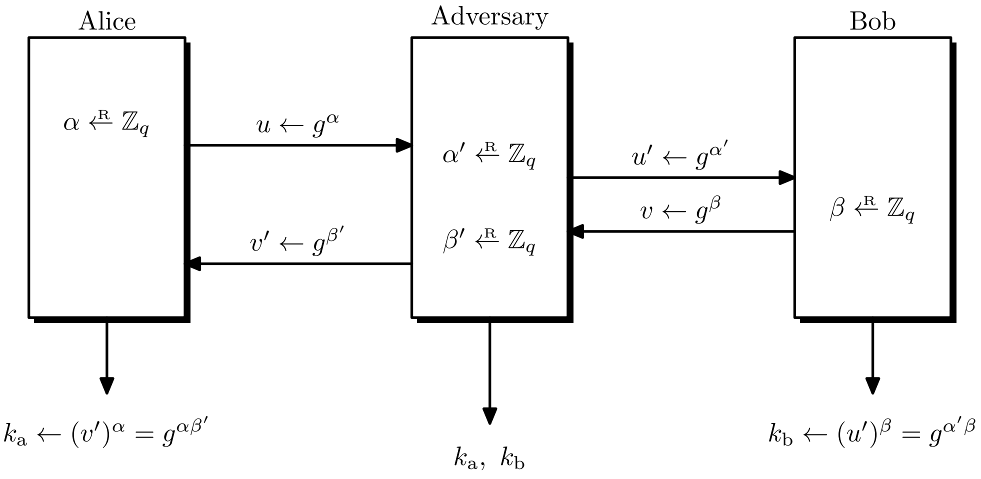

Alice
Adversary
Bob
$\alpha \xleftarrow{R} \mathbb{Z}_q$
$u \leftarrow g^{\alpha}$
$\alpha' \xleftarrow{R} \mathbb{Z}_q$
$u' \leftarrow g^{\alpha'}$
$v \leftarrow g^{\beta}$
$\beta \xleftarrow{R} \mathbb{Z}_q$
$v' \leftarrow g^{\beta'}$
$\beta' \xleftarrow{R} \mathbb{Z}_q$
$k_{\mathrm{a}} \leftarrow (v')^{\alpha} = g^{\alpha\beta'}$
$k_{\mathrm{a}},\ k_{\mathrm{b}}$
$k_{\mathrm{b}} \leftarrow (u')^{\beta} = g^{\alpha'\beta}$

Figure 10.4: Man in the middle attack

This generic attack explains why we view key exchange secure against eavesdropping as a toy problem. Protocols secure in this model can completely fall apart once the adversary can tamper with traffic. We will come back to this problem in Chapter 21, where we design protocols secure against active attackers.

10.8 Merkle puzzles: a partial solution to key exchange using block ciphers

Can we build a secure key exchange protocol using symmetric-key primitives? The answer is yes, but the resulting protocol is very inefficient. We show how to do key exchange using a block cipher $\mathcal{E} = (E, D)$ defined over $(\mathcal{K}, \mathcal{M})$. Alice and Bob want to generate a random $s \in \mathcal{M}$ that is unknown to the adversary. They use a protocol called **Merkle puzzles** (due to the same Merkle from the Merkle-Damgård hashing paradigm). The protocol, shown in Fig. 10.5, works as follows:

Protocol 10.1 (Merkle puzzles).

1. Alice chooses random pairs $(k_i, s_i) \xleftarrow{R} \mathcal{K} \times \mathcal{M}$ for $i = 1, \ldots, L$. We will determine the optimal value for L later. She constructs L puzzles where puzzle P'_i is defined as a triple:

$$P'_i := \big(\ E(k_i, s_i),\ E(k_i, i),\ E(k_i, 0)\ \big).$$

   Next, she sends the L puzzles in a random order to Bob. That is, she picks a random permutation $\pi \xleftarrow{R} \text{Perms}[\{1, \ldots, L\}]$ and sends $(P_1, \ldots, P_L) := (P'_{\pi(1)}, \ldots, P'_{\pi(L)})$ to Bob.

2. Bob picks a random puzzle $P_j = (c_1, c_2, c_3)$ where $j \xleftarrow{R} \{1, \ldots, L\}$. He solves the puzzle by brute force, by trying all keys $k \in \mathcal{K}$ until he finds one such that

$$D(k, c_3) = 0. \tag{10.4}$$

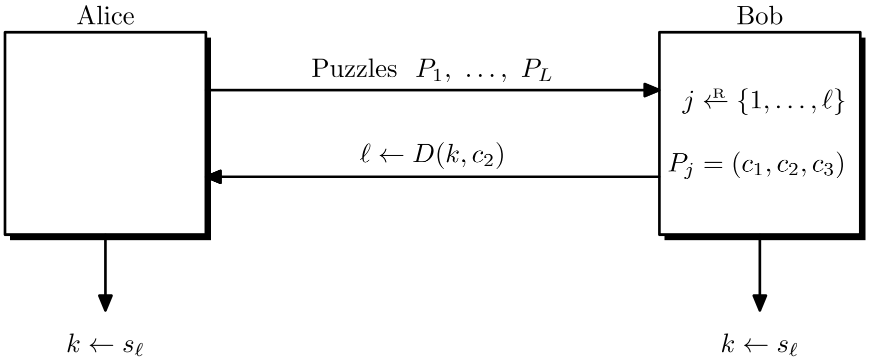

Alice
Bob
Puzzles $P_1, \ldots, P_L$
$j \xleftarrow{R} \{1, \ldots, \ell\}$
$\ell \leftarrow D(k, c_2)$
$P_j = (c_1, c_2, c_3)$
$k \leftarrow s_\ell$
$k \leftarrow s_\ell$

Figure 10.5: Merkle puzzles protocol

In the unlikely event that Bob finds two different keys that satisfy (10.4), he indicates to Alice that the protocol failed, and they start over. Otherwise, Bob computes $\ell \leftarrow D(k, c_2)$ and $s \leftarrow D(k, c_1)$, and sends ℓ back to Alice.

3. Alice locates puzzle P'_ℓ and sets $s \leftarrow s_\ell$. Both parties now know the shared secret $s \in \mathcal{M}$.

Clearly, when the protocol terminates successfully, both parties agree on the same secret $s \in \mathcal{M}$. Moreover, when $|\mathcal{M}|$ is much larger than $|\mathcal{K}|$, the protocol is very likely to terminate successfully, because under these conditions (10.4) is likely to have a unique solution.

The work for each party in this protocol is as follows:

$$\text{Alice's work} = O(L), \qquad \text{Bob's work} = O(|\mathcal{K}|).$$

Hence, to make the workload for the two parties about the same we need to set $L \approx |\mathcal{K}|$. Either way, the size of L and $\mathcal{K}$ needs to be within reason so that both parties can perform the computation in a reasonable time. For example, one can set $L \approx |\mathcal{K}| \approx 2^{30}$. When using AES one can force $\mathcal{K}$ to have size 2^{30} by fixing the 98 most significant bits of the key to zero.

Security. The adversary sees the protocol transcript which includes all the puzzles and the quantity ℓ sent by Bob. Since the adversary does not know which puzzle Bob picked, intuitively, he needs to solve all puzzles until he finds puzzle P_ℓ. Thus, to recover $s \in \mathcal{M}$ the adversary must solve L puzzles each one taking $O(|\mathcal{K}|)$ time to solve. Overall, the adversary must spend time $O(L|\mathcal{K}|)$.

One can make this argument precise, by modeling the block cipher $\mathcal{E}$ as an ideal cipher, as we did in Section 4.7. We can assume that $|\mathcal{K}|$ is poly-bounded, and that $|\mathcal{M}|$ is super-poly. Then the analysis shows that if the adversary makes at most Q queries to the ideal cipher, then its probability of learning the secret $s \in \mathcal{M}$ is bounded by approximately $Q/L|\mathcal{K}|$. Working out the complete proof and the exact bound is a good exercise in working with the ideal cipher model.

Performance. Suppose we set $L \approx |\mathcal{K}|$. Then the adversary must spend time $O(L^2)$ to break the protocol, while each participant spends time $O(L)$. This gives a quadratic gap between the work of the participants and the work to break the protocol. Technically speaking, this doesn't satisfy our definitions of security — with constant work the adversary has advantage about $1/L^2$

which is non-negligible. Even worse, in practice one would have to make L extremely large to have a reasonable level of security against a determined attacker. The resulting protocol is then very inefficient.

Nevertheless, the Merkle puzzles protocol is very elegant and shows what can be done using block ciphers alone. As the story goes, Merkle came up with this clever protocol while taking a seminar as an undergraduate student. The professor gave the students the option of submitting a research paper instead of taking the final exam. Merkle submitted his key exchange protocol as the research project. These ideas, however, were ahead of their time and the professor rejected the paper. Merkle still had to take the final exam. Subsequently, for his Ph.D. work, Merkle chose to move to a different school to work with Martin Hellman.

It is natural to ask if a better key exchange protocol, based on block ciphers, can achieve better than quadratic separation between the participants and the adversary. Unfortunately, a result by Impagliazzo and Rudich [92] suggests that one cannot achieve better separation using block ciphers alone.

10.9 A fun application: accumulators

In Section 8.9 we saw how Alice can use a Merkle tree to commit to an ordered tuple of elements $(x_1, \ldots, x_n) \in \mathcal{X}^n$. She can later prove to Bob that x_i is the value at position i, for any $1 \leq i \leq n$ of her choice, where each such proof is a sequence of $\log_2 n$ hashes.

The tools developed in this chapter give a very different construction that has some advantages over Merkle trees. Our goal here is to show how Alice can commit to an *unordered* set $S = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$, so that she can later prove to Bob that a particular $x \in \mathcal{X}$ satisfies $x \in S$ (a membership proof), or satisfies $x \notin S$ (a non-membership proof). A commitment scheme for unordered sets that supports both membership and non-membership proofs is called an **accumulator**. An accumulator is said to be a **dynamic accumulator** if it is easy to add elements to the set S. In particular, there is an efficient algorithm that takes as input a commitment to a set S along with an element $x \notin S$, and outputs a commitment to the set $S' := S \cup \{x\}$

Groups of unknown order. We construct a simple dynamic accumulator using an algebraic tool called a **group of unknown order** or **GUO**. A GUO is a pair $(\mathbb{G}, g)$, where $\mathbb{G}$ is the description of a finite abelian group, and g is an element of $\mathbb{G}$. As the name suggests, it should be difficult to deduce the size of $\mathbb{G}$ from $(\mathbb{G}, g)$. A group of unknown order is generated by a probabilistic group generation algorithm GGen that is invoked as $\text{GGen}() \to (\mathbb{G}, g)$.

We say that GGen satisfies the **strong RSA assumption** if it is difficult to find a non-trivial root of g. In particular, it should be difficult to find a pair $(x, e) \in \mathbb{G} \times \mathbb{Z}$, such that $x^e = g$ and $e \neq \pm 1$. For an adversary $\mathcal{A}$ we define the strong RSA game as follows: the game begins by running $(\mathbb{G}, g) \xleftarrow{\text{R}} \text{GGen}()$ and sending $(\mathbb{G}, g)$ to $\mathcal{A}$. Then $\mathcal{A}$ outputs (x, e) and wins the game if $x^e = g$ and $e \neq \pm 1$. We say that GGen satisfies the strong RSA assumption if no efficient adversary can win the strong RSA game with respect to GGen with non-negligible advantage.

This assumption is called *strong* RSA because the adversary is given g and gets to choose e however it wants (as long as $e \neq \pm 1$). In the standard RSA assumption (Definition 10.5), the adversary is given g, and must find an e-th root of g for a pre-specified e. Giving the adversary the flexibility to choose e makes the adversary's job easier. Hence, assuming that this seemingly easier problem is hard, is a stronger assumption than standard RSA.

If the order of g is a known value $u \in \mathbb{Z}$, then the strong RSA assumption is clearly false. The adversary could choose some integer $v > 1$ that is relatively prime to u, compute $x := g^{(v^{-1} \bmod u)}$, and output the pair (x, v). Then $x^v = g$, which breaks the assumption. Hence, for strong RSA to hold, the order of g in $\mathbb{G}$, or even a multiple of the order, must be unknown given $(\mathbb{G}, g)$.

There are a number of candidate groups of unknown order where the strong RSA assumption is believed to hold. We saw one example in Section 10.3: the group $\mathbb{Z}_n^*$ of integers modulo an RSA modulus $n = pq$, where $n \leftarrow \text{RSAGen}(\ell, e)$, for some ℓ and e. In particular, GGen operates as follows:

$$\text{GGen}() = \left\{ \; n \xleftarrow{\text{R}} \text{RSAGen}(\ell, e), \quad g \xleftarrow{\text{R}} \mathbb{Z}_n^*, \quad \text{output } (n, g) \; \right\}.$$

The order of $\mathbb{Z}_n^*$ is $\varphi(n) = (p-1)(q-1)$ which is, as far as we know, difficult to compute just given n. Moreover, the strong RSA assumption is believed to hold for this GGen. We will discuss other candidate groups of unknown order at the end of the section.

An accumulator construction. Let's assume that a group of unknown order $(\mathbb{G}, g)$ has already been generated and is known to all parties. We treat $(\mathbb{G}, g)$ as the public parameters of the scheme.

In addition, the accumulator construction needs a collision resistant hash function $H : \mathcal{X} \to \mathit{Primes}(L)$, where $\mathit{Primes}(L)$ is the set of smallest L prime numbers: $\mathit{Primes}(L) := \{2, 3, 5, \ldots, p_L\}$ where p_L is the L-th prime number. We need L to be sufficiently large so that H can be collision resistant. Taking $L = 2^{256}$ is sufficient.

The **GUO accumulator**, an accumulator from a group of unknown order, works as follows:

- Alice commits to an unordered set $S = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$ as follows:

$$C(S) := \left\{ \begin{array}{l} e_i \leftarrow H(x_i) \;\text{ for } i = 1, \ldots, n \\ E \leftarrow e_1 \cdot e_2 \cdots e_n \in \mathbb{Z}, \quad \text{output } c := g^E \end{array} \right\} \tag{10.5}$$

  She sends $c = C(S) \in \mathbb{G}$ to Bob as a short commitment to the entire set S.

  This accumulator is clearly dynamic: given a commitment c to a set S along with an element $x \notin S$, the commitment to $S' := S \cup \{x\}$ is simply $c' := c^{H(x)}$.

- **A membership proof** for $x \in \mathcal{X}$ with respect to c. Alice has (S, x), Bob has (c, x), and Alice wants to convince Bob that x is in S. Observe that if $x \in S$ then $\beta := E/H(x)$ is an integer, where E is defined in (10.5).
  - Alice can prove to Bob that x is in S by computing $b \leftarrow g^\beta = g^{E/H(x)} \in \mathbb{G}$ and outputting the short proof $\pi_x := b$.
  - Bob verifies the proof $\pi_x = b$ by checking that $b^{H(x)} = c$ holds in $\mathbb{G}$.

- **A non-membership proof** for $x \in \mathcal{X}$ with respect to c. Alice has (S, x), Bob has (c, x), and Alice wants to convince Bob that x is not in S. We will argue below that if $x \notin S$ then the integers $H(x)$ and E must be relatively prime. The extended Euclidean algorithm can then be used to find $\beta, \gamma \in \mathbb{Z}$ such that

$$\beta \cdot H(x) + \gamma \cdot E = 1 \qquad \text{and} \qquad |\gamma| \leq H(x).$$

  - Alice proves to Bob that x is not in S by computing $b \leftarrow g^\beta$ and outputting the short proof $\pi_{\neg x} := (b, \gamma) \in \mathbb{G} \times \mathbb{Z}$.

– Bob verifies the proof $\pi_{\neg x} = (b, \gamma)$ by checking that $b^{H(x)} \cdot c^{\gamma} = g$.

That's the entire construction. We say that an accumulator is **correct** if for all sets $S \subseteq \mathcal{X}$ and all $x \in \mathcal{X}$, if $x \in S$ then Alice can generate a membership proof for x, and if $x \notin S$ then Alice can generate a non-membership proof for x. For the GUO accumulator, if $x \in S$ then Alice can always generate a membership proof for x. However, if $x \notin S$, Alice can only generate a non-membership proof when $\gcd(H(x), E) = 1$, for E is defined in (10.5). Let's show that it is infeasible to find a set S and an $x \notin S$ for which $\gcd(H(x), E) > 1$. The argument proceeds in two steps. First, because $H(x)$ is a prime number, this would imply that $H(x)$ must be one of the prime factors of E. Second, the only way $H(x)$ is one of the prime factors of E is if there is some $y \in S$ such that $H(x) = H(y)$. Since $x \notin S$ and $y \in S$ we deduce that $x \neq y$ and $H(x) = H(y)$. Then x and y are a collision for H, which contradicts the collision resistance of H. Therefore, if H is collision resistant, then $\gcd(H(x), E) = 1$ for any explicit $x \notin S$, and Alice can generate a non-membership proof for this x. We conclude that the GUO accumulator is correct, but correctness relies on the collision resistance of H.

Security of the GUO accumulator. A correct accumulator is said to be secure if no efficient adversary can simultaneously prove that $x \in S$ and $x \notin S$, for some committed set S and some $x \in \mathcal{X}$. More precisely, no efficient adversary can output a 4-tuple $(c, x, \pi_x, \pi_{\neg x})$ so that (i) π_x is a valid membership proof for x with respect to c, and (ii) $\pi_{\neg x}$ is a valid non-membership proof for x with respect to c.

For the GUO accumulator, suppose an adversary breaks security by outputting a 4-tuple $\big(c, x, a, (b, \gamma)\big)$ where both the membership proof a and the non-membership proof (b, γ) are valid. Then:

$$a^{H(x)} = c \qquad \text{and} \qquad b^{H(x)} \cdot c^{\gamma} = g.$$

The left equality implies $(a^{\gamma})^{H(x)} = c^{\gamma}$. Substituting $(a^{\gamma})^{H(x)}$ for c^{γ} in the right equality gives $(b \cdot a^{\gamma})^{H(x)} = g$, which breaks the strong RSA assumption. It follows that if the strong RSA assumption holds for GGen then the GUO accumulator is secure.

Accumulator proof size. Membership and non-membership proofs for the GUO accumulator are quite short; their size is independent of the number of elements n in the accumulator. This is better than Merkle tree proofs where the proof size grows with $\log_2 n$. In practice, the GUO proofs are shorter than Merkle tree proofs whenever $n > 256$.

GUO accumulator proofs have another advantage over Merkle tree proofs: Alice can aggregate multiple proofs into a single proof. Suppose Alice wants to prove that both x and y are in S, for some $x \neq y$. Alice can do so by producing a single proof $\pi_{xy} := b$, where $b \leftarrow g^{E/(H(x) \cdot H(y))} \in \mathbb{G}$. Bob verifies the proof by checking that $b^{H(x) \cdot H(y)} = c$. Hence, a single group element can convince Bob that both x and y are in S. If needed, Bob can recover the individual proofs π_x and π_y from π_{xy} by computing $\pi_x \leftarrow b^{H(y)}$ and $\pi_y \leftarrow b^{H(x)}$. Hence, the aggregate proof b is purely a proof compression mechanism that cannot harm security. Notice that Bob has to do a little more work to verify the aggregate proof π_{xy} because the exponent $H(x) \cdot H(y)$ is a little bigger than usual.

An interesting property of aggregate proofs is that *anyone* can aggregate membership proofs. Suppose Carol obtains distinct $x, y \in S$, along with their membership proofs $\pi_x = g^{E/H(x)}$ and $\pi_y = g^{E/H(y)}$. Carol can construct the short aggregate proof $\pi_{xy} = g^{E/(H(x) \cdot H(y))}$ herself using Shamir's trick described in Theorem 10.6. Indeed, $\pi_x^{H(x)} = \pi_y^{H(y)} = g^E$, and therefore applying

Theorem 10.6 lets Carol compute the aggregate proof $\pi_{xy} = g^{E/(H(x) \cdot H(y))}$. Hence, anyone can compress two membership proofs into one.

More generally, Alice can prove that an entire set T is a subset of S using a proof that is a *single* group element, namely $\pi_T := g^{E/\prod_{x \in T} H(x)}$. Naively, Bob's work to verify the proof would grow linearly in $|T|$ due to the larger exponent needed to verify the proof. However, using additional tools, Bob's work to verify the proof can be reduced to about the same as verifying a membership proof for a single element (see [31, §4.2]). Proofs of non-membership can be similarly aggregated: Alice can construct a constant size proof to prove that a set T is disjoint from S (i.e., $T \cap S = \emptyset$). This requires more sophisticated techniques that we will not describe here (see [31, §4.2]).

In comparison, Merkle tree proofs cannot be aggregated. Proving membership or non-membership for t elements requires a proof whose size grows linearly in t.

Remark 10.1. Suppose Alice commits to a set S of size n, and wants to compute the membership proof for every element $x \in S$. Naively, these n membership proofs take quadratic time in n to compute. However, we show in Exercise 10.32 that Alice can compute all n membership proofs in time proportional to $n \log n$. □

More groups of unknown order. The GUO accumulator needs a group of unknown order $(\mathbb{G}, g)$ where no one knows the order of the group $\mathbb{G}$. The group $\mathbb{Z}_n^*$, where $n = pq$ is an RSA modulus, is a good candidate. However, it requires a trusted party to generate two primes p and q, publish $n = pq$, and then erase the primes p and q. If the trusted party does not erase p and q, then it could create false membership and non-membership proofs for the GUO accumulator, as discussed in Exercise 10.31.

It would better if we had a group of unknown order that does not require a trusted setup. One option is to use the group $\mathbb{Z}_n^*$, where n is a large random integer, so large that no one can factor n. This way, no one would know the size of $\mathbb{Z}_n^*$. Unfortunately, n needs to be more than sixty thousand bits long to ensure that a random integer n is hard to factor with high probability. This makes the group operation very slow, and makes group elements quite large.

A better option is to use a family of groups, called **ideal class groups**, from algebraic number theory. In particular, the class group of an imaginary quadratic field is defined by a negative integer Δ, called the discriminant. Once Δ is fixed, the class group, denoted $\mathit{Cl}(\Delta)$, is completely defined. For cryptographic applications one typically takes Δ to be a negative prime number where $\Delta \equiv 1 \pmod{4}$. The group size is then approximately $|\Delta|^{1/2}$. The best known algorithm for computing the size of $\mathit{Cl}(\Delta)$ takes sub-exponential time in $\log|\Delta|$. In practice, choosing the discriminant Δ as a random 2048 bit prime seems sufficient to obtain a group of unknown order. See [43] for more information about computing in ideal class groups.

One caveat in using $\mathit{Cl}(\Delta)$ is that there is an efficient algorithm for computing square roots in this group [37]. As a result, we need to amend the Strong RSA assumption to require that the exponent e in the adversary's output satisfies $|e| > 2$. In addition, the number 2 needs to be removed from the range of the hash function H used in the GUO accumulator.

10.10 Notes

Citations to the literature to be added.

10.11 Exercises

10.1 (Computationally unbounded adversaries). Show that an anonymous key exchange protocol P (as in Definition 10.1) cannot be secure against a computationally unbounded adversary. This explains why all protocols in this chapter must rely on computational assumptions.

10.2 (Key exchange requires randomness from both parties). Consider an anonymous key exchange protocol P between two probabilistic machines A and B (as in Definition 10.1). Suppose that one of the machines is deterministic, so that it uses no random bits when participating in the protocol. Show that protocol P cannot be secure: there is an adversary $\mathcal{A}$ for which $\mathrm{AnonKE}\mathsf{adv}[\mathcal{A}, P] = 1$. This explains why in all the protocols in this chapter, *both* parties are randomized.

10.3 (DDH PRG). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Consider the following PRG defined over $(\mathbb{Z}_q^2,\ \mathbb{G}^3)$:

$$G(\alpha, \beta) := (g^{\alpha}, g^{\beta}, g^{\alpha\beta}).$$

Show that G is a secure PRG assuming DDH holds in $\mathbb{G}$.

10.4 (The Naor-Reingold PRF). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let us show that the following PRF defined over $\big(\mathbb{Z}_q^{n+1},\ \{0,1\}^n,\ \mathbb{G}\big)$ is secure assuming DDH holds in $\mathbb{G}$:

$$F_{\mathrm{NR}}\Big(\ (\alpha_0, \alpha_1, \ldots, \alpha_n),\ (x_1, \ldots, x_n)\ \Big) := g^{\left(\alpha_0 \cdot \alpha_1^{x_1} \cdots \alpha_n^{x_n}\right)}$$

This secure PRF is called the Naor-Reingold PRF.

(a) We prove security of F_{NR} using Exercise 4.18. First, show that F_{NR} is an *augmented tree construction* constructed from the PRG: $G_{\mathrm{NR}}(\alpha, g^{\beta}) := (g^{\beta}, g^{\alpha\beta})$.

(b) Second, show that G_{NR} satisfies the hypothesis of Exercise 4.18 part (b), assuming DDH holds in $\mathbb{G}$. Use the result of Exercise 10.11.

Security of F_{NR} now follows from Exercise 4.18 part (b).

Discussion: See Exercise 11.2 for a simpler PRF from the DDH assumption, but in the random oracle model.

10.5 (Random self-reduction for CDH (I)). Consider a specific cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. For $u = g^{\alpha} \in \mathbb{G}$ and $v = g^{\beta} \in \mathbb{G}$, define $[u, v] = g^{\alpha\beta}$, which is the solution instance (u, v) of the CDH problem. Consider the randomized mapping from $\mathbb{G}^2$ to $\mathbb{G}^2$ that sends (u, v) to $(\tilde{u}, v)$, where

$$\rho \xleftarrow{\text{R}} \mathbb{Z}_q, \quad \tilde{u} \leftarrow g^{\rho} u.$$

Show that

(a) $\tilde{u}$ is uniformly distributed over $\mathbb{G}$;

(b) $[\tilde{u}, v] = [u, v] \cdot v^{\rho}$.

10.6 (Random self-reduction for CDH (II)). Continuing with the previous exercise, suppose $\mathcal{A}$ is an efficient algorithm that solves the CDH problem with success probability ϵ on random inputs. That is, if $u, v \in \mathbb{G}$ are chosen at random, then $\Pr[\mathcal{A}(u,v) = [u,v]] = \epsilon$, where the probability is over the random choice of u and v, as well as any random choices made by $\mathcal{A}$. Using $\mathcal{A}$, construct an efficient algorithm $\mathcal{B}$ that solves the CDH problem with success probability ϵ for *all* inputs. More precisely, for all $u, , v \in \mathbb{G}$, we have $\Pr[\mathcal{B}(u,v) = [u,v]] = \epsilon$, where the probability is now only over the random choices made by $\mathcal{B}$.

Remark: If we iterate $\mathcal{B}$ on the same input (u,v) many times, say $n\lceil 1/\epsilon \rceil$ times for some n, at least one of these iterations will output the correct result $[u,v]$ with probability $1-(1-\epsilon)^{n\lceil 1/\epsilon \rceil} \geq 1 - \exp(-n)$. Unfortunately, assuming the DDH is true, we will have no way of knowing which of these outputs is the correct result.

10.7 (An alternative DDH characterization). Let $\mathbb{G}$ by a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $\mathcal{P}$ be the uniform distribution over $\mathbb{G}^3$. Let $\mathcal{P}_{\text{dh}}$ be the uniform distribution over the set of all DH-triples $(g^\alpha, g^\beta, g^{\alpha\beta})$. Let $\mathcal{P}_{\text{ndh}}$ be the uniform distribution over the set of all non-DH-triples $(g^\alpha, g^\beta, g^\gamma), \gamma \neq \alpha\beta$.

(a) Show that the statistical distance (as in Definition 3.5) between $\mathcal{P}$ and $\mathcal{P}_{\text{ndh}}$ is $1/q$.

(b) Using part (a), deduce that under the DDH assumption, the distributions $\mathcal{P}_{\text{dh}}$ and $\mathcal{P}_{\text{ndh}}$ are computationally indistinguishable (as in Definition 3.4). In particular, show that for every adversary $\mathcal{A}$, we have $\text{Dist}\mathsf{adv}[\mathcal{A}, \mathcal{P}_{\text{dh}}, \mathcal{P}_{\text{ndh}}] \leq \text{DDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}] + 1/q$.

10.8 (Random self-reduction for DDH (I)). Consider a specific cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. Let $\mathbf{DH}$ be the set of all DH-triples, i.e.,

$$\mathbf{DH} := \{(g^\alpha, g^\beta, g^{\alpha\beta}) \in \mathbb{G}^3 : \alpha, \beta \in \mathbb{Z}_q\}.$$

For fixed $u \in \mathbb{G}$, and let $\mathbf{T}_u$ be the subset of $\mathbb{G}^3$ whose first coordinate is u. Consider the randomized mapping from $\mathbb{G}^3$ to $\mathbb{G}^3$ that sends (u,v,w) to (u, v^*, w^*), where

$$\sigma \xleftarrow{\text{R}} \mathbb{Z}_q, \quad \tau \xleftarrow{\text{R}} \mathbb{Z}_q, \quad v^* \leftarrow g^\sigma v^\tau, \quad w^* \leftarrow u^\sigma w^\tau.$$

Prove the following:

(a) if $(u,v,w) \in \mathbf{DH}$, then (u, v^*, w^*) is uniformly distributed over $\mathbf{DH} \cap \mathbf{T}_u$;

(b) if $(u,v,w) \notin \mathbf{DH}$, then (u, v^*, w^*) is uniformly distributed over $\mathbf{T}_u$.

10.9 (Random self-reduction for DDH (II)). Continuing with the previous exercise, consider the randomized mapping from $\mathbb{G}^3$ to $\mathbb{G}^3$ that sends (u,v,w) to $(\tilde{u}, v, \tilde{w})$, where

$$\rho \xleftarrow{\text{R}} \mathbb{Z}_q, \quad \tilde{u} \leftarrow g^\rho u, \quad \tilde{w} \leftarrow v^\rho w.$$

Prove the following:

(a) $\tilde{u}$ is uniformly distributed over $\mathbb{G}$;

(b) $(u,v,w) \in \mathbf{DH} \iff (\tilde{u}, v, \tilde{w}) \in \mathbf{DH}$;

(c) if we apply the randomized mapping from the previous exercise to $(\tilde{u}, v, \tilde{w})$, obtaining the triple $(\tilde{u}, v^*, \tilde{w}^*)$, then we have

- if $(u, v, w) \in \mathbf{DH}$, then $(\tilde{u}, v^*, \tilde{w}^*)$ is uniformly distributed over $\mathbf{DH}$;
- if $(u, v, w) \notin \mathbf{DH}$, then $(\tilde{u}, v^*, \tilde{w}^*)$ is uniformly distributed over $\mathbb{G}^3$.

10.10 (Random self-reduction for DDH (III)). Continuing with the previous exercise, prove the following. Suppose $\mathcal{A}$ is an efficient algorithm that takes as input three group elements and outputs a bit, and which satisfies the following property: if $\alpha, \beta, \gamma \in \mathbb{Z}_q$ are chosen at random, then

$$\left|\Pr[\mathcal{A}(g^\alpha, g^\beta, g^{\alpha\beta}) = 1] - \Pr[\mathcal{A}(g^\alpha, g^\beta, g^\gamma) = 1]\right| = \epsilon,$$

where the probability is over the random choice of α, β, γ, as well as any random choices made by $\mathcal{A}$. Assuming that $1/\epsilon$ is poly-bounded, show how to use $\mathcal{A}$ to build an efficient algorithm $\mathcal{B}$ that for all inputs (u, v, w) correctly decides whether or not $(u, v, w) \in \mathbf{DH}$ with negligible error probability. That is, adversary $\mathcal{B}$ may output an incorrect answer, but for all inputs, the probability that its answer is incorrect should be negligible.

Hint: Use a Chernoff bound.

10.11 (Multi-DDH (I)). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let n and m be positive integers. Define the following two distributions over $\mathbb{G}^{n+2nm}$:

$$\mathcal{D}: \quad g^{\alpha_i} \quad (i = 1, \ldots, n), \qquad\qquad g^{\beta_{ij}}, \quad g^{\alpha_i\beta_{ij}} \quad (i = 1, \ldots, n, \quad j = 1, \ldots, m),$$

and

$$\mathcal{R}: \quad g^{\alpha_i} \quad (i = 1, \ldots, n), \qquad\qquad g^{\beta_{ij}}, \quad g^{\gamma_{ij}} \quad (i = 1, \ldots, n, \quad j = 1, \ldots, m).$$

where the α_i's, β_{ij}'s, and γ_{ij}'s are uniformly and independently distributed over $\mathbb{Z}_q$. Show that under the DDH assumption, $\mathcal{D}$ and $\mathcal{R}$ are computationally indistinguishable (as in Definition 3.4). In particular, show that for every adversary $\mathcal{A}$ that distinguishes $\mathcal{D}$ and $\mathcal{R}$, there exists a DDH adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) such that

$$\mathsf{Distadv}[\mathcal{A}, \mathcal{D}, \mathcal{R}] \leq 1/q + \mathrm{DDH}\mathsf{adv}[\mathcal{B}, \mathbb{G}].$$

Hint: Apply Exercises 10.7, 10.8, and 10.9.

10.12 (Multi-DDH (II)). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $n \leq m$ be positive integers. Define the following two distributions over $\mathbb{G}^{n \cdot m + n + m}$:

$$\begin{aligned} \mathcal{D}: \quad & g^{\alpha_i} \quad (i = 1, \ldots, n), \qquad g^{\beta_j} \quad (j = 1, \ldots, m) \\ & g^{\alpha_i\beta_j} \quad (i = 1, \ldots, n, \quad j = 1, \ldots, m), \end{aligned}$$

and

$$\begin{aligned} \mathcal{R}: \quad & g^{\alpha_i} \quad (i = 1, \ldots, n), \qquad g^{\beta_j} \quad (j = 1, \ldots, m) \\ & g^{\gamma_{ij}} \quad (i = 1, \ldots, n, \quad j = 1, \ldots, m). \end{aligned}$$

where the α_i's, β_j's, and γ_{ij}'s are uniformly and independently distributed over $\mathbb{Z}_q$. Show that under the DDH assumption, $\mathcal{D}$ and $\mathcal{R}$ are computationally indistinguishable (as in Definition 3.4).

In particular, show that for every adversary $\mathcal{A}$ that distinguishes $\mathcal{D}$ and $\mathcal{R}$, there exists a DDH adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) such that

$$\mathsf{Distadv}[\mathcal{A}, \mathcal{D}, \mathcal{R}] \leq n \cdot (1/q + \mathrm{DDH}\mathsf{adv}[\mathcal{B}, \mathbb{G}]).$$

Hint: First give a proof for the case $n = 1$ using the results of Exercise 10.7 and Exercise 10.8, and then generalize to arbitrary n using a hybrid argument.

Discussion: This result gives us a DDH-based PRG G defined over $(\mathbb{Z}_q^{n+m},\ \mathbb{G}^{n \cdot m+n+m})$, with a nice expansion rate, given by

$$G\Big(\ \{\alpha_i\}_{i=1}^{n},\ \{\beta_j\}_{j=1}^{m}\ \Big) := \Big(\ \{g^{\alpha_i}\}_{i=1}^{n},\ \{g^{\beta_j}\}_{j=1}^{m},\ \{g^{\alpha_i \beta_j}\}_{\substack{i=1,\dots,n \\ j=1,\dots,m}}\ \Big).$$

The reader should also compare this exercise to the previous one: security in this construction degrades linearly in n, while the security in the construction in the previous exercise does not degrade at all as n increases.

10.13 (Matrix DDH). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let n and m be positive integers, and assume $n \leq m$. For $A = (\alpha_{ij}) \in \mathbb{Z}_q^{n \times m}$ (i.e., A is an $n \times m$ matrix with entries in $\mathbb{Z}_q$), let g^A be the $n \times m$ matrix whose entry at row i column j is the group element $g^{\alpha_{ij}}$. For $k = 1, \dots, n$, define the random variable $R(k)$ to be a random matrix uniformly distributed over all $n \times m$ matrices over $\mathbb{Z}_q$ of of rank at most k. Let $1 \leq k_1 < k_2 \leq n$. Show that $g^{R(k_1)}$ and $g^{R(k_2)}$ are computationally indistinguishable under the DDH. In particular, show that for every adversary $\mathcal{A}$ that distinguishes $g^{R(k_1)}$ and $g^{R(k_2)}$ there exists a DDH adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) such that

$$\mathsf{Distadv}[\mathcal{A}, g^{R(k_1)}, g^{R(k_2)}] \leq (k_2 - k_1) \cdot (1/q + \mathrm{DDH}\mathsf{adv}[\mathcal{B}, \mathbb{G}]).$$

Hint: Use the fact that if $A \in \mathbb{Z}_q^{n \times m}$ is a fixed matrix of rank k, and if $U \in \mathbb{Z}_q^{n \times n}$ and $V \in \mathbb{Z}_q^{m \times m}$ are a random *invertible* matrices, then the matrix $UAV \in \mathbb{Z}_q^{n \times m}$ is uniformly distributed over all $n \times m$ matrices of rank k. You might also try to prove this fact, which is not too hard.

Discussion: For $k_1 = 1$ and $k_2 = n$, this result is almost the same as Exercise 10.12. In this sense, this exercise is a generalization of Exercise 10.12.

10.14 (A trapdoor function from DDH). Let $q := |\mathbb{G}|$ and $n \geq 2\log_2 q$. In this exercise we construct a trapdoor function scheme (G, F, I) defined over $(\mathcal{X}, \mathcal{Y})$, where $\mathcal{X} := \{0,1\}^n$ and $\mathcal{Y} := \mathbb{G}^n$. Security will follow from DDH in $\mathbb{G}$. The scheme works as follows:

- Algorithm G chooses a random matrix $A \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times n}$ and outputs $pk \leftarrow g^A \in \mathbb{G}^{n \times n}$ and $sk \leftarrow A^{-1} \in \mathbb{Z}_q^{n \times n}$ (the notation g^A is defined in the previous exercise). Note that A is invertible with overwhelming probability, so that sk is well defined.
- For $pk = g^A$ and $x \in \mathcal{X}$ define $F(pk, x) := g^{A \cdot x} \in \mathcal{Y}$.

(a) Show how to use sk to invert the function: given sk and $y := g^{A \cdot x} \in \mathcal{Y}$ as input, show how to compute $x \in \mathcal{X}$.

(b) Show that if DDH holds in $\mathbb{G}$ then the function is one way in the sense of Definition 10.3.

Hint: Use Exercise 10.13, and the DDH assumption, to argue that the advantage of an inversion adversary will be only negligibly affected if the public key matrix $A \in \mathbb{Z}_q^{n\times n}$ is replaced by a random matrix $A' \in \mathbb{Z}_q^{n\times n}$ of rank 1. Once the change is made, using the assumption that $n \geq 2\log_2 q$, argue that it is likely that there are several 0/1 vectors $x' \in \mathcal{X}$ such that $F(pk, x) = F(pk, x')$, and so the adversay is likely to pick the wrong inverse.

Discussion: This trapdoor function is quite different from the RSA trapdoor permutation. In particular, it is not known how to randomly sample from its image without first sampling a preimage. This prevents it from being used in certain key applications, such as full domain hash signatures discussed in Section 13.3.

10.15 (A trapdoor test). Consider a specific cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. Let $u \in \mathbb{G}$ and $f : \mathbb{G} \to \mathbb{G}^3$. Now set

$$\sigma \xleftarrow{\text{R}} \mathbb{Z}_q, \quad \tau \xleftarrow{\text{R}} \mathbb{Z}_q, \quad \bar{u} \leftarrow g^\sigma u^\tau, \quad (v, w, \bar{w}) \leftarrow f(\bar{u}).$$

Let S be the event that (u, v, w) and $(\bar{u}, v, \bar{w})$ are both DH-triples. Let T be the event that $\bar{w} = v^\sigma w^\tau$. Show that:

(a) $\bar{u}$ is uniformly distributed over $\mathbb{G}$;

(b) $\Pr[S \wedge \neg T] = 0$;

(c) $\Pr[\neg S \wedge T] \leq 1/q$.

Remark: This result gives us a kind of trapdoor test. Suppose a group element $u \in \mathbb{G}$ is given (it could be chosen at random or adversarially chosen). Then we can generate a random element $\bar{u}$ and a "trapdoor" (σ, τ). Using this trapdoor, given group elements $v, w, \bar{w} \in \mathbb{G}$ (possibly adversarially chosen in a way that depends on $\bar{u}$), we can reliably test if (u, v, w) and $(\bar{u}, v, \bar{w})$ are both DH-triples, even though we do not know either $\mathsf{Dlog}_g(u)$ or $\mathsf{Dlog}_g(\bar{u})$, and even though we cannot tell whether (u, v, w) and $(\bar{u}, v, \bar{w})$ are individually DH-triples. This rather technical result has several nice applications, one of which is developed in the following exercise.

10.16 (A CDH self-corrector). Consider a specific cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. Let $\mathcal{A}$ be an efficient algorithm with the following property: if $\alpha, \beta \in \mathbb{Z}_q$ are chosen at random, then $\Pr[\mathcal{A}(g^\alpha, g^\beta) = g^{\alpha\beta}] = \epsilon$. Here, the probability is over the random choice of α and β, as well as any random choices made by $\mathcal{A}$. Assuming $1/\epsilon$ is poly-bounded and $|\mathbb{G}|$ is super-poly, show how to use $\mathcal{A}$ to build an efficient algorithm $\mathcal{B}$ that solves the CDH problem on all inputs with negligible error probability; that is, on every input (g^α, g^β), algorithm $\mathcal{B}$ outputs a single group element w, and $w \neq g^{\alpha\beta}$ with negligible probability (and this probability is just over the random choices made by $\mathcal{B}$).

Here is a high-level sketch of how $\mathcal{B}$ might work on input (u, v).

```
somehow choose $\bar{u} \in \mathbb{G}$
somehow use $\mathcal{A}$ to generate lists $L, \bar{L}$ of group elements
for each $w$ in $L$ and each $\bar{w}$ in $\bar{L}$ do
    if $(u, v, w)$ and $(\bar{u}, v, \bar{w})$ are both DH-triples then
        output $w$ and halt
output an arbitrary group element
```

As stated, this algorithm is not fully specified. Nevertheless, you can use this rough outline, combined with the CDH random self reduction in Exercise 10.5 and the trapdoor test in Exercise 10.15, to prove the desired result.

For the next problem, we need the following notions from complexity theory:

- *We say problem A is* deterministic poly-time reducible *to problem B if there exists a deterministic algorithm R for solving problem A on all inputs that makes calls to a subroutine that solves problem B on all inputs, where the running time of R (not including the running time for the subroutine for B) is polynomial in the input length.*
- *We say that A and B are* deterministic poly-time equivalent *if A is deterministic poly-time reducible to B and B is deterministic poly-time reducible to A.*

10.17 (Problems equivalent to CDH). Consider a specific cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. Show that the following problems are deterministic poly-time equivalent:

(a) Given g^{α} and g^{β}, compute $g^{\alpha\beta}$ (this is just the Computational Diffie-Hellman problem).

(b) Given g^{α}, compute $g^{(\alpha^2)}$.

(c) Given g^{α} with $\alpha \neq 0$, compute $g^{1/\alpha}$.

(d) Given g^{α} and g^{β} with $\beta \neq 0$, compute $g^{\alpha/\beta}$.

Note that all problem instances are defined with respect to the same group $\mathbb{G}$ and generator $g \in \mathbb{G}$.

10.18 (System parameters). In formulating the discrete-log Attack Game 10.4, we assume that the description of $\mathbb{G}$, including $g \in \mathbb{G}$ and q, is a system parameter that is generated once and for all at system setup time and shared by all parties involved. This parameter may be generated via some randomized process, in which case the advantage $\epsilon = \mathrm{DL}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ is a probability over the choice of system parameter, as well as the random choice of $\alpha \in \mathbb{Z}_q$ made by the challenger and any random choices made by adversary. So we can think of the system parameter as a random variable Λ, and for any specific system parameter Λ_0, we can consider the corresponding conditional advantage $\epsilon(\Lambda_0)$ given that $\Lambda = \Lambda_0$, which is a probability just over the random choice of $\alpha \in \mathbb{Z}_q$ made by the challenger and any random choices made by adversary. Let us call Λ_0 a "vulnerable" parameter if $\epsilon(\Lambda_0) \geq \epsilon/2$.

(a) Prove that the probability that Λ is vulnerable is at least $\epsilon/2$.

Note that even if an adversary breaks the DL with respect to a randomly generated system parameter, there could be many particular system parameters for which the adversary cannot or will not break the DL (it is helpful to imagine an adversary that is all powerful yet capricious, who simply refuses to break the DL for certain groups and generators which he finds distasteful). This result says, however, that there is still a non-negligible fraction of vulnerable system parameters for which the adversary breaks the DL.

(b) State and prove an analogous result for the CDH problem.

(c) State and prove an analogous result for the DDH problem.

10.19 (Choice of generators). In formulating the DL, CDH, and DDH assumptions, we work with a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. We do not specify how the generator g is chosen. Indeed, it may be desirable to choose a specific g that allows for more efficient implementations. Conceivably, such a g could be a "weak" generator that makes it easier for an adversary to break the DL, CDH, or DDH assumptions. So to be on the safe side, we might insist that the generator g is uniformly distributed over $\mathbb{G}\backslash\{1\}$. If we do this, we obtain new assumptions, which we call the rDL, rCDH, and rDDH assumptions. Show that:

(a) the rDL and DL assumptions are equivalent;

(b) the rCDH and CDH assumptions are equivalent;

(c) the DDH assumption implies the rDDH assumption.

Hint: To start with, you might first consider the setting where we are working with a specific group, then generalize your result to incorporate all the aspects of the asymptotic attack game (see Section 10.5.2), including the security parameter and the system parameter (where the group is selected at system setup time).

Remark: The rDDH assumption is not known to imply the DDH assumption, so for applications that use the DDH assumption, it seems safest to work with a random generator.

10.20 (Relation finding is as hard as discrete log). Let $\mathbb{G}$ be a cyclic group of prime order q, and let n be a poly-bounded parameter. A relation finding algorithm $\mathcal{A}$ takes as input $(g_1, \ldots, g_n) \in \mathbb{G}^n$, and outputs $\boldsymbol{\alpha} := (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$ such that $g_1^{\alpha_1} \cdots g_n^{\alpha_n} = 1$ where $\boldsymbol{\alpha} \neq (0, \ldots, 0)$. Such an $\boldsymbol{\alpha} \in \mathbb{Z}_q^n$ is called a *relation* for $g_1, \ldots, g_n \in \mathbb{G}$. Let us show that finding a relation among a random set of group elements is as hard as computing discrete log in $\mathbb{G}$. For a relation finding algorithm $\mathcal{A}$, random generators $(g_1, \ldots, g_n) \xleftarrow{R} \mathbb{G}^n$, and $(\alpha_1, \ldots, \alpha_n) \xleftarrow{R} \mathcal{A}(g_1, \ldots, g_n)$, define

$$\mathrm{REL}\mathsf{adv}[\mathcal{A}, \mathbb{G}] := \Pr\big[g_1^{\alpha_1} \cdots g_n^{\alpha_n} = 1\big].$$

Show that for every relation finding algorithm $\mathcal{A}$, there exists a DL adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that $\mathrm{REL}\mathsf{adv}[\mathcal{A}, \mathbb{G}] \leq \mathrm{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}] + 1/q$.

10.21 (Collision resistance from discrete-log). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let n be a poly-bounded parameter. Let us define a hash function H defined over $(\mathbb{Z}_q^n, \mathbb{G})$. The hash function is parameterized by the group $\mathbb{G}$ and n randomly chosen group elements $g_1, \ldots, g_n \in \mathbb{G}$. For $(\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$, define

$$H(\alpha_1, \ldots, \alpha_n) := g_1^{\alpha_1} \cdots g_n^{\alpha_n}.$$

Our goal is to show that H is collision resistant under the DL assumption for $\mathbb{G}$. A collision finding adversary $\mathcal{A}$ for H takes as input $(g_1, \ldots, g_n) \in \mathbb{G}^n$ and outputs a collision for H. Use Exercise 10.20 to show that for every collision-finding adversary $\mathcal{A}$ for H, there exists a DL adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H] \leq \mathrm{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}] + 1/q$. Note that this result is a generalization of the collision resistant hash function from Section 10.6.1.

10.22 (Collision resistance in $\mathbb{Z}_p^*$). This exercise asks you to prove that the hash function presented in Section 8.5.1 is collision resistant under an appropriate DL assumption. Let us define things a bit more precisely. Let p be a large prime such that $q := (p-1)/2$ is also prime. The prime q is called a *Sophie Germain* prime, and p is sometimes called a "strong" prime. Such primes are often very convenient to use in cryptography. Suppose x is a randomly chosen integer in the range $[2, q]$ and y is a randomly chosen integer in the range $[1, q]$. These parameters define a hash function H that takes as input two integers in $[1, q]$ and outputs an integer in $[1, q]$, as specified in (8.3). Let $\mathbb{G}$ be the subgroup of order q in $\mathbb{Z}_p^*$, and consider the DL assumption for $\mathbb{G}$ with respect to a randomly chosen generator. Show that H is collision resistant under this DL assumption.

Hint: Use the fact that the map that sends $\alpha \in \mathbb{Z}_p^*$ to $\alpha^2 \in \mathbb{Z}_p^*$ is a group homomorphism with image $\mathbb{G}$ and kernel ± 1; also use the fact that there is an efficient algorithm for taking square roots in $\mathbb{Z}_p^*$.

10.23 (A broken CRHF). Consider the following variation of the hash construction in the previous exercise. Let p be a large prime such that $q := (p-1)/2$ is also prime. Let x and y be randomly chosen integers in the range $[2, p-2]$ (so neither can be $\pm 1 \pmod{p}$). These parameters define a hash function H that takes as input two integers in $[1, p-1]$ and outputs an integer in $[1, p-1]$, as follows:

$$H(a, b) := x^a y^b \bmod p.$$

Give an efficient, deterministic algorithm that takes as input p, x, y as above, and computes a collision on the corresponding H. Your algorithm should work for all inputs p, x, y.

10.24 (DDH is easy in groups of even order). We have restricted the DL, CDH, and DDH assumptions to prime order groups $\mathbb{G}$. Consider the DDH assumption for a cyclic group $\mathbb{G}$ of *even* order q with generator $g \in \mathbb{G}$. Except for dropping the restriction that q is prime, the attack game is identical to Attack Game 10.6. Give an efficient adversary that has advantage $1/2$ in solving the DDH for $\mathbb{G}$.

Remark: For a prime $p > 2$, the group $\mathbb{Z}_p^*$ is a cyclic group of even order $p-1$. This exercise shows that the DDH assumption is false in this group. Exercise 10.23 gives another reason to restrict ourselves to groups of prime order. See also Exercise 16.2 for a generalization of this result.

10.25 (RSA variant (I)). Let n be an RSA modulus generated by RSAGen(ℓ, e). Let X and X^* be random variables, where X is uniformly distributed over $\mathbb{Z}_n$ and X^* is uniformly distributed over $\mathbb{Z}_n^*$. Show that the statistical distance $\Delta[X, X^*]$ is less than $2^{-(\ell-2)}$.

10.26 (RSA variant (II)). In Theorem 10.5, we considered a variant of the RSA assumption where the challenger chooses the preimage x at random from $\mathbb{Z}_n^*$, rather than $\mathbb{Z}_n$. That theorem showed that the standard RSA assumption implies this variant RSA assumption. In this exercise, you are to show the converse. In particular, show that $\mathrm{RSA}\mathsf{adv}[\mathcal{A}, \ell, e] \le \mathrm{uRSA}\mathsf{adv}[\mathcal{B}, \ell, e] + 2^{-(\ell-2)}$ for every adversary $\mathcal{A}$.

Hint: Use the result of the previous exercise.

10.27 (A proper trapdoor permutation scheme based on RSA). As discussed in Section 10.3, our RSA-based trapdoor permutation scheme does not quite satisfy our definitions, simply because the domain on which it acts varies with the public key. This exercise shows one way to patch things up. Let ℓ and e be parameters used for RSA key generation, and let G be the key generation algorithm, which outputs a pair (pk, sk). Recall that $pk = (n, e)$, where n is an RSA modulus, which is the product of two ℓ-bit primes, and e is the encryption exponent. The secret key is $sk = (n, d)$, where d is the decryption exponent corresponding to the encryption exponent e.

Choose a parameter L that is a substantially larger than 2ℓ, so that $n/2^L$ is negligible. Let $\mathcal{X}$ be the set of integers in the range $[0, 2^L)$. We shall present a trapdoor permutation scheme (G, F^*, I^*), defined over $\mathcal{X}$. The function F^* takes two inputs: a public key pk as above and an integer $x \in \mathcal{X}$, and outputs an integer $y \in \mathcal{X}$, computed as follows. Divide x by n to obtain the integer quotient Q and remainder R, so that $x = nQ + R$ and $0 \leq R < n$. If $Q > 2^L/n - 1$, then set $S := R$; otherwise, set $S := R^e \bmod n$. Finally, set $y := nQ + S$.

(a) Show that $F^*(pk, \cdot)$ is a permutation on $\mathcal{X}$, and give an efficient inversion function I^* that satisfies $I^*(sk, F^*(pk, x)) = x$ for all $x \in \mathcal{X}$.

(b) Show under the RSA assumption, (G, F^*, I^*) is one-way.

10.28 (Random self-reduction for RSA). Suppose we run $(n, d) \xleftarrow{\text{R}} \text{RSAGen}(\ell, e)$. There could be "weak" RSA moduli n for which an adversary can break the the RSA assumption with some probability ϵ. More precisely, suppose that there is an efficient algorithm $\mathcal{A}$ such that for any such "weak" modulus n, if $x \in \mathbb{Z}_n^*$ is chosen at random, then $\Pr[\mathcal{A}(x^e) = x] \geq \epsilon$, where the probability is over the random choice of x, as well as any random choices made by $\mathcal{A}$. Using $\mathcal{A}$, construct an efficient algorithm $\mathcal{B}$ such that for every "weak" modulus n, and every $x \in \mathbb{Z}_n$, we have $\Pr[\mathcal{A}(x^e) = x] \geq \epsilon$, where the probability is now only over the random choices made by $\mathcal{B}$.

Hint: Use the randomized mapping from $\mathbb{Z}_n^*$ to $\mathbb{Z}_n^*$ that sends y to $\tilde{y}$, where $r \xleftarrow{\text{R}} \mathbb{Z}_n^*$, $\tilde{y} \leftarrow r^e y$. Show that for every $y \in \mathbb{Z}_n^*$, the value $\tilde{y}$ is uniformly distributed over $\mathbb{Z}_n^*$.

10.29 (n-product CDH). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. The following attack game defines the **n-product CDH problem** (here, n is a poly-bounded parameter, not necessarily constant). The challenger begins by choosing $\alpha_i \xleftarrow{\text{R}} \mathbb{Z}_q$ for $i = 1, \ldots, n$. The adversary then makes a sequence of queries. In each query, the adversary submits a proper subset of indices $S \subsetneq \{1, \ldots, n\}$, and the challenger responds with

$$g^{\prod_{i \in S} \alpha_i}.$$

The adversary wins the game if it outputs

$$g^{\alpha_1 \cdots \alpha_n}.$$

We relate the hardness of solving the n-product CDH problem to another problem, called the **n-power CDH problem**. In the attack game for this problem, the challenger begins by choosing $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q^*$, and gives

$$g, g^{\alpha}, \ldots, g^{\alpha^{n-1}}$$

to the adversary. The adversary wins the game if it outputs $g^{(\alpha^n)}$.

Show that if there is an efficient adversary $\mathcal{A}$ that breaks n-product CDH with non-negligible probability, then there is an efficient adversary $\mathcal{B}$ that breaks n-power CDH with non-negligible probability.

10.30 (Trapdoor collison resistance). Let us show that the collision resistant hash functions H_{dl} and H_{rsa}, presented in Section 10.6, are *trapdoor* collision resistant.

(a) Recall that H_{dl} is defined as $H_{\text{dl}}(\alpha, \beta) := g^\alpha h^\beta \in \mathbb{G}$, where g and h are parameters chosen at setup. Show that anyone who knows the discrete-log of h base g (the trapdoor), can break the 2nd-preimage resistance of H_{dl}. That is, given (α, β) as input, along with the trapdoor, one can efficiently compute $(\alpha', \beta') \neq (\alpha, \beta)$ such that $H_{\text{dl}}(\alpha', \beta') = H_{\text{dl}}(\alpha, \beta)$.

(b) Recall that H_{rsa} is defined as $H_{\text{rsa}}(a, b) := a^e y^b \in \mathbb{Z}_n$, where n, e and y are parameters chosen at setup. Show that anyone who knows the eth root of y in $\mathbb{Z}_n$ (the trapdoor), can break the 2nd-preimage resistance of H_{rsa}.

(c) Continuing with part (b), show that anyone who knows the factorization of n (the trapdoor), can invert H_{rsa}. That is, given $z \in Z_n$ as input, one can find (a, b) such that $H_{\text{rsa}}(a, b) = z$.

Discussion: Part (c) shows that the factorization of n is a "stronger" trapdoor for H_{rsa} than the eth root of y. The latter only breaks 2nd-preimage resistance of H_{rsa}, whereas the former enables complete inversion. Both trapdoors break collision resistance.

10.31 (An attack on the GUO accumulator). Consider again the accumulator described in Section 10.9 which uses a finite group of unknown order $(\mathbb{G}, g)$ and a hash function H. Let d be the unknown order of g in $\mathbb{G}$, and suppose that Mel somehow discovers the value of $d \in \mathbb{Z}$.

(a) Show that Mel can use d to create a false accumulator membership proof. Specifically, show that Mel can produce a tuple (S, c, x, b), where $S \subseteq \mathcal{X}$ is some set, $c = g^{\prod_{y \in S} H(y)} \in \mathbb{G}$ is the accumulator commitment to S, $x \in \mathcal{X}$ is not a member of S, but $b \in \mathbb{G}$ is a valid membership proof for x with respect to c, so that, $b^{H(x)} = c$. This b is a false membership proof for x in S. Show that Mel can produce the required tuple (S, c, x, b).

(b) Show that Mel can use d to create a false accumulator non-membership proof. Specifically, show that Mel can produce a tuple (S, c, x, b, γ), where S and c are as in part (a), $x \in \mathcal{X}$ is a member of S, but $(b, \gamma) \in \mathbb{G} \times \mathbb{Z}$ is a valid non-membership proof for x with respect to c. That is, $b^{H(x)} \cdot c^\gamma = g$. This (b, γ) is a false non-membership proof for x in S. Show that Mel can produce the required tuple (S, c, x, b, γ).

10.32 (Fast generation of accumulator proofs). Let $e_1 < \ldots < e_n$ be a set of distinct prime numbers, and let $E = e_1 \cdots e_n \in \mathbb{Z}$. Let $(\mathbb{G}, g)$ be a group of unknown order (GUO), as discussed in Section 10.9. Show how to generate the set of n group elements $\{g^{E/e_i}\}_{i=1}^n$ using only $n \log_2 n$ exponentiations in $\mathbb{G}$ where every exponent is no bigger than e_n. Your solution gives the algorithm mentioned in Remark 10.1.

Chapter 11

Public key encryption

In this chapter, we consider again the basic problem of encryption. As a motivating example, suppose Alice wants to send Bob an encrypted email message, even though the two of them do not share a secret key (nor do they share a secret key with some common third party). Surprisingly, this can be done using a technology called **public-key encryption**.

The basic idea of public-key encryption is that the receiver, Bob in this case, runs a key generation algorithm G, obtaining a pair of keys:

$$(pk, sk) \xleftarrow{\text{R}} G().$$

The key pk is Bob's *public key*, and sk is Bob's *secret key*. As their names imply, Bob should keep sk secret, but may publicize pk.

To send Bob an encrypted email message, Alice needs two things: Bob's email address, and Bob's public key pk. How Alice reliably obtains this information is a topic we shall explore later in Section 13.8. For the moment, one might imagine that this information is placed by Bob in some kind of public directory to which Alice has read-access.

So let us assume now that Alice has Bob's email address and public key pk. To send Bob an encryption of her email message m, she computes the ciphertext

$$c \xleftarrow{\text{R}} E(pk, m).$$

She then sends c to Bob, using his email address. At some point later, Bob receives the ciphertext c, and decrypts it, using his *secret key*:

$$m \leftarrow D(sk, c).$$

Public-key encryption is sometimes called **asymmetric encryption** to denote the fact that the encryptor uses one key, pk, and the decryptor uses a different key, sk. This is in contrast with *symmetric encryption*, discussed in Part 1, where both the encryptor and decryptor use the same key.

A few points deserve further discussion:

- Once Alice obtains Bob's public key, the only interaction between Alice and Bob is the actual transmission of the ciphertext from Alice to Bob: no further interaction is required. In fact, we chose encrypted email as our example problem precisely to highlight this feature, as email delivery protocols do not allow any interaction beyond delivery of the message.

- As we will discuss later, the same public key may be used many times. Thus, once Alice obtains Bob's public key, she may send him encrypted messages as often as she likes. Moreover, other users besides Alice may send Bob encrypted messages using the same public key pk.

- As already mentioned, Bob may publicize his public key pk. Obviously, for any secure public-key encryption scheme, it must be hard to compute sk from pk, since anyone can decrypt using sk.

11.1 Two further example applications

Public-key encryption is used in many real-world settings. We give two more examples.

11.1.1 Sharing encrypted files

In a modern encrypted file system, a user Alice can grant read access to other users. In particular, for every file that Alice owns, she can select a group of users who are allowed to decrypt the stored file and read its contents in the clear. This is done using a combination of public-key encryption and an ordinary symmetric cipher.

Here is how it works. Alice encrypts a file f with a random key k using an ordinary symmetric cipher. The resulting ciphertext c_f is stored on the file system. If Alice wants to grant Bob access to the contents of f, she encrypts k under Bob's public key pk_{B} by computing $c_{\mathrm{B}} \overset{\mathrm{R}}{\leftarrow} E(pk_{\mathrm{B}}, k)$. The short ciphertext c_{B} is then stored near the ciphertext c_f, say, as part of the metadata associated with the file f. Now when Bob wants to read f, he decrypts c_{B} using his secret key sk_{B} to obtain k, and then decrypts c_f using k to obtain f in the clear. Alice grants access to herself just as she does for Bob, by encrypting k under her own public key pk_{A}, and storing the resulting ciphertext c_{A} as part of the file metadata.

This scheme scales nicely when Alice wants to grant access to more users. Only one copy of the encrypted file f is stored on the file system. In addition, for each user that is granted access to f, Alice writes an appropriate encryption of the key k in the file's metadata section. Each of these encryptions of k is fairly short, even if the file itself is large.

11.1.2 Key escrow

Consider a company that deploys an encrypted file system such as the one described above. One day Alice is traveling, but her manager needs to read one of her files to prepare for a meeting with an important client. Unfortunately, the manager is unable to decrypt the file because it is encrypted and Alice is unreachable.

Large companies solve this problem using a mechanism called **key escrow**. The company runs a key escrow server that works as follows: at setup time the key escrow server generates a secret key sk_{ES} and a corresponding public key pk_{ES}. It keeps the secret key to itself and makes the public key available to all employees.

When Alice encrypts a file f using a random symmetric key k, she also encrypts k under pk_{ES}, and stores the resulting ciphertext c_{ES} in the file metadata section. Every file created by an employee is encrypted this way. Now, if Alice's manager needs access to f, and Alice is unreachable, the manager sends c_{ES} to the escrow service. The server decrypts c_{ES} to obtain k, and sends k back to the manager. The manager can then use this k to decrypt c_f and obtain f.

Public-key encryption makes it possible for the escrow server to remain offline, until someone needs to decrypt an inaccessible file. Although the escrow service enables Alice's manager to access her files, the escrow service itself cannot read Alice's files, since it does not get to see any encrypted file.

We will discuss this application in more detail in Section 12.2.3.

11.2 Basic definitions

We begin by defining the basic syntax and correctness properties of a public-key encryption scheme.

Definition 11.1. *A* ***public-key encryption scheme*** *$\mathcal{E} = (G, E, D)$ is a triple of efficient algorithms: a* ***key generation algorithm*** *G, an* ***encryption algorithm*** *E, a* ***decryption algorithm*** *D.*

- *G is a probabilistic algorithm that is invoked as $(pk, sk) \overset{\text{R}}{\leftarrow} G()$, where pk is called a* ***public key*** *and sk is called a* ***secret key****.*
- *E is a probabilistic algorithm that is invoked as $c \overset{\text{R}}{\leftarrow} E(pk, m)$, where pk is a public key (as output by G), m is a message, and c is a ciphertext.*
- *D is a deterministic algorithm that is invoked as $m \leftarrow D(sk, c)$, where sk is a secret key (as output by G), c is a ciphertext, and m is either a message, or a special* reject *value (distinct from all messages).*
- *As usual, we require that decryption undoes encryption; specifically, for all possible outputs (pk, sk) of G, and all messages m, we have*

$$\Pr[D(sk,\ E(pk,\ m)\ ) = m] = 1.$$

- *Messages are assumed to lie in some finite* ***message space*** *$\mathcal{M}$, and ciphertexts in some finite* ***ciphertext space*** *$\mathcal{C}$. We say that $\mathcal{E} = (G, E, D)$ is defined over $(\mathcal{M}, \mathcal{C})$.*

We next define the notion of semantic security for a public-key encryption scheme. We stress that this notion of security only models an eavesdropping adversary. We will discuss stronger security properties in the next chapter.

Attack Game 11.1 (semantic security). For a given public-key encryption scheme $\mathcal{E} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $(b = 0, 1)$:

- The challenger computes $(pk, sk) \overset{\text{R}}{\leftarrow} G()$, and sends pk to the adversary.
- The adversary computes $m_0, m_1 \in \mathcal{M}$, of the same length, and sends them to the challenger.
- The challenger computes $c \overset{\text{R}}{\leftarrow} E(pk, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

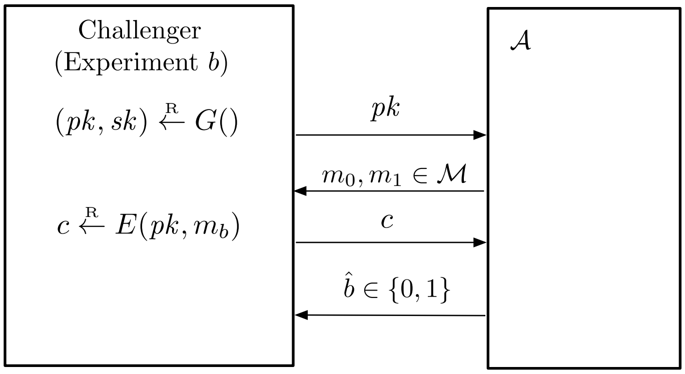

Challenger
(Experiment b)
$(pk, sk) \xleftarrow{R} G()$
$c \xleftarrow{R} E(pk, m_b)$
pk
$m_0, m_1 \in \mathcal{M}$
c
$\hat{b} \in \{0,1\}$
$\mathcal{A}$

Figure 11.1: Experiment b of Attack Game 11.1

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Note that in the above game, the events W_0 and W_1 are defined with respect to the probability space determined by the random choices made by the key generation and encryption algorithms, and the random choices made by the adversary. See Fig. 11.1 for a schematic diagram of Attack Game 11.1.

Definition 11.2 (semantic security). *A public-key encryption scheme $\mathcal{E}$ is* ***semantically secure*** *if for all efficient adversaries $\mathcal{A}$, the value* $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

As discussed in Section 2.2.5, Attack Game 11.1 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0,1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.2.5 (namely, (2.11)) applies here as well:

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]. \tag{11.1}$$

11.2.1 Mathematical details

We give a more mathematically precise definition of a public-key encryption scheme, using the terminology defined in Section 2.3.

Definition 11.3 (public-key encryption scheme). *A* ***public-key encryption scheme*** *consists of three algorithms, G, E, and D, along with two families of spaces with system parameterization P:*

$$\mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda} \quad \textit{and} \quad \mathbf{C} = \{\mathcal{C}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

such that

1. $\mathbf{M}$ *and* $\mathbf{C}$ *are efficiently recognizable.*

2. $\mathbf{M}$ *has an effective length function.*

3. *Algorithm G is an efficient probabilistic algorithm that on input λ, Λ, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \operatorname{Supp}(P(\lambda))$, outputs a pair (pk, sk), where pk and sk are bit strings whose lengths are always bounded by a polynomial in λ.*

4. *Algorithm E is an efficient probabilistic algorithm that on input λ, Λ, pk, m, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \operatorname{Supp}(P(\lambda))$, $(pk, sk) \in \operatorname{Supp}(G(\lambda, \Lambda))$ for some sk, and $m \in \mathcal{M}_{\lambda,\Lambda}$, always outputs an element of $\mathcal{C}_{\lambda,\Lambda}$.*

5. *Algorithm D is an efficient* deterministic *algorithm that on input λ, Λ, sk, c, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \operatorname{Supp}(P(\lambda))$, $(pk, sk) \in \operatorname{Supp}(G(\lambda, \Lambda))$ for some pk, and $c \in \mathcal{C}_{\lambda,\Lambda}$, outputs either an element of $\mathcal{M}_{\lambda,\Lambda}$, or a special symbol* reject $\notin \mathcal{M}_{\lambda,\Lambda}$.

6. *For all $\lambda, \Lambda, pk, sk, m, c$, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \operatorname{Supp}(P(\lambda))$, $(pk, sk) \in \operatorname{Supp}(G(\lambda, \Lambda))$, $k \in \mathcal{K}_{\lambda,\Lambda}$, $m \in \mathcal{M}_{\lambda,\Lambda}$, and $c \in \operatorname{Supp}(E(\lambda, \Lambda; pk, m))$, we have $D(\lambda, \Lambda; sk, c) = m$.*

As usual, the proper interpretation of Attack Game 11.1 is that both challenger and adversary receive λ as a common input, and that the challenger generates Λ and sends this to the adversary before the game begins. The advantage is actually a function of λ, and security means that this is a negligible function of λ.

11.3 Implications of semantic security

Before constructing semantically secure public-key encryption schemes, we first explore a few consequences of semantic security. We first show that any semantically secure public-key scheme must use a *randomized* encryption algorithm. We also show that in the public-key setting, semantic security implies CPA security. This was not true for symmetric encryption schemes: the one-time pad is semantically secure, but not CPA secure.

11.3.1 The need for randomized encryption

Let $\mathcal{E} = (G, E, D)$ be a semantically secure public-key encryption scheme defined over $(\mathcal{M}, \mathcal{C})$ where $|\mathcal{M}| \geq 2$. We show that the encryption algorithm E must be randomized, otherwise the scheme cannot be semantically secure.

To see why, suppose E is deterministic. Then the following adversary $\mathcal{A}$ breaks semantic security of $\mathcal{E} = (G, E, D)$:

- $\mathcal{A}$ receives a public key pk from its challenger.

- $\mathcal{A}$ chooses two distinct messages m_0 and m_1 in $\mathcal{M}$ and sends them to its challenger. The challenger responds with $c := E(pk, m_b)$ for some $b \in \{0, 1\}$.

- $\mathcal{A}$ computes $c_0 := E(pk, m_0)$ and outputs 0 if $c = c_0$. Otherwise, it outputs 1.

Because E is deterministic, we know that $c = c_0$ whenever $b = 0$. Therefore, when $b = 0$ the adversary always outputs 0. Similarly, when $b = 1$ it always outputs 1. Therefore

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] = 1$$

showing that $\mathcal{E}$ is insecure.

This generic attack explains why semantically secure public-key encryption schemes must be randomized. All the schemes we construct in this chapter and the next use randomized encryption. This is quite different from the symmetric key settings where a deterministic encryption scheme can be semantically secure; for example, the one-time pad.

11.3.2 Semantic security against chosen plaintext attack

Recall that when discussing symmetric ciphers, we introduced two distinct notions of security: semantic security, and semantic security against chosen plaintext attack (or CPA security, for short). We showed that for symmetric ciphers, semantic security *does not* imply CPA security. However, for public-key encryption schemes, semantic security *does* imply CPA security. Intuitively, this is because in the public-key setting, the adversary can encrypt any message he likes, without knowledge of any secret key material. The adversary does so using the given public key and never needs to issue encryption queries to the challenger. In contrast, in the symmetric key setting, the adversary cannot encrypt messages on his own.

The attack game defining CPA security in the public-key setting is the natural analog of the corresponding game in the symmetric setting (see Attack Game 5.2 in Section 5.3):

Attack Game 11.2 (CPA security). For a given public-key encryption scheme $\mathcal{E} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b ($b = 0, 1$):

- The challenger computes $(pk, sk) \xleftarrow{\text{R}} G()$, and sends pk to the adversary.
- The adversary submits a sequence of queries to the challenger.

  For $i = 1, 2, \ldots$, the ith query is a pair of messages, $m_{i0}, m_{i1} \in \mathcal{M}$, of the same length.

  The challenger computes $c_i \xleftarrow{\text{R}} E(pk, m_{ib})$, and sends c_i to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, then we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 11.4 (CPA security). *A public-key encryption scheme $\mathcal{E}$ is called **semantically secure against chosen plaintext attack**, or simply **CPA secure**, if for all efficient adversaries $\mathcal{A}$, the value* $\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Theorem 11.1. *If a public-key encryption scheme $\mathcal{E}$ is semantically secure, then it is also CPA secure.*

> *In particular, for every CPA adversary $\mathcal{A}$ that plays Attack Game 11.2 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger, there exists an SS adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = Q \cdot \text{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}].$$

Proof. The proof is a straightforward hybrid argument, and is very similar to the proof of Theorem 5.1. Suppose $\mathcal{E} = (G, E, D)$ is defined over $(\mathcal{M}, \mathcal{C})$. Let $\mathcal{A}$ be a CPA adversary that plays Attack Game 11.2 with respect to $\mathcal{E}$, and which makes at most Q queries to its challenger.

We describe the relevant hybrid games. For $j = 0, \ldots, Q$, Hybrid j is played between $\mathcal{A}$ and a challenger who works as follows:

```
$(pk, sk) \overset{R}{\leftarrow} G()$
Send $pk$ to $\mathcal{A}$
Upon receiving the $i$th query $(m_{i0}, m_{i1}) \in \mathcal{M}^2$ from $\mathcal{A}$ do:
    if $i > j$
        then $c_i \overset{R}{\leftarrow} E(pk, m_{i0})$
        else $c_i \overset{R}{\leftarrow} E(pk, m_{i1})$
    send $c_i$ to $\mathcal{A}$.
```

Put another way, the challenger in Hybrid j encrypts

$$m_{11}, \ldots, m_{j1}, \quad m_{(j+1)0}, \ldots, m_{Q0},$$

As usual, we define p_j to be the probability that $\mathcal{A}$ outputs 1 in Hybrid j. Clearly,

$$\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = |p_Q - p_0|.$$

Next, we define an appropriate adversary $\mathcal{B}$ that plays Attack Game 11.1 with respect to $\mathcal{E}$:

First, $\mathcal{B}$ chooses $\omega \in \{1, \ldots, Q\}$ at random.

Then, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$: it obtains a public key pk from its own challenger, and forwards this to $\mathcal{A}$; when $\mathcal{A}$ makes a query (m_{i0}, m_{i1}), $\mathcal{B}$ computes its response c_i as follows:

```
if $i > \omega$ then
    $c \overset{R}{\leftarrow} E(pk, m_{i0})$
else if $i = \omega$ then
    $\mathcal{B}$ submits $(m_{i0}, m_{i1})$ to its own challenger
    $c_i$ is set to the challenger's response
else  //  $i < \omega$
    $c_i \overset{R}{\leftarrow} E(pk, m_{i1})$.
```

Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

The crucial difference between the proof of this theorem and that of Theorem 5.1 is that for $i \neq \omega$, adversary $\mathcal{B}$ can encrypt the relevant message using the public key.

For $b = 0, 1$, let W_b be the event that $\mathcal{B}$ outputs 1 in Experiment b of its attack game. It is clear that for $j = 1, \ldots, Q$,

$$\Pr[W_0 \mid \omega = j] = p_{j-1} \quad \text{and} \quad \Pr[W_1 \mid \omega = j] = p_j,$$

and the theorem follows by the usual telescoping sum calculation. □

One can also consider multi-key CPA security, where the adversary sees many encryptions under many public keys. In the public-key setting, semantic security implies not only CPA security, but multi-key CPA security — see Exercise 11.10.

11.4 Encryption based on a trapdoor function scheme

In this section, we show how to use a trapdoor function scheme (see Section 10.2) to build a semantically secure public-key encryption scheme. In fact, this scheme makes use of a hash function, and our proof of security works only when we model the hash function as a random oracle (see Section 8.10.2). We then present a concrete instantiation of this scheme, based on RSA (see Section 10.3).

Our encryption scheme is called $\mathcal{E}_{\text{TDF}}$, and is built out of several components:

- a trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$,
- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a hash function $H : \mathcal{X} \to \mathcal{K}$.

The message space for $\mathcal{E}_{\text{TDF}}$ is $\mathcal{M}$, and the ciphertext space is $\mathcal{Y} \times \mathcal{C}$. We now describe the key generation, encryption, and decryption algorithms for $\mathcal{E}_{\text{TDF}}$.

- The key generation algorithm for $\mathcal{E}_{\text{TDF}}$ is the key generation algorithm for $\mathcal{T}$.
- For a given public key pk, and a given message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & x \xleftarrow{\text{R}} \mathcal{X}, \quad y \leftarrow F(pk, x), \quad k \leftarrow H(x), \quad c \xleftarrow{\text{R}} E_{\text{s}}(k, m) \\ & \text{output } (y, c). \end{aligned}$$

- For a given secret key sk, and a given ciphertext $(y, c) \in \mathcal{Y} \times \mathcal{C}$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk,\ (y, c)\ ) := \quad & x \leftarrow I(sk, y), \quad k \leftarrow H(x), \quad m \leftarrow D_{\text{s}}(k, c) \\ & \text{output } m. \end{aligned}$$

Thus, $\mathcal{E}_{\text{TDF}} = (G, E, D)$, and is defined over $(\mathcal{M}, \mathcal{Y} \times \mathcal{C})$.

The correctness property for $\mathcal{T}$ immediately implies the correctness property for $\mathcal{E}_{\text{TDF}}$. If H is modeled as a random oracle (see Section 8.10), one can prove that $\mathcal{E}_{\text{TDF}}$ is semantically secure, assuming that $\mathcal{T}$ is one-way, and that $\mathcal{E}_{\text{s}}$ is semantically secure.

Recall that in the random oracle model, the function H is modeled as a random function $\mathcal{O}$ chosen at random from the set of all functions $\text{Funs}[\mathcal{X}, \mathcal{K}]$. More precisely, in the random oracle version of Attack Game 11.1, the challenger chooses $\mathcal{O}$ at random. In any computation where the challenger would normally evaluate H, it evaluates $\mathcal{O}$ instead. In addition, the adversary is allowed to ask the challenger for the value of the function $\mathcal{O}$ at any point of its choosing. The adversary may make any number of such "random oracle queries" at any time of its choosing. We use $\text{SS}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{TDF}}]$ to denote $\mathcal{A}$'s advantage against $\mathcal{E}_{\text{TDF}}$ in the random oracle version of Attack Game 11.1.

Theorem 11.2. *Assume $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle. If $\mathcal{T}$ is one-way and $\mathcal{E}_{\text{s}}$ is semantically secure, then $\mathcal{E}_{\text{TDF}}$ is semantically secure.*

> *In particular, for every SS adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\text{TDF}}$ as in the random oracle version of Attack Game 11.1, there exist an inverting adversary $\mathcal{B}_{\text{ow}}$ that attacks $\mathcal{T}$ as in Attack Game 10.2,*

and an SS adversary $\mathcal{B}_{\rm s}$ that attacks $\mathcal{E}_{\rm s}$ as in Attack Game 2.1, where $\mathcal{B}_{\rm ow}$ and $\mathcal{B}_{\rm s}$ are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{SS^{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\rm TDF}] \leq 2 \cdot \mathrm{OW}\mathsf{adv}[\mathcal{B}_{\rm ow}, \mathcal{T}] + \mathrm{SS}\mathsf{adv}[\mathcal{B}_{\rm s}, \mathcal{E}_{\rm s}]. \tag{11.2}$$

Proof idea. Suppose the adversary sees the ciphertext (y, c), where $y = F(pk, x)$. If H is modeled as a random oracle, then intuitively, the only way the adversary can learn anything at all about the symmetric key k used to generate c is to explicitly evaluate the random oracle representing H at the point x; however, if he could do this, we could easily convert the adversary into an adversary that inverts the function $F(pk, \cdot)$, contradicting the one-wayness assumption. Therefore, from the adversary's point of view, k is completely random, and semantic security for $\mathcal{E}_{\rm TDF}$ follows directly from the semantic security of $\mathcal{E}_{\rm s}$. In the detailed proof, we implement the random oracle using the same "faithful gnome" technique as was used to efficiently implement random functions (see Section 4.4.2); that is, we represent the random oracle as a table of input/output pairs corresponding to points at which the adversary actually queried the random oracle (as well as the point at which the challenger queries the random oracle when it runs the encryption algorithm). We also use many of the same proof techniques introduced in Chapter 4, specifically, the "forgetful gnome" technique (introduced in the proof of Theorem 4.6) and the Difference Lemma (Theorem 4.7). □

Proof. It is convenient to prove the theorem using the bit-guessing versions of the semantic security game. We prove:

$$\mathrm{SS^{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\rm TDF}] \leq \mathrm{OW}\mathsf{adv}[\mathcal{B}_{\rm ow}, \mathcal{T}] + \mathrm{SS}\mathsf{adv}^*[\mathcal{B}_{\rm s}, \mathcal{E}_{\rm s}]. \tag{11.3}$$

Then (11.2) follows by (11.1) and (2.10).

Define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the bit-guessing version of Attack Game 11.1 with respect to $\mathcal{E}_{\rm TDF}$. We then modify the challenger to obtain Game 1. In each game, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, 1$, we define W_j to be the event that $\hat{b} = b$ in Game j. We will show that $|\Pr[W_1] - \Pr[W_0]|$ is negligible, and that $\Pr[W_1]$ is negligibly close to 1/2. From this, it follows that

$$\mathrm{SS^{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\rm TDF}] = \bigl|\Pr[W_0] - 1/2\bigr| \tag{11.4}$$

is also negligible.

Game 0. Note that the challenger in Game 0 also has to respond to the adversary's random oracle queries. The adversary can make any number of random oracle queries, but at most one encryption query. Recall that in addition to direct access to the random oracle via explicit random oracle queries, the adversary also has indirect access to the random oracle via the encryption query, where the challenger also makes use of the random oracle. In describing this game, we directly implement the random oracle as a "faithful gnome." This is done using an associative array $Map : \mathcal{X} \to \mathcal{K}$. The details are in Fig. 11.2. In the initialization step, the challenger prepares some quantities that will be used later in processing the encryption query. In particular, in addition to computing $(pk, sk) \xleftarrow{\rm R} G()$, the challenger precomputes $x \xleftarrow{\rm R} \mathcal{X}$, $y \leftarrow F(pk, x)$, $k \xleftarrow{\rm R} \mathcal{K}$. It also sets $Map[x] \leftarrow k$, which means that the value of the random oracle at x is equal to k.

Game 1. This game is precisely the same as Game 0, except that we make our gnome "forgetful" by deleting line (3) in Fig. 11.2.

Let Z be the event that the adversary queries the random oracle at the point x in Game 1. Clearly, Games 0 and 1 proceed identically unless Z occurs, and so by the Difference Lemma, we

```
        initialization:
(1)         $(pk, sk) \overset{R}{\leftarrow} G()$, $x \overset{R}{\leftarrow} \mathcal{X}$, $y \leftarrow F(pk, x)$
            initialize an empty associative array $Map : \mathcal{X} \to \mathcal{K}$
(2)         $k \overset{R}{\leftarrow} \mathcal{K}$, $b \overset{R}{\leftarrow} \{0, 1\}$
(3)         $Map[x] \leftarrow k$
            send the public key $pk$ to $\mathcal{A}$;

        upon receiving a single encryption query $(m_0, m_1) \in \mathcal{M}^2$:
(4)         $c \overset{R}{\leftarrow} E_s(k, m_b)$
            send $(y, c)$ to $\mathcal{A}$;

        upon receiving a random oracle query $\hat{x} \in \mathcal{X}$:
            if $\hat{x} \notin \text{Domain}(Map)$ then $Map[\hat{x}] \overset{R}{\leftarrow} \mathcal{K}$
            send $Map[\hat{x}]$ to $\mathcal{A}$
```

Figure 11.2: Game 0 challenger

have

$$\big|\Pr[W_1] - \Pr[W_0]\big| \leq \Pr[Z]. \tag{11.5}$$

If event Z happens, then one of the adversary's random oracle queries is the inverse of y under $F(pk, \cdot)$. Moreover, in Game 1, the value x is used only to define $y = F(pk, x)$, and nowhere else. Thus, we can use adversary $\mathcal{A}$ to build an efficient adversary $\mathcal{B}_{\text{ow}}$ that breaks the one-wayness assumption for $\mathcal{T}$ with an advantage equal to $\Pr[Z]$.

Here is how adversary $\mathcal{B}_{\text{ow}}$ works in detail. This adversary plays Attack Game 10.2 against a challenger $\mathbf{C}_{\text{ow}}$, and plays the role of challenger to $\mathcal{A}$ as in Fig. 11.2, except with the following lines modified as indicated:

```
(1)     obtain $(pk, y)$ from $\mathbf{C}_{\text{ow}}$
(3)     *(deleted)*
```

Additionally,

```
    when $\mathcal{A}$ terminates:
        if $F(pk, \hat{x}) = y$ for some $\hat{x} \in \text{Domain}(Map)$
            then output $\hat{x}$
            else  output "failure".
```

To analyze $\mathcal{B}_{\text{ow}}$, we may naturally view Game 1 and the game played between $\mathcal{B}_{\text{ow}}$ and $\mathbf{C}_{\text{ow}}$ as operating on the same underlying probability space. By definition, Z occurs if and only if $x \in \text{Domain}(Map)$ when $\mathcal{B}_{\text{ow}}$ finishes its game. Therefore,

$$\Pr[Z] = \text{OW}\mathsf{adv}[\mathcal{B}_{\text{ow}}, \mathcal{T}]. \tag{11.6}$$

Observe that in Game 1, the key k is only used to encrypt the challenge plaintext. As such, the adversary is essentially attacking $\mathcal{E}_s$ as in the bit-guessing version of Attack Game 2.1 at this point. More precisely, we derive an efficient SS adversary $\mathcal{B}_s$ based on Game 1 that uses $\mathcal{A}$ as a subroutine, such that

$$\bigl|\Pr[W_1] - 1/2\bigr| = \mathsf{SSadv}^*[\mathcal{B}_s, \mathcal{E}_s]. \tag{11.7}$$

Adversary $\mathcal{B}_s$ plays the bit-guessing version of Attack Game 2.1 against a challenger $\mathbf{C}_s$, and plays the role of challenger to $\mathcal{A}$ as in Fig. 11.2, except with the following lines modified as indicated:

```
(2)     (deleted)
(3)     (deleted)
(4)     forward $(m_0, m_1)$ to $\mathbf{C}_s$, obtaining $c$
```

Additionally,

```
when $\mathcal{A}$ outputs $\hat{b}$:
    output $\hat{b}$
```

To analyze $\mathcal{B}_s$, we may naturally view Game 1 and the game played between $\mathcal{B}_s$ and $\mathbf{C}_s$ as operating on the same underlying probability space. By construction, $\mathcal{B}_s$ and $\mathcal{A}$ output the same thing, and so (11.7) holds.

Combining (11.4), (11.5), (11.6), and (11.7), yields (11.3). □

11.4.1 Instantiating $\mathcal{E}_{\text{TDF}}$ with RSA

Suppose we now use RSA (see Section 10.3) to instantiate $\mathcal{T}$ in the above encryption scheme $\mathcal{E}_{\text{TDF}}$. This scheme is parameterized by two quantities: the length ℓ of the prime factors of the RSA modulus, and the encryption exponent e, which is an odd, positive integer. Recall that the RSA scheme does not quite fit the definition of a trapdoor permutation scheme, because the domain of the trapdoor permutation is not a fixed set, but varies with the public key. Let us assume that $\mathcal{X}$ is a fixed set into which we may embed $\mathbb{Z}_n$, for every RSA modulus n generated by $\text{RSAGen}(\ell, e)$ (for example, we could take $\mathcal{X} = \{0,1\}^{2\ell}$). The scheme also makes use of a symmetric cipher $\mathcal{E}_s = (E_s, D_s)$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, as well as a hash function $H : \mathcal{X} \to \mathcal{K}$.

The basic RSA encryption scheme is $\mathcal{E}_{\text{RSA}} = (G, E, D)$, with message space $\mathcal{M}$ and ciphertext space $\mathcal{X} \times \mathcal{C}$, where

- the key generation algorithm runs as follows:

$$\begin{aligned} G() := \quad & (n,d) \xleftarrow{R} \text{RSAGen}(\ell, e), \quad pk \leftarrow (n,e), \quad sk \leftarrow (n,d) \\ & \text{output } (pk, sk); \end{aligned}$$

- for a given public key $pk = (n, e)$, and message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & x \xleftarrow{R} \mathbb{Z}_n, \quad y \leftarrow x^e, \quad k \leftarrow H(x), \quad c \xleftarrow{R} E_s(k, m) \\ & \text{output } (y, c) \in \mathcal{X} \times \mathcal{C}; \end{aligned}$$

- for a given secret key $sk = (n, d)$, and a given ciphertext $(y, c) \in \mathcal{X} \times \mathcal{C}$, where y represents an element of $\mathbb{Z}_n$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk,\ (y,c)\ ) := \quad & x \leftarrow y^d, \quad k \leftarrow H(x), \quad m \leftarrow D_s(k, c) \\ & \text{output } m. \end{aligned}$$

Theorem 11.3. *Assume $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle. If the RSA assumption holds for parameters (ℓ, e), and $\mathcal{E}_{\mathrm{s}}$ is semantically secure, then $\mathcal{E}_{\mathrm{RSA}}$ is semantically secure.*

> *In particular, for any SS adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\mathrm{RSA}}$ as in the random oracle version of Attack Game 11.1, there exist an RSA adversary $\mathcal{B}_{\mathrm{rsa}}$ that breaks the RSA assumption for (ℓ, e) as in Attack Game 10.3, and an SS adversary $\mathcal{B}_{\mathrm{s}}$ that attacks $\mathcal{E}_{\mathrm{s}}$ as in Attack Game 2.1, where $\mathcal{B}_{\mathrm{rsa}}$ and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{SS}^{\mathrm{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\mathrm{RSA}}] \leq \mathrm{RSA}\mathsf{adv}[\mathcal{B}_{\mathrm{rsa}}, \ell, e] + \mathrm{SS}\mathsf{adv}^*[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}].$$

Proof. The proof of Theorem 11.2 carries over, essentially unchanged. □

11.5 ElGamal encryption

In this section we show how to build a public-key encryption scheme from Diffie-Hellman. Security will be based on either the CDH or DDH assumptions from Section 10.5.

The encryption scheme is a variant of a scheme first proposed by ElGamal, and we call it $\mathcal{E}_{\mathrm{EG}}$. It is built out of several components:

- a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$,
- a symmetric cipher $\mathcal{E}_{\mathrm{s}} = (E_{\mathrm{s}}, D_{\mathrm{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a hash function $H : \mathbb{G}^2 \to \mathcal{K}$.

The message space for $\mathcal{E}_{\mathrm{EG}}$ is $\mathcal{M}$, and the ciphertext space is $\mathbb{G} \times \mathcal{C}$. We now describe the key generation, encryption, and decryption algorithms for $\mathcal{E}_{\mathrm{EG}}$.

- the key generation algorithm runs as follows:

$$\begin{aligned} G() := \quad & \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha} \\ & pk \leftarrow u, \quad sk \leftarrow \alpha \\ & \text{output } (pk, sk); \end{aligned}$$

- for a given public key $pk = u \in \mathbb{G}$ and message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

$$\begin{aligned} E(pk, m) := \quad & \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad v \leftarrow g^{\beta}, \quad w \leftarrow u^{\beta}, \quad k \leftarrow H(v, w), \quad c \leftarrow E_{\mathrm{s}}(k, m) \\ & \text{output } (v, c); \end{aligned}$$

- for a given secret key $sk = \alpha \in \mathbb{Z}_q$ and a ciphertext $(v, c) \in \mathbb{G} \times \mathcal{C}$, the decryption algorithm runs as follows:

$$\begin{aligned} D(sk,\ (v, c)\ ) := \quad & w \leftarrow v^{\alpha}, \quad k \leftarrow H(v, w), \quad m \leftarrow D_{\mathrm{s}}(k, c) \\ & \text{output } m. \end{aligned}$$

Thus, $\mathcal{E}_{\mathrm{EG}} = (G, E, D)$, and is defined over $(\mathcal{M}, \mathbb{G} \times \mathcal{C})$.

Note that the description of the group $\mathbb{G}$ and generator $g \in \mathbb{G}$ is considered to be a system parameter, rather than part of the public key.

Remark 11.1 (Hashing (v, w) vs hashing only w). In $\mathcal{E}_{\text{EG}}$, we derive the key k by hashing both v and w. In a variant, which we call $\mathcal{E}'_{\text{EG}}$, we could derive the key k by hashing just w. The security results that we prove in this chapter hold for $\mathcal{E}'_{\text{EG}}$ as well as for $\mathcal{E}_{\text{EG}}$ (and the proofs carry over with little change). However, in the next chapter we prove further security properties of $\mathcal{E}_{\text{EG}}$ that do not hold as cleanly for $\mathcal{E}'_{\text{EG}}$. As such, the scheme $\mathcal{E}_{\text{EG}}$ is generally preferred, and is not significantly more computationally expensive than $\mathcal{E}'_{\text{EG}}$. □

11.5.1 Semantic security of ElGamal in the random oracle model

We shall analyze the security of $\mathcal{E}_{\text{EG}}$ under two different sets of assumptions. In this section we do the analysis modeling $H : \mathbb{G}^2 \to \mathcal{K}$ as a random oracle, under the CDH assumption for $\mathbb{G}$, and the assumption that $\mathcal{E}_{\text{s}}$ is semantically secure. In the next section we analyze $\mathcal{E}_{\text{EG}}$ without the random oracle model, but using the stronger DDH assumption for $\mathbb{G}$.

Theorem 11.4. *Assume $H : \mathbb{G}^2 \to \mathcal{K}$ is modeled as a random oracle. If the CDH assumption holds for $\mathbb{G}$, and $\mathcal{E}_{\text{s}}$ is semantically secure, then $\mathcal{E}_{\text{EG}}$ is semantically secure.*

> *In particular, for every SS adversary $\mathcal{A}$ that plays the random oracle version of Attack Game 11.1 with respect to $\mathcal{E}_{\text{EG}}$, and makes at most Q queries to the random oracle, there exist a CDH adversary $\mathcal{B}_{\text{cdh}}$ that plays Attack Game 10.5 with respect to $\mathbb{G}$, and an SS adversary $\mathcal{B}_{\text{s}}$ that plays Attack Game 2.1 with respect to $\mathcal{E}_{\text{s}}$, where $\mathcal{B}_{\text{cdh}}$ and $\mathcal{B}_{\text{s}}$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{SS}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{EG}}] \leq 2Q \cdot \text{CDH}\mathsf{adv}[\mathcal{B}_{\text{cdh}}, \mathbb{G}] + \text{SS}\mathsf{adv}[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}]. \tag{11.8}$$

Proof idea. Suppose the adversary sees the ciphertext (v, c), where $v = g^{\beta}$. If H is modeled as a random oracle, then intuitively, the only way the adversary can learn anything at all about the symmetric key k used to generate c is to explicitly evaluate the random oracle representing H at the point (v, w), where $w = v^{\alpha}$; however, if he could do this, we could convert the adversary into an adversary that breaks the CDH assumption for $\mathbb{G}$. One wrinkle is that we cannot recognize the correct solution to the CDH problem when we see it (if the DDH assumption is true), so we simply guess by choosing at random from among all of the adversary's random oracle queries. This is where the factor of Q in (11.8) comes from. So unless the adversary can break the CDH assumption, from the adversary's point of view, k is completely random, and semantic security for $\mathcal{E}_{\text{EG}}$ follows directly from the semantic security of $\mathcal{E}_{\text{s}}$. □

Proof. It is convenient to prove the theorem using the bit-guessing version of the semantic security game. We prove:

$$\text{SS}^{\text{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\text{EG}}] \leq Q \cdot \text{CDH}\mathsf{adv}[\mathcal{B}_{\text{cdh}}, \mathbb{G}] + \text{SS}\mathsf{adv}^*[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}]. \tag{11.9}$$

Then (11.8) follows from (11.1) and (2.10).

We define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the bit-guessing version of Attack Game 11.1 with respect to $\mathcal{E}_{\text{EG}}$. We then modify the challenger to obtain Game 1. In each game, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, 1$, we define W_j to be the event that $\hat{b} = b$ in Game j. We will show that $\left|\Pr[W_1] - \Pr[W_0]\right|$ is negligible, and that $\Pr[W_1]$ is negligibly close to 1/2. From this, it follows that

$$\text{SS}^{\text{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\text{EG}}] = \left|\Pr[W_0] - 1/2\right| \tag{11.10}$$

is negligible.

```
        initialization:
(1)         $\alpha, \beta \stackrel{\textrm{R}}{\leftarrow} \mathbb{Z}_q$, $u \leftarrow g^{\alpha}$, $v \leftarrow g^{\beta}$, $w \leftarrow g^{\alpha\beta}$
            initialize an empty associative array $Map : \mathbb{G}^2 \to \mathcal{K}$
(2)         $k \stackrel{\textrm{R}}{\leftarrow} \mathcal{K}$, $b \stackrel{\textrm{R}}{\leftarrow} \{0, 1\}$
(3)         $Map[v, w] \leftarrow k$
            send the public key $u$ to $\mathcal{A}$;

        upon receiving a single encryption query $(m_0, m_1) \in \mathcal{M}^2$:
(4)         $c \stackrel{\textrm{R}}{\leftarrow} E_{\textrm{s}}(k, m_b)$
            send $(v, c)$ to $\mathcal{A}$;

        upon receiving a random oracle query $(\hat{v}, \hat{w}) \in \mathbb{G}^2$:
            if $(\hat{v}, \hat{w}) \notin \textrm{Domain}(Map)$ then $Map[\hat{v}, \hat{w}] \stackrel{\textrm{R}}{\leftarrow} \mathcal{K}$
            send $Map[\hat{v}, \hat{w}]$ to $\mathcal{A}$
```

Figure 11.3: Game 0 challenger

Game 0. The adversary can make any number of random oracle queries, but at most one encryption query. Again, recall that in addition to direct access to the random oracle via explicit random oracle queries, the adversary also has indirect access to the random oracle via the encryption query, where the challenger also makes use of the random oracle. The random oracle is implemented using an associative array $Map : \mathbb{G}^2 \to \mathcal{K}$. The details are in Fig. 11.3. At line (3), we effectively set the random oracle at the point (v, w) to k.

Game 1. This is the same as Game 0, except we delete line (3) in Fig. 11.3.

Let Z be the event that the adversary queries the random oracle at (v, w) in Game 1. Clearly, Games 0 and 1 proceed identically unless Z occurs, and so by the Difference Lemma, we have

$$\big|\Pr[W_1] - \Pr[W_0]\big| \le \Pr[Z]. \tag{11.11}$$

If event Z happens, then one of the adversary's random oracle queries is (v, w), where w is the solution to the instance (u, v) of the CDH problem. Moreover, in Game 1, the values α and β are only needed to compute u and v, and nowhere else. Thus, we can use adversary $\mathcal{A}$ to build an adversary $\mathcal{B}_{\textrm{cdh}}$ to break the CDH assumption: we simply choose one of the adversary's random oracle queries $(\hat{v}, \hat{w})$ at random, and output $\hat{w}$ — with probability at least $\Pr[Z]/Q$, this will be the solution to the given instance of the CDH problem.

In more detail, adversary $\mathcal{B}_{\textrm{cdh}}$ plays Attack Game 10.5 against a challenger $\mathbf{C}_{\textrm{cdh}}$, and plays the role of challenger to $\mathcal{A}$ as in Fig. 11.3, except with the following lines modified as indicated:

```
(1)     obtain $(u, v)$ from $\mathbf{C}_{\textrm{cdh}}$
(3)     (deleted)
```

Additionally,

```
    when $\mathcal{A}$ terminates:
        if Domain($Map$) $\neq \emptyset$
            then $(\hat{v}, \hat{w}) \stackrel{\textrm{R}}{\leftarrow} \textrm{Domain}(Map)$, output $\hat{w}$
            else  output “failure”
```

To analyze $\mathcal{B}_{\mathrm{cdh}}$, we may naturally view Game 1 and the game played between $\mathcal{B}_{\mathrm{cdh}}$ and $\mathbf{C}_{\mathrm{cdh}}$ as operating on the same underlying probability space. By definition, Z occurs if and only if $(v, w) \in \mathrm{Domain}(\mathit{Map})$ when $\mathcal{B}_{\mathrm{cdh}}$ finishes its game. Moreover, since $\left|\mathrm{Domain}(\mathit{Map})\right| \leq Q$, it follows that

$$\mathsf{CDHadv}[\mathcal{B}_{\mathrm{cdh}}, \mathbb{G}] \geq \Pr[Z]/Q. \tag{11.12}$$

Observe that in Game 1, the key k is only used to encrypt the challenge plaintext. We leave it to the reader to describe an efficient SS adversary $\mathcal{B}_{\mathrm{s}}$ that uses $\mathcal{A}$ as a subroutine, such that

$$\left|\Pr[W_1] - 1/2\right| = \mathsf{SSadv}^*[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{11.13}$$

Combining (11.10), (11.11), (11.12), and (11.13), yields (11.9), which completes the proof of the theorem. □

11.5.2 Semantic security of ElGamal without random oracles

As we commented in Section 8.10.2, security results in the random oracle model do not necessarily imply security in the real world. When it does not hurt efficiency, it is better to avoid the random oracle model. By replacing the CDH assumption by the stronger, but still reasonable, DDH assumption, and by making an appropriate, but reasonable, assumption about H, we can prove that the same system $\mathcal{E}_{\mathrm{EG}}$ is semantically secure without resorting to the random oracle model.

We thus obtain two security analyses of $\mathcal{E}_{\mathrm{EG}}$: one in the random oracle model, but using the CDH assumption. The other, without the random oracle model, but using the stronger DDH assumption. We are thus using the random oracle model as a hedge: in case the DDH assumption turns out to be false in the group $\mathbb{G}$, the scheme remains secure assuming CDH holds in $\mathbb{G}$, but in a weaker random oracle semantic security model. In Exercise 11.14 we develop yet another analysis of ElGamal without random oracles, but using a weaker assumption than DDH called **hash Diffie-Hellman** (HDH) which more accurately captures the exact requirement needed to prove security.

To carry out the analysis using the DDH assumption in $\mathbb{G}$ we make a specific assumption about the hash function $H : \mathbb{G}^2 \to \mathcal{K}$, namely that H is a **secure key derivation function**, or **KDF** for short. We already introduced a very general notion of a key derivation function in Section 8.10. What we describe here is more focused and tailored precisely to our current situation.

Intuitively, $H : \mathbb{G}^2 \to \mathcal{K}$ is a secure KDF if no efficient adversary can effectively distinguish between $(v, H(w))$ and (v, k), where v and w are randomly chosen from $\mathbb{G}$, and k is randomly chosen from $\mathcal{K}$. To be somewhat more general, we consider an arbitrary, efficiently computable hash function $F : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, where $\mathcal{X}$, $\mathcal{Y}$, and $\mathcal{Z}$ are arbitrary, finite sets.

Attack Game 11.3 (secure key derivation). For a given hash function $F : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $(b = 0, 1)$:

- The challenger computes

  $$x \overset{\mathrm{R}}{\leftarrow} \mathcal{X}, \quad y \overset{\mathrm{R}}{\leftarrow} \mathcal{Y}, \quad z_0 \leftarrow F(x, y), \quad z_1 \overset{\mathrm{R}}{\leftarrow} \mathcal{Z},$$

  and sends (x, z_b) to the adversary.

- The adversary outputs a bit $\hat{b} \in \{0,1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, then we define $\mathcal{A}$'s **advantage** with respect to F as

$$\text{KDF}\mathsf{adv}[\mathcal{A}, F] := \big|\Pr[W_0] - \Pr[W_1]\big|. \quad \square$$

Definition 11.5 (secure key derivation). *A hash function $F : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is a* ***secure KDF*** *if for every efficient adversary $\mathcal{A}$, the value* KDF$\mathsf{adv}[\mathcal{A}, F]$ *is negligible.*

It is plausible to conjecture that an "off the shelf" hash function, like SHA256 or HKDF (see Section 8.10.5), is a secure KDF. In fact, one may justify this assumption modeling the hash function as a random oracle; however, using this explicit computational assumption, rather than the random oracle model, yields more meaningful results.

One may even build a secure KDF without making any assumptions at all: the construction in Section 8.10.4 based on a universal hash function and the leftover hash lemma yields an unconditionally secure KDF. Even though this construction is theoretically attractive and quite efficient, it may not be a wise choice from a security point of view: as already discussed above, if the DDH turns out to be false, we can still rely on the CDH in the random oracle model, but for that, it is better to use something based on SHA256 or HKDF, which can more plausibly be modeled as a random oracle.

Theorem 11.5. *If the DDH assumption holds for $\mathbb{G}$, $H : \mathbb{G}^2 \to \mathcal{K}$ is a secure KDF, and $\mathcal{E}_\text{s}$ is semantically secure, then $\mathcal{E}_\text{EG}$ is semantically secure.*

> *In particular, for every SS adversary $\mathcal{A}$ that plays Attack Game 11.1 with respect to $\mathcal{E}_\text{EG}$, there exist a DDH adversary $\mathcal{B}_\text{ddh}$ that plays Attack Game 10.6 with respect to $\mathbb{G}$, a KDF adversary $\mathcal{B}_\text{kdf}$ that plays Attack Game 11.3 with respect to H, and an SS adversary $\mathcal{B}_\text{s}$ that plays Attack Game 2.1 with respect to $\mathcal{E}_\text{s}$, where $\mathcal{B}_\text{ddh}$, $\mathcal{B}_\text{kdf}$, and $\mathcal{B}_\text{s}$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}_\text{EG}] \leq 2 \cdot \text{DDH}\mathsf{adv}[\mathcal{B}_\text{ddh}, \mathbb{G}] + 2 \cdot \text{KDF}\mathsf{adv}[\mathcal{B}_\text{kdf}, H] + \text{SS}\mathsf{adv}[\mathcal{B}_\text{s}, \mathcal{E}_\text{s}]. \tag{11.14}$$

Proof idea. Suppose the adversary sees the ciphertext (v, c), where $v = g^\beta$ and c is a symmetric encryption created using the key $k := H(v, w)$, where $w = u^\beta$. Suppose the challenger replaces w by a random independent group element $\tilde{w} \in \mathbb{G}$ and constructs k as $k := H(v, \tilde{w})$. By the DDH assumption the adversary cannot tell the difference between w and $\tilde{w}$ and hence its advantage is only negligibly changed. Under the KDF assumption, $k := H(v, \tilde{w})$ looks like a random key in $\mathcal{K}$, independent of the adversary's view, and therefore security follows by semantic security of $\mathcal{E}_\text{s}$. □

Proof. More precisely, it is convenient to prove the theorem using the bit-guessing version of the semantic security game. We prove:

$$\text{SS}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_\text{EG}] \leq \text{DDH}\mathsf{adv}[\mathcal{B}_\text{ddh}, \mathbb{G}] + \text{KDF}\mathsf{adv}[\mathcal{B}_\text{kdf}, H] + \text{SS}\mathsf{adv}^*[\mathcal{B}_\text{s}, \mathcal{E}_\text{s}]. \tag{11.15}$$

Then (11.14) follows by (11.1) and (2.10).

Define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the bit-guessing version of Attack Game 11.1 with respect to $\mathcal{E}_\text{EG}$. We then modify the challenger to obtain Games 1 and 2. In each game, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, 1, 2$, we define W_j to be the event that $\hat{b} = b$ in Game j. We will show that $\big|\Pr[W_2] - \Pr[W_0]\big|$ is negligible, and that $\Pr[W_2]$ is negligibly close to 1/2. From this, it follows that

$$\text{SS}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_\text{EG}] = \big|\Pr[W_0] - 1/2\big| \tag{11.16}$$

is negligible.

```
    initialization:
(1)     $\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $\gamma \leftarrow \alpha\beta$, $u \leftarrow g^{\alpha}$, $v \leftarrow g^{\beta}$, $w \leftarrow g^{\gamma}$
(2)     $k \leftarrow H(v, w)$
        $b \xleftarrow{\mathrm{R}} \{0, 1\}$
        send the public key $u$ to $\mathcal{A}$;
    upon receiving $(m_0, m_1) \in \mathcal{M}^2$:
        $c \leftarrow E_{\mathrm{s}}(k, m_b)$, send $(v, c)$ to $\mathcal{A}$
```

Figure 11.4: Game 0 challenger

Game 0. The logic of the challenger in this game is presented in Fig. 11.4.

Game 1. We first play our "DDH card." The challenger in this game is as in Fig. 11.4, except that line (1) is modified as follows:

```
(1)     $\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $\gamma \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $u \leftarrow g^{\alpha}$, $v \leftarrow g^{\beta}$, $w \leftarrow g^{\gamma}$
```

We describe an efficient DDH adversary $\mathcal{B}_{\mathrm{ddh}}$ that uses $\mathcal{A}$ as a subroutine, such that

$$\big|\Pr[W_0] - \Pr[W_1]\big| = \mathrm{DDH}\mathsf{adv}[\mathcal{B}_{\mathrm{ddh}}, \mathbb{G}]. \tag{11.17}$$

Adversary $\mathcal{B}_{\mathrm{ddh}}$ plays Attack Game 10.6 against a challenger $\mathbf{C}_{\mathrm{ddh}}$, and plays the role of challenger to $\mathcal{A}$ as in Fig. 11.4, except with line (1) modified as follows:

```
(1)     obtain $(u, v, w)$ from $\mathbf{C}_{\mathrm{ddh}}$
```

Additionally,

```
    when $\mathcal{A}$ outputs $\hat{b}$:
        if $b = \hat{b}$ then output 1 else output 0
```

Let p_0 be the probability that $\mathcal{B}_{\mathrm{ddh}}$ outputs 1 when $\mathbf{C}_{\mathrm{ddh}}$ is running Experiment 0 of the DDH Attack Game 10.6, and let p_1 be the probability that $\mathcal{B}_{\mathrm{ddh}}$ outputs 1 when $\mathbf{C}_{\mathrm{ddh}}$ is running Experiment 1. By definition, $\mathrm{DDH}\mathsf{adv}[\mathcal{B}_{\mathrm{ddh}}, \mathbb{G}] = |p_1 - p_0|$. Moreover, if $\mathbf{C}_{\mathrm{ddh}}$ is running Experiment 0, then adversary $\mathcal{A}$ is playing our Game 0, and so $p_0 = \Pr[W_0]$, and if $\mathbf{C}_{\mathrm{ddh}}$ is running Experiment 1, then $\mathcal{A}$ is playing our Game 1, and so $p_1 = \Pr[W_1]$. Equation (11.17) now follows immediately.

Game 2. Observe that in Game 1, w is completely random, and is used only as an input to H. This allows us to play our "KDF card." The challenger in this game is as in Fig. 11.4, except with the following lines modified as indicated:

```
(1)     $\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $\gamma \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $u \leftarrow g^{\alpha}$, $v \leftarrow g^{\beta}$, $w \leftarrow g^{\gamma}$
(2)     $k \xleftarrow{\mathrm{R}} \mathcal{K}$
```

We may easily derive an efficient KDF adversary $\mathcal{B}_{\mathrm{kdf}}$ that uses $\mathcal{A}$ as a subroutine, such that

$$|\Pr[W_1] - \Pr[W_2]| = \mathrm{KDF}\mathsf{adv}[\mathcal{B}_{\mathrm{kdf}}, H]. \tag{11.18}$$

Adversary $\mathcal{B}_{\mathrm{kdf}}$ plays Attack Game 11.3 against a challenger $\mathbf{C}_{\mathrm{kdf}}$, and plays the role of challenger to $\mathcal{A}$ as in Fig. 11.4, except with the following lines modified as indicated:

```
(1)    $\alpha \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$, $\gamma \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$, $u \leftarrow g^\alpha$, $w \leftarrow g^\gamma$
(2)    obtain $(v, k)$ from $\mathbf{C}_{\text{kdf}}$
```

Additionally,

```
when $\mathcal{A}$ outputs $\hat{b}$:
    if $b = \hat{b}$ then output 1 else output 0
```

We leave it to the reader to verify (11.18).

Observe that in Game 2, the key k is only used to encrypt the challenge plaintext. As such, the adversary is essentially just playing the SS game with respect to $\mathcal{E}_\text{s}$ at this point. We leave it to the reader to describe an efficient SS adversary $\mathcal{B}_\text{s}$ that uses $\mathcal{A}$ as a subroutine, such that

$$|\Pr[W_2] - 1/2| = \text{SS}\mathsf{adv}^*[\mathcal{B}_\text{s}, \mathcal{E}_\text{s}]. \tag{11.19}$$

Combining (11.16), (11.17), (11.18), and (11.19), yields (11.15), which completes the proof of the theorem. □

11.6 A fun application: oblivious transfer based on Diffie-Hellman

Bob likes to be well informed. Every day he logs into his favorite newspaper site, reads the headlines, and downloads the articles that he wants to read. The newspaper charges Bob a few cents for every article that he reads, a price that Bob is happy to pay to support the newspaper. All articles have the same price.

One day Bob realizes that the newspaper can track what articles he is downloading and use this information to learn his political leanings, age group, and other personal information that Bob would rather keep private. He asks his friend Alice the following question:

> The newspaper has articles $m_1, \ldots, m_n \in \mathcal{M}$ and I am interested in reading article number $i \in \{1, \ldots, n\}$. Can I download article m_i without the newspaper learning what article I downloaded?

Alice says that the newspaper can simply send Bob all the articles $m_1, \ldots, m_n$ and Bob will read the ones he is interested in. The newspaper does not learn what articles Bob reads.

When Bob asked the newspaper to implement this scheme, they refused saying that this is a sure way for the newspaper to go out of business. Their readers will pay a few cents for a single article and get the entire newspaper. The newspaper would go along with the proposal if it could ensure that Bob only receives one article per request.

Oblivious transfer. Let's call the newspaper the *sender* and call Bob the *receiver*. Bob wants a solution to the following problem:

> The sender has data $m_1, \ldots, m_n \in \mathcal{M}$ and the receiver has an index $i \in \{1, \ldots, n\}$. They want a protocol with the following property: when the protocol completes the receiver learns m_i and nothing else, while the sender learns nothing about i.

This problem is called 1-out-of-n **oblivious transfer**, or simply 1-out-of-n **OT**. It comes up in many settings beyond newspapers, such as privately buying digital content from an online store without the store learning what item is being purchased. More generally, OT is useful whenever there is a need to privately request a record in a proprietary database. It is also a central building block in many cryptographic protocols, as we will see in Chapter 23.

A warning. Defining security for OT protocols is quite subtle. Here we present two very simple OT protocols and discuss their security somewhat informally. We caution that both of these protocols require modification to ensure that they can be used securely when running several instances of the protocols concurrently, and when using them as subprotocols in arbitrary applications. Our goal here is to just get across some of the essential ideas, not to present protocols that should be deployed "as is" in arbitrary applications. We will discuss these security issues in Chapter 23.

11.6.1 A secure OT from ElGamal encryption

Our first 1-out-of-n OT protocol satisfies the basic OT security properties in the random oracle model, assuming only CDH. We let $\mathcal{M}$ denote the message space for the sender's messages, so that $m_1, \ldots, m_n \in \mathcal{M}$. The protocol uses the following ingredients:

- a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$;
- a hash function $H : \mathbb{G}^2 \to \mathcal{K}$, which will be modeled as a random oracle;
- a semantically secure symmetric cipher $(E_{\mathrm{s}}, D_{\mathrm{s}})$ with key space $\mathcal{K}$ and message space $\mathcal{M}$.

We also assume that the sender and receiver communicate over a secure channel (i.e., one that utilizes authenticated encryption as discussed in Chapter 9).

With these ingredients in place, we are ready to describe an elegant OT protocol using ElGamal encryption in the group $\mathbb{G}$. We use a variant of the system $\mathcal{E}_{\mathrm{EG}}$ from Section 11.5. The sender generates n ElGamal public keys $u_1, \ldots, u_n \in \mathbb{G}$ and encrypts message m_j under public key u_j, for $j = 1, \ldots, n$. The clever idea is a mechanism that ensures that the recipient knows the secret key for the public key u_i, but does not know the secret key for any the other key. This lets the recipient decrypt the ith ciphertext, obtaining m_i, but reveals nothing else. Let's see how the protocol works and then discuss its security.

- *step 1:* the sender chooses $\beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, computes $v \leftarrow g^{\beta} \in \mathbb{G}$, and sends v to the receiver.
- *step 2:* the receiver chooses $\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, computes $u \leftarrow g^{\alpha} v^{-i} \in \mathbb{G}$, and sends u to the sender.
- *step 3:* for $j = 1, \ldots, n$ the sender computes

$$\begin{array}{lll} u_j \leftarrow u \cdot v^j \in \mathbb{G} & // & \textit{construct an ElGamal public key } u_j = g^{\alpha} v^{j-i} \\ w_j \leftarrow u_j^{\beta} & // & \textit{construct an ElGamal ciphertext } (v, c_j) = (g^{\beta}, c_j) \\ k_j \leftarrow H(v, w_j) \in \mathcal{K} & // & \quad \textit{using randomness } \beta \textit{ and public key } u_j, \\ c_j \xleftarrow{\mathrm{R}} E_{\mathrm{s}}(k_j, m_j) & // & \quad \textit{that is the encryption of } m_j \in \mathcal{M} \end{array}$$

  and sends the vector of ciphertexts $C := (c_1, \ldots, c_n)$ to the receiver. Note that all n ElGamal ciphertexts $(v, c_1), \ldots, (v, c_n)$ are generated using the same encryption randomness β.

- *step 4:* the receiver, who has the secret key α for the ElGamal public-key $u_i = g^{\alpha}$, decrypts c_i as follows:

$$\begin{aligned}&\text{compute } w \leftarrow v^{\alpha},\ k \leftarrow H(v, w)\\ &\text{output } m \leftarrow D_{\mathrm{s}}(k, c_i)\end{aligned}$$

Let's verify that this works correctly. During decryption in step 4, the key k is computed as $k \leftarrow H(v, w)$, where $w = v^{\alpha} = g^{\alpha\beta}$. In step 3 the key k_i is computed as $k_i \leftarrow H(v, w_i)$, where

$$w_i = u_i^{\beta} = (uv^i)^{\beta} = \big((g^{\alpha}v^{-i})v^i\big)^{\beta} = g^{\alpha\beta}.$$

Hence, when the parties honestly follow the protocol, we have $w_i = w$ and $k_i = k$, and thus the receiver correctly obtains m_i.

Is the protocol secure? The only information that the sender learns about the receiver's input $i \in \{1, \ldots, n\}$ is the group element $u = g^{\alpha}v^{-i}$ computed in step 2. Because α is not used in any other message sent to the sender, the quantity g^{α} is uniformly distributed over the group $\mathbb{G}$, and therefore perfectly hides v^{-i}. Hence, the sender learns nothing about i, as required.

To argue that the receiver learns at most one message, it suffices to argue that the receiver can query the random oracle representing H at no more than one point of the form (v, w_j) for $j = 1, \ldots, n$. This will ensure that the receiver learns at most one decryption key, and hence at most one message. We briefly argue that if the receiver could query the random oracle at two such points, say (v, w_{j_1}) and (v, w_{j_2}), then we could break the CDH for $\mathbb{G}$. In particular, we can transform such a receiver into an algorithm that takes as input $v = g^{\beta}$ and produces a tuple

$$\big(u, j_1, j_2,\quad w_{j_1} = (uv^{j_1})^{\beta},\quad w_{j_2} = (uv^{j_2})^{\beta}\big) \tag{11.20}$$

for some $j_1 \neq j_2$. Dividing the two right most terms in (11.20) one by the other, and raising the result to the power of $1/(j_1 - j_2)$ gives v^{β} which is the same as $g^{(\beta^2)}$. Hence, we get an algorithm that takes as input g^{β} and computes $g^{(\beta^2)}$. We showed in Exercise 10.17 that this problem is equivalent to CDH in $\mathbb{G}$. We conclude that if CDH holds in $\mathbb{G}$ then the receiver can learn at most one message.

Remark 11.2. As we cautioned at the beginning of the section, the protocol above must be modified in order to satisfy stronger security properties. See [10] for a similar protocol that can be proven to satisfy such properties. In particular, the above informal security analysis did not take into account the possibility of attacks that can arise when multiple instances of the protocol run concurrently. Exercise 11.20 discusses such attacks, and shows how to prevent them by modifying the protocol in two ways. First, in deriving the secret keys using the hash function H, the hash function should also take as input the identities of the two parties (and possibly some type of "channel binding" or "session ID", as in Section 21.8). Second, instead of a semantically secure symmetric cipher, we should use a cipher that provides one-time authenticated encryption. □

11.6.2 Adaptive oblivious transfer

Going back to our newspaper scenario, the OT protocol presented above satisfies the security properties we want. However, when trying to deploy it, the newspaper hits a serious performance problem. For every article that Bob wants to download, the newspaper has to re-encrypt all the articles in the newspaper under new keys and send all these fresh ciphertexts to Bob. The computation and communication costs per download would be prohibitive. In case of an online bookstore, the entire bookstore would need to be re-encrypted for every purchase.

It would be much better if the newspaper could encrypt all the articles once and post the encrypted articles on its web site for anyone to download. Bob, and other readers, will download the entire encrypted newspaper. Later, every time Bob pays for an article, the newspaper will quickly open the article for Bob without learning which article was opened. The computation and communication costs should be independent of the number of articles n. Bob will repeat this for as many articles as he wants to read.

An OT protocol that operates this way is called an **adaptive OT**. The word "adaptive" refers to Bob choosing the next article to open in a way that depends on the previously opened articles. Here we describe a simple protocol that satisfies the basic adaptive OT security requirements. That is, a receiver (even a malicious one) learns at most one message per run of the protocol, and the sender (even a malicious one) learns nothing about which messages were opened. In addition, the protocol should ensure that a malicious sender cannot make the receiver open the wrong message. This will provide consistency: whenever one or several receivers request the ith message, they always get the same message.

As we cautioned above, this protocol must be modified in order to satisfy stronger security properties. See [79] for a protocol that can be proven to satisfy such properties.

Before describing the adaptive OT we need a quick interlude to describe a useful, related concept called an *oblivious PRF*.

11.6.3 Oblivious PRFs

Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$. Suppose that one party, the *sender*, has the PRF key $k \in \mathcal{K}$. Another party, the *receiver*, has an input $x \in \mathcal{X}$. An **oblivious PRF protocol**, or simply an **OPRF protocol**, is a protocol that lets the receiver learn the output $y = F(k, x)$ without learning anything else about the value of the PRF at any other input; in addition, the sender learns nothing about the input x. Moreover, the OPRF protocol may be run many times, allowing the receiver to learn the value of $F(k, \cdot)$ for several inputs. The sender should learn nothing about any of these inputs, and the receiver should learn nothing about the value of $F(k, \cdot)$ at any other inputs. Actually, all we will really require here is that the receiver cannot learn anything about the value of $F(k, \cdot)$ at ℓ inputs unless it interacts with the sender ℓ or more times.

We can construct a simple OPRF protocol from the secure PRF F' presented in Exercise 11.3. That PRF makes use of a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$ and is defined as

$$F'(k, x) := H'\Big(x,\ H(x)^k\Big) \quad \text{for } k \in \mathbb{Z}_q,\ \ x \in \mathcal{X}. \tag{11.21}$$

Here $H : \mathcal{X} \to \mathbb{G}$ and $H' : \mathcal{X} \times \mathbb{G} \to \mathcal{Y}$ are hash functions modeled as random oracles. The key space is $\mathcal{K} = \mathbb{Z}_q$. In Exercise 11.3, you are asked to prove that this PRF is secure assuming CDH holds for $\mathbb{G}$, and we model H and H' as random oracles.

Protocol OPRF1

We now describe an OPRF protocol for F', which we call *Protocol OPRF1.* Suppose that the sender has a key $k \in \mathbb{Z}_q$. In each run of the protocol, the receiver has an input $x \in \mathcal{X}$, and the sender and receiver interact (over a secure channel) as follows:

- *receiver:* choose $\rho \xleftarrow{\text{R}} \mathbb{Z}_q \setminus \{0\}$, compute $v \leftarrow H(x)^\rho \in \mathbb{G}$, and send v to the sender.
- *sender:* compute $w \leftarrow v^k \in \mathbb{G}$ and send w to the receiver.
- *receiver:* compute and output $y \leftarrow H'\big(x,\ w^{1/\rho}\big) \in \mathcal{Y}$.

First, observe that if sender and receiver are honest, then in the last step, the receiver computes

$$w^{1/\rho} = (v^k)^{1/\rho} = \Big(\big(H(x)^\rho\big)^k\Big)^{1/\rho} = H(x)^k,$$

and so the receiver ends up with the correct output.

Second, observe that because of the way the receiver computes v, the value v leaks no information to the sender about the input x: the sender just sees a random group element, statistically independent of x.

Now suppose the receiver is honest but the sender is malicious. In this case, the sender still learns nothing about x; however, the sender could send a wrong value for w, causing the receiver to end up with an incorrect value for the output $y = F'(k, x)$. Thus, Protocol OPRF1 is only partially secure if the sender is malicious: while the sender does not learn the receiver's input, the receiver may end up with the wrong output.

Now consider how much information a possibly malicious receiver can obtain by interacting with the sender. We want to argue that a receiver cannot learn anything about the value of $F'(k, \cdot)$ at ℓ inputs unless it interacts with the sender at least ℓ times. This follows directly from a reasonable assumption called the **one-more Diffie-Hellman** or **1MDH** assumption, if we model H and H' as random oracles.

Very briefly, we can describe the 1MDH assumption in terms of the following attack game. In this game, the challenger chooses $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$, computes $u \leftarrow g^\alpha \in \mathbb{G}$, and sends u to the adversary. Next, the adversary makes a sequence of queries to the challenger, each of which can be one of the following types:

Challenge query: the challenger chooses $v \xleftarrow{\text{R}} \mathbb{G}$, and sends v to the adversary.

CDH solve query: the adversary submits $\hat{v} \in \mathbb{G}$ to the challenger, who computes and sends $\hat{w} \leftarrow \hat{v}^\alpha$ to the adversary.

At the end of the game, the adversary outputs a list of distinct pairs, each of the form (i, w), where i is a positive integer bounded by the number of challenge queries, and $w \in \mathbb{G}$. We call such a pair (i, w) *correct* if $w = v^\alpha$ and v is the challenger's response to the ith challenge query. We say the adversary wins the game if the number of correct pairs exceeds the number of CDH solve queries. The 1MDH assumption says that every efficient adversary wins this game with at most negligible probability. In particular, an efficient adversary who sees a number of challenges raised to the power α, cannot produce any other challenge raised to the power α.

Clearly, the 1MDH assumption implies the CDH assumption. It is also not too hard to see that the 1MDH assumption implies that in the above OPRF construction, a receiver cannot learn anything about the value of $F'(k, \cdot)$ at ℓ inputs unless it interacts with the sender at least ℓ times. In particular, the receiver cannot query the random oracle representing H' at ℓ distinct points of the form $(x, H(x)^k)$ unless it interacts with the sender at least ℓ times. We leave this argument as an exercise to the reader.

We note that in certain groups $\mathbb{G}$, winning in the 1MDH game can be slightly easier than computing discrete log in $\mathbb{G}$. In these groups there is also an attack on the scheme OPRF1 that is slightly faster than computing discrete log in $\mathbb{G}$. The details are in Section 16.1.4.

Remark 11.3. When using the scheme OPRF1, it is important that the sender verify that v is in $\mathbb{G}$ before responding, otherwise the sender's response could inadvertently expose the secret key k (e.g., using an attack similar to the one in Exercises 12.3 or 15.1). □

Protocol OPRF2

As we saw, Protocol OPRF1 is only partially secure in the presence of a malicious sender. We can easily modify the protocol to get one that provides a stronger security property in the presence of a malicious sender. In particular, this modified protocol, which we call *Protocol OPRF2*, guarantees that if the receiver's computed output y is wrong, then it is in fact a random value in $\mathcal{Y}$ that is completely independent of the sender's view. The protocol begins by having the sender publish the value $u = g^k$, where k is the sender's secret key. In each run of the protocol, the receiver has an input $x \in \mathcal{X}$, and the sender and receiver interact (over a secure channel) as follows:

- *receiver:* choose $\rho \xleftarrow{\text{R}} \mathbb{Z}_q \setminus \{0\}$, $\tau \xleftarrow{\text{R}} \mathbb{Z}_q$, compute $v \leftarrow H(x)^\rho \cdot g^\tau \in \mathbb{G}$, and send v to sender. Note that this v is computed differently than in OPRF1.
- *sender:* compute $w \leftarrow v^k \in \mathbb{G}$ and send w to the receiver.
- *receiver:* compute and output $y \leftarrow H'\big(x,\ (w/u^\tau)^{1/\rho}\big) \in \mathcal{Y}$.

Observe that if sender and receiver are honest, then in the last step, the receiver computes

$$(w/u^\tau)^{1/\rho} = \Big(v^k/u^\tau\Big)^{1/\rho} = \Big(\big(H(x)^\rho \cdot g^\tau\big)^k/g^{k\tau}\Big)^{1/\rho} = H(x)^k,$$

and so the receiver ends up with the correct output $y = F'(k, x)$.

Next, observe that because of the way the receiver computes v, the value v leaks no information to the sender about the input x, and also leaks no information about the exponent ρ.

Now suppose the receiver is honest but the sender is malicious. If the sender responds in the second step with a wrong value for w, say $w = v^k \cdot d$, where $d \neq 1$, then the receiver computes

$$(w/u^\tau)^{1/\rho} = \Big(v^k d/u^\tau\Big)^{1/\rho} = H(x)^k \cdot d^{1/\rho}.$$

As we observed above, the sender knows nothing about ρ, and so from the sender's point of view, the value $d^{1/\rho} \in \mathbb{G}$ is just a random group element (uniformly distributed over $\mathbb{G}$). It then follows that the output value y computed by the receiver is also just a random element of $\mathcal{Y}$.

Finally, suppose the sender is honest but the receiver is malicious. Just as for Protocol OPRF1, under the 1MDH assumption, if we model H and H' as random oracles, then the receiver cannot learn anything about the value of $F'(k, \cdot)$ at ℓ inputs unless it interacts with the sender ℓ or more times.

11.6.4 A simple adaptive OT from an oblivious PRF

We will need the following ingredients:

- A PRF F defined over $(\mathcal{K}_{\text{prf}}, \{1, \ldots, n\}, \mathcal{K})$.
- An OPRF protocol for a PRF F, like Protocol OPRF2 in Section 11.6.3, which is secure against a malicious sender, in the sense that the receiver will either get a correct output or a random output.

- A symmetric cipher (E_s, D_s) defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ that provides one-time authenticated encryption.

The protocol works as follows. Recall that the sender has messages $m_1, \ldots, m_n \in \mathcal{M}$.

- *step 1:* the sender begins by doing:

```
$k \xleftarrow{R} \mathcal{K}_{\text{prf}}$
for $i = 1, \ldots, n$:   $k_i \leftarrow F(k, i)$,   $c_i \xleftarrow{R} E_s(k_i, m_i)$   //  encrypt all messages
send $C := (c_1, \ldots, c_n)$ to the receiver                      //  send all ciphertexts to the receiver
```

- *step 2:* when the receiver wants m_i, the receiver and sender establish a secure channel between them and use the OPRF protocol to compute $F(k, i)$:

  the sender has k and the receiver has $i \in \{1, \ldots, n\}$,
  when done, the receiver has $k_i = F(k, i)$

  Finally, the receiver computes $m_i \leftarrow D_s(k_i, c_i)$.

That's it. If both parties honestly follow the protocol then the receiver obtains m_i, as required. Moreover, step 2 can be repeated as many times as needed. Every time the receiver gets to open another message of its choice from the sender's list. The amount of traffic generated in each invocation of step 2 is relatively small, and in particular, is independent of the size of the sender's list.

As for security, first suppose that the sender is honest but the receiver is malicious. In this case, from the security property of the OPRF protocol, we know that the number of messages that the receiver can decrypt is no more than the number of times the receiver runs step 2 of the protocol.

Second, suppose that the receiver is honest but the sender is malicious. In this case, from the security property of the OPRF protocol, we know that the sender learns nothing about which ciphertexts get decrypted by the receiver. Moreover, we know that if the sender deviates from the OPRF protocol, the receiver will obtain a random decryption key, and by the ciphertext integrity property of a symmetric cipher, the receiver will reject the ciphertext with overwhelming probability.

A final word of caution. To conclude this section we note that there are some inherent limitations to the security of oblivious transfer against corrupt parties. Consider again our newspaper example. Suppose that a corrupt newspaper wants to learn if Bob is reading articles in the finance section of the newspaper. The newspaper could choose its input articles $m_1, \ldots, m_n$ to the OT protocol so that all the articles in the finance section are blank, while articles in other sections are correct newspaper articles. Now, when Bob wants to read an article in the finance section he will carry out an OT with the newspaper and end up with a blank article, despite having paid the newspaper for the article. Bob will undoubtedly complain that the oblivious transfer failed, at which point the newspaper learns that Bob requested an article in the finance section. This violates OT privacy. Of course, Bob could preserve his privacy by not complaining, but that is difficult to enforce in real life.

11.7 Notes

Citations to the literature to be added.

11.8 Exercises

11.1 (From weak PRF to PRF via hashing). This exercise gives a simple application of the random oracle model that will be useful later. Let $F'(k,x)$ be a PRF defined over $(\mathcal{K},\mathcal{X},\mathcal{Y})$ and suppose that F' is weakly secure (as in Definition 4.3). Let $H : \mathcal{M} \to \mathcal{X}$ be a hash function, and define a new PRF over $(\mathcal{K},\mathcal{M},\mathcal{Y})$:

$$F(k,m) := F'\big(k, H(m)\big).$$

Show that F is a secure PRF (in the standard sense of a secure PRF) when H is modeled as a random oracle. In particular, you should show that for every adversary $\mathcal{A}$ attacking F as a PRF, there exists an adversary $\mathcal{B}$ attacking F' as a weak PRF, which is an elementary wrapper around $\mathcal{A}$, such that $\mathrm{PRF}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A},F] \le \mathrm{wPRF}\mathsf{adv}[\mathcal{B},F']$.

11.2 (Simple PRF from DDH). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $H : \mathcal{M} \to \mathbb{G}$ be a hash function, which we shall model as a random oracle (see Section 8.10.2). Let F be the PRF defined over $(\mathbb{Z}_q,\mathcal{M},\mathbb{G})$ as follows:

$$F(k,m) := H(m)^k \quad \text{for } k \in \mathbb{Z}_q,\ m \in \mathcal{M}.$$

Show that F is a secure PRF in the random oracle model for H under the DDH assumption for $\mathbb{G}$. In particular, you should show that for every adversary $\mathcal{A}$ attacking F as a PRF, there exists a DDH adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that $\mathrm{PRF}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A},F] \le \mathrm{DDH}\mathsf{adv}[\mathcal{B},\mathbb{G}] + 1/q$.

Hint: First show that DDH implies that the PRF $F'(k,v) := v^k$ defined over $(\mathbb{Z}_q,\mathbb{G},\mathbb{G})$ is a *weak* PRF (as in Definition 4.3). Use Exercise 10.11 for this. Then use Exercise 11.1 to deduce that F is a secure PRF.

11.3 (Simple PRF from CDH). Continuing with Exercise 11.2, let $H' : \mathcal{M} \times \mathbb{G} \to \mathcal{Y}$ be a hash function, which we again model as a random oracle. Let F' be the PRF defined over $(\mathbb{Z}_q,\mathcal{M},\mathcal{Y})$ as follows:

$$F'(k,m) := H'\left(m,\ H(m)^k\right) \quad \text{for } k \in \mathbb{Z}_q,\ m \in \mathcal{M}.$$

Show that F' is a secure PRF assuming CDH for $\mathbb{G}$, and when H and H' are modeled as random oracles. Section 11.6.2 gives an interesting application for this PRF, exploiting the fact that it can be evaluated obliviously.

Hint: Use the result of Exercise 10.5.

11.4 (Broken variant of RSA). Consider the following broken version of the RSA public-key encryption scheme: key generation is as in $\mathcal{E}_{\mathrm{RSA}}$, but to encrypt a message $m \in \mathbb{Z}_n$ with public key $pk = (n,e)$ do $E(pk,m) := m^e$ in $\mathbb{Z}_n$. Decryption is done using the RSA trapdoor.

Clearly this scheme is not semantically secure. Even worse, suppose one encrypts a *random* message $m \in \{0,1,\ldots,2^{64}\}$ to obtain $c := m^e \bmod n$. Show that for 35% of plaintexts in $[0,2^{64}]$, an adversary can recover the complete plaintext m from c using only 2^{35} eth powers in $\mathbb{Z}_n$.

Hint: Use the fact that about 35% of the integers m in $[0,2^{64}]$ can be written as $m = m_1 \cdot m_2$ where $m_1, m_2 \in [0,2^{34}]$.

11.5 (Multiplicative ElGamal). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Consider a simple variant of the ElGamal encryption system $\mathcal{E}_{\mathrm{MEG}} = (G, E, D)$ that is defined over $(\mathbb{G}, \mathbb{G}^2)$. The key generation algorithm G is the same as in $\mathcal{E}_{\mathrm{EG}}$, but encryption and decryption work as follows:

- for a given public key $pk = u \in \mathbb{G}$ and message $m \in \mathbb{G}$:

$$E(pk, m) := \quad \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \;\; v \leftarrow g^{\beta}, \;\; e \leftarrow u^{\beta} \cdot m, \;\; \text{output } (v, e)$$

- for a given secret key $sk = \alpha \in \mathbb{Z}_q$ and a ciphertext $(v, e) \in \mathbb{G}^2$:

$$D(sk, \; (v, e) \;) := e/v^{\alpha}$$

(a) Show that $\mathcal{E}_{\mathrm{MEG}}$ is semantically secure assuming the DDH assumption holds in $\mathbb{G}$. In particular, you should show that the advantage of any adversary $\mathcal{A}$ in breaking the semantic security of $\mathcal{E}_{\mathrm{MEG}}$ is bounded by 2ϵ, where ϵ is the advantage of an adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) in the DDH attack game.

(b) Show that $\mathcal{E}_{\mathrm{MEG}}$ is not semantically secure if the DDH assumption does not hold in $\mathbb{G}$.

(c) Show that $\mathcal{E}_{\mathrm{MEG}}$ has the following property: given a public key pk, and two ciphertexts $c_1 \xleftarrow{\text{R}} E(pk, m_1)$ and $c_2 \xleftarrow{\text{R}} E(pk, m_2)$, it is possible to create a new ciphertext c which is an encryption of $m_1 \cdot m_2$. This property is called a **multiplicative homomorphism**.

11.6 (An attack on multiplicative ElGamal). Let p and q be large primes such that q divides $p - 1$. Let $\mathbb{G}$ be the order q subgroup of $\mathbb{Z}_p^*$ generated by $g \in \mathbb{G}$ and assume that the DDH assumption holds in $\mathbb{G}$. Suppose we instantiate the ElGamal system from Exercise 11.5 with the group $\mathbb{G}$. However, plaintext messages are chosen from the entire group $\mathbb{Z}_p^*$ so that the system is defined over $(\mathbb{Z}_p^*, \mathbb{G} \times \mathbb{Z}_p^*)$. Show that the resulting system is not semantically secure.

11.7 (Extending the message space). Suppose that we have a public-key encryption scheme $\mathcal{E} = (G, E, D)$ with message space $\mathcal{M}$. From this, we would like to build an encryption scheme with message space $\mathcal{M}^2$. To this end, consider the following encryption scheme $\mathcal{E}^2 = (G^2, E^2, D^2)$, where

$$\begin{aligned}
G^2() \quad &:= \quad (pk_0, sk_0) \xleftarrow{\text{R}} G(), \quad (pk_1, sk_1) \xleftarrow{\text{R}} G(), \\
&\qquad \text{output } pk := (pk_0, pk_1) \text{ and } sk := (sk_0, sk_1) \\
E^2\big(pk, (m_0, m_1)\big) \quad &:= \quad \big(E(pk_0, m_0), \; E(pk_1, m_1)\big) \\
D^2\big(sk, (c_0, c_1)\big) \quad &:= \quad \big(D(sk_0, c_0), \; D(sk_1, c_1)\big)
\end{aligned}$$

Show that $\mathcal{E}^2$ is semantically secure, assuming $\mathcal{E}$ itself is semantically secure.

11.8 (Encrypting many messages with multiplicative ElGamal). Consider again the multiplicative ElGamal scheme in Exercise 11.5. To increase the message space from a single group element to several, say n, group elements, we could proceed as in the previous exercise. However, the following scheme, $\mathcal{E}_{\mathrm{MMEG}} = (G, E, D)$ defined over $(\mathbb{G}^n, \mathbb{G}^{n+1})$, is more efficient.

- the key generation algorithm runs as follows:

$$\begin{aligned}
G() := \quad & \alpha_i \xleftarrow{\text{R}} \mathbb{Z}_q, \;\; u_i \leftarrow g^{\alpha_i} \;\; (i = 1, \ldots, n) \\
& pk \leftarrow (u_1, \ldots, u_n), \;\; sk \leftarrow (\alpha_1, \ldots, \alpha_n) \\
& \text{output } (pk, sk)
\end{aligned}$$

- for a given public key $pk = (u_1, \ldots, u_n) \in \mathbb{G}^n$ and message $m = (m_1, \ldots, m_n) \in \mathbb{G}^n$:

$$\begin{aligned} E(pk, m) := \quad & \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad v \leftarrow g^{\beta} \\ & e_i \leftarrow u_i^{\beta} \cdot m_i \quad (i = 1, \ldots, n) \\ & \text{output } (v, e_1, \ldots, e_n) \end{aligned}$$

- for a given secret key $sk = (\alpha_1, \ldots, \alpha_n) \in \mathbb{Z}_q^n$ and a ciphertext $c = (v, e_1, \ldots, e_n) \in \mathbb{G}^{n+1}$:

$$D(sk, c) := (e_1/v^{\alpha_1}, \ldots, e_n/v^{\alpha_n})$$

(a) Show that $\mathcal{E}_{\text{MMEG}}$ is semantically secure assuming the DDH assumption holds in $\mathbb{G}$. In particular, you should show that the advantage of any adversary $\mathcal{A}$ in breaking the semantic security of $\mathcal{E}_{\text{MMEG}}$ is bounded by $2(\epsilon + 1/q)$, where ϵ is the advantage of an adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, in the DDH attack game.

Hint: Use Exercise 10.11 with $n = 1$.

(b) Theorem 11.1 shows that semantic security implies CPA security, but the concrete security bound degrades by a factor equal to the number Q of encryption queries. Show that for $\mathcal{E}_{\text{MMEG}}$, the security bound for CPA security only degrades by a factor of n, which is typically much smaller than Q. In particular, show that for every CPA adversary $\mathcal{A}$ there is a DDH adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that $\text{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{MMEG}}]$ is bounded by $2n \cdot (\epsilon + 1/q)$, where ϵ is the advantage of $\mathcal{B}$ in the DDH attack game.

Hint: Use Exercise 10.12.

11.9 (Modular hybrid construction). Both of the encryption schemes presented in this chapter, $\mathcal{E}_{\text{TDF}}$ in Section 11.4 and $\mathcal{E}_{\text{EG}}$ in Section 11.5, as well as many other schemes used in practice, have a "hybrid" structure that combines an asymmetric component and a symmetric component in a fairly natural and modular way. The symmetric part is, of course, the symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$. The asymmetric part can be understood in abstract terms as what is called a **key encapsulation mechanism**, or **KEM**.

A KEM $\mathcal{E}_{\text{kem}}$ consists of a tuple of algorithms $(G, E_{\text{kem}}, D_{\text{kem}})$. Algorithm G is invoked as $(pk, sk) \xleftarrow{\text{R}} G()$. Algorithm E_{kem} is invoked as $(k, c_{\text{kem}}) \xleftarrow{\text{R}} E_{\text{kem}}(pk)$, where $k \in \mathcal{K}$ and $c_{\text{kem}} \in \mathcal{C}_{\text{kem}}$. Algorithm D_{kem} is invoked as $k \leftarrow D_{\text{kem}}(sk, c_{\text{kem}})$, where $k \in \mathcal{K} \cup \{\mathsf{reject}\}$ and $c_{\text{kem}} \in \mathcal{C}_{\text{kem}}$. We say that $\mathcal{E}_{\text{kem}}$ is **defined over** $(\mathcal{K}, \mathcal{C}_{\text{kem}})$. We require that $\mathcal{E}_{\text{kem}}$ satisfies the following **correctness requirement**: for all possible outputs (pk, sk) of $G()$, and all possible outputs (k, c_{kem}) of $E_{\text{kem}}(pk)$, we have $D_{\text{kem}}(sk, c_{\text{kem}}) = k$.

We can define a notion of semantic security in terms of an attack game between a challenger and an adversary $\mathcal{A}$, as follows. In Experiment b, for $b = 0, 1$, the challenger computes

$$(pk, sk) \xleftarrow{\text{R}} G(), \quad (k_0, c_{\text{kem}}) \xleftarrow{\text{R}} E_{\text{kem}}(pk), \quad k_1 \xleftarrow{\text{R}} \mathcal{K},$$

and sends (k_b, c_{kem}) to $\mathcal{A}$. Finally, $\mathcal{A}$ outputs $\hat{b} \in \{0, 1\}$. As usual, if W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}_{\text{kem}}$ as $\text{SS}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{kem}}] := |\Pr[W_0] - \Pr[W_1]|$, and if this advantage is negligible for all efficient adversaries, we say that $\mathcal{E}_{\text{kem}}$ is **semantically secure**.

Now consider the **hybrid public-key encryption scheme** $\mathcal{E} = (G, E, D)$, constructed out of $\mathcal{E}_{\text{kem}}$ and $\mathcal{E}_{\text{s}}$, and defined over $(\mathcal{M}, \mathcal{C}_{\text{kem}} \times \mathcal{C})$. The key generation algorithm for $\mathcal{E}$ is the same as that of $\mathcal{E}_{\text{kem}}$. The encryption algorithm E works as follows:

$$E(pk, m) := \{ \ (k, c_{\text{kem}}) \xleftarrow{\text{R}} E_{\text{kem}}(pk), \ c \xleftarrow{\text{R}} E_{\text{s}}(k, m), \ \text{output } (c_{\text{kem}}, c) \ \}.$$

The decryption algorithm D works as follows:

$$\begin{aligned} D(sk, (c_{\text{kem}}, c)) := \{ \ & m \leftarrow \mathsf{reject}, \ k \leftarrow D_{\text{kem}}(sk, c_{\text{kem}}), \ \text{if } k \neq \mathsf{reject} \text{ then } m \leftarrow D_{\text{s}}(k, c), \\ & \text{output } m \ \}. \end{aligned}$$

(a) Prove that $\mathcal{E}$ satisfies the correctness requirement for a public key encryption scheme, assuming $\mathcal{E}_{\text{kem}}$ and $\mathcal{E}_{\text{s}}$ satisfy their corresponding correctness requirements.

(b) Prove that $\mathcal{E}$ is semantically secure, assuming that $\mathcal{E}_{\text{kem}}$ and $\mathcal{E}_{\text{s}}$ are semantically secure. You should prove a concrete security bound that says that for every adversary $\mathcal{A}$ attacking $\mathcal{E}$, there are adversaries $\mathcal{B}_{\text{kem}}$ and $\mathcal{B}_{\text{s}}$ (which are elementary wrappers around $\mathcal{A}$) such that

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] \leq 2 \cdot \mathsf{SSadv}[\mathcal{B}_{\text{kem}}, \mathcal{E}_{\text{kem}}] + \mathsf{SSadv}[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}].$$

(c) Describe the KEM corresponding to $\mathcal{E}_{\text{TDF}}$ and prove that it is semantically secure (in the random oracle model, assuming $\mathcal{T}$ is one way).

(d) Describe the KEM corresponding to $\mathcal{E}_{\text{EG}}$ and prove that it is semantically secure (in the random oracle model, under the CDH assumption for $\mathbb{G}$).

(e) Let $\mathcal{E}_{\text{a}} = (G, E_{\text{a}}, D_{\text{a}})$ be a public-key encryption scheme defined over $(\mathcal{K}, \mathcal{C}_{\text{a}})$. Define the KEM $\mathcal{E}_{\text{kem}} = (G, E_{\text{kem}}, D_{\text{a}})$, where

$$E_{\text{kem}}(pk) := \{ \ k \xleftarrow{\text{R}} \mathcal{K}, \ c_{\text{kem}} \xleftarrow{\text{R}} E_{\text{a}}(pk, k), \ \text{output } (k, c_{\text{kem}}) \ \}.$$

Show that $\mathcal{E}_{\text{kem}}$ is semantically secure, assuming that $\mathcal{E}_{\text{a}}$ is semantically secure.

Discussion: Part (e) shows that one can always build a KEM from a public-key encryption scheme by just using the encryption scheme to encrypt a symmetric key; however, parts (c) and (d) show that there are more direct and efficient ways to do this.

11.10 (Multi-key CPA security). Generalize the definition of CPA security for a public-key encryption scheme to the multi-key setting. In this attack game, the adversary gets to obtain encryptions of many messages under many public keys. Show that semantic security implies multi-key CPA security. You should show that security degrades linearly in $Q_{\text{k}}Q_{\text{e}}$, where Q_{k} is a bound on the number of keys, and Q_{e} is a bound on the number of encryption queries per key. That is, the advantage of any adversary $\mathcal{A}$ in breaking the multi-key CPA security of a scheme is at most $Q_{\text{k}}Q_{\text{e}} \cdot \epsilon$, where ϵ is the advantage of an adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) that breaks the scheme's semantic security.

11.11 (A tight reduction for multiplicative ElGamal). We proved in Exercise 11.10 that semantic security for a public-key encryption scheme implies multi-key CPA security; however, the security degrades significantly as the number of keys and encryptions increases. Consider the multiplicative ElGamal encryption scheme $\mathcal{E}_{\text{MEG}}$ from Exercise 11.5. You are to show a *tight* reduction from multi-key CPA security for $\mathcal{E}_{\text{MEG}}$ to the DDH assumption, which does not degrade at all as the number of keys and encryptions increases. In particular, you should show that the advantage of any adversary $\mathcal{A}$ in breaking the multi-key CPA security of $\mathcal{E}_{\text{MEG}}$ is bounded by $2(\epsilon + 1/q)$, where ϵ is the advantage of an adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) in the DDH attack game.

Note: You should assume that in the multi-key CPA game, the same group $\mathbb{G}$ and generator $g \in \mathbb{G}$ is used throughout.

Hint: Use Exercise 10.11.

11.12 (An easy discrete log group). Let n be a large integer and consider the following subset of $\mathbb{Z}_{n^2}^*$:

$$\mathbb{G}_n := \big\{ \ [an+1]_{n^2} \in \mathbb{Z}_{n^2}^* : \quad a \in \{0, \ldots, n-1\} \big\}$$

(a) Show that $\mathbb{G}_n$ is a multiplicative subgroup of $\mathbb{Z}_{n^2}^*$ of order n.

(b) Which elements of $\mathbb{G}_n$ are generators?

(c) Choose an arbitrary generator $g \in \mathbb{G}_n$ and show that the discrete log problem in $\mathbb{G}_n$ is easy.

11.13 (Paillier encryption). Let us construct another public-key encryption scheme (G, E, D) that makes use of RSA composites:

- The key generation algorithm is parameterized by a fixed value ℓ and runs as follows:

  $$\begin{aligned} G(\ell) := \quad & \text{generate two distinct random } \ell\text{-bit primes } p \text{ and } q, \\ & n \leftarrow pq, \quad d \leftarrow (p-1)(q-1)/2 \\ & pk \leftarrow n, \quad sk \leftarrow d \\ & \text{output } (pk, sk) \end{aligned}$$

- for a given public key $pk = n$ and message $m \in \{0, \ldots, n-1\}$, set $g := [n+1]_{n^2} \in \mathbb{Z}_{n^2}^*$. The encryption algorithm runs as follows:

  $$E(pk, m) := \quad h \xleftarrow{\text{R}} \mathbb{Z}_{n^2}^*, \quad c \xleftarrow{\text{R}} g^m h^n \in \mathbb{Z}_{n^2}^*, \quad \text{output } c.$$

(a) Explain how the decryption algorithm $D(sk, c)$ works.

Hint: Using the notation of Exercise 11.12, observe that c^d falls in the subgroup $\mathbb{G}_n$ which has an easy discrete log.

(b) Show that this public-key encryption scheme is semantically secure under the following assumption:

let n be a product of two random ℓ-bit primes,
let u be uniform in $\mathbb{Z}_{n^2}^*$,
let v be uniform in the subgroup $(\mathbb{Z}_{n^2})^n := \{h^n \ : \ h \in \mathbb{Z}_{n^2}^*\}$,

then the distribution (n, u) is computationally indistinguishable from the distribution (n, v).

Discussion: This encryption system, called **Paillier encryption**, has a useful property called an additive homomorphism: for ciphertexts $c_0 \xleftarrow{\text{R}} E(pk, m_0)$ and $c_1 \xleftarrow{\text{R}} E(pk, m_1)$, the product $c \leftarrow c_0 \cdot c_1$ is an encryption of $m_0 + m_1 \bmod n$.

11.14 (Hash Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $H : \mathbb{G} \to \mathcal{K}$ be a hash function. We say that the **Hash Diffie-Hellman** (HDH) assumption holds for $(\mathbb{G}, H)$ if the distribution $\big(g^\alpha, g^\beta, H(g^\beta, g^{\alpha\beta})\big)$ is computationally indistinguishable from the distribution (g^α, g^β, k) where $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$ and $k \xleftarrow{R} \mathcal{K}$.

(a) Show that if H is modeled as a random oracle and the CDH assumption holds for $\mathbb{G}$, then the HDH assumption holds for $(\mathbb{G}, H)$.

(b) Show that if H is a secure KDF and the DDH assumption holds for $\mathbb{G}$, then the HDH assumption holds for $(\mathbb{G}, H)$.

(c) Prove that the ElGamal public-key encryption scheme $\mathcal{E}_{\text{EG}}$ is semantically secure if the HDH assumption holds for $(\mathbb{G}, H)$.

11.15 (Anonymous public-key encryption). Suppose t people publish their public-keys $pk_1, \ldots, pk_t$. Alice sends an encrypted message to one of them, say pk_5, but she wants to ensure that no one (other than user 5) can tell which of the t users is the intended recipient. You may assume that every user, other than user 5, who tries to decrypt Alice's message with their secret key, obtains fail.

(a) Define a security model that captures this requirement. The adversary should be given t public keys $pk_1, \ldots, pk_t$ and it then selects the message m that Alice sends. Upon receiving a challenge ciphertext, the adversary should learn nothing about which of the t public keys is the intended recipient. A system that has this property is said to be **an anonymous public-key encryption scheme**.

(b) Show that the ElGamal public-key encryption system $\mathcal{E}_{\text{EG}}$ is anonymous.

(c) Show that the RSA public-key encryption system $\mathcal{E}_{\text{RSA}}$ is not anonymous. Assume that all t public keys are generated using the same RSA parameters ℓ and e.

11.16 (Proxy re-encryption). Bob works for the Acme corporation and publishes a public-key pk_{bob} so that all incoming emails to Bob are encrypted under pk_{bob}. When Bob goes on vacation he instructs the company's mail server to forward all his incoming encrypted email to Alice. Alice's public key is pk_{alice}. The mail server needs a way to translate an email encrypted under public-key pk_{bob} into an email encrypted under public-key pk_{alice}. This would be easy if the mail server had sk_{bob}, but then the mail server can read all of Bob's incoming email.

Consider the variation $\mathcal{E}'_{\text{EG}}$ of $\mathcal{E}_{\text{EG}}$, in which we only hash w instead of (v, w) (see Remark 11.1). Suppose that pk_{bob} and pk_{alice} are public keys for $\mathcal{E}'_{\text{EG}}$. Then the mail server can do the translation from pk_{bob} to pk_{alice} while learning nothing about the email contents.

(a) Suppose $pk_{\text{alice}} = g^\alpha$ and $pk_{\text{bob}} = g^{\alpha'}$. Show that giving $\tau := \alpha'/\alpha$ to the mail server lets it translate an email encrypted under pk_{bob} into an email encrypted under pk_{alice}, and vice-versa.

(b) Assume that $\mathcal{E}'_{\text{EG}}$ is semantically secure. Show that the adversary cannot break semantic security for Alice, even if it is given Bob's public key $g^{\alpha'}$ along with the translation key τ.

11.17 (Online-offline oblivious transfer). In Section 11.6 we looked at protocols for oblivious transfer where one party, a sender, has messages $m_0, \ldots, m_{n-1} \in \mathcal{M} = \{0,1\}^\ell$ and another party, a receiver, has an index $i \in \mathbb{Z}_n$. At the end of the protocol the receiver has m_i and neither party learns anything else about the other party's data. Suppose that the sender and receiver had previously established a random OT instance: the sender has random $x_0, \ldots, x_{n-1} \xleftarrow{\text{R}} \mathcal{M}$ and the recipient has (j, x_j) for some random $j \xleftarrow{\text{R}} \mathbb{Z}_n$. This random data is called an **OT correlation**. Let's show that the parties can quickly solve the OT problem using a random OT correlation:

> Sender has: $(m_1, \ldots, m_{n-1})$, $(x_1, \ldots, x_{n-1})$, Receiver has: (i, j, x_j).
> Receiver wants m_i.
>
> - *receiver:* compute $\Delta \leftarrow (j - i) \in \mathbb{Z}_n$ and send Δ to the sender.
> - *sender:* for all $u = 0, 1, \ldots, n-1$ compute $c_u \leftarrow m_u \oplus x_{(u+\Delta)}$ and send $C := (c_0, \ldots, c_{n-1})$ to the receiver. Here each index $(u + \Delta)$ is an element in $\mathbb{Z}_n$.
> - *receiver:* output $c_i \oplus x_j$.

(a) Show that when both parties honestly follow the protocol, the receiver learns the required value m_i.

(b) Prove that a (possibly malicious) sender learns nothing about the receiver's index i. Similarly, a (possibly malicious) receiver cannot learn anything about any messages besides m_i, where i is defined as $i := j - \Delta \in \mathbb{Z}_n$.

(c) A protocol between the sender and the receiver that constructs a random OT correlation, but where the recipient can choose the $j \in \mathbb{Z}_n$ that it wants, is called a **Random OT** or **ROT protocol**. ROT is discussed in more detail in Section 23.7. Adapt the protocol in this exercise to show that an ROT protocol implies an OT protocol.

11.18 (All-but-one oblivious transfer). **All-but-one OT** refers to the following problem: a sender has messages $m_1, \ldots, m_n \in \mathcal{M}$ and a receiver has an index $i \in \{1, \ldots, n\}$. At the end of the protocol the receiver should have all of $m_1, \ldots, m_n$ *except* for message m_i. Neither party should learn anything else about the other party's inputs. Here is a simple solution using a CPA secure cipher (E, D) defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$:

- *sender:* choose a random k in $\mathcal{K}$ and compute $c_j \xleftarrow{\text{R}} E(k, m_j)$ for $j = 1, \ldots, n$.
- *both:* the sender and receiver engage in n independent 1-out-of-2 oblivious transfers. In OT instance number $j = 1, \ldots, n$, the sender's input data is the pair (c_j, k), and the receiver's input data is a bit b_j in $\{0, 1\}$.

(a) Explain why the receiver can learn at most $n-1$ of the sender's messages, and why the sender learns nothing about i.

(b) Generalize the protocol to allow the receiver to learn at most $n-2$ of the sender's messages, where the sender learns nothing about which ones.

Hint: try using n independent 1-out-of-3 oblivious transfers.

Discussion: The all-but-one OT protocol described in this problem make use of n executions of 1-out-of-2 oblivious transfer. It is possible to implement all-but-one OT using only $\log_2 n$ executions of 1-out-of-2 oblivious transfer [142, Sec. 3].

11.19 (Expanding oblivious transfer). Suppose we are given a 1-out-of-2 oblivious transfer protocol Π. We wish to construct a 1-out-of-n oblivious transfer protocol, for $n > 2$, using only $\lceil \log_2 n \rceil$ parallel executions of the 1-out-of-2 OT protocol Π. For simplicity assume $n = 2^t$, for some $t > 1$. We will need a hash function $H : \mathcal{K}^t \to \mathcal{K}'$, where $\mathcal{K} := \{0,1\}^w$, and a cipher (E, D) defined over $(\mathcal{K}', \mathcal{M}, \mathcal{C})$. The 1-out-of-$n$ OT protocol Π' works as follows:

Sender has: $m_0, \dots, m_{n-1} \in \mathcal{M}$, Receiver has: $i \in \{0, \dots, n-1\}$.
Receiver wants m_i.

- *Setup:* The receiver computes the binary representation of i, namely $b_0, \dots, b_{t-1} \in \{0,1\}$ so that $i = \sum_{j=0}^{t-1} 2^j b_j$. The sender chooses random $k_j[0], k_j[1] \xleftarrow{\text{R}} \mathcal{K}$ for $j = 0, \dots, t-1$.
- *OT phase:* The sender and receiver engage in $t = \log_2 n$ parallel instances of the 1-out-of-2 OT protocol Π, where in instance number j, the receiver's input is $b_j \in \{0,1\}$, and the sender's input is $(k_j[0], k_j[1]) \in \mathcal{K}^2$. At the end of the t protocols, the receiver has $k_0[b_0], \dots, k_{t-1}[b_{t-1}] \in \mathcal{K}$.
- *Sender:* For $\ell = 0, \dots, n-1$, the sender does:
  - let $d_0, \dots, d_{t-1} \in \{0,1\}$ be the binary representation of ℓ,
  - compute $k_\ell \leftarrow H\big(k_0[d_0], \dots, k_{t-1}[d_{t-1}]\big) \in \mathcal{K}'$,
  - set $c_\ell \xleftarrow{\text{R}} E(k_\ell, m_\ell)$.

  Send $\boldsymbol{c} = (c_0, \dots, c_{n-1})$ to the receiver.
- *Receiver:* set $\hat{k}_i \leftarrow H\big(k_0[b_0], \dots, k_{t-1}[b_{t-1}]\big) \in \mathcal{K}'$ and outputs $\hat{m} \leftarrow D(\hat{k}_i, c_i)$.

(a) Show that protocol Π' is correct so that $\hat{m} = m_i$.

(b) Let's prove a minimal security property for Π': the receiver learns nothing about m_j for $j \neq i$. More precisely, consider an efficient adversary that outputs i, and two tuples $\boldsymbol{m} = (m_0, \dots, m_{n-1})$ and $\boldsymbol{m}' = (m'_0, \dots, m'_{n-1})$, with $m_i = m'_i$. The adversary then plays the role of receiver with input i. We say that Π' is secure if the adversary cannot distinguish the message $\boldsymbol{c}$ from the sender when the sender takes $\boldsymbol{m}$ as input versus when the sender takes $\boldsymbol{m}'$ as input. Show that when H is modeled as a random oracle, (E, D) is semantically secure, and the 1-out-of-2 OT protocol Π is secure, then the 1-out-of-n OT protocol Π' is secure.

Discussion: This protocol runs many instances of 1-out-of-2 OT. In Section 23.7 we will show how to efficiently construct many instances of 1-out-of-2 OT, as needed here, using a technique called OT extension.

11.20 (An attack on oblivious transfer). The first OT protocol presented in Section 11.6 is subject to a kind of "man in the middle" attack, when multiple instances of the protocol may run concurrently. However, the protocol is easily repaired to prevent this. Suppose we run *two* OT instances of the protocol: in one instance the adversary plays the role of receiver and interacts with an honest sender; in the other instance the adversary plays the role of sender and interacts with an honest receiver. The adversary relays all messages unchanged between the honest sender and honest receiver. However, when the honest receiver sends $u := g^\alpha v^{-i}$, the adversary sends $\hat{u} := u \cdot v$ to the honest sender.

(a) Show that when the two OT protocols complete, the receiver incorrectly obtains the message m_{i-1} instead of m_i. You may assume $i > 1$.

(b) Now suppose we modify the protocol along the lines discussed on Remark 11.2. Explain why the attack does not work any more.

Discussion: One of the reasons we use secure channels in this protocol is to prevent the adversary from modifying protocol messages. However, even with secure channels, the protocol as stated is subject to a "man in the middle attack" as illustrated in this exercise. Here, the honest sender and honest receiver are talking to the adversary in two separate protocol instances, and we would ideally like these two protocol instances to run relatively independently of each other, without any "strange interactions" between them, such as that illustrated in this exercise. This type of subtle attack, which arises when we run protocol instances concurrently, is one reason why we need better security definitions for interactive protocols, which we will discuss in Section 23.5.

11.21 (Group key exchange). Suppose n parties $A_1, \dots, A_n$ want to setup a group secret key that they can use for encrypted group messaging. They have at their disposal a public bulletin board that they can all post messages to and whose contents are public. No peer-to-peer communication is allowed. Your goal is to design a group key exchange that is secure against passive eavesdropping.

(a) Use a public-key encryption scheme (G, E, D) to design a protocol that runs in two rounds. One party, say A_1, reads and writes n values to and from the bulletin board. All other parties read and write only one value to the bulletin board.

(b) Define a security model for a group key exchange protocol that ensures security against a passive eavesdropper. Prove security of your protocol from part (a) assuming (G, E, D) is semantically secure.

(c) Design a fairer protocol where the work is evenly distributed. Specifically, design a protocol based on two-party Diffie-Hellman that runs in at most k rounds, assuming $n = 2^k$. Let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$ and let $H : \mathbb{G} \to \mathbb{Z}_q$ be a hash function. In each round every party posts at most one group element in $\mathbb{G}$ to the bulletin board and reads at most one group element from the bulletin board. At the end of the protocol, after k rounds, all parties obtain the same secret key and there are at most $2n$ group elements on the bulletin board. You may assume that the parties know each other's identities and can order themselves lexicographically by identity.

Hint: think of an n-leaf binary tree where each leaf corresponds to one party.

(d) Prove security of your protocol from part (c) against a passive eavesdropper assuming CDH holds in $\mathbb{G}$ and H is modeled as a random oracle.

(e) Suppose one of the n parties has a malfunctioning random number generator — whenever that party needs to sample a random value, its random number generator always returns "5". Show that this will sink the entire protocol from parts (a) and (c), even if all the other participants have well functioning random number generators. Specifically, show that an eavesdropper will learn the group secret key.

(f) Is it possible to design a group key exchange protocol that is secure even if at most one of the participants suffers from the problem in part (e)?

Chapter 12

Chosen ciphertext secure public key encryption

In Chapter 11, we introduced the notion of public-key encryption. We also defined a basic form of security called *semantic security*, which is completely analogous to the corresponding notion of semantic security in the symmetric-key setting. We observed that in the public-key setting, semantic security implies security against a chosen plaintext attack, i.e., CPA security.

In this chapter, we study the stronger notion of security against chosen ciphertext attack, or **CCA security**. In the CPA attack game, the decryption key is never used, and so CPA security provides no guarantees in any real-world setting in which the decryption key is actually used to decrypt messages. The notion of CCA security is designed to model a wide spectrum of real-world attacks, and it is considered the "gold standard" for security in the public-key setting.

We briefly introduced the notion of CCA security in the symmetric-key setting in Section 9.2, and the definition in the public-key setting is a straightforward translation of the definition in the symmetric-key setting. However, it turns out CCA security plays a more fundamental role in the public-key setting than in the symmetric-key setting.

12.1 Basic definitions

As usual, we formulate this notion of security using an attack game, which is a straightforward adaptation of the CCA attack game in the symmetric setting (Attack Game 9.2) to the public-key setting.

Attack Game 12.1 (CCA security). For a given public-key encryption scheme $\mathcal{E} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b **(**$b = 0, 1$**):**

- The challenger computes $(pk, sk) \overset{\text{R}}{\leftarrow} G()$ and sends pk to the adversary.
- $\mathcal{A}$ then makes a series of queries to the challenger. Each query can be one of two types:
  - *Encryption query:* for $i = 1, 2, \ldots$, the ith encryption query consists of a pair of messages $(m_{i0}, m_{i1}) \in \mathcal{M}^2$, of the same length. The challenger computes $c_i \overset{\text{R}}{\leftarrow} E(pk, m_{ib})$ and sends c_i to $\mathcal{A}$.

– *Decryption query:* for $j = 1, 2, \ldots$, the jth decryption query consists of a ciphertext $\hat{c}_j \in \mathcal{C}$ that is not among the responses to the previous encryption queries, i.e.,

$$\hat{c}_j \notin \{c_1, c_2, \ldots\}.$$

The challenger computes $\hat{m}_j \leftarrow D(sk, \hat{c}_j)$, and sends $\hat{m}_j$ to $\mathcal{A}$.

- At the end of the game, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b and define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\mathrm{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \bigl|\Pr[W_0] - \Pr[W_1]\bigr|. \quad \square$$

Definition 12.1 (CCA Security). *A public-key encryption scheme $\mathcal{E}$ is called **semantically secure against a chosen ciphertext attack**, or simply **CCA secure**, if for all efficient adversaries $\mathcal{A}$, the value $\mathrm{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible.*

Just as we did in the symmetric-key setting, we can consider a restricted attack game in which the adversary makes only a single encryption query:

Definition 12.2 (1CCA security). *In Attack Game 12.1, if the adversary $\mathcal{A}$ is restricted to making a single encryption query, we denote its advantage by $1\mathrm{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$. A public-key encryption scheme $\mathcal{E}$ is **one-time semantically secure against chosen ciphertext attack**, or simply, **1CCA secure**, if for all efficient adversaries $\mathcal{A}$, the value $1\mathrm{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ is negligible.*

Notice that if we strip away the decryption queries, 1CCA security corresponds to semantic security, and CCA security corresponds to CPA security. We showed in Theorem 11.1 that semantic security for a public-key encryption scheme implies CPA security. A similar result holds with respect to chosen ciphertext security, namely, that 1CCA security implies CCA security.

Theorem 12.1. *If a public-key encryption scheme $\mathcal{E}$ is 1CCA secure, then it is also CCA secure.*

In particular, for every CCA adversary $\mathcal{A}$ that plays Attack Game 12.1 with respect to $\mathcal{E}$, and which makes at most Q_{e} encryption queries to its challenger, there exists a 1CCA adversary $\mathcal{B}$ as in Definition 12.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = Q_{\mathrm{e}} \cdot 1\mathrm{CCA}\mathsf{adv}[\mathcal{B}, \mathcal{E}].$$

The proof is a simple hybrid argument that is almost identical to that of Theorem 11.1, and we leave the details as an easy exercise to the reader. Using another level of hybrid argument, one can also extend this to the multi-key setting as well — see Exercise 12.6.

Since 1CCA security implies CCA security, if we want to prove that a particular public-key encryption scheme is CCA secure, we will typically simply prove 1CCA security. So it will be helpful to study the 1CCA attack game in a bit more detail. We can view the 1CCA attack game as proceeding in a series of phases:

Initialization phase: the challenger generates $(pk, sk) \xleftarrow{\mathrm{R}} G()$ and sends pk to the adversary.

Phase 1: the adversary submits a series of decryption queries to the challenger; each such query is a ciphertext $\hat{c} \in \mathcal{C}$, to which the challenger responds with $\hat{m} \leftarrow D(sk, \hat{c})$.

Encryption query: the adversary submits a single encryption query (m_0, m_1) to the challenger; in Experiment b (where $b = 0, 1$), the challenger responds with $c \xleftarrow{R} E(pk, m_b)$.

Phase 2: the adversary again submits a series of decryption queries to the challenger; each such query is a ciphertext $\hat{c} \in \mathcal{C}$, subject to the restriction that $\hat{c} \neq c$, to which the challenger responds with $\hat{m} \leftarrow D(sk, \hat{c})$.

Finish: at the end of the game, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

As usual, as discussed in Section 2.2.5, Attack Game 12.1 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0, 1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\text{CCA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]$ (and $1\text{CCA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]$) as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.2.5 applies here as well:

$$\text{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \text{CCA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]. \tag{12.1}$$

And similarly, for adversaries restricted to a single encryption query, we have:

$$1\text{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot 1\text{CCA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]. \tag{12.2}$$

12.2 Understanding CCA security

The definition of CCA security may seem rather unintuitive at first. Indeed, one might ask: in the attack game, why can the adversary get any message decrypted except the ones he really wants to decrypt? One answer is that without this restriction, it would be impossible to satisfy the definition. However, this is not a very satisfying answer, and it begs the question as to whether the entire definitional framework makes sense.

In this section, we explore the definition of CCA security from several angles. Hopefully, by the end, the reader will understand why this definition makes sense, and what it is good for.

12.2.1 CCA security and ciphertext malleability

Our first example illustrates an important property of CCA secure systems: they are **non-malleable**. We will not formally define it here, but very roughly speaking, *non-malleability* means that given the encryptions of several unknown messages, it is infeasible for an adversary to create new ciphertexts that decrypt to messages that bear some specific relation with the original messages.

Consider a professor, Bob, who collects homework by email. Moreover, assume that Bob generates a public key/secret key pair (pk, sk) for a public-key encryption scheme, and gives pk to all of his students. When a student Alice submits an email, she encrypts it under pk.

To make things concrete, suppose that the public-key encryption scheme is the semantically secure scheme $\mathcal{E}_{\text{TDF}}$ presented in Section 11.4, which is based on a trapdoor function along with some symmetric cipher $\mathcal{E}_{\text{s}}$. The only requirement on $\mathcal{E}_{\text{s}}$ is that it is semantically secure, so let us assume that $\mathcal{E}_{\text{s}}$ is a stream cipher (such as AES in counter mode).

When Alice encrypts the email message m containing her homework using $\mathcal{E}_{\text{TDF}}$ and pk, the resulting ciphertext is of the form (y, c), where $y = F(pk, x)$ and $c = G(H(x)) \oplus m$. Here, H is a hash function and G is a PRG.

As we saw in Section 3.3.2, any stream cipher is extremely *malleable*, and the public-key scheme $\mathcal{E}_{\mathrm{TDF}}$ inherits this weakness. In particular, an attacker Molly can do essentially the same thing here as she did in Section 3.3.2. Namely, assuming that Alice's email message m starts with the header `From:Alice`, by flipping a few bits of the symmetric-key ciphertext c, Molly obtains another ciphertext c' that decrypts (under the same symmetric key) to a message m' that is identical to m, except that the header now reads `From:Molly`. Molly can do this without knowing anything at all about Alice's email message other than the header.

Using the above technique, Molly can "steal" Alice's homework as follows. She intercepts Alice's ciphertext (y, c). She then modifies the symmetric-key ciphertext c to obtain c' as above, and sends the public-key ciphertext (y, c') to Bob. Now, when Professor Bob decrypts (y, c'), he will essentially see Alice's homework, but Bob will mistakenly think that the homework was submitted by Molly, and give Molly credit for it.

The attack described so far is a good example of a chosen ciphertext attack, which could not succeed if the public-key encryption scheme were actually CCA secure. Indeed, if given (y, c) it is possible for Molly to create a new ciphertext (y, c') where the header `From:Alice` is changed to `From:Molly`, then the system cannot be CCA secure. For such a system, we can design a simple CCA adversary $\mathcal{A}$ that has advantage 1 in the CCA security game. Here is how.

- Create a pair of messages, each with the same header, but different bodies. Our adversary $\mathcal{A}$ submits this pair as an encryption query, obtaining (y, c).
- $\mathcal{A}$ then uses Molly's algorithm to create a ciphertext (y, c'), which should encrypt a message with a different header but the same body.
- $\mathcal{A}$ then submits (y, c') as a decryption query, and outputs 0 or 1, depending on which body it sees.

As we have shown, if Alice encrypts her homework using a CCA-secure system, she is assured that no one can steal her homework by modifying the ciphertext she submitted. CCA security, however, does not prevent all attacks on this homework submission system. An attacker can maliciously submit a homework on behalf of Alice, and possibly hurt her grade in the class. Indeed, anyone can send an encrypted homework to the professor, and in particular, a homework that begins with `From:Alice`. Preventing this type of attack requires tools that we will develop later. In Section 13.7, where we develop the notion of signcryption, which is one way to prevent this attack.

12.2.2 CCA security vs. authentication

When we first encountered the notion of CCA security in the symmetric-key setting, back in Section 9.2, we saw that CCA security was implied by AE security, i.e., ciphertext integrity plus CPA security. Moreover, we saw that ciphertext integrity could be easily added to any CPA-secure encryption scheme using the encrypt-then-MAC method. We show here that this does not work in the public-key setting: simply adding an authentication wrapper does not make the system CCA secure.

Consider again the homework submission system example in the previous section. If we start with a scheme, like $\mathcal{E}_{\mathrm{TDF}}$, which is not itself CCA secure, we might hope to make it CCA secure using encrypt-then-MAC: Alice wraps the ciphertext (y, c) with some authentication data computed from (y, c). Say, Alice computes a MAC tag t over (y, c) using a secret key that she shares with Bob and sends (y, c, t) to Bob (or, instead of a MAC, she computes a digital signature on (y, c), a concept discussed in Chapter 13). Bob can check the authentication data to make sure the ciphertext was generated by Alice. However, regardless of the authentication wrapper used, Molly can still carry out the attack described in the previous section. Here is how. Molly intercepts Alice's ciphertext (y, c, t), and computes (y, c') exactly as before. Now, since Molly is a registered student in Bob's course, she presumably is using the same authentication mechanism as all other students, so she simply computes her own authentication tag t' on ciphertext (y, c') and sends (y, c', t') to Bob. Bob receives (y, c', t'), and believes the authenticity of the ciphertext. When Bob decrypts (y, c'), the header `From:Molly` will look perfectly consistent with the authentication results.

What went wrong? Why did the strategy of authenticating ciphertexts provide us with CCA security in the symmetric-key setting, but not in the public-key setting? The reason is simply that in the public-key setting, anyone is allowed to send an encrypted message to Bob using Bob's public key. The added flexibility that public-key encryption provides makes it more challenging to achieve CCA security, yet CCA security is vital for security in real-world systems. (We will discuss in detail how to securely combine CCA-secure public-key encryption and digital signatures when we discuss signcryption in Section 13.7.)

12.2.3 CCA security and key escrow

Consider again the key escrow example discussed in Section 11.1.2. Recall that in that example, Alice encrypts a file f using a symmetric key k. Among other things, Alice stores along with the encrypted file an *escrow* of the file's encryption key. Here, the escrow is an encryption c_{ES} of k under the public key of some *escrow service*. If Alice works for some company, then if need be, Alice's manager or other authorized entity can retrieve the file's encryption key by presenting c_{ES} to the escrow service for decryption.

If the escrow service uses a CCA-secure encryption scheme, then it is possible to implement an *access control policy* which can guard against potential abuse. This can be done as follows. Suppose that in forming the escrow-ciphertext c_{ES}, Alice encrypts the pair (k, h) under the escrow service's public key, where h is a collision-resistant hash of the metadata md associated with the file f: this might include the identity of owner of the file (Alice, in this case), the name of the file, the time that it was created and/or modified. Let us also assume that all of this metadata md is stored on the file system in the clear along with the encrypted file.

Now suppose a requesting entity presents the escrow-ciphertext c_{ES} to the escrow service, along with the corresponding metadata md. The escrow service may impose some type of access control policy, based on the given metadata, along with the identity or credentials of the requesting entity. Such a policy could be very specific to a particular company or organization. For example, the requesting entity may be Alice's manager, and it is company policy that Alice's manager should have access to all files owned by Alice. Or the requesting entity may be an external auditor that is to have access to all files created by certain employees on a certain date.

To actually enforce this access control policy, not only must the escrow service verify that the requesting entity's credentials and the supplied metadata conform to the access control policy, the escrow service must also perform the following check: after decrypting the escrow-ciphertext c_{ES} to obtain the pair (k, h), it must check that h matches the hash of the metadata supplied by the requesting entity. Only if these match does the escrow service release the key k to the requesting entity.

This type of access control can prevent certain abuses. For example, consider the external auditor who has the right to access all files created by certain employees on a certain date. Suppose the auditor himself is a bit too nosy, and during the audit, wants to find out some information in a personal file of Alice that is not one of the files targeted by the audit. The above implementation of the escrow service, along with CCA security, ensures that the nosy auditor cannot obtain this unauthorized information. Indeed, suppose c_{ES} is the escrow-ciphertext associated with Alice's personal file, which is not subject to the audit, and that this file has metadata md. Suppose the auditor submits a pair (c'_{ES}, md') to the escrow service. There are several cases to consider:

- if $md' = md$, then the escrow service will reject the request, as the metadata md of Alice's personal file does not fit the profile of the audit;
- if $md' \neq md$ and $c'_{\mathrm{ES}} = c_{\mathrm{ES}}$, then the collision resistance of the hash ensures that the escrow service will reject the request, as the hash embedded in the decryption of c'_{ES} will not match the hash of the supplied metadata md';
- if $md' \neq md$ and $c'_{\mathrm{ES}} \neq c_{\mathrm{ES}}$, then the escrow service may or may not accept the request, but even if it does, CCA security and the fact that $c'_{\mathrm{ES}} \neq c_{\mathrm{ES}}$ ensures that no information about the encryption key for Alice's personal file is revealed.

This implementation of an escrow service is pretty good, but it is far from perfect:

- It assumes that Alice follows the protocol of actually encrypting the file encryption key along with the correct metadata. Actually, this may not be such an unreasonable assumption, as these tasks will be performed automatically by the file system on Alice's behalf, and so it may not be so easy for a misbehaving Alice to circumvent this protocol.
- It assumes that the requesting entity and the escrow service do not collude.

Treating the metadata as associated data. In Section 12.7 we define public-key encryption with associated data, which is the public-key analogue of symmetric encryption with associated data from Section 9.5. Here the public-key encryption and decryption algorithms take a third input called associated data. The point is that decryption reveals no useful information if the given associated data used in decryption is different from the one used in encryption.

The metadata information md in the escrow system above can be treated as associated data, instead of appending it to the plaintext. This will result in a smaller ciphertext while achieving the same security goals. In fact, associating metadata to a ciphertext for the purpose described above is a very typical application of associated data in a public-key encryption scheme.

12.2.4 Encryption as an abstract interface

To conclude our motivational discussion of CCA security we show that it abstractly captures a "correct" and very natural notion of security. We do this by describing encryption as an abstract interface, as discussed in Section 9.3 in the symmetric case.

The setting is as follows. We have a sender S and receiver R, who are participating in some protocol, during which S drops messages $m_1, m_2, \ldots$ into his out-box, and R retrieves messages from his in-box. While S and R do not share a secret key, we assume that R has generated public key/secret key pair (pk, sk), and that S knows R's public key pk.

That is the abstract interface. In a real implementation, when m_i is placed in S's out-box, it is encrypted under pk, yielding a corresponding ciphertext c_i, which is sent over the wire to R. On the receiving end, when a ciphertext $\hat{c}$ is received at R's end of the wire, it is decrypted using sk, and if the decryption is a message $\hat{m} \neq \mathsf{reject}$, the message $\hat{m}$ is placed in R's in-box.

Note that while we are syntactically restricting ourselves to a single sender S, this restriction is superficial: in a system with many users, all of them have access to R's public key, and so we can model such a system by allowing all users to place messages in S's out-box.

Just as in Section 9.3, an attacker may attempt to subvert communication in several ways:

- The attacker may drop, re-order, or duplicate the ciphertexts sent by S.

- The attacker may modify ciphertexts sent by S, or inject ciphertexts computed in some arbitrary fashion.

- The attacker may have partial knowledge — or even influence the choice — of the messages sent by S.

- The attacker can obtain partial knowledge of some of the messages retrieved by R, and determine if a given ciphertext delivered to R was rejected.

We now describe an *ideal implementation* of this interface. It is slightly different from the ideal implementation in Section 9.3 — in that section, we were working with the notion of AE security, while here we are working with the notion of CCA security. When S drops m_i in its out-box, instead of encrypting m_i, the ideal implementation creates a ciphertext c_i by encrypting a dummy message $dummy_i$, that has nothing to do with m_i (except that it should be of the same length). Thus, c_i serves as a "handle" for m_i, but does not contain any information about m_i (other than its length). When c_i arrives at R, the corresponding message m_i is magically copied from S's out-box to R's in-box. If a ciphertext $\hat{c}$ arrives at R that is not among the previously generated c_i's, the ideal implementation *decrypts $\hat{c}$ using sk as usual.*

CCA security implies that this ideal implementation of the service is for all practical purposes equivalent to the real implementation. In the ideal implementation, we see that messages magically jump from S to R, in spite of any information the adversary may glean by getting R to decrypt other ciphertexts — the ciphertexts generated by S in the ideal implementation serve simply as handles for the corresponding messages, but do not carry any other useful information. Hopefully, analyzing the security properties of a higher-level protocol will be much easier using this ideal implementation.

Note that even in the ideal implementation, the attacker may still drop, re-order, or duplicate ciphertexts, and these will cause the corresponding messages to be dropped, re-ordered, or duplicated. A higher-level protocol can easily take measures to deal with these issues.

We now argue informally that when $\mathcal{E}$ is CCA secure, the real world implementation is indistinguishable from the ideal implementation. The argument is similar to that in Section 9.3. It proceeds in two steps, starting with the real implementation, and in each step, we make a slight modification.

- First, we modify the real implementation of R's in-box, as follows. When a ciphertext $\hat{c}$ arrives on R's end, the list of ciphertexts $c_1, c_2, \ldots$ previously generated by S is scanned, and if $\hat{c} = c_i$, then the corresponding message m_i is magically copied from S's out-box into R's in-box, without actually running the decryption algorithm.

The correctness property of $\mathcal{E}$ ensures that this modification behaves exactly the same as the real implementation. Note that in this modification, any ciphertext that arrives at R's end that is not among the ciphertexts previously generated by S will be decrypted as usual using sk.

- Second, we modify the implementation of S's out-box, replacing the encryption of m_i with the encryption of $dummy_i$. The implementation of R's in-box remains as in the first modification.

  Here is where we use the CCA security property: if the attacker could distinguish the second modification from the first, we could use the attacker to break the CCA security of $\mathcal{E}$.

Since the second modification is identical to the ideal implementation, we see that the real and ideal implementations are indistinguishable from the adversary's point of view.

Just as in Section 9.3, we have ignored the possibility that the c_i's generated by S are not unique. Certainly, if we are going to view the c_i's as handles in the ideal implementation, uniqueness would seem to be an essential property. Just as in the symmetric case, CPA security (which is implied by CCA security) guarantees that the c_i's are unique with overwhelming probability (the reader can verify that the result of Exercise 5.12 holds in the public-key setting as well).

12.3 CCA-secure encryption from trapdoor function schemes

We now turn to constructing CCA-secure public-key encryption schemes. We begin with a construction from a general trapdoor function scheme satisfying certain properties. We use this to obtain a CCA-secure system from RSA. Later, in Section 12.6, we will show how to construct suitable trapdoor functions (in the random oracle model) from arbitrary, CPA-secure public-key encryption schemes. Using the result in this section, all these trapdoor functions give us CCA-secure encryption schemes.

Consider again the public-key encryption scheme $\mathcal{E}_{\text{TDF}} = (G, E, D)$ discussed in Section 11.4, which is based on an arbitrary trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$. Let us briefly recall this scheme: it makes use of a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and a hash function $H : \mathcal{X} \to \mathcal{K}$, which we model as a random oracle. The message space for $\mathcal{E}_{\text{TDF}}$ is $\mathcal{M}$ and the ciphertext space is $\mathcal{Y} \times \mathcal{C}$. The key generation algorithm for $\mathcal{E}_{\text{TDF}}$ is the same as the key generation algorithm for $\mathcal{T}$, and encryption and decryption work as follows:

```
$E(pk, m)$  :=  $x \xleftarrow{R} \mathcal{X},\ y \leftarrow F(pk, x),\ k \leftarrow H(x),\ c \xleftarrow{R} E_{\text{s}}(k, m)$
                output $(y, c)$;
$D(sk,\ (y, c)\ )$  :=  $x \leftarrow I(sk, y),\ k \leftarrow H(x),\ m \leftarrow D_{\text{s}}(k, c)$
                output $m$.
```

If $\mathcal{X} \neq \mathcal{Y}$, that is, if $\mathcal{T}$ is not a trapdoor permutation scheme, we have to modify the scheme slightly to get a scheme that is CCA secure. Basically, we modify the decryption algorithm to explicitly check that the given value $y \in \mathcal{Y}$ is actually in the image of $F(pk, \cdot)$. So the scheme we will analyze is $\mathcal{E}'_{\text{TDF}} = (G, E, D')$, where

```
$D'(sk,\ (y, c)\ )$  :=  $x \leftarrow I(sk, y)$
                  if $F(pk, x) = y$
                      then $k \leftarrow H(x),\ m \leftarrow D_{\text{s}}(k, c)$
                      else  $m \leftarrow$ reject
                  output $m$.
```

We will prove that $\mathcal{E}'_{\text{TDF}}$ is CCA secure if we model H as a random oracle, under appropriate assumptions. The first assumption we will make is that $\mathcal{E}_{\text{s}}$ is 1CCA secure (see Section 9.6). We also have to assume that $\mathcal{T}$ is one-way. However, when $\mathcal{X} \neq \mathcal{Y}$, we need a somewhat stronger assumption: that $\mathcal{T}$ is one-way even given access to an "image oracle". Essentially, this means that given pk and $y = F(pk, x)$ for randomly chosen $x \in \mathcal{X}$, it is hard to compute x, even given access to an oracle that will answer arbitrary questions of the form "does a given $\hat{y} \in \mathcal{Y}$ lie in the image of $F(pk, \cdot)$?". We formalize this notion by giving an attack game that is similar to Attack Game 10.2, but where the adversary has access to an image oracle.

Attack Game 12.2 (One-way trapdoor function scheme even with image oracle). For a given trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \mathcal{Y})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger computes

$$(pk, sk) \xleftarrow{\text{R}} G(), \quad x \xleftarrow{\text{R}} \mathcal{X}, \quad y \leftarrow F(pk, x)$$

  and sends (pk, y) to the adversary.

- The adversary makes a series of *image oracle queries* to the challenger. Each such query is of the form $\hat{y} \in \mathcal{Y}$, to which the challenger replies "yes" if $F(pk, I(sk, \hat{y})) = \hat{y}$, and "no" otherwise.

- The adversary outputs $\hat{x} \in \mathcal{X}$.

We define the adversary's advantage in inverting $\mathcal{T}$ given access to an image oracle, denoted $\text{IOW}\mathsf{adv}[\mathcal{A}, \mathcal{T}]$, to be the probability that $\hat{x} = x$. □

Definition 12.3. *We say that a trapdoor function scheme $\mathcal{T}$ is* ***one way given an image oracle*** *if for all efficient adversaries $\mathcal{A}$, the quantity $\text{IOW}\mathsf{adv}[\mathcal{A}, \mathcal{T}]$ is negligible.*

In Exercise 12.15 we show that (in the random oracle model) every one way trapdoor function scheme can be easily converted into one that is one way given an image oracle.

The next theorem proves the CCA security of $\mathcal{E}'_{\text{TDF}}$, assuming $\mathcal{T}$ is one-way given an image oracle, $\mathcal{E}_{\text{s}}$ is 1CCA secure (see Definition 9.6), and H is modeled as a random oracle. In Exercise 12.14 we explore an alternative analysis of this scheme under different assumptions.

In proving this theorem, we just prove that $\mathcal{E}'_{\text{TDF}}$ is 1CCA secure (see Definition 12.2). By virtue of Theorem 12.1, this is sufficient. Recall that in the random oracle model (see Section 8.10), the function H is modeled as a random function $\mathcal{O}$ chosen at random from the set of all functions $\text{Funs}[\mathcal{X}, \mathcal{K}]$. This means that in the random oracle version of the 1CCA attack game, the challenger chooses $\mathcal{O}$ at random. In any computation where the challenger would normally evaluate H, it evaluates $\mathcal{O}$ instead. In addition, the adversary is allowed to ask the challenger for the value of the function $\mathcal{O}$ at any point of its choosing. The adversary may make any number of such "random oracle queries" at any time of its choosing, arbitrarily interleaved with its usual encryption and decryption queries. We use $1\text{CCA}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}'_{\text{TDF}}]$ to denote $\mathcal{A}$'s advantage against $\mathcal{E}'_{\text{TDF}}$ in the random oracle version of the 1CCA attack game.

Theorem 12.2. *Assume $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle. If $\mathcal{T}$ is one-way given an image oracle, and $\mathcal{E}_{\text{s}}$ is 1CCA secure, then $\mathcal{E}'_{\text{TDF}}$ is CCA secure.*

In particular, for every 1CCA adversary $\mathcal{A}$ that attacks $\mathcal{E}'_{\mathrm{TDF}}$ as in the random oracle version of Definition 12.2, there exist an inverting adversary $\mathcal{B}_{\mathrm{iow}}$ that breaks the one-wayness assumption for $\mathcal{T}$ as in Attack Game 12.2, and a 1CCA adversary $\mathcal{B}_{\mathrm{s}}$ that attacks $\mathcal{E}_{\mathrm{s}}$ as in Definition 9.6, where $\mathcal{B}_{\mathrm{iow}}$ and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{1CCA^{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}'_{\mathrm{TDF}}] \leq 2 \cdot \mathrm{IOW}\mathsf{adv}[\mathcal{B}_{\mathrm{iow}}, \mathcal{T}] + \mathrm{1CCA}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{12.3}$$

For applications of this theorem in the sequel, we record here some further technical properties that the adversary $\mathcal{B}_{\mathrm{iow}}$ satisfies.

If $\mathcal{A}$ makes at most Q_{d} decryption queries, then $\mathcal{B}_{\mathrm{iow}}$ makes at most Q_{d} image-oracle queries. Also, the only dependence of $\mathcal{B}_{\mathrm{iow}}$ on the function F is that it invokes $F(pk, \cdot)$ as a subroutine, at most Q_{ro} times, where Q_{ro} is a bound on the number of random-oracle queries made by $\mathcal{A}$; moreover, if $\mathcal{B}_{\mathrm{iow}}$ produces an output $\hat{x}$, it always evaluates $F(pk, \cdot)$ at $\hat{x}$.

Proof idea. The crux of the proof is to show that the adversary's decryption queries do not help him in any significant way. What this means technically is that we have to modify the challenger so that it can compute responses to the decryption queries *without using the secret key sk*. The trick to achieve this is to exploit the fact that our challenger is in charge of implementing the random oracle, maintaining a table of all input/output pairs. Assume the target ciphertext (i.e., the one resulting from the encryption query) is (y, c), where $y = F(pk, x)$, and suppose the challenger is given a decryption query $(\hat{y}, \hat{c})$, where $y \neq \hat{y} = F(pk, \hat{x})$.

- If the adversary has previously queried the random oracle at $\hat{x}$, and if $\hat{k}$ was the output of the random oracle at $\hat{x}$, then the challenger simply decrypts $\hat{c}$ using $\hat{k}$.
- Otherwise, if the adversary has not made such a random oracle query, then the challenger does not know the correct value of the symmetric key — but neither does the adversary. The challenger is then free to choose a key $\hat{k}$ at random, and decrypt $\hat{c}$ using this key; however, the challenger must do some extra book-keeping to ensure consistency, so that if the adversary ever queries the random oracle in the future at the point $\hat{x}$, then the challenger "back-patches" the random oracle, so that its output at $\hat{x}$ is set to $\hat{k}$.

We also have to deal with decryption queries of the form $(y, \hat{c})$, where $\hat{c} \neq c$. Intuitively, under the one-wayness assumption for $\mathcal{T}$, the adversary will never query the random oracle at x, and so from the adversary's point of view, the symmetric key k used in the encryption query, and used in decryption queries of the form $(y, \hat{c})$, is as good as random, and so CCA security for $\mathcal{E}'_{\mathrm{TDF}}$ follows immediately from 1CCA security for $\mathcal{E}_{\mathrm{s}}$.

In the above, we have ignored ciphertext queries of the form $(\hat{y}, \hat{c})$ where $\hat{y}$ has no preimage under $F(pk, \cdot)$. The real decryption algorithm rejects such queries. This is why we need to assume $\mathcal{T}$ is one-way given an image oracle — in the reduction, we need this image oracle to reject ciphertexts of this form. □

Proof. It is convenient to prove the theorem using the bit-guessing versions of the 1CCA attack games. We prove:

$$\mathrm{1CCA^{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'_{\mathrm{TDF}}] \leq \mathrm{IOW}\mathsf{adv}[\mathcal{B}_{\mathrm{iow}}, \mathcal{T}] + \mathrm{1CCA}\mathsf{adv}^*[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{12.4}$$

Then (12.3) follows by (12.2) and (9.2).

As usual, we define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the bit-guessing version of the 1CCA attack game with respect to $\mathcal{E}'_{\mathrm{TDF}}$. We then modify the challenger to obtain Game 1. In each game, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. Also, for $j = 0, 1$, we define W_j to be the event that $\hat{b} = b$ in Game j.

Game 0. The logic of the challenger is shown in Fig. 12.1. The challenger has to respond to random oracle queries, in addition to encryption and decryption queries. The adversary can make any number of random oracle queries, and any number of decryption queries, but at most one encryption query. Recall that in addition to direct access to the random oracle via explicit random oracle queries, the adversary also has indirect access to the random oracle via the encryption and decryption queries, where the challenger also makes use of the random oracle. In the initialization step, the challenger computes $(pk, sk) \xleftarrow{\text{R}} G()$; we also have our challenger make those computations associated with the encryption query that can be done without yet knowing the challenge plaintext. To facilitate the proof, we want our challenger to use the secret key sk as little as possible in processing decryption queries. This will motivate a somewhat nontrivial strategy for implementing the decryption and random oracle queries.

As usual, we will make use of an associative array to implement the random oracle. In the proof of Theorem 11.2, which analyzed the semantic security of $\mathcal{E}_{\mathrm{TDF}}$, we did this quite naturally by using an associative array $Map : \mathcal{X} \to \mathcal{K}$. We could do the same thing here, but because we want our challenger to use the secret key as little as possible, we adopt a different strategy. Namely, we will represent the random oracle using associative array $Map' : \mathcal{Y} \to \mathcal{K}$, with the convention that if the value of the oracle at $\hat{x} \in \mathcal{X}$ is equal to $\hat{k} \in \mathcal{K}$, then $Map'[\hat{y}] = \hat{k}$, where $\hat{y} = F(pk, \hat{x})$. We will also make use of an associative array $Pre : \mathcal{Y} \to \mathcal{X}$ that is used to track explicit random oracle queries made by the adversary: if $Pre[\hat{y}] = \hat{x}$, this means that the adversary queried the oracle at the point $\hat{x}$, and $\hat{y} = F(pk, \hat{x})$. Note that Map' will in general be defined at points other than those at which Pre is defined, since the challenger also makes random oracle queries.

In preparation for the encryption query, in the initialization step, the challenger precomputes $x \xleftarrow{\text{R}} \mathcal{X}$, $y \leftarrow F(pk, x)$, $k \xleftarrow{\text{R}} \mathcal{K}$. It also sets $Map'[y] \leftarrow k$, which means that the value of the random oracle at x is equal to k. Also note that in the initialization step, the challenger sets $c \leftarrow \perp$, and in processing the encryption query, overwrites c with a ciphertext in $\mathcal{C}$. Thus, decryption queries processed while $c = \perp$ are phase 1 queries, while those processed while $c \neq \perp$ are phase 2 queries.

To process a decryption query $(\hat{y}, \hat{c})$, making minimal use of the secret key, the challenger uses the following strategy.

- If $\hat{y} = y$, the challenger just uses the prepared key k directly to decrypt $\hat{c}$.

- Otherwise, the challenger checks if Map' is defined at the point $\hat{y}$, and if not, it assigns to $Map'[\hat{y}]$ a random value $\hat{k}$. If $\hat{y}$ has a preimage $\hat{x}$ and Map' was not defined at $\hat{y}$, this means that neither the adversary nor the challenger previously queried the random oracle at $\hat{x}$, and so this new random value $\hat{k}$ represents the value or the random oracle at $\hat{x}$; in particular, if the adversary later queries the random oracle at the point $\hat{x}$, this same value of $\hat{k}$ will be used. If $\hat{y}$ has no preimage, then assigning $Map'[\hat{y}]$ a random value $\hat{k}$ has no real effect — it just streamlines the logic a bit.

- Next, the challenger tests if $\hat{y}$ is in the image of $F(pk, \cdot)$. If $\hat{y}$ is not in the image, the challenger just rejects the ciphertext. In Fig. 12.1, we implement this by invoking the function

```
    initialization:
        $(pk, sk) \xleftarrow{\mathrm{R}} G()$, $x \xleftarrow{\mathrm{R}} \mathcal{X}$, $y \leftarrow F(pk, x)$
        $c \leftarrow \bot$
        initialize empty associative arrays $Pre : \mathcal{Y} \to \mathcal{X}$ and $Map' : \mathcal{Y} \to \mathcal{K}$
        $k \xleftarrow{\mathrm{R}} \mathcal{K}$, $b \xleftarrow{\mathrm{R}} \{0, 1\}$
(1)     $Map'[y] \leftarrow k$
        send the public key $pk$ to $\mathcal{A}$;

    upon receiving an encryption query $(m_0, m_1) \in \mathcal{M}^2$:
        $c \xleftarrow{\mathrm{R}} E_{\mathrm{s}}(k, m_b)$, send $(y, c)$ to $\mathcal{A}$;

    upon receiving a decryption query $(\hat{y}, \hat{c}) \in \mathcal{X} \times \mathcal{C}$, where $(\hat{y}, \hat{c}) \neq (y, c)$:
        if $\hat{y} = y$ then
            $\hat{m} \leftarrow D_{\mathrm{s}}(k, \hat{c})$
        else
            if $\hat{y} \notin \mathrm{Domain}(Map')$ then $Map'[\hat{y}] \xleftarrow{\mathrm{R}} \mathcal{K}$
(2)         if $Image(pk, sk, \hat{y}) =$ “no”    // i.e., $\hat{y}$ is not in the image of $F(pk, \cdot)$
                then $\hat{m} \leftarrow$ reject
                else  $\hat{k} \leftarrow Map'[\hat{y}]$, $\hat{m} \leftarrow D_{\mathrm{s}}(\hat{k}, \hat{c})$
        send $\hat{m}$ to $\mathcal{A}$;

    upon receiving a random oracle query $\hat{x} \in \mathcal{X}$:
        $\hat{y} \leftarrow F(pk, \hat{x})$, $Pre[\hat{y}] \leftarrow \hat{x}$
        if $\hat{y} \notin \mathrm{Domain}(Map')$ then $Map'[\hat{y}] \xleftarrow{\mathrm{R}} \mathcal{K}$
        send $Map'[\hat{y}]$ to $\mathcal{A}$
```

Figure 12.1: Game 0 challenger in the proof of Theorem 12.2

$\mathit{Image}(pk, sk, \hat{y})$. For now, we can think of *Image* as being implemented as follows:

$$\mathit{Image}(pk, sk, \hat{y}) \quad := \quad \left\{ \text{ return “yes” if } F(pk, I(sk, \hat{y})) = \hat{y} \text{ and “no” otherwise } \right\}.$$

This is the only place where our challenger makes use of the secret key.

- Finally, if $\hat{y}$ is in the range of $F(pk, \cdot)$, the challenger simply decrypts $\hat{c}$ directly using the symmetric key $\hat{k} = Map'[\hat{y}]$, which at this point is guaranteed to be defined, and represents the value of the random oracle at the preimage $\hat{x}$ of $\hat{y}$. Note that our challenger can do this, *without actually knowing $\hat{x}$.* This is the crux of the proof.

Despite this somewhat involved bookkeeping, it should be clear that our challenger behaves *exactly* as in the usual attack game.

Game 1. This game is precisely the same as Game 0, except that we delete the line marked (1) in Fig. 12.1.

Let Z be the event that the adversary queries the random oracle at x in Game 1. Clearly, Games 0 and 1 proceed identically unless Z occurs, and so by the Difference Lemma, we have

$$|\Pr[W_1] - \Pr[W_0]| \leq \Pr[Z]. \tag{12.5}$$

If event Z happens, then at the end of Game 1, we have $Pre[y] = x$. What we want to do, therefore, is use $\mathcal{A}$ to build an efficient adversary $\mathcal{B}_{\text{iow}}$ that breaks the one-wayness assumption for $\mathcal{T}$ with an advantage equal to $\Pr[Z]$, with the help of an image oracle. The logic of $\mathcal{B}_{\text{iow}}$ is very straightforward. Basically, after obtaining the public key pk and $y \in \mathcal{Y}$ from its challenger in Attack Game 12.2, $\mathcal{B}_{\text{iow}}$ plays the role of challenger to $\mathcal{A}$ as in Game 1. The value of x is never explicitly used in that game (other than to compute y), and the value of the secret key sk is not used, except in the evaluation of the *Image* function, and for this, $\mathcal{B}_{\text{iow}}$ can use the image oracle provided to it in Attack Game 12.2. At the end of the game, if $y \in \text{Domain}(Pre)$, then $\mathcal{B}_{\text{iow}}$ outputs $x = Pre[y]$. It should be clear, by construction, that

$$\Pr[Z] = \text{IOW}\mathsf{adv}[\mathcal{B}_{\text{iow}}, \mathcal{T}]. \tag{12.6}$$

Finally, note that in Game 1, the key k is only used to encrypt the challenge plaintext, and to process decryption queries of the form $(y, \hat{c})$, where $\hat{c} \neq c$. As such, the adversary is essentially just playing the 1CCA attack game against $\mathcal{E}_{\text{s}}$ at this point. More precisely, we can easily derive an efficient 1CCA adversary $\mathcal{B}_{\text{s}}$ based on Game 1 that uses $\mathcal{A}$ as a subroutine, such that

$$|\Pr[W_1] - 1/2| = \text{1CCA}\mathsf{adv}^*[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}]. \tag{12.7}$$

This adversary $\mathcal{B}_{\text{s}}$ generates (pk, sk) itself and uses sk to answer queries from $\mathcal{A}$.

Combining (12.5), (12.6) and (12.7), we obtain (12.4). That completes the proof of the theorem. □

12.3.1 Instantiating $\mathcal{E}'_{\text{TDF}}$ with RSA

Suppose we instantiate $\mathcal{E}'_{\text{TDF}}$ using RSA just as we did in Section 11.4.1. The underlying trapdoor function is actually a permutation on $\mathbb{Z}_n$. This implies two things. First, we can omit the check in the decryption algorithm that y is in the image of the trapdoor function, and so we end up with exactly the same scheme $\mathcal{E}_{\text{RSA}}$ as was presented in Section 11.4.1. Second, the implementation of the image oracle in Attack Game 12.2 is trivial to implement, and so we end up back with Attack Game 10.2. Theorem 12.2 specializes as follows:

Theorem 12.3. *Assume $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle. If the RSA assumption holds for parameters (ℓ, e), and $\mathcal{E}_{\text{s}}$ is 1CCA secure, then $\mathcal{E}_{\text{RSA}}$ is CCA secure.*

> *In particular, for every 1CCA adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\text{RSA}}$ as in the random oracle version of Definition 12.2, there exist an RSA adversary $\mathcal{B}_{\text{rsa}}$ that breaks the RSA assumption for (ℓ, e) as in Attack Game 10.3, and a 1CCA adversary $\mathcal{B}_{\text{s}}$ that attacks $\mathcal{E}_{\text{s}}$ as in Definition 9.6, where $\mathcal{B}_{\text{rsa}}$ and $\mathcal{B}_{\text{s}}$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{1CCA}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{RSA}}] \leq 2 \cdot \text{RSA}\mathsf{adv}[\mathcal{B}_{\text{rsa}}, \ell, e] + \text{1CCA}\mathsf{adv}[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}].$$

12.4 CCA-secure ElGamal encryption

We saw that the basic RSA encryption scheme $\mathcal{E}_{\text{RSA}}$ could be shown to be CCA secure in the random oracle model under the RSA assumption (and assuming the underlying symmetric cipher was 1CCA secure). It is natural to ask whether the basic ElGamal encryption scheme $\mathcal{E}_{\text{EG}}$, discussed in Section 11.5, is CCA secure in the random oracle model, under the CDH assumption. Unfortunately, this is not the case: it turns out that a slightly stronger assumption than the CDH assumption is both necessary and sufficient to prove the security of $\mathcal{E}_{\rm EG}$.

Recall the basic ElGamal encryption scheme, $\mathcal{E}_{\rm EG} = (G, E, D)$, introduced in Section 11.5. It is defined in terms of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$, a symmetric cipher $\mathcal{E}_{\rm s} = (E_{\rm s}, D_{\rm s})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and a hash function $H : \mathbb{G}^2 \to \mathcal{K}$. The message space of $\mathcal{E}_{\rm EG}$ is $\mathcal{M}$ and the ciphertext space is $\mathbb{G} \times \mathcal{C}$. Public keys are of the form $u \in \mathbb{G}$ and secret keys are of the form $\alpha \in \mathbb{Z}_q$. The algorithms G, E, and D are defined as follows:

$$\begin{aligned}
G() \quad &:= \quad \alpha \xleftarrow{\rm R} \mathbb{Z}_q, \;\; u \leftarrow g^\alpha, \;\; pk \leftarrow u, \;\; sk \leftarrow \alpha \\
& \text{output } (pk, sk); \\
E(u, m) \quad &:= \quad \beta \xleftarrow{\rm R} \mathbb{Z}_q, \;\; v \leftarrow g^\beta, \;\; w \leftarrow u^\beta, \;\; k \leftarrow H(v, w), \;\; c \xleftarrow{\rm R} E_{\rm s}(k, m) \\
& \text{output } (v, c); \\
D(\alpha, \; (v, c) \;) \quad &:= \quad w \leftarrow v^\alpha, \;\; k \leftarrow H(v, w), \;\; m \leftarrow D_{\rm s}(k, c) \\
& \text{output } m.
\end{aligned}$$

To see why the CDH assumption by itself is not sufficient to establish the security of $\mathcal{E}_{\rm EG}$ against chosen ciphertext attack, suppose the public key is $u = g^\alpha$. Now, suppose an adversary selects group elements $\hat{v}$ and $\hat{w}$ in some arbitrary way, and computes $\hat{k} \leftarrow H(\hat{v}, \hat{w})$ and $\hat{c} \xleftarrow{\rm R} E_{\rm s}(\hat{k}, \hat{m})$ for some arbitrary message $\hat{m}$. Further, suppose the adversary can obtain the decryption m^* of the ciphertext $(\hat{v}, \hat{c})$. Now, it is very likely that $\hat{m} = m^*$ if and only if $\hat{w} = \hat{v}^\alpha$, or in other words, if and only if $(u, \hat{v}, \hat{w})$ is a DH-triple. Thus, in the chosen ciphertext attack game, decryption queries can be effectively used by the adversary to answer questions of the form "is $(u, \hat{v}, \hat{w})$ a DH-triple?" for group elements $\hat{v}$ and $\hat{w}$ of the adversary's choosing. In general, the adversary would not be able to efficiently answer such questions on his own (this is the DDH assumption), and so these decryption queries may potentially leak some information about the secret key α. Based on the current state of our knowledge, this leakage does not seem to compromise the security of the scheme; however, we do need to state this as an explicit assumption.

Intuitively, the **interactive CDH assumption** states that given a random instance (g^α, g^β) of the DH problem, it is hard to compute $g^{\alpha\beta}$, even when given access to a "DH-decision oracle" that recognizes DH-triples of the form $(g^\alpha, \cdot, \cdot)$. More formally, this assumption is defined in terms of the following attack game.

Attack Game 12.3 (Interactive Computational Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger computes

  $$\alpha, \beta \xleftarrow{\rm R} \mathbb{Z}_q, \;\; u \leftarrow g^\alpha, \;\; v \leftarrow g^\beta, \;\; w \leftarrow g^{\alpha\beta}$$

  and gives (u, v) to the adversary.

- The adversary makes a sequence of *DH-decision oracle queries* to the challenger. Each query is of the form $(\tilde{v}, \tilde{w}) \in \mathbb{G}^2$. Upon receiving such a query, the challenger tests if $\tilde{v}^\alpha = \tilde{w}$; if so, he sends "yes" to the adversary, and otherwise, sends "no" to the adversary.

- Finally, the adversary outputs some $\hat{w} \in \mathbb{G}$.

We define $\mathcal{A}$'s **advantage in solving the interactive computational Diffie-Hellman problem**, denoted ICDH$\mathsf{adv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{w} = w$. □

We stress that in the above attack game, the adversary can ask the challenger for help in determining whether certain triples are DH-triples, but only triples of the form $(u, \cdot, \cdot)$, where u is generated by the challenger.

Definition 12.4 (Interactive Computational Diffie-Hellman assumption). *We say that the* ***interactive computational Diffie-Hellman (ICDH)*** *assumption holds for* $\mathbb{G}$ *if for all efficient adversaries* $\mathcal{A}$ *the quantity* $\mathrm{ICDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ *is negligible.*

By the above discussion, we see (at least heuristically) that the ICDH assumption is necessary to establish the CCA security of $\mathcal{E}_{\mathrm{EG}}$. Conversely, one can prove that $\mathcal{E}_{\mathrm{EG}}$ is CCA secure in the random oracle model under the ICDH assumption (and assuming also that $\mathcal{E}_{\mathrm{s}}$ is 1CCA secure).

Remark 12.1 (Group membership verification). To prove the CCA security of $\mathcal{E}_{\mathrm{EG}}$, we must insist that given a ciphertext (v, c), the decryption algorithm verifies that $v \in \mathbb{G}$. For example, if $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$ of order q, where p is a large prime, the decryption algorithm should not only check that $v \in \mathbb{Z}_p^*$ (which means, as an integer, it is in the range $[1, p)$), but should also check that $v^q = 1$ (which costs another exponentiation). Without this check, the scheme may be vulnerable to a CCA attack (see Exercise 12.3). Later, in Chapter 15, we will see other cryptographically useful groups (elliptic curves) where group membership verification can be much less expensive. □

Remark 12.2 (Hashing (v, w) ***vs hashing only*** w***).*** Analogous to Remark 11.1, the variant $\mathcal{E}'_{\mathrm{EG}}$, which hashes only w instead of (v, w), is also CCA secure under the same assumptions; however, the security reduction for $\mathcal{E}_{\mathrm{EG}}$ is simpler and more efficient. □

Theorem 12.4. *Assume* $H : \mathbb{G}^2 \to \mathcal{K}$ *is modeled as a random oracle. If the ICDH assumption holds for* $\mathbb{G}$*, and* $\mathcal{E}_{\mathrm{s}}$ *is 1CCA secure, then* $\mathcal{E}_{\mathrm{EG}}$ *is CCA secure.*

In particular, for every 1CCA adversary $\mathcal{A}$ *that attacks* $\mathcal{E}_{\mathrm{EG}}$ *as in the random oracle version of Definition 12.2, there exist an ICDH adversary* $\mathcal{B}_{\mathrm{icdh}}$ *for* $\mathbb{G}$ *as in Attack Game 12.3, and a 1CCA adversary* $\mathcal{B}_{\mathrm{s}}$ *that attacks* $\mathcal{E}_{\mathrm{s}}$ *as in Definition 9.6, where* $\mathcal{B}_{\mathrm{icdh}}$ *and* $\mathcal{B}_{\mathrm{s}}$ *are elementary wrappers around* $\mathcal{A}$*, such that*

$$\mathrm{1CCA^{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{EG}}] \le 2 \cdot \mathrm{ICDH}\mathsf{adv}[\mathcal{B}_{\mathrm{icdh}}, \mathbb{G}] + \mathrm{1CCA}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{12.8}$$

In addition, the number of DH-decision oracle queries made by $\mathcal{B}_{\mathrm{icdh}}$ *is bounded by the number of random oracle queries made by* $\mathcal{A}$*.*

Proof. The basic structure of the proof is very similar to that of Theorem 12.2. As in that proof, it is convenient to use the bit-guessing versions of the 1CCA attack games. We prove

$$\mathrm{1CCA^{ro}}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\mathrm{EG}}] \le \mathrm{ICDH}\mathsf{adv}[\mathcal{B}_{\mathrm{icdh}}, \mathbb{G}] + \mathrm{1CCA}\mathsf{adv}^*[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{12.9}$$

Then (12.8) follows by (12.2) and (9.2).

We define Games 0 and 1. Game 0 is the bit-guessing version of Attack Game 12.1 played by $\mathcal{A}$ with respect to $\mathcal{E}_{\mathrm{EG}}$. In each game, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. For $j = 0, 1$, we define W_j to be the event that $\hat{b} = b$ in Game j.

Game 0. The logic of the challenger is shown in Fig. 12.2. The adversary can make any number of random oracle queries, and any number of decryption queries, but at most one encryption query. As usual, in addition to direct access the random oracle via explicit random oracle queries, the adversary also has indirect access to the random oracle via the encryption and decryption queries, where the challenger also makes use of the random oracle.

In the initialization step, the challenger computes the secret key $\alpha \in \mathbb{Z}_q$ and the public key $u = g^{\alpha}$; it also makes those computations associated with the encryption query that can be done without yet knowing the challenge plaintext. As in the proof of Theorem 12.2, we want our challenger to use the secret key α as little as possible in processing decryption queries, and again, we use a somewhat nontrivial strategy for implementing the decryption and random oracle queries. Nevertheless, despite the significant superficial differences, this implementation will be logically equivalent to the actual attack game.

As usual, we will implement the random oracle using an associative array $Map : \mathbb{G}^2 \to \mathcal{K}$. However, we will also make use of an auxiliary associative array $Map' : \mathbb{G} \to \mathcal{K}$. The convention is that if $(u, \hat{v}, \hat{w})$ is a DH-triple, and the value of the random oracle at the point $(\hat{v}, \hat{w})$ is $\hat{k}$, then $Map[\hat{v}, \hat{w}] = Map'[\hat{v}] = \hat{k}$. However, in processing a decryption query $(\hat{v}, \hat{c})$, we may speculatively assign a random value $\hat{k}$ to $Map'[\hat{v}]$, and then later, if the adversary queries the random oracle at the point $(\hat{v}, \hat{w})$, where $(u, \hat{v}, \hat{w})$ is a DH-triple, we assign the value $\hat{k}$ to $Map[\hat{v}, \hat{w}]$, in order to maintain consistency.

Now for more details. In preparation for the encryption query, in the initialization step, the challenger precomputes $\beta \xleftarrow{\text{R}} \mathbb{Z}_q$, $v \leftarrow g^{\beta}$, $w \leftarrow g^{\alpha\beta}$, $k \xleftarrow{\text{R}} \mathcal{K}$. It also sets $Map[v, w]$ and $Map'[v]$ to k, which means that the value of the random oracle at (v, w) is equal to k. Also note that in the initialization step, the challenger sets $c \leftarrow \bot$, and in processing the encryption query, overwrites c with a ciphertext in $\mathcal{C}$. Thus, decryption queries processed while $c = \bot$ are phase 1 queries, while those processed while $c \neq \bot$ are phase 2 queries.

Processing random oracle queries. When processing a random oracle query $(\hat{v}, \hat{w})$, if $Map[\hat{v}, \hat{w}]$ has not yet been defined, the challenger proceeds as follows.

- First, it tests if $(u, \hat{v}, \hat{w})$ is a DH-triple. In Fig. 12.2, we implement this by invoking the function $DHP(\alpha, \hat{v}, \hat{w})$. For now, we can think of DHP as being implemented as follows:

  $$DHP(\alpha, \hat{v}, \hat{w}) \quad := \quad \hat{v}^{\alpha} = \hat{w}.$$

  This is the only place where our challenger makes use of the secret key.

- If $(u, \hat{v}, \hat{w})$ is a DH-triple, the challenger sets $Map'[\hat{v}]$ to a random value, if it is not already defined, and then sets $Map[\hat{v}, \hat{w}] \leftarrow Map'[\hat{v}]$. It also sets $Sol[\hat{v}] \leftarrow \hat{w}$, where $Sol : \mathbb{G} \to \mathbb{G}$ is another associative array. The idea is that Sol records solutions to Diffie-Hellman instances $(u, \hat{v})$ that are discovered while processing random oracle queries.

- If $(u, \hat{v}, \hat{w})$ is not a DH-triple, then the challenger just sets $Map[\hat{v}, \hat{w}]$ to a random value.

The result of the random oracle query is always $Map[\hat{v}, \hat{w}]$.

Processing decryption queries. In processing a decryption query $(\hat{v}, \hat{c})$, the challenger proceeds as follows.

- If $\hat{v} = v$, the challenger just uses the prepared key k directly to decrypt $\hat{c}$.

- Otherwise, the challenger checks if Map' is defined at the point $\hat{v}$, and if not, it assigns to $Map'[\hat{v}]$ a random value. It then uses the value $\hat{k} = Map'[\hat{v}]$ directly to decrypt $\hat{c}$. Observe that our challenger performs the decryption without using the solution $\hat{w}$ to the instance

```
initialization:
    $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$, $u \leftarrow g^{\alpha}$, $v \leftarrow g^{\beta}$, $w \leftarrow g^{\alpha\beta}$
    $k \xleftarrow{R} \mathcal{K}$, $b \xleftarrow{R} \{0,1\}$
    $c \leftarrow \perp$
    initialize three empty associative arrays
        $Map : \mathbb{G}^2 \to \mathcal{K}$,  $Map' : \mathbb{G} \to \mathcal{K}$, and  $Sol : \mathbb{G} \to \mathbb{G}$
(1) $Map[v, w] \leftarrow k$, $Map'[v] \leftarrow k$
    send the public key $u$ to $\mathcal{A}$;

upon receiving an encryption query $(m_0, m_1) \in \mathcal{M}^2$:
    $c \xleftarrow{R} E_s(k, m_b)$, send $(v, c)$ to $\mathcal{A}$;

upon receiving a decryption query $(\hat{v}, \hat{c}) \in \mathbb{G} \times \mathcal{C}$, where $(\hat{v}, \hat{c}) \neq (v, c)$:
    if $\hat{v} = v$ then
        $\hat{m} \leftarrow D_s(k, \hat{c})$
    else
        if $\hat{v} \notin$ Domain($Map'$) then $Map'[\hat{v}] \xleftarrow{R} \mathcal{K}$
        $\hat{k} \leftarrow Map'[\hat{v}]$, $\hat{m} \leftarrow D_s(\hat{k}, \hat{c})$
    send $\hat{m}$ to $\mathcal{A}$;

upon receiving a random oracle query $(\hat{v}, \hat{w}) \in \mathbb{G}^2$:
    if $(\hat{v}, \hat{w}) \notin$ Domain($Map$) then
        if DHP$(\alpha, \hat{v}, \hat{w})$ then
            if $\hat{v} \notin$ Domain($Map'$) then $Map'[\hat{v}] \xleftarrow{R} \mathcal{K}$
            $Map[\hat{v}, \hat{w}] \leftarrow Map'[\hat{v}]$, $Sol[\hat{v}] \leftarrow \hat{w}$
        else
            $Map[\hat{v}, \hat{w}] \xleftarrow{R} \mathcal{K}$
    send $Map[\hat{v}, \hat{w}]$ to $\mathcal{A}$
```

Figure 12.2: Game 0 challenger in the proof of Theorem 12.4

$(u, \hat{v})$ of the CDH problem. However, if the adversary queries the random oracle at the point $(\hat{v}, \hat{w})$, the adversary will see the same value $\hat{k}$, and so consistency is maintained.

Hopefully, it is clear that our challenger behaves *exactly* as in the usual attack game, despite the more elaborate bookkeeping.

Game 1. This game is the same as Game 0, except that we delete line (1) in Fig. 12.2.

Let Z be the event that $\mathcal{A}$ queries the random oracle at (v, w) in Game 1. It is not hard to see that Games 0 and 1 proceed identically, unless Z occurs. By the Difference Lemma, we have

$$|\Pr[W_1] - \Pr[W_0]| \leq \Pr[Z]. \tag{12.10}$$

If event Z happens, then at the end of Game 1, we have $Sol[v] = w$. What we want to do, therefore, is use $\mathcal{A}$ to build an efficient adversary $\mathcal{B}_{\text{icdh}}$ that breaks the CDH assumption for $\mathbb{G}$, with the help of a DH-decision oracle, with an advantage equal to $\Pr[Z]$. The logic of $\mathcal{B}_{\text{icdh}}$ is very straightforward. Basically, after obtaining u and v from its challenger in Attack Game 12.3, $\mathcal{B}_{\text{icdh}}$

plays the role of challenger to $\mathcal{A}$ as in Game 1. Besides the computation of u, the value of α is never explicitly used in that game, other than in the evaluation of the *DHP* function, and for this, $\mathcal{B}_{\text{icdh}}$ can use the DH-decision oracle provided to it in Attack Game 12.3. At the end of the game, if $v \in \text{Domain}(Sol)$, then $\mathcal{B}_{\text{icdh}}$ outputs $w = Sol[v]$.

By construction, it is clear that

$$\Pr[Z] = \text{ICDH}\mathsf{adv}[\mathcal{B}_{\text{icdh}}, \mathbb{G}]. \tag{12.11}$$

Finally, note that in Game 1, the key k is only used to encrypt the challenge plaintext, and to process decryption queries of the form $(v, \hat{c})$, where $\hat{c} \neq c$. As such, the adversary is essentially just playing the 1CCA attack game against $\mathcal{E}_{\text{s}}$ at this point. More precisely, we can easily derive an efficient 1CCA adversary $\mathcal{B}_{\text{s}}$ based on Game 1 that uses $\mathcal{A}$ as a subroutine, such that

$$|\Pr[W_1] - 1/2| = 1\text{CCA}\mathsf{adv}^*[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}]. \tag{12.12}$$

We leave the details of $\mathcal{B}_{\text{s}}$ to the reader.

Combining (12.10), (12.11), and (12.12), we obtain (12.9). That completes the proof of the theorem. □

Discussion. We proved that $\mathcal{E}_{\text{EG}}$ is CCA-secure, in the random oracle model, under the ICDH assumption. Is the ICDH assumption reasonable? On the one hand, in Chapter 15 we will see groups $\mathbb{G}$ where the ICDH assumption is equivalent to the CDH assumption. In such groups there is no harm in assuming ICDH. On the other hand, the ElGamal system is most commonly implemented in groups where ICDH is not known to be equivalent to CDH. Is it reasonable to assume ICDH in such groups? Currently, we do not know of any group where CDH holds, but ICDH does not hold. As such, it appears to be a reasonable assumption to use when constructing cryptographic schemes. Later, in Section 12.6.2, we will see a variant of ElGamal encryption that is CCA-secure, in the random oracle model, under the normal CDH assumption. See also Exercise 12.31, where we develop a more modular analysis of $\mathcal{E}_{\text{EG}}$ based on a new assumption, called the interactive hash Diffie-Hellman (IHDH) assumption, which itself is implied by the ICDH assumption.

12.5 CCA security from DDH without random oracles

In Section 11.5.2, we proved that $\mathcal{E}_{\text{EG}}$ was semantically secure without relying on the random oracle model. Rather, we used the DDH assumption (among other assumptions). Unfortunately, it seems unlikely that we can ever hope to prove that $\mathcal{E}_{\text{EG}}$ is CCA secure without relying on random oracles.

In this section, we present a public key encryption scheme that can be proved CCA secure without relying on the random oracle heuristic. The scheme is based on the DDH assumption (as well as a few other standard assumptions). The scheme is a variant of one designed by Cramer and Shoup, and we call it $\mathcal{E}_{\text{CS}}$.

12.5.1 Universal projective hash functions

We introduce here the tool used in the design and analysis of $\mathcal{E}_{\text{CS}}$. Defining this tool in its full generality would take us too far afield. Rather, we give just an intuitive description of this tool in its general form, and instantiate it more rigorously in the specific form in which we will need it here.

The tool is called a **projective hash function**. It can perhaps be best understood as a form of *function delegation*. Suppose Alice has a secret function $f : \mathcal{Y} \to \mathcal{Z}$. She would like to delegate the ability to evaluate f to Bob — but not entirely. Specifically, she wants to give Bob just enough information about f so that he can evaluate f on a specific subset $\mathcal{L} \subseteq \mathcal{Y}$, but nowhere else. We denote by h the information about f that Alice gives to Bob. In our applications, $\mathcal{L}$ will always be the image of some function $\theta : \mathcal{X} \to \mathcal{Y}$, and to efficiently evaluate f at a point $y \in \mathcal{L}$, Bob will need $x \in \mathcal{X}$ such that $\theta(x) = y$, along with the auxiliary information h provided to him by Alice.

Such a scheme is called a *projective hash function*. Given the auxiliary information h, the behavior of f is completely defined on $\mathcal{L}$. However, we also require that h does not reveal any information about the behavior of f outside of $\mathcal{L}$. Somewhat more precisely, the requirement is that if f is chosen at random (from some family of functions), then for every $y \in \mathcal{Y} \setminus \mathcal{L}$, the values $f(y)$ and h are independent, with $f(y)$ uniformly distributed over $\mathcal{Z}$. If this additional requirement is satisfied, then we say this scheme is a **universal** projective hash function.

A concrete instantiation. We now give a concrete example of the above idea. Suppose $\mathbb{G}$ is a cyclic group of prime order q with generator $g \in \mathbb{G}$. Further, suppose $u \in \mathbb{G}$ is some fixed group element. The set $\mathcal{Y}$ above will consist of all pairs $(v, w) \in \mathbb{G}^2$, while the set $\mathcal{L} = \mathcal{L}_u$ will consist of those pairs (v, w) for which (u, v, w) is a DH-triple. Note that the set $\mathcal{L}_u$ is the image of the function

$$\begin{aligned} \theta : \mathbb{Z}_q &\to \mathbb{G}^2, \\ \beta &\mapsto (g^\beta, u^\beta). \end{aligned}$$

The function $f := f_{\sigma,\tau}$ is indexed by randomly chosen $\sigma, \tau \in \mathbb{Z}_q$, and is defined as follows:

$$\begin{aligned} f_{\sigma,\tau} : \qquad \mathbb{G}^2 &\to \mathbb{G}, \\ (v, w) &\mapsto v^\sigma w^\tau. \end{aligned} \tag{12.13}$$

The auxiliary information h that defines f on $\mathcal{L}_u$ is

$$h := f(g, u) = g^\sigma u^\tau. \tag{12.14}$$

So if Alice chooses $\sigma, \tau \in \mathbb{Z}_q$ at random, which defines f, and gives h to Bob, then for any $(v, w) = (g^\beta, u^\beta) = \theta(\beta) \in \mathcal{L}_u$, Bob can compute $f(v, w)$ as h^β, since

$$f(v, w) = v^\sigma w^\tau = (g^\beta)^\sigma (u^\beta)^\tau = (g^\sigma u^\tau)^\beta = h^\beta.$$

So this is a projective hash function. To show that it is universal, it suffices to show that h and $f(v, w)$ are uniformly and independently distributed over $\mathbb{G}$, for all $(v, w) \in \mathbb{G}^2 \setminus \mathcal{L}_u$.

Lemma 12.5. *Suppose σ and τ are uniformly and independently distributed over $\mathbb{Z}_q$. Then for all $u, v, w, h, z \in \mathbb{G}$, if (u, v, w) is not a DH-triple, then*

$$\Pr[g^\sigma u^\tau = h \wedge v^\sigma w^\tau = z] = \frac{1}{q^2}.$$

Proof. Let $u, v, w, h, z \in \mathbb{G}$ be fixed, and assume that (u, v, w) is not a DH-triple. Suppose $u = g^\alpha$, $v = g^\beta$, and $w = g^\gamma$. Since (u, v, w) is not a DH-triple, we have $\gamma \neq \alpha\beta$. Consider the event

$g^\sigma u^\tau = h \wedge v^\sigma w^\tau = z$. Taking discrete logarithms, we can write this as a matrix equation:

$$\begin{pmatrix} \mathsf{Dlog}_g h \\ \mathsf{Dlog}_g z \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & \alpha \\ \beta & \gamma \end{pmatrix}}_{=:M} \begin{pmatrix} \sigma \\ \tau \end{pmatrix}. \tag{12.15}$$

We claim that the matrix M is non-singular. One way to see this is to calculate its determinant $\det(M) = \gamma - \alpha\beta \neq 0$. Another way to see this is to observe that the second row of M cannot be a scalar multiple of the first: if it were, then by looking at the first column of M, the second row of M would have to be equal to β times the first, and by looking at the second column of M, this would imply $\gamma = \alpha\beta$, which is not the case.

Since M is non-singular, (12.15) is satisfied by a unique pair (σ, τ). Moreover, since σ and τ are distributed uniformly and independently over $\mathbb{Z}_q$, this happens with probability $1/q^2$. □

The way we will use the above property in the analysis of our encryption scheme $\mathcal{E}_{\mathrm{CS}}$ is characterized by the following game:

Attack Game 12.4 (Universal distinguishing game). For a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $\quad (b = 0, 1)$:

- $\mathcal{A}$ chooses $u \in \mathbb{G}$ and $(v, w) \in \mathbb{G}^2 \setminus \mathcal{L}_u$, and sends (u, v, w) to the challenger.
- The challenger chooses $\sigma, \tau \in \mathbb{Z}_q$ at random, defining $f := f_{\sigma,\tau}$ as in (12.13), and computes the auxiliary information h that defines f on $\mathcal{L}_u$ as in (12.14). The challenger then computes

  $$z_0 \leftarrow f(v, w), \quad z_1 \xleftarrow{\mathrm{R}} \mathbb{G},$$

  and sends both h and z_b to $\mathcal{A}$.
- $\mathcal{A}$ then makes a series of *evaluation queries* to the challenger. Each such query is of the form $(\tilde{v}, \tilde{w}) \in \mathcal{L}_u$, to which the challenger replies with $\tilde{z} \leftarrow f(\tilde{v}, \tilde{w})$.
- At the end of the game, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b. □

Lemma 12.6. *In Attack Game 12.4,* $\Pr[W_0] = \Pr[W_1]$ *for all adversaries* $\mathcal{A}$.

Proof sketch. The proof follows almost immediately from Lemma 12.5, which says that h and z_0 are independent, so replacing z_0 by random z_1 does not change the distribution of the adversary's view. The only additional observation is that the evaluation queries do not leak any additional information about f, since if $(\tilde{v}, \tilde{w}) \in \mathcal{L}_u$, the value $f(\tilde{v}, \tilde{w})$ is completely determined by h. □

Note that in Attack Game 12.4, the challenger does not explicitly check that $(\tilde{v}, \tilde{w}) \in \mathcal{L}_u$ for the evaluation queries — we just assume that the adversary adheres to this restriction. In any case, the result of Lemma 12.6 applies to computationally unbounded adversaries, so this is not really an issue. Additionally, in our eventual application of Lemma 12.6, the adversary will in fact know $\alpha = \mathsf{Dlog}_g u$. See Exercise 12.26 for an analysis of a stronger version of Attack Game 12.4.

12.5.2 Universal$_2$ projective hash functions

In our encryption scheme, we will need an independence property that is a bit stronger than universal, which is called **universal$_2$**. Again, we present the intuitive idea in terms of function delegation. As before, we have a function $\theta : \mathcal{X} \to \mathcal{Y}$, and $\mathcal{L} \subseteq \mathcal{Y}$ is the image of θ. In this scenario, Alice has a function $f' : \mathcal{Y} \times \mathcal{T} \to \mathcal{Z}$, and she wants to give Bob auxiliary information h' that will allow him to compute f' on $\mathcal{L} \times \mathcal{T}$. The values in the set $\mathcal{T}$ may be thought of as "tags" that are used to separate the inputs to the function. The stronger property we want is this: for all $y, \hat{y} \in \mathcal{Y} \setminus \mathcal{L}$ and $t, \hat{t} \in \mathcal{T}$ with $t \neq \hat{t}$, the values h', $f'(y,t)$, and $f'(\hat{y},\hat{t})$ are mutually independent, with $f'(y,t)$ and $f'(\hat{y},\hat{t})$ each uniformly distributed over $\mathcal{Z}$. In particular, given h' and $f'(y,t)$, the value $f'(\hat{y},\hat{t})$ is completely unpredictable.

We can easily extend our universal projective hash function scheme for $\mathcal{L}_u \subseteq \mathbb{G}^2$ in Section 12.5.1 to obtain a universal$_2$ projective hash function scheme for $\mathcal{L}_u$. In this scheme, our "tags" will be elements of $\mathbb{Z}_q$. For $\sigma, \tau \in \mathbb{Z}_q$, let $f_{\sigma,\tau} : \mathbb{G}^2 \to \mathbb{G}$ be defined as in (12.16). We define a new function $f' := f'_{\sigma_1,\tau_1,\sigma_2,\tau_2}$, indexed by randomly chosen $\sigma_1, \tau_1, \sigma_2, \tau_2 \in \mathbb{Z}_q$, as follows

$$\begin{aligned} f'_{\sigma_1,\tau_1,\sigma_2,\tau_2} : \qquad & \mathbb{G}^2 \times \mathbb{Z}_q \to \mathbb{G}, \\ & (v, w, \rho) \mapsto f_{\sigma_1,\tau_1}(v,w) \cdot \big(f_{\sigma_2,\tau_2}(v,w)\big)^{\rho} = v^{\sigma_1+\rho\sigma_2} w^{\tau_1+\rho\tau_2}. \end{aligned} \tag{12.16}$$

The auxiliary information that defines f' on $\mathcal{L}_u \times \mathbb{Z}_q$ is $h' := (h_1, h_2)$, where h_i is the auxiliary information that defines f_{σ_i,τ_i} on $\mathcal{L}_u$; that is

$$h_i := f_{\sigma_i,\tau_i}(g, u) = g^{\sigma_i} u^{\tau_i} \quad \text{for } i = 1, 2. \tag{12.17}$$

It should be clear that if Alice chooses $\sigma_1, \tau_1, \sigma_2, \tau_2 \in \mathbb{Z}_q$ at random, which defines f', and gives $h' = (h_1, h_2)$ to Bob, then for any $(v, w) = (g^{\beta}, u^{\beta}) \in \mathcal{L}_u$, and any $\rho \in \mathbb{Z}_q$, Bob can compute $f'(v, w)$ as $(h_1 h_2^{\rho})^{\beta}$. The universal$_2$ independence property is established by the following lemma, which says that for all $(v, w), (\hat{v}, \hat{w}) \in \mathbb{G}^2 \setminus \mathcal{L}_u$ and $\rho \neq \hat{\rho}$, the values h_1, h_2, $f'(v, w, \rho)$, and $f'(\hat{v}, \hat{w}, \hat{\rho})$ are uniformly and independently distributed over $\mathbb{G}$.

Lemma 12.7. *Suppose $\sigma_1, \tau_1, \sigma_2, \tau_2$ are uniformly and independently distributed over $\mathbb{Z}_q$. Then for all $u, v, w, \hat{v}, \hat{w}, h_1, h_2, z, \hat{z} \in \mathbb{G}$ and all $\rho, \hat{\rho} \in \mathbb{Z}_q$, if (u, v, w) and $(u, \hat{v}, \hat{w})$ are not DH-triples, and $\rho \neq \hat{\rho}$, then*

$$\Pr[g^{\sigma_1} u^{\tau_1} = h_1 \ \wedge \ g^{\sigma_2} u^{\tau_2} = h_2 \ \wedge \ v^{\sigma_1+\rho_1\sigma_2} w^{\tau_1+\rho_1\tau_2} = z \ \wedge \ \hat{v}^{\sigma_1+\rho_2\sigma_2} \hat{w}^{\tau_1+\rho_2\tau_2} = \hat{z}] = \frac{1}{q^4}.$$

Proof sketch. The basic idea is the same as the proof of Lemma 12.5. The relevant matrix equation now is:

$$\begin{pmatrix} \mathsf{Dlog}_g h_1 \\ \mathsf{Dlog}_g h_2 \\ \mathsf{Dlog}_g z \\ \mathsf{Dlog}_g \hat{z} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & \alpha & 0 & 0 \\ 0 & 0 & 1 & \alpha \\ \beta & \gamma & \rho\beta & \rho\gamma \\ \hat{\beta} & \hat{\gamma} & \hat{\rho}\hat{\beta} & \hat{\rho}\hat{\gamma} \end{pmatrix}}_{=:\overline{M}} \begin{pmatrix} \sigma_1 \\ \tau_1 \\ \sigma_2 \\ \tau_2 \end{pmatrix}. \tag{12.18}$$

Here, $u = g^{\alpha}$, $v = g^{\beta}$, $w = g^{\gamma}$, $\hat{v} = g^{\hat{\beta}}$, and $\hat{w} = g^{\hat{\gamma}}$. The key fact is that the matrix $\overline{M}$ is non-singular. Indeed, one can again just compute the determinant

$$\det(\overline{M}) = (\rho - \hat{\rho})(\gamma - \alpha\beta)(\hat{\gamma} - \alpha\hat{\beta}),$$

which is nonzero under our assumptions. □

The way we will use the above property in the analysis of our encryption scheme $\mathcal{E}_{\text{CS}}$ is characterized by the following game:

Attack Game 12.5 (Universal$_2$ guessing game). For a given adversary $\mathcal{A}$, the game runs as follows.

- $\mathcal{A}$ chooses $u \in \mathbb{G}$, $(v, w) \in \mathbb{G}^2 \setminus \mathcal{L}_u$, and $\rho \in \mathbb{Z}_q$, and sends (u, v, w, ρ) to the challenger.
- The challenger chooses $\sigma_1, \tau_1, \sigma_2, \tau_2 \in \mathbb{Z}_q$ at random, defining $f' := f'_{\sigma_1,\tau_1,\sigma_2,\tau_2}$ as in (12.16). In addition, the challenger computes the auxiliary information (h_1, h_2) that defines f' on $\mathcal{L}_u \times \mathbb{Z}_q$ as in (12.17). The challenger then computes

$$z \leftarrow f'(v, w, \rho)$$

  and sends h_1, h_2, and z to $\mathcal{A}$.
- $\mathcal{A}$ then makes a series of *evaluation queries* to the challenger. Each such query is of the form $(\tilde{v}, \tilde{w}, \tilde{\rho}) \in \mathbb{G}^2 \times \mathbb{Z}_q$, where $(\tilde{v}, \tilde{w}) \in \mathcal{L}_u$, to which the challenger replies with $\tilde{z} \leftarrow f'(\tilde{v}, \tilde{w}, \tilde{\rho})$.
- At the end of the game, $\mathcal{A}$ outputs a list of tuples

$$(\hat{z}_i, \hat{v}_i, \hat{w}_i, \hat{\rho}_i) \in \mathbb{G}^3 \times \mathbb{Z}_q \quad (i = 1, \ldots, Q).$$

We say $\mathcal{A}$ wins the game if for some $i = 1, \ldots, Q$, we have

$$(\hat{v}_i, \hat{w}_i) \notin \mathcal{L}_u, \quad \hat{\rho}_i \neq \rho, \quad \text{and} \quad \hat{z}_i = f'(\hat{v}_i, \hat{w}_i, \hat{\rho}_i). \quad \square$$

Lemma 12.8. *In Attack Game 12.5, for any adversary $\mathcal{A}$ that outputs at most Q tuples, the probability that it wins is at most Q/q.*

Proof sketch. The proof follows almost immediately from the Union Bound, along with Lemma 12.7, which says that for each $i = 1, \ldots, Q$, the values h_1, h_2, z, and z_i are mutually independent. As we observed in the proof of Lemma 12.6, the evaluation queries do not leak any additional information about f', since if $(\tilde{v}, \tilde{w}) \in \mathcal{L}_u$, the value $f'(\tilde{v}, \tilde{w}, \tilde{\rho})$ is completely determined by (h_1, h_2). □

See Exercise 12.26 for an analysis of a stronger version of Attack Game 12.5.

12.5.3 The $\mathcal{E}_{\text{CS}}$ scheme

Without further ado, we present the scheme $\mathcal{E}_{\text{CS}}$. It makes use of

- a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$,
- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a hash function $H : \mathbb{G}^2 \to \mathcal{K}$,
- a hash function $H' : \mathbb{G}^2 \to \mathbb{Z}_q$.

The message space for $\mathcal{E}_{\text{CS}}$ is $\mathcal{M}$, and the ciphertext space is $\mathbb{G}^3 \times \mathcal{C}$. We now describe the key generation, encryption, and decryption algorithms for $\mathcal{E}_{\text{CS}}$.

- the key generation algorithm runs as follows:

```
$G() :=$  $\alpha \xleftarrow{R} \mathbb{Z}_q,\ \ u \leftarrow g^\alpha$
          $\sigma, \tau \xleftarrow{R} \mathbb{Z}_q,\ h \leftarrow g^\sigma u^\tau$
          $\sigma_1, \tau_1, \sigma_2, \tau_2 \xleftarrow{R} \mathbb{Z}_q,\ h_1 \leftarrow g^{\sigma_1} u^{\tau_1},\ h_2 \leftarrow g^{\sigma_2} u^{\tau_2}$
          $pk \leftarrow (u, h, h_1, h_2),\ \ sk \leftarrow (\sigma, \tau, \sigma_1, \tau_1, \sigma_2, \tau_2)$
          output $(pk, sk)$;
```

- for a given public key $pk = (u, h, h_1, h_2) \in \mathbb{G}^4$ and message $m \in \mathcal{M}$, the encryption algorithm runs as follows:

```
$E(pk, m) :=$  $\beta \xleftarrow{R} \mathbb{Z}_q,\ \ v \leftarrow g^\beta,\ \ w \leftarrow u^\beta,\ \ \rho \leftarrow H'(v, w)$
               $z \leftarrow h^\beta,\ \ z' \leftarrow (h_1 h_2^\rho)^\beta$
               $k \leftarrow H(v, z),\ \ c \xleftarrow{R} E_s(k, m)$
               output $(v, w, z', c)$;
```

- for a given secret key $sk = (\sigma, \tau, \sigma_1, \tau_1, \sigma_2, \tau_2) \in \mathbb{Z}_q^6$ and a ciphertext $(v, w, z', c) \in \mathbb{G}^3 \times \mathcal{C}$, the decryption algorithm runs as follows:

```
$D(sk,\ (v, w, z', c)\ ) :=$  $\rho \leftarrow H'(v, w)$
                               if $v^{\sigma_1 + \rho\sigma_2} w^{\tau_1 + \rho\tau_2} = z'$
                                   then $z \leftarrow v^\sigma w^\tau,\ \ k \leftarrow H(v, z),\ \ m \leftarrow D_s(k, c)$
                                   else $m \leftarrow$ reject
                               output $m$.
```

To understand what is going on, it is best to view the above construction in terms of the projective hash functions defined in Sections 12.5.1 and 12.5.2.

- The key generation algorithm chooses $u \in \mathbb{G}$ at random, which defines $\mathcal{L}_u = \{(g^\beta, u^\beta) : \beta \in \mathbb{Z}_q\}$. The choice of σ, τ defines the function $f = f_{\sigma,\tau}$ as in (12.13), and the value h is the auxiliary information that defines f on $\mathcal{L}_u$, as in (12.14). The choice of $\sigma_1, \tau_1, \sigma_2, \tau_2$ defines the function $f' = f'_{\sigma_1,\tau_1,\sigma_2,\tau_2}$ as in (12.16), and the value (h_1, h_2) is the auxiliary information that defines f' on $\mathcal{L}_u \times \mathbb{Z}_q$, as in (12.17).

- The encryption algorithm chooses a random $(v, w) \in \mathcal{L}_u$, and computes $z = f(v, w)$ and $z' = f'(v, w, \rho)$, where $\rho = H'(v, w)$. These computations are done using the auxiliary information in the public key. A symmetric key is then derived from (v, z) using H, which is used to encrypt m using E_s.

- The decryption algorithm first checks that $z' = f'(v, w, \rho)$, where $\rho = H'(v, w)$. If this check passes, the algorithm then computes $z = f(v, w)$, derives a symmetric key from (v, z) using H, and uses this to decrypt c using D_s.

These observations immediately imply that decryption undoes encryption, so the basic correctness requirements are met. Combined with Lemmas 12.6 and 12.8, these observations will also allow us to prove that $\mathcal{E}_{CS}$ is CCA secure under the DDH assumption.

Theorem 12.9. *If the DDH assumption holds in $\mathbb{G}$, $\mathcal{E}_s$ is 1CCA secure, H is a secure KDF, and H' is collision resistant, then $\mathcal{E}_{CS}$ is CCA secure.*

In particular, for every 1CCA adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\mathrm{CS}}$ as in Definition 12.2, and makes at most Q_{d} decryption queries, there exist a DDH adversary $\mathcal{B}_{\mathrm{ddh}}$ for $\mathbb{G}$ as in Attack Game 10.6, a 1CCA adversary $\mathcal{B}_{\mathrm{s}}$ that attacks $\mathcal{E}_{\mathrm{s}}$ as in Definition 9.6, a KDF adversary $\mathcal{B}_{\mathrm{kdf}}$ that attacks H as in Attack Game 11.3, and a collision-finding adversary $\mathcal{B}_{\mathrm{cr}}$ that attacks H' as in Attack Game 8.1, where $\mathcal{B}_{\mathrm{ddh}}$, $\mathcal{B}_{\mathrm{s}}$, $\mathcal{B}_{\mathrm{kdf}}$, $\mathcal{B}_{\mathrm{cr}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\begin{aligned}\mathrm{1CCA}\mathsf{adv}[\mathcal{A},\mathcal{E}_{\mathrm{CS}}] \le 2\Big(&\mathrm{DDH}\mathsf{adv}[\mathcal{B}_{\mathrm{ddh}},\mathbb{G}] + \mathrm{KDF}\mathsf{adv}[\mathcal{B}_{\mathrm{kdf}},H] \\ &+ \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{cr}},H'] + \frac{Q_{\mathrm{d}}+1}{q}\Big) + \mathrm{1CCA}\mathsf{adv}[\mathcal{B}_{\mathrm{s}},\mathcal{E}_{\mathrm{s}}].\end{aligned} \tag{12.19}$$

Proof. As usual, it is convenient to use the bit-guessing versions of the 1CCA attack games. We prove

$$\begin{aligned}\mathrm{1CCA}\mathsf{adv}^*[\mathcal{A},\mathcal{E}_{\mathrm{CS}}] \le\ &\mathrm{DDH}\mathsf{adv}[\mathcal{B}_{\mathrm{ddh}},\mathbb{G}] + \mathrm{KDF}\mathsf{adv}[\mathcal{B}_{\mathrm{kdf}},H] \\ &+ \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{cr}},H'] + \frac{Q_{\mathrm{d}}+1}{q} + \mathrm{1CCA}\mathsf{adv}^*[\mathcal{B}_{\mathrm{s}},\mathcal{E}_{\mathrm{s}}].\end{aligned} \tag{12.20}$$

Then (12.19) follows by (12.2) and (9.2).

We define a series of games, Game j for $j = 0, \ldots, 7$. Game 0 is the bit-guessing version of Attack Game 12.1 played by $\mathcal{A}$ with respect to $\mathcal{E}_{\mathrm{CS}}$. In each game, b denotes the random bit chosen by the challenger, while $\hat{b}$ denotes the bit output by $\mathcal{A}$. For $j = 0, \ldots, 7$, we define W_j to be the event that $\hat{b} = b$ in Game j.

Game 0. The logic of the challenger is shown in Fig. 12.3. The adversary can make any number of decryption queries, but at most one encryption query. Note that in the initialization step, the challenger performs those computations associated with the encryption query that it can, without yet knowing the challenge plaintext. Also note that in the initialization step, the challenger sets $c \leftarrow \perp$, and in processing the encryption query, overwrites c with a ciphertext in $\mathcal{C}$. Thus, decryption queries processed while $c = \perp$ are phase 1 queries, while those processed while $c \neq \perp$ are phase 2 queries. The game is described using the terminology of projective hash functions, as discussed above.

Game 1. We replace the lines marked (2) and (3) in Fig. 12.3 as follows:

```
(2)     $z \leftarrow f(v, w)$
(3)     $z' \leftarrow f'(v, w, \rho)$
```

Here, instead of using the auxiliary information that allows us to compute f on $\mathcal{L}_u$ and f' on $\mathcal{L}_u \times \mathbb{Z}_q$, we compute them directly, using the secret key *sk*. This does not change the result of the computation in any way. Therefore,

$$\Pr[W_1] = \Pr[W_0]. \tag{12.21}$$

The motivation for making this change is that now, the only place where we use the exponents α, β, and γ is in the definition of the group elements u, v, and w, which allows us to play the "DDH card" in the next step of the proof.

Game 2. We replace the line marked (1) in Fig. 12.3 with

```
(1)     $\gamma \xleftarrow{\mathrm{R}} \mathbb{Z}_q$
```

```
        initialization:
                $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$
(1)             $\gamma \leftarrow \alpha\beta$
                $u \leftarrow g^{\alpha}$, $v \leftarrow g^{\beta}$, $w \leftarrow g^{\gamma}$
                $\rho \leftarrow H'(v, w)$
                $\sigma, \tau \xleftarrow{R} \mathbb{Z}_q$, $h \leftarrow g^{\sigma} u^{\tau}$, let $f := f_{\sigma,\tau}$ (as in (12.13))
                $\sigma_1, \tau_1, \sigma_2, \tau_2 \xleftarrow{R} \mathbb{Z}_q$, $h_1 \leftarrow g^{\sigma_1} u^{\tau_1}$, $h_2 \leftarrow g^{\sigma_2} u^{\tau_2}$, let $f' := f'_{\sigma_1,\tau_1,\sigma_2,\tau_2}$ (as in (12.16))
(2)             $z \leftarrow h^{\beta}$
(3)             $z' \leftarrow (h_1 h_2^{\rho})^{\beta}$
(4)             $k \leftarrow H(v, z)$
                $b \xleftarrow{R} \{0, 1\}$, $c \leftarrow \bot$
                send the public key $(u, h, h_1, h_2)$ to $\mathcal{A}$;

        upon receiving an encryption query $(m_0, m_1) \in \mathcal{M}^2$:
                $c \xleftarrow{R} E_s(k, m_b)$, send $(v, w, z', c)$ to $\mathcal{A}$;

        upon receiving a decryption query $(\hat{v}, \hat{w}, \hat{z}', \hat{c}) \in \mathbb{G}^3 \times \mathcal{C}$, where $(\hat{v}, \hat{w}, \hat{z}', \hat{c}) \neq (v, w, z', c)$:
                if $(\hat{v}, \hat{w}, \hat{z}') = (v, w, z')$ then
                        $\hat{m} \leftarrow D_s(k, \hat{c})$
                else
                        $\hat{\rho} \leftarrow H'(\hat{v}, \hat{w})$
(5)                     if $\hat{z}' \neq f'(\hat{v}, \hat{w}, \hat{\rho})$
                                then $\hat{m} \leftarrow$ reject
                                else  $\hat{z} \leftarrow f(\hat{v}, \hat{w})$, $\hat{k} \leftarrow H(\hat{v}, \hat{z})$, $\hat{m} \leftarrow D_s(\hat{k}, \hat{c})$
                send $\hat{m}$ to $\mathcal{A}$.
```

Figure 12.3: Game 0 challenger in the proof of Theorem 12.9

It is easy to see that

$$\left|\Pr[W_1] - \Pr[W_2]\right| \le \mathrm{DDH}\mathsf{adv}[\mathcal{B}_{\mathrm{ddh}}, \mathbb{G}] \tag{12.22}$$

for an efficient DDH adversary $\mathcal{B}_{\mathrm{ddh}}$, which works as follows. After it obtains its DDH problem instance (u, v, w) from its own challenger, adversary $\mathcal{B}_{\mathrm{ddh}}$ plays the role of challenger to $\mathcal{A}$ in Game 1, but using the given values u, v, w. If (u, v, w) is a random DH-triple, then this is equivalent to Game 1, and if (u, v, w) is a random triple, this is equivalent to Game 2. At the end of the game, $\mathcal{B}_{\mathrm{ddh}}$ outputs 1 if $\hat{b} = b$ and 0 otherwise.

Game 3. We replace the line marked (1) in Fig. 12.3 with

(1) $\gamma \xleftarrow{\mathrm{R}} \mathbb{Z}_q \setminus \{\alpha\beta\}$

Since the statistical distance between the uniform distribution on all triples and the uniform distribution on all non-DH-triples is $1/q$ (see Exercise 10.7), it follows that:

$$\left|\Pr[W_2] - \Pr[W_3]\right| \le \frac{1}{q}. \tag{12.23}$$

Game 4. We now play our "CR card". Let us define $\mathrm{Coll}_{u,v}(\hat{v}, \hat{w})$ to be *true* if $(\hat{v}, \hat{w}) \neq (v, w)$ and $H'(\hat{v}, \hat{w}) = H'(v, w)$, and to be *false*, otherwise. In this game, we "widen" the rejection rule at line (5), replacing it with

(5) if $\mathrm{Coll}_{u,v}(\hat{u}, \hat{v})$ or $z' \neq f'(\hat{v}, \hat{w}, \hat{\rho})$

Let Z_4 be the event that in Game 4, some decryption query, which would not have triggered the rejection rule of Game 3, does trigger the wider rejection rule in Game 4. Clearly, Games 3 and 4 proceed identically unless event Z_4 occurs. By the Difference Lemma, we have

$$\left|\Pr[W_3] - \Pr[W_4]\right| \le \Pr[Z_4]. \tag{12.24}$$

It should be clear that

$$\Pr[Z_4] \le \mathrm{CR}\mathsf{adv}[\mathcal{B}_{\mathrm{cr}}, H']. \tag{12.25}$$

for an efficient collision-finding adversary $\mathcal{B}_{\mathrm{cr}}$. Indeed, adversary $\mathcal{B}_{\mathrm{cr}}$ just plays Game 4 and waits for the event Z_4 to happen.

Game 5. We again widen the rejection rule at line (5), replacing it with:

(5) if $\hat{v}^{\alpha} \neq \hat{w}$ or $\mathrm{Coll}_{u,v}(\hat{u}, \hat{v})$ or $z' \neq f'(\hat{v}, \hat{w}, \hat{\rho})$

So this rule will reject the ciphertext if $(\hat{v}, \hat{w}) \notin \mathcal{L}_u$.

Let Z_5 be the event that in Game 5, some decryption query, which would not have triggered the rejection rule of Game 4, does trigger the wider rejection rule in Game 5. Clearly, Games 4 and 5 proceed identically unless event Z_5 occurs. By the Difference Lemma, we have

$$\left|\Pr[W_4] - \Pr[W_5]\right| \le \Pr[Z_5]. \tag{12.26}$$

We will argue that

$$\Pr[Z_5] \le \frac{Q_{\mathrm{d}}}{q}. \tag{12.27}$$

Suppose Z_5 happened on a particular decryption query $(\hat{v}, \hat{w}, \hat{z}', \hat{c})$. We claim that for this ciphertext, we have (i) $(\hat{v}, \hat{w}) \notin \mathcal{L}_u$, (ii) $\hat{\rho} \neq \rho$, and (iii) $\hat{z}' = f'(\hat{v}, \hat{w}, \hat{\rho})$. Clearly, we must have (i), as otherwise, this ciphertext could not have triggered the rejection rule in Game 5. We must also have (iii), as otherwise, this ciphertext would have been rejected under the original rejection rule. Suppose (ii) did not hold. Then we must have $(\hat{v}, \hat{w}) = (v, w)$, as otherwise, this ciphertext would have been rejected under the collision rule added in Game 4. So we have $\hat{z}' = f'(\hat{v}, \hat{w}, \hat{\rho}) = f(v, w, \rho) = z$. But then this decryption query would not even have reached line (5) in the first place (it would have been decrypted directly as $D_{\mathrm{s}}(k, \hat{c})$ three lines above).

Using the claim, we will show how to design an adversary that wins Attack Game 12.5 with probability at least $\Pr[Z_5]$, and then use Lemma 12.8 to get an upper bound on $\Pr[Z_5]$. We shall refer to Attack Game 12.5 as "the guessing game" from here on out.

We can play the guessing game by running Game 5, but using the challenger in the guessing game to evaluate f', as needed. That challenger gives us $f'(v, w, \rho)$, along with h_1 and h_2, at the beginning of the guessing game. Now, whenever $\mathcal{A}$ makes a decryption query $(\hat{v}, \hat{w}, \hat{z}', \hat{c})$ that brings us to line (5), we first check if $\hat{v}^{\alpha} = \hat{w}$; if so, we evaluate the rest of the test at line (5) by making the evaluation query $(\hat{v}, \hat{w}, \hat{\rho})$ in the guessing game, obtaining the value $f'(\hat{v}, \hat{w}, \hat{\rho})$, and comparing this to $\hat{z}'$; otherwise, we simply reject the decryption query, and append $(\hat{z}', \hat{v}, \hat{w}, \hat{\rho})$ to our output list in the guessing game. The reader may verify that we win the guessing game with probability at least $\Pr[Z_5]$. The bound (12.27) follows from Lemma 12.8, and the fact that our output list in the guessing game contains at most Q_{d} guesses.

Game 6. Everything we did so far was leading to this point, which is the crux of the proof. We replace line (2) in Fig. 12.3 with

(2) $\qquad z \xleftarrow{\mathrm{R}} \mathbb{G}$

We claim that

$$\Pr[W_6] = \Pr[W_5]. \tag{12.28}$$

This follows from Lemma 12.6, and the fact that in processing decryption queries in Game 5, we only need to evaluate $f(\hat{v}, \hat{w})$ at points $(\hat{v}, \hat{w}) \in \mathcal{L}_u$.

Game 7. Finally, the stage is set to play our "KDF card" and "1CCA card". We replace the line marked (4) by

(4) $\qquad k \xleftarrow{\mathrm{R}} \mathcal{K}$

It should be clear that

$$\bigl|\Pr[W_6] - \Pr[W_7]\bigr| \leq \mathrm{KDF}\mathsf{adv}[\mathcal{B}_{\mathrm{kdf}}, H] \tag{12.29}$$

and

$$\bigl|\Pr[W_7] - 1/2\bigr| = 1\mathrm{CCA}\mathsf{adv}^*[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}], \tag{12.30}$$

where $\mathcal{B}_{\mathrm{kdf}}$ is an efficient adversary attacking H as a KDF, and $\mathcal{B}_{\mathrm{s}}$ is a 1CCA adversary attacking $\mathcal{E}_{\mathrm{s}}$.

The bound (12.20) now follows directly from (12.21)–(12.30). □

Remark 12.3 (Group membership verification). For reasons similar to that discussed in Remark 12.1, it is essential that given a ciphertext (v, w, z', c), the decryption algorithm for $\mathcal{E}_{\mathrm{CS}}$ verifies that v and w are in $\mathbb{G}$. It is not necessary to explicitly check that z' is in $\mathbb{G}$, since the check that $v^{\sigma_1 + \rho\sigma_2} w^{\tau_1 + \rho\tau_2} = z'$ implies that z' is in $\mathbb{G}$. □

12.6 CCA security via a generic transformation

We have presented several constructions of CCA-secure public key encryption schemes. In Section 12.3, we saw how to achieve CCA security in the random oracle model using a trapdoor function scheme, and in particular (in Section 12.3.1) with RSA. In Section 12.4, we saw how to achieve CCA security in the random oracle model under the interactive CDH assumption, and with a bit more effort, we were able to achieve CCA security in Section 12.5 without resorting to the random oracle model, but under the DDH assumption.

It is natural to ask if there is a generic transformation that converts any CPA-secure public key encryption scheme into one that is CCA-secure, as we did for symmetric encryption in Chapter 9. The answer is yes. In the random oracle model it is possible to give a simple and efficient transformation from CPA-security to CCA-security. This transformation, called the **Fujisaki-Okamoto transformation**, allows one to efficiently convert any public-key encryption scheme that satisfies a very weak security property (weaker than CPA security) into a public-key encryption scheme that is CCA-secure in the random oracle model. It is possible, in principle, to give a similar transformation without relying on random oracles; however, the known constructions are too inefficient to be used in practice [56].

Applications. We show in Section 12.6.2 that applying the Fujisaki-Okamoto transformation to a variant of ElGamal encryption, gives a public key encryption scheme that is CCA-secure in the random oracle model under the ordinary CDH assumption, rather than the stronger, interactive CDH assumption. (Exercise 12.33 develops another approach to achieving the same result, with a tighter security reduction to the CDH assumption).

Beyond ElGamal, the Fujisaki-Okamoto transformation can be applied to other public key encryption schemes, such as Regev's lattice-based encryption scheme discussed in Chapter 17, the McEliece coding-based scheme [111], and the NTRU scheme [88]. All these systems can be made CCA secure, in the random oracle model, using the technique in this section.

The Fujisaki-Okamoto transformation. It is best to understand the Fujisaki-Okamoto transformation as a technique that allows us to build a trapdoor function scheme $\mathcal{T}_{\text{FO}}$ that is one way, even given an image oracle (as in Definition 12.3), starting from any one-way, probabilistic public-key encryption scheme $\mathcal{E}_{\text{a}} = (G_{\text{a}}, E_{\text{a}}, D_{\text{a}})$. We can then plug $\mathcal{T}_{\text{FO}}$ into the construction $\mathcal{E}'_{\text{TDF}}$ presented in Section 12.3, along with a 1CCA symmetric cipher, to obtain a public-key encryption scheme $\mathcal{E}_{\text{FO}}$ that is CCA secure in the random oracle model.

Let $\mathcal{E}_{\text{a}} = (G_{\text{a}}, E_{\text{a}}, D_{\text{a}})$ be an arbitrary public-key encryption scheme with message space $\mathcal{X}$ and ciphertext space $\mathcal{Y}$.

- The encryption algorithm E_{a} may be probabilistic, and in this case, it will be convenient to make its random coin tosses explicit. To this end, let us view E_{a} as a *deterministic* algorithm that takes three inputs: a public key pk, a message $x \in \mathcal{X}$, and a randomizer $r \in \mathcal{R}$, where $\mathcal{R}$ is some finite **randomizer space**. To encrypt a message $x \in \mathcal{X}$ under a public key pk, one chooses $r \in \mathcal{R}$ at random, and then computes the ciphertext $E_{\text{a}}(pk, x; r)$.

- In general, the decryption algorithm D_{a} may return the special symbol reject; however, we will assume that this is not the case. That is, we will assume that D_{a} always returns an element in the message space $\mathcal{X}$. This is not a serious restriction, as we can always modify the decryption algorithm so as to return some default message instead of reject. This assumption will simplify the presentation somewhat.

The Fujisaki-Okamoto transformation applied to $\mathcal{E}_{\mathrm{a}} = (G_{\mathrm{a}}, E_{\mathrm{a}}, D_{\mathrm{a}})$ works as follows. We will also need a hash function $U : \mathcal{X} \to \mathcal{R}$, mapping messages to randomizers, which will be modeled as a random oracle in the security analysis. The trapdoor function scheme is $\mathcal{T}_{\mathrm{FO}} = (G_{\mathrm{a}}, F, D_{\mathrm{a}})$, defined over $(\mathcal{X}, \mathcal{Y})$, where

$$F(pk, x) := E_{\mathrm{a}}(pk, x; U(x)). \tag{12.31}$$

To prove that $\mathcal{T}_{\mathrm{FO}}$ is one way given an image oracle, in addition to modeling U as a random oracle, we will need to make the following assumptions, which will be made more precise below:

1. $\mathcal{E}_{\mathrm{a}}$ is **one way**, which basically means that given an encryption of a random message $x \in \mathcal{X}$, it is hard to compute x;

2. $\mathcal{E}_{\mathrm{a}}$ is **unpredictable**, which basically means that a random re-encryption of any ciphertext $y \in \mathcal{Y}$ is unlikely to be equal to y.

We now make the above assumptions more precise. As usual, the one-wayness property is defined in terms of an attack game.

Attack Game 12.6 (One-way encryption). For a given public-key encryption scheme $\mathcal{E}_{\mathrm{a}} = (G_{\mathrm{a}}, E_{\mathrm{a}}, D_{\mathrm{a}})$ with message space $\mathcal{X}$, ciphertext space $\mathcal{Y}$, and randomizer space $\mathcal{R}$, and a given adversary $\mathcal{A}$, the attack game proceeds as follows:

- The challenger computes

$$(pk, sk) \xleftarrow{\mathrm{R}} G_{\mathrm{a}}(),\ \ x \xleftarrow{\mathrm{R}} \mathcal{X},\ \ r \xleftarrow{\mathrm{R}} \mathcal{R},\ \ y \leftarrow E_{\mathrm{a}}(pk, x; r),$$

  and sends (pk, y) to the adversary.

- The adversary outputs $\hat{x} \in \mathcal{R}$.

We say $\mathcal{A}$ wins the above game if $\hat{x} = x$, and we define $\mathcal{A}$'s advantage $\mathrm{OW}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{a}}]$ to be the probability that $\mathcal{A}$ wins the game. □

Definition 12.5 (One-way encryption). *A public-key encryption scheme $\mathcal{E}_{\mathrm{a}}$ is **one way** if for every efficient adversary $\mathcal{A}$, the value $\mathrm{OW}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{a}}]$ is negligible.*

Note that because $\mathcal{E}_{\mathrm{a}}$ may be probabilistic, an adversary that wins Attack Game 12.6 may not even know that they have won the game.

We define unpredictable encryption as follows.

Definition 12.6 (Unpredictable encryption). *Let $\mathcal{E}_{\mathrm{a}} = (G_{\mathrm{a}}, E_{\mathrm{a}}, D_{\mathrm{a}})$ be a given public-key encryption scheme with message space $\mathcal{X}$, ciphertext space $\mathcal{Y}$, and randomizer space $\mathcal{R}$. We say $\mathcal{E}_{\mathrm{a}}$ is **ϵ-unpredictable** if for every possible output (pk, sk) of G_{a} and every $y \in \mathcal{Y}$, if we choose $r \in \mathcal{R}$ at random, then we have*

$$\Pr[E_{\mathrm{a}}(pk, D_{\mathrm{a}}(sk, y); r) = y] \leq \epsilon.$$

*We say $\mathcal{E}_{\mathrm{a}}$ is **unpredictable** if it is ϵ-unpredictable for negligible ϵ.*

We note that the one-wayness assumption is implied by semantic security (see Exercise 12.9). We also note that, any public-key encryption scheme that is semantically secure typically is also unpredictable, even though this is not implied by the definition. Moreover, any public-key encryption scheme can be easily transformed into one that satisfies this assumption, without affecting the one-wayness assumption (see Exercise 12.10).

Theorem 12.10. *If U is modeled as a random oracle, and if $\mathcal{E}_{\rm a}$ is one way and unpredictable, then the trapdoor function scheme $\mathcal{T}_{\rm FO}$, resulting from the Fujisaki-Okamoto transformation (12.31), is one way given an image oracle.*

> *In particular, assume that $\mathcal{E}_{\rm a}$ is ϵ-unpredictable. Also assume that adversary $\mathcal{A}$ attacks $\mathcal{T}_{\rm FO}$ as in the random oracle version of Attack Game 12.2, and makes at most $Q_{\rm io}$ image oracle queries and $Q_{\rm ro}$ random oracle queries. Moreover, assume that $\mathcal{A}$ always includes its output among its random oracle queries. Then there exists an adversary $\mathcal{B}_{\rm ow}$ that breaks the one-wayness assumption for $\mathcal{E}_{\rm a}$ as in Attack Game 12.6, where $\mathcal{B}_{\rm ow}$ is an elementary wrapper around $\mathcal{A}$, such that*

$$\mathrm{IOW}^{\rm ro}\mathsf{adv}[\mathcal{A},\mathcal{T}_{\rm FO}] \le Q_{\rm io}\cdot\epsilon + Q_{\rm ro}\cdot \mathrm{OW}\mathsf{adv}[\mathcal{B}_{\rm ow},\mathcal{E}_{\rm a}]. \qquad (12.32)$$

Proof. We define Game 0 to be the game played between $\mathcal{A}$ and the challenger in the random oracle version of Attack Game 12.2 with respect to $\mathcal{T}_{\rm FO} = (G_{\rm a}, F, D_{\rm a})$. We then modify the challenger several times to obtain Games 1, 2, and so on. In each game, x denotes the random element of $\mathcal{X}$ chosen by the challenger. For $j = 0, 1, \ldots,$ we define W_j to be the event that x is among the random oracle queries made by $\mathcal{A}$ in Game j. As stated above, we assume that $\mathcal{A}$ always queries the random oracle at its output value: this is a reasonable assumption, and we can always trivially modify an any adversary to ensure that it behaves this way, increasing its random-oracle queries by at most 1. Clearly, we have

$$\mathrm{IOW}^{\rm ro}\mathsf{adv}[\mathcal{A},\mathcal{T}_{\rm FO}] \le \Pr[W_0]. \qquad (12.33)$$

Game 0. The challenger in Game 0 has to respond to random oracle queries, in addition to image oracle queries. We make use of an associative array $Map : \mathcal{X} \to \mathcal{R}$ to implement the random oracle representing the hash function U. The logic of the challenger is shown in Fig. 12.4. The adversary can make any number of random oracle queries and any number of image queries. The associative array $Pre : \mathcal{Y} \to \mathcal{X}$ is used to track the adversary's random oracle queries. Basically, $Pre[\hat{y}] = \hat{x}$ means that $\hat{y}$ is the image of $\hat{x}$ under $F(pk, \cdot)$.

Game 1. In this game, we make the following modification to the challenger. The line marked (2) in the logic for processing decryption queries is modified as follows:

```
(2)            if $\hat{y} \in \mathrm{Domain}(Pre)$
```

Let Z_1 be the event that in Game 1, the adversary submits an image oracle query $\hat{y}$ such that

$$\hat{y} \ne y, \quad \hat{y} \notin \mathrm{Domain}(Pre), \quad \text{and} \quad E_{\rm a}(pk, \hat{x}; \hat{r}) = \hat{y},$$

where $\hat{x}$ and $\hat{r}$ are computed as in the challenger. It is clear that Games 0 and 1 proceed identically unless Z_1 occurs, and so by the Difference Lemma, we have

$$|\Pr[W_1] - \Pr[W_0]| \le \Pr[Z_1]. \qquad (12.34)$$

```
    initialization:
        $(pk, sk) \overset{R}{\leftarrow} G_a()$, $x \overset{R}{\leftarrow} \mathcal{X}$, $r \overset{R}{\leftarrow} \mathcal{R}$, $y \leftarrow E_a(pk, x; r)$
        initialize empty associative arrays $Map : \mathcal{X} \to \mathcal{R}$ and $Pre : \mathcal{Y} \to \mathcal{X}$
(1)     $Map[x] \leftarrow r$
        send the public key $pk$ to $\mathcal{A}$;

    upon receiving an image oracle query $\hat{y} \in \mathcal{Y}$:
        if $\hat{y} = y$ then
            $result \leftarrow$ “yes”
        else
            $\hat{x} \leftarrow D_a(sk, \hat{y})$
            if $\hat{x} \notin \text{Domain}(Map)$ then $Map[\hat{x}] \overset{R}{\leftarrow} \mathcal{R}$
            $\hat{r} \leftarrow Map[\hat{x}]$
(2)         if $E_a(pk, \hat{x}; \hat{r}) = \hat{y}$
                then $result \leftarrow$ “yes”
                else  $result \leftarrow$ “no”
        send $result$ to $\mathcal{A}$;

    upon receiving a random oracle query $\hat{x} \in \mathcal{X}$:
        if $\hat{x} \notin \text{Domain}(Map)$ then $Map[\hat{x}] \overset{R}{\leftarrow} \mathcal{R}$
        $\hat{r} \leftarrow Map[\hat{x}]$, $\hat{y} \leftarrow E_a(pk, \hat{x}; \hat{r})$, $Pre[\hat{y}] \leftarrow \hat{x}$
        send $\hat{r}$ to $\mathcal{A}$
```

Figure 12.4: Game 0 challenger in the proof of Theorem 12.10

```
upon receiving an image oracle query $\hat{y} \in \mathcal{Y}$:
    if $\hat{y} \in \{y\} \cup \text{Domain}(Pre)$:
        then $result \leftarrow$ "yes"
        else $result \leftarrow$ "no"
    send $result$ to $\mathcal{A}$
```

Figure 12.5: Modified logic for image oracle queries

We argue that

$$\Pr[Z_1] \leq Q_{\text{io}} \cdot \epsilon, \tag{12.35}$$

where we are assuming that $\mathcal{E}_{\text{a}}$ is ϵ-unpredictable. Indeed, observe that in Game 1, if $\mathcal{A}$ makes an image query $\hat{y}$ with

$$\hat{y} \neq y \quad \text{and} \quad \hat{y} \notin \text{Domain}(Pre),$$

then either

- $\hat{x} = x$, and so $E_{\text{a}}(pk, \hat{x}; \hat{r}) = y \neq \hat{y}$ with certainty, or
- $\hat{x} \neq x$, and so $\hat{r}$ is independent of $\mathcal{A}$'s view, from which it follows that $E_{\text{a}}(pk, \hat{x}; \hat{r}) = \hat{y}$ with probability at most ϵ.

The inequality (12.35) then follows by the union bound.

Game 2. This game is the same Game 1, except that we implement the image oracle queries using the logic described in Fig. 12.5. The idea is that in Game 1, we do not really need to use the secret key to implement the image oracle queries.

It should be clear that

$$\Pr[W_2] = \Pr[W_1]. \tag{12.36}$$

Since we do not use the secret key at all in Game 2, this makes it easy to play our "one-wayness card."

Game 3. In this game, we delete the line marked (1) in Fig. 12.4.

We claim that

$$\Pr[W_3] = \Pr[W_2]. \tag{12.37}$$

Indeed, Games 2 and 3 proceed identically until $\mathcal{A}$ queries the random oracle at x. So if W_2 does not occur, neither does W_3, and if W_3 does not occur, neither does W_2. That is, W_2 and W_3 are identical events.

We sketch the design of an efficient adversary $\mathcal{B}$ such that

$$\Pr[W_3] \leq Q_{\text{ro}} \cdot \text{OW}\mathsf{adv}[\mathcal{B}, \mathcal{E}_{\text{a}}]. \tag{12.38}$$

The basic idea, as usual, is that $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$, as in Game 3, except that the values pk, sk, x, r, and y are generated by $\mathcal{B}$'s OW challenger, from which $\mathcal{B}$ obtains the values pk and y. Adversary $\mathcal{B}$ interacts with $\mathcal{A}$ just as the challenger in Game 3. The key observation is that $\mathcal{B}$ does not need to know the values sk, x, and r in order to carry out its duties. At the end of the game, if $\mathcal{A}$ made a random oracle query at the point x, then the value x will be contained in the set Domain(Map). In general, it may not be easy to determine which of the values in this set is the correct decryption of y, and so we use our usual guessing strategy; namely, $\mathcal{B}$ simply chooses an element at random from Domain(Map) as its guess at the decryption of y. It is clear that the inequality (12.38) holds.

The inequality (12.32) now follows from (12.33)–(12.38). That proves the theorem. □

12.6.1 A generic instantiation

Putting all the pieces together, we get the following public-key encryption scheme $\mathcal{E}_{\mathrm{FO}}$. The components consist of:

- a public-key encryption scheme $\mathcal{E}_{\mathrm{a}} = (G_{\mathrm{a}}, E_{\mathrm{a}}, D_{\mathrm{a}})$, with message space $\mathcal{X}$, ciphertext space $\mathcal{Y}$, and randomizer space $\mathcal{R}$;
- a symmetric cipher $\mathcal{E}_{\mathrm{s}} = (E_{\mathrm{s}}, D_{\mathrm{s}})$, with key space $\mathcal{K}$ and message space $\mathcal{M}$;
- hash functions $H : \mathcal{X} \to \mathcal{K}$ and $U : \mathcal{X} \to \mathcal{R}$.

The scheme $\mathcal{E}_{\mathrm{FO}} = (G_{\mathrm{a}}, E, D)$ has message space $\mathcal{M}$ and ciphertext space $\mathcal{Y} \times \mathcal{C}$. Encryption and decryption work as follows:

```
$E(pk, m)$        :=  $x \xleftarrow{R} \mathcal{X},\ r \leftarrow U(x),\ y \leftarrow E_a(pk, x; r)$
                      $k \leftarrow H(x),\ c \xleftarrow{R} E_s(k, m)$
                      output $(y, c)$;
$D(sk,\ (y, c)\ )$  :=  $x \leftarrow D_a(sk, y),\ r \leftarrow U(x)$
                      if $E_a(pk, x; r) \neq y$
                          then $m \leftarrow$ reject
                          else $k \leftarrow H(x),\ m \leftarrow D_s(k, c)$
                      output $m$.
```

Combining Theorem 12.2 and Theorem 12.10, we immediately get the following:

Theorem 12.11. *If H and U are modeled as a random oracles, $\mathcal{E}_{\mathrm{a}}$ is one way and unpredictable, and $\mathcal{E}_{\mathrm{s}}$ is 1CCA secure, then the above public-key encryption scheme $\mathcal{E}_{\mathrm{FO}}$ is CCA secure.*

In particular, assume that $\mathcal{E}_{\mathrm{a}}$ is ϵ-unpredictable. Then for every 1CCA adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\mathrm{FO}}$ as in the random oracle version of Definition 12.2, and which makes at most Q_{d} decryption queries, Q_H queries to the random oracle for H, and Q_U queries to the random oracle for U, there exist an adversary $\mathcal{B}_{\mathrm{ow}}$ that breaks the one-wayness assumption for $\mathcal{E}_{\mathrm{a}}$ as in Attack Game 12.6, and a 1CCA adversary $\mathcal{B}_{\mathrm{s}}$ that attacks $\mathcal{E}_{\mathrm{s}}$ as in Definition 9.6, where $\mathcal{B}_{\mathrm{ow}}$ and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that

$$1\mathrm{CCA}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{FO}}] \leq 2(Q_H + Q_U) \cdot \mathrm{OW}\mathsf{adv}[\mathcal{B}_{\mathrm{ow}}, \mathcal{E}_{\mathrm{a}}] + 2Q_{\mathrm{d}} \cdot \epsilon + 1\mathrm{CCA}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{12.39}$$

12.6.2 A concrete instantiation with ElGamal

In the Fujisaki-Okamoto transformation, we can easily use a variant of ElGamal encryption in the role of $\mathcal{E}_{\mathrm{a}}$. Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. We define a public-key encryption scheme $\mathcal{E}_{\mathrm{a}} = (G_{\mathrm{a}}, E_{\mathrm{a}}, D_{\mathrm{a}})$, with message space $\mathbb{G}$, ciphertext space $\mathbb{G}^2$, and randomizer space $\mathbb{Z}_q$. Public keys are of the form $u \in \mathbb{G}$ and secret keys of the form $\alpha \in \mathbb{Z}_q$. Key generation, encryption, and decryption work as follows:

$$
\begin{array}{rcl}
G_{\mathrm{a}}() & := & \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q,\ \ u \leftarrow g^{\alpha},\ \ pk \leftarrow u,\ \ sk \leftarrow \alpha \\
 & & \text{output } (pk, sk); \\
E_{\mathrm{a}}(u, x; \beta) & := & v \leftarrow g^{\beta},\ \ w \leftarrow u^{\beta},\ \ y \leftarrow wx \\
 & & \text{output } (v, y); \\
D_{\mathrm{a}}(\alpha,\ (v, y)) & := & w \leftarrow v^{\alpha},\ \ x \leftarrow y/w \\
 & & \text{output } x.
\end{array}
$$

We called this scheme multiplicative ElGamal in Exercise 11.5, where we showed that it is semantically secure under the DDH assumption. It easily verified that $\mathcal{E}_{\mathrm{a}}$ has the following properties:

- $\mathcal{E}_{\mathrm{a}}$ is one-way under the CDH assumption. Indeed, an adversary $\mathcal{A}$ that breaks the one-wayness assumption for $\mathcal{E}_{\mathrm{a}}$ is easily converted to an adversary $\mathcal{B}$ that breaks the CDH with same advantage. Given an instance $(u, v) \in \mathbb{G}^2$ of the CDH problem, adversary $\mathcal{B}$ plays the role of challenger against $\mathcal{A}$ in Attack Game 12.6 as follows:

  - $\mathcal{B}$ sets $y \xleftarrow{\mathrm{R}} \mathbb{G}$, and gives $\mathcal{A}$ the public key u and the ciphertext (v, y);
  - when $\mathcal{A}$ outputs $x \in \mathbb{G}$, adversary $\mathcal{B}$ outputs $w \leftarrow y/x$.

  Clearly, if x is the decryption of (v, y), then $w = y/x$ is the solution to the given instance (u, v) of the CDH problem.

- $\mathcal{E}_{\mathrm{a}}$ is $1/q$-unpredictable. Moreover, under the CDH assumption, it must be the case that $1/q$ is negligible.

Putting all the pieces together, we get the following public-key encryption scheme $\mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}} = (G, E, D)$. The components consist of:

- a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$;
- a symmetric cipher $\mathcal{E}_{\mathrm{s}} = (E_{\mathrm{s}}, D_{\mathrm{s}})$, with key space $\mathcal{K}$ and message space $\mathcal{M}$;
- hash functions $H : \mathbb{G} \to \mathcal{K}$ and $U : \mathbb{G} \to \mathbb{Z}_q$.

The message space of $\mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}}$ is $\mathcal{M}$ and its ciphertext space is $\mathbb{G}^2 \times \mathcal{C}$. Public keys are of the form $u \in \mathbb{G}$ and secret keys of the form $\alpha \in \mathbb{Z}_q$. The key generation, encryption, and decryption algorithms work as follows:

$$
\begin{array}{rcl}
G() & := & \alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q,\ \ u \leftarrow g^{\alpha},\ \ pk \leftarrow u,\ \ sk \leftarrow \alpha \\
 & & \text{output } (pk, sk); \\
E(u, m) & := & x \xleftarrow{\mathrm{R}} \mathbb{G},\ \ \beta \leftarrow U(x),\ \ v \leftarrow g^{\beta},\ \ w \leftarrow u^{\beta},\ \ y \leftarrow w \cdot x \\
 & & k \leftarrow H(x),\ \ c \xleftarrow{\mathrm{R}} E_{\mathrm{s}}(k, m) \\
 & & \text{output } (v, y, c); \\
D(\alpha,\ (v, y, c)) & := & w \leftarrow v^{\alpha},\ \ x \leftarrow y/w,\ \ \beta \leftarrow U(x) \\
 & & \text{if } g^{\beta} = v \\
 & & \quad\quad \text{then } k \leftarrow H(x),\ \ m \leftarrow D_{\mathrm{s}}(k, c) \\
 & & \quad\quad \text{else}\ \ m \leftarrow \mathsf{reject} \\
 & & \text{output } m.
\end{array}
$$

Here, we have optimized the decryption algorithm a bit: if $v = g^{\beta}$, then it follows that $E_{\mathrm{a}}(pk, x; \beta) = (g^{\beta}, u^{\beta} x) = (v, y)$, and so it is unnecessary to execute all of algorithm E_{a}.

As a special case of Theorem 12.11, we get the following:

Theorem 12.12. *If H and U are modeled as a random oracles, the CDH assumption holds for $\mathbb{G}$, and $\mathcal{E}_{\mathrm{s}}$ is 1CCA secure, then the above public-key encryption scheme $\mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}}$ is CCA secure.*

In particular, for every 1CCA adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}}$ as in the random oracle version of Definition 12.2, and which makes at most Q_{d} decryption queries, Q_H queries to the random oracle for H, and Q_U queries to the random oracle for U, there exist an adversary $\mathcal{B}_{\mathrm{cdh}}$ that breaks the CDH assumption for $\mathbb{G}$ as in Attack Game 10.5, and a 1CCA adversary $\mathcal{B}_{\mathrm{s}}$ that attacks $\mathcal{E}_{\mathrm{s}}$ as in Definition 9.6, where $\mathcal{B}_{\mathrm{cdh}}$ and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{1CCA}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}}] \leq 2(Q_H + Q_U) \cdot \mathrm{CDH}\mathsf{adv}[\mathcal{B}_{\mathrm{cdh}}, \mathbb{G}] + 2Q_{\mathrm{d}}/q + \mathrm{1CCA}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \tag{12.40}$$

Contrast this result to the construction in Section 12.4: to achieve CCA security, instead of the ordinary CDH assumption, that scheme requires the stronger, interactive CDH assumption. See Exercise 12.33 for another scheme with a tighter reduction to CDH.

Remark 12.4 (Group membership verification). Based on the discussion in Remark 12.1, one might presume that given a ciphertext (v, y, c), the decryption algorithm for $\mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}}$ should verify that v and y are in $\mathbb{G}$. However, the check $g^{\beta} = v$ already ensures that v is in $\mathbb{G}$. This leaves the question of whether the decryption algorithm needs to check that y is in $\mathbb{G}$. It turns out that this check is unnecessary (see Exercise 12.13 for details). □

12.7 CCA-secure public-key encryption with associated data

In Section 9.6, we introduced the notion of CCA security for symmetric-key ciphers with associated data. In this section, we briefly sketch how this notion can be adapted to public-key encryption.

First, we have to deal with the syntactic changes. A public-key encryption scheme $\mathcal{E} = (G, E, D)$ with associated data, or **AD public-key encryption scheme**, has the same basic structure as an ordinary public-key encryption scheme, except that the encryption algorithm E and decryption algorithm D each take an additional input d, called the **associated data**. Thus, E gets invoked as $c \xleftarrow{\mathrm{R}} E(pk, m, d)$, and D gets invoked as $m \leftarrow D(sk, c, d)$. As usual, we require that ciphertexts generated by E are correctly decrypted by D, as long as both are given the same associated data. That is, for all possible outputs (pk, sk) of G, and all messages m and associated data d, we have

$$\Pr[D(sk,\ E(pk,\ m,\ d),\ d\ ) = m] = 1.$$

Messages lie in some finite message space $\mathcal{M}$, ciphertexts in some finite ciphertext space $\mathcal{C}$, and associated data in some finite space $\mathcal{D}$. We say that $\mathcal{E}$ is defined over $(\mathcal{M}, \mathcal{D}, \mathcal{C})$.

Definition 12.7 (CCA and 1CCA security with associated data). *The definition of CCA security for ordinary public-key encryption schemes carries over naturally to AD public-key encryption schemes. Attack Game 12.1 is modified as follows. For encryption queries, in addition to a pair of messages (m_{i0}, m_{i1}), the adversary also submits associated data d_i, and the challenger computes $c_i \xleftarrow{\mathrm{R}} E(pk, m_{ib}, d_i)$. For decryption queries, in addition to a ciphertext $\hat{c}_j$, the adversary submits associated data $\hat{d}_j$, and the challenger computes $\hat{m}_j \leftarrow D(sk, \hat{c}_j, \hat{d}_j)$. The restriction is that* *the* pair $(\hat{c}_j, \hat{d}_j)$ *may not be among the pairs* $(c_1, d_1), (c_2, d_2), \ldots$ *corresponding to previous encryption queries. An adversary* $\mathcal{A}$*'s advantage in this game is denoted* $\mathrm{CCA}_{\mathrm{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$*, and the scheme is said to be* ***CCA secure*** *if this advantage is negligible for all efficient adversaries* $\mathcal{A}$*. If we restrict the adversary to a single encryption query, as in Definition 12.2, the advantage is denoted* $1\mathrm{CCA}_{\mathrm{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$*, and the scheme is said to be* ***1CCA secure*** *if this advantage is negligible for all efficient adversaries* $\mathcal{A}$*.*

Observations. We make a couple of simple observations.

- Theorem 12.1 carries over to AD schemes. That is, if an AD public-key encryption scheme is 1CCA secure, then it is also CCA secure. The proof and concrete security bounds go through with no real changes.

- All of the CCA-secure public-key encryption schemes presented in this chapter can be trivially converted to CCA-secure AD public-key encryption schemes, simply by replacing the symmetric cipher $\mathcal{E}_{\mathrm{s}}$ used in each construction with a 1CCA-secure AD cipher. The associated data for the AD public-key scheme is simply passed through to the AD symmetric-key cipher, in both the encryption and decryption algorithms. See part (g) of Exercise 12.5; see also Exercise 12.17 for an alternative approach.

Applications. CCA-secure AD public-key encryption has a number of natural applications. One such application is the key-escrow application, which we discussed in Section 12.2.3. In this application, we escrowed a file-encryption key k by encrypting the pair (k, h) under the public-key of a key escrow service. Here, h was the collision-resistant hash of some metadata *md* associated with the file, and the public-key encryption scheme used by the escrow service was assumed CCA secure. By encrypting the pair (k, h), the escrow service could enforce various access control policies, based on the metadata and the identity or credentials of an entity requesting the key k. However, the metadata itself was considered public information, and it did not really need to be encrypted, except that we wanted it to be bundled in some non-malleable way with the key k. This same effect can be achieved more naturally and efficiently by using a CCA-secure AD public-key encryption scheme, as follows. When the key k is escrowed, the escrow-ciphertext is generated by encrypting k using the metadata *md* as associated data. When a requesting entity presents a pair (c, md) to the escrow service, the service checks that the requesting entity's credentials and the supplied metadata conform to the access control policy, and if so, decrypts c using the supplied metadata *md* as associated data. The access control policy is enforced by the CCA-security property: attempting to decrypt the escrow-ciphertext using non-matching metadata as associated data will not leak any information about the corresponding file-encryption key.

We will also make use of CCA-secure AD public-key encryption in building signcryption schemes (see Section 13.7.3).

12.7.1 AD-only CCA security

A somewhat weaker notion of security that is sufficient in many applications is called **AD-only CCA security**. This notion of security is obtained by changing Definition 12.7 so that the restriction placed on a decryption query $(\hat{C}_j, \hat{d}_j)$ is that $\hat{d}_j$ should not be equal to any of the associated data values d_i submitted as part of any previous encryption query. Otherwise, the definition of AD-only CCA security is exactly the same as the definition of CCA security, with the corresponding advantage denoted $\mathrm{CCA}_{\mathrm{ado}}\mathsf{adv}[\mathcal{A},\mathcal{E}]$. We can also naturally define **AD-only 1CCA** security by restricting the adversary to a single decryption query, with the corresponding advantage denoted $1\mathrm{CCA}_{\mathrm{ado}}\mathsf{adv}[\mathcal{A},\mathcal{E}]$.

Observations. We make a few simple observations.

- AD-only CCA security constrains the adversary's ability to issue decryption queries compared to CCA security in Definition 12.7. Therefore, the notion of AD-only CCA security is no stronger than CCA security. However, if the associated data space $\mathcal{D}$ is sufficiently large, then under reasonable assumptions, one can convert an AD-only CCA secure scheme into a CCA secure scheme. This is explored in Exercise 14.13.

- AD-only CCA secure schemes can produce ciphertexts that are somewhat more compact that CCA secure schemes. See Exercise 12.19 for this and other results.

- Theorem 12.1 carries over to AD-only security. That is, if an AD public-key encryption scheme is AD-only 1CCA secure, then it is also AD-only CCA secure. Again, the proof and concrete security bounds go through with no real changes.

- In the key-escrow application discussed above, where we use associated data to enforce an access control policy, it is typically enough to use an AD-only CCA secure encryption scheme to prevent the types of abuses discussed in Section 12.2.3.

  Indeed, suppose Alice encrypts a file-encryption key k with associated data equal to the metadata md of the file f: the metadata includes Alice's identity, the name of the file, as well as the time the file was created and/or modified. Assuming Alice encrypts k using an AD-only CCA-secure scheme, one can prove the following: if an auditor is to get any information about k, then she must submit a decryption request to the escrow service with credentials that are consistent with md (but not necessarily with an identical ciphertext).

12.8 Case study: PKCS1, OAEP, OAEP+, and SAEP

The most widely used public-key encryption scheme using RSA is described in a standard from RSA Labs called PKCS1. This scheme is quite different from the scheme $\mathcal{E}_{\mathrm{RSA}}$ we presented in Section 12.3.1.

Why does the PKCS1 standard not use $\mathcal{E}_{\mathrm{RSA}}$? The reason is that when encrypting a short message — much shorter than the RSA modulus n — a PKCS1 ciphertext is more compact than an $\mathcal{E}_{\mathrm{RSA}}$ ciphertext. The $\mathcal{E}_{\mathrm{RSA}}$ scheme outputs a ciphertext (y, c) where y is in $\mathbb{Z}_n$ and c is a symmetric ciphertext, while a PKCS1 ciphertext is only a single element of $\mathbb{Z}_n$.

Public-key encryption for short messages is used in a variety of settings. For example, in some key exchange protocols, public-key encryption is only applied to short messages: a symmetric key and some metadata. Similarly, in some access control systems, one encrypts a short access token and nothing else. In these settings, schemes like PKCS1 are more space efficient than $\mathcal{E}_{\mathrm{RSA}}$. It is worth noting, however, that the ElGamal scheme $\mathcal{E}_{\mathrm{EG}}$ can produce even shorter ciphertexts (although encryption time with ElGamal is typically higher than with RSA).

Our goal in this section is to study PKCS1, and more generally, public-key encryption schemes based on a trapdoor function $\mathcal{T} = (G, F, I)$ defined over $(\mathcal{X}, \mathcal{Y})$, where the ciphertext is just a single element of $\mathcal{Y}$.

12.8.1 Padding schemes

Let $\mathcal{T} = (G, F, I)$ be a trapdoor function defined over $(\mathcal{X}, \mathcal{Y})$, and let $\mathcal{M}$ be some message space, where $|\mathcal{M}| \ll |\mathcal{X}|$. Our goal is to design a public-key encryption scheme where a ciphertext is just a single element in $\mathcal{Y}$. To do so, we use the following general paradigm: to encrypt a message $m \in \mathcal{M}$, the encryptor "encodes" the given message as an element of $\mathcal{X}$, and then applies the trapdoor function to the encoded element to obtain a ciphertext $c \in \mathcal{Y}$. The decryptor inverts the trapdoor function at c, and decodes the resulting value to obtain the message m.

As a first naive attempt, suppose $\mathcal{X} := \{0,1\}^t$ and $\mathcal{M} := \{0,1\}^s$, where, say, $t = 2048$ and $s = 256$. To encrypt a message $m \in \mathcal{M}$ using the public key pk do

$$E(pk, m) := F\big(pk,\ 0^{t-s} \parallel m\big).$$

Here we pad the message m in $\mathcal{M}$ with zeros so that it is in $\mathcal{X}$. To decrypt a ciphertext c, invert the trapdoor function by computing $I(sk, c)$ and strip off the $(t - s)$ zeros on the left.

This naive scheme uses deterministic encryption and is therefore not even CPA secure. It should never be used. Instead, to build a secure public-key scheme we need a better way to encode the message $m \in \mathcal{M}$ into the domain $\mathcal{X}$ of the trapdoor function. The encoding should be invertible to enable decryption, and should be randomized to have some hope of providing CPA security, let alone CCA security. Towards this goal, let us define the notion of a padding scheme.

Definition 12.8. *A **padding scheme** $\mathcal{PS} = (P, U)$, defined over $(\mathcal{M}, \mathcal{R}, \mathcal{X})$, is a pair of efficient algorithms, P and U, where $P : \mathcal{M} \times \mathcal{R} \to \mathcal{X}$ and $U : \mathcal{X} \to \mathcal{M} \cup \{ \textsf{reject} \}$ is its inverse in the following sense: $U(x) = m$ whenever $x = P(m, r)$ for some $(m, r) \in \mathcal{M} \times \mathcal{R}$, and $U(x) = \textsf{reject}$ if x is not in the image of P.*

For a given padding scheme (P, U) defined over $(\mathcal{M}, \mathcal{R}, \mathcal{X})$, let us define the following public-key encryption scheme $\mathcal{E}_{\text{pad}} = (G, E, D)$ derived from the trapdoor function $\mathcal{T} = (G, F, I)$:

$$\begin{array}{llll} E(pk,m) \ := & & D(sk,c) \ := & \\ & r \overset{\text{R}}{\leftarrow} \mathcal{R}, \ \ x \leftarrow P(m, r), & & x \leftarrow I(sk, c), \\ & c \leftarrow F(pk, x), & & m \leftarrow U(x), \\ & \text{output } c; & & \text{output } m. \end{array} \tag{12.41}$$

When the trapdoor function $\mathcal{T}$ is RSA it will be convenient to call this scheme RSA-$\mathcal{PS}$ encryption. For example, when RSA is coupled with PKCS1 padding we obtain RSA-PKCS1 encryption.

The challenge now is to design a padding scheme $\mathcal{PS}$ for which $\mathcal{E}_{\text{pad}}$ can be proven CCA secure, in the random oracle, under the assumption that $\mathcal{T}$ is one way. Many such padding schemes have been developed with varying properties. In the next subsections we describe several such schemes, their security properties, and limitations.

12.8.2 PKCS1 padding

The oldest padding scheme, which is still in use today, is called PKCS1 padding.

To describe this padding scheme let us assume from now on that the domain $\mathcal{X}$ of the trapdoor function is $0^8 \times \{0,1\}^{t-8}$, where t is a multiple of 8. That is, $\mathcal{X}$ consists of all t-bit strings whose left-most 8 bits are zero. These zero bits are meant to accommodate a t-bit RSA modulus, so that all such strings are binary encodings of numbers that are less than the RSA modulus. The message

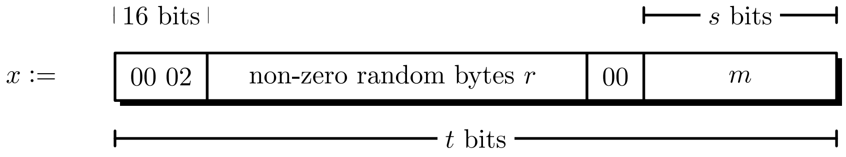


Figure 12.6: PKCS1 padding (mode 2)

space $\mathcal{M}$ consists of all bit strings whose length is a multiple of 8, but at most $t - 88$. The PKCS1 standard is very much byte oriented, which is why all bit strings are multiples of 8. The number 88 is specified in the standard: the message to be encrypted must be at least 11 bytes (88 bits) shorter than the RSA modulus. For an RSA modulus of size 2048 bits, the message can be at most 245 bytes (1960 bits). In practice, messages are often only 32 bytes (256 bits).

The PKCS1 padding algorithm is shown in Fig. 12.6. A double-digit number, like 00 or 02, in the figure denotes a one-byte (8-bit) value in hexadecimal notation. Here, s is the length of the message m. The randomizer r shown in the figure is a sequence of $(t - s)/8 - 3$ random *non-zero* bytes.

The PKCS1 padding scheme (P, U) works as follows. We can take the randomizer space $\mathcal{R}$ to be the set of of all strings r' of non-zero bytes of length $t/8 - 3$; to pad a particular message m, we use a prefix r of r' of appropriate length so that the resulting string x is exactly t-bits long. Here are the details of algorithms P and U.

```
Algorithm $P(m, r')$:
    output $x := \big(00 \parallel 02 \parallel r \parallel 00 \parallel m\big) \in \{0,1\}^t$,
        where $r$ is the appropriate prefix of $r'$

Algorithm $U(x)$:
(1) parse $x$ as $\big(00 \parallel 02 \parallel \text{non-zero bytes } r \parallel 00 \parallel m\big)$
    if $x$ cannot be parsed this way, output reject
    else, output $m$
```

Because the string r contains only non-zero bytes, parsing x in line (1) can be done unambiguously by scanning the string x from left to right. The 16 bits representing 00 02 at the left of the string is the reason why this padding is called PKCS1 mode 2 (mode 1 is discussed in the next chapter).

By coupling PKCS1 padding with RSA, as in (12.41), we obtain the **RSA-PKCS1** encryption scheme. What can we say about the security of RSA-PKCS1? As it turns out, not much. In fact, there is a devastating chosen ciphertext attack on it, which we discuss next.

12.8.3 Bleichenbacher's attack on the RSA-PKCS1 encryption scheme

RSA-PKCS1 encryption is not secure against chosen ciphertext attacks. We describe an attack, due to Bleichenbacher, as it applies to the SSL 3.0 protocol used to establish a secure session between a client and a server. The SSL 3.0 protocol was later replaced by an improved protocol called TLS 1.0 that defends against this attack, as discussed below. The latest version of TLS, called TLS 1.3, has moved away from RSA encryption altogether (see Section 21.10).

The only details of SSL 3.0 relevant to this discussion are the following:

- During session setup, the client chooses a random 48-byte (384-bit) string, called the `pre_master_secret`, and encrypts it with RSA-PKCS1 under the server's public-key. It sends the resulting ciphertext c to the server in a message called `client_key_exchange`.

- When the server receives a `client_key_exchange` message it extracts the ciphertext c and attempts to decrypt it. If PKCS1 decoding returns reject, the server sends an abort message to the client. Otherwise, it continues normally with session setup.

Let us show a significant vulnerability in this system that is a result of a chosen ciphertext attack on RSA-PKCS1. Suppose the attacker has a ciphertext c that it intercepted from an earlier SSL session with the server. This c is an encryption generated using the server's RSA public key (n, e), with RSA modulus n and encryption exponent e. The attacker's goal is to decrypt c. Let x be the eth root of c in $\mathbb{Z}_n$, so that $x^e = c$ in $\mathbb{Z}_n$. We show how the attacker can learn x, which is sufficient to decrypt c.

The attacker's strategy is based on the following observation: let r be some element in $\mathbb{Z}_n$ and define $c' \leftarrow c \cdot r^e$ in $\mathbb{Z}_n$; then

$$c' = c \cdot r^e = (x \cdot r)^e \;\; \in \mathbb{Z}_n.$$

The attacker plays the role of a client and attempts to establish a SSL connection with the server. The attacker creates a `client_key_exchange` message that contains c' as the encrypted `pre_master_secret` and sends the message to the server. The server, following the protocol, computes the eth root of c' to obtain $x' = x \cdot r$ in $\mathbb{Z}_n$. Next, the server checks if x' is a proper PKCS1 encoding: does x' begin with the two bytes 00 02, and if so, is it followed by non-zero bytes, then a zero byte, and then 48 additional (message) bytes? If not, the server sends an abort message to the attacker. Otherwise, decryption succeeds and it sends the next SSL message to the attacker. Consequently, the server's response to the attacker's `client_key_exchange` message reveals some information about $x' = x \cdot r$. It tells the attacker if x' is a valid PKCS1 encoding.

The attacker can repeat this process over and over with different values of $r \in \mathbb{Z}_n$ of its choosing. Every time the attacker learns if $x \cdot r$ is a valid PKCS1 encoding or not. In effect, the server becomes an oracle that implements the following predicate for the attacker:

$$P_x(r) := \begin{cases} 1 & \text{if } x \cdot r \text{ in } \mathbb{Z}_n \text{ is a valid PKCS1 encoding;} \\ 0 & \text{otherwise.} \end{cases}$$

The attacker can query this predicate for any $r \in \mathbb{Z}_n$ of its choice and as many times as it wants.

Bleichenbacher showed that for a 2048-bit RSA modulus, this oracle is sufficient to recover all of x with several million queries to the server. Exercise 12.21 gives a simple example of this phenomenon.

This attack is a classic example of a real-world chosen ciphertext attack. The adversary has a challenge ciphertext c that it wants to decrypt. It does so by creating a number of related ciphertexts and asks the server to "partially decrypt" those ciphertexts (i.e., evaluate the predicate P_x). After enough queries, the adversary is able to obtain the decryption of c. Clearly, this attack would not be possible if RSA-PKCS1 were CCA-secure: CCA security implies that such attacks are not possible even given a *full* decryption oracle, let alone a partial decryption oracle like P_x.

This devastating attack lets the attacker eavesdrop on any SSL session of its choice. Given the wide deployment of RSA-PKCS1 encryption, the question then is how to best defend against this attack.

The TLS defense. When Bleichenbacher's attack was discovered in 1998, there was a clear need to fix SSL. Moving away from PKCS1 to a completely different padding scheme would have been difficult since it would have required updating both clients and servers, and this can take decades for everyone to update. The challenge was to find a solution that requires only server-side changes, so that deployment can be done server-side only. This will protect all clients, old and new, connecting to an updated server.

The solution, implemented in TLS 1.0, changes the RSA-PKCS1 server-side decryption process to the following procedure:

```
1.   generate a string $r$ of 48 random bytes,
2.   decrypt the RSA-PKCS1 ciphertext to recover the plaintext $m$,
3.   if the PKCS1 padding is invalid, or the length of $m$ is not exactly 48 bytes:
4.        set $m \leftarrow r$
5.   return $m$
```

In other words, when PKCS1 parsing fails, simply choose a random plaintext r and use this r as the decrypted value. Clearly, the TLS session setup will fail further down the line and setup will abort, but presumably doing so at that point reveals no useful information about the decryption of c. Some justification for this process is provided by Jonsson and Kaliski [95]. The TLS 1.2 standard goes further and includes the following warning about this decryption process:

> In any case, a TLS server MUST NOT generate an alert if processing an RSA-encrypted pre-master secret message fails [...] Instead, it MUST continue the handshake with a randomly generated pre-master secret. It may be useful to log the real cause of failure for troubleshooting purposes; however, care must be taken to avoid leaking the information to an attacker (through, e.g., timing, log files, or other channels.)

Note the point about side channels, such as timing attacks, in the last sentence. Suppose the server takes a certain amount of time to respond to a `client_key_exchange` message when the PKCS1 padding is valid, and a different amount of time when it is invalid. Then by measuring the server's response time, the Bleichenbacher attack is easily made possible again.

The DROWN attack. To illustrate the cost of cryptographic mistakes, we mention an interesting attack called DROWN [9]. While implementations of TLS 1.0 and above are immune to Bleichenbacher's attack, a very old version of the protocol, called SSL 2.0, is still vulnerable. SSL 2.0 is still supported by some Internet servers so that old clients can connect. The trouble is that, in a common TLS deployment, the server has only one TLS public-key pair. The *same* public key is used to establish a session when the latest version of TLS is used, as when the old SSL 2.0 is used. As a result, an attacker can record the ciphertext c used in a TLS 1.2 session, encrypted under the server's public key, and then use Bleichenbacher's attack on the SSL 2.0 implementation to decrypt this c. This lets the attacker decrypt the TLS session, despite the fact that TLS is immune to Bleichenbacher's attack. Effectively, the old SSL 2.0 implementation compromises the modern TLS.

This attack shows that once a cryptographically flawed protocol is deployed, it is very difficult to get rid of it. Even more troubling is that flaws in a protocol can be used to attack later versions of the protocol that have supposedly corrected those flaws. The lesson is: make sure to get the cryptography right the first time. The best way to do that is to only use schemes that have been properly analyzed.

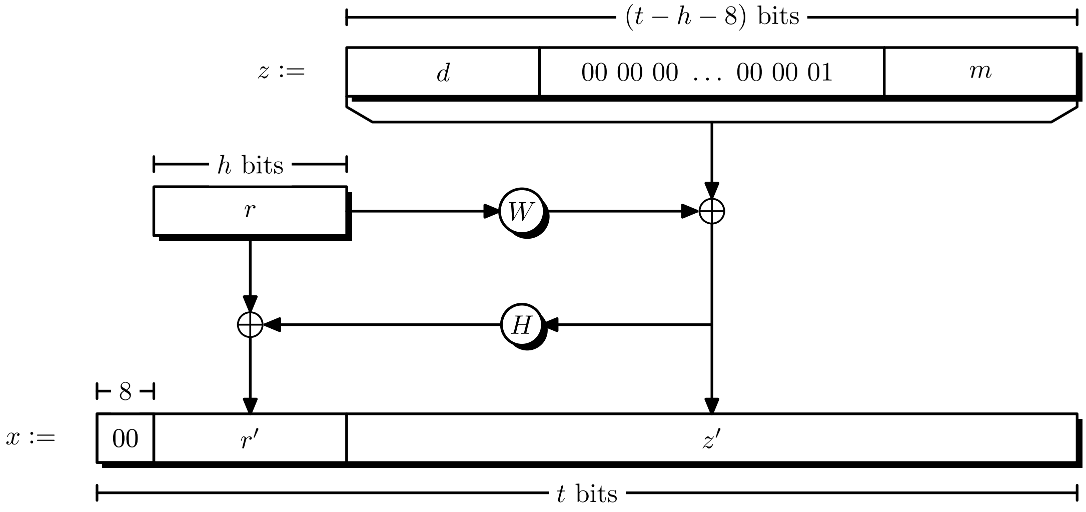


Figure 12.7: OAEP padding using hash functions H and W, and optional associated data d

12.8.4 Optimal Asymmetric Encryption Padding (OAEP)

The failure of RSA-PKCS1 leaves us with the original question: is there a padding scheme (P, U) so that the resulting encryption scheme $\mathcal{E}_{\text{pad}}$ from (12.41) can be shown to be CCA-secure, in the random oracle model, based on the one-wayness of the trapdoor function?

The answer is yes, and the first attempt at such a padding scheme was proposed by Bellare and Rogaway in 1994. This padding, is called Optimal Asymmetric Encryption Padding (OAEP), and the derived public-key encryption scheme was standardized in the PKCS1 version 2.0 standard. It is called "optimal" because the ciphertext is a single element of $\mathcal{Y}$, and nothing else.

The OAEP padding scheme (P, U) is defined over $(\mathcal{M}, \mathcal{R}, \mathcal{X})$, where $\mathcal{R} := \{0,1\}^h$ and $\mathcal{X} := 0^8 \times \{0,1\}^{t-8}$. As usual, we assume that h and t are multiples of eight so that lengths can be measured in bytes. As before, in order to accommodate a t-bit RSA modulus, we insist that the left-most 8 bits of any element in $\mathcal{X}$ are zero. The message space $\mathcal{M}$ consists of all bit strings whose length is a multiple of 8, but at most $t - 2h - 16$.

The scheme also uses two hash functions H and W, where

$$H : \{0,1\}^{t-h-8} \to \mathcal{R}\ , \qquad W : \mathcal{R} \to \{0,1\}^{t-h-8}. \tag{12.42}$$

The set $\mathcal{R}$ should be sufficiently large to be the range of a collision resistant hash. Typically, SHA256 is used as the function H and we set $h := 256$. The function W is derived from SHA256 (see Section 8.10.3 for recommended derivation techniques).

OAEP padding is used to build a public-key encryption scheme with associated data (as discussed in Section 12.7). As such, the padding algorithm P takes an optional third argument $d \in \mathcal{R} = \{0,1\}^h$, representing the associated data. To support associated data that is more than h bits long one can first hash the associated data using a collision resistant hash to obtain an element of $\mathcal{R}$. If no associated data is provided as input to P, then d is set to a constant that identifies the hash function H, as specified in the standard. For example, for SHA256, one sets d to the following

256-bit hex value:

$$d := \texttt{E3B0C442 98FC1C14 9AFBF4C8 996FB924 27AE41E4 649B934C A495991B 7852B855}.$$

Algorithm $P(m, r, d)$ is shown in Fig. 12.7. Every pair of digits in the figure represents one byte (8 bits). The variable length string of zeros in z is chosen so that the total length of z is exactly $(t - h - 8)$ bits. The algorithm outputs an $x \in \mathcal{X}$.

The inverse algorithm U, on input $x \in \mathcal{X}$ and $d \in \mathcal{R}$, is defined as follows:

```
      parse $x$ as $(00 \parallel r' \parallel z')$ where $r' \in \mathcal{R}$ and $z' \in \{0,1\}^{t-h-8}$
(1)   if $x$ cannot be parsed this way, set $m \leftarrow \mathsf{reject}$
      else
          $r \leftarrow H(z') \oplus r', \quad z \leftarrow W(r) \oplus z'$
          parse $z$ as $(d \parallel 00 \ \ldots \ 00\ 01 \parallel m)$ where $d \in \mathcal{R}$ and $m \in \mathcal{M}$
          if $z$ cannot be parsed this way, set $m \leftarrow \mathsf{reject}$
      output $m$
```

Finally, the public-key encryption scheme RSA-OAEP is obtained by combining the RSA trapdoor function with the OAEP padding scheme, as in (12.41). When referring to OAEP coupled with a general trapdoor function $\mathcal{T} = (G, F, I)$, we denote the resulting encryption scheme by $\mathcal{E}_{\text{OAEP}} = (G, E, D)$.

The security of $\mathcal{E}_{\text{OAEP}}$. One might hope to prove CCA security of $\mathcal{E}_{\text{OAEP}}$ in the random oracle model using only the assumption that $\mathcal{T}$ is one-way. Unfortunately, that is unlikely because of a counter-example: there is a plausible trapdoor function $\mathcal{T}$ for which the resulting $\mathcal{E}_{\text{OAEP}}$ is vulnerable to a CCA attack. See Exercise 12.23.

Nevertheless, it is possible to prove security of $\mathcal{E}_{\text{OAEP}}$ by making a stronger one-wayness assumption about $\mathcal{T}$, called partial one-wayness. Recall that in the game defining a one-way function, the adversary is given pk and $y \leftarrow F(pk, x)$, for some pk and random $x \in \mathcal{X}$, and is asked to produce x. In the game defining a partial one-way function, the adversary is given pk and y, but is only asked to produce, say, certain bits of x. If no efficient adversary can accomplish even this simpler task, then we say that $\mathcal{T}$ is partial one-way. More generally, instead of producing some bits of x, the adversary is asked to produce a particular function f of x. This is captured in the following game.

Attack Game 12.7 (Partial one-way trapdoor function scheme). For a given trapdoor function scheme $\mathcal{T} = (G, F, I)$, defined over $(\mathcal{X}, \ \mathcal{Y})$, a given efficiently computable function $f : \mathcal{X} \to \mathcal{Z}$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger computes

  $$(pk, sk) \xleftarrow{\text{R}} G(), \quad x \leftarrow \mathcal{X}, \quad y \leftarrow F(pk, x)$$

  and sends (pk, y) to the adversary.

- The adversary outputs $\hat{z} \in \mathcal{Z}$.

We define the adversary's advantage, denoted $\text{POW}\mathsf{adv}[\mathcal{A}, \mathcal{T}, f]$, to be the probability that $\hat{z} = f(x)$. $\square$

Definition 12.9. *We say that a trapdoor function scheme $\mathcal{T}$ defined over $\big(\mathcal{X},\ \mathcal{Y}\big)$ is* ***partial one way with respect to*** *$f : \mathcal{X} \to \mathcal{Z}$ if, for all efficient adversaries $\mathcal{A}$, the quantity $\mathrm{POW}\mathsf{adv}[\mathcal{A}, \mathcal{T}, f]$ is negligible.*

Clearly, a partial one-way trapdoor function is also a one-way trapdoor function: if an adversary can recover x it can also recover $f(x)$. Therefore, the assumption that a trapdoor function is partial one way is at least as strong as assuming that the trapdoor function is one way.

The following theorem, due to Fujisaki, Okamoto, Pointcheval, and Stern, shows that $\mathcal{E}_{\mathrm{OAEP}}$ is CCA-secure in the random oracle model, assuming $\mathcal{T}$ is partial one-way. The proof can be found in their paper [66].

Theorem 12.13. *Let t, h, $\mathcal{X}$, H, and W be as in the OAEP construction. Assume H and W are modeled as random oracles. Let $\mathcal{T} = (G, F, I)$ be a trapdoor function defined over $\big(\mathcal{X}, \mathcal{Y}\big)$. Let $f : \mathcal{X} \to \{0,1\}^{t-h-8}$ be the function that returns the right-most $(t - h - 8)$ bits of its input. If $\mathcal{T}$ is partial one way with respect to f, and 2^h is super-poly, then $\mathcal{E}_{\mathrm{OAEP}}$ is CCA secure.*

Given Theorem 12.13 the question is then: is RSA a partial one-way function? We typically assume RSA is one-way, but is it partial one-way when the adversary is asked to compute only $(t - h - 8)$ bits of the pre-image? As it turns out, if RSA is one-way then it is also partial one-way. More precisely, suppose there is an efficient adversary $\mathcal{A}$ that given an RSA modulus n and encryption exponent e, along with $y \leftarrow x^e \in \mathbb{Z}_n$ as input, outputs more than half the least significant bits of x. Then there is an efficient adversary $\mathcal{B}$ that uses $\mathcal{A}$ and recovers all the bits of x. See Exercise 12.24.

As a result of this wonderful fact, we obtain as a corollary of Theorem 12.13 that RSA-OAEP is CCA-secure in the random oracle model assuming only that RSA is a one-way function. However, the concrete security bounds obtained when proving CCA security of RSA-OAEP based on the one-wayness of RSA are quite poor.

Manger's timing attack. RSA-OAEP is tricky to implement securely. Suppose the OAEP algorithm $U(x, d)$ were implemented so that it takes a certain amount of time when the input is rejected because of the test on line (1), and a different amount of time when the test succeeds. Notice that rejection on line (1) occurs when the eight most significant bits of x are not all zero. Now, consider again the setting of Bleichenbacher's attack on PKCS1. The adversary has a ciphertext c, generated using the server's RSA public key, with RSA modulus n and encryption exponent e. The adversary wants to decrypt c. It can repeatedly interact with the server, sending it $c' \leftarrow c \cdot r^e$ in $\mathbb{Z}_n$, for various values of r of the adversary's choice. By measuring the time that the server takes to respond, the attacker can tell if rejection happened because of line (1). Therefore, the attacker learns if the eight most significant bits of $(c')^{1/e}$ in $\mathbb{Z}_n$ are all zero. As in Bleichenbacher's attack, this partial decryption oracle is sufficient to decrypt all of c. See Exercise 12.21, or Manger [106], for the full details.

12.8.5 OAEP+ and SAEP+

In the previous section we saw that RSA-OAEP is CCA-secure assuming RSA is a one-way function. However, for other one-way trapdoor functions, the derived scheme $\mathcal{E}_{\mathrm{OAEP}}$ may not be CCA-secure.

The next question is then: is there a padding scheme (P, U) that, when coupled with a general trapdoor function, gives a CCA-secure scheme in the random oracle model? The answer is yes, and a padding scheme that does so, called OAEP+, is a variation of OAEP [147]. The difference, essentially, is that the block of zero bytes in Fig. 12.7 is replaced with the value $H'(m, r)$ for some hash function H'. This block is verified during decryption by recomputing $H'(m, r)$ from the recovered values for m and r. The ciphertext is rejected if the wrong value is found in this block.

For RSA specifically, it is possible to use a simpler CCA-secure padding scheme. This simpler padding scheme, called SAEP+, eliminates the hash function H and the corresponding xor on the left of H in Fig. 12.7. The randomizer r needs to be longer than in OAEP. Specifically, r must be slightly longer than half the size of the modulus, that is, slightly more than $t/2$ bits. RSA-SAEP+ is CCA-secure, in the random oracle model, assuming the RSA function is one-way [28]. It provides a simple alternative padding scheme for RSA.

12.9 A fun application: private set intersection

This application has little to do with CCA security, but rather, is a continuation of the fun application in the previous chapter (see Section 11.6).

Alice and Bob are sick: they both caught the flu. Alice has a list of people $S_{\mathrm{a}} = \{u_1, \ldots, u_n\} \subseteq \mathcal{ID}$ that she recently came in contact with. Similarly, Bob has a list of people $S_{\mathrm{b}} \subseteq \mathcal{ID}$ that he recently came in contact with. They want to identify the subset of people that they both came in contact with, namely the people in $S_{\mathrm{a}} \cap S_{\mathrm{b}}$, who could have been the source of the flu. The problem is that Bob does not want to reveal his contacts S_{b} to Alice, and similarly, Alice does not want to reveal her contacts S_{a} to Bob. How can they compute the intersection?

This problem is called *private set intersection*: each party should learn the items in the intersection of S_{a} and S_{b}, but nothing else should be revealed about the sets. An elegant solution uses a mechanism called an *oblivious PRF* that we previously discussed in Section 11.6.3. Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{ID}, \mathcal{Y})$, where $\mathcal{Y} = \{0, 1\}^b$ for, say, $b = 256$. Suppose Alice has some $u \in \mathcal{ID}$, and Bob has a random $k \in \mathcal{K}$. Recall that an oblivious PRF evaluation is a protocol between Alice and Bob, where at the end of the protocol Alice learns $y := F(k, u)$, but Bob learns nothing about u, and Alice learns nothing else about k. In Section 11.6.3 we saw a simple construction for an oblivious PRF based on the one-more Diffie-Hellman assumption in the random oracle model.

Now, suppose Alice has $S_{\mathrm{a}} = \{u_1, \ldots, u_n\} \subseteq \mathcal{ID}$, and Bob has $S_{\mathrm{b}} = \{v_1, \ldots, v_m\} \subseteq \mathcal{ID}$. To compute the intersection $S_{\mathrm{a}} \cap S_{\mathrm{b}}$, they decide to use the following protocol:

– *step 1:* Bob chooses $k \xleftarrow{\mathrm{R}} \mathcal{K}$.

– *step 2:* Alice and Bob run the oblivious PRF protocol n times, once for each element in S_{a}. Alice learns $\hat{u}_i := F(k, u_i)$ for $i = 1, \ldots, n$.

– *step 3:* Bob computes $\hat{v}_j := F(k, v_j)$ for all $j = 1, \ldots, m$ and sends $\hat{v}_1, \ldots, \hat{v}_m \in \mathcal{Y}$ to Alice.

– *step 4:* Alice finds all $u \in S_{\mathrm{a}}$ such that $\hat{u} := F(k, u)$ is in $\{\hat{v}_1, \ldots, \hat{v}_m\}$. She outputs the set of all such u as the intersection $S_{\mathrm{a}} \cap S_{\mathrm{b}}$.

Clearly, if Alice and Bob honestly follow the protocol, then once the protocol terminates, Alice learns the intersection $S_{\mathrm{a}} \cap S_{\mathrm{b}}$. Alice can then send the intersection to Bob. The running time for each party is linear in $|S_{\mathrm{a}}| + |S_{\mathrm{b}}|$, as is the total communication.

Notice that the protocol leaks the size of the sets: Bob learns the size of S_{a}, and Alice learns the size of S_{b}. If needed, this leakage can be prevented by first padding the sets to a maximum size using distinct unused dummy elements.

Let's see why the protocol reveals nothing beyond the intersection and the size of the sets. First, thanks to the oblivious property of the evaluation protocol, Bob learns nothing about S_{a} beyond its size. Second, Alice learns nothing about S_{b} beyond its size and the elements in the intersection. To see why, observe that for each $v \in S_{\mathrm{b}}$ that is not in the intersection, Alice learns $\hat{v} := F(k, v)$. Because F is a secure PRF, and Alice knows nothing about k, this $\hat{v}$ is indistinguishable from a random element in $\mathcal{Y}$ that is independent of v. Hence, Alice learns nothing about elements $v \in S_{\mathrm{b}}$ that are outside of the intersection.

A note on security. The above protocol is only secure when both Alice and Bob actually follow the protocol — this is the so-called "honest but curious" model. There are protocols for this problem that remain secure even for arbitrarily malicious parties, but they are more complicated.

12.10 Notes

Citations to the literature to be added.

12.11 Exercises

12.1 (Insecurity of multiplicative ElGamal). Show that multiplicative ElGamal from Exercise 11.5 is not CCA secure. Your adversary should have an advantage of 1 in the 1CCA attack game.

12.2 (Sloppy CCA). Let $\mathcal{E} = (G, E, D)$ be a CCA-secure public-key encryption scheme defined over $(\mathcal{M}, \mathcal{C})$ where $\mathcal{C} := \{0,1\}^{\ell}$. Consider the encryption scheme $\mathcal{E}' = (G, E', D')$ defined over $(\mathcal{M}, \mathcal{C}')$ where $\mathcal{C} := \{0,1\}^{\ell+1}$ as follows:

$$E'(pk, m) := E(pk, m) \parallel 0 \quad \text{and} \quad D'(sk, c) := D(sk, c[0 \mathinner{..} \ell - 1]).$$

That is, the last ciphertext bit can be 0 or 1, but the decryption algorithm ignores this bit. Show that $\mathcal{E}'$ is not CCA secure. Your adversary should have an advantage of 1 in the 1CCA attack game.

Discussion: Clearly, adding a bit to the ciphertext does not harm security in practice, yet it breaks CCA security of the scheme. This issue suggests that the definition of CCA security may be too strong. A different notion, called **generalized CCA** (gCCA), weakens the definition of CCA security so that simple transformations of the ciphertext, like the one in $\mathcal{E}'$, do not break gCCA security. More formally, we assume that for each key pair (pk, sk), there is an equivalence relation $\equiv_{pk}$ on ciphertexts such that

$$c \equiv_{pk} c' \implies D(sk, c) = D(sk, c').$$

Moreover, we assume that given pk, c, c', it is easy to tell if $c \equiv_{pk} c'$. Note that the relation $\equiv_{pk}$ is specific to the particular encryption scheme. Then, in Attack Game 12.1, we insist each decryption query is not equivalent to (as opposed to not equal to) any ciphertext arising from a previous encryption query.

12.3 (Small subgroup attack). We mentioned in Remark 12.1 that the decryption algorithm for $\mathcal{E}_{\mathrm{EG}}$ should verify that in a given ciphertext (v, c), the element v actually belongs to the group $\mathbb{G}$. This exercise illustrates why this is important. Suppose that $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$ of prime order q, where p is prime. We assume that the ICDH assumption holds for $\mathbb{G}$. Suppose that the decryption algorithm checks that $v \in \mathbb{Z}_p^*$ (which is typically quite trivial to do), but does not check that $v \in \mathbb{G}$ (which can be more costly). In particular, the decryption algorithm just computes $w \leftarrow v^\alpha \in \mathbb{Z}_p^*$ and uses v, w, c to decrypt the given ciphertext. Here, we treat α as an integer in the range $[0, q)$, rather than an element of $\mathbb{Z}_q$. We also view H as a function $H : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \mathcal{K}$.

Suppose $p-1$ can be written as a product $p-1 = q \cdot t_1 \cdots t_r$, where $q, t_1, \ldots, t_r$ are distinct primes, and each t_i is poly-bounded. Show that it is possible to completely recover the secret key via a chosen ciphertext attack. The number of decryption queries and the computation time of the adversary in this attack is poly-bounded and its success probability is $1 - \epsilon$, where ϵ is negligible. To simplify the analysis of your adversary's success probability, you may model $H : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \mathcal{K}$ as a random oracle and assume that the symmetric cipher provides one-time ciphertext integrity.

Hint: Use the fact that for each $i = 1, \ldots, t$, you can efficiently find an element $g_i \in \mathbb{Z}_p^*$ of order t_i. Use this g_i to learn $\alpha \bmod t_i$.

12.4 (Extending the message space). Continuing with Exercise 11.7. Show that even if $\mathcal{E}$ is CCA secure, $\mathcal{E}^2$ is not CCA secure. For this, you should assume $\mathcal{M}$ is non-trivial (i.e., contains at least two messages of the same length).

Note: The next exercise presents a correct way to extend the message space of a CCA-secure encryption scheme.

12.5 (Modular hybrid construction). All of the public-key encryption schemes presented in this chapter can be viewed as special cases of the general hybrid construction introduced in Exercise 11.9.

Consider a KEM $\mathcal{E}_{\mathrm{kem}} = (G, E_{\mathrm{kem}}, D_{\mathrm{kem}})$, defined over $(\mathcal{K}, \mathcal{C}_{\mathrm{kem}})$. We define 1CCA security for $\mathcal{E}_{\mathrm{kem}}$ in terms of an attack game, played between a challenger and an adversary $\mathcal{A}$, as follows. In Experiment b, for $b = 0, 1$, the challenger first computes

$$(pk, sk) \xleftarrow{\mathrm{R}} G(), \;\; (k_0, c_{\mathrm{kem}}) \xleftarrow{\mathrm{R}} E_{\mathrm{kem}}(pk), \;\; k_1 \xleftarrow{\mathrm{R}} \mathcal{K},$$

and sends (k_b, c_{kem}) to $\mathcal{A}$. Next, the adversary submits a sequence of decryption queries to the challenger. Each such query is of the form $\hat{c}_{\mathrm{kem}} \in \mathcal{C}_{\mathrm{kem}}$, subject to the constraint that $\hat{c}_{\mathrm{kem}} \neq c_{\mathrm{kem}}$, to which the challenger responds with $D_{\mathrm{kem}}(sk, \hat{c}_{\mathrm{kem}})$. Finally, $\mathcal{A}$ outputs $\hat{b} \in \{0, 1\}$. As usual, if W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}_{\mathrm{kem}}$ as $1\mathrm{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{kem}}] := |\Pr[W_0] - \Pr[W_1]|$, and if this advantage is negligible for all efficient adversaries, we say that $\mathcal{E}_{\mathrm{kem}}$ is **1CCA secure**.

If $\mathcal{E}_{\mathrm{s}}$ is a symmetric cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, then as in Exercise 11.9, we also consider the hybrid public-key encryption scheme $\mathcal{E} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C}_{\mathrm{kem}} \times \mathcal{C})$, constructed out of $\mathcal{E}_{\mathrm{kem}}$ and $\mathcal{E}_{\mathrm{s}}$.

(a) Prove that $\mathcal{E}$ is CCA secure, assuming that $\mathcal{E}_{\mathrm{kem}}$ and $\mathcal{E}_{\mathrm{s}}$ are 1CCA secure. You should prove a concrete security bound that says that for every adversary $\mathcal{A}$ attacking $\mathcal{E}$, there are adversaries $\mathcal{B}_{\mathrm{kem}}$ and $\mathcal{B}_{\mathrm{s}}$ (which are elementary wrappers around $\mathcal{A}$) such that

$$1\mathrm{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq 2 \cdot 1\mathrm{CCA}\mathsf{adv}[\mathcal{B}_{\mathrm{kem}}, \mathcal{E}_{\mathrm{kem}}] + 1\mathrm{CCA}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}].$$

Discussion: Using this result, one can arbitrarily extend the message space of any CCA-secure encryption scheme whose message space is already large enough to contain the key space for a 1CCA-secure symmetric cipher. For example, in practice, a 128-bit message space suffices. Interestingly, one can arbitrarily extend the message space even when starting from a CCA-secure scheme for 1-bit messages [118, 89].

(b) Describe the KEM corresponding to $\mathcal{E}'_{\mathrm{TDF}}$ in Section 12.3 and prove that it is 1CCA secure (in the random oracle model, assuming $\mathcal{T}$ is one way given an image oracle).

(c) Describe the KEM corresponding to $\mathcal{E}_{\mathrm{EG}}$ in Section 12.4 and prove that it is 1CCA secure (in the random oracle model, under the ICDH assumption for $\mathbb{G}$).

(d) Describe the KEM corresponding to $\mathcal{E}_{\mathrm{CS}}$ in Section 12.5 and prove that it is 1CCA secure (assuming the DDH, H is a secure KDF, and H' is collision resistant).

(e) Give examples that show that if one of $\mathcal{E}_{\mathrm{kem}}$ and $\mathcal{E}_{\mathrm{s}}$ is 1CCA secure, while the other is only semantically secure, then $\mathcal{E}$ need not be CCA secure.

(f) Let $\mathcal{E}_{\mathrm{a}}$ be a public-key encryption scheme. Consider the KEM $\mathcal{E}_{\mathrm{kem}}$ constructed out of $\mathcal{E}_{\mathrm{a}}$ as in part (e) of Exercise 11.9. Show that $\mathcal{E}_{\mathrm{kem}}$ is 1CCA secure, assuming that $\mathcal{E}_{\mathrm{a}}$ is 1CCA secure.

(g) Assume $\mathcal{E}_{\mathrm{kem}}$ is a 1CCA-secure KEM. Assume $\mathcal{E}_{\mathrm{s}}$ is a 1CCA-secure AD cipher (see Section 9.6). Suppose we modify the hybrid public-key encryption scheme $\mathcal{E}$ from Exercise 11.9 so that it supports associated data, where the associated data is simply passed through to the symmetric AD cipher. Show that the resulting scheme is a 1CCA-secure AD public-key encryption.

12.6 (Multi-key CCA security). Generalize the definition of CCA security for a public-key encryption scheme to the multi-key setting. In this attack game, the adversary gets to obtain encryptions of many messages under many public keys, and can make as decryption queries with respect to any of these keys. Show that 1CCA security implies multi-key CCA security. You should show that security degrades linearly in $Q_{\mathrm{k}}Q_{\mathrm{e}}$, where Q_{k} is a bound on the number of keys, and Q_{e} is a bound on the number of encryption queries per key. That is, the advantage of any adversary $\mathcal{A}$ in breaking the multi-key CCA security of a scheme is at most $Q_{\mathrm{k}}Q_{\mathrm{e}} \cdot \epsilon$, where ϵ is the advantage of an adversary $\mathcal{B}$ (which is an elementary wrapper around $\mathcal{A}$) that breaks the scheme's 1CCA security.

12.7 (Multi-key CCA security of ElGamal). Consider a slight modification of the public-key encryption scheme $\mathcal{E}_{\mathrm{EG}}$, which was presented an analyzed in Section 12.4. This new scheme, which we call $x\mathcal{E}_{\mathrm{EG}}$, is exactly the same as $\mathcal{E}_{\mathrm{EG}}$, except that instead of deriving the symmetric key as $k = H(v, w)$, we derive it as $k = H(u, v, w)$. Consider the security of $x\mathcal{E}_{\mathrm{EG}}$ in the multi-key CCA attack game, discussed above in Exercise 12.6. In that attack game, suppose Q_{te} is a bound on the total number of encryptions — clearly, Q_{te} is at most $Q_{\mathrm{k}}Q_{\mathrm{e}}$, but it could be smaller. Let $\mathcal{A}$ be an adversary that attacks the multi-key CCA security of $x\mathcal{E}_{\mathrm{EG}}$. Show that $\mathcal{A}$'s advantage is at most

$$2\epsilon_{\mathrm{icdh}} + Q_{\mathrm{te}} \cdot \epsilon_{\mathrm{s}},$$

where ϵ_{icdh} is that advantage of an ICDH adversary $\mathcal{B}_{\mathrm{icdh}}$ attacking $\mathbb{G}$ and ϵ_{s} is the advantage of a 1CCA adversary $\mathcal{B}_{\mathrm{s}}$ attacking $\mathcal{E}_{\mathrm{s}}$ (where both $\mathcal{B}_{\mathrm{icdh}}$ and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$).

Hint: Use the random self reduction for CDH (see Exercise 10.5).

12.8 (Fujisaki-Okamoto with verifiable ciphertexts). Consider the Fujisaki-Okamoto transformation presented in Section 12.6. Suppose that the asymmetric cipher $\mathcal{E}_{\rm a}$ has *verifiable ciphertexts*, which means that there is an efficient algorithm that given a public key pk, along with $x \in \mathcal{X}$ and $y \in \mathcal{Y}$, determines whether or not y is an encryption of x under pk. Under this assumption, improve the security bound (12.32) to

$$\mathrm{IOW}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{T}_{\mathrm{FO}}] \leq Q_{\mathrm{io}} \cdot \epsilon + \mathrm{OW}\mathsf{adv}[\mathcal{B}, \mathcal{E}_{\mathrm{a}}].$$

Notice that this bound does not degrade as $Q_{\rm ro}$ grows.

12.9. Show that any semantically secure public-key encryption scheme with a super-poly-sized message space is one way (as in Definition 12.5).

12.10 (Any cipher can be made unpredictable). Let $(G_{\rm a}, E_{\rm a}, D_{\rm a})$ be a public key encryption scheme with message space $\mathcal{X}$, ciphertext space $\mathcal{Y}$, and randomizer space $\mathcal{R}$. Let $\mathcal{S}$ be some super-poly-sized finite set. Consider the encryption scheme $(G_{\rm a}, E'_{\rm a}, D'_{\rm a})$, with message space $\mathcal{X}$, ciphertext space $\mathcal{Y} \times \mathcal{S}$, and randomizer space $\mathcal{R} \times \mathcal{S}$, where $E'_{\rm a}(pk, x; (r, s)) := (E_{\rm a}(pk, x; r), s)$ and $D'_{\rm a}(sk, (y, s)) := D_{\rm a}(sk, y)$. Show that $(G_{\rm a}, E'_{\rm a}, D'_{\rm a})$ is unpredictable (as in Definition 12.6). Also show that if $(G_{\rm a}, E_{\rm a}, D_{\rm a})$ is one way (as in Definition 12.5), then so is $(G_{\rm a}, E'_{\rm a}, D'_{\rm a})$.

12.11 (Fujisaki-Okamoto with semantically secure encryption). Consider the Fujisaki-Okamoto transformation presented in Section 12.6. Suppose that the asymmetric cipher $\mathcal{E}_{\rm a}$ is semantically secure. Under this assumption, improve the security bound (12.32) to

$$\mathrm{IOW}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{T}_{\mathrm{FO}}] \leq Q_{\mathrm{io}} \cdot \epsilon + \mathrm{SS}\mathsf{adv}[\mathcal{B}, \mathcal{E}_{\mathrm{a}}] + Q_{\mathrm{ro}}/|\mathcal{X}|.$$

12.12 (Analysis of a more general version of Fujisaki-Okamoto). This exercise develops an analysis of a slightly more general version of the Fujisaki-Okamaoto transform in which we allow the value $x \in \mathcal{X}$ to be chosen from some arbitrary distribution P on $\mathcal{X}$. We assume that there is an efficient, probabilistic algorithm that samples elements of $\mathcal{X}$ according to P.

(a) Suppose that in Attack Game 12.2, the value $x \in \mathcal{X}$ is sampled according to P. Show that Theorem 12.2 still holds.

(b) Suppose that in Attack Game 12.6, the value $x \in \mathcal{X}$ is sampled according to P. Show that Theorem 12.10 still holds.

12.13 (Subgroup membership checks for $\mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}}$). This exercise justifies the claim made in Remark 12.4. Consider the concrete instantiation $\mathcal{E}_{\mathrm{FO}}^{\mathrm{EG}}$ of Fujisaki-Okamoto using the multiplicative ElGamal encryption scheme over a group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. Let us assume that $\mathbb{G}$ is a subgroup of some larger group $\mathbb{G}'$. For example, we might have $\mathbb{G}' = \mathbb{Z}_p^*$. The point is, checking membership in $\mathbb{G}'$ may be much cheaper that checking membership in $\mathbb{G}$. Now consider a variant of the multiplicative ElGamal encryption scheme, where the plaintext space is $\mathbb{G}'$ and the ciphertext space is $\mathbb{G} \times \mathbb{G}'$.

(a) Show that if the plaintext x is sampled uniformly over $\mathbb{G}$, then this ElGamal variant is one-way under the CDH, using the generalized notion of one-way as discussed in part (b) of the previous exercise (using the uniform distribution over $\mathbb{G}$ rather than over the entire plaintext space $\mathbb{G}'$).

(b) Show that this ElGamal variant is still $1/q$-unpredictable.

(c) Using part (b) of the previous exercise, show that if we instantiate Fujisaki-Okamoto with this ElGamal variant, Theorem 12.12 still holds.

Discussion: This exercise shows that while $\mathcal{E}_{\rm FO}^{\rm EG}$ decryption should check that v and y are in $\mathbb{G}'$, it need not explicitly check that they are in $\mathbb{G} \subseteq \mathbb{G}'$. As discussed in Exercise 15.1, the check that v and y are in $\mathbb{G}'$ is vitally important, as otherwise, a CCA attack could result in key exposure.

12.14 (An analysis of $\mathcal{E}'_{\rm TDF}$ without image oracles). Theorem 12.2 shows that $\mathcal{E}'_{\rm TDF}$ is CCA-secure assuming the trapdoor function scheme $\mathcal{T}$ is one-way given access to an image oracle, and $\mathcal{E}_{\rm s}$ is 1CCA secure. It is possible to prove security of $\mathcal{E}'_{\rm TDF}$ assuming only that $\mathcal{T}$ is one-way (i.e., without assuming it is one-way given access to an image oracle), provided that $\mathcal{E}_{\rm s}$ is 1AE secure (see Section 9.1.1). Note that we are making a slightly stronger assumption about $\mathcal{E}_{\rm s}$ (1AE instead of 1CCA), but prove security under a weaker assumption on $\mathcal{T}$. Prove the following statement: if $H : \mathcal{X} \to \mathcal{K}$ is modeled as a random oracle, $\mathcal{T}$ is one-way, and $\mathcal{E}_{\rm s}$ is 1AE secure, then $\mathcal{E}'_{\rm TDF}$ is CCA secure.

Hint: The proof is similar to the proof of Theorem 12.2. Let $(\hat{y}, \hat{c})$ be a decryption query from the adversary where $\hat{y} \neq y$. If $\mathcal{E}_{\rm s}$ provides ciphertext integrity, then in testing whether $\hat{y}$ is in the image of $F(pk, \cdot)$, we can instead test if the adversary queried the random oracle at a preimage $\hat{x}$ of $\hat{y}$. If not, we can safely reject the ciphertext — ciphertext integrity implies that the original decryption algorithm would have anyway rejected the ciphertext with overwhelming probability.

Discussion: The analysis in this exercise requires that when a ciphertext (y, c) fails to decrypt, the adversary does not learn why. In particular, the adversary must not learn if decryption failed because the inversion of y failed, or because the symmetric decryption of c failed. This means, for example, if the time to decrypt is not the same in both cases, and this discrepancy is detectable by the adversary, then the analysis in this exercise no longer applies. By contrast, the analysis in Theorem 12.2 is unaffected by this side-channel leak: the adversary is given an image oracle and can determine, by himself, the reason for a decryption failure. In this respect, the analysis of Theorem 12.2 is more robust to side-channel attacks and is the preferable way to think of this system.

12.15 (Immunizing against image queries). Let (G, F, I) be a trapdoor function scheme defined over $(\mathcal{X}, \mathcal{Y})$. Let $U : \mathcal{X} \to \mathcal{R}$ be a hash function. Consider the trapdoor function scheme (G, F', I') defined over $(\mathcal{X}, \mathcal{Y} \times \mathcal{R})$, where $F'(pk, x) := (F(pk, x), U(x))$ and $I'(sk, (y, r)) := I(sk, y)$. Show that if U is modeled as a random oracle, (G, F, I) is one way, and $|\mathcal{R}|$ is super-poly, then (G, F', I') is one way given an image oracle.

12.16 (A broken CPA to CCA transformation). Consider the following attempt at transforming a CPA-secure scheme to a CCA-secure one. Let (G, E, D) be a CPA-secure encryption scheme defined over $(\mathcal{K} \times \mathcal{M},\ \mathcal{C})$, and let (S, V) be a secure MAC with key space $\mathcal{K}$. We construct a new encryption scheme (G, E', D'), with message space $\mathcal{M}$, as follows:

$$E'(pk, m) := \left\{ \begin{array}{l} k \xleftarrow{\rm R} \mathcal{K}, \\ c \xleftarrow{\rm R} E\big(pk,\ (k, m)\big), \\ t \xleftarrow{\rm R} S(k, c), \\ \text{output } (c, t) \end{array} \right\} \qquad D'\big(sk, (c, t)\big) := \left\{ \begin{array}{l} (k, m) \leftarrow D(sk, c), \\ \text{if } V(k, c, t) = \textsf{accept} \text{ output } m, \\ \text{otherwise output } \textsf{reject} \end{array} \right\}$$

One might expect this scheme to be CCA-secure because a change to a ciphertext (c, t) will invalidate the MAC tag t. Show that this is incorrect. That is, show a CPA-secure encryption scheme

(G, E, D) for which (G, E', D') is not CCA-secure (for any choice of MAC).

12.17 (Public-key encryption with associated data). In Section 12.7 we defined public-key encryption with associated data. We mentioned that the CCA-secure schemes in this chapter can be made into public-key encryption schemes with associated data by replacing the symmetric cipher used with an AD symmetric cipher. Here we develop another approach.

(a) Consider the scheme $\mathcal{E}'_{\mathrm{TDF}}$ from Section 12.3. Suppose that we add an extra input d to the encryption and decryption algorithms, representing the associated data, and that in both algorithms we compute k as $k \leftarrow H(x, d)$, rather than $k \leftarrow H(x)$. Show that under the same assumptions used in the analysis of $\mathcal{E}'_{\mathrm{TDF}}$, this modified scheme is a CCA-secure scheme with associated data.

(b) Consider the scheme $\mathcal{E}_{\mathrm{EG}}$ from Section 12.4. Suppose that we add an extra input d to the encryption and decryption algorithms, representing the associated data, and that in both algorithms we compute k as $k \leftarrow H(v, w, d)$, rather than $k \leftarrow H(v, w)$. Show that under the same assumptions used in the analysis of $\mathcal{E}_{\mathrm{EG}}$, this modified scheme is a CCA-secure scheme with associated data.

(c) Consider the scheme $\mathcal{E}_{\mathrm{CS}}$ from Section 12.5. Suppose that we add an extra input d to the encryption and decryption algorithms, representing the associated data, and that in both algorithms we compute ρ as $\rho \leftarrow H'(v, w, d)$, rather than $\rho \leftarrow H'(v, w)$. Show that under the same assumptions used in the analysis of $\mathcal{E}_{\mathrm{CS}}$, this modified scheme is a CCA-secure scheme with associated data.

12.18 (KEMs with associated data). Exercise 12.5 introduced the notion of a CCA secure key encapsulation mechanism (KEM). One might also consider a KEM with associated data (AD KEM), so that both encryption and decryption take as input associated data d. Because the input d may be adversarially chosen, we have to modify the attack game in Exercise 12.5, so that the adversary is first given pk, then makes a series of decryption queries, followed by one encryption query, followed by a sequence of additional decryption queries. In the encryption query, the adversary supplies d, the challenger computes $(k_0, c_{\mathrm{kem}}) \overset{\mathrm{R}}{\leftarrow} E_{\mathrm{kem}}(pk, d)$ and $k_1 \overset{\mathrm{R}}{\leftarrow} \mathcal{K}$, and sends either (k_0, c_{kem}) or (k_1, c_{kem}) to the adversary. Decryption queries work just as in Exercise 12.5, except the adversary chooses the associated data $\hat{d}$ as well as the ciphertext $\hat{c}_{\mathrm{kem}}$, with the restriction that after the encryption query is made, $(\hat{c}_{\mathrm{kem}}, \hat{d}) \neq (c_{\mathrm{kem}}, d)$.

(a) Flesh out the details of the above attack game to obtain a formal definition of **1CCA-secure AD KEM**.

(b) Assume $\mathcal{E}_{\mathrm{a}}$ is a 1CCA-secure AD KEM. Assume $\mathcal{E}_{\mathrm{s}}$ is a 1CCA-secure cipher. Suppose we modify the hybrid public-key encryption scheme $\mathcal{E}$ in Exercise 12.5 so that is supports associated data, where the associated data is simply passed through to the AD KEM. Show that the resulting scheme is a 1CCA-secure AD public-key encryption scheme.

(c) Describe the AD KEM corresponding to the construction in part (a) of the previous exercise and prove that it is 1CCA secure.

(d) Describe the AD KEM corresponding to the construction in part (b) of the previous exercise and prove that it is 1CCA secure.

(e) Describe the AD KEM corresponding to the construction in part (c) of the previous exercise and prove that it is 1CCA secure.

12.19 (AD-only CCA security). Consider the notion of AD-only CCA security discussed in Section 12.7.1.

(a) Suppose that in the previous exercise, we modify the attack game so that the restriction on decryption queries is $\hat{d} \neq d$ (rather than $(\hat{c}_{\text{kem}}, \hat{d}) \neq (c_{\text{kem}}, d)$). This defines the notion of **AD-only 1CCA-secure KEM**. (Clearly, any AD KEM that is CCA secure is also AD-only CCA secure.)

Assume $\mathcal{E}_{\text{a}}$ is an AD-only 1CCA-secure AD KEM. Assume $\mathcal{E}_{\text{s}}$ is a *semantically secure* cipher. Suppose we modify the hybrid public-key encryption scheme $\mathcal{E}$ in Exercise 12.5 so that is supports associated data, where the associated data is simply passed through to the AD KEM. Show that the resulting scheme is AD-only 1CCA-secure.

Discussion: In contrast to part (b) of the previous exercise, we can use a semantically secure symmetric cipher, instead of a 1CCA-secure cipher, leading to a somewhat simpler scheme with more compact ciphertexts.

(b) In contrast to Exercise 12.4, show that if $\mathcal{E}$ is AD-only CCA secure, then $\mathcal{E}^2$ is also AD-only CCA secure.

12.20 (An AD-only CCA secure scheme). This exercise explores a particular AD-only CCA secure scheme $\mathcal{E}_{\text{GS}}$ that has some nice properties that will prove useful later. The scheme makes use of the following components:

- a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$;
- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$;
- a hash function $H_{\mathcal{K}} : \mathbb{G} \times \mathbb{G} \to \mathcal{K}$;
- a hash function $H_{\mathbb{G}} : \mathbb{G} \times \mathcal{D} \to \mathbb{G}$;
- an efficient algorithm $\mathcal{O}_{\text{DDH}}$ that on input $(g^{\alpha}, g^{\beta}, g^{\gamma})$ that outputs accept if $\gamma = \alpha\beta$, and otherwise outputs reject.

The encryption scheme $\mathcal{E}_{\text{GS}} = (G, E, D)$ has message space $\mathcal{M}$ and associated data space $\mathcal{D}$ and is defined as follows:

- the key generation algorithm runs as follows:

$$\begin{aligned} G() := \quad & \alpha \in \mathbb{Z}_q,\ u \leftarrow g^{\alpha} \\ & \text{output } (pk, sk) \leftarrow (u, \alpha) \end{aligned}$$

- the encryption algorithm runs as follows:

$$\begin{aligned} E(pk := u, m, d) := \quad & \beta \xleftarrow{\text{R}} \mathbb{Z}_q,\ \ v \leftarrow g^{\beta},\ \ w \leftarrow u^{\beta},\ \ k \leftarrow H(v, w),\ \ c \xleftarrow{\text{R}} E_{\text{s}}(k, m), \\ & \underline{u} \leftarrow H_{\mathbb{G}}(v, d),\ \ \underline{w} \leftarrow \underline{u}^{\beta} \\ & \text{output } (v, c, \underline{w}) \end{aligned}$$

- the decryption algorithm runs as follows:

```
$D(sk := \alpha,\ (v, c, \underline{w}),\ d) :=$    $\underline{u} := H_{\mathbb{G}}(v, d)$
                                   if $\mathcal{O}_{\text{DDH}}(\underline{u}, v, \underline{w}) = \textsf{reject}$
                                       then output $\textsf{reject}$
                                       else $w \leftarrow v^{\alpha}$,   $k \leftarrow H(v, w)$,   $m \leftarrow D_{\text{s}}(k, c)$
                                            output $m$
```

Your task is to prove that $\mathcal{E}_{\text{GS}}$ is AD-only CCA secure under the CDH assumption for $\mathbb{G}$, and assuming that $\mathcal{E}_{\text{s}}$ is semantically secure, and modeling $H_{\mathcal{K}}$ and $H_{\mathbb{G}}$ as random oracles. You should show that this holds even if in processing a decryption query $((\hat{v}, \hat{c}, \underline{\hat{w}}), \hat{d})$, the adversary is given more information than just $\hat{m}$, namely, $\hat{w} := \hat{v}^{\alpha}$ (from which $\hat{m}$ can certainly be computed). In particular, show that the advantage of any adversary $\mathcal{A}$ in this strengthened AD-only CCA attack game is bounded by

$$2 \cdot \text{CDH}\mathsf{adv}[\mathcal{B}_{\text{cdh}}, \mathbb{G}] + Q_{\text{e}} \cdot \text{SS}\mathsf{adv}[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}],$$

where Q_{e} is a bound on the number of encryption queries made by $\mathcal{A}$, and $\mathcal{B}_{\text{cdh}}$ and $\mathcal{B}_{\text{s}}$ are elementary wrappers around $\mathcal{A}$.

Hint: Let (u^*, v^*) be a random pair in $\mathbb{G} \times \mathbb{G}$. Set the public key $u := u^*$. For a random oracle query $(\hat{v}, \hat{d})$ to $H_{\mathbb{G}}$ made explicitly by the adversary, respond with $\underline{\hat{u}} := ug^{\rho}$ for randomly generated $\rho \in \mathbb{Z}_q$. For encryption queries, set the random oracle $H_{\mathbb{G}}$ on input (v, d) to $\underline{u} := g^{\sigma}$ for randomly generated $\sigma \in \mathbb{Z}_q$, and set $v := v^* g^{\tau}$ for randomly generated $\tau \in \mathbb{Z}_q$. With this, you can arrange to process all decryption queries without knowing the secret key, to process all encryption queries without knowing $\log_g v$, and to ensure that the adversary learns nothing about encrypted messages unless he can solve the instance (u^*, v^*) of the CDH problem (which will be evident by inspecting the adversary's queries to the random oracle $H_{\mathcal{K}}$) or break the semantic security of $\mathcal{E}_{\text{s}}$.

Discussion: This scheme is the same as $\mathcal{E}_{\text{EG}}$, except for the additional group element $\underline{w}$ placed in the ciphertext by the encryption algorithm, and the additional test $\mathcal{O}_{\text{DDH}}(\underline{u}, v, \underline{w}) = \textsf{reject}$ performed by the decryption algorithm. As we will see in Section 22.3.3, this scheme has nice properties that are convenient in building a simple and secure *threshold* decryption scheme, where the decryption key is never stored in one location and decryption is performed via a distributed computation. Obviously, the need for the algorithm $\mathcal{O}_{\text{DDH}}$ limits the applicability of this encryption scheme. We will see later in Chapter 15 how we can efficiently implement $\mathcal{O}_{\text{DDH}}$ in some settings. We will also see later in Chapter 20 how we can eliminate the need for algorithm $\mathcal{O}_{\text{DDH}}$ altogether (see Exercise 20.19).

12.21 (Baby Bleichenbacher attack). Consider an RSA public key (n, e), where n is an RSA modulus, and e is an encryption exponent. For $x \in \mathbb{Z}_n$, consider the predicate $P_x : \mathbb{Z}_n \to \{0, 1\}$ defined as:

$$P_x(r) := \left\{ \begin{array}{l} y \leftarrow x \cdot r \in \mathbb{Z}_n \\ \text{treat } y \text{ as an integer in the interval } [0, n) \\ \text{if } y > n/2\text{, output } 1 \\ \text{else, output } 0 \end{array} \right\}$$

(a) Show that by querying the predicate P_x at about $\log_2 n$ points, it is possible to learn the value of x.

(b) Suppose an attacker obtains an RSA public key and an element $c \in \mathbb{Z}_n$. It wants to compute the eth root of c in $\mathbb{Z}_n$. To do so, the attacker can query an oracle that takes $z \in \mathbb{Z}$ as input, and outputs 1 when $[z^{1/e} \bmod n] > n/2$, and outputs 0 otherwise. Here $[z^{1/e} \bmod n]$

is an integer w in the interval $[0, n)$ such that $w^e \equiv z \bmod n$. Use part (a) to show how the adversary can recover the eth root of c.

12.22 (OAEP is CPA-secure for any trapdoor function). Let $\mathcal{T} = (G, F, I)$ be a trapdoor function defined over $(\mathcal{X}, \mathcal{Y})$ where $\mathcal{X} = 0^8 \times \{0,1\}^{t-8}$. Consider the OAEP padding scheme from Fig. 12.7, omitting the associated data input d, and let $\mathcal{E}_{\text{OAEP}}$ be the public key encryption scheme that results from coupling $\mathcal{T}$ with OAEP, as in (12.41). Show that $\mathcal{E}_{\text{OAEP}}$ is CPA secure in the random oracle model.

12.23 (A counter-example to the CCA-security of OAEP). Let $\mathcal{T}_0 = (G, F_0, I_0)$ be a one-way trapdoor permutation defined over $\mathcal{R} := \{0,1\}^h$. Suppose, $\mathcal{T}_0$ is xor-homomorphic in the following sense: there is an efficient algorithm C that for all pk output by G and all $r, \Delta \in \mathcal{R}$, we have $C(F_0(pk, r)) = F_0(pk, r \oplus \Delta)$. Next, if $t > 2h + 16$, let $\mathcal{T} = (G, F, I)$ be the trapdoor permutation defined over $0^8 \times \{0,1\}^{t-8}$ as follows:

$$F\big(pk,\ (00 \parallel r \parallel z)\big) = 00 \parallel F_0(pk, r) \parallel z.$$

Notice that from $F\big(pk,\ (00 \parallel r \parallel z)\big)$ it is easy to recover z, but not the entire preimage. Consider the public-key encryption $\mathcal{E}_{\text{OAEP}}$ obtained by coupling this $\mathcal{T}$ with OAEP as in (12.41). Show a CCA attack on this scheme that has advantage 1 in winning the CCA game. Your attack shows that for some one-way trapdoor functions, the scheme $\mathcal{E}_{\text{OAEP}}$ may not be CCA-secure.

12.24 (RSA is partial one-way). Consider an RSA public key (n, e), where n is an RSA modulus, and e is an encryption exponent. Suppose n is a t-bit integer where t is even, and let T be an integer that is a little bit smaller than $2^{(t/2)}$. Let x be a random integer in the interval $[0, n)$ and $y := (x^e \bmod n) \in \mathbb{Z}_n$. Suppose $\mathcal{A}$ is an algorithm so that

$$\Pr\Big[\mathcal{A}(n, e, y) = z \text{ and } 0 \leq x - zT < T\Big] > \epsilon.$$

The fact that the integer zT is so close to x means that z reveals half of the most significant bits of x. Hence, $\mathcal{A}$ is an RSA partial one-way adversary for the most significant bits.

(a) Construct an algorithm $\mathcal{B}$ that takes (n, e, y) as input, and outputs x with probability ϵ^2. For this, you should determine a more precise value for the parameter T.

Hint: Algorithm $\mathcal{B}$ works by choosing a random $r \in \mathbb{Z}_n$ and running $z_0 \leftarrow \mathcal{A}(n, e, y)$ and $z_1 \leftarrow \mathcal{A}(n, e,\ y \cdot r^e)$. If $\mathcal{A}$ outputs valid z_0 and z_1 both times — an event that happens with probability ϵ^2 (explain why) — then

$$\begin{aligned} x &\equiv z_0 T + \Delta_0 \pmod{n} \\ x \cdot r &\equiv z_1 T + \Delta_1 \pmod{n} \end{aligned}$$

where $0 \leq \Delta_0, \Delta_1 < T$. Show an efficient algorithm that given such r, z_0, z_1, outputs x, Δ_0, Δ_1, with high probability. Your algorithm $\mathcal{B}$ should make use of an algorithm for finding shortest vectors in 2-dimensional lattices (see, for example, [153]). If you get stuck, see [66].

Discussion: This result shows that if RSA is one-way, then an adversary cannot even compute the most significant bits of a preimage.

(b) Show that a similar result holds if an algorithm $\mathcal{A}'$ outputs more than half the *least* significant bits of x.

12.25 (Simplified Cramer-Shoup decryption). Consider the following simplified version $\mathcal{E}_{\text{SCS}}$ of the Cramer-Shoup encryption scheme (presented in Section 12.5):

- the key generation algorithm runs as follows:

```
G() :=   $\alpha, \delta, \delta_1, \delta_2 \xleftarrow{R} \mathbb{Z}_q$,  $u \leftarrow g^\alpha$,  $h \leftarrow g^\delta$,  $h_1 \leftarrow g^{\delta_1}$,  $h_2 \leftarrow g^{\delta_2}$
         $pk \leftarrow (u, h, h_1, h_2)$,  $sk \leftarrow (\alpha, \delta, \delta_1, \delta_2)$
         output $(pk, sk)$;
```

for a given secret key $sk = (\alpha, \delta, \delta_1, \delta_2) \in \mathbb{Z}_q^4$ and a ciphertext $(v, w, z', c) \in \mathbb{G}^3 \times \mathcal{C}$, the decryption algorithm runs as follows:

```
D(sk, (v, w, z', c) ) :=   $\rho \leftarrow H'(v, w)$
                           if $v^\alpha = w$ and $v^{\delta_1 + \rho\delta_2} = z'$
                               then $z \leftarrow v^\delta$,  $k \leftarrow H(v, z)$,  $m \leftarrow D_s(k, c)$
                               else  $m \leftarrow$ reject
                           output $m$.
```

Encryption is the same as in $\mathcal{E}_{\text{CS}}$.

Show that

$$\left|\text{1CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{CS}}] - \text{1CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{SCS}}]\right| \leq 2Q_{\text{d}}/q,$$

for every adversary $\mathcal{A}$ that makes at most Q_{d} decryption queries. Conclude that $\mathcal{E}_{\text{SCS}}$ is CCA secure under the same assumption as in Theorem 12.9.

12.26 (Stronger properties for projective hash functions). We can strengthen Attack Games 12.4 and 12.5, allowing the adversary to choose the values (v, w) (and ρ) adaptively.

(a) Consider a variant of Attack Game 12.4 in which the adversary first submits $u \in \mathbb{G}$ to the challenger (which defines $\mathcal{L}_u$), obtaining the auxiliary information h; then the adversary makes some number of evaluation queries; at some point, the adversary submits $(v, w) \in \mathbb{G}^2 \setminus \mathcal{L}_u$ to the challenger, obtaining z_b; finally, the adversary continues making evaluation queries, and outputs a bit, as usual. Show that Lemma 12.6 still holds for this variant.

(b) Consider a variant of Attack Game 12.5 in which the adversary first submits $u \in \mathbb{G}$ to the challenger (which defines $\mathcal{L}_u$), obtaining the auxiliary information (h_1, h_2); then the adversary makes some number of evaluation queries; at some point, the adversary submits $(v, w) \in \mathbb{G}^2 \setminus \mathcal{L}_u$ and $\rho \in \mathbb{Z}_q$ to the challenger, obtaining z; finally, the adversary continues making evaluation queries, and outputs a list of tuples, as usual. Show that Lemma 12.8 still holds for this variant.

12.27 (Multiplicative Cramer-Shoup encryption). Consider the following multiplicative version of the Cramer-Shoup encryption scheme (presented in Section 12.5) that supports associated data (see Section 12.7) coming from a set $\mathcal{D}$. Let $\mathbb{G}$ be a cyclic group of prime order q with generator $g \in \mathbb{G}$. Let $H' : \mathbb{G}^3 \times \mathcal{D} \to \mathbb{Z}_q$ be a hash function. The encryption scheme $\mathcal{E}_{\text{MCS}} = (G, E, D)$ is defined over $(\mathbb{G}, \mathcal{D}, \mathbb{G}^4)$ as follows. Key generation is exactly as in $\mathcal{E}_{\text{CS}}$. For a given public key $pk = (u, h, h_1, h_2) \in \mathbb{G}^4$ message $m \in \mathbb{G}$, and associated data $d \in \mathcal{D}$, the encryption algorithm runs as follows:

```
E(pk, m, d) :=   $\beta \xleftarrow{R} \mathbb{Z}_q$,  $v \leftarrow g^\beta$,  $w \leftarrow u^\beta$,  $e \leftarrow h^\beta \cdot m$
                 $\rho \leftarrow H'(v, w, e, d)$,  $z' \leftarrow (h_1 h_2^\rho)^\beta$,  output $(v, w, e, z')$.
```

For a given secret key $sk = (\sigma, \tau, \sigma_1, \tau_1, \sigma_2, \tau_2) \in \mathbb{Z}_q^6$ and a ciphertext $(v, w, e, z') \in \mathbb{G}^4$, and associated data $d \in \mathcal{D}$, the decryption algorithm runs as follows:

```
D(sk, (v, w, e, z', d) ) :=   $\rho \leftarrow H'(v, w, e, d)$
                              if $v^{\sigma_1+\rho\sigma_2} w^{\tau_1+\rho\tau_2} = z'$
                                  then output $e/(v^{\sigma} w^{\tau})$
                                  else  output reject.
```

Show that $\mathcal{E}_{\text{MCS}}$ is CCA secure, provided H' is collision resistant and the DDH assumption holds for $\mathbb{G}$.

Hint: Part (b) of the previous exercise may be helpful.

Note: This scheme can be simplified, without sacrificing security, along the same lines discussed in Exercise 12.25, where the secret key is $(\alpha, \delta, \delta_1, \delta_2) \in \mathbb{Z}_q^4$, with $h = g^{\delta}$, $h_1 = g^{\delta_1}$, $h_2 = g^{\delta_2}$, and where the decryption algorithm tests if $v^{\alpha} = w$ and $v^{\delta_1+\rho\delta_2} = z'$, and if so outputs e/v^{δ}.

12.28 (Non-adaptive CCA security and Cramer-Shoup lite). One can define a weaker notion of CCA security, corresponding to a variant of the CCA attack game in which the adversary must make all of its decryption queries before making any of its encryption queries. Moreover, just as we did for ordinary CCA security, it suffices to assume that the adversary makes just a single encryption query. Let us call the corresponding security notion **non-adaptive 1CCA security**.

Now consider the following simplified version of the encryption scheme in the previous exercise. Again, $\mathbb{G}$ is a cyclic group of prime order q with generator $g \in \mathbb{G}$. The encryption scheme $\mathcal{E}_{\text{MCSL}} = (G, E, D)$ is defined over $(\mathbb{G}, \mathbb{G}^4)$ as follows. The key generation algorithm runs as follows:

```
G() :=   $\alpha \xleftarrow{R} \mathbb{Z}_q$,  $u \leftarrow g^{\alpha}$
         for $i = 0, 1$: $\sigma_i, \tau_i \xleftarrow{R} \mathbb{Z}_q$, $h_i \leftarrow g^{\sigma_i} u^{\tau_i}$
         $pk \leftarrow (u, h_0, h_1)$,  $sk \leftarrow (\sigma_0, \tau_0, \sigma_1, \tau_1)$
         output $(pk, sk)$.
```

For a given public key $pk = (u, h_0, h_1) \in \mathbb{G}^3$ and message $m \in \mathbb{G}$, the encryption algorithm runs as follows:

```
E(pk, m) :=   $\beta \xleftarrow{R} \mathbb{Z}_q$,  $v \leftarrow g^{\beta}$,  $w \leftarrow u^{\beta}$,  $e \leftarrow h_0^{\beta} \cdot m$
              $z' \leftarrow h_1^{\beta}$,  output $(v, w, z', e)$.
```

for a given secret key $sk = (\sigma_0, \tau_0, \sigma_1, \tau_1) \in \mathbb{Z}_q^4$ and a ciphertext $(v, w, z', e) \in \mathbb{G}^4$, the decryption algorithm runs as follows:

```
D(sk, (v, w, z', e) ) :=   if $v^{\sigma_1} w^{\tau_1} = z'$
                               then output $e/(v^{\sigma_0} w^{\tau_0})$
                               else  output reject.
```

(a) Show that $\mathcal{E}_{\text{MCSL}}$ is non-adaptive 1CCA secure, provided the DDH assumption holds for $\mathbb{G}$.

(b) Show that $\mathcal{E}_{\text{MCSL}}$ is not CCA secure.

Note: This scheme can also be simplified along the same lines discussed in Exercise 12.25, and the same results hold.

12.29 (Generalizing universal projective hash functions). This exercise develops a construction for universal projective hash functions that generalizes the one presented in Section 12.5.1. Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $\mathbb{G}^{1\times n}$ be the set of row vectors with entries in $\mathbb{G}$ and $\mathbb{Z}_q^{n\times 1}$ the set of column vectors with entries in $\mathbb{Z}_q$. For $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{G}^{1\times n}$ and $\beta \in \mathbb{Z}_q$, define $\boldsymbol{u}^\beta = (u_1^\beta, \ldots, u_n^\beta) \in \mathbb{G}^{1\times n}$. For $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{G}^{1\times n}$ and $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{G}^{1\times n}$, define $\boldsymbol{u} \cdot \boldsymbol{v} = (u_1v_1, \ldots, u_nv_n) \in \mathbb{G}^{1\times n}$. Finally, for $\boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{G}^{1\times n}$ and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \mathbb{Z}_q^{n\times 1}$, define $\boldsymbol{v}^{\boldsymbol{\sigma}} = v_1^{\sigma_1} \cdots v_n^{\sigma_n} \in \mathbb{G}$.

Now let $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k \in \mathbb{G}^{1\times n}$ be fixed throughout the remainder of the exercise, and define $\mathcal{L} \subseteq \mathbb{G}^{1\times n}$ to be the set of all elements of $\mathbb{G}^{1\times n}$ that can be written as $\boldsymbol{u}_1^{\beta_1} \cdots \boldsymbol{u}_k^{\beta_k}$ for some $\beta_1, \ldots, \beta_k \in \mathbb{Z}_q$.

(a) Show how to efficiently compute $\boldsymbol{v}^{\boldsymbol{\sigma}}$, given $\beta_1, \ldots, \beta_k \in \mathbb{Z}_q$ such that $\boldsymbol{v} = \boldsymbol{u}_1^{\beta_1} \cdots \boldsymbol{u}_k^{\beta_k}$, along with $h_1, \ldots, h_k \in \mathbb{G}$, where $h_i = \boldsymbol{u}_i^{\boldsymbol{\sigma}}$ for $i = 1, \ldots, k$.

(b) Suppose that $\boldsymbol{\sigma} \in \mathbb{Z}_q^{n\times 1}$ is chosen uniformly at random. Show that for each $\boldsymbol{v} \in \mathbb{G}^{1\times n} \setminus \mathcal{L}$, the random variable $\boldsymbol{v}^{\boldsymbol{\sigma}}$ is uniformly distributed over $\mathbb{G}$, independently of the random variable $(\boldsymbol{u}_1^{\boldsymbol{\sigma}}, \ldots, \boldsymbol{u}_k^{\boldsymbol{\sigma}})$.

12.30 (A universal projective hash function for $\mathcal{E}_{\text{MCS}}$). Consider the encryption scheme $\mathcal{E}_{\text{MCS}}$ from Exercise 12.27. Let the public key $pk = (u, h, h_1, h_2)$, message m, and associated data d be fixed. Define $\mathcal{L} \subseteq \mathbb{G}^4$ to be the set of possible outputs of the encryption algorithm on these inputs:

$$\mathcal{L} := \{\ (v, w, e, z') : v = g^\beta, w = u^\beta, e = h^\beta \cdot m, z' = (h_1 h_2^\rho)^\beta), \rho = H'(v, w, e, d) \text{ for some } \beta \in \mathbb{Z}_q\ \}.$$

Design a universal projective hash function for $\mathcal{L}$ with outputs in $\mathbb{G}$. The algorithm to evaluate the function on $(v, w, e, z') \in \mathcal{L}$ takes as input the corresponding β value, along with whatever auxiliary information is provided to facilitate computation of the function on $\mathcal{L}$. For inputs not in $\mathcal{L}$, the output of the function should be uniformly distributed over $\mathbb{G}$, independently of the auxiliary information.

Hint: Use the result of the previous exercise.

12.31 (Interactive hash Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $H : \mathbb{G}^2 \to \mathcal{K}$ be a hash function. We say that the **Interactive Hash Diffie-Hellman** (IHDH) assumption holds for $(\mathbb{G}, H)$ if it is infeasible for an efficient adversary to distinguish between the following two experiments. In Experiment 0, the challenger computes

$$\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g^\alpha, v \leftarrow g^\beta, w \leftarrow g^{\alpha\beta}, \quad k \leftarrow H(v, w)$$

and sends (u, v, k) to the adversary. After that, the adversary is allowed to make a series queries. Each query is of the form $\tilde{v} \in \mathbb{G}^2$. Upon receiving such a query, the challenger computes

$$\tilde{w} \leftarrow \tilde{v}^\beta, \quad \tilde{k} \leftarrow H(\tilde{v}, \tilde{w})$$

and sends $\tilde{k}$ to the adversary. Experiment 1 is exactly the same as Experiment 0, except that the challenger computes $k \xleftarrow{\text{R}} \mathcal{K}$.

(a) Show that if H is modeled as a random oracle and the ICDH assumption holds for $\mathbb{G}$, then the IHDH assumption holds for $(\mathbb{G}, H)$.

(b) Prove that the ElGamal public-key encryption scheme $\mathcal{E}_{\text{EG}}$ is CCA secure if the IHDH assumption holds for $(\mathbb{G}, H)$ and $\mathcal{E}_{\text{s}}$ is 1CCA secure.

12.32 (The twin CDH problem). In Section 12.4, we saw that the basic ElGamal encryption scheme could not be proved secure under the ordinary CDH assumption, even in the random oracle model. To analyze the scheme, we had to introduce a new, stronger assumption, called the interactive CDH (ICDH) assumption (see Definition 12.4). In this exercise and the next, we show how to avoid this stronger assumption with just a slightly more involved encryption scheme.

Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. The **Twin CDH (2CDH)** problem is this: given

$$g^{\alpha_1}, g^{\alpha_2}, g^{\beta}$$

compute the *pair*

$$(g^{\alpha_1\beta}, g^{\alpha_2\beta}).$$

A tuple of the form

$$(g^{\alpha_1}, g^{\alpha_2}, g^{\beta}, g^{\alpha_1\beta}, g^{\alpha_2\beta})$$

is called **Twin DH (2DH) tuple**. The **interactive Twin CDH (I2CDH) assumption** is this: it is hard to solve a random instance $(g^{\alpha_1}, g^{\alpha_2}, g^{\beta})$ of the 2DH problem, given access to an oracle that recognizes 2DH-tuples of the form $(g^{\alpha_1}, g^{\alpha_2}, \cdot, \cdot, \cdot)$.

(a) Flesh out the details of the I2CDH assumption by giving an attack game analogous to Attack Game 12.3. In particular, you should define an analogous advantage I2CDH$\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ for an adversary $\mathcal{A}$ in this attack game.

(b) Using the trapdoor test in Exercise 10.15, show that the CDH assumption implies the I2CDH assumption. In particular, show that for every I2CDH adversary $\mathcal{A}$, there exists a CDH adversary $\mathcal{B}$ (where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$), such that

$$\text{I2CDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}] \leq \text{CDH}\mathsf{adv}[\mathcal{B}, \mathbb{G}] + \frac{Q_{\text{ro}}}{q},$$

where Q_{ro} is an upper bound on the number of oracle queries made by $\mathcal{A}$.

12.33 (Twin CDH encryption). The **Twin CDH encryption scheme**, $\mathcal{E}_{\text{2cdh}} = (G, E, D)$, is a public-key encryption scheme whose CCA security (in the random oracle model) is based on the I2CDH assumption (see previous exercise). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. We also need a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and a hash function $H : \mathbb{G}^3 \to \mathcal{K}$. The algorithms G, E, and D are defined as follows:

```
          G()  :=  $\alpha_1 \xleftarrow{R} \mathbb{Z}_q$,  $\alpha_2 \xleftarrow{R} \mathbb{Z}_q$,  $u_1 \leftarrow g^{\alpha_1}$,  $u_2 \leftarrow g^{\alpha_2}$
                   $pk \leftarrow (u_1, u_2)$,  $sk \leftarrow (\alpha_1, \alpha_2)$
                   output $(pk, sk)$;
     E(pk, m)  :=  $\beta \xleftarrow{R} \mathbb{Z}_q$,  $v \leftarrow g^{\beta}$,  $w_1 \leftarrow u_1^{\beta}$,  $w_2 \leftarrow u_2^{\beta}$
                   $k \leftarrow H(v, w_1, w_2)$,  $c \xleftarrow{R} E_{\text{s}}(k, m)$
                   output $(v, c)$;
D( sk, (v,c) ) :=  $w_1 \leftarrow v^{\alpha_1}$,  $w_2 \leftarrow v^{\alpha_2}$,  $k \leftarrow H(v, w_1, w_2)$,  $m \leftarrow D_{\text{s}}(k, c)$
                   output $m$.
```

The message space is $\mathcal{M}$ and the ciphertext space is $\mathbb{G} \times \mathcal{C}$.

(a) Suppose that we model the hash function H as a random oracle. Show that $\mathcal{E}_{\text{2cdh}}$ is CCA secure under the I2CDH assumption, also assuming that $\mathcal{E}_{\text{s}}$ is 1CCA secure. In particular, show that for every 1CCA adversary $\mathcal{A}$ attacking $\mathcal{E}_{\text{2cdh}}$, there exist an I2CDH adversary $\mathcal{B}_{\text{i2cdh}}$ and a 1CCA adversary $\mathcal{B}_{\text{s}}$, where $\mathcal{B}_{\text{i2cdh}}$ and $\mathcal{B}_{\text{s}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\text{1CCA}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{2cdh}}] \leq 2 \cdot \text{I2CDH}\mathsf{adv}[\mathcal{B}_{\text{i2cdh}}, \mathbb{G}] + \text{1CCA}\mathsf{adv}[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}].$$

(b) Now use the result of part (b) of the previous exercise to show that $\mathcal{E}_{\text{2cdh}}$ is secure in the random oracle model under the ordinary CDH assumption for $\mathbb{G}$ (along with the assumption that $\mathcal{E}_{\text{s}}$ is 1CCA secure). In particular, show that for every 1CCA adversary $\mathcal{A}$ attacking $\mathcal{E}_{\text{2cdh}}$, there exist a CDH adversary $\mathcal{B}_{\text{cdh}}$ and a 1CCA adversary $\mathcal{B}_{\text{s}}$, where $\mathcal{B}_{\text{cdh}}$ and $\mathcal{B}_{\text{s}}$ are elementary wrappers around $\mathcal{A}$, such that

$$\text{1CCA}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{2cdh}}] \leq 2 \cdot \text{CDH}\mathsf{adv}[\mathcal{B}_{\text{cdh}}, \mathbb{G}] + \frac{2Q_{\text{ro}}}{q} + \text{1CCA}\mathsf{adv}[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}],$$

where Q_{ro} is a bound on the number of random oracle queries made by $\mathcal{A}$.

Discussion: Compared to the ElGamal encryption scheme, $\mathcal{E}_{\text{EG}}$, which we analyzed in Section 12.4, this scheme achieves CCA security under the CDH assumption, rather than the stronger ICDH assumption. Also, compared to the instantiation of the Fujisaki-Okamoto transformation with ElGamal, $\mathcal{E}_{\text{FO}}^{\text{EG}}$, which we analyzed in Section 12.6.2, the reduction to CDH here is much tighter, as we do not need to multiply CDH$\mathsf{adv}[\mathcal{B}_{\text{cdh}}, \mathbb{G}]$ by a factor of Q_{ro} as in (12.40). This tight reduction even extends to the more general multi-key CCA setting, as explored in the next exercise.

12.34 (Multi-key CCA security of Twin CDH). Consider a slight modification of the public-key encryption scheme $\mathcal{E}_{\text{2cdh}}$ from the previous exercise. This new scheme, which we call $x\mathcal{E}_{\text{2cdh}}$, is exactly the same as $\mathcal{E}_{\text{2cdh}}$, except that instead of deriving the symmetric key as $k = H(v, w_1, w_2)$, we derive it as $k = H(u_1, u_2, v, w_1, w_2)$. Consider the security of $x\mathcal{E}_{\text{2cdh}}$ in the multi-key CCA attack game, discussed above in Exercise 12.6. In that attack game, suppose Q_{te} is a bound on the total number of encryptions. Also, let Q_{ro} be a bound on the total number of random oracle queries. Let $\mathcal{A}$ be an adversary that attacks the multi-key CCA security of $x\mathcal{E}_{\text{2cdh}}$. Show that $\mathcal{A}$'s advantage is at most

$$2 \cdot \epsilon_{\text{cdh}} + \frac{2Q_{\text{ro}}}{q} + Q_{\text{te}} \cdot \epsilon_{\text{s}},$$

where ϵ_{cdh} is that advantage of a CDH adversary $\mathcal{B}_{\text{cdh}}$ attacking $\mathbb{G}$ and ϵ_{s} is the advantage of a 1CCA adversary $\mathcal{B}_{\text{s}}$ attacking $\mathcal{E}_{\text{s}}$ (where both $\mathcal{B}_{\text{cdh}}$ and $\mathcal{B}_{\text{s}}$ are elementary wrappers around $\mathcal{A}$).

Hint: Use the random self reduction for CDH (see Exercise 10.5).

Chapter 13

Digital signatures

In this chapter and the next we develop the concept of a digital signature. Although there are some parallels between physical world signatures and digital signatures, the two are quite different. We motivate digital signatures with three examples.

Example 1: Software distribution. Suppose a software company, SoftAreUs, releases a software update for its product. Customers download the software update file U by some means, say from a public distribution site or from a peer-to-peer network. Before installing U on their machine, customers want to verify that U really is from SoftAreUs. To facilitate this, SoftAreUs appends a short tag to U, called a signature. Only SoftAreUs can generate a signature on U, but anyone in the world can verify it. Note that there are no secrecy issues here — the update file U is available in the clear to everyone. A MAC system is of no use in this setting because SoftAreUs does not maintain a shared secret key with each of its customers. Some software distribution systems use collision resistant hashing, but that requires an online read-only server that every customer uses to check that the hash of the received file U matches the hash value on the read-only server.

To provide a clean solution, with no additional security infrastructure, we need a new cryptographic mechanism called a digital signature. The signing process works as follows:

- First, SoftAreUs generates a secret signing key sk along with some corresponding public key denoted pk. SoftAreUs keeps the secret key sk to itself. The public key pk is hard-coded into all copies of the software sold by SoftAreUs and is used to verify signatures issued using sk.

- To sign a software update file U, SoftAreUs runs a signing algorithm S that takes (sk, U) as input. The algorithm outputs a short signature σ. SoftAreUs then ships the pair (U, σ) to all its customers.

- A customer Bob, given the update (U, σ) and the public key pk, checks validity of this message-signature pair using a signature verification algorithm V that takes (pk, U, σ) as input. The algorithm outputs either accept or reject depending on whether the signature is valid or not. Recall that Bob obtains pk from the pre-installed software system from SoftAreUs.

This mechanism is widely used in practice in a variety of software update systems. For security we must require that an adversary, who has pk, cannot generate a valid signature on a fake update file. We will make this precise in the next section.

We emphasize that a digital signature σ is a function of the data U being signed. This is very different from signatures in the physical world where the signature is always the same no matter what document is being signed.

Example 2: Authenticated email. As a second motivating example, suppose Bob receives an email claiming to be from his friend Alice. Bob wants to verify that the email really is from Alice. A MAC system would do the job, but requires that Alice and Bob have a shared secret key. What if they never met before and do not share a secret key? Digital signatures provide a simple solution. First, Alice generates a public/secret key pair (pk, sk). For now, we assume Alice places pk in a public read-only directory. We will discuss how to get rid of this directory in just a minute.

When sending an email m to Bob, Alice generates a signature σ on m derived using her secret key. She then sends (m, σ) to Bob. Bob receives (m, σ) and verifies that m is from Alice in two steps. First, Bob retrieves Alice's public key pk. Second, Bob runs the signature verification algorithm on the triple (pk, m, σ). If the algorithm outputs accept then Bob is assured that the message came from Alice. More precisely, Bob is assured that the message was sent by someone who knows Alice's secret key. Normally this would only be Alice, but if Alice's key is stolen then the message could have come from the thief.

As a more concrete example of this, the *domain keys identified mail (DKIM)* system is an email-signing system that is widely used on the Internet. An organization that uses DKIM generates a public/secret key pair (pk, sk) and uses sk to sign every outgoing email from the organization. The organization places the public key pk in the DNS records associated with the organization, so that anyone can read pk. An email recipient verifies the signature on every incoming DKIM email to ensure that the email source is the claimed organization. If the signature is valid the email is delivered, otherwise it is dropped. DKIM is widely used as a mechanism to make it harder for spammers to send spam email that pretends to be from a reputable source.

Example 3: Certificates. As a third motivating example for digital signatures, we consider their most widely used application. In Chapter 11 and in the authenticated email system above, we assumed public keys are obtained from a read-only public directory. In practice, however, there is no public directory. Instead, Alice's public key pk is certified by some third party called a **certificate authority** or CA for short. We will see how this process works in more detail in Section 13.8. For now, we briefly explain how signatures are used in the certification process.

To generate a certified public key, Alice first generates a public/private key pair (pk, sk) for some public-key cryptosystem, such as a public-key encryption scheme or a signature scheme. Next, Alice presents her public key pk to the CA. The CA then verifies that Alice is who she claims to be, and once the CA is convinced that it is speaking with Alice, the CA constructs a statement m saying "public key pk belongs to Alice." Finally, the CA signs the message m using its own secret key sk_{CA} and sends the pair $Cert := (m, \sigma_{\text{CA}})$ back to Alice. This pair $Cert$ is called a **certificate** for pk. When Bob needs Alice's public key, he first obtains Alice's certificate from Alice and verifies the CA's signature in the certificate. If the signature is valid, Bob has some confidence that pk is Alice's public key. The main purpose of the CA's digital signature is to prove to Bob that the statement m was issued by the CA. Of course, to verify the CA's signature, Bob needs the CA's public key pk_{CA}. Typically, CA public keys come pre-installed with an operating system or a Web browser. In other words, we simply assume that the CA's public key is already available on Bob's machine.

Of course, the above can be generalized so that the CA's certificate for Alice associates several public keys with her identity, such as public keys for both encryption and signatures.

Non-repudiation. An interesting property of the authenticated email system above is that Bob now has evidence that the message m is from Alice. He could show the pair (m, σ) to a judge who could also verify Alice's signature. Thus, for example, if m says that Alice agrees to sell her car to Bob, then Alice is (in some sense) committed to this transaction. Bob can use Alice's signature as proof that Alice agreed to sell her car to Bob — the signature binds Alice to the message m. This property provided by digital signatures is called **non-repudiation**.

Unfortunately, things are not quite that simple. Alice can repudiate the signature by claiming that the public key *pk* is not hers and therefore the signature was not issued by her. Or she can claim that her secret key *sk* was stolen and the signature was issued by the thief. After all, computers are compromised and keys are stolen all the time. Even worse, Alice could deliberately leak her secret key right after generating it thereby invalidating all her signatures. The judge at this point has no idea who to believe.

These issues are partially the reason why digital signatures are not often used for legal purposes. Digital signatures are primarily a cryptographic tool used for authenticating data in computer systems. They are a useful building block for higher level mechanisms such as key-exchange protocols, but have little to do with the legal system. Several legislative efforts in the U.S. and Europe attempt to clarify the process of digitally signing a document. In the U.S., for example, electronically signing a document does not require a cryptographic digital signature. We discuss the legal aspects of digital signatures in Section 13.9.

Non-repudiation does not come up in the context of MACs because MACs are non-binding. To see why, suppose Alice and Bob share a secret key and Alice sends a message to Bob with an attached MAC tag. Bob cannot use the tag to convince a judge that the message is from Alice since Bob could have just as easily generated the tag himself using the MAC key. Hence Alice can easily deny ever sending the message. The asymmetry of a signature system — the signer has *sk* while the verifier has *pk* — makes it harder (though not impossible) for Alice to deny sending a signed message.

13.1 Definition of a digital signature

Now that we have an intuitive feel for how digital signature schemes work, we can define them more precisely. Functionally, a digital signature is similar to a MAC. The main difference is that in a MAC, both the signing and verification algorithms use the same secret key, while in a signature scheme, the signing algorithm uses one key, *sk*, while the verification algorithm uses another, *pk*.

Definition 13.1. *A **signature scheme** $\mathcal{S} = (G, S, V)$ is a triple of efficient algorithms, G, S and V, where G is called a **key generation algorithm**, S is called a **signing algorithm**, and V is called a **verification algorithm**. Algorithm S is used to generate signatures and algorithm V is used to verify signatures.*

- *G is a probabilistic algorithm that takes no input. It outputs a pair (pk, sk), where sk is called a secret **signing key** and pk is called a public **verification key**.*

- *S is a probabilistic algorithm that is invoked as $\sigma \xleftarrow{R} S(sk, m)$, where sk is a secret key (as output by G) and m is a message. The algorithm outputs a **signature** σ.*

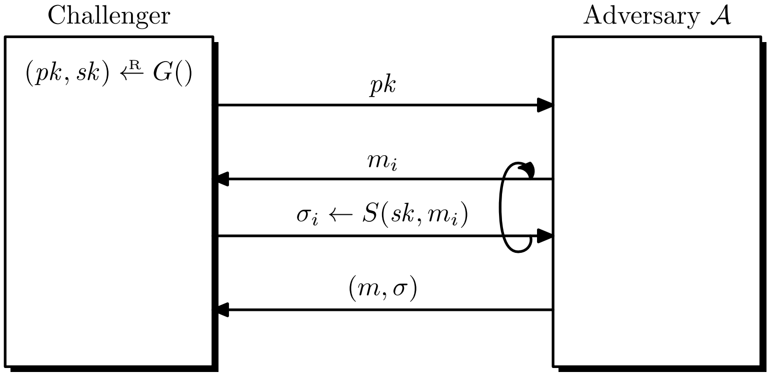

Challenger
Adversary $\mathcal{A}$
$(pk, sk) \xleftarrow{R} G()$
pk
m_i
$\sigma_i \leftarrow S(sk, m_i)$
(m, σ)

Figure 13.1: Signature attack game (Attack Game 13.1)

- *V is a deterministic algorithm invoked as $V(pk, m, \sigma)$. It outputs either* accept *or* reject*.*
- *We require that a signature generated by S is always accepted by V. That is, for all (pk, sk) output by G and all messages m, we have*

$$\Pr[V(pk,\ m,\ S(sk,\ m)\ ) = \textsf{accept}] = 1.$$

*As usual, we say that messages lie in a finite **message space** $\mathcal{M}$, and signatures lie in some finite **signature space** Σ. We say that $\mathcal{S} = (G, S, V)$ is defined over $(\mathcal{M}, \Sigma)$.*

13.1.1 Secure signatures

The definition of a secure signature scheme is similar to the definition of a secure MAC. We give the adversary the power to mount a **chosen message attack**, namely the attacker can request the signature on any message of his choice. Even with such power, the adversary should not be able to create an **existential forgery**, namely the attacker cannot output a valid message-signature pair (m, σ) for some new message m. Here "new" means a message that the adversary did not previously request a signature for.

More precisely, we define secure signatures using an attack game between a challenger and an adversary $\mathcal{A}$. The game is described below and in Fig. 13.1.

Attack Game 13.1 (Signature security). For a given signature scheme $\mathcal{S} = (G, S, V)$, defined over $(\mathcal{M}, \Sigma)$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger runs $(pk, sk) \xleftarrow{R} G()$ and sends pk to $\mathcal{A}$.
- $\mathcal{A}$ queries the challenger several times. For $i = 1, 2, \ldots,$ the ith *signing query* is a message $m_i \in \mathcal{M}$. Given m_i, the challenger computes $\sigma_i \xleftarrow{R} S(sk, m_i)$, and then gives σ_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, \sigma) \in \mathcal{M} \times \Sigma$.

We say that the adversary wins the game if the following two conditions hold:

- $V(pk, m, \sigma) =$ accept, and

- m is new, namely $m \notin \{m_1, m_2, \ldots\}$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$, denoted $\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. Finally, we say that $\mathcal{A}$ is a **Q-query adversary** if $\mathcal{A}$ issues at most Q signing queries. □

Definition 13.2 (secure signature). *We say that a signature scheme $\mathcal{S}$ is secure if for all efficient adversaries $\mathcal{A}$, the quantity* $\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

In case the adversary wins Attack Game 13.1, the pair (m, σ) it outputs is called an **existential forgery**. Systems that satisfy Definition 13.2 are said to be **existentially unforgeable under a chosen message attack**.

Verification queries. In our discussion of MACs we proved Theorem 6.1, which showed that tag verification queries do not help the adversary forge MACs. In the case of digital signatures, verification queries are a non-issue — the adversary can always verify message-signature pairs for himself. Hence, there is no need for an analogue to Theorem 6.1 for digital signatures.

Security against multi-key attacks. In real systems there are many users, and each one of them can have a signature key pair (pk_i, sk_i) for $i = 1, \ldots, n$. Can a chosen message attack on pk_1 help the adversary forge signatures for pk_2? If that were possible then our definition of secure signature would be inadequate since it would not model real-world attacks. Just as we did for other security primitives, one can generalize the notion of a secure signatures to the multi-key setting, and prove that a secure signature is also secure in the multi-key settings. See Exercise 13.2. We proved a similar fact for a secure MAC system in Exercise 6.3.

Strongly unforgeable signatures Our definition of existential forgery is a little different than the definition of secure MACs. Here we only require that the adversary cannot forge a signature on a new message m. We do not preclude the adversary from producing a new signature on m from some other signature on m. That is, a signature scheme is secure even if the adversary can transform a valid pair (m, σ) into a new valid pair (m, σ').

In contrast, for MAC security we insisted that given a message-tag pair (m, t) the adversary cannot create a new valid tag $t' \neq t$ for m. This was necessary for proving security of the encrypt-then-MAC construction in Section 9.4.1. It was also needed for proving that MAC verification queries do not help the adversary (see Theorem 6.1 and Exercise 6.7).

One can similarly strengthen Definition 13.2 to require this more stringent notion of existential unforgeability. We capture this in the following modified attack game.

Attack Game 13.2. For a given signature scheme $\mathcal{S} = (G, S, V)$, and a given adversary $\mathcal{A}$, the game is identical to Attack Game 13.1, except that the second bullet in the winning condition is changed to:

- (m, σ) is new, namely $(m, \sigma) \notin \big\{(m_1, \sigma_1),\ (m_2, \sigma_2), \ldots\big\}$

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$, denoted $\mathrm{stSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 13.3. *We say that a signature scheme $\mathcal{S}$ is* ***strongly secure*** *if for all efficient adversaries $\mathcal{A}$, the quantity* $\mathrm{stSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

Strong security ensures that for a secure signature scheme, the adversary cannot create a new signature on a previously signed message, as we required for MACs. There are a few specific situations that require signatures satisfying this stronger security notion, such as [56, 32] and a signcryption construction described in Section 13.7. However, most often Definition 13.2 is sufficient. At any rate, any secure signature scheme $\mathcal{S} = (G, S, V)$ can be converted into a strongly secure signature scheme $\mathcal{S}' = (G', S', V')$. See Exercise 14.10.

13.1.1.1 Strong binding

Definition 13.2 ensures that generating a valid message-signature pair is difficult without the secret key. The definition, however, does not capture several additional desirable properties for a signature scheme. Specifically, it does not preclude a signer from misbehaving in the following ways.

Message confusion: what was signed? Definition 13.2 does not preclude the signer from obtaining two distinct messages m and m' and a signature σ, so that σ is a valid signature for both m and m'. The message m might say "Alice owes Bob ten dollars" while m' says "Alice owes Bob one dollar." Since σ is a valid signature for both messages, a judge cannot tell what message Alice actually signed. If a signer can make this happen, we say the scheme is vulnerable to **message confusion**. See Exercise 13.3 for such a scheme.

In common applications of digital signatures the signer is bound to every message that has a valid signature by the signer. In the example above, Alice would be bound to both messages, and owe Bob eleven dollars. Consequently, in this setting, there is no harm if the signer produces two distinct messages with the same signature. However, in other settings, this may be undesirable. The constructions given in this chapter and the next are not vulnerable to message confusion. Indeed, for these constructions, a signature is a *binding commitment* to the message, which means that the signer cannot produce a public key, a signature, and two distinct messages so that the signature is valid for both messages.

Signer confusion: who signed? Let $\mathcal{S} = (G, S, V)$ be a signature scheme and let (m, σ) be a valid message-signature pair with respect to some public key pk. Suppose an attacker who sees (pk, m, σ), can generate a new public key pk', where $pk' \neq pk$, such that (m, σ) is also valid with respect to the public key pk'. This can cause confusion over who signed m: was it pk or was it pk'? Both can claim ownership of (m, σ).

A signature scheme that can be attacked in this way is said to be vulnerable to **signer confusion**. Exercise 13.4 gives an example of a signature scheme that is vulnerable to signer confusion. In this example the attacker not only generates pk', but also the corresponding secret key sk'.

Signer confusion can lead to some undesirable consequences. For example, suppose (m, σ) is a signed homework solution set submitted by a student Alice. After the submission deadline, an attacker Molly, who did not submit a solution set, can use signer confusion to claim that the homework submission (m, σ) is hers. To do so, Molly generates a public key pk' such that (m, σ) is a valid message-signature pair for the key pk'. Because the assignment is properly signed with respect to both public keys, pk and pk', the Professor cannot tell who submitted the assignment (assuming the homework m does not identify Alice). In practice, signer confusion has been used to attack certain key exchange protocols.

Strongly binding signatures. In Exercise 13.5 we define the concept of a strongly binding signature scheme which prevents both message confusion and signer confusion. The exercise shows that, if needed, any signature scheme can be made strongly binding by having the signer append a collision resistant hash of (pk, m) to the signature (Exercise 13.5(c)).

13.1.2 Mathematical details

As usual, we give a more mathematically precise definition of a signature, using the terminology defined in Section 2.3. This section may be safely skipped on first reading.

Definition 13.4 (Signature). *A* ***signature*** *scheme is a triple of efficient algorithms (G, S, V), along with two families of spaces with system parameterization P:*

$$\mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \textit{and} \quad \mathbf{\Sigma} = \{\Sigma_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

As usual, $\lambda \in \mathbb{Z}_{\geq 1}$ is a security parameter and $\Lambda \in \mathrm{Supp}(P(\lambda))$ is a system parameter. We require that

1. *$\mathbf{M}$ and $\mathbf{\Sigma}$ are efficiently recognizable.*
2. *Algorithm G is an efficient probabilistic algorithm that on input λ, Λ, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, outputs a pair (pk, sk), where pk and sk are bit strings whose lengths are always bounded by a polynomial in λ.*
3. *Algorithm S is an efficient probabilistic algorithm that on input λ, Λ, sk, m, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $(pk, sk) \in \mathrm{Supp}(G(\lambda, \Lambda))$ for some pk, and $m \in \mathcal{M}_{\lambda,\Lambda}$, always outputs an element of $\Sigma_{\lambda,\Lambda}$.*
4. *Algorithm V is an efficient deterministic algorithm that on input $\lambda, \Lambda, pk, m, \sigma$, where $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, $(pk, sk) \in \mathrm{Supp}(G(\lambda, \Lambda))$ for some sk, $m \in \mathcal{M}_{\lambda,\Lambda}$, and $\sigma \in \Sigma_{\lambda,\Lambda}$, and outputs either* accept *or* reject.

In defining security, we parameterize Attack Game 13.1 by the security parameter λ which is given to both the adversary and the challenger. The advantage $\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ is then a function of λ. Definition 13.2 should be read as saying that $\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}](\lambda)$ is a negligible function. Similarly for Definition 13.3.

13.2 Extending the message space with collision resistant hashing

Suppose we are given a secure digital signature scheme with a small message space, say $\mathcal{M} = \{0, 1\}^{256}$. We show how to extend the message space to much larger messages using a collision resistant hash function. We presented a similar construction for MACs in Fig. 8.1. Let $\mathcal{S} = (G, S, V)$ be a signature scheme defined over $(\mathcal{M}, \Sigma)$ and let $H : \mathcal{M}' \to \mathcal{M}$ be a hash function, where the set $\mathcal{M}'$ is much larger than $\mathcal{M}$. Define a new signature scheme $\mathcal{S}' = (G, S', V')$ over $(\mathcal{M}', \Sigma)$ as

$$S'(sk,\ m) := S(sk,\ H(m)) \quad \text{and} \quad V'(pk,\ m,\ \sigma) := V(pk,\ H(m),\ \sigma) \tag{13.1}$$

The new scheme signs much larger message than the original scheme. This approach is often called the **hash-and-sign paradigm**. As a concrete example, suppose we take H to be SHA256. Then any signature scheme capable of signing 256-bit messages can be securely extended to a signature scheme capable of signing arbitrary long messages. Hence, from now on it suffices to focus on building signature schemes for short 256-bit messages.

The following simple theorem shows that this construction is secure. Its proof is essentially identical to the proof of Theorem 8.1.

Theorem 13.1. *Suppose the signature scheme $\mathcal{S}$ is secure and the hash function H is collision resistant. Then the derived signature scheme $\mathcal{S}' = (G, S', V')$ defined in (13.1) is a secure signature.*

> *In particular, suppose $\mathcal{A}$ is a signature adversary attacking $\mathcal{S}'$ (as in Attack Game 13.1). Then there exist an efficient signature adversary $\mathcal{B}_{\mathcal{S}}$ and an efficient collision finder $\mathcal{B}_H$, which are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}'] \leq \mathrm{SIG}\mathsf{adv}[\mathcal{B}_{\mathcal{S}}, \mathcal{S}] + \mathrm{CR}\mathsf{adv}[\mathcal{B}_H, H]$$

13.2.1 Extending the message space using TCR functions

We briefly show that collision resistance is not necessary for extending the message space of a signature scheme. A second pre-image resistant (SPR) hash function is sufficient. Recall that in Section 8.11.2 we used SPR hash functions to build target collision resistant (TCR) hash functions. We then used a TCR hash function to extend the message space of a MAC. We can do the same here to extend the message space of a signature scheme.

Let H be a TCR hash function defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{T})$. Let $\mathcal{S} = (G, S, V)$ be a signature scheme for short messages in $\mathcal{K}_H \times \mathcal{T}$. We build a new signature scheme $\mathcal{S}' = (G, S', V')$ for signing messages in $\mathcal{M}$ as follows:

```
S'(sk, m) :=                               V'(pk, m, (σ, r) ) :=
        r ←R K_H                                   h ← H(r, m)                 (13.2)
        h ← H(r, m)                                Output V(pk, (r, h), σ)
        σ ← S(sk, (r, h))
        Output (σ, r)
```

The signing procedure chooses a random TCR key r, includes r as part of the message being signed, and outputs r as part of the final signature. As a result, signatures produced by this scheme are longer than signatures produced by extending the domain using a collision resistant hash, as above. Using the TCR construction from Fig. 8.15, the length of r is logarithmic in the size of the message being signed. This extra logarithmic size key must be included in every signature. Exercise 13.7 proposes a way to get shorter signatures.

The benefit of the TCR construction is that security only relies on H being TCR, which is a much weaker property than collision resistance and hence more likely to hold for H. For example, the function SHA256 may eventually be broken as a collision-resistant hash, but the function $H(r, m) := \mathrm{SHA256}(r \parallel m)$ may still be secure as a TCR.

The following theorem proves security of the construction in (13.2) above. The theorem and its proof are almost identical to the same theorem and proof applied to MAC systems (Theorem 8.14). Note that the concrete bound in the theorem below has an extra factor of Q that does not appear in Theorem 13.1 above. The reason for this extra Q factor is the same as in the proof for MAC systems (Theorem 8.14).

Theorem 13.2. *Suppose $\mathcal{S} = (G, S, V)$ is a secure signature scheme and the hash function H is TCR. Then the derived signature scheme $\mathcal{S}' = (G, S', V')$ defined in (13.2) is secure.*

> *In particular, for every signature adversary $\mathcal{A}$ attacking $\mathcal{S}'$ (as in Attack Game 13.1) that issues at most Q signing queries, there exist an efficient signature adversary $\mathcal{B}_{\mathcal{S}}$ and an efficient TCR adversary $\mathcal{B}_H$, which are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}'] \le \mathrm{SIG}\mathsf{adv}[\mathcal{B}_{\mathcal{S}}, \mathcal{S}] + Q \cdot \mathrm{TCR}\mathsf{adv}[\mathcal{B}_H, H].$$

13.3 Signatures from trapdoor permutations: the full domain hash

We now turn to constructing signature schemes. Secure signature schemes can be built from many cryptographic primitives.

- *Signature schemes from hash functions:* In Chapter 14 we will construct signature schemes that require nothing more than a collision resistant hash function. These signature schemes can have fast signing and verification algorithms, but the resulting signatures are several killobytes long, much longer than signatures generated by an algebraic signature scheme. As such, they can be useful for signing software distribution packages. These packages tend to be quite large and the signature, even a relatively long one, has little impact on the overall package size. An important benefit of hash-based schemes is that they remain secure against an adversary who has access to a quantum computer, and are thus said to be **post-quantum secure** (we discuss quantum computing attacks in Sections 4.3.4 and 16.5).

- *Signature schemes from a trapdoor permutation:* This is the topic of this chapter. As we will see in a moment, a trapdoor permutation gives a simple and direct construction for a signature scheme.

- *Signature schemes from discrete log:* In Chapters 15 and 19 we will construct signature schemes using a finite cyclic group where the discrete log problem is difficult. Signatures and public keys in these schemes are short, only a few tens of bytes. These signature schemes have many useful properties, and are frequently used in networking and financial applications.

The signature schemes presented in this chapter are proven secure in the random oracle model. We will present practical non-random-oracle constructions in Chapters 14 and 15.

Now, let us build our first signature scheme.

The full domain hash signature scheme. We begin with a simple construction based on a trapdoor permutation. We then present a concrete signature scheme from the only trapdoor permutation we have, namely RSA. Recall that a trapdoor permutation scheme defined over $\mathcal{X}$ is a triple of algorithms $\mathcal{T} = (G, F, I)$, where G generates a public key/secret key pair (pk, sk), $F(pk, \cdot)$ evaluates a permutation on $\mathcal{X}$ in the forward direction, and $I(sk, \cdot)$ evaluates the permutation in the reverse direction. See Section 10.2 for details.

We show that a trapdoor permutation $\mathcal{T}$ gives a simple signature scheme. The only other ingredient we need is a hash function H that maps messages in $\mathcal{M}$ to elements in $\mathcal{X}$. This function will be modeled as a random oracle in the security analysis. The signature scheme, called **full domain hash** (FDH), denoted $\mathcal{S}_{\text{FDH}}$, works as follows:

- The key generation algorithm for $\mathcal{S}_{\text{FDH}}$ is the key generation algorithm G of the trapdoor permutation scheme $\mathcal{T}$. It outputs a pair (pk, sk).
- The signature on m is simply the inverse of $H(m)$ with respect to the function $F(pk, \cdot)$. That is, to sign a message $m \in \mathcal{M}$ using sk, the signing algorithm S runs as follows:

$$\begin{aligned} S(sk, m) := \quad & y \leftarrow H(m), \quad \sigma \leftarrow I(sk, y) \\ & \text{output } \sigma. \end{aligned}$$

- To verify a signature σ on a message m the verification algorithm V checks that $F(pk, \sigma)$ is equal to $H(m)$. More precisely, V works as follows:

$$\begin{aligned} V(pk, m, \sigma) := \quad & y \leftarrow F(pk, \sigma) \\ & \text{if } y = H(m) \text{ output } \mathsf{accept}\text{; otherwise, output } \mathsf{reject}. \end{aligned}$$

We will analyze $\mathcal{S}_{\text{FDH}}$ by modeling the hash function H as a random oracle. Recall that in the random oracle model (see Section 8.10), the function H is modeled as a random function $\mathcal{O}$ chosen at random from the set of all functions $\text{Funs}[\mathcal{M}, \mathcal{X}]$. More precisely, in the random oracle version of Attack Game 13.1, the challenger chooses $\mathcal{O}$ at random. In any computation where the challenger would normally evaluate H, it evaluates $\mathcal{O}$ instead. In addition, the adversary is allowed to ask the challenger for the value of the function $\mathcal{O}$ at any point of its choosing. The adversary may make any number of such "random oracle queries" at any time of its choosing. We use $\text{SIG}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{FDH}}]$ to denote $\mathcal{A}$'s advantage against $\mathcal{S}_{\text{FDH}}$ in the random oracle version of Attack Game 13.1.

Theorem 13.3. *Let $\mathcal{T} = (G, F, I)$ be a one-way trapdoor permutation defined over $\mathcal{X}$. Let $H : \mathcal{M} \to \mathcal{X}$ be a hash function. Then the derived FDH signature scheme $\mathcal{S}_{\text{FDH}}$ is a secure signature scheme when H is modeled as a random oracle.*

> *In particular, let $\mathcal{A}$ be an efficient adversary attacking $\mathcal{S}_{\text{FDH}}$ in the random oracle version of Attack Game 13.1. Moreover, assume that $\mathcal{A}$ issues at most Q_{ro} random oracle queries and Q_{s} signing queries. Then there exists an efficient inverting adversary $\mathcal{B}$ that attacks $\mathcal{T}$ as in Attack Game 10.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{SIG}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{FDH}}] \leq (Q_{\text{ro}} + 1) \cdot \text{OW}\mathsf{adv}[\mathcal{B}, \mathcal{T}] \tag{13.3}$$

An overview of the proof of security for $\mathcal{S}_{\text{FDH}}$. We defer the full proof of Theorem 13.3 to Section 13.4.2. For now, we sketch the main ideas. To forge a signature on a message m, an adversary has to compute $\sigma = I(sk, y)$, where $y = H(m)$. With H modeled as a random oracle, the value y is essentially just a random point in $\mathcal{X}$, and so this should be hard to do, assuming $\mathcal{T}$ is one way. Unfortunately, this argument does not deal with the fact that in a chosen message attack, the adversary can get arbitrary messages signed before producing its forgery. Again, since H is modeled as a random oracle, this effectively means that to break the signature scheme, the adversary must win the following game: after seeing several random points $y_1, y_2, \ldots$ in $\mathcal{X}$ (corresponding to the hash outputs on various messages), the adversary can ask to see preimages of some of the y_i's (corresponding to the signing queries), and then turn around and produce the preimage of one of the remaining y_i's. It turns out that winning this game is not too much easier than breaking the one-wayness of $\mathcal{T}$ in the usual sense. This will be proved below in Lemma 13.5 using a kind of "guessing argument": in the reduction, we will have to guess in advance at which of the random points the adversary will invert $F(pk, \cdot)$. This is where the factor $Q_{\text{ro}} + 1$ in (13.3) comes from.

Unique signatures. The $\mathcal{S}_{\text{FDH}}$ scheme is a **unique signature scheme**: for a given public key, every message m has a *unique* signature σ that will be accepted as valid for m by the verification algorithm. This means that if $\mathcal{S}_{\text{FDH}}$ is secure, it must also be strongly secure in the sense of Definition 13.3.

The importance of hashing. The hash function H is crucial to the security of $\mathcal{S}_{\text{FDH}}$. Without first hashing the message, the system is trivially insecure. To see why, suppose we incorrectly define the signature on $m \in \mathcal{X}$ as $\sigma := I(sk, m)$. That is, we apply I without first hashing m. Then to forge a signature, the adversary simply chooses a random $\sigma \in \mathcal{X}$ and computes $m \leftarrow F(pk, \sigma)$. The pair (m, σ) is an existential forgery. Note that this forgery is created without using the chosen message attack. Of course this m is likely to be gibberish, but is a valid existential forgery.

This attack shows that the hash function H plays a central role in ensuring that $\mathcal{S}_{\text{FDH}}$ is secure. Unfortunately, we can only prove security when H is modeled as a random oracle. We cannot prove security of $\mathcal{S}_{\text{FDH}}$, when H is a concrete hash function, using standard assumptions about $\mathcal{T}$ and H.

13.3.1 Signatures based on the RSA trapdoor permutation

We instantiate the $\mathcal{S}_{\text{FDH}}$ construction with the only trapdoor permutation at our disposal, namely RSA. We obtain the **RSA full domain hash** signature scheme, denoted $\mathcal{S}_{\text{RSA-FDH}}$. Recall that parameters for RSA are generated using algorithm $\text{RSAGen}(\ell, e)$ which outputs a pair (pk, sk) where $pk = (n, e)$. Here n is a product of two ℓ-bit primes. The RSA trapdoor permutation $F(pk, \cdot) : \mathbb{Z}_n \to \mathbb{Z}_n$ is defined as $F(pk, x) := x^e$.

For each public key $pk = (n, e)$, the $\mathcal{S}_{\text{RSA-FDH}}$ system needs a hash function H that maps messages in $\mathcal{M}$ to $\mathbb{Z}_n$. This is a problem — the output space of H depends on n which is different for every public key. Since hash functions generally have a fixed output space, it is preferable that the range of H be fixed and independent of n. To do so, we define the range of H to be $\mathcal{Y} := \{1, \ldots, 2^{2\ell-2}\}$ which, when embedded in $\mathbb{Z}_n$, covers a large fraction of $\mathbb{Z}_n$, for all the RSA moduli n output by $\text{RSAGen}(\ell, e)$.

We describe the signature scheme $\mathcal{S}_{\text{RSA-FDH}}$ using a hash function H defined over $(\mathcal{M}, \mathcal{Y})$. We chose $\mathcal{Y}$ as above so that $|\mathcal{Y}| \geq n/4$ for all n output by $\text{RSAGen}(\ell, e)$. This is necessary for the proof of security. Because an RSA modulus n is large, at least 2048 bits, the hash function H must produce a large output, approximately 2048 bits long. One cannot simply use SHA256. We described appropriate long-output hash functions in Section 8.10.2.

For a given hash function $H : \mathcal{M} \to \mathcal{Y}$, the $\mathcal{S}_{\text{RSA-FDH}}$ signature scheme works as follows:

- the key generation algorithm G uses parameters ℓ and e and runs as follows:

$$\begin{aligned} G() := \quad & (n, d) \xleftarrow{\text{R}} \text{RSAGen}(\ell, e), \quad pk \leftarrow (n, e), \quad sk \leftarrow (n, d) \\ & \text{output } (pk, sk); \end{aligned}$$

- for a given secret key $sk = (n, d)$, and message $m \in \mathcal{M}$, algorithm S runs as follows:

$$\begin{aligned} S(sk, m) := \quad & y \leftarrow H(m) \in \mathcal{Y}, \qquad \sigma \leftarrow y^d \in \mathbb{Z}_n \\ & \text{output } \sigma; \end{aligned}$$

- for a given public key $pk = (n, e)$ the verification algorithm runs as follows:

$$\begin{aligned} V(pk, m, \sigma) := \quad & y \leftarrow \sigma^e \in \mathbb{Z}_n \\ & \text{if } y = H(m) \text{ output } \mathsf{accept}\text{; otherwise, output } \mathsf{reject}. \end{aligned}$$

Signing and verification speed. Recall that typically the public key exponent e is small, often $e = 3$ or $e = 65537$, while the secret key exponent d is as large as n. Consequently, signature generation, which uses a d exponentiation, is much slower than signature verification. In fact, RSA has the fastest signature verification algorithm among all the standardized signature schemes. This makes RSA very attractive for applications where a signature is generated offline, but needs to be quickly verified online. Certificates used in a public key infrastructure are a good example where fast verification is attractive. We discuss ways to speed-up the RSA signing procedure in Chapter 16.

Signature size. One downside of RSA is that the signatures are much longer than in other signature schemes, such as the ones presented in Chapter 19. To ensure that factoring the RSA modulus n is sufficiently difficult, the size of n must be at least 2048 bits (256 bytes). As a result, RSA signatures are 256 bytes, which is considerably longer than in other schemes. This causes difficulties in heavily congested or low bandwidth networks as well as in applications where space is at a premium. For example, at one point the post office looked into printing digital signatures on postage stamps. The signatures were intended to authenticate the recipient's address and were to be encoded as a two dimensional bar code on the stamp. RSA signatures were quickly ruled out because there is not enough space on a postage stamp. We will discuss short signatures in Section 15.5.

The importance of hashing. We showed above that $\mathcal{S}_{\text{FDH}}$ is insecure without first hashing the message. In particular, consider the **unhashed RSA system** where a signature on $m \in \mathbb{Z}_n$ is defined as $\sigma := m^d$. We showed that this system is insecure since anyone can create an existential forgery (m, σ). Recall, however, that this attack typically forges a signature on a message m that is likely to be gibberish.

We can greatly strengthen the attack on this unhashed RSA using the random self-reducibility property of RSA (see Exercise 10.28). In particular, we show that an attacker can obtain the signature on any message m of his choice by issuing a single signing query for a random $\hat{m} \in \mathbb{Z}_n^*$. Let (n, e) be an RSA public key and let $m \in \mathbb{Z}_n$ be some message. As the reader should verify, we may assume that $m \in \mathbb{Z}_n^*$. To obtain the signature on m the attacker does the following:

```
$r \xleftarrow{R} \mathbb{Z}_n^*$,   $\hat{m} \leftarrow m \cdot r^e$
Request the signature on $\hat{m}$ and obtain $\hat{\sigma}$
Output $\sigma \leftarrow \hat{\sigma}/r$
```

Indeed, if $\hat{\sigma}^e = \hat{m}$ then $\sigma \in \mathbb{Z}_n$ is a valid signature on m since

$$\sigma^e = (\hat{\sigma}/r)^e = \hat{\sigma}^e/r^e = \hat{m}/r^e = m. \tag{13.4}$$

The attack shows that by fooling the user into signing a random message $\hat{m}$ the adversary can obtain the signature on a message m of his choice. We say that unhashed RSA signatures are **universally forgeable** and thus should never be used.

Surprisingly, the fact that an attacker can convert a signature on a random message into a signature on a chosen message turns out to play a central role in the construction of so called blind signatures. Blind signatures are used in protocols for anonymous electronic cash and anonymous electronic voting. In both applications blind signatures are the main ingredient for ensuring privacy (see Exercise 13.15).

Security of RSA full domain hash. Recall that the security proof for the general full domain hash $\mathcal{S}_{\text{FDH}}$ (Theorem 13.3) was very loose: an adversary $\mathcal{A}$ with advantage ϵ in attacking $\mathcal{S}_{\text{FDH}}$ gives an adversary $\mathcal{B}$ with advantage $\epsilon/(Q_{\text{ro}}+Q_{\text{s}}+1)$ in attacking the underlying trapdoor permutation.

Can we do better? Indeed, we can: using the random self-reducibility property of RSA, we can prove security with a much tighter bound, as shown in Theorem 13.4 below. In particular, the factor $Q_{\text{ro}}+Q_{\text{s}}+1$ is replaced by (approximately) Q_{s}. This is significant, because in a typical attack, the number of signing queries Q_{s} is likely to be much smaller than the number of random oracle queries Q_{ro}. Indeed, on the one hand, Q_{ro} represents the number of times an attacker evaluates the hash function H. These computations can be done by the attacker "off line," and the attacker is only bounded by his own computing resources. On the other hand, each signing query requires that an honest user sign a message. Concretely, a conservative bound on Q_{ro} could perhaps be as large as 2^{128}, while Q_{s} could perhaps be reasonably bounded by 2^{40}. We thus obtain a much tighter reduction for $\mathcal{S}_{\text{RSA-FDH}}$ than for $\mathcal{S}_{\text{FDH}}$ with a general trapdoor permutation. However, even for $\mathcal{S}_{\text{RSA-FDH}}$ the reduction is not tight due to the Q_{s} factor. We will address that later in Section 13.5.

As in the proof of $\mathcal{S}_{\text{FDH}}$, our security proof for $\mathcal{S}_{\text{RSA-FDH}}$ models the hash function $H : \mathcal{M} \to \mathcal{Y}$ as a random oracle. The proof requires that $\mathcal{Y}$ is a large subset of $\mathbb{Z}_n$ (we specifically assume that $|\mathcal{Y}| \geq n/4$, but any constant fraction would do). In what follows, we use 2.72 as an upper bound on the base of the natural logarithm $e \approx 2.718$ (not to be confused with the RSA public exponent e).

Theorem 13.4. *Let $H : \mathcal{M} \to \mathcal{Y}$ be a hash function, where $\mathcal{Y} = \{1, \ldots, 2^{2\ell-2}\}$. If the RSA assumption holds for (ℓ, e), then $\mathcal{S}_{\text{RSA-FDH}}$ with parameters (ℓ, e) is a secure signature scheme, when H is modeled as a random oracle.*

> *In particular, let $\mathcal{A}$ be an efficient adversary attacking $\mathcal{S}_{\text{RSA-FDH}}$ in the random oracle version of Attack Game 13.1. Moreover, assume that $\mathcal{A}$ issues at most Q_{s} signing queries. Then there exists an efficient RSA adversary $\mathcal{B}$ as in Attack Game 10.3, where and $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*

$$\text{SIG}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{RSA-FDH}}] \leq 2.72 \cdot (Q_{\text{s}} + 1) \cdot \text{RSA}\mathsf{adv}[\mathcal{B}, \ell, e] \tag{13.5}$$

We defer the proof of Theorem 13.4 to Section 13.4.2.

13.4 Security analysis of full domain hash

The goal of this section is to analyze the security of the the full domain hash signature scheme; specifically, we prove Theorems 13.3 and 13.4. We begin with a tool that will be helpful, and is interesting and useful in its own right.

13.4.1 Repeated one-way functions: a useful lemma

Let f be a one-way function over $(\mathcal{X}, \mathcal{Y})$. Briefly, this means that given $y \leftarrow f(x)$ for a random $x \in \mathcal{X}$, it is difficult to find a pre-image of y. This notion was presented in Definition 8.6.

Consider the following, seemingly easier, problem: we give the adversary $\big(f(x_1), \ldots, f(x_t)\big)$ and allow the adversary to request some, but not all, of the x_i's. To win, the adversary must produce one of the remaining x_i's. We refer to this as the **t-repeated one-way problem**. More precisely, the problem is defined using the following game.

Attack Game 13.3 (t-repeated one-way problem). For a given positive integer t and a given adversary $\mathcal{A}$, the game runs as follows:

- The challenger computes

  $$x_1, \ldots, x_t \xleftarrow{\text{R}} \mathcal{X}, \quad y_1 \leftarrow f(x_1), \ldots, y_t \leftarrow f(x_t)$$

  and sends $(y_1, \ldots, y_t)$ to the adversary.

- $\mathcal{A}$ makes a sequence of *reveal queries*. Each reveal query consists of an index $j \in \{1, \ldots, t\}$. Given j, the challenger sends x_j to $\mathcal{A}$.

- Eventually, $\mathcal{A}$ the adversary outputs (ν, x), where $\nu \in \{1, \ldots, t\}$ and $x \in \mathcal{X}$.

We say that $\mathcal{A}$ wins the game if index ν is not among $\mathcal{A}$'s reveal queries, and $f(x) = y_\nu$. We define $\mathcal{A}$'s advantage, denoted $\text{rOW}\mathsf{adv}[\mathcal{A}, f, t]$, as the probability that $\mathcal{A}$ wins the game. □

The following lemma shows that the repeated one-way problem is equivalent to the standard one-way problem given in Definition 8.6. That is, winning in Attack Game 13.3 is not much easier than inverting f.

Lemma 13.5. *For every t-repeated one-way adversary $\mathcal{A}$ there exists a standard one-way adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*

$$\text{rOW}\mathsf{adv}[\mathcal{A}, f, t] \leq t \cdot \text{OW}\mathsf{adv}[\mathcal{B}, f]. \tag{13.6}$$

Proof idea. The proof is a kind of "guessing argument", somewhat similar to what we did, for example, in the proof of Theorem 6.1. We want to use $\mathcal{A}$ to build an adversary $\mathcal{B}$ that breaks the one-wayness of f. So $\mathcal{B}$ starts with $y_* \in \mathcal{Y}$ and wants to find a preimage of y_* under f, using $\mathcal{A}$ as a subroutine. The first thing that $\mathcal{B}$ does is make a guess ω at the value of the index ν that $\mathcal{A}$ will ultimately choose. Our adversary $\mathcal{B}$ then prepares values $y_1, \ldots, y_t \in \mathcal{Y}$ as follows: for $i \neq \omega$, it sets $y_i \leftarrow f(x_i)$ for random $x_i \in \mathcal{X}$; it also sets $y_\omega \leftarrow y_*$. It then sends $(y_1, \ldots, y_t)$ to $\mathcal{A}$, as in Attack Game 13.3. If $\mathcal{B}$'s guess was correct (which happens with probability $1/t$), it will be able to respond to all of $\mathcal{A}$'s queries, and $\mathcal{A}$'s final output will provide the preimage of y that $\mathcal{B}$ was looking for. □

Proof. In more detail, our adversary $\mathcal{B}$ is given $y_* := f(x_*)$ for a random $x_* \in \mathcal{X}$, and then plays the role of challenger to $\mathcal{A}$ as in Attack Game 13.3 as follows:

```
Initialize:
        $x_1, \ldots, x_t \xleftarrow{\text{R}} \mathcal{X}$
        $y_1 \leftarrow f(x_1), \ldots, y_t \leftarrow f(x_t)$
        $\omega \xleftarrow{\text{R}} \{1, \ldots, t\}, \quad y_\omega \leftarrow y_*$   // Plug $y_*$ at position $\omega$
        Send $(y_1, \ldots, y_t)$ to $\mathcal{A}$
// $\mathcal{B}$ now knows pre-images for all $y_i$'s other than $y_\omega$
Upon receiving a query $j \in \{1, \ldots, t\}$ from $\mathcal{A}$:
        if $j \neq \omega$
                then send $x_j$ to $\mathcal{A}$
                else  output fail and stop
When $\mathcal{A}$ outputs a pair $(\nu, x)$:
        if $\nu = \omega$
                then output $x$ and stop
                else  output fail and stop
```

Now we argue that the inequality (13.6) holds.

Define Game 0 to be the game played between $\mathcal{A}$ and the challenger in Attack Game 13.3, and let W_0 be the event that $\mathcal{A}$ wins the game.

Now define a new Game 1, which is the same as Game 0, except that the challenger chooses $\omega \in \{1, \ldots, t\}$ at random. Also, we say that $\mathcal{A}$ wins Game 1 if it wins as in Game 0 with output (ν, x) such that $\nu = \omega$. Define W_1 to be the event that $\mathcal{A}$ wins Game 1.

We can think of Games 0 and 1 as operating on the same underlying probability space. Really, the two games are exactly the same: all that changes is the winning condition. Moreover, as ω is independent of everything else, we have

$$\Pr[W_1] = \Pr[W_0 \wedge \nu = \omega] = \Pr[W_0] \cdot \Pr[\nu = \omega \mid W_0] = (1/t) \cdot \Pr[W_0].$$

Moreover, it is clear that $\text{OW}\mathsf{adv}[\mathcal{B}, f] = \Pr[W_1]$. Adversary $\mathcal{B}$ is really just playing Game 1 — it only aborts when it is clear that it will not win Game 1 anyway, and it wins Game 1 if and only if it succeeds in finding a preimage of y_*. □

Application to trapdoor functions. Lemma 13.5 applies equally well to trapdoor functions. If $\mathcal{T} = (G, F, I)$ is a trapdoor function scheme defined over $(\mathcal{X}, \mathcal{Y})$, then $\mathcal{T}$ is one way in the sense of Definition 10.3 if and only if $f := F(pk, \cdot)$ is one way in the sense of Definition 8.6. Indeed, for any adversary, the respective advantages in the corresponding attack games are equal. Technically, with $f := F(pk, \cdot)$, the public key pk is viewed as a "system parameter" defining f.

A tighter reduction for RSA. For a general one-way function f, the concrete bound in Lemma 13.5 is quite poor: if adversary $\mathcal{A}$ has advantage ϵ in winning the t-repeated one-way game, then the lemma constructs a one-way attacker with advantage only ϵ/t.

When f is derived from the RSA function we can obtain a tighter reduction using the random self-reducibility property of RSA. We replace the factor t by a factor of (about) Q, where Q is the number of reveal queries from $\mathcal{A}$. This Q is usually much smaller than t.

We first restate Attack Game 13.3 as it applies to the RSA function. We slightly tweak the game and require that the images $y_1, \ldots, y_t$ given to $\mathcal{A}$ lie in a certain large subset of $\mathbb{Z}_n$ denoted $\mathcal{Y}$. For

RSA parameters ℓ and e, we set $\mathcal{Y} := \{1, 2, \ldots, 2^{2\ell-2}\}$ so that for all n generated by RSAGen(ℓ, e), we have $|\mathcal{Y}| \geq n/4$.

Attack Game 13.4 (t-repeated RSA). For given RSA parameters ℓ and e, a given positive integer t, and a given adversary $\mathcal{A}$, the game runs as follows:

- The challenger computes

  $(n, d) \xleftarrow{\text{R}} \text{RSAGen}(\ell, e)$
  $y_1, \ldots, y_t \xleftarrow{\text{R}} \mathcal{Y}$ // *Recall that* $\mathcal{Y} := \{1, 2, \ldots, 2^{2\ell-2}\}$

  and sends (n, e) and $(y_1, \ldots, y_t)$ to $\mathcal{A}$.

- $\mathcal{A}$ makes a sequence of *reveal queries*. Each reveal query consists of an index $j \in \{1, \ldots, t\}$. Given j, the challenger sends $x_j := y_j^d \in \mathbb{Z}_n$ to $\mathcal{A}$.

- Eventually the adversary outputs (ν, x), where $\nu \in \{1, \ldots, t\}$ and $x \in \mathbb{Z}_n$.

We say that $\mathcal{A}$ wins the game if index ν is not among $\mathcal{A}$'s reveal queries, and $x^e = y_\nu$. We define $\mathcal{A}$'s advantage, denoted rRSA$\mathsf{adv}[\mathcal{A}, \ell, e, t]$, as the probability that $\mathcal{A}$ wins the game. □

We show that the t-repeated RSA problem is equivalent to the basic RSA problem, but with a tighter concrete bound than in Lemma 13.5. In particular, the factor of t is replaced by $2.72 \cdot (Q+1)$. The constant 2.72 is an upper on the base of the natural logarithm $e \approx 2.718$.

Lemma 13.6. *Let ℓ and e be RSA parameters. For every t-repeated RSA adversary $\mathcal{A}$ that makes at most Q reveal queries, there exists a standard RSA adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*

$$\text{rRSA}\mathsf{adv}[\mathcal{A}, \ell, e, t] \leq 2.72 \cdot (Q+1) \cdot \text{RSA}\mathsf{adv}[\mathcal{B}, \ell, e]. \tag{13.7}$$

Proof idea. The proof is similar to that of Lemma 13.5. In that proof, we plugged the challenge instance y_* of the one-way attack game at a random position among the y_i's, and using $\mathcal{A}$, we succeed if $\mathcal{A}$ does not issue a reveal query at the plugged position, and its output inverts at the plugged position. Now, using the random self-reducibility property for RSA, we take the challenge y_*, and "spread it around," plugging related, randomized versions of y_* at *many* randomly chosen positions. We succeed if $\mathcal{A}$'s reveal queries avoid the plugged positions, but its output inverts at one of them. By increasing the number of plugged positions, the chance of hitting one at the output stage increases (which is good), but the chance of avoiding them during a reveal query decreases (which is bad). Using a clever strategy for sampling the set of plugged positions, we can optimize the success probability to get the desired result. □

Proof. We describe an adversary $\mathcal{B}$ that is given (n, e) and a random $y_* \in \mathbb{Z}_n$, and then attempts to compute an eth root of y_*.

We first deal with an annoying corner case. It may happen (albeit with very small probability) that $y_* \notin \mathbb{Z}_n^*$. However, in this case, it is easy to compute the eth root of y_*: if $y_* = 0$, the eth root is 0; otherwise, $\gcd(y_*, n)$ gives us the prime factorization of n, which allows us to compute the decryption exponent d, and hence the eth root of y_*.

So from now on, we assume $y_* \in \mathbb{Z}_n^*$. Adversary $\mathcal{B}$ uses $\mathcal{A}$ to compute an eth root of y_* as shown in Fig. 13.2. First, $\mathcal{B}$ generates t random values $y_1, \ldots, y_t \in \mathcal{Y}$ and sends them to $\mathcal{A}$. For

```
    Initialize:   // Generate random $y_1, \ldots, y_t \in \mathcal{Y}$
        $\Omega \leftarrow \emptyset$
        for $i = 1, \ldots, t$:
            flip a biased coin $c_i \in \{0,1\}$ such that $\Pr[c_i = 1] = 1/(Q+1)$
            if $c_i = 1$ then $\Omega \leftarrow \Omega \cup \{i\}$
(1)         repeat
                $x_i \xleftarrow{R} \mathbb{Z}_n$,   $y_i \leftarrow x_i^e \cdot y_*^{c_i}$   // So $y_i = x_i^e$ or $y_i = x_i^e \cdot y_*$
            until $y_i \in \mathcal{Y}$
        Send $(n, e)$ and $(y_1, \ldots, y_t)$ to $\mathcal{A}$

    // $\mathcal{B}$ now knows pre-images for all $y_i$ where $i \notin \Omega$
    Upon receiving a reveal query $j \in \{1, \ldots, t\}$ from $\mathcal{A}$:
        if $j \notin \Omega$
            then send $x_j$ to $\mathcal{A}$
            else  output fail and stop

    When $\mathcal{A}$ outputs a pair $(\nu, x)$:
        if $\nu \in \Omega$
(2)         then $\tilde{x} \leftarrow x/x_\nu$, output $\tilde{x}$
            else  output fail and stop
```

Figure 13.2: Algorithm $\mathcal{B}$ in the proof of Lemma 13.6

each $i = 1, \ldots, t$, either $y_i = x_i^e$, in which case $\mathcal{B}$ knows an eth root of y_i and can respond to a reveal query for i, or $y_i = x_i^e \cdot y_*$ in which case $\mathcal{B}$ does not know an eth root of y_i. Here, Ω is the set of indices i for which $\mathcal{B}$ does not know an eth root of y_i.

If $\mathcal{B}$ reaches the line marked (2) and x is an eth root of y_ν, we have

$$\tilde{x}^e = (x/x_\nu)^e = x^e/x_\nu^e = y_\nu/x_\nu^e = (x_\nu^e \cdot y_*)/x_\nu^e = y_*,$$

and so $\mathcal{B}$'s output $\tilde{x}$ is an eth root of y_*.

Actually, we have ignored another corner case. Namely, it may happen (again, with very small probability) that the value x_ν computed above does not lie in $\mathbb{Z}_n^*$. However, if that happens, it must be the case that $x_\nu \neq 0$ (since $0 \notin \mathcal{Y}$), and as in the other corner case, we can use x_ν to factor n and compute the decryption exponent.

Let us analyze the repeat/until loop at the line marked (1) for a fixed $i = 1, \ldots, t$. Since $y_* \in \mathbb{Z}_n^*$, each candidate value for y_i generated in the loop body is uniformly distributed over $\mathbb{Z}_n$. Since $|\mathcal{Y}| \geq n/4$, the probability that each candidate y_i lies in $\mathcal{Y}$ at at least $1/4$. Therefore, the expected number of loop iterations is at most 4. Moreover, when the loop terminates, the final value of y_i is uniformly distributed over $\mathcal{Y}$.

We now argue that (13.7) holds. The basic structure of the argument is the same as in Lemma 13.5. Define Game 0 to be the game played between $\mathcal{A}$ and the challenger in Attack Game 13.4, and let W_0 be the event that $\mathcal{A}$ wins the game.

Now define a new Game 1, which is the same as Game 0, except that the challenger generates a set of indices $\Omega \subseteq \{1, \ldots, t\}$, as follows: each $i = 1, \ldots, t$ is independently added to Ω with probability $1/(Q+1)$. Let $\mathcal{R}$ be the set of reveal queries made by $\mathcal{A}$. We say that $\mathcal{A}$ wins Game 1 if it wins as in Game 0 with output (ν, x), and in addition, $\mathcal{R} \cap \Omega = \emptyset$ and $\nu \in \Omega$. Define W_1 to be the event that $\mathcal{A}$ wins Game 1. We have

$$\Pr[W_1] = \Pr\big[W_0 \text{ and } \mathcal{R} \cap \Omega = \emptyset \text{ and } \nu \in \Omega\big] = \Pr[W_0] \cdot \Pr\big[\mathcal{R} \cap \Omega = \emptyset \text{ and } \nu \in \Omega \;\big|\; W_0\big].$$

Moreover, it is not hard to see that

$$\mathrm{RSA}\mathsf{adv}[\mathcal{B}, \ell, e] \geq \Pr[W_1].$$

Indeed, when $\mathcal{B}$'s input y_* lies in $\mathbb{Z}_n^*$, adversary $\mathcal{B}$ is essentially just playing Game 1: the distributions of $(y_1, \ldots, y_t, \Omega)$ are identical in both games. The condition $\mathcal{R} \cap \Omega = \emptyset$ corresponds to the condition that $\mathcal{B}$ does not abort in processing one of $\mathcal{A}$'s reveal queries. The condition $\nu \in \Omega$ corresponds to the condition that $\mathcal{B}$ does not abort at $\mathcal{A}$'s output stage. When $\mathcal{B}$'s input y_* lies outside of $\mathbb{Z}_n^*$, adversary $\mathcal{B}$ always wins.

Since Ω is independent of $\mathcal{A}$'s view, it suffices to prove the following:

> *Claim.* Let Ω be a randomly generated subset of $\{1, \ldots, t\}$, as above. Let $\mathcal{R} \subseteq \{1, \ldots, t\}$ be a fixed set of at most Q indices, and let $\nu \in \{1, \ldots, t\}$ be a fixed index not in $\mathcal{R}$. Let X be the event that $\mathcal{R} \cap \Omega = \emptyset$ and $\nu \in \Omega$. Then we have
>
> $$\Pr[X] \geq \frac{1}{2.72 \cdot (Q+1)}.$$

The claim is trivially true if $Q = 0$; otherwise, we have:

$$\Pr[X] = \Pr[\mathcal{R} \cap \Omega = \emptyset] \cdot \Pr[\nu \in \Omega] \geq \left(1 - \frac{1}{Q+1}\right)^Q \cdot \frac{1}{Q+1} \geq \frac{1}{2.72 \cdot (Q+1)}.$$

Here, we have made use of the inequality $\big(1 - 1/(n+1)\big)^n \geq 1/e$ which holds for all $n \geq 1$. That proves the claim and concludes the proof of the lemma. □

13.4.2 Proofs of Theorems 13.3 and 13.4

Armed with Lemma 13.5, the proof of Theorem 13.3 is quite straightforward.

Proof of Theorem 13.3. Let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\text{FDH}}$ as in the theorem statement. Using $\mathcal{A}$, we wish to construct an adversary $\mathcal{B}$ that breaks the one-wayness of $\mathcal{T}$ with advantage as in (13.3).

We first make a couple of simplifying assumptions about $\mathcal{A}$. First, when $\mathcal{A}$ outputs its forgery on a particular message, it has previously queried the random oracle on that message. Second, $\mathcal{A}$ never makes the same random oracle query twice, that is, all of its random oracle queries are distinct. Third, $\mathcal{A}$ never makes the same signing query twice, that is, all of its signing queries are distinct. If $\mathcal{A}$ does not already satisfy these properties, we can always convert it to an adversary $\mathcal{A}'$ that does, increasing the number of random oracle queries by at most 1.

So from now on, let us work with the more convenient adversary $\mathcal{A}'$, which makes at most $t := Q_{\text{ro}} + 1$ random oracle queries, and whose advantage in breaking the signature scheme $\mathcal{S}_{\text{FDH}}$ is the same as that of $\mathcal{A}$. From $\mathcal{A}'$, we construct an adversary $\mathcal{B}'$ that wins the t-repeated one-way attack game against $f := F(pk, \cdot)$, where $t := Q_{\text{ro}} + 1$, with the same advantage that $\mathcal{A}'$ wins the signature game. After we have $\mathcal{B}'$, the theorem follows immediately from Lemma 13.5.

Adversary $\mathcal{B}'$ works as follows:

- It obtains $(y_1, \ldots, y_t)$ from its own t-repeated one-way challenger.
- It responds to the ith random oracle query from $\mathcal{A}'$ with y_i.
- If $\mathcal{A}'$ asks to sign a particular message $\hat{m}$:
    - if the random oracle has already been queried at $\hat{m}$, and if this was the jth random oracle query, $\mathcal{B}'$ makes a reveal query at position j to obtain x_j, and forwards x_j to $\mathcal{A}'$;
    - otherwise, if the random oracle has not been queried at $\hat{m}$, then $\mathcal{B}'$ generates a random $\hat{x} \in \mathcal{X}$, computes $\hat{y} \leftarrow F(pk, \hat{x})$, and forwards $\hat{x}$ to $\mathcal{A}'$; moreover, if $\mathcal{A}'$ ever queries the random oracle at $\hat{m}$ in the future, $\mathcal{B}'$ will respond to that query with the value $\hat{y}$.
- Finally, when $\mathcal{A}'$ outputs its candidate forgery (m, σ), then by assumption, the random oracle query was already queried at m; if this was query number ν, then $\mathcal{B}'$ outputs (ν, σ).

Clearly, $\mathcal{B}'$ simulates the signature attack game perfectly for $\mathcal{A}'$, and wins its attack game precisely when $\mathcal{A}'$ wins its game.

Proof of Theorem 13.4. This is almost identical to the proof of Theorem 13.3. The only difference is that we use Lemma 13.6 instead of Lemma 13.5. In the application of Lemma 13.6, the number of reveal queries Q in Attack Game 13.4 is bounded by Q_{s}.

13.5 An RSA-based signature scheme with a tight security proof

Theorem 13.4 shows that $\mathcal{S}_{\text{RSA-FDH}}$ is a secure signature scheme in the random oracle model, but with a relatively loose security reduction. In particular, let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\text{RSA-FDH}}$ that issues at most Q_{s} signing queries and succeeds in breaking $\mathcal{S}_{\text{RSA-FDH}}$ with probability ϵ. Then $\mathcal{A}$ can be used to break the RSA assumption with probability about ϵ/Q_{s}. It is unlikely that $\mathcal{S}_{\text{RSA-FDH}}$ has a tighter security reduction to the RSA assumption.

Surprisingly, a small modification to $\mathcal{S}_{\text{RSA-FDH}}$ gives a signature scheme that has a tight reduction to the RSA assumption in the random oracle model. The only difference is that instead of computing an eth root of $H(m)$, the signing algorithm computes an eth root of $H(b, m)$ for some random bit $b \in \{0, 1\}$. The signature includes the eth root along with the bit b. We call this modified signature scheme $\mathcal{S}'_{\text{RSA-FDH}}$.

We describe $\mathcal{S}'_{\text{RSA-FDH}}$ using the notation of Section 13.3.1. Let $\mathcal{M}' := \{0, 1\} \times \mathcal{M}$. We will need a hash function $H : \mathcal{M}' \to \mathcal{Y}$. Furthermore, we will need a PRF F defined over $(\mathcal{K}, \mathcal{M}, \{0, 1\})$. The $\mathcal{S}'_{\text{RSA-FDH}}$ signature scheme is defined as follows:

- The key generation algorithm G uses fixed RSA parameters ℓ and e, and runs as follows:

$$\begin{aligned} G() := \quad & k \xleftarrow{\text{R}} \mathcal{K}, \quad (n, d) \xleftarrow{\text{R}} \text{RSAGen}(\ell, e) \\ & pk \leftarrow (n, e), \quad sk \leftarrow (k, n, d) \\ & \text{output } (pk, sk). \end{aligned}$$

- For a given secret key $sk = (k, n, d)$ and $m \in \mathcal{M}$, the signing algorithm S runs as follows:

$$\begin{aligned} S(sk, m) := \quad & b \leftarrow F(k, m) \in \{0, 1\} \\ & y \leftarrow H(b, m) \in \mathcal{Y}, \quad \sigma \leftarrow y^d \in \mathbb{Z}_n \\ & \text{output } (b, \sigma). \end{aligned}$$

- For a given public key $pk = (n, e)$ and signature (b, σ), the verification algorithm does:

```
V(pk, m, (b,σ)) :=   $y \leftarrow H(b, m)$
                     if $y = \sigma^e$ output accept; otherwise, output reject.
```

Security. The $\mathcal{S}'_{\text{RSA-FDH}}$ system can be shown to be secure under the RSA assumption, when H is modeled as a random oracle. The security proof uses the random self reduction of RSA to obtain a tight reduction to the RSA problem. The point is that the factor $2.72(Q_\text{s} + 1)$ in Theorem 13.4 is replaced by a factor of 2 in the theorem below.

Theorem 13.7. *Let $H : \mathcal{M}' \to \mathcal{Y}$ be a hash function. Assume that the RSA assumption holds for (ℓ, e), and F is a secure PRF. Then $\mathcal{S}'_{\text{RSA-FDH}}$ is a secure signature scheme when H is modeled as a random oracle.*

> *In particular, let $\mathcal{A}$ be an efficient adversary attacking $\mathcal{S}'_{\text{RSA-FDH}}$. Then there exist an efficient RSA adversary $\mathcal{B}$ and a PRF adversary $\mathcal{B}_F$, where $\mathcal{B}$ and $\mathcal{B}_F$ are elementary wrappers around $\mathcal{A}$, such that*
> $$\text{SIG}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}'_{\text{RSA-FDH}}] \le 2 \cdot \text{RSA}\mathsf{adv}[\mathcal{B}, \ell, e] + \text{PRF}\mathsf{adv}[F, \mathcal{B}_F]$$

Proof idea. Suppose the PRF F is a random function $f : \mathcal{M} \to \{0, 1\}$. We build an algorithm $\mathcal{B}$ that uses an existential forger $\mathcal{A}$ to break the RSA assumption. Let $(n, d) \xleftarrow{\text{R}} \text{RSAGen}(\ell, e), x_* \xleftarrow{\text{R}} \mathbb{Z}_n$, and $y_* \leftarrow x_*^e \in \mathbb{Z}_n$. Algorithm $\mathcal{B}$ is given n, y_* and its goal is to output x_*. First $\mathcal{B}$ sends the public key $pk = (n, e)$ to $\mathcal{A}$. Now $\mathcal{A}$ issues random oracle queries and signing queries. To obtain a tight reduction, $\mathcal{B}$ must properly answer all signing queries from $\mathcal{A}$. In other words, $\mathcal{B}$ must be able to sign every message in $\mathcal{M}$. But this seems impossible — if $\mathcal{B}$ already knows the signature on all messages, how can an existential forgery from $\mathcal{A}$ possibly help $\mathcal{B}$ solve the challenge (n, y_*)? The signature produced by $\mathcal{A}$ seems to give $\mathcal{B}$ no new information.

The solution comes from the extra bit in the signature. Recall that in $\mathcal{S}'_{\text{RSA-FDH}}$ every message $m \in \mathcal{M}$ has two valid signatures, namely $\sigma_0 = (0,\ H(m, 0)^d)$ and $\sigma_1 = (1,\ H(m, 1)^d)$. Algorithm $\mathcal{B}$ sets things up so that it knows exactly one of these signatures for every message. In particular, $\mathcal{B}$ will know the signature $(b, H(b, m))$ where $b \leftarrow f(m)$. The forger $\mathcal{A}$ will output an existential forgery $(m, (b, \sigma))$ where, with probability 1/2, (b, σ) is the signature on m that $\mathcal{B}$ does not know. We will use the random self reduction of RSA to ensure that any such signature enables $\mathcal{B}$ to solve the original challenge. For this to work, $\mathcal{A}$ must not know which of the two signatures $\mathcal{B}$ knows. Otherwise, a malicious $\mathcal{A}$ could always output a signature forgery that is of no use to $\mathcal{B}$. This is the purpose of the PRF.

To implement this idea, $\mathcal{B}$ responds to random oracle queries and signing queries as follows. We let $\mathcal{O}$ denote the random oracle implementing H.

- upon receiving a random oracle query $(b, m) \in \mathcal{M}'$ from $\mathcal{A}$ do:

```
if $b = f(m)$ then $c \leftarrow 0$ else $c \leftarrow 1$
repeat until $y \in \mathcal{Y}$
    $x \xleftarrow{\text{R}} \mathbb{Z}_n,\quad y \leftarrow x^e \cdot y_*^c \in \mathbb{Z}_n$    // So $y = x^e$ or $y = x^e \cdot y_*$
send $y$ to $\mathcal{A}$    // This defines $\mathcal{O}(b, m) := y$
```

Observe that in either case $\mathcal{O}(b, m)$ is a uniform value in $\mathcal{Y}$ as required. In particular, $\mathcal{A}$ learns nothing about the value of $f(m)$.

When $b = f(m)$ the random oracle value $\mathcal{O}(b, m)$ is a random value y for which $\mathcal{B}$ knows an eth root, namely x. When $b \neq f(m)$ then $\mathcal{O}(b, m)$ is a random value y for which $\mathcal{B}$ does not know an eth root. In fact, an eth root of $y = x^e \cdot y_*$ will solve the original challenge — if σ is an eth root of y then $x_* = \sigma/x \in \mathbb{Z}_n$ is an eth root of y_*, since:

$$x_*^e = \sigma^e/x^e = y/x^e = (x^e \cdot y_*)/x^e = y_*. \tag{13.8}$$

In effect, $\mathcal{B}$ uses the random self reduction of RSA to map the original challenge y_* to a random challenge y. It then maps $\mathcal{O}(b, m)$ to this random y.

- Upon receiving a signing query $m \in \mathcal{M}$ from $\mathcal{A}$, respond as follows. First, compute $b \leftarrow f(m)$ and let $y \leftarrow \mathcal{O}(b, m) \in \mathcal{Y}$. By construction, $\mathcal{B}$ defined $\mathcal{O}(b, m) = x^e$ for some random $x \in \mathbb{Z}_n$ chosen by $\mathcal{B}$. Hence, $\mathcal{B}$ has an eth root x for this y. It sends $\mathcal{A}$ the signature (b, x).

So far, $\mathcal{B}$ simulates the challenger perfectly. Its responses to $\mathcal{A}$'s oracle queries are uniform and random in $\mathcal{Y}$ and all its responses to signing queries are valid. Therefore, $\mathcal{A}$ produces an existential forgery (b, σ) on some message m. Then $\sigma^e = \mathcal{O}(b, m)$. Now, if $b \neq f(m)$ then $\mathcal{O}(b, m) = x^e \cdot y_*$ and hence $x_* = \sigma/x$ as in (13.8).

In summary, assuming $b \neq f(m)$, algorithm $\mathcal{B}$ obtains a solution to the challenge y_*. But, by construction of $\mathcal{O}$, the adversary learns no information about the function f. In particular, $f(m)$ is a random bit, and is independent of the adversary's view. Therefore, $b \neq f(m)$ happens with probability 1/2. This is the source of the factor of 2 in Theorem 13.7. □

So what does this mean? The $\mathcal{S}'_{\text{RSA-FDH}}$ system is a minor modification of $\mathcal{S}_{\text{RSA-FDH}}$. Signatures include one additional bit which leads to a tighter reduction to the RSA assumption.

Despite this tighter reduction, $\mathcal{S}'_{\text{RSA-FDH}}$ has not gained much adoption in practice. Most practitioners do not view the extra complexity as a worthwhile tradeoff against the tighter reduction, especially since this reduction is ultimately heuristic, as it models H as a random oracle. It is not clear that $\mathcal{S}'_{\text{RSA-FDH}}$ is any more secure than $\mathcal{S}_{\text{RSA-FDH}}$ for any particular instantiation of H. This is an open question. Conversely, Exercise 13.9 shows that for every instantiation of H, the signature scheme $\mathcal{S}'_{\text{RSA-FDH}}$ is *no less* secure than $\mathcal{S}_{\text{RSA-FDH}}$.

Another difference is that while $\mathcal{S}_{\text{RSA-FDH}}$ is a *unique* signature scheme — every message has a unique valid signature — the signature scheme $\mathcal{S}_{\text{RSA-FDH}}$ is not unique. Most messages have *two* valid signatures.

13.6 Case study: PKCS1 signatures

The most widely deployed standard for RSA signatures is known as PKCS1 version 1.5 mode 1. This RSA signing method is commonly used for signing X.509 certificates. Let n be a t-bit RSA modulus. The standard requires that t is a multiple of 8. Let e be the encryption exponent (or signature verification exponent). To sign a message m, the standard specifies the following steps:

- Hash m to an h-bit hash value using a collision resistant hash function H, where h is also required to be a multiple of 8. The standard requires that $h < t - 88$.

- Let $D \in \{0, 1\}^t$ be the binary string shown in Fig. 13.3. The string starts with the two bytes 00 01. It then contains a padding sequence of FF-bytes that ends with a single 00 byte. Next

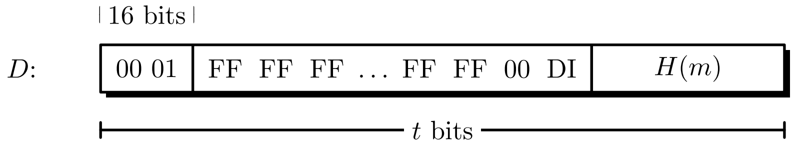


Figure 13.3: PKCS1 signatures: the quantity D signed by RSA

a short DigestInfo (DI) field is appended that encodes the name of the hash function H used to hash m. For example, when SHA256 is used the DigestInfo field is a *fixed* 19-byte string. Finally, $H(m)$ is appended. The length of the padding sequence of FF-bytes is such that D is exactly t bits.

- View D as a t-bit integer, which we further interpret as an element of $\mathbb{Z}_n$, and output the eth root of D as the signature σ.

To verify the signature, first compute $\sigma^e \in \mathbb{Z}_n$, and then interpret this as a t-bit string D. Finally, verify that D contains all the fields shown in Fig. 13.3, and no other fields.

The reason for prepending the fixed PKCS1 pad to the hash value prior to signing is to avoid a chosen message attack due to Desmedt and Odlyzko [51]. The attack is based on the following idea. Suppose PKCS1 directly signed a 256-bit message digest with RSA, without first expanding it to a long string as in Fig. 13.3. Further, suppose the attacker finds three messages m_1, m_2, m_3 such that

$$H(m_1) = p_1, \quad H(m_2) = p_2, \quad H(m_3) = p_1 \cdot p_2, \tag{13.9}$$

where $H(m_1), H(m_2), H(m_3)$ are viewed as integers in the interval $[0, 2^{256})$. The attacker can request the signatures on m_1 and m_2 and from them deduce the signature on m_3 by multiplying the two given signatures. Hence, the attacker obtains an existential forgery by issuing two chosen message queries. The attack of Desmedt and Odlyzko extends this basic idea so that the attack succeeds with high probability using many chosen message queries. The reason for the padding in Fig. 13.3 is so that the numbers for which an eth root is computed are much longer than 256 bits. As a result, it is much less likely that an attacker can find messages satisfying a condition such as (13.9).

Security. PKCS1 is an example of a **partial domain hash** signature. The message m is hashed into an h-bit string that is mapped into a fixed interval I inside of $\mathbb{Z}_n$. The interval has size $|I| = 2^h$. Typically, the hash size h is 160 or 256 bits, and the modulus size t is at least 2048 bits. Hence, I is a tiny subset of $\mathbb{Z}_n$.

Unfortunately, the proof of Theorem 13.4 requires that the output of the hash function H be uniformly distributed over a large subset $\mathcal{Y}$ of $\mathbb{Z}_n$. This was necessary for the proof of Lemma 13.6. The set $\mathcal{Y}$ had to be large so that we could pick a random $y \in \mathcal{Y}$ for which we knew an eth root.

When hashing into a tiny subset I of $\mathbb{Z}_n$ the proof of Lemma 13.6 breaks down. The problem is that we cannot pick a random $y \in I$ so that an eth root of y is known. More precisely, the obstruction to the proof is the following problem:

> $(*)$ given an RSA modulus n, output a pair (y, x) where y is uniformly distributed in a subset $I \subseteq \mathbb{Z}_n$ and x is an eth root of y.

A solution to this problem will enable us to prove security of PKCS1 under the assumption that computing eth roots is hard in the interval I. Problem $(*)$ is currently open. The best known algorithm [46] solves the problem for $e = 2$ whenever $|I| \geq n^{2/3}$. However, typically in PKCS1, $|I|$ is far smaller than $n^{2/3}$ (and for RSA we use $e > 2$).

In summary, although PKCS1 v1.5 is a widely used standard for signing using RSA, we cannot prove it secure under the standard RSA assumption. An updated version of PKCS1 known as PKCS1 v2.1 includes an additional RSA-based signature method called PSS, discussed in the chapter notes.

13.6.1 Bleichenbacher's attack on PKCS1 signatures

Implementing cryptography is not easy. In this section, we give a clever attack on a once-popular implementation of PKCS1 that illustrates its fragility. Let $pk = (n, 3)$ be an RSA public key for the PKCS1 signature scheme: n is a t-bit RSA modulus and the signature verification exponent is 3. We assume $t \geq 2048$.

When signing a message m using PKCS1 the signer forms the block D shown in Fig. 13.3, and then, treating D as an integer, computes the cube root of D modulo n as the signature σ.

Consider the following erroneous implementation of the verification algorithm. To verify a message-signature pair (m, σ), with SHA256 as the hash function, the verifier does:

1. compute $\sigma^e \in \mathbb{Z}_n$, and then interpret this as a t-bit string D
2. parse D from left to right as follows:
   (a) reject if the top most 2 bytes are not 00 01
   (b) skip over all FF-bytes until reaching a 00 byte and skip over it too
   (c) reject if the next bytes are not the DigestInfo field for the SHA256 function
   (d) read the following 32 bytes (256 bits), compare them to the hash value SHA256(m), and reject if not equal
3. if all the checks above pass successfully, accept the signature

While this procedure appears to correctly verify the signature, it ignores one very crucial step: it does not check that D contains nothing to the right of the hash value. In particular, this verification procedure accepts a t-bit block D^* that looks as follows:

$$D^* := \begin{array}{|c|c|c|c|}\hline \text{00 01} & \text{FF}\ldots\text{FF 00 DI} & \text{hash} & \text{more bits } J \\ \hline \end{array}$$

Here J is some sequence of bits chosen by the attacker. The attacker shortened the variable length padding block of FF's to make room for the quantity J, so that the total length of D^* is still t bits.

This minor-looking oversight leads to a complete break of the signature scheme. An attacker can generate a valid signature on any message m of its choice, as we now proceed to demonstrate.

Let $w \in \mathbb{Z}$ be the largest multiple of eight smaller than $t/3 - 3$. To forge the signature on m, the attacker first computes $H(m) = \text{SHA256}(m)$ and constructs the block D, as in Fig. 13.3, but where D is only w bits long (note that $w \approx t/3$). To make D this short, simply make the variable length padding block sufficiently short. Next, viewing D as an integer, the attacker computes:

$$s \leftarrow \sqrt[3]{D \cdot 2^{t-w}} \in \mathbb{R}, \qquad x \leftarrow \lceil\ s\ \rceil \ \in \mathbb{Z}, \qquad \text{output } x.$$

Here, the cube root s of $D \cdot 2^{t-w}$ is computed over the real numbers and rounded up to the next integer x.

We show that x, when viewed as an element of $\mathbb{Z}_n$, will be accepted as a valid signature on m. Since $0 \leq x - s < 1$, we obtain

$$0 \leq x^3 - (D \cdot 2^{t-w}) = x^3 - s^3 = (x-s)(x^2 + xs + s^2) < 3(s+1)^2.$$

Observe that $s^3 = D \cdot 2^{t-w} < 2^t$, because the leading bits of D are zero. Moreover, for $s \geq 3$, we have that $(s+1)^2 \leq 2s^2 < 2 \cdot 2^{(2/3)t}$, and therefore

$$0 \leq x^3 - (D \cdot 2^{t-w}) < 3(s+1)^2 < 6 \cdot 2^{(2/3)t} < 2^{t-[(t/3)-3]} < 2^{t-w}.$$

In other words, $x^3 = (D \cdot 2^{t-w}) + J$ where $0 \leq J < 2^{t-w}$.

It follows that if we treat x as an element of $\mathbb{Z}_n$, it will be accepted as a signature on m. Indeed, x^3 will be strictly less than n, so the computation of x^3 mod n will not wrap around at all. Moreover, when the verifier interprets x^3 as a t-bit string D^*, the w most significant bits of D^* are equal to D, ensuring that x will be accepted as a signature on m with respect to the public key $(n, 3)$.

This attack applies to RSA public keys that use a small public exponent, such as $e = 3$. When it was originally discovered, it was shown to work well against several popular PKCS1 implementations. The attack exploits a bug in the implementation of PKCS1 that is easily mitigated: the verifier must reject the signature if D is not the correct length, or there are bits in D to the right of the hash value. Nevertheless, it is a good illustration of the difficulty of correctly implementing cryptographic primitives. A simple misunderstanding in reading the PKCS1 specification resulted in a devastating attack on its implementation.

13.7 Signcryption: combining signatures and encryption

A signcryption scheme lets a sender, Alice, send an encrypted message to a recipient, Bob, so that (1) only Bob can read the message, and (2) Bob is convinced that the message came from Alice. Signcryption schemes are needed in messaging systems that provide end-to-end security, but where Bob may be offline at the time that Alice sends the message. Because Bob is offline, Alice cannot interact with Bob to establish a shared session key. Instead, she encrypts the message intended for Bob, and Bob receives and decrypts it at a later time. The ciphertext she sends to Bob must convince Bob that the message is from Alice.

Since anyone can generate public-private key pairs, signcryption only makes sense in an environment where every identity is publicly bound to one or more public keys. More precisely, Bob can tell what public keys are bound to Alice's identity, and an attacker cannot cause Bob to associate an incorrect public key to Alice. If this were not the case, that is, if an attacker can generate a public-private key pair and convince Bob that this public key belongs to Alice, then the goals of signcryption cannot be achieved: the attacker could send a message on behalf of Alice, and Bob could not tell the difference; similarly, the attacker could decrypt messages that Bob thinks he is sending to Alice.

To capture this requirement on public keys and identities, we assign to every user X of the system a unique identity id_{X}. Moreover, we assume that any other user can fetch the public key pk_{X} that is bound to the identity id_{X}. So, Alice can obtain a public key bound to Bob, and she can be reasonably confident that only Bob knows the corresponding private key. Abstractly, one can think of a public directory that maintains a mapping from identities to public keys. Anyone can read the directory, but only the user with identity id_{X} can update the record associated with id_{X} (in today's technology, Facebook user profiles serve as such a global directory). In Section 13.8 we will see that certificates are another way to reliably bind public keys to identities.

We will denote the sender's identity by id_{S} and the recipient's identity by id_{R}. We denote the sender's public-private key pair by pk_{S} and sk_{S} and the recipients key pair by pk_{R} and sk_{R}. To encrypt a message m intended for a specific recipient, the sender needs its own identity id_{S} and secret key sk_{S} as well as the recipients identity id_{R} and public key pk_{R}. To decrypt an incoming ciphertext, the recipient needs the sender's identity id_{S} and public key pk_{S} as well as its own identity id_{R} and secret key sk_{R}. With this in place we can define the syntax for signcryption.

Definition 13.5. *A* ***signcryption*** *scheme $\mathcal{SC} = (G, E, D)$ is a triple of efficient algorithms, G, E and D, where G is called a* ***key generation algorithm****, E is called an* ***encryption algorithm****, and D is called a* ***decryption algorithm****.*

- *G is a probabilistic algorithm that takes no input. It outputs a pair (pk, sk), where sk is called a* ***secret key*** *and pk is called a* ***public key****.*

- *E is a probabilistic algorithm that is invoked as $c \xleftarrow{\mathrm{R}} E\big(sk_{\mathrm{S}}, id_{\mathrm{S}}, pk_{\mathrm{R}}, id_{\mathrm{R}},\ m\big)$, where sk_{S} and id_{S} are the secret key and identity of the sender, pk_{R} and id_{R} are the public key and identity of the recipient, and m is a message. The algorithm outputs a* ***ciphertext*** *c.*

- *D is a deterministic algorithm invoked as $D\big(pk_{\mathrm{S}}, id_{\mathrm{S}}, sk_{\mathrm{R}}, id_{\mathrm{R}},\ c\big)$. It outputs either a message m or a special symbol* reject*.*

- *We require that a ciphertext generated by E is always accepted by D. That is, for all possible outputs $(pk_{\mathrm{S}}, sk_{\mathrm{S}})$ and $(pk_{\mathrm{R}}, sk_{\mathrm{R}})$ of G, all identities $id_{\mathrm{S}}, id_{\mathrm{R}}$, and all messages m*

$$\Pr\Big[D\big(pk_{\mathrm{S}}, id_{\mathrm{S}}, sk_{\mathrm{R}}, id_{\mathrm{R}},\ E(sk_{\mathrm{S}}, id_{\mathrm{S}}, pk_{\mathrm{R}}, id_{\mathrm{R}},\ m)\big) = m\Big] = 1.$$

As usual, we say that messages lie in a finite ***message space*** *$\mathcal{M}$, ciphertexts lie in some finite* ***ciphertext space*** *$\mathcal{C}$, and identities lie in some finite* ***identity space*** *$\mathcal{I}$. We say that $\mathcal{SC} = (G, E, D)$ is defined over $(\mathcal{M}, \mathcal{C}, \mathcal{I})$.*

We can think of signcryption as the public-key analogue of authenticated encryption for symmetric ciphers. Authenticated encryption is designed to achieve the same confidentiality and authenticity goals as signcryption, but assuming the sender and recipient have already established a shared secret key. Signcryption is intended for a non-interactive setting where no shared secret key is available. With this analogy in mind we can consider two signcryption constructions, similar to the ones in Chapter 9:

- The signcryption analogue of encrypt-then-MAC is encrypt-then-sign: first encrypt the message with the recipient's public encryption key and then sign the resulting ciphertext with the sender's secret signing key.

- The signcryption analogue of MAC-then-encrypt is sign-then-encrypt: first sign the message with the sender's secret signing key and then encrypt the message-signature pair with the recipient's public encryption key.

Which of these is secure? Is one method better than the other? To answer these questions we must first formally define what it means for a signcryption scheme to be secure, and then analyze these and other signcryption schemes.

We begin in Section 13.7.1 with a formal definition of security for signcryption. Admittedly, our definition of secure signcryption is a bit lengthy, and it may not be immediately clear that it captures the "right" properties. In Section 13.7.2, we discuss how this definition can be used to derive more intuitive security properties of signcryption in a multi-user setting. It is precisely these implications that give us confidence that the basic definition in Section 13.7.1 is sufficiently strong. In Sections 13.7.3 and 13.7.4 we turn to the problem of constructing secure signcryption schemes. Finally, in Section 13.7.5, we investigate some additional desirable security properties for signcryption, called forward-secrecy and non-repudiation, and show how to achieve them.

13.7.1 Secure signcryption

We begin with the basic security requirements for a signcryption scheme. As we did for authenticated encryption, we define secure signcryption using two games. One game captures data confidentiality: an adversary who does not have Alice's or Bob's secret key cannot break semantic security for a set of challenge ciphertexts from Alice to Bob. The other game captures data authenticity: an adversary who does not have Alice's or Bob's secret key cannot make Bob accept a ciphertext that was not generated by Alice with the intent of sending it to Bob.

In both games the adversary is active. In addition to asking Alice to encrypt messages intended for Bob, and asking Bob to decrypt messages supposedly coming from Alice, the adversary is free to ask Alice to encrypt messages intended for any other user of the adversary's choosing, and to ask Bob to decrypt messages supposedly coming from any other user of the adversary's choosing. Moreover, the attack game reflects the fact that while Alice may be sending messages to Bob, she may also be receiving messages from other users. Therefore, the adversary is free to ask Alice to decrypt messages supposedly coming from any other user of the adversary's choosing. Similarly, modeling the fact that Bob may also be playing the role of sender, the adversary is free to ask Bob to encrypt messages intended for any other user of the adversary's choosing.

Ciphertext integrity. We start with the data authenticity game, which is an adaptation of the ciphertext integrity game used in the definition of authenticated encryption (Attack Game 9.1).

Attack Game 13.5 (ciphertext integrity). For a given signcryption scheme $\mathcal{SC} = (G, E, D)$ defined over $(\mathcal{M}, \mathcal{C}, \mathcal{I})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The adversary chooses two distinct identities $id_{\rm S}$ (the sender identity) and $id_{\rm R}$ (the receiver identity), and gives these to the challenger. The challenger runs G twice to obtain $(pk_{\rm S}, sk_{\rm S})$ and $(pk_{\rm R}, sk_{\rm R})$ and gives $pk_{\rm S}$ and $pk_{\rm R}$ to $\mathcal{A}$.

- $\mathcal{A}$ issues a sequence of queries to the challenger. Each query is one of the following types:

  S → R *encryption query:* a message m.
  The challenger computes $c \xleftarrow{\rm R} E(sk_{\rm S}, id_{\rm S}, pk_{\rm R}, id_{\rm R}, m)$, and gives c to $\mathcal{A}$.

  X → Y *encryption query:* a tuple $(id_{\rm X}, id_{\rm Y}, pk_{\rm Y}, m)$, where $id_{\rm X} \in \{id_{\rm S}, id_{\rm R}\}$ and $(id_{\rm X}, id_{\rm Y}) \neq (id_{\rm S}, id_{\rm R})$. The challenger responds to $\mathcal{A}$ with c, computed as follows:

if $id_{\rm X} = id_{\rm S}$ then $c \xleftarrow{R} E(sk_{\rm S}, id_{\rm S}, pk_{\rm Y}, id_{\rm Y}, m)$,
if $id_{\rm X} = id_{\rm R}$ then $c \xleftarrow{R} E(sk_{\rm R}, id_{\rm R}, pk_{\rm Y}, id_{\rm Y}, m)$.

X → Y *decryption query:* a tuple $(id_{\rm X}, id_{\rm Y}, pk_{\rm X}, \hat{c})$, where $id_{\rm Y} \in \{id_{\rm S}, id_{\rm R}\}$ and $(id_{\rm X}, id_{\rm Y}) \neq (id_{\rm S}, id_{\rm R})$. The challenger responds to $\mathcal{A}$ with $\hat{m}$, computed as follows:

if $id_{\rm Y} = id_{\rm S}$ then $\hat{m} \leftarrow D(pk_{\rm X}, id_{\rm X}, sk_{\rm S}, id_{\rm S}, \hat{c})$,
if $id_{\rm Y} = id_{\rm R}$ then $\hat{m} \leftarrow D(pk_{\rm X}, id_{\rm X}, sk_{\rm R}, id_{\rm R}, \hat{c})$.

- Finally, $\mathcal{A}$ outputs a candidate ciphertext forgery $c' \in \mathcal{C}$, where c' is not among the responses to an S → R encryption query.

We say that $\mathcal{A}$ wins the game if its candidate ciphertext forgery c' is a valid ciphertext from $id_{\rm S}$ to $id_{\rm R}$, that is, $D(pk_{\rm S}, id_{\rm S}, sk_{\rm R}, id_{\rm R}, c') \neq \mathsf{reject}$. We define $\mathcal{A}$'s advantage, denoted SCI$\mathsf{adv}[\mathcal{A}, \mathcal{SC}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 13.6. *We say that $\mathcal{SC} = (G, E, D)$ provides* ***signcryption ciphertext integrity****, or* SCI *for short, if for every efficient adversary $\mathcal{A}$, the value* SCI$\mathsf{adv}[\mathcal{A}, \mathcal{SC}]$ *is negligible.*

Security against a chosen ciphertext attack. Next, we define the data confidentiality game, which is an adaptation of the game used to define chosen ciphertext security (Attack Game 12.1). Note that in this game, the syntax of the X → Y encryption and decryption queries are exactly the same as in Attack Game 13.5.

Attack Game 13.6 (CCA security). For a given signcryption scheme $\mathcal{SC} = (G, E, D)$, defined over $(\mathcal{M}, \mathcal{C}, \mathcal{I})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b ($b = 0, 1$):

- The adversary chooses two distinct identities $id_{\rm S}$ (the sender identity) and $id_{\rm R}$ (the receiver identity), and gives these to the challenger. The challenger runs G twice to obtain $(pk_{\rm S}, sk_{\rm S})$ and $(pk_{\rm R}, sk_{\rm R})$ and gives $pk_{\rm S}$ and $pk_{\rm R}$ to $\mathcal{A}$.

- $\mathcal{A}$ issues a sequence of queries to the challenger. Each query is one of the following types:

S → R *encryption query:* a pair of equal-length messages (m_0, m_1).
The challenger computes $c \xleftarrow{R} E(sk_{\rm S}, id_{\rm S}, pk_{\rm R}, id_{\rm R}, m_b)$, and gives c to $\mathcal{A}$.

S → R *decryption query:* a ciphertext $\hat{c}$, where $\hat{c}$ is not among the outputs of any previous S → R encryption query.
The challenger computes $\hat{m} \xleftarrow{R} D(pk_{\rm S}, id_{\rm S}, sk_{\rm R}, id_{\rm R}, \hat{c})$, and gives $\hat{m}$ to $\mathcal{A}$.

The remaining two query types are the same as in the ciphertext integrity game:

X → Y *encryption query:* a tuple $(id_{\rm X}, id_{\rm Y}, pk_{\rm Y}, m)$, where $id_{\rm X} \in \{id_{\rm S}, id_{\rm R}\}$ and $(id_{\rm X}, id_{\rm Y}) \neq (id_{\rm S}, id_{\rm R})$. The challenger responds to $\mathcal{A}$ with c, computed as follows:

if $id_{\rm X} = id_{\rm S}$ then $c \xleftarrow{R} E(sk_{\rm S}, id_{\rm S}, pk_{\rm Y}, id_{\rm Y}, m)$,
if $id_{\rm X} = id_{\rm R}$ then $c \xleftarrow{R} E(sk_{\rm R}, id_{\rm R}, pk_{\rm Y}, id_{\rm Y}, m)$.

X → Y *decryption query:* a tuple $(id_{\rm X}, id_{\rm Y}, pk_{\rm X}, \hat{c})$, where $id_{\rm Y} \in \{id_{\rm S}, id_{\rm R}\}$ and $(id_{\rm X}, id_{\rm Y}) \neq (id_{\rm S}, id_{\rm R})$. The challenger responds to $\mathcal{A}$ with $\hat{m}$, computed as follows:

if $id_{\mathrm{Y}} = id_{\mathrm{S}}$ then $\hat{m} \leftarrow D(pk_{\mathrm{X}}, id_{\mathrm{X}}, sk_{\mathrm{S}}, id_{\mathrm{S}}, \hat{c})$,
if $id_{\mathrm{Y}} = id_{\mathrm{R}}$ then $\hat{m} \leftarrow D(pk_{\mathrm{X}}, id_{\mathrm{X}}, sk_{\mathrm{R}}, id_{\mathrm{R}}, \hat{c})$.

- At the end of the game, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b and define $\mathcal{A}$'s advantage as

$$\mathrm{SCCA}\mathsf{adv}[\mathcal{A}, \mathcal{SC}] := \big|\Pr[W_0] - \Pr[W_1]\big|. \quad \Box$$

Definition 13.7 (CCA Security). *A signcryption scheme $\mathcal{SC}$ is called* ***semantically secure against a chosen ciphertext attack****, or simply* ***CCA secure****, if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{SCCA}\mathsf{adv}[\mathcal{A}, \mathcal{SC}]$ *is negligible.*

Finally, we define a secure signcryption scheme as one that is both CCA secure and has ciphertext integrity.

Definition 13.8. *We say that a signcryption scheme $\mathcal{SC} = (G, E, D)$ is secure if $\mathcal{SC}$ is (1) CCA secure, and (2) provides signcryption ciphertext integrity.*

From two users to multiple users. While this security definition focuses on just two honest users, it actually implies a strong security property in a multi-user setting. We will flesh this out below in Section 13.7.2.

Replay attacks. One thing the definition does not prevent is a "replay" attack: an attacker can record a valid ciphertext c from Alice to Bob and at a later time, say a week later, resend the same c to Bob. Bob receives the replayed ciphertext c and, because it is a valid ciphertext, he might mistakenly believe that Alice sent him the same message again. For example, if the message from Alice is "please transfer $10 to Charlie," then Bob might incorrectly transfer another $10 to Charlie.

Signcryption is not designed to prevent replay attacks. Higher level protocols that use signcryption must themselves take measures to counter-act them. We will discuss replay attacks and how to prevent them when we discuss authenticated key exchange in Chapter 21.

Statically vs adaptively chosen user IDs. Our definition of secure signcryption is subject to a rather subtle criticism, related to the manner in which user IDs are chosen. While we leave it to the adversary to choose the user IDs of the sender and receiver (that is, id_{S} and id_{R}), this choice is "static" in the sense that it is made at the very beginning of the game. A more robust definition would allow a more "adaptive" strategy, in which the adversary gets to choose these IDs after seeing one or both of the public keys, or even after seeing the response to one or more X → Y queries. For most realistic schemes (including all of those discussed here), this distinction makes no difference, but it is possible to dream up contrived schemes where it does (see Exercise 13.18). We have presented the definition with statically chosen IDs mainly for the sake of simplicity (and because, arguably, honest users choose their IDs in a manner that is not so much under an adversary's control).

13.7.2 Signcryption as an abstract interface

Our definition of secure signcryption may seem a bit technical, and it is perhaps useful to discuss how this definition can be applied. As we did in Sections 9.3 and 12.2.4, we do so by describing signcryption as an abstract interface. However, unlike in those two sections, it makes more sense here to explicitly model a system consisting of many users who are trying to send messages to one another over an insecure network.

The setting is as follows. We have a system of many users: some are "honest" and some are "corrupt." The honest users are assumed to follow the specified communication protocol correctly, while the corrupt users may do anything they like to try and subvert the protocol. The corrupt users may collude with each other, and may also attempt to subvert communications by eavesdropping on and tampering with network communication. In fact, we can just assume there is a single attacker who orchestrates the behavior of all the corrupt users and completely controls the network. Moreover, this attacker may have some knowledge of or influence over messages sent by honest users, and may have some knowledge of messages received by honest users.

To start with, we assume that each honest user somehow registers with the system by providing a user ID and a public key. We do not worry about the details of this registration process, except that we require each honest user to have a unique ID and to generate its public key using the key generation algorithm of the signcryption scheme (and, of course, keep the corresponding secret key to itself).

We require that the corrupt users also register with the system. While we insist that all users (honest and corrupt) have unique IDs, we do not make any requirements on how the corrupt users generate their public keys: they may use the prescribed key generation algorithm, or they may do something else entirely, including computing their public key as some function of one or more honest users' public keys. In fact, we may even allow the corrupt users to register with the system after it has been running for a while, choosing their public keys (and even their user IDs) in some way that depends in some malicious way on everything that has happened so far (including all network traffic).

We model the communication interface as a collection of in-boxes and out-boxes.

For each *honest user* id_{S} and each *registered user* (honest or corrupt) $id_{\text{R}} \neq id_{\text{S}}$, we have an *out-box* denoted $Out(id_{\text{S}}, id_{\text{R}})$. If id_{R} belongs to an honest user, we say that the out-box is *safe*; otherwise, we say that it is *unsafe*. From time to time, user id_{S} may want to send a message to user id_{R}, and he does so by dropping the message in the out-box $Out(id_{\text{S}}, id_{\text{R}})$.

For each *registered user* (honest or corrupt) id_{S} and each *honest user* $id_{\text{R}} \neq id_{\text{S}}$, we have an *in-box* denoted $In(id_{\text{S}}, id_{\text{R}})$. If id_{S} belongs to an honest user, we say that the in-box is *safe*; otherwise, we say that it is *unsafe*. From time to time, a message may appear in the in-box $In(id_{\text{S}}, id_{\text{R}})$, which user id_{R} may then retrieve.

That is the abstract interface. We now describe the real implementation.

First, consider an out-box $Out(id_{\text{S}}, id_{\text{R}})$ associated with an honest user id_{S}. The user id_{R} may or may not be honest. When user id_{S} user drops a message in the out-box, the message is encrypted using the secret key associated with user id_{S} and the public key associated with user id_{R} (along with the given user IDs). The resulting ciphertext is sent out to the network.

In a properly functioning network, if user id_{R} is an honest user, this ciphertext will eventually be presented to the matching in-box $In(id_{\text{S}}, id_{\text{R}})$.

Now consider an in-box $In(id_{\text{S}}, id_{\text{R}})$ associated with an honest user id_{R}. The user id_{S} may or may not be honest. Whenever the network presents a ciphertext to this in-box, it is decrypted using the public key of id_{S} and the secret key id_{R} (along with the given user IDs). If the ciphertext is not rejected, the resulting message is placed in the in-box for later consumption by user id_{R}.

We now describe an *ideal implementation* of this interface.

Here is what happens when an honest user drops a message in an out-box $Out(id_{\mathrm{S}}, id_{\mathrm{R}})$. If the out-box is safe (i.e., user id_{R} is an honest user), instead of encrypting the given message, a dummy message is encrypted. This dummy message has nothing to do with the real message (except that it should be of the same length), and the resulting ciphertext just serves as a "handle". Otherwise, if the out-box is unsafe, the real message is encrypted as in the real implementation.

Here is what happens when the network presents a ciphertext to an in-box $In(id_{\mathrm{S}}, id_{\mathrm{R}})$. If the in-box is safe (i.e., user id_{S} is an honest user), the ideal implementation checks if this ciphertext was previously generated as a handle by the matching out-box $Out(id_{\mathrm{S}}, id_{\mathrm{R}})$, and if so, copies the corresponding message directly from the out-box to the in-box; otherwise, the ciphertext is discarded. If the in-box is unsafe, the ciphertext is decrypted as in the real implementation.

We hope that it is intuitively clear that this ideal implementation provides all the security one could possibly hope for. In this ideal implementation, messages magically "jump" from honest senders to honest receivers: the attacker cannot tamper with or glean any information about these messages, even if honest users interact with corrupt users. At worst, an attacker reorders or duplicates messages by reordering or duplicating the corresponding handles (indeed, as already mentioned, our definition of secure signcryption does not rule out "replay" attacks). Typically, this is an issue that a higher level protocol can easily deal with.

We now argue informally that if the signcryption scheme is secure, as in Definition 13.8, then the real world implementation is indistinguishable from the ideal implementation. The argument proceeds in three steps. We start with the real implementation, and in each step, we make a slight modification.

- First, we modify the behavior of the safe in-boxes. Whenever the network presents a ciphertext to the in-box that came from the matching out-box, the corresponding message is copied directly from the out-box to the in-box.

  The correctness property of the signcryption scheme ensures that this modification behaves exactly the same as the real implementation.

- Second, we modify the behavior of the safe in-boxes again. Whenever the network presents a ciphertext to the in-box that did not come from the matching out-box, the ciphertext is discarded.

  The ciphertext integrity property ensures that this modification is indistinguishable from the first. To reduce from the multi-user setting to the two-user setting, one must employ a "guessing argument".

- Third, we modify the behavior of the safe out-boxes, so that dummy messages are encrypted in place of the real messages.

  The CCA security property ensures that this modification is indistinguishable from the second. To reduce from the multi-user setting to the two-user setting, one must employ a "hybrid argument".

Just as in Sections 9.3 and 12.2.4, we have ignored the possibility that the ciphertexts generated in a safe out-box are not unique. If we are going to view these ciphertexts as handles in the ideal implementation, uniqueness is an essential property. However, just as in those cases, the CCA security property implies that these ciphertexts are unique with overwhelming probability.

13.7.3 Constructions: encrypt-then-sign and sign-then-encrypt

We begin by analyzing the two most natural constructions. Both are a combination of a CCA-secure public-key encryption scheme and a secure signature scheme. Getting these combinations right is a little tricky and small variations can be insecure. We explore some insecure variations in Exercises 13.16 and 13.17.

Let $\mathcal{E} = (G_{\text{ENC}}, E, D)$ be a public-key encryption scheme with associated data (see Section 12.7). Recall that this means that E is invoked as $c \xleftarrow{\text{R}} E(pk, m, d)$, and D is invoked as $m \xleftarrow{\text{R}} D(sk, c, d)$, where d is the "associated data". Also, let $\mathcal{S} = (G_{\text{SIG}}, S, V)$ be a signature scheme. Define algorithm G as:

$$\begin{aligned} G() \quad := \quad & (pk_{\text{ENC}},\ sk_{\text{ENC}}) \xleftarrow{\text{R}} G_{\text{ENC}}(), \quad (pk_{\text{SIG}},\ sk_{\text{SIG}}) \xleftarrow{\text{R}} G_{\text{SIG}}() \\ & \text{output } pk := (pk_{\text{ENC}},\ pk_{\text{SIG}}) \text{ and } sk := (sk_{\text{ENC}},\ sk_{\text{SIG}}) \end{aligned}$$

In what follows we use the shorthand $E(pk, m, d)$ to mean $E(pk_{\text{ENC}}, m, d)$ and $S(sk, m)$ to mean $S(sk_{\text{SIG}}, m)$, for some message m. We use a similar shorthand for $V(pk, m, \sigma)$ and $D(sk, c, d)$. We next define two natural signcryption schemes, each of which has a message space $\mathcal{M}$ and an identity space $\mathcal{I}$.

Encrypt-then-sign. The scheme $\mathcal{SC}_{\text{EtS}} = (G,\ E_{\text{EtS}},\ D_{\text{EtS}})$ is defined as

$$\begin{aligned} E_{\text{EtS}}(sk_{\text{S}}, id_{\text{S}}, pk_{\text{R}}, id_{\text{R}},\ m) \quad &:= \quad c \xleftarrow{\text{R}} E\big(pk_{\text{R}},\ m,\ id_{\text{S}}\big), \quad \sigma \xleftarrow{\text{R}} S\big(sk_{\text{S}},\ (c, id_{\text{R}})\big) \\ & \qquad \text{output } (c, \sigma); \\ D_{\text{EtS}}\big(pk_{\text{S}}, id_{\text{S}}, sk_{\text{R}}, id_{\text{R}},\ (c, \sigma)\big) \quad &:= \quad \text{if } V(pk_{\text{S}},\ (c, id_{\text{R}}),\ \sigma) = \mathsf{reject}, \text{ output } \mathsf{reject} \\ & \qquad \text{otherwise, output } D(sk_{\text{R}},\ c,\ id_{\text{S}}). \end{aligned}$$

Here the encryption scheme $\mathcal{E}$ is assumed to be defined over $(\mathcal{M},\ \mathcal{I},\ \mathcal{C})$, so that $\mathcal{I}$ is the associated data space for $\mathcal{E}$. The signature scheme $\mathcal{S}$ is assumed to be defined over $(\mathcal{C} \times \mathcal{I},\ \Sigma)$.

Sign-then-encrypt. The scheme $\mathcal{SC}_{\text{StE}} = (G,\ E_{\text{StE}},\ D_{\text{StE}})$ is defined as

$$\begin{aligned} E_{\text{StE}}(sk_{\text{S}}, id_{\text{S}}, pk_{\text{R}}, id_{\text{R}},\ m) \quad &:= \quad \sigma \xleftarrow{\text{R}} S\big(sk_{\text{S}},\ (m, id_{\text{R}})\big), \quad c \xleftarrow{\text{R}} E\big(pk_{\text{R}},\ (m, \sigma),\ id_{\text{S}}\big) \\ & \qquad \text{output } c; \\ D_{\text{EtS}}\big(pk_{\text{S}}, id_{\text{S}}, sk_{\text{R}}, id_{\text{R}},\ c\big) \quad &:= \quad \text{if } D(sk_{\text{R}}, c, id_{\text{S}}) = \mathsf{reject}, \text{ output } \mathsf{reject}, \text{ otherwise:} \\ & \qquad\quad (m, \sigma) \leftarrow D(sk_{\text{R}},\ c,\ id_{\text{S}}) \\ & \qquad\quad \text{if } V(pk_{\text{S}},\ (m, id_{\text{R}}),\ \sigma) = \mathsf{reject}, \text{ output } \mathsf{reject} \\ & \qquad\quad \text{otherwise, output } m. \end{aligned}$$

Here the encryption scheme $\mathcal{E}$ is assumed to be defined over $(\mathcal{M} \times \Sigma,\ \mathcal{I},\ \mathcal{C})$, where $\mathcal{I}$ is the associated data space. The signature scheme $\mathcal{S}$ is assumed to be defined over $(\mathcal{M} \times \mathcal{I},\ \Sigma)$. Moreover, we shall assume that the signatures are bit strings whose length only depends on the message being signed (this technical requirement will be required in the security analysis).

The following two theorems show that both schemes are secure signcryption schemes. Notice that the corresponding symmetric constructions analyzed in Section 9.4 were not both secure.

Encrypt-then-MAC provides authenticated encryption while MAC-then-encrypt might not. In the signcryption setting, both constructions are secure. The reason sign-then-encrypt is secure is that we are starting from a CCA-secure public-key system $\mathcal{E}$, where as MAC-then-encrypt was built from a CPA-secure cipher. In fact, we know by Exercise 9.15 that MAC-then-encrypt, where the encryption scheme is CCA secure, provides authenticated encryption. Therefore, it should not be too surprising that sign-then-encrypt is secure.

Unlike the encrypt-then-MAC construction, the encrypt-then-sign method requires a CCA-secure encryption scheme for security, rather than just a CPA-secure encryption scheme. We already touched on this issue back in Section 12.2.2 as one of the motivations for studying CCA-secure public-key encryption.

The encrypt-then-sign method requires a *strongly secure* signature scheme for security, as defined in Definition 13.3. Without this, the scheme can be vulnerable to a CCA attack: if an adversary, given a challenge ciphertext (c, σ), can produce a new valid signature σ' on the same data, then the adversary can win the CCA attack game by asking for a decryption of (c, σ'). To prevent this, we require that the signature scheme is strongly secure. This is perhaps to be expected, as in the symmetric setting, the encrypt-then-MAC construction requires a secure MAC, and our definition of a secure MAC is the direct analogue of our definition of a strongly secure signature scheme. In contrast, sign-then-encrypt requires just a secure signature scheme — the scheme need not be strongly secure.

We now present the security theorems for both schemes.

Theorem 13.8. *$\mathcal{SC}_{\text{EtS}}$ is a secure signcryption scheme assuming $\mathcal{E}$ is a CCA-secure public-key encryption scheme with associated data and $\mathcal{S}$ is a strongly secure signature scheme.*

> *In particular, for every ciphertext integrity adversary $\mathcal{A}_{\text{ci}}$ that attacks $\mathcal{SC}_{\text{EtS}}$ as in Attack Game 13.5 there exists a strong signature adversary $\mathcal{B}_{\text{sig}}$ that attacks $\mathcal{S}$ as in Attack Game 13.2, where $\mathcal{B}_{\text{sig}}$ is an elementary wrapper around $\mathcal{A}_{\text{ci}}$, such that*
>
> $$\text{SCI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{SC}_{\text{EtS}}] = \text{stSIG}\mathsf{adv}[\mathcal{B}_{\text{sig}}, \mathcal{S}].$$
>
> *In addition, for every CCA adversary $\mathcal{A}_{\text{cca}}$ that attacks $\mathcal{SC}_{\text{EtS}}$ as in Attack Game 13.6 there exists a CCA adversary $\mathcal{B}_{\text{cca}}$ that attacks $\mathcal{E}$ as in Definition 12.7, and a strong signature adversary $\mathcal{B}'_{\text{sig}}$ that attacks $\mathcal{S}$ as in Attack Game 13.2, where $\mathcal{B}_{\text{cca}}$ and $\mathcal{B}'_{\text{sig}}$ are elementary wrappers around $\mathcal{A}_{\text{cca}}$, such that*
>
> $$\text{SCCA}\mathsf{adv}[\mathcal{A}_{\text{cca}}, \mathcal{SC}_{\text{EtS}}] \leq \text{CCA}_{\text{ad}}\mathsf{adv}[\mathcal{B}_{\text{cca}}, \mathcal{E}] + \text{stSIG}\mathsf{adv}[\mathcal{B}'_{\text{sig}}, \mathcal{S}].$$

Proof sketch. We have to prove both ciphertext integrity and security against chosen ciphertext attack. Both proofs make essential use of the placement of the identifiers id_{S} and id_{R} as defined in the encryption and decryption algorithms. We start with ciphertext integrity.

Proving ciphertext integrity. We begin by constructing adversary $\mathcal{B}_{\text{sig}}$ that interacts with a signature challenger for $\mathcal{S}$, while playing the role of challenger to $\mathcal{A}_{\text{ci}}$ in Attack Game 13.5. $\mathcal{B}_{\text{sig}}$ first obtains a signature public key pk^*_{SIG} from its own challenger.

Next, $\mathcal{A}_{\text{ci}}$ supplies two identities id_{S} and id_{R}. $\mathcal{B}_{\text{sig}}$ then uses G_{ENC} and G_{SIG} to generate two public-key encryption key-pairs $(pk_{\text{ENC,S}}, sk_{\text{ENC,S}})$ and $(pk_{\text{ENC,R}}, sk_{\text{ENC,R}})$, and one signature key-pair $(pk_{\text{SIG,R}}, sk_{\text{SIG,R}})$. It sends to $\mathcal{A}_{\text{ci}}$ the two public keys

$$pk_{\text{S}} := (pk_{\text{ENC,S}},\ pk^*_{\text{SIG}}) \qquad \text{and} \qquad pk_{\text{R}} := (pk_{\text{ENC,R}},\ pk_{\text{SIG,R}}).$$

Note that $\mathcal{B}_{\text{sig}}$ knows all the corresponding secret keys, except for the secret key corresponding to pk^*_{SIG}, which is the challenge signature public key that $\mathcal{B}_{\text{sig}}$ is trying to attack.

$\mathcal{A}_{\text{ci}}$ then issues several encryption and decryption queries.

To process an encryption query, $\mathcal{B}_{\text{sig}}$ begins by encrypting the given message m using the encryption algorithm E with the appropriate public key. This generates a ciphertext c. Next, $\mathcal{B}_{\text{sig}}$ must generate an appropriate signature σ. For an S → R encryption query, $\mathcal{B}_{\text{sig}}$ obtains a signature σ under pk^*_{SIG} on the message (c, id_{R}) by using its own signature challenger. For an X → Y encryption query with $id_{\text{X}} = id_{\text{S}}$, $\mathcal{B}_{\text{sig}}$ obtains a signature σ under pk^*_{SIG} on the message (c, id_{Y}), again, by using its own signature challenger. For an X → Y encryption query with $id_{\text{X}} = id_{\text{R}}$, $\mathcal{B}_{\text{sig}}$ generates σ by signing the message (c, id_{Y}) directly, using the secret key $sk_{\text{SIG,R}}$. In any case, $\mathcal{B}_{\text{sig}}$ responds to the encryption query with the ciphertext/signature pair (c, σ).

$\mathcal{B}_{\text{sig}}$ answers decryption queries from $\mathcal{A}_{\text{ci}}$ by simply running algorithm D_{EtS} on the given data in the query. Indeed, $\mathcal{B}_{\text{sig}}$ has all the required keys to do so.

Eventually, $\mathcal{A}_{\text{ci}}$ outputs a valid ciphertext forgery (c', σ'), where σ' is a valid signature on the message (c', id_{R}). We argue that the message-signature pair $\big((c', id_{\text{R}}),\ \sigma'\big)$ is a strong existential forgery for the signature scheme $\mathcal{S}$. The only way this can fail is if $\mathcal{B}_{\text{sig}}$ had previously asked its challenger for a signature on (c', id_{R}) and the challenger responded with σ'. Observe that the only reason $\mathcal{B}_{\text{sig}}$ would ask for a signature on (c', id_{R}) is as part of responding to an S → R encryption query from $\mathcal{A}_{\text{ci}}$. This is where we make essential use of the fact that the identity id_{R} is included in the data being signed. We conclude that the signature from the challenger cannot be σ' because the ciphertext forgery (c', σ') must be different from all the S → R ciphertexts generated by $\mathcal{B}_{\text{sig}}$. It follows that $\big((c', id_{\text{R}}),\ \sigma'\big)$ is a valid strong existential forgery on $\mathcal{S}$, as required.

Proving chosen ciphertext security. Next, we sketch the proof of CCA security. It is convenient to modify the attack game slightly. Let Game 0 be the original signcryption CCA game between a $\mathcal{SC}_{\text{EtS}}$ challenger and an adversary $\mathcal{A}_{\text{cca}}$. We then define Game 1, which is the same as Game 0, except that we add a "special rejection rule" in the challenger's logic for processing S → R decryption queries. Namely, given an S → R decryption query $(\hat{c}, \hat{\sigma})$, where $\hat{\sigma}$ is a valid signature on $(\hat{c}, id_{\text{R}})$, and $\hat{c}$ is the first component of a response to a previous S → R encryption query, the challenger returns reject without further processing.

It is not difficult to see that Games 0 and 1 proceed identically, unless the challenger rejects a ciphertext $(\hat{c}, \hat{\sigma})$ in Game 1 that would not be rejected in Game 0. However, if $(\hat{c}, \hat{\sigma})$ is such a ciphertext, then $\big((\hat{c}, id_{\text{R}}), \hat{\sigma}\big)$ is a strong existential forgery for $\mathcal{S}$. Therefore, we can construct an adversary $\mathcal{B}'_{\text{sig}}$ whose advantage in strong existential forgery game against $\mathcal{S}$ is equal to the probability that such a ciphertext gets rejected in Game 1.

We now construct an adversary $\mathcal{B}_{\text{cca}}$ whose CCA advantage is the same as $\mathcal{A}_{\text{cca}}$'s advantage in Game 1. As usual, $\mathcal{B}_{\text{cca}}$ interacts with its own CCA challenger, while playing the role of challenger to $\mathcal{A}_{\text{cca}}$ in Game 1.

Adversary $\mathcal{B}_{\text{cca}}$ first obtains an encryption public key pk^*_{ENC} from its own challenger.

Next, $\mathcal{A}_{\text{cca}}$ supplies two identities id_{S} and id_{R}. $\mathcal{B}_{\text{cca}}$ then runs the key-generation algorithm for the signature scheme twice and the key-generation algorithm for the encryption scheme once, and sends to $\mathcal{A}_{\text{cca}}$ the two public keys

$$pk_{\text{S}} := (pk_{\text{ENC,S}},\ pk_{\text{SIG,S}}) \qquad \text{and} \qquad pk_{\text{R}} := (pk^*_{\text{ENC}},\ pk_{\text{SIG,R}}),$$

where it knows all the corresponding secret keys, except for the secret key corresponding to pk^*_{ENC}.

$\mathcal{A}_{\text{cca}}$ then issues several encryption and decryption queries.

Processing encryption queries. Adversary $\mathcal{B}_{\text{cca}}$ answers an S → R encryption query for message pair (m_0, m_1) by issuing an encryption query for (m_0, m_1) to its challenger, relative to the associated data id_{S}. It gets back a ciphertext c, signs (c, id_{R}) to get σ, and sends (c, σ) to $\mathcal{A}_{\text{cca}}$ as a response to the query.

To answer an X → Y encryption query, $\mathcal{B}_{\text{cca}}$ runs algorithm E_{EtS} on the given data in the query. Indeed, $\mathcal{B}_{\text{cca}}$ has all the required keys to do so.

Processing decryption queries. Consider first an S → R decryption query $(\hat{c}, \hat{\sigma})$. Our adversary $\mathcal{B}_{\text{cca}}$ uses the following steps:

1. return reject if $\hat{\sigma}$ is an invalid signature on $(\hat{c}, id_{\text{R}})$ under $pk_{SIG,S}$;

2. return reject if $\hat{c}$ is the first component of any response to an S → R encryption query (this is the special rejection rule we introduced in Game 1);

3. ask the CCA challenger to decrypt $\hat{c}$ using the associated data id_{S}, and return the result (note that because of the logic of Steps 1 and 2, $\mathcal{B}_{\text{cca}}$ has not issued an encryption query to its own challenger corresponding to $(\hat{c}, id_{\text{S}})$).

The logic for processing an X → Y decryption query $(id_{\text{X}}, id_{\text{Y}}, pk_{\text{X}}, (\hat{c}, \hat{\sigma}))$ with $id_{\text{Y}} = id_{\text{R}}$ is similar:

1. return reject if $\hat{\sigma}$ is an invalid signature on $(\hat{c}, id_{\text{R}})$ under pk_{X};

2. ask the CCA challenger to decrypt $\hat{c}$ using the associated data id_{X}, and return the result (note that because $id_{\text{X}} \neq id_{\text{S}}$, $\mathcal{B}_{\text{cca}}$ has not issued an encryption query to its own challenger corresponding to $(\hat{c}, id_{\text{X}})$).

For other decryption queries, we have all the keys necessary to perform the decryption directly.

Finishing up. Eventually, $\mathcal{A}_{\text{cca}}$ outputs a guess $\hat{b} \in \{0, 1\}$. This guess gives $\mathcal{B}_{\text{cca}}$ the same advantage against its CCA challenger that $\mathcal{A}_{\text{cca}}$ has in Game 1. □

Theorem 13.9. *$\mathcal{SC}_{\text{StE}}$ is a secure signcryption scheme assuming $\mathcal{E}$ is a CCA-secure public-key encryption scheme with associated data and $\mathcal{S}$ is a secure signature scheme.*

> *In particular, for every ciphertext integrity adversary $\mathcal{A}_{\text{ci}}$ that attacks $\mathcal{SC}_{\text{EtS}}$ as in Attack Game 13.5 there exists a signature adversary $\mathcal{B}_{\text{sig}}$ that attacks $\mathcal{S}$ as in Attack Game 13.1, and a CCA adversary $\mathcal{B}_{\text{cca}}$ that attacks $\mathcal{E}$ as in Definition 12.7, where $\mathcal{B}_{\text{sig}}$ and $\mathcal{B}'_{\text{cca}}$ are elementary wrappers around $\mathcal{A}_{\text{ci}}$, such that*
>
> $$\text{SCI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{SC}_{\text{EtS}}] \leq \text{SIG}\mathsf{adv}[\mathcal{B}_{\text{sig}}, \mathcal{S}] + \text{CCA}_{\text{ad}}\mathsf{adv}[\mathcal{B}'_{\text{cca}}, \mathcal{E}]$$
>
> *In addition, for every CCA adversary $\mathcal{A}_{\text{cca}}$ that attacks $\mathcal{SC}_{\text{EtS}}$ as in Attack Game 13.6 there exists a CCA adversary $\mathcal{B}_{\text{cca}}$ that attacks $\mathcal{E}$ as in Definition 12.7, where $\mathcal{B}_{\text{cca}}$ is an elementary wrapper around $\mathcal{A}_{\text{cca}}$, such that*
>
> $$\text{SCCA}\mathsf{adv}[\mathcal{A}_{\text{cca}}, \mathcal{SC}_{\text{EtS}}] = \text{CCA}_{\text{ad}}\mathsf{adv}[\mathcal{B}_{\text{cca}}, \mathcal{E}]$$

Proof idea. CCA security for the signcryption scheme follows almost immediately from the CCA security of $\mathcal{E}$. The reader can easily fill in the details.

Proving CI for the signcryption scheme is slightly trickier. Let Game 0 be the original CI attack game. We modify Game 0 so that for each S → R encryption query, instead of computing

$$c \xleftarrow{\text{R}} E(pk_{\text{R}}, (m, \sigma), id_{\text{S}})$$

where

$$\sigma \xleftarrow{\text{R}} S(sk_{\text{S}}, (m, id_{\text{R}})),$$

the challenger instead computes

$$c \xleftarrow{\text{R}} E(pk_{\text{R}}, (m, dummy), id_{\text{S}}).$$

Call this Game 1. Under CCA security for $\mathcal{E}$, the adversary's advantage in breaking CI in Game 0 must be negligibly close to the corresponding advantage in Game 1. However, in Game 1, since the challenger never signs any message of the form $(\cdot, id_{\text{R}})$, breaking CI in Game 1 is tantamount to forging a signature on just such a message.

In proving both security properties, we need to make use of the technical requirement that signatures are bit strings whose length only depends on the message being signed. □

13.7.4 A construction based on Diffie-Hellman key exchange

Our next signcryption construction does not use signatures at all. Instead, we use a non-interactive variant of the Diffie-Hellman key exchange protocol from Section 10.4.1. The protocol uses a group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$. This variant is said to be non-interactive because once every party publishes its contribution to the protocol — g^{α} for some random $\alpha \in \mathbb{Z}_q$ — no more interaction is needed to establish a shared key between any pair of parties. For example, once Alice publishes g^{α} and Bob publishes g^{β}, their shared secret is derived from $g^{\alpha\beta}$. The signcryption scheme we describe can be built from any non-interactive key exchange, but here we present it concretely using Diffie-Hellman key exchange.

The signcryption scheme $\mathcal{SC}_{\text{DH}}$ is built from three ingredients:

- a symmetric cipher $\mathcal{E} = (E_{\text{s}}, D_{\text{s}})$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,
- a group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$, and
- a hash function $H : \mathbb{G}^3 \times \mathcal{I}^2 \to \mathcal{K}$.

Given these ingredients, the system $\mathcal{SC}_{\text{DH}}$ is defined over $(\mathcal{M}, \mathcal{C}, \mathcal{I})$ and works as follows:

- The key generation algorithm G runs as follows:

  $$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad h \leftarrow g^{\alpha}.$$

  The public key is $pk := h$, and the secret key is $sk := \alpha$. We use h_{X} to denote the public key associated with identity id_{X} and use α_{X} to denote the associated secret key.

- $E\big(\alpha_{\text{S}}, id_{\text{S}}, h_{\text{R}}, id_{\text{R}},\ m\big)$ works by first deriving the Diffie-Hellman secret between users S and R, namely $h_{\text{SR}} := g^{\alpha_{\text{S}} \cdot \alpha_{\text{R}}}$, and then encrypting the message m using the symmetric cipher with a key derived from h_{SR}. More precisely, encryption works as follows, where $h_{\text{S}} := g^{\alpha_{\text{S}}}$:

  $$h_{\text{SR}} \leftarrow (h_{\text{R}})^{\alpha_{\text{S}}} = g^{\alpha_{\text{S}} \cdot \alpha_{\text{R}}}, \qquad k \leftarrow H\big(h_{\text{S}}, h_{\text{R}}, h_{\text{SR}},\ id_{\text{S}}, id_{\text{R}}\big), \qquad \text{output } c \xleftarrow{\text{R}} E_{\text{s}}(k, m).$$

- $D\big(h_{\rm S}, id_{\rm S}, \alpha_{\rm R}, id_{\rm R},\ c\big)$ works as follows, where $h_{\rm R} := g^{\alpha_{\rm R}}$:

$$h_{\rm SR} \leftarrow (h_{\rm S})^{\alpha_{\rm R}} = g^{\alpha_{\rm S}\cdot\alpha_{\rm R}}, \qquad k \leftarrow H\big(h_{\rm S}, h_{\rm R}, h_{\rm SR},\ id_{\rm S}, id_{\rm R}\big), \qquad \text{output } D_{\rm s}(k, c).$$

It is easy to verify that $\mathcal{SC}_{\rm DH}$ is correct. To state the security theorem we must first introduce a new assumption, called the *double-interactive CDH assumption.* The assumption is related to, but a little stronger than, the interactive CDH assumption introduced in Section 12.4.

Intuitively, the double-interactive CDH assumption states that given a random instance (g^α, g^β) of the DH problem, it is hard to compute $g^{\alpha\beta}$, even when given access to a DH-decision oracle that recognizes DH-triples of the form $(g^\alpha, \cdot, \cdot)$ or of the form $(\cdot, g^\beta, \cdot)$. More formally, this assumption is defined in terms of the following attack game.

Attack Game 13.7 (Double-Interactive Computational Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger computes

  $$\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \qquad u \leftarrow g^\alpha, \qquad v \leftarrow g^\beta, \qquad w \leftarrow g^{\alpha\beta}$$

  and gives (u, v) to the adversary.

- The adversary makes a sequence of queries to the challenger. Each query is one of the following types:

  α-query: given $(\tilde{v}, \tilde{w}) \in \mathbb{G}^2$, the challenger tests if $\tilde{v}^\alpha = \tilde{w}$;

  β-query: given $(\tilde{u}, \tilde{w}) \in \mathbb{G}^2$, the challenger tests if $\tilde{u}^\beta = \tilde{w}$.

  In either case, if equality holds the challenger sends “yes” to the adversary, and otherwise, sends “no” to the adversary.

- Finally, the adversary outputs some $\hat{w} \in \mathbb{G}$.

We define $\mathcal{A}$’s **advantage in solving the double-interactive computational Diffie-Hellman problem**, denoted $\text{I}^2\text{CDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\hat{w} = w$. □

Definition 13.9 (Double-Interactive computational Diffie-Hellman assumption). *We say that the **double-interactive computational Diffie-Hellman** (I^2CDH) assumption holds for $\mathbb{G}$ if for all efficient adversaries $\mathcal{A}$ the quantity $\text{I}^2\text{CDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ is negligible.*

The following theorem shows $\mathcal{SC}_{\rm DH}$ is a secure signcryption scheme where security is defined as in the previous section (Definition 13.8).

Theorem 13.10. *$\mathcal{SC}_{\rm DH}$ is a secure signcryption scheme assuming $\mathcal{E}$ is an* AE*-secure cipher, the* I^2CDH *assumption holds for $\mathbb{G}$, and the hash function H is modeled as a random oracle.*

> *In particular, for every ciphertext integrity adversary $\mathcal{A}_{\rm ci}$ that attacks $\mathcal{SC}_{\rm DH}$ as in the random oracle variant of Attack Game 13.5, there exists a ciphertext integrity adversary $\mathcal{B}_{\rm ci}$ that attacks $\mathcal{E}$ as in Attack Game 9.1, and an* I^2CDH *adversary $\mathcal{B}_{\rm dh}$ for $\mathbb{G}$, where $\mathcal{B}_{\rm ci}$ and $\mathcal{B}_{\rm dh}$ are elementary wrappers around $\mathcal{A}_{\rm ci}$, such that*
>
> $$\text{SCI}\mathsf{adv}[\mathcal{A}_{\rm ci}, \mathcal{SC}_{\rm DH}] \le \text{CI}\mathsf{adv}[\mathcal{B}_{\rm ci}, \mathcal{E}] + \text{I}^2\text{CDH}\mathsf{adv}[\mathcal{B}_{\rm dh}, \mathbb{G}]$$

In addition, for every CCA adversary $\mathcal{A}_{\rm cca}$ that attacks $\mathcal{SC}_{\rm DH}$ as in the random oracle variant of Attack Game 13.6, there exists a CCA adversary $\mathcal{B}_{\rm cca}$ that attacks $\mathcal{E}$ as in Attack Game 9.2, and an I^{2}CDH *adversary $\mathcal{B}'_{\rm dh}$ for $\mathbb{G}$, where $\mathcal{B}_{\rm cca}$ and $\mathcal{B}'_{\rm iidh}$ are elementary wrappers around $\mathcal{A}_{\rm ci}$, such that*

$$\mathrm{SCCA}\mathsf{adv}[\mathcal{A}_{\rm cca}, \mathcal{SC}_{\rm DH}] \leq \mathrm{CCA}\mathsf{adv}[\mathcal{B}_{\rm cca}, \mathcal{E}] + 2 \cdot \mathrm{I}^2\mathrm{CDH}\mathsf{adv}[\mathcal{B}'_{\rm dh}, \mathbb{G}]$$

The proof of Theorem 13.10 follows from the analysis of Diffie-Hellman as a non-interactive key exchange scheme (Exercise 21.12).

13.7.5 Additional desirable properties: forward secrecy and non-repudiation

So far we looked at three signcryption schemes: $\mathcal{SC}_{\rm DH}$ presented in the previous section and the two schemes presented in Section 13.7.3. All three schemes satisfy the signcryption security definition (Definition 13.8). However, there are significant differences between $\mathcal{SC}_{\rm DH}$ and the two schemes in Section 13.7.3. One difference between $\mathcal{SC}_{\rm DH}$ and the others is a simple inter-operability issue: it requires all users of the system to use the same group $\mathbb{G}$ for generating their keys. This may be acceptable in some settings but not in others, and is inherent to how $\mathcal{SC}_{\rm DH}$ operates.

There are two other, more fundamental, differences that are worth examining further. We explore these differences by defining two new signcryption properties: (1) forward secrecy, and (2) non-repudiation.

13.7.5.1 Property I: forward secrecy (security in case of a sender corruption)

Suppose Alice encrypts a message to Bob and sends the resulting ciphertext c to Bob. A week later the adversary corrupts Alice and steals her secret key. Bob's key remains intact and only known to Bob. One might reasonably expect that the adversary should not be able to decrypt c using Alice's secret key. We refer to this property as *sender corruption forward secrecy* or simply *forward secrecy.*

Let us define more precisely what it means for a signcryption scheme to provide sender corruption forward secrecy. The goal is to ensure that CCA security is maintained even if the adversary obtains the sender's secret key. To do so we make a small tweak to the CCA security game (Attack Game 13.6).

Attack Game 13.8 (CCA security with sender corruption forward secrecy). The game is identical to Attack Game 13.6 except that we change the setup step as follows: in addition to giving the adversary the public keys $pk_{\rm S}$ and $pk_{\rm R}$, the challenger gives the adversary the sender's secret key $sk_{\rm S}$. The corresponding advantage is denoted $\mathrm{SCCA}'\mathsf{adv}[\mathcal{A}, \mathcal{SC}]$. □

Definition 13.10. *A signcryption scheme $\mathcal{SC}$ is said to provide* ***forward secrecy*** *if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{SCCA}'\mathsf{adv}[\mathcal{A}, \mathcal{SC}]$ *is negligible.*

Forward secrecy for sign-then-encrypt. The sign-then-encrypt construction provides forward secrecy: the secret key $sk_{\rm S}$ is only used for signing messages and does not help to decrypt anything. Indeed, from the concrete security bound given in Theorem 13.9, one can see that the bound on the SCCA advantage does not depend at all on the security of the signature scheme.

Forward secrecy for encrypt-then-sign. One might be tempted to say the same thing for encrypt-then-sign; however, this is not quite true in general. Observe that in the concrete security bound in Theorem 13.8, the bound on the SCCA advantage depends on the security of both the signature scheme and the encryption scheme. Indeed, as we already discussed in relation to the need for a strongly secure signature scheme, if the adversary obtains a ciphertext (c, σ) in response to an S → R encryption query, and could compute a valid signature $\sigma' \neq \sigma$ on (c, id_{R}), then by the rules of the CCA attack game, the adversary would be free to submit (c, σ') as an S → R decryption query, completely breaking CCA security.

Now, without the sender's signing key, this attack would be infeasible. But with the signing key, it is easy if the signature algorithm is probabilistic (we will see such signature schemes later): the adversary can use the sender's signing key to generate a different signature on an inner S → R ciphertext and obtain a "new" encrypt-then-sign ciphertext that it can submit to the decryption oracle.

However, all is not lost. There are a couple of ways to salvage the forward secrecy property of encrypt-then-sign. One way is to salvage the situation is to employ a signature scheme that has *unique signatures* (i.e., for every public key and message, there is at most one valid signature — full domain hash is such a scheme). Then the above attack becomes impossible, even with the signing key. See also Exercise 13.19, which discusses a modification of encrypt-then-sign which achieves forward secrecy more generically.

Another way to salvage the situation is to weaken the security definition slightly, by simply not allowing the adversary to submit a decryption query for the ciphertext (c, σ') in the attack game. Is this reasonable? Arguably, it is, as anyone can easily tell that the (c, σ) and (c, σ') decrypt to the same thing if σ and σ' are both valid signatures on c. Indeed, such a restriction on the adversary corresponds to the notion of gCCA security discussed in Exercise 12.2, and is actually quite acceptable for most applications.

Forward secrecy for $\mathcal{SC}_{\mathrm{DH}}$. The $\mathcal{SC}_{\mathrm{DH}}$ signcryption system is not forward secure: given the secret key of the sender, the adversary can decrypt any ciphertext generated by the sender. Fortunately, we can enhance $\mathcal{SC}_{\mathrm{DH}}$ to provide forward secrecy against sender corruptions.

Enhanced $\mathcal{SC}_{\mathrm{DH}}$. Using the notation of Section 13.7.4, the enhanced $\mathcal{SC}_{\mathrm{DH}}$ signcryption system, denoted $\mathcal{SC}'_{\mathrm{DH}}$, is defined over $(\mathcal{M},\ \mathbb{G} \times \mathcal{C},\ \mathcal{I})$ and works as follows:

- The key generation algorithm G is as in $\mathcal{SC}_{\mathrm{DH}}$. We use h_{X} to denote the public key associated with identity id_{X} and use α_{X} to denote the associated secret key.
- $E\big(\alpha_{\mathrm{S}}, id_{\mathrm{S}}, h_{\mathrm{R}}, id_{\mathrm{R}},\ m\big)$ works as follows, where $h_{\mathrm{S}} := g^{\alpha_{\mathrm{S}}}$:

$$\begin{aligned}
&\beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad v \leftarrow g^{\beta},\\
&h_{\mathrm{SR}} \leftarrow (h_{\mathrm{R}})^{\alpha_{\mathrm{S}}}, \quad w \leftarrow (h_{\mathrm{R}})^{\beta},\\
&k \leftarrow H\big(v, w, h_{\mathrm{S}}, h_{\mathrm{R}}, h_{\mathrm{SR}},\ id_{\mathrm{S}},\ id_{\mathrm{R}}\big), \quad c \leftarrow E_{\mathrm{s}}(k, m)\\
&\text{output } (v, c).
\end{aligned}$$

- $D\big(h_{\mathrm{S}}, id_{\mathrm{S}}, \alpha_{\mathrm{R}}, id_{\mathrm{R}},\ (v, c)\big)$ works as follows, where $h_{\mathrm{R}} := g^{\alpha_{\mathrm{R}}}$:

$$h_{\mathrm{SR}} \leftarrow (h_{\mathrm{S}})^{\alpha_{\mathrm{R}}}, \quad w \leftarrow v^{\alpha_{\mathrm{R}}}, \quad k \leftarrow H\big(v, w, h_{\mathrm{S}}, h_{\mathrm{R}}, h_{\mathrm{SR}},\ id_{\mathrm{S}},\ id_{\mathrm{R}}\big), \quad \text{output } D_{\mathrm{s}}(k, c).$$

In this scheme, the symmetric encryption key is derived from the long term secret key $h_{\text{SR}} = g^{\alpha_{\text{S}} \cdot \alpha_{\text{R}}}$ along with an ephemeral secret key $w = g^{\beta \cdot \alpha_{\text{R}}}$. The ephemeral secret key ensures CCA security even when the attacker knows the sender's secret key α_{S}. The long term secret key ensures ciphertext integrity, as before.

The following theorem proves security of $\mathcal{SC}'_{\text{DH}}$ in this stronger signcryption security model. Interestingly, the proof of CCA security for $\mathcal{SC}'_{\text{DH}}$ only relies on the simpler interactive Diffie-Hellman assumption from Section 12.4, not the double-interactive assumption I^2CDH that we used in proving CCA-security for $\mathcal{SC}_{\text{DH}}$.

Theorem 13.11. *$\mathcal{SC}'_{\text{DH}}$ is a secure signcryption scheme that provides forward secrecy assuming $\mathcal{E}$ is an* AE*-secure cipher, the* I^2CDH *assumption (Definition 13.9) holds in $\mathbb{G}$, and the hash function H is modeled as a random oracle.*

> *In particular, for every ciphertext integrity adversary $\mathcal{A}_{\text{ci}}$ that attacks $\mathcal{SC}'_{\text{DH}}$ as in the random oracle variant of Attack Game 13.5, there exists a ciphertext integrity adversary $\mathcal{B}_{\text{ci}}$ that attacks $\mathcal{E}$ as in Attack Game 9.1, and an* I^2CDH *adversary $\mathcal{B}_{\text{dh}}$ for $\mathbb{G}$, where $\mathcal{B}_{\text{s}}$ and $\mathcal{B}_{\text{dh}}$ are elementary wrappers around $\mathcal{A}_{\text{ci}}$, such that*
>
> $$\text{SCI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{SC}_{\text{DH}}] \leq \text{CI}\mathsf{adv}[\mathcal{B}_{\text{ci}}, \mathcal{E}] + \text{I}^2\text{CDH}\mathsf{adv}[\mathcal{B}_{\text{dh}}, \mathbb{G}].$$
>
> *In addition, for every CCA adversary $\mathcal{A}_{\text{cca}}$ that attacks $\mathcal{SC}_{\text{DH}}$ as in the random oracle variant of Attack Game 13.6, there exists a 1CCA adversary $\mathcal{B}_{\text{1cca}}$ that attacks $\mathcal{E}$ as in Definition 9.6, and an ICDH adversary $\mathcal{B}'_{\text{dh}}$ for $\mathbb{G}$, where $\mathcal{B}_{\text{s}}$ and $\mathcal{B}'_{\text{dh}}$ are elementary wrappers around $\mathcal{A}_{\text{ci}}$, such that*
>
> $$\text{SCCA}'\mathsf{adv}[\mathcal{A}_{\text{cca}}, \mathcal{SC}_{\text{DH}}] \leq \text{1CCA}\mathsf{adv}[\mathcal{B}_{\text{1cca}}, \mathcal{E}] + 2 \cdot \text{ICDH}\mathsf{adv}[\mathcal{B}'_{\text{dh}}, \mathbb{G}].$$

Proof idea. The proof of ciphertext integrity is very similar to the proof in Theorem 13.10. The proof of CCA security with forward secrecy, where the adversary is given the sender's secret key, is almost identical to the proof of ElGamal CCA security (Theorem 12.4), together with the random self reduction for CDH (see Exercise 10.5); as such, the ICDH assumption is sufficient for the proof.
□

13.7.5.2 Property II: non-repudiation (security in case of a recipient corruption)

Suppose Alice encrypts a message m to Bob and obtains the ciphertext c. The question is, does c, together with Bob's secret key, provide Bob with enough evidence to convince a third party that Alice actually sent the message m to Bob? We call this property **non-repudiation**. We explained at the beginning of the chapter that such evidence is inherently limited in its persuasive powers: Alice can simply claim that her secret key was stolen from her and that someone else produced c, or she can deliberately leak her secret key in order to repudiate c. Nevertheless, since non-repudiation may be required in some situations, we define it and show how to construct signcryption schemes that provide it.

Non-repudiation is also useful as a partial defense against a compromise of Bob's secret key. If the signcryption scheme does not provide non-repudiation, then an attacker can use Bob's compromised secret key to send messages to Bob pretending to be from Alice. This attack is called **key compromise impersonation** or KCI. Non-repudiation ensures that Bob's key cannot be used to impersonate Alice and therefore a KCI attack is not possible.

Defining non-repudiation. We define non-repudiation by slightly tweaking the ciphertext integrity game (Attack Game 13.5). The goal is to ensure that ciphertext integrity is maintained even if the adversary obtains the recipient's secret key. The modified game is as follows:

Attack Game 13.9 (Ciphertext integrity with non-repudiation). The game is identical to Attack Game 13.5 except that we change the setup step as follows: in addition to giving the adversary the public keys pk_{S} and pk_{R}, the challenger gives the adversary the receiver's secret key sk_{R}. The corresponding advantage is denoted $\mathrm{SCI}'\mathsf{adv}[\mathcal{A}, \mathcal{SC}]$. □

Definition 13.11. *A signcryption scheme $\mathcal{SC}$ is said to provide **non-repudiation**, if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{SCI}'\mathsf{adv}[\mathcal{A}, \mathcal{SC}]$ *is negligible.*

Non-repudiation for encrypt-then-sign. The encrypt-then-sign construction provides non-repudiation: the secret key sk_{R} is only used to decrypt ciphertexts and does not help in signing anything. Indeed, in the concrete security bound given in Theorem 13.8, one can see that the bound on SCI advantage does not depend at all on the security of the signature scheme.

Non-repudiation for sign-then-encrypt. The same argument cannot be made for the sign-then-encrypt construction. Observe that in the concrete security bound given in Theorem 13.9, the bound on the SCCI advantage depends on both the security of the encryption scheme and the signature scheme. In fact, it is easy to see that this scheme cannot provide non-repudiation as we have defined it. Indeed, given the decryption key, one can always decrypt a ciphertext encrypting (m, σ) and then simply re-encrypt it, obtaining a different, but still valid, ciphertext.

Although sign-then-encrypt does not satisfy our definition of non-repudiation, it does satisfy a weaker notion that corresponds to plaintext integrity, rather than ciphertext integrity. Roughly speaking, this property corresponds to a modification of Attack Game 13.9 in which the winning condition is changed: to win the game, its candidate forgery $\hat{c}$ must decrypt to a message that was never submitted as an S → R encryption query. We leave it to the reader to flesh out the details of this definition, and to show that sign-then-encrypt satisfies this weaker notion of non-repudiation. See also Exercise 9.15.

Non-repudiation for $\mathcal{SC}_{\mathrm{DH}}$. The $\mathcal{SC}_{\mathrm{DH}}$ scheme does not provide non-repudiation, in a very strong sense: the recipient can encrypt any message just as well as the sender. The same is true for $\mathcal{SC}'_{\mathrm{DH}}$. Because of this property, both these schemes provide *complete deniability* — the sender can always claim (correctly) that any ciphertext it generated could have been generated by the receiver. In real-world settings this deniability property may be considered a feature rather than a bug.

Summary. Forward secrecy is clearly a desirable property in real-world systems. Non-repudiation, in the context of signcryption, is not always needed. In situations where forward secrecy is desirable, but non-repudiation is not, the $\mathcal{SC}'_{\mathrm{DH}}$ scheme is a very efficient solution. In situations where both properties are needed, encrypt-then-sign is a safer option than sign-then-encrypt, despite only providing a slightly weaker notion of CCA security, as discussed above. Exercise 13.19 is a variation of encrypt-then-sign that is also an attractive option to ensure both forward secrecy and non-repudiation.

13.8 Certificates and the public-key infrastructure

We next turn to one of the central applications of digital signatures, namely, their use in certificates and public-key infrastructure. In its simplest form, a certificate is a blob of data that binds a public-key to an identity. This binding is asserted by a third party called a **certificate authority**, or simply a CA. We first discuss the mechanics of how certificates are issued and then discuss some real-world complications in managing certificates — specifically, how to cope with misbehaving CAs and how to revoke certificates.

Obtaining a certificate. Say Alice wishes to obtain a certificate for her domain `alice.com`. She sends a **certificate signing request** (CSR) to the CA, that contains Alice's identity, her email address, and the public key that she wishes to bind to her domain.

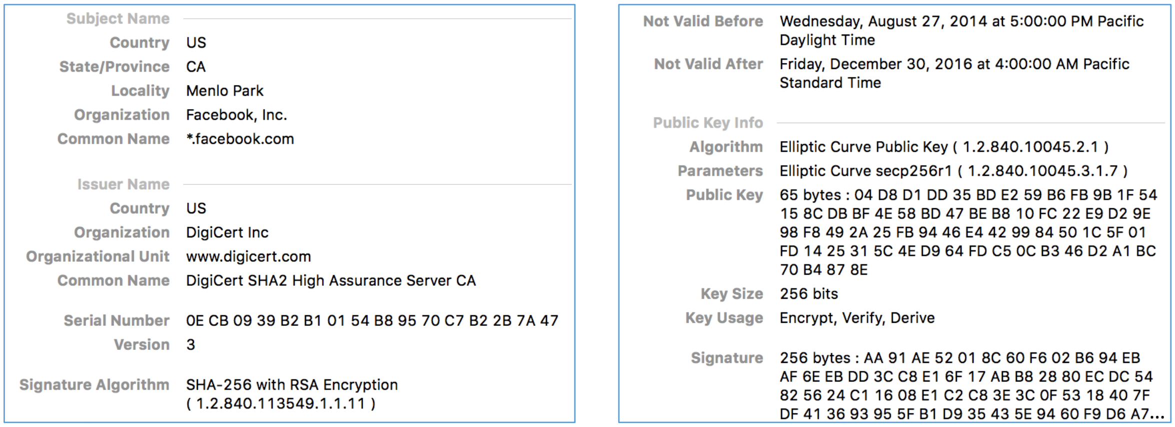
Once the CA receives the CSR, it checks that Alice is who she claims to be. In some cases this check is as naive as sending a challenge email to Alice's address and verifying that she can read the email. In other cases this is done by requiring notarized documents proving Alice's identity. We emphasize that certifying Alice's real-world identity is the primary service that the CA provides. If all the checks succeed, the CA assembles the relevant data into a certificate structure, and signs it using the CA's secret signing key. The resulting signed blob is a certificate that binds the public key in the CSR to Alice's identity. Some CAs issue certificates for free, while others require payment from Alice to issue a certificate.

The resulting signed certificate can be sent to anyone that needs to communicate securely with Alice. Anyone who has the CA's verification key can verify the certificate and gain some confidence that the certified public key belongs to Alice.

X.509 certificates. Certificates are formatted according to a standard called X.509. Fig. 13.4 gives an example X.509 certificate that binds a public key to an entity identified in the subject field. Here the entity happens to be Facebook Inc., and its public key is an (elliptic-curve) ElGamal public key, shown on the right side of the figure. The certificate was issued by a CA called DigiCert Inc., who used its RSA signing key to sign the certificate using the PKCS1 standard with SHA256 as the hash function. A portion of the CA's signature is shown on the bottom right of the figure. To verify this certificate one would need the public key for DigiCert Inc.

Every X.509 certificate has a serial number that plays a role in certificate revocation, as explained in Section 13.8.2 below. Certificates also have a validity window: a time when the certificate becomes active, and a time when the certificate expires. A certificate is considered invalid outside of its validity window, and should be rejected by the verifier. The validity window is typically one or two years, but can be longer or shorter. For example, the certificate in Fig. 13.4 has a validity window of about seventeen months. The reason for limiting certificate lifetime is to ensure that if the private key is stolen by an attacker, that attacker can only abuse the key for a limited period of time. The longer the validity window, the longer an attacker can abuse a stolen secret key. We discuss this further in Section 13.8.2 where we discuss certificate revocation.

A certificate issued by a CA can be verified by anyone who has that CA's public key. If there were only one CA in the world then everyone could store a copy of that CA's public key and use it to verify all certificates. However, a single global CA would not work well. First, every country wants to run a CA for local businesses in its region. Second, to keep the price of certificates low, it is best to enable multiple CAs to compete for the business of issuing certificates. Currently there are thousands of active CAs issuing certificates.



Subject Name	
Country	US
State/Province	CA
Locality	Menlo Park
Organization	Facebook, Inc.
Common Name	*.facebook.com
Issuer Name	
Country	US
Organization	DigiCert Inc
Organizational Unit	www.digicert.com
Common Name	DigiCert SHA2 High Assurance Server CA
Serial Number	0E CB 09 39 B2 B1 01 54 B8 95 70 C7 B2 2B 7A 47
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)

Not Valid Before	Wednesday, August 27, 2014 at 5:00:00 PM Pacific Daylight Time
Not Valid After	Friday, December 30, 2016 at 4:00:00 AM Pacific Standard Time
Public Key Info	
Algorithm	Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters	Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key	65 bytes : 04 D8 D1 DD 35 BD E2 59 B6 FB 9B 1F 54 15 8C DB BF 4E 58 BD 47 BE B8 10 FC 22 E9 D2 9E 98 F8 49 2A 25 FB 94 46 E4 42 99 84 50 1C 5F 01 FD 14 25 31 5C 4E D9 64 FD C5 0C B3 46 D2 A1 BC 70 B4 87 8E
Key Size	256 bits
Key Usage	Encrypt, Verify, Derive
Signature	256 bytes : AA 91 AE 52 01 8C 60 F6 02 B6 94 EB AF 6E EB DD 3C C8 E1 6F 17 AB B8 28 80 EC DC 54 82 56 24 C1 16 08 E1 C2 C8 3E 3C 0F 53 18 40 7F DF 41 36 93 95 5F B1 D9 35 43 5E 94 60 F9 D6 A7...

Figure 13.4: An example X.509 certificate

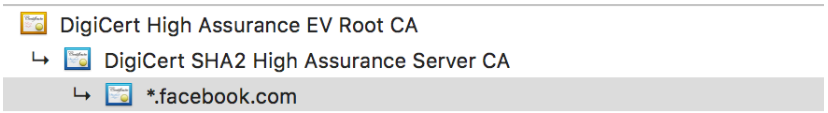
Certificate chains. Since there are multiple CAs issuing certificates, and new ones can appear at any time, the challenge is to distribute CA public keys to end-users who need to verify certificates. The solution, called a **certificate chain**, is to allow one CA to certify the public key of another CA. This process can repeat recursively, resulting in a chain of certificates where every certificate in the chain certifies the public key of the next CA in the chain.

The public key of top level CAs, called **root CAs**, are pre-installed on all clients that need to verify certificates. There are several hundred such root CAs that ship with every standard operating system. A root CA can issue a certificate to an **intermediate CA**, and an intermediate CA can issue a certificate to another intermediate CA. Continuing this way we obtain a chain of certificates starting from the root and containing one or more intermediate CAs. Finally, the CA at the bottom of the chain issues a client certificate for the end identity, such as Facebook in Fig. 13.4.

The certificate chain for the Facebook certificate is shown in Fig. 13.5. The root CA is DigiCert Inc., but its secret key is kept offline to reduce the risk of theft. The root secret key is only used for one thing: to issue a certificate for an intermediate CA, that is also owned by DigiCert Inc. That intermediate CA then uses its secret key to issue client certificates to customers like Facebook. If the intermediate CA's secret key is lost or stolen, the corresponding certificate can be revoked, and the root CA can issue a new certificate for the intermediate CA.

To verify this certificate chain of length three, the verifier needs a local trusted copy of the public key of the root CA. That public key lets the verifier check validity of the certificate issued to the intermediate CA. If valid, it has some assurance that the intermediate CA can be trusted. The verifier then checks validity of the certificate issued to Facebook by the intermediate CA. If valid, the verifier has some assurance that it has the correct public key for Facebook.

Certificate chains and basic constraints. X.509 certificates contain many fields and we only scratched the surface in our discussion above. In the context of certificate chains we mention two fields that play an important security role. In Fig. 13.5 we saw that the certificate chain issued to Facebook has length three. What is to prevent Facebook from behaving like a CA and generating a certificate chain of length four for another identity, say `alice.com`? This certificate chain, unbeknownst to Alice, would enable Facebook to impersonate `alice.com` and even eavesdrop



DigiCert High Assurance EV Root CA

↳ DigiCert SHA2 High Assurance Server CA

↳ *.facebook.com

Figure 13.5: An example certificate chain

on traffic to `alice.com` by acting as a "man in the middle," similar to what we saw in Section 10.7.

The reason Facebook cannot issue certificates is because of a **basic constraint** field that every CA must embed in the certificates that it issues. This field, called the "CA" field, is set to true if the entity being certified is allowed to act as a CA, and is set to false otherwise. For a certificate chain of length ℓ to be valid, it must be the case that the top $\ell - 1$ certificates in the chain have their CA basic constraint set to true. If not, the chain must be rejected by the verifier. Facebook's certificate has its CA field set to "false," preventing Facebook from acting as an intermediate CA.

Certificate validation includes many other such subtle checks, and is generally quite tricky to implement correctly. Many systems that implement custom certificate validation were found to be insecure [72], making them vulnerable to impersonation and man-in-the-middle attacks.

13.8.1 Coping with malicious or negligent certificate authorities

By now it should be clear that CAs have a lot of power. Any CA can issue a rogue certificate and bind the wrong public key to Facebook. If left unchecked, a rogue certificate would enable an adversary to mount a man-in-the-middle attack on traffic to Facebook and eavesdrop on all traffic between Facebook and unsuspecting users. We will discuss these attacks in detail in Chapter 21 after we discuss the TLS session setup mechanism. Several commercial tools make this quite easy to do in practice.

There are currently thousands of intermediate CAs operating on the Internet and all are trusted to issue certificates. Due to the large number of CAs, it is not surprising that wrong certificates are routinely discovered. Here is a small sample of incidents:

- Diginotar was a Dutch certificate authority that was hacked in 2011. The attacker obtained a Diginotar signed certificate for `*.google.com`, and for many other domains, letting the attacker mount a man-in-the-middle attack on all these domains. In response, major Web browser vendors revoked trust in all certificates issued by the Diginotar CA, causing Diginotar to declare bankruptcy in Sep. 2011.

- India NIC in 2013 erroneously issued certificates for several Google and Yahoo domains [101]. This intermediate CA was certified by India CCA, a root CA trusted by Microsoft Windows. As a result, the Chrome browser no longer trusts certificates issued by India NIC. Furthermore, following this incident, the India CCA root CA is only trusted to issue certificates for domains ending in `.in`, such as `google.co.in`.

- Verisign in 2001 erroneously issued a Microsoft code-signing certificate to an individual masquerading as a Microsoft employee [113]. This certificate enabled that individual to distribute code that legitimately looked like it was written by Microsoft. In response, Microsoft issued a Windows software patch that revoked trust in this certificate.

As we can see, many of these events are due to an erroneous process at the CA. Any time a certificate is issued that binds a wrong public key to a domain, that certificate enables a man-in-the-middle attack on the target domain. The end result is that the attacker can inspect and modify traffic to and from the victim domain.

The question then is how to identify and contain misbehaving CAs. We discuss two ideas below.

Certificate pinning. The reader must be wondering how the incidents mentioned above were discovered in the first place. The answer is a mechanism called **certificate pinning**, which is now widely supported by Web browsers. The basic idea is that browsers are pre-configured to know that the only CA authorized to issue certificates for the domain `facebook.com` is "DigiCert SHA2 High Assurance Server CA," as shown in Fig. 13.5. If a browser ever sees a certificate for `facebook.com` that is issued by a different CA, it does two things: first, it treats the certificate as invalid and closes the connection, and second, it optionally alerts an administrator at Facebook that a rogue certificate was discovered. The incident discussed above, involving `India NIC`, was discovered thanks to a certificate pin for `gmail.com`. Browsers in India alerted Google to the existence of a rogue certificate chain for `gmail.com`. Google then took action to revoke the chain and launch an investigation. The signatures in the rogue chain provide irrefutable evidence that something went wrong at the issuing CA.

In more detail, certificate pinning works as follows. Every browser maintains a pinning database, where, roughly speaking, every row in the database is a tuple of the form

$$(\text{domain},\ \text{hash}_0,\ \text{hash}_1,\ \ldots).$$

Each hash_i is the output of a hash function (so for SHA256, a 32-byte string). The data for each record is provided by the domain owner. Facebook, for example, provides the hashes for the `facebook.com` domain.

When the browser connects to a domain using HTTPS, that domain sends its certificate chain to the browser. If the domain is in the pinning database, the browser computes the hash of each certificate in the chain. Let S be the resulting set of hash values. Let T be the set of hash values in the pinning record for this domain. If the intersection of S and T is empty, the certificate chain is rejected, and the browser optionally sends an alert to the domain administrator indicating that a rogue certificate chain was encountered.

To see how this works, consider again the example chain in Fig. 13.5. The pinning record for the domain `facebook.com` is just a single hash, namely the hash of the certificate for "DigiCert SHA2 High Assurance Server CA." In other words, the set T contains a single hash value. If the browser encounters a certificate chain for `facebook.com` where none of the certificates in the chain hash to the pinned value, the certificate chain is rejected. More generally, domains that purchase certificates from multiple CAs include the hash of all those CA certificates in their pinning record.

Why does Facebook write the hash of its CA certificate in the Facebook pinning record? Why not write the hash of the Facebook certificate from Fig. 13.4 in the pinning record? In fact, writing the CA certificate in the pinning record seems insecure; it makes it possible for DigiCert to issue a rogue certificate for `facebook.com` that will be accepted by browsers, despite the pinning record. If instead, Facebook wrote the Facebook certificate in Fig. 13.4 as the only hash value in the pinning record, then DigiCert would be unable to issue a rogue certificate for `facebook.com`. The only certificate for `facebook.com` that browsers would accept would be the certificate in Fig. 13.4. However, there is enormous risk in doing so. If Facebook somehow lost its own secret key, then no browser in the world will be able to connect to `facebook.com`. Pinning the CA certificate lets Facebook recover from key loss by simply asking DigiCert to issue a new certificate for `facebook.com`. Thus, the risk of bringing down the site outweighs the security risk of DigiCert issuing a rogue certificate. While losing the secret key may not be a concern for a large site like Facebook, it is a significant concern for smaller sites who use certificate pinning.

Finally we mention that there are two mechanisms for creating a pinning record: static and dynamic. Static pins are maintained by the browser vendor and shipped with the browser. Dynamic pins allow a domain to declare its own pins via an HTTP header, sent from the server to the browser, as follows:

```
Public-Key-Pins:  pin-sha256="hash"; max-age=expireTime
                  [; report-uri="reportURI"] [; includeSubDomains]
```

Here `pin-sha256` is the hash value to pin to, `max-age` indicates when the browser will forget the pin, and `report-uri` is an optional address where pin validation failures should be reported. The HTTP header is accepted by the browser only if it is sent over an encrypted HTTPS session. The header is ignored when sent over unencrypted HTTP. This prevents a network attacker from injecting invalid pins.

Certificate transparency. A completely different approach to coping with misbehaving CAs is based on public certificate logs. Suppose there existed a public **certificate log** that contained a list of all the certificates ever issued. Then a company, like Facebook, could monitor the log and learn when someone issues a rogue certificate for `facebook.com`. This idea, called certificate transparency, is compelling, but is not easy to implement. How do we ensure that every certificate ever issued is on the log? How do we ensure that the log is append-only so that a rogue certificate cannot be removed from the log? How do we ensure that everyone in the world sees the same version of the log?

Certificate transparency provides answers to all these questions. Here, we just sketch the architecture. When a CA decides to support certificate transparency, it chooses one of the public certificate logs and augments its certificate issuance procedure as follows: (1) before signing a new certificate, the CA sends the certificate data to the log, (2) the log signs the certificate data and sends back the signature, called a **signed certificate timestamp** (SCT), (3) the CA adds the SCT as an extension to the certificate data and signs the resulting structure, to obtain the final issued certificate. The SCT is embedded as an extension in the newly issued certificate.

The SCT is a promise by the certificate log to post the certificate to its log within a certain time period, say one day. At noon every day, the certificate log appends all the new certificates it received during that day to the log. It then computes a hash of the entire log and signs the hash along with the current timestamp. The log data and the signature are made publicly available for download by anyone.

The next piece of the architecture is a set of auditors that run all over the world and ensure that the certificate logs are behaving honestly — they are posting to the log as required, and they never remove data from the log. Every day the auditors download all the latest logs and their signatures, and check that no certificates were removed from the logs. If they find that a certificate on some day t is missing from the log on day $t + 1$, then the log signatures from days t and $t + 1$ are evidence that the certificate log is misbehaving.

Moreover, every auditor crawls the Internet looking for certificates. For each certificate that contains an SCT extension, the auditor does an **inclusion check**: it verifies that the certificate appears on the latest version of the log that the SCT points to. If not, then the signed SCT along with the signed log, are evidence that the certificate log is misbehaving. This process ensures that all deployed certificates with an SCT extension must appear on one of the logs; otherwise one of the certificate logs is caught misbehaving. Anyone can run the auditor protocol. In particular, every Web browser can optionally function as an auditor and run the inclusion check before choosing to trust a certificate. If the inclusion check fails, the browser notifies the browser vendor who can launch an investigation into the practices of the certificate log in question. We note that by using a data structure, called a Merkle hash tree, the inclusion check can be done very efficiently, without having to download the entire log. We discuss Merkle hash trees and their applications in Section 8.9.

Unfortunately, auditing is not enough. A devious certificate log can misbehave in a way that will not be caught by the auditing process above. Suppose that a CA issues a rogue certificate for `facebook.com` and writes it to a certificate log, as required. Now, the certificate log creates two signed versions of the log: one with the rogue certificate and one without. Whenever an auditor downloads the log, it is given the version of the log with the rogue certificate. To the auditor, all seems well. However, when Facebook reads the log to look for rogue `facebook.com` certificates, it is given the version without the rogue certificate. This prevents Facebook from discovering the rogue certificate, even though all the auditors believe that the certificate log is behaving honestly. The architecture mitigates this attack in two ways. First, every certificate must be written to at least two logs, so that both certificate logs must be corrupt for the attack to succeed. Second, there is a broadcast mechanism in which the daily hash of all the logs is broadcast to all entities in the system. A log that does not match the broadcast hash is simply ignored.

The final piece of the architecture is mandating certificate transparency on all CAs. At some point in the future, browser vendors could decide to reject all certificates that do not have a valid SCT from a trusted certificate log. This will effectively force universal adoption of certificate transparency by all CAs. At that point, if a rogue certificate is issued, it will be discovered on one of the certificate logs and revoked. We note that many of the large CAs already support certificate transparency.

13.8.2 Certificate revocation

We next look at the question of revoking certificates. The goal of certificate revocation is to ensure that, after a certificate is revoked, all clients treat that certificate as invalid.

There are many reasons why a certificate may need to be revoked. The certificate could have been issued in error, as discussed in the previous subsection. The private key corresponding to the certificate may have been stolen, in which case the certificate owner will want to revoke the certificate so it cannot be abused. This happens all the time; sites get hacked and their secrets are stolen. One well-publicized example is the **heartbleed** event. Heartbleed is a bug in the OpenSSL library that was introduced in 2012. The bug was publicly discovered and fixed in 2014, but during those two years, from 2012 to 2014, a remote attacker could have easily extracted the secret key from every server that used OpenSSL, by simply sending a particular malformed request to the server. When the vulnerability was discovered in 2014, thousands of certificates had to be revoked because of concern that the corresponding secret keys were compromised.

Given the need to revoke certificates, we next describe a few techniques to do so.

Short-lived certificates. Recall that every certificate has a validity period and the certificate is no longer valid after its expiration date. Usually, when an entity like Facebook buys a one-year certificate, the CA issues a certificate that expires a year after it was issued. Imagine that instead, the CA generated 365 certificates, where each one is valid for exactly one day during that year. All 365 certificates are for the same public key; the only difference is the validity window. These certificates are called **short-lived certificates** because each is valid for only one day.

The CA keeps all these certificates to itself, and releases each one at most a week before it becomes valid. So, the certificate to be used on January 28 is made available on January 21, but no sooner. Every day Facebook connects to a public site provided by the CA and fetches the certificate to be used a week later. This is a simple process to automate, and if anything goes wrong, there is an entire week to fix the problem.

Now, when Facebook needs to revoke its certificate, it simply instructs the CA to stop releasing short-lived certificates for its domain. This effectively makes the stolen private key useless after at most one week. If faster revocation is needed, the CA can be told to release each short-lived certificate only an hour before it becomes valid, in which case the secret key becomes useless at most 25 hours after it is revoked.

The use of short-lived certificates is the simplest and most practical technique for certificate revocation available, yet it is not widely used. The next two techniques are more cumbersome, but are the ones most often used by CAs.

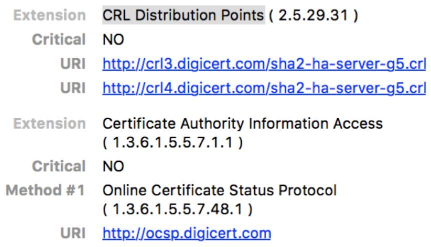
Certificate revocation lists (CRLs). A very different approach is to have the CA collect all certificate revocation requests from all its customers, and on a weekly basis issue a signed list of all certificates that were revoked during that week. This list, called a **certificate revocation list** (CRL), contains the serial numbers of all the certificates that were revoked during that week. The list is signed by the CA.

Every certificate includes a special extension field called **CRL Distribution Points**, as shown in Fig. 13.6. This field instructs the verifier where to obtain the CRL from the issuing CA. The CA must run a public server that serves this list to anyone who asks for it.

When a client needs to validate a certificate, it is expected to download the CRL from the CRL distribution point, and reject the certificate if its serial number appears in the CRL. For performance reasons, the CRL has a validity period of, say one week, and the client can cache the CRL for that period. As a result, it may take a week from the time a revocation request is issued until all clients learn that the certificate has been revoked.

There are two significant difficulties with this approach. First, what should the client do if the CRL server does not respond to a CRL download request? If the client were to accept the certificate, then this opens up a very serious attack. An attacker can cause the client to accept a revoked certificate by simply blocking its connection to the CRL server. Clearly the safe thing to do is to reject the certificate; however, this is also problematic. It means that if the CRL server run by Facebook's CA were to accidentally crash, then no one could connect to Facebook until the CA fixes the CRL server. As you can imagine, this does not go over well with Facebook.

A second difficulty with CRLs is that they force the client to download a large list of revoked certificates that the client does not need. The client is only interested in learning the validity status of a single certificate: the one it is trying to validate. The client does not need, and is not interested in, the status of other certificates. This inefficiency is addressed by a better mechanism called OCSP, which we discuss next.



Extension CRL Distribution Points (2.5.29.31)
Critical NO
URI http://crl3.digicert.com/sha2-ha-server-g5.crl
URI http://crl4.digicert.com/sha2-ha-server-g5.crl

Extension Certificate Authority Information Access (1.3.6.1.5.5.7.1.1)
Critical NO
Method #1 Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)
URI http://ocsp.digicert.com

Figure 13.6: The CRL and OCSP fields in the certificate from Fig. 13.4.

The online certificate status protocol (OCSP). A client that needs to validate a certificate can use the OCSP protocol to query the CA about the status of that specific certificate. To make this work, the CA includes an OCSP extension field in the certificate, as shown in Fig. 13.6. This field tells the client where to send its OCSP query. In addition, the CA must setup a server, called an **OCSP responder**, that responds to OCSP queries from clients.

When the client needs to validate a certificate, it sends the certificate's serial number to the OCSP responder. Roughly speaking, the responder sends back a signed message saying "valid" or "invalid". If "invalid" the client rejects the certificate. OCSP responses can be cached for, say a week, and consequently revocation only takes effect a week after a request is issued.

As with CRLs, it is not clear what the client should do when the OCSP responder simply does not respond. Moreover, OCSP introduces yet another problem. Because a client, such as a Web browser, sends to the CA the serial number of every certificate it encounters, the CA can effectively learn what web sites the user is visiting. This is a breach of user privacy. The problem can be partially mitigated by an extension to OCSP, called **OCSP stapling**, but this extension is rarely used.

13.9 Case study: legal aspects of digital signatures

While cryptographers say that a signature scheme is secure if it existentially unforgeable under a chosen message attack, the legal standard for what constitutes a valid digital signature on an electronic document is quite different. The legal definition tries to capture the notion of intent: a signature is valid if the signer "intended" to sign the document. Here we briefly review a few legislative efforts that try to articulate this notion. This discussion shows that a cryptographic digital signature is very different from a legally binding electronic signature.

Electronic signatures in the United States. On June 30, 2000, the U.S. Congress enacted the Electronic Signatures in Global and National Commerce Act, known as E-SIGN. The goal of E-SIGN is to facilitate the use of electronic signatures in interstate and foreign commerce.

The U.S. statute of frauds requires that contracts for the sale of goods in excess of $500 be signed. To be enforceable under U.S. law, E-SIGN requires that an electronic signature possess three elements: (1) a symbol or sound, (2) attached to or logically associated with an electronic record, and (3) made with the intent to sign the electronic record. Here we only discuss the first element. The U.S. definition of electronic signatures recognizes that there are many different methods by which one can sign an electronic record. Examples of electronic signatures that qualify under E-SIGN include:

1. a name typed at the end of an e-mail message by the sender,
2. a digitized image of a handwritten signature that is attached to an electronic document,
3. a secret password or PIN to identify the sender to the recipient,
4. a mouse click, such as on an "I accept" button,
5. a sound, such as the sound created by pressing '9' on a phone,
6. a cryptographic digital signature.

Clearly, the first five examples are easily forgeable and thus provide little means of identifying the signatory. However, recall that under U.S. law, signing a paper contract with an 'X' constitutes a binding signature, as long as one can establish intent of the signatory to sign the contract. Hence, the first five examples should be treated as the legal equivalent of signing with an 'X'.

United nations treaty on electronic signatures. In November 2005 the United Nations adopted its convention on the use of electronic communications in international contracts. The signature requirements of the 2005 U.N. convention go beyond those required under E-SIGN. In particular, the convention focuses on the issue of security, by requiring the use of a method that (1) identifies the signer, and (2) is reliable. In particular, the convention observes that there is a big difference between an electronic signature that merely satisfies the basic requirements of applicable U.S. law (e.g., a mouse click) and a trustworthy electronic signature. Thus, under the U.N. convention a mouse click qualifies as a digital signature only if it allows the proponent to ultimately prove "who" clicked, and to establish the intention behind the click.

European Community framework for electronic signatures. in December 1999, the European Parliament adopted the Electronic Signatures Directive. The directive addresses three forms of electronic signatures. The first can be as simple as signing an e-mail message with a person's name or using a PIN-code. The second is called the "advanced electronic signature" (AES). The directive is technology neutral but, in practice, AES refers mainly to a cryptographic digital signature based on a public key infrastructure (PKI). An AES is considered to be more secure, and thus enjoys greater legal acceptability. An electronic signature qualifies as an AES if it is: (1) uniquely linked to the signatory, (2) capable of identifying the signatory, (3) created using means that the signatory can maintain under his sole control, and (4) is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable.

13.10 A fun application: forward secure signatures

To be written.

13.11 Notes

Citations to the literature to be added.

13.12 Exercises

13.1 (Exercising the definition). Let (G, S, V) be a secure signature scheme with message space $\{0,1\}^n$. Generate two signing/verification key pairs $(\mathrm{pk}_0, \mathrm{sk}_0) \xleftarrow{R} G()$ and $(\mathrm{pk}_1, \mathrm{sk}_1) \xleftarrow{R} G()$. Which of the following are secure signature schemes? Show an attack or prove security.

(a) Accept one valid: $S_1\big((\mathrm{sk}_0, \mathrm{sk}_1),\ m\big) := \big(S(\mathrm{sk}_0, m),\ S(\mathrm{sk}_1, m)\big)$. Verify:

$$V_1\big((\mathrm{pk}_0, \mathrm{pk}_1),\ m,\ (\sigma_0, \sigma_1)\big) = \text{‘accept’} \quad \Longleftrightarrow$$

$$\big[V(\mathrm{pk}_0, m, \sigma_0) = \text{‘accept’} \quad \text{or} \quad V(\mathrm{pk}_1, m, \sigma_1) = \text{‘accept’}\big]$$

(b) Sign halves: $S_2\big((\mathrm{sk}_0, \mathrm{sk}_1),\ (m_{\mathrm{L}}, m_{\mathrm{R}})\big) := \big(S(\mathrm{sk}_0, m_{\mathrm{L}}),\ S(\mathrm{sk}_1, m_{\mathrm{R}})\big)$

$$V_2\big((\mathrm{pk}_0, \mathrm{pk}_1),\ (m_{\mathrm{L}}, m_{\mathrm{R}}), (\sigma_0, \sigma_1)\big) = \text{‘accept’} \quad \Longleftrightarrow$$

$$V(\mathrm{pk}_0, m_{\mathrm{L}},\ \sigma_0) = V(\mathrm{pk}_1, m_{\mathrm{R}},\ \sigma_1) = \text{‘accept’}$$

(c) Sign with randomness: for $m \in \{0,1\}^n$ do

$S_3\big(\mathrm{sk}_0,\ m\big) := \big[\text{choose random } r \leftarrow \{0,1\}^n,\ \text{ output } \big(r,\ \ S(\mathrm{sk}_0,\ m \oplus r),\ \ S(\mathrm{sk}_0,\ r)\big)\ \ \big].$

$V_3\big(\mathrm{pk}_0,\ m,\ (r, \sigma_0, \sigma_1)\big) = \text{‘accept’} \quad \Longleftrightarrow \quad V(\mathrm{pk}_0,\ m \oplus r,\ \sigma_0) = V(\mathrm{pk}_0,\ r,\ \sigma_1) = \text{‘accept’}$

13.2 (Multi-key signature security). Just as we did for secure MACs in Exercise 6.3, show that security in the single-key signature setting implies security in the multi-key signature setting.

(a) Show how to extend Attack Game 13.1 so that an attacker can submit signing queries with respect to several signing keys. This is analogous to the multi-key generalization described in Exercise 6.3.

(b) Show that every efficient adversary $\mathcal{A}$ that wins your multi-key attack game with probability ϵ can be transformed into an efficient adversary $\mathcal{B}$ that wins Attack Game 13.1 with probability ϵ/Q, where Q is the number of signature keys. The proof uses the same “plug-and-pray” technique as in Exercise 6.3.

13.3 (Non-binding signatures). In Section 13.1.1.1 we mentioned that secure signature schemes can be non-binding: for a given (pk, sk), the signer can find two distinct messages m_0 and m_1 where the same signature σ is valid for both messages with respect to pk. Give an example of a secure signature scheme that is non-binding.

Hint: Consider using the hash-and-sign paradigm of Section 13.2, but with the collision resistant hash functions discussed in Exercise 10.30.

13.4 (A signer confusion attack on RSA). Let us show that $\mathcal{S}_{\text{RSA-FDH}}$ is vulnerable to a signer confusion attack, as discussed in Section 13.1.1.1. Let (n, e) be Alice's public key and $\sigma \in Z_n$ be a signature on some message m. Then $\sigma^e = H(m)$ in $\mathbb{Z}_n$. Show that an efficient adversary can construct a new public key $pk' = (n', e')$, along with the corresponding secret key, such that (m, σ) is valid message-signature pair with respect to pk'.

Hint: We show in Section 16.1.2.2 that for some primes p, the discrete-log problem in $\mathbb{Z}_p^*$ can be solved efficiently. For example, when $p = 2^\ell + 1$ is prime, and ℓ is poly-bounded, the discrete-log problem in $\mathbb{Z}_p^*$ is easy. Show that by forming n' as a product of two such primes, the adversary can come up with an e' such that $\sigma^{(e')} = H(m)$ in $\mathbb{Z}_{n'}$.

13.5 (Strongly binding signatures). In this exercise we explore a general defense against message and signer confusion discussed in Section 13.1.1.1. To this end, we define the concept of a **strongly binding signature scheme**. For a given signature scheme $\mathcal{S} = (G, S, V)$, a strong binding adversary $\mathcal{A}$ takes no input and outputs (pk, m, pk', m', σ). We say that $\mathcal{A}$ *defeats strong binding* if

$$(pk, m) \neq (pk', m') \qquad \text{and} \qquad V(pk, m, \sigma) = V(pk', m', \sigma) = \mathsf{accept}.$$

Define $\mathcal{A}$'s *advantage* with respect to $\mathcal{S}$, denoted sbSIG$\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ defeats strong binding. We say that $\mathcal{S}$ is **strongly binding** if sbSIG$\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ is negligible for all efficient adversaries $\mathcal{A}$.

(a) Let us augment $\mathcal{S}$ so that the signer always attaches the public key pk to the message prior to signing, and the verifier does the same. That is, define $\mathcal{S}' = (G, S', V')$ where

$$S'(sk, m) := S\big(sk, (pk, m)\big) \qquad \text{and} \qquad V'(pk, m, \sigma) := V\big(pk, (pk, m), \sigma\big).$$

Here we are assuming for simplicity that pk can be easily derived from sk. Show that if we apply this augmentation to $\mathcal{S}_{\text{RSA-FDH}}$ from Section 13.3.1 and model the hash function H as a random oracle, then the augmented scheme is still secure and becomes strongly binding.

(b) Give an example of a secure signature scheme that is not strongly binding even after the augmentation from part (a) is applied.

(c) Let us look at a stronger augmentation that makes every signature scheme strongly binding. For a hash function H'', define $\mathcal{S}'' = (G, S'', V'')$ where

- $S''(sk, m) := \big\{h \leftarrow H''(pk, m), \quad \sigma \xleftarrow{\text{R}} S(sk, m), \quad \text{output } (h, \sigma)\big\}$, and
- $V''\big(pk, m, (h, \sigma)\big)$ accepts if $h = H''(pk, m)$ and $V(pk, m, \sigma) = \mathsf{accept}$.

Show that (i) if $\mathcal{S}$ is secure, then so is $\mathcal{S}''$, and (ii) if H'' is collision resistant, then $\mathcal{S}''$ is strongly binding.

Discussion: this augmentation results in a slightly longer signature, whereas the augmentation from part (a) does not affect signature length.

13.6 (Derandomizing signatures). Let $\mathcal{S} = (G, S, V)$ be a secure signature scheme defined over $(\mathcal{M}, \Sigma)$, where the signing algorithm S is probabilistic. In particular, algorithm S uses randomness chosen from a space $\mathcal{R}$. We let $S(sk, m; r)$ denote the execution of algorithm S with randomness r. Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{M}, \mathcal{R})$. Show that the following signature scheme $\mathcal{S}' = (G', S', V)$ is secure:

$$G'() := \big\{(pk, sk) \xleftarrow{\text{R}} G(), \quad k \xleftarrow{\text{R}} \mathcal{K}, \quad sk' := (sk, k), \quad \text{output } (pk, sk')\big\};$$
$$S'(sk', m) := \{r \leftarrow F(k, m), \quad \sigma \leftarrow S(sk, m; r), \quad \text{output } \sigma\}.$$

Now the signing algorithm for $\mathcal{S}'$ is deterministic.

13.7 (Extending the domain using enhanced TCR). In Exercise 8.27 we defined the notion of an enhanced-TCR. Show how to use an enhanced-TCR to efficiently extend the domain of a signature. In particular, let H be an enhanced-TCR defined over $(\mathcal{K}_H, \mathcal{M}, \mathcal{X})$ and let $\mathcal{S} = (G, S, V)$ be a secure signature scheme with message space $\mathcal{X}$. Show that $\mathcal{S}' = (G, S', V')$ is a secure signature scheme:

$$S'(pk, m) := \{\ r \xleftarrow{R} \mathcal{K}_H,\ \sigma \leftarrow S\big(sk, H(r, m)\big),\ \text{output } (\sigma, r)\}\ ;$$
$$V'\big(pk, m, (\sigma, r)\big) := \{\ \mathsf{accept} \text{ if } \sigma = V(pk, H(r, m))\}\ .$$

The benefit over the construction in Section 13.2.1 is that r is not part of the message given to the signing procedure.

13.8 (Selective security). Selective security is a weak notion of signature security, where the adversary has to commit ahead of time to the message m for which it will forge a signature. Let (G, S, V) be a signature scheme defined over $(\mathcal{M}, \Sigma)$. The selective security game begins with the adversary sending a message $m \in \mathcal{M}$ to the challenger. The challenger runs $(pk, sk) \xleftarrow{R} G()$ and sends pk to the adversary. The adversary then issues a sequence of signing queries $m_1, \ldots, m_Q$, as in Attack Game 13.1, where $m \neq m_i$ for all $i = 1, \ldots, Q$. The adversary wins if it can produce a valid signature on m, and the scheme (G, S, V) is **selectively secure** if no efficient adversary can win this game with non-negligible probability. Note that unlike Attack Game 13.1, here the adversary has to commit to the message m before it even sees the public key pk.

Now, for a hash function $H : \mathcal{M}' \to \mathcal{M}$, define a new signature scheme (G, S', V') as in (13.1). Show that if (G, S, V) is selectively secure, and H is modeled as a random oracle, then (G, S', V') is existentially unforgeable. In particular, for every existential forgery adversary $\mathcal{A}$ against $\mathcal{S}' = (G, S', V')$ there exists a selective forgery adversary $\mathcal{B}$ against $\mathcal{S} = (G, S, V)$ such that

$$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}'] \leq Q_{\mathrm{ro}} \cdot \mathrm{SEL}\mathsf{adv}[\mathcal{B}, \mathcal{S}] + Q_{\mathrm{s}}/|\mathcal{M}|,$$

where $\mathcal{A}$ makes at most Q_{ro} queries to H and at most Q_{s} signing queries. Here $\mathrm{SEL}\mathsf{adv}[\mathcal{B}, \mathcal{S}]$ is $\mathcal{B}$'s advantage in winning the selective security game against $\mathcal{S}$.

13.9 (FDH variant). Show that the signature scheme $\mathcal{S}'_{\text{RSA-FDH}}$ (defined in Section 13.5) is *no less secure* than the signature scheme $\mathcal{S}_{\text{RSA-FDH}}$ (defined in Section 13.3.1). You should show that if $\mathcal{A}$ is an adversary that succeeds with probability ϵ in breaking $\mathcal{S}'_{\text{RSA-FDH}}$ (which has message space $\mathcal{M}$), then there exists an adversary $\mathcal{B}$ (whose running time is roughly the same as that of $\mathcal{A}$) that succeeds with probability ϵ in breaking $\mathcal{S}_{\text{RSA-FDH}}$ (with message space $\mathcal{M}' = \{0, 1\} \times \mathcal{M}$). This should hold for any hash function H.

13.10 (Probabilistic full domain hash). Consider the following signature scheme $\mathcal{S} = (G, S, V)$ with message space $\mathcal{M}$, and using a hash function $H : \mathcal{M} \times \mathcal{R} \to \mathbb{Z}_n$:

$$G() := \{(n, d) \xleftarrow{R} \mathrm{RSAGen}(\ell, e), \quad pk := (n, e), \quad sk := (n, d), \quad \text{output } (pk, sk)\}\ ;$$
$$S(sk, m) := \Big\{ r \xleftarrow{R} \mathcal{R}, \quad y \leftarrow H(m, r), \quad \sigma \leftarrow y^d \in \mathbb{Z}_n, \quad \text{output } (\sigma, r) \Big\}\ ;$$
$$V\big(pk, m, (\sigma, r)\big) := \{y \leftarrow H(m, r), \quad \mathsf{accept} \text{ if } y = \sigma^e \text{ and } \mathsf{reject} \text{ otherwise}\}\ .$$

Show that this signature is secure if the RSA assumption holds for (ℓ, e), the quantity $1/|\mathcal{R}|$ is negligible, and H is modeled as a random oracle. Moreover, the reduction to inverting RSA is tight.

Discussion: While $\mathcal{S}'_{\text{RSA-FDH}}$, from Section 13.5, also has a tight reduction, the construction here does not use a PRF. The cost is that signatures are longer because r is included in the signature.

13.11 (Batch RSA). Let us show how to speed up signature generation in $\mathcal{S}_{\text{RSA-FDH}}$.

(a) Let $n = pq$ such that neither 3 nor 5 divide $(p-1)(q-1)$. We are given p, q and $y_1, y_2 \in \mathbb{Z}_n$. Show how to compute both $x_1 := y_1^{1/3} \in \mathbb{Z}_n$ and $x_2 := y_2^{1/5} \in \mathbb{Z}_n$ by just computing the 15th root of $t := (y_1)^5(y_2)^3 \in \mathbb{Z}_n$ and doing a bit of extra arithmetic. In other words, show that given $t^{1/15} \in \mathbb{Z}_n$, it is possible to compute both x_1 and x_2 using a constant number of arithmetic operations in $\mathbb{Z}_n$.

(b) Describe an algorithm for computing a 15th root in $\mathbb{Z}_n$ using a single exponentiation, for n as in part (a).

(c) Explain how to use parts (a) and (b) to speed up the $\mathcal{S}_{\text{RSA-FDH}}$ signature algorithm. Specifically, show that the signer can sign two messages at once using about the same work as signing a single message. The first message will be signed under the public key $(n, 3)$ and the other under the public key $(n, 5)$. This method generalizes to fast RSA signature generation in larger batches.

13.12 (Shortening RSA signatures). Let us see how to shorten an $\mathcal{S}_{\text{RSA-FDH}}$ signature by about one third when the public key is (n, e) with $e = 3$. We will need the following fact: for every $x \in \mathbb{Z}_n$ there exist integers $0 \leq r < n^{1/3}$ and $0 \leq s \leq n^{2/3}$ such that $x = (r/s \bmod n)$. These r and s can be efficiently found. We will use this fact to shorten an $\mathcal{S}_{\text{RSA-FDH}}$ signature. Let $pk = (n, 3)$ be an $\mathcal{S}_{\text{RSA-FDH}}$ public key. To sign a message m the signer does: (i) compute the $\mathcal{S}_{\text{RSA-FDH}}$ signature $\sigma := H(m)^{1/3} \in \mathbb{Z}_n$, (ii) find integers r and s so that $\sigma = (r/s \bmod n)$ where r and s satisfy the bounds from the fact, and (iii) output the shortened signature $\hat{\sigma} := s$. Note that $\hat{\sigma}$ is two thirds the length of σ.

(a) Show that the verifier can recover σ from $\hat{\sigma}$, n, and m. Hint: first show how to compute the integer r. Then $\sigma = (r/s \bmod n)$.

(b) Once the verifier recovers σ from $\hat{\sigma}$ it can run the standard $\mathcal{S}_{\text{RSA-FDH}}$ verification algorithm.

Show that using $\hat{\sigma}$ as the signature does not harm security by showing that an adversary that breaks security of the shortened scheme can be used to break security of $\mathcal{S}_{\text{RSA-FDH}}$.

13.13 (Signature with message recovery). Let $\mathcal{T} = (G, F, I)$ be a one-way trapdoor permutation defined over $\mathcal{X} := \{0,1\}^n$. Let $\mathcal{R} := \{0,1\}^\ell$ and $\mathcal{U} := \{0,1\}^{n-\ell}$, for some $0 < \ell < n$. Let H be a hash function defined over $(\mathcal{M} \times \mathcal{U},\ \mathcal{R})$, and let W be a hash function defined over $(\mathcal{R},\mathcal{U})$. Consider the following signature scheme $\mathcal{S} = (G, S, V)$ defined over $(\mathcal{M} \times \mathcal{U},\ \mathcal{X})$ where

$$S\big(sk,\ (m_0, m_1)\big) := \Big\{ h \leftarrow H(m_0, m_1), \quad \sigma \leftarrow I\big(sk,\ h \,\|\, (W(h) \oplus m_1)\big), \quad \text{output } \sigma \Big\}$$

(a) Explain how the verification algorithm works.

(b) Show that the scheme is secure assuming $\mathcal{T}$ is one-way, $1/|\mathcal{R}|$ is negligible, and H and W are modeled as random oracles.

(c) Show that just given (m_0, σ), where σ is a valid signature on the message (m_0, m_1), it is possible to recover m_1. A signature scheme that has this property is called a **signature with message recovery**. It lets the signer send shorter transmissions: the signer need only transmit (m_0, σ) and the recipient can recover m_1 by itself. This can somewhat mitigate the cost of long signatures with RSA.

(d) Can the technique of Section 13.5 be used to provide a tight security reduction for this construction?

13.14 (An insecure signature with message recovery). Let $\mathcal{T} = (G, F, I)$ be a one-way trapdoor permutation defined over $\mathcal{X} := \{0,1\}^n$. Let H be a hash function defined over $(\mathcal{M}_0, \mathcal{X})$. Consider the following signature scheme $\mathcal{S} = (G, S, V)$ defined over $(\mathcal{M}_0 \times \mathcal{X},\ \mathcal{X})$ where

$$\begin{aligned} S\big(sk,\ (m_0, m_1)\big) &:= \big\{\sigma \leftarrow I(sk,\ H(m_0) \oplus m_1), \quad \text{output } \sigma\big\} \\ V\big(pk,\ (m_0, m_1),\ \sigma\big) &:= \big\{y \leftarrow F(pk, \sigma), \quad \textsf{accept} \text{ if } y = H(m_0) \oplus m_1 \text{ and } \textsf{reject} \text{ otherwise}\big\} \end{aligned}$$

(a) Show that given (m_0, σ), where σ is a valid signature on the message (m_0, m_1), it is possible to recover m_1.

(b) Show that this signature scheme is insecure, even when $\mathcal{T}$ is one-way and H is modeled as a random oracle.

13.15 (Blind signatures). At the end of Section 13.3.1 we mentioned the RSA signatures can be adapted to give blind signatures. A **blind signature scheme** lets one party, Alice, obtain a signature on a message m from Bob, so that Bob learns nothing about m. Blind signatures are used in e-cash systems and anonymous voting systems.

Let $(n, d) \xleftarrow{R} \text{RSAGen}(\ell, e)$ and set (n, e) as Bob's RSA public key and (n, d) as his corresponding private key. As usual, let $H : \mathcal{M} \to \mathbb{Z}_n$ be a hash function. Alice wants Bob to sign a message $m \in \mathcal{M}$. They engage in the following three-message protocol:

(1) Alice chooses $r \xleftarrow{R} \mathbb{Z}_n$, sets $m' \leftarrow H(m) \cdot r^e \in \mathbb{Z}_n$, and sends m' to Bob,
(2) Bob computes $\sigma' \leftarrow (m')^d \in \mathbb{Z}_n$ and sends σ' to Alice,
(3) Alice computes the signature σ on m as $\sigma \leftarrow \sigma'/r \in \mathbb{Z}_n$.

Equation (13.4) shows that σ is a valid signature on m.

(a) Show that Bob sees a random message m' in $\mathbb{Z}_n$ that is independent of m. Therefore, he learns nothing about m.

(b) We say that a blind signature protocol is secure if the adversary, given a public key and the ability to request Q blind signatures on messages of his choice, cannot produce $Q + 1$ valid message-signature pairs. Write out the precise definition of security.

(c) Show that security of the RSA blind signature scheme follows from an assumption called the **one more RSA** or **1MRSA** assumption, if we model H as a random oracle.

The 1MRSA assumption is the RSA analogue of the 1MDH assumption defined in Section 11.6.3. It is defined using the following game. The challenger runs $(n, d) \xleftarrow{R} \text{RSAGen}(\ell, e)$ and sends (n, e) to the adversary. Next, the adversary makes a sequence of queries to the challenger, each of which can be one of the following types:

Challenge query: the challenger chooses $v \xleftarrow{R} \mathbb{Z}_n$, and sends v to the adversary.

RSA solve query: the adversary submits $\hat{v} \in \mathbb{Z}_n$ to the challenger, who computes and sends $\hat{w} \leftarrow \hat{v}^{1/e} \in \mathbb{Z}_n$ to the adversary.

At the end of the game, the adversary outputs a list of distinct pairs, each of the form (i, w), where i is a positive integer bounded by the number of challenge queries, and $w \in \mathbb{Z}_n$. We call such a pair (i, w) *correct* if $w = v^e$ in $\mathbb{Z}_n$ and v is the challenger's response to the ith challenge query. We say the adversary wins the game if the number of correct pairs exceeds the number of RSA solve queries. The 1MRSA assumption says that every efficient challenger wins this game with at most negligible probability. In particular, an efficient adversary who sees the e-th root of a number of challenges cannot produce the e-th root of any other challenge.

13.16 (Insecure signcryption). Let $\mathcal{E} = (G_{\mathrm{E}}, E, D)$ be a CCA-secure public-key encryption scheme with associated data and let $\mathcal{S} = (G_{\mathrm{S}}, S, V)$ be a strongly secure signature scheme. Define algorithm G as in Section 13.7.3. Show that the following encrypt-then-sign signcryption scheme (G, E', D') is insecure:

$$\begin{aligned}
E'(sk_{\mathrm{S}}, id_{\mathrm{S}}, pk_{\mathrm{R}}, id_{\mathrm{R}},\ m) \quad &:= \quad c \xleftarrow{\mathrm{R}} E\big(pk_{\mathrm{R}},\ m,\ id_{\mathrm{R}}\big), \quad \sigma \xleftarrow{\mathrm{R}} S\big(sk_{\mathrm{S}},\ (c, id_{\mathrm{S}})\big)\\
& \quad \text{output } (c, \sigma)\\
D'\big(pk_{\mathrm{S}}, id_{\mathrm{S}}, sk_{\mathrm{R}}, id_{\mathrm{R}},\ (c, \sigma)\big) \quad &:= \quad \text{if } V(pk_{\mathrm{S}},\ (c, id_{\mathrm{S}}),\ \sigma) = \mathsf{reject}, \text{ output } \mathsf{reject}\\
& \quad \text{otherwise, output } D(sk_{\mathrm{R}},\ c,\ id_{\mathrm{R}})
\end{aligned}$$

13.17 (The iMessage attack). Let $\mathcal{E} = (G_{\mathrm{E}}, E, D)$ be a CCA-secure public-key encryption scheme and let $\mathcal{S} = (G_{\mathrm{S}}, S, V)$ be a strongly secure signature scheme. Let $(E_{\mathrm{sym}}, D_{\mathrm{sym}})$ be a symmetric cipher with key space $\mathcal{K}$ that implements deterministic counter mode. Define algorithm G as in Section 13.7.3. Consider the following encrypt-then-sign signcryption scheme (G, E', D'):

$$\begin{aligned}
E'(sk_{\mathrm{S}}, id_{\mathrm{S}}, pk_{\mathrm{R}}, id_{\mathrm{R}},\ m) \quad &:= \quad k \xleftarrow{\mathrm{R}} \mathcal{K}, \quad c_1 \leftarrow E_{\mathrm{sym}}\big(k,\ (id_{\mathrm{S}}, m)\big), \quad c_0 \xleftarrow{\mathrm{R}} E(pk_{\mathrm{R}},\ k)\\
& \quad \sigma \xleftarrow{\mathrm{R}} S\big(sk_{\mathrm{S}},\ (c_0, c_1, id_{\mathrm{R}})\big)\\
& \quad \text{output } (c_0, c_1, \sigma)\\
D'\big(pk_{\mathrm{S}}, id_{\mathrm{S}}, sk_{\mathrm{R}}, id_{\mathrm{R}},\ (c_0, c_1, \sigma)\big) \quad &:= \quad \text{if } V(pk_{\mathrm{S}},\ (c_0, c_1, id_{\mathrm{R}}),\ \sigma) = \mathsf{reject}, \text{ output } \mathsf{reject}\\
& \quad k \leftarrow D(sk_{\mathrm{R}},\ c_0), \quad (id, m) \leftarrow D_{\mathrm{sym}}(k, c_1)\\
& \quad \text{if } id \neq id_{\mathrm{S}} \text{ output } \mathsf{reject}\\
& \quad \text{otherwise, output } m
\end{aligned}$$

Because the symmetric ciphertext c_1 is part of the data being signed by the sender, the designers assumed that there is no need to use an AE cipher and that deterministic counter mode is sufficient. Show that this system is an insecure signcryption scheme by giving a CCA attack. At one point, a variant of this scheme was used by Apple's iMessage system and this lead to a significant breach of iMessage [69]. Because every plaintext message m included a checksum (CRC), an adversary could decrypt arbitrary encrypted messages using a chopchop-like attack (Exercise 9.5).

13.18 (Signcryption: statically vs adaptively chosen user IDs). In the discussion following Definition 13.8, we briefly discussed the possibility of a more robust security definition in which the adversary is allowed to choose the sender and receiver user IDs adaptively, after seeing one or both of the public keys, or even after seeing the response to one or more X → Y queries.

(a) Work out the details of this more robust definition, defining corresponding SCI and SCCA attack games.

(b) Give an example of a signcryption scheme that satisfies Definition 13.8 but does not satisfy your more robust definition. To this end, you should start with a scheme that satisfies Definition 13.8, and then "sabotage" the scheme somehow so that it still satisfies Definition 13.8, but no longer satisfies your more robust definition. You may make use of any other standard cryptographic primitives, as convenient.

13.19 (Signcryption: encrypt-and-sign-then-sign). In this exercise, we develop a variation on encrypt-then-sign called encrypt-and-sign-then-sign. As does the scheme $\mathcal{SC}_{\text{EtS}}$, this new scheme, denoted $\mathcal{SC}_{\text{EaStS}}$, makes use of a public-key encryption scheme with associated data $\mathcal{E} = (G_{\text{ENC}}, E, D)$, and a signature scheme $\mathcal{S} = (G_{\text{SIG}}, S, V)$. Key generation for $\mathcal{SC}_{\text{EaStS}}$ is identical to that in $\mathcal{SC}_{\text{EtS}}$. However, $\mathcal{SC}_{\text{EaStS}}$ makes use of another signature scheme $\mathcal{S}' = (G'_{\text{SIG}}, S', V')$. The encryption algorithm $E_{\text{EaStS}}(sk_{\text{S}}, id_{\text{S}}, pk_{\text{R}}, id_{\text{R}}, m)$ runs as follows:

$$\begin{aligned}&(pk', sk') \xleftarrow{\text{R}} G'_{\text{SIG}}, \quad c \xleftarrow{\text{R}} E(pk_{\text{R}}, m, pk'), \quad \sigma \xleftarrow{\text{R}} S(sk_{\text{S}}, pk'),\\ &\sigma' \xleftarrow{\text{R}} S'(sk', (c, \sigma, id_{\text{S}}, id_{\text{R}})), \quad \text{output } (pk', c, \sigma, \sigma')\end{aligned}$$

The decryption algorithm $D_{\text{EaStS}}(pk_{\text{S}}, id_{\text{S}}, sk_{\text{R}}, id_{\text{R}}, (pk', c, \sigma, \sigma'))$ runs as follows:

```
if $V(pk_{\text{S}}, pk', \sigma) = \mathsf{reject}$ or $V'(pk', (c, \sigma, id_{\text{S}}, id_{\text{R}}), \sigma') = \mathsf{reject}$
    then output $\mathsf{reject}$
    else  output $D(sk_{\text{R}}, c, pk')$
```

Here, the value ephemeral public verification key pk' is used as associated data for the encryption scheme $\mathcal{E}$.

Your task is to show that $\mathcal{SC}_{\text{EaStS}}$ is a secure signcryption scheme that provides both forward secrecy and non-repudiation, under the following assumptions:

(i) $\mathcal{E}$ is CCA secure;

(ii) $\mathcal{S}$ is secure (not necessarily strongly secure);

(iii) $\mathcal{S}'$ is strongly secure — in fact, it is sufficient to assume that $\mathcal{S}'$ is strongly secure against an adversary that makes at most one signing query in Attack Game 13.2 (we will see very efficient signature schemes that achieve this level of security in the next chapter).

Discussion: Note that we have to run the key generation algorithm $\mathcal{S}'$ every time we encrypt, thereby generating an ephemeral signing key that is only used to sign a single message. The fact that we only need security against 1-query adversaries means that it is possible to very efficiently implement $\mathcal{S}'$ under reasonable assumptions. This is the topic of the next chapter.

Another feature is that in algorithm E_{EaStS}, we can run algorithms E and S in parallel; moreover, we can even run algorithms G'_{SIG} and S before algorithm E_{EaStS} is invoked (as discussed in Section 14.5.1). Similarly, in algorithm D_{EaStS}, we can run algorithms V, V', and D in parallel.

13.20 (Verifiable random functions). A verifiable random function (VRF) is a PRF, with the additional property that anyone can verify that the PRF value at a given point is computed correctly. Specifically, a VRF defined over $(\mathcal{X}, \mathcal{Y})$ is a triple of efficient algorithms (G, F, V), where algorithm G outputs a public key pk and a secret key sk. Algorithm F is invoked as $(y, \pi) \leftarrow F(sk, x)$ where $x \in \mathcal{X}$, $y \in \mathcal{Y}$, and where π is called a validity proof. Algorithm V is invoked as $V(pk, x, y, \pi)$, and outputs either accept or reject. We say that y is the value of the VRF at the point x, and π is the validity proof for y. The VRF must satisfy the following two properties:

- Correctness: for all (pk, sk) output by G, and all $x \in \mathcal{X}$, if $(y, \pi) \leftarrow F(sk, x)$ then $V(pk, x, y, \pi) = \textsf{accept}$.

- Uniqueness: for all pk and every $x \in \mathcal{X}$, only a single $y \in \mathcal{Y}$ can have a valid proof π. More precisely, if $V(pk, x, y, \pi) = V(pk, x, y', \pi') = \textsf{accept}$ then $y = y'$. This ensures that even with the secret key, an adversary cannot lie about the value of the VRF at the point x.

VRF security is defined using two experiments, analogous to the characterization of a PRF given in Exercise 4.7. In both experiments, the challenger generates (pk, sk) using G, and gives pk to the adversary. The adversary then makes a number of function queries and a single test query (with any number of function queries before and after the test query). In a function query, the adversary submits $x \in \mathcal{X}$ and obtains $(y, \pi) \leftarrow F(sk, x)$. In the test query, the adversary submits $\tilde{x} \in \mathcal{X}$: in one experiment, he obtains $\tilde{y}$, where $(\tilde{y}, \tilde{\pi}) \leftarrow F(sk, \tilde{x})$; in the other experiment, he obtains a random $\tilde{y} \in \mathcal{Y}$. The test point $\tilde{x}$ is not allowed among the function queries. The VRF is secure if the adversary cannot distinguish the two experiments.

(a) Show that a secure VRF (G, F, V) defined over $(\mathcal{X}, \mathcal{Y})$ can be constructed from a unique signature scheme (G, S', V') with message space $\mathcal{X}$ (unique signatures were defined in Section 13.3). Try defining $F(sk, x)$ as follows: compute $\sigma \leftarrow S'(sk, x)$, and then output $y := H(\sigma)$ as the value of the VRF at x and $\pi := \sigma$ as the validity proof for y. Here H is a hash function that maps signatures to elements of $\mathcal{Y}$. Explain how the VRF algorithm V works, and prove security of the construction when H is modeled as a random oracle.

(b) Given a secure VRF (G, F, V) defined over $(\mathcal{X}, \mathcal{Y})$, where $|\mathcal{Y}|$ is super-poly, show how to construct a secure signature scheme with message space $\mathcal{X}$.

Discussion: Another VRF scheme is presented in Exercise 20.16. To see why VRFs are useful, let's see how they can be used to convince a verifier that a ciphertext in a symmetric cipher is decrypted correctly. Let (G, F, V) be a secure VRF defined over $(\mathcal{X}, \mathcal{Y})$ where $\mathcal{Y} := \{0,1\}^n$, for some n. Consider the symmetric cipher (E, D) with message space $\mathcal{Y}$ where encryption is defined as

$$E(sk, m) := \big\{ r \xleftarrow{\text{R}} \mathcal{X}, \quad (y, \pi) \leftarrow F(sk, r), \quad c \leftarrow m \oplus y, \quad \text{output } (r, c) \big\}.$$

$D\big(sk,\ (r, c)\big)$ is defined analogously. Now, let (r, c) be a ciphertext and let m be its alleged decryption. Using the VRF property, it is easy to convince anyone that m is the correct decryption of (r, c), without revealing anything else. Simply give the verifier the proof π that $m \oplus c$ is the value of the VRF at the point r.

Chapter 14

Fast hash-based signatures

In the previous chapter we presented a number of signature schemes built from a trapdoor permutation like RSA. In this chapter we return to more basic primitives, and construct signature schemes from one-way and collision resistant hash functions. The resulting signatures, called **hash-based signatures**, can be much faster to generate and verify than RSA signatures. An important feature of hash-based signatures is that, with suitable parameters, they are secure against an adversary who has access to a quantum computer. The RSA trapdoor permutation is insecure against such attacks, as explained in Section 16.5. The post-quantum security of hash-based signatures drives much of the interest in these schemes. We will therefore use post-quantum security parameters to evaluate their performance.

The drawback of hash-based signature schemes is that the signatures themselves are much longer than RSA signatures. As such, they are well suited for applications like signing a software update where signature size is not important because the data being signed is quite large to begin with. They are not ideal for signing Web certificates where short signatures are needed to reduce network traffic.

We begin by constructing hash-based **one-time signatures**, where a key pair (pk, sk) can be used to securely sign a *single* message. Security can break down completely if (pk, sk) is used to sign multiple messages. More generally, we define a **q-time signature**, where a key pair (pk, sk) can be used to securely sign q messages, for some small q. In our context, q is typically rather small, say less than a hundred.

Definition 14.1. *We say that a signature system $\mathcal{S}$ is a **secure q-time signature** if for all efficient signature adversaries $\mathcal{A}$ that issue at most q signature queries, the value* $\mathsf{SIGadv}[\mathcal{A}, \mathcal{S}]$ *defined in Attack Game 13.1 is negligible. When $q = 1$ we say that $\mathcal{S}$ is a **secure one-time signature**.*

We shall first construct fast one-time signatures from one-way functions and then describe their many applications. In particular, we show how to construct a regular (many-time) signature scheme from a one-time signature. When using one-time signatures, one typically attaches the public-key to the signature. Therefore, we will usually aim to minimize the combined length of the public-key and the signature.

As we did in Section 13.1.1, we can define a stronger notion of security, where it is hard to come up with a signature on a new message, and it is also hard to come up with a new signature on a previously signed message.

Definition 14.2. *We say that a signature system $\mathcal{S}$ is a* ***strongly secure*** q***-time signature*** *if for all efficient signature adversaries $\mathcal{A}$ that issue at most q signature queries, the value* $\mathrm{stSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *defined in Attack Game 13.2 is negligible. When $q = 1$ we say that $\mathcal{S}$ is a* ***strongly secure one-time signature****.*

We shall explore this stronger notion in the exercises.

14.1 Basic Lamport signatures

In Section 8.11 we defined the notion of a one-way function. Let f be such a one-way function defined over $(\mathcal{X}, \mathcal{Y})$. We can use f to construct a simple one-time signature for signing a one-bit message $m \in \{0, 1\}$. Simply choose two random values x_0 and x_1 in $\mathcal{X}$ and set

$$pk := \big(f(x_0),\ f(x_1)\big) \quad ; \qquad sk := (x_0, x_1)$$

Write $pk = (y_0, y_1)$. To sign a one bit message $m \in \{0, 1\}$ output the signature $S(sk, m) := x_m$. Concretely, the signature on the message '0' is x_0 and the signature on the message '1' is x_1. To verify a signature σ on m simply check that $f(\sigma) = y_m$. We call this system $\mathcal{S}_{\mathrm{1bit}}$.

If f is a one-way function then an adversary cannot recover x_0 or x_1 from the public-key pk. Hence, just given pk, the adversary cannot forge a signature on either one of the two messages in $\{0, 1\}$. Similarly, the signature on a message $m \in \{0, 1\}$ does not help the adversary forge a signature on the complementary message $m \oplus 1$. Therefore, this simple signature scheme is a secure one-time signature, as summarized in the following theorem.

Theorem 14.1. *Let f be a one-way function over $(\mathcal{X}, \mathcal{Y})$. Then $\mathcal{S}_{\mathrm{1bit}}$ is a secure one-time signature for messages in $\{0, 1\}$.*

Proof. Let $\mathcal{A}$ be a one-time signature adversary that attacks $\mathcal{S}_{\mathrm{1bit}}$. The adversary asks for the signature on a message $b \in \{0, 1\}$ and outputs the signature on the message $1 - b$. Then by Lemma 13.5, using $t = 2$, there exists an algorithm $\mathcal{B}$ for inverting f that satisfies:

$$\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{1bit}}] \leq 2 \cdot \mathrm{OW}\mathsf{adv}[\mathcal{B}, f] \quad \Box$$

Basic Lamport signatures. Extending the idea above lets us build a one-time signature for 256-bit messages, which is sufficient for signing an arbitrary long message as discussed in Section 13.2. More generally, to sign a v-bit message we simply repeat the one-time one-bit signature above v times. The resulting signature system, called the **basic Lamport signature system** $\mathcal{S}_{\mathrm{L}} = (G, S, V)$, is defined as follows (see Fig. 14.1):

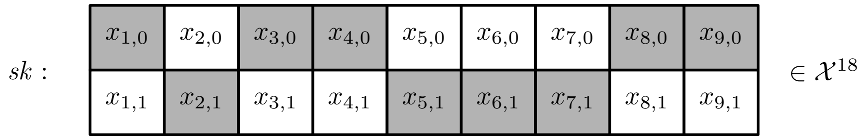
- Algorithm G outputs a public-key $pk \in \mathcal{Y}^{2v}$ and secret key $sk \in \mathcal{X}^{2v}$ as follows:

  choose $2v$ random values: $\begin{pmatrix} x_{1,0}, & \dots, & x_{v,0} \\ x_{1,1}, & \dots, & x_{v,1} \end{pmatrix} \xleftarrow{\mathrm{R}} \mathcal{X}^{2v}$

  for $i = 1, \dots, v$ and $j = 0, 1$ do: $\quad y_{i,j} \leftarrow f(x_{i,j})$

  output:

  $$sk := \begin{pmatrix} x_{1,0}, & \dots, & x_{v,0} \\ x_{1,1}, & \dots, & x_{v,1} \end{pmatrix} \in \mathcal{X}^{2v} \qquad \text{and} \qquad pk := \begin{pmatrix} y_{1,0}, & \dots, & y_{v,0} \\ y_{1,1}, & \dots, & y_{v,1} \end{pmatrix} \in \mathcal{Y}^{2v}$$



sk :	$x_{1,0}$	$x_{2,0}$	$x_{3,0}$	$x_{4,0}$	$x_{5,0}$	$x_{6,0}$	$x_{7,0}$	$x_{8,0}$	$x_{9,0}$	$\in \mathcal{X}^{18}$
	$x_{1,1}$	$x_{2,1}$	$x_{3,1}$	$x_{4,1}$	$x_{5,1}$	$x_{6,1}$	$x_{7,1}$	$x_{8,1}$	$x_{9,1}$	

Lamport signature on a message $m = 010011100$ consists of all shaded squares

Figure 14.1: Lamport signatures: an example

- Algorithm $S(sk, m)$, where $m = m_1 \dots m_v \in \{0,1\}^v$, outputs the signature:

$$\sigma := (x_{1,m_1},\ x_{2,m_2},\ \dots,\ x_{v,m_v}) \in \mathcal{Y}^v$$

- Algorithm $V(pk, m, \sigma)$ where $m \in \{0,1\}^v$ and $\sigma = (\sigma_1, \dots, \sigma_v) \in \mathcal{X}^v$ outputs

$$\begin{cases} \textsf{accept} & \text{if } f(\sigma_i) = y_{i,m_i} \text{ for all } i = 1, \dots, v \\ \textsf{reject} & \text{otherwise} \end{cases}$$

Signature generation takes no work at all. The signer simply reveals certain values already in its possession. Verifying a signature takes v evaluations of the function f.

The proof of security for this system follows from Theorem 14.2 below where we prove security of a more general system. Alternatively, one can view this v-bit system as v independent instances of the one-bit system discussed in Theorem 14.1. Security of the v-bit system is then an immediate corollary of multi-key security discussed in Exercise 13.2.

Shrinking the secret-key. Because the secret key is just a sequence of random elements in $\mathcal{X}$, it can be generated using a secure PRG. The signer keeps the short PRG seed as the secret key and nothing else. It evaluates the PRG when signing a message and outputs the appropriate elements as the signature. This shrinks the size of the secret key to a single PRG seed, but at the cost of slightly increasing the work to sign messages. If ultra fast signing is needed, this optimization can be ignored.

Shrinking the public-key. The size of the public-key in the basic Lamport scheme is quite large, but can be made short at the cost of increasing the signature length. We do so using a generic transformation described in Exercise 14.1 that shows that the public-key in every signature scheme can be made short.

14.1.1 Shrinking the signature using an enhanced TCR

The length of a Lamport signature is linear in the length v of the message being signed. So far we assumed $v = 256$ bits which is the output length of SHA256. We can reduce v using the ideas developed in Exercise 8.27, where we showed how an enhanced TCR hash function can be used in place of a collision resistant hash function. This lets us halve the hash length v without hurting security. Shrinking v this way will approximately halve the size of the Lamport signature.

For completeness, we briefly present the resulting signature scheme (G, S', V'), which we call **randomized Lamport**. Let H_{etcr} be an enhanced TCR function defined over $(\mathcal{R},\ \mathcal{M},\ \{0,1\}^v)$. Here $\mathcal{R}$ is a nonce space and $\mathcal{M}$ is (possibly much larger) message space. Algorithm G is unchanged from the basic Lamport scheme $\mathcal{S}_{\text{L}} = (G, S, V)$. Algorithm S' and V' work as follows:

- $S'(sk, M)$: given $M \in \mathcal{M}$ as input, do:

  $$r \xleftarrow{\text{R}} \mathcal{R}, \quad m \leftarrow H_{\text{etcr}}(r, M), \quad \sigma' \leftarrow S(sk, m), \quad \text{output } \sigma := (r, \sigma').$$

- $V'\big(pk,\ M,\ (r, \sigma')\big)$: given $M \in \mathcal{M}$ and (r, σ') as input, do:

  $$m \leftarrow H_{\text{etcr}}(r, M), \quad \text{output } V(pk, m, \sigma').$$

The same argument as in Exercise 8.27 shows that this construction is secure as long as the basic Lamport signature scheme is secure and H_{etcr} is an enhanced TCR. Moreover, suppose we want the adversary to make at least 2^{128} evaluations of H_{etcr} to win the enhanced TCR game with advantage $1/2$. Then part (b) of Exercise 8.27 shows that it suffices to take $v = 130$ instead of $v = 256$. This approximately halves the size of the Lamport signature. The signature includes the random nonce r, but this nonce can be short, only about the size of a single element in $\mathcal{X}$.

Post-quantum security. In Section 4.3.4 we discussed quantum exhaustive search attacks. These attacks show that a quantum adversary can win the enhanced TCR game for a v-bit hash function in time $2^{v/2}$. Therefore, for post-quantum security we must use $v = 256$ even when using an enhanced TCR. For this reason, we will evaluate all the schemes in this chapter using $v = 256$. Of course, if one is only concerned with classical adversaries then $v = 130$ is sufficient.

14.2 A general Lamport framework

Our description of the basic Lamport signature, while simple, is not optimal. We can further shrink the signature size by quite a lot. To do so, we first develop a general framework for Lamport-like signatures. This framework reduces the security of Lamport signatures to an elegant combinatorial property that will let us build better one-time and q-time signatures.

As in the previous section, let f be a one-way function over $(\mathcal{X}, \mathcal{Y})$. We wish to sign messages in $\mathcal{M} := \{0,1\}^v$ for some fixed v. As usual, this lets us sign arbitrary length messages by first hashing the given message using a collision resistant function or an enhanced TCR. The general Lamport framework works as follows:

- A secret key is n random values $x_1, \ldots, x_n \in \mathcal{X}$ for some n that will be determined later. The public-key consists of the n hashes $y_i := f(x_i)$ for $i = 1, \ldots, n$.

- To sign a message $m \in \mathcal{M}$ we use a special function P that maps m to a subset of $\{1, \ldots, n\}$. We will see examples of such P in just a minute. To sign m we first compute $P(m)$ to obtain a subset $s \leftarrow P(m) \subseteq \{1, \ldots, n\}$. The signature is just the subset of preimages $\sigma := \{x_i\}_{i \in s}$.

- To verify a signature σ on a message m the verifier checks that σ contains the pre-images of all public-key values $\{y_i\}_{i \in P(m)}$.

As in the basic Lamport scheme, the signer need not store a large secret key $sk := (x_1, \ldots, x_n)$. Instead, he keeps a single PRF key $sk = (k)$ and generates the x_i as $x_i \leftarrow F(k, i)$. All we need is a secure PRF defined over $(\mathcal{K},\ \{1, \ldots, n\},\ \mathcal{X})$. In more detail, the generalized Lamport system $\mathcal{S}_P = (G, S, V)$ works as follows:

```
Algorithm $G()$:
$k \xleftarrow{R} \mathcal{K}$
for $i = 1, \ldots, n$:
    $x_i \leftarrow F(k, i) \in \mathcal{X}$
    $y_i \leftarrow f(x_i) \in \mathcal{Y}$
output:
    $pk = (y_1, \ldots, y_n)$
    $sk = (k)$
```

```
Algorithm $S(sk, m)$:
$s \leftarrow P(m) \subseteq \{1, \ldots, n\}$
let $s := \{s_1, \ldots, s_\ell\}$
for $j = 1, \ldots, \ell$:
    $\sigma_j \leftarrow F(k, s_j)$
output:
    $\sigma \leftarrow (\sigma_1, \ldots, \sigma_\ell)$
```

```
Algorithm $V(pk, m, \sigma)$:
let $P(m) = \{s_1, \ldots, s_\ell\}$
let $\sigma := (\sigma_1, \ldots, \sigma_u)$
if $\ell = u$ and $f(\sigma_i) = y_{s_i}$
    for all $i = 1, \ldots, \ell$
then output accept
otherwise output reject
```

Now that we understand the general framework, the question is how to choose the function P. Specifically, for what functions P is this a secure one-time signature scheme? The adversary sees the signature on a single message $m \in \mathcal{M}$ of his choice, and wants to forge a signature on some other message $m' \in \mathcal{M}$. Clearly, if the set $P(m')$ is contained in the set $P(m)$ then the signature on m also gives a signature on m'. Hence, for security we must insist that it be difficult for the adversary to find distinct messages m and m' such that $P(m)$ contains $P(m')$. For now we focus on functions where such containment is not possible, no matter how powerful the adversary is.

Definition 14.3. *We say that a function P from $\mathcal{M}$ to subsets of $\{1, \ldots, n\}$ is* ***containment free*** *if for all distinct messages $m, m' \in \mathcal{M}$ the set $P(m)$ is not contained in the set $P(m')$.*

Containment free functions are easy to build: take P to be an injective function that always outputs subsets of a fixed size ℓ. Clearly a subset of size ℓ cannot contain another subset of size ℓ and hence such a P is containment free. The basic Lamport system $\mathcal{S}_{\mathrm{L}}$ of Section 14.1 is a special case of this general framework. It uses $n = 2v$ and a containment free function P that always outputs subsets of size v.

The following theorem shows that every containment free P gives a secure one-time signature system. Security of the basic Lamport signature system follows as a special case.

Theorem 14.2. *Suppose f is a one-way hash over $(\mathcal{X}, \mathcal{Y})$ and F is a secure PRF defined over $(\mathcal{K},\ \{1, \ldots, n\},\ \mathcal{X})$. Let P be a containment free function from $\mathcal{M}$ to subsets of $\{1, \ldots, n\}$. Then $\mathcal{S}_P$ is a secure one-time signature for messages in $\mathcal{M}$.*

In particular, suppose $\mathcal{A}$ is a signature adversary attacking $\mathcal{S}_P$ that issues at most one signature query. Then there exist an efficient adversary $\mathcal{B}_f$ attacking the one-wayness of f, and a PRF adversary $\mathcal{B}_F$, where $\mathcal{B}_f$ and $\mathcal{B}_F$ are elementary wrappers around $\mathcal{A}$, such that

$$\mathrm{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}_P] \leq n \cdot \mathrm{OW}\mathsf{adv}[\mathcal{B}_f, f] + \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F] \tag{14.1}$$

Proof idea. The proof shows that $\mathcal{A}$ can be used to solve the repeated one-way problem for f as defined in Section 13.4.1. We construct an adversary $\mathcal{B}$ that uses $\mathcal{A}$ to win the repeated one-way game. We then use Lemma 13.5 with $t = n$ to convert $\mathcal{B}$ into an algorithm for breaking the one-wayness of f. This is the source of the factor of n in (14.1).

The repeated one-way game starts with the repeated one-way challenger $\mathbf{C}$ giving $\mathcal{B}$ a list of n elements $y_1, \ldots, y_n \in \mathcal{Y}$. $\mathcal{B}$ needs to invert one of them. It runs $\mathcal{A}$ and does the following:

- $\mathcal{B}$ sends $(y_1, \ldots, y_n)$ as the public-key to $\mathcal{A}$. Since F is a secure PRF, this public-key is indistinguishable from a public-key generated by $G()$.

- $\mathcal{A}$ requests the signature on some message m_1. Our $\mathcal{B}$ requests from $\mathbf{C}$ the preimages of all the y_i where $i \in P(m_1)$, and sends these pre-images as the signature to $\mathcal{A}$.

- Finally, $\mathcal{A}$ outputs a forgery σ for some message $m \neq m_1$. Since P is containment free we know that $P(m) \setminus P(m_1)$ is not empty and hence there exists some j in $P(m) \setminus P(m_1)$. If σ is a valid signature on m then σ contains a pre-image $x_j \in \mathcal{X}$ of $y_j \in \mathcal{Y}$. Our $\mathcal{B}$ outputs (j, x_j) as its solution to the repeated one-way problem.

Since $j \notin P(m_1)$ we know that $\mathcal{B}$ never requested a pre-image for y_j. Hence (j, x_j) is a valid solution to the repeated one-way problem. The theorem now follows from Lemma 13.5. □

q-time signatures. The general containment free framework presented here directly extends to give q-time signatures for small q. The only difference is that the function P must satisfy a stronger property called q-containment freeness. We explore this extension in Exercise 14.5.

14.2.1 An explicit containment free function

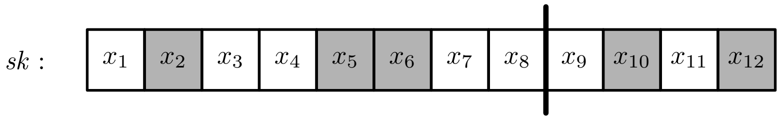
When using a one-time signature we often aim to minimize the total combined length of the public-key and the signature. In the general Lamport framework, this amounts to setting n to be the smallest value for which there is an efficiently computable function from $\mathcal{M} := \{0,1\}^v$ to subsets of $\{1, \ldots, n\}$ that is containment free. One can show (using Sperner's theorem) that the smallest possible n is about $n_{min} := v + (\log_2 v)/2$. Recall that the basic Lamport system uses $n = 2v$, which is almost twice as big as this lower bound.

We present an efficient containment free function P_{opt} that uses $n := v + 1 + \lceil \log_2 v \rceil$, which is close to the optimal value of n. For simplicity, let us assume that v is a power of 2. Recall that the weight of a bit string $m \in \{0,1\}^v$ is the number of bits in m that are set to 1. The function P_{opt} is defined as follows:

```
input: $m \in \{0,1\}^v$
output: $P_{\text{opt}}(m) \subseteq \{1, \ldots, n\}$
$P_{\text{opt}}(m) :=$    $c \leftarrow v - \text{weight}(m)$    // $c \in [0, v]$ is the number of 0s in m
                 encode $c$ as a binary string in $\{0,1\}^{(\log_2 v)+1}$
                 $m' \leftarrow m \parallel c \quad \in \{0,1\}^n$    // $c$ is called a checksum
                 output the set $\{i \text{ s.t. } m'_i = 1\} \subseteq \{1, \ldots, n\}$    // here $m' = m'_1 \ldots m'_n$
```

Fig. 14.2 gives an example. The function is clearly injective: if $P_{\text{opt}}(m_0) = P_{\text{opt}}(m_1)$ then $m_0 = m_1$. The following lemma shows that it is also containment free.

Lemma 14.3. *For every distinct $m_0, m_1 \in \{0,1\}^v$ we have that $P_{\text{opt}}(m_0) \not\subseteq P_{\text{opt}}(m_1)$.*



For $m = 01001100$ we have checksum = 0101. The signature consists of all shaded squares.

Figure 14.2: Optimized Lamport signatures: an example

Proof. Let m_0, m_1 be distinct messages and let c_0, c_1 be the checksums for m_0, m_1 respectively as defined in algorithm P_{opt}. Suppose $P_{\text{opt}}(m_0) \subseteq P_{\text{opt}}(m_1)$. Then clearly m_0 contains fewer 1 bits than m_1 implying that $c_0 > c_1$. But if $c_0 > c_1$ then there must exist some bit in the binary representations of c_0 and c_1 that is 0 in c_1 but 1 in c_0. This bit implies that $P_{\text{opt}}(m_0) \not\subseteq P_{\text{opt}}(m_1)$ as required. □

Fig. 14.2 shows the resulting optimized Lamport system in action. Since P_{opt} is containment free, Theorem 14.2 shows that the resulting signature system is a secure one-time signature system.

Concrete parameters. The public-key length is $|pk| = v + 1 + \log_2 v$ elements in $\mathcal{Y}$. The expected length of a signature for a random messages $m \in \{0,1\}^v$ is about $|pk|/2 \approx v/2$ elements in $\mathcal{X}$. Thus, in the optimized Lamport system, both the public-key and the signature are about half the length of those in the basic Lamport system $\mathcal{S}_{\text{L}}$. Using Exercise 14.1, the total combined size of the public-key and the signature is v elements in $\mathcal{X} \cup \mathcal{Y}$ plus one hash.

Concretely, for post-quantum security one typically takes $\mathcal{X} = \mathcal{Y} = \{0,1\}^{256}$ and $v = 256$. With these parameters, the combined size of the public-key and the signature is about 8.5 KB. In the next two sections we show how to greatly reduce this size.

14.3 Winternitz one-time signatures

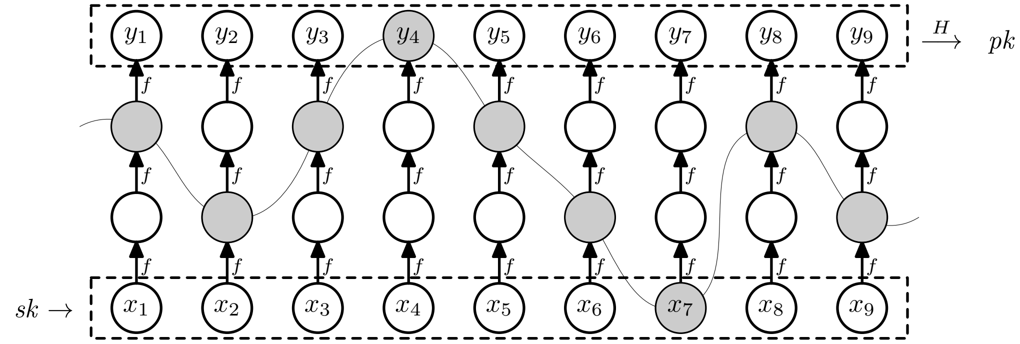
We next present a beautiful generalization of the Lamport framework that dramatically shrinks the signature and public-key size. But there is no free lunch. This improvement comes at the cost of more work to generate and verify signatures. We begin by defining the notion of a hash chain.

Hash chains. Let $f : \mathcal{X} \to \mathcal{X}$ be a function. For a non-negative integer j we let $f^{(j)}(x)$ denote the ***j*th iterate of** f, namely $f^{(j)}(x) := f(f(f(\cdots(x)\cdots)))$ where f is repeated j times. For example,

$$f^{(0)}(x) := x \ ; \quad f^{(1)}(x) := f(x) \ ; \quad f^{(2)}(x) := f(f(x)) \ ; \quad f^{(3)}(x) := f(f(f(x)))$$

and so on. For $x \in \mathcal{X}$ the sequence $f^{(0)}(x), f^{(1)}(x), \ldots, f^{(d)}(x)$ is called a **hash chain** of length $d + 1$. The value x is called the base of the chain and $y := f^{(d)}(x)$ is called its top.

The Winternitz scheme. We wish to sign messages in $\mathcal{M} := \{0,1\}^v$ for some fixed v using a one-way function f defined over $(\mathcal{X}, \mathcal{X})$. The scheme operates as follows (see also Fig. 14.3):



The secret key sk is used to derive $x_1, \ldots, x_9 \in \mathcal{X}$. The public key pk is the hash of $y_1, \ldots, y_9 \in \mathcal{X}$. The shaded circles represent the signature on a message m where $P(m) = (2, 1, 2, 3, 2, 1, 0, 2, 1)$. This $P(m)$ describes a cut through the rectangle illustrated by the thin line.

Figure 14.3: Winternitz signatures with $n = 9$ and $d = 3$

- Fix parameters n and d that will be determined later. A secret key is n random values $x_1, \ldots, x_n \xleftarrow{R} \mathcal{X}$. The public-key consists of the n hashes $y_i := f^{(d)}(x_i)$ for $i = 1, \ldots, n$.

  As before, we can compress the secret key by generating $(x_1, \ldots, x_n)$ using a PRG defined over $(\mathcal{S}, \mathcal{X}^n)$. Then the actual secret key is just a short seed in $\mathcal{S}$. Similarly, we can compress the public-key by only publishing a short collision resistant hash of the vector $(y_1, \ldots, y_n)$, as shown in Fig. 14.3.

- To sign a message $m \in \mathcal{M}$ we use a special function P that maps m to a vector s of length n. Every component of s is a number in $\{0, \ldots, d\}$. More precisely, let $I_d^n := (\{0, \ldots, d\})^n$. Then P is a function $P : \mathcal{M} \to I_d^n$.

  To sign m we first compute $s \leftarrow P(m)$. Let $s = (s_1, \ldots, s_n) \in I_d^n$. Then the signature is the vector $\sigma := \big(f^{(s_1)}(x_1), \ldots, f^{(s_n)}(x_n)\big) \in \mathcal{X}^n$, as illustrated by the shaded circles in Fig. 14.3. The signature corresponds to a cut through the rectangle in the figure, represented by the thin line through the shaded circles.

- To verify a signature $\sigma = (\sigma_1, \ldots, \sigma_n) \in \mathcal{X}^n$ on a message m first compute $P(m) = (s_1, \ldots, s_n) \in I_d^n$. Next, compute the vector

  $$\hat{y} := \Big(f^{(d-s_1)}(\sigma_1), \ldots, f^{(d-s_n)}(\sigma_n)\Big) \in \mathcal{X}^n.$$

  The signature σ is valid only if $\hat{y}$ is equal to the public-key vector $(y_1, \ldots, y_n)$.

In more detail, the Winternitz scheme $\mathcal{S}_{\text{win}}$, parameterized by n and d, works as follows. Here we use a PRG G_{prg} defined over $(\mathcal{S}, \mathcal{X}^n)$ and a collision resistant hash function $H : \mathcal{X}^n \to \mathcal{T}$.

```
Algorithm $G()$:
$k \xleftarrow{R} \mathcal{S}$
$(x_1, \ldots, x_n) \leftarrow G_{\mathrm{prg}}(k)$
for $i = 1, \ldots, n$:
    $y_i \leftarrow f^{(d)}(x_i)$
output:
    $pk := H(y_1, \ldots, y_n)$
    $sk := (k)$
```

```
Algorithm $S(sk, m)$:
$(x_1, \ldots, x_n) \leftarrow G_{\mathrm{prg}}(k)$
$s \leftarrow P(m) \in I_d^n$
let $s = (s_1, \ldots, s_n)$
for $i = 1, \ldots, n$:
    $\sigma_i \leftarrow f^{(s_i)}(x_i)$
output:
    $\sigma \leftarrow (\sigma_1, \ldots, \sigma_n)$
```

```
Algorithm $V(pk, m, \sigma)$:
let $P(m) = (s_1, \ldots, s_n)$
let $\sigma = (\sigma_1, \ldots, \sigma_n)$
for $i = 1, \ldots, n$:
    $\hat{y}_i \leftarrow f^{(d-s_i)}(\sigma_i)$
$\hat{y} \leftarrow (\hat{y}_1, \ldots, \hat{y}_n)$
if $H(\hat{y}) = pk$ output accept
otherwise output reject
```

This scheme is a generalization of the general Lamport framework. Specifically, when $d = 1$ the scheme is equivalent to the Lamport framework.

Security. For what functions P is this system a secure one-time signature? Suppose the adversary finds two messages $m, m' \in \mathcal{M}$ such that every entry in the vector $P(m')$ is greater than or equal to the corresponding entry in $P(m)$. We say that $P(m')$ dominates $P(m)$. Any such pair can be used to forge signatures: given a signature on m the adversary can compute everything needed for a signature on m'. For example, the signature on the message m in Fig. 14.3 can be used to derive a signature on a message m' where $P(m') = (2, 2, 2, 3, 2, 2, 2, 2, 2)$.

Hence, at a minimum we must insist that it be difficult for the adversary to find distinct messages m and m' such that $P(m')$ dominates $P(m)$. This motivates the following definition:

Definition 14.4. *Let s, s' be vectors in I_n^d. We say that s'* ***dominates*** *s if $s'_i \geq s_i$ for all $i = 1, \ldots, n$. We say that a function $P : \mathcal{M} \to I_n^d$ is* ***domination free*** *if for all distinct messages $m, m' \in \mathcal{M}$ the vector $P(m')$ does not dominate $P(m)$.*

Visually, P is domination free if for every pair of messages m, m' in $\mathcal{M}$, the cuts corresponding to $P(m)$ and $P(m')$ cross at at least one point. We will construct such a function P after we prove security of the signature scheme.

The security analysis of Winternitz requires that $f : \mathcal{X} \to \mathcal{X}$ be a strong one-way function in the following sense: we say that f is **one-way on d iterates** if for all $j = 1, \ldots, d$, it is hard to find an f-inverse of $f^{(j)}(x)$, where $x \xleftarrow{R} \mathcal{X}$. We capture this property in the following game:

Attack Game 14.1 (One-way on d iterates). For a given function $f : \mathcal{X} \to \mathcal{X}$ and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The adversary chooses $j \in \{1, \ldots, d\}$ and sends j to the challenger.
- The challenger computes $x \xleftarrow{R} \mathcal{X}$ and $y \leftarrow f^{(j)}(x)$, and sends y to $\mathcal{A}$.
- The adversary outputs $x' \in \mathcal{X}$.

We say $\mathcal{A}$ wins the game if $f(x') = y$. We define the adversary's advantage $\mathrm{iOW}\mathsf{adv}[\mathcal{A}, f, d]$ to be the probability that it wins. □

Definition 14.5. *For an integer $d > 0$, we say that $f : \mathcal{X} \to \mathcal{X}$ is* ***one-way on d iterates*** *if $\mathrm{iOW}\mathsf{adv}[\mathcal{A}, f, d]$ is negligible for all efficient adversaries $\mathcal{A}$.*

Exercise 14.16 shows that a one-way function f need not be one-way on d iterates, even when $d = 2$. Nevertheless, standard cryptographic functions such as SHA256 are believed to be one-way on d iterates for reasonable values of d, say $d \leq 10^6$. This strong one-way property holds if f is a random oracle and $|\mathcal{X}|$ is large, as discussed in Exercise 14.16.

Armed with the definition of one-way on iterates and domination free functions, we can now state the security of Winternitz signatures.

Theorem 14.4. *Let f be a one-way function on d iterates defined over $(\mathcal{X}, \mathcal{X})$. Let G_{prg} be a secure PRG over $(\mathcal{S}, \mathcal{X}^n)$, let H be collision resistant over $(\mathcal{X}^n, \mathcal{T})$, and let $P : \mathcal{M} \to I_n^d$ be domination free. Then the Winternitz scheme $\mathcal{S}_{\text{win}}$ is a secure one-time signature for messages in $\mathcal{M}$.*

> *In particular, suppose $\mathcal{A}$ is a signature adversary attacking $\mathcal{S}_{\text{win}}$ that issues at most one signature query. Then there exist efficient adversaries $\mathcal{B}_f, \mathcal{B}_G, \mathcal{B}_H$, where all three are elementary wrappers around $\mathcal{A}$, such that*

$$\text{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{win}}] \leq nd \cdot \text{iOW}\mathsf{adv}[\mathcal{B}_f, f, d] + \text{PRG}\mathsf{adv}[\mathcal{B}_G, G_{\text{prg}}] + \text{CR}\mathsf{adv}[\mathcal{B}_H, H] \tag{14.2}$$

Proof idea. The proof proceeds along the same lines as the proof of Theorem 14.2. The main difference is that we need to generalize Lemma 13.5 so that it applies to iterates of a one-way function. We explore this generalization in Exercise 14.15. The bound in Exercise 14.15 is the source of the factor nd in (14.2). The rest of the proof is essentially as in Theorem 14.2. □

14.3.1 A domination free function for Winternitz signatures

It remains to provide a domination free function $P : \mathcal{M} \to I_n^d$ for parameters n and d. When $|\mathcal{M}| = 2^v$ we describe a construction that satisfies

$$n \approx v / \log_2(d+1). \tag{14.3}$$

Taking, for example, $d = 15$ gives $n \approx (v/4) + 1$. Since a Winternitz signature contains n elements in $\mathcal{X}$, this leads to a *fourfold* reduction in combined signature and public-key length compared to the optimized Lamport signature (Section 14.2.1). That signature corresponds to setting $d = 1$.

To be fair, this reduction in length comes at the expense of verification time. When $d = 15$ signature verification requires $8n$ evaluations of the one-way function on average. Note that $8n$ is approximately $2v$. In comparison, optimized Lamport requires only about $v/2$ evaluations on average. Hence, verification is about four times slower.

Concretely, when $v = 256$ and $\mathcal{X} = \{0,1\}^{256}$, the function P described below provides the following combined signature and public-key size for different values of d:

d:	1	3	15	1023
minimum n:	265	133	67	28
combined size (KB):	8.5	4.2	2.1	0.9

A domination free function P. We describe the function $P : \{0,1\}^v \to I_d^{n_0}$ as a generalization of the containment free function P_{opt} from Section 14.2.1. Fix d and let n_0 be the smallest integer such that $2^v \leq (d+1)^{n_0}$. If we treat the message $m \in \{0,1\}^v$ as an integer in $[0, 2^v)$, we can write m in base $(d+1)$ and obtain a vector of digits $(s_1, \ldots, s_{n_0})$ in $I_d^{n_0}$ (possibly with leading zeros). When $d+1$ is a power of two this is done by simply partitioning $m \in \{0,1\}^v$ into consecutive blocks of $\log_2(d+1)$ bits.

Next, set $n_1 := \lceil \log_{d+1}(dn_0) \rceil + 1$ and $n := n_0 + n_1$. One can verify that indeed n is about $v/\log_2(d+1)$ as promised in (14.3). Now, using d, n_0, n_1, the function P works as follows:

```
input: $m \in \{0,1\}^v$
output: $(s_1, \ldots, s_n) \in I_n^d$
$P(m) :=$     write $m$ as an $n_0$-digit number in base $(d+1)$: $(s_1, \ldots, s_{n_0}) \in I_d^{n_0}$
              $c \leftarrow dn_0 - (s_1 + \cdots + s_{n_0})$   //  $c \in [0, dn_0]$ is called a checksum
              write $c$ as an $n_1$-digit number in base $(d+1)$:   $c = (c_1, \ldots, c_{n_1}) \in I_d^{n_1}$
              $m' \leftarrow (\ s_1, \ldots, s_{n_0},\ c_1, \ldots, c_{n_1}\ )\quad \in I_d^n$
              output $m'$
```

When $d = 1$ this function is equivalent to the function P_{opt}. The following lemma shows that it is domination free. This completes our description of Winternitz signatures.

Lemma 14.5. *For every distinct $m_0, m_1 \in \{0,1\}^v$ we have that $P(m_0)$ does not dominate $P(m_1)$.*

Proof. Let m_0, m_1 be distinct messages and let c_0, c_1 be the checksums for m_0, m_1 respectively, as defined in algorithm P. Because P is injective, $P(m_0) \neq P(m_1)$. Suppose $P(m_1)$ dominates $P(m_0)$. Then clearly $c_1 < c_0$. But if $c_1 < c_0$ there must exist some digit in their $(d+1)$-ary representations that is smaller in c_1 than in c_0. This digit implies that $P(m_1)$ does not dominate $P(m_0)$, as required. □

14.4 HORS: short Lamport signatures

Our final Lamport variation shows how to shrink the signature without increasing verification time. This expands the public-key, but we then show how to shrink the public-key using a Merkle tree (Section 8.9).

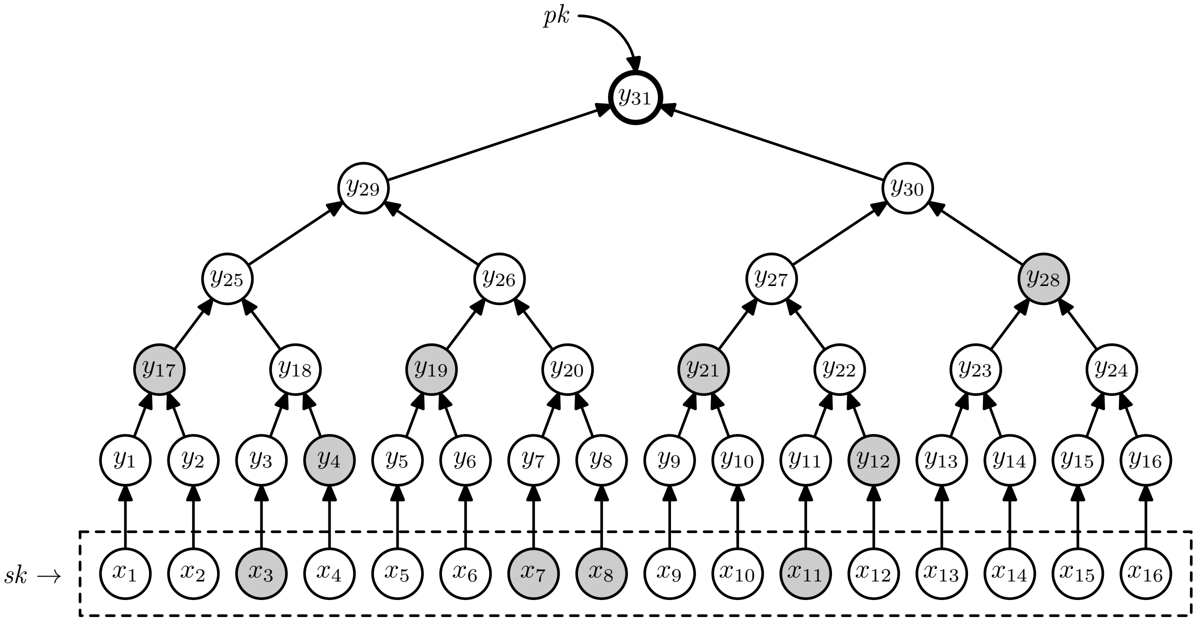
Let $\text{Sets}[n, \ell]$ denote the set of all subsets of $\{1, \ldots, n\}$ of size ℓ. This set contains $\binom{n}{\ell}$ elements. Suppose we had an injective and efficiently computable function $P_{\text{hors}} : \{0,1\}^v \to \text{Sets}[n, \ell]$ for some parameters n and ℓ. Such a function is containment free, and can therefore be used in the general Lamport one-time signature framework (Section 14.2) to sign messages in $\{0,1\}^v$. The resulting signature scheme is called **hash to obtain a random subset** or simply HORS.

We show in Exercise 14.3 how to construct the function P_{hors} for every choice of sufficiently large parameters n and ℓ. Exercise 14.4 gives another approach.

Concrete parameters. Because the function P_{hors} is injective, it must be the case that its range is at least as large as its domain. In other words, we must choose the parameters n and ℓ so that $\binom{n}{\ell} \geq 2^v$. When $v = 256$ some viable options for n and ℓ that satisfy $\binom{n}{\ell} \geq 2^v$ are as follows:

pk size:	n	512	1024	2048	8192
min signature size:	ℓ	58	44	36	27

In particular, when the public-key contains $n = 1024$ elements of $\mathcal{Y}$, the signature need only contain $\ell = 44$ elements of $\mathcal{X}$. This is far shorter than the optimized Lamport signature (Section 14.2.1) and verification time is much faster than with Winternitz signatures of comparable size. This comes at the expense of a large public-key, which we address next.



The HORST system with $n = 16$ and $\ell = 4$. When $P_{\text{hors}}(M) = \{3, 7, 8, 11\}$ the signature is the set of shaded nodes. The secret key *sk* is a short PRF key from which $x_1, \ldots, x_{16} \in \mathcal{X}$ are derived.

Figure 14.4: HORST signature: an example

14.4.1 Shrinking the public-key using a Merkle tree

For many applications of one-time signatures one wants to minimize the total combined size of the public-key and the signature. As specified above, the public-key consists of n elements in $\mathcal{Y}$ and the signature consists of ℓ elements in $\mathcal{X}$. This can be reduced significantly using the Merkle Tree technique of Section 8.9. Let H be a hash function from $\widehat{\mathcal{Y}}^2$ to $\widehat{\mathcal{Y}}$ and let us assume that $\mathcal{Y}$ is a subset of $\widehat{\mathcal{Y}}$. At key generation time, algorithm G places all the $y_1, \ldots, y_n \in \mathcal{Y}$ at the leaves of a Merkle tree and sets the public-key *pk* to be the root of the Merkle tree after iteratively hashing the leaves using H. The public-key *pk* is then a single element in $\widehat{\mathcal{Y}}$. Signatures produced by this method include the t pre-image values in $\mathcal{X}$ plus proofs that the corresponding y values are in the Merkle tree.

This signature scheme is called **HORS tree**, or simply HORST. An example of the system in action is shown in Fig. 14.4.

In Section 8.9 we showed that ℓ proofs in a Merkle tree with n leaves require at most $\ell \log_2(n/\ell)$ tree nodes. Hence, the total combined length of the signature and public-key is ℓ elements in $\mathcal{X}$ and $1 + \ell \log_2(n/\ell)$ elements in $\widehat{\mathcal{Y}}$. Since $\ell \log_2(n/\ell)$ is often smaller than n this Merkle technique results in significant savings over the HORS method.

Concretely, the combined public-key and signature size is only a small improvement over the Lamport scheme (Section 14.2.1). The improvement becomes substantial when we consider q-time signatures for small q. Exercise 14.6 shows how HORST gives an efficient q-time signature scheme.

14.5 Applications of one-time signatures

One-time signatures constructed from one-way functions can be much faster than RSA signatures. Their speed makes them useful for several applications. We give two examples here.

14.5.1 Online/offline signatures from one-time signatures

Let us see how to speed up signature generation in all (many-time) signature schemes. The idea is to split up the signing algorithm S in to two phases. The bulk of the signing work is done before the message to be signed is known. We call this the **offline phase**. Then, once a message m is given we quickly output a signature on m. We call this the **online phase**. Our goal is minimize the work in the online phase.

Using one-time signatures, we can easily modify any signature system so that the online work is fast. The idea is as follows: in the offline phase we generate an ephemeral key pair (pk_1, sk_1) for the one-time signature system and sign pk_1 using the long-term signing key. Then, when the message m is given, we quickly sign m using the one-time signature. Thus, the online signing work is just the time to sign m using a one-time system.

More precisely, let $\mathcal{S}_\infty = (G_\infty, S_\infty, V_\infty)$ be a long-term signature system such as $\mathcal{S}_{\text{RSA-FDH}}$. Let $\mathcal{S}_1 = (G_1, S_1, V_1)$ be a fast one-time signature system. Define a hybrid signature system $\mathcal{S} = (G, S, V)$ as follows:

- G runs G_∞ to obtain a key pair (pk_∞, sk_∞).
- $S(sk_\infty, m)$ works as follows:

```
1.   $(pk_1, sk_1) \xleftarrow{R} G_1()$           // Generate a one-time key pair
2.   $\sigma_0 \xleftarrow{R} S_\infty(sk_\infty,\ pk_1)$     // Sign the one-time public-key

3.   $\sigma_1 \xleftarrow{R} S_1(sk_1,\ m)$           // Once $m$ is known, sign $m$ using the one-time system
4.   output $\sigma := (pk_1, \sigma_0, \sigma_1)$
```

- $V(pk_\infty, m, \sigma)$ parses σ as $\sigma := (pk_1, \sigma_0, \sigma_1)$ and outputs accept only if:

$$V_\infty(pk_\infty,\ pk_1, \sigma_0) = \mathsf{accept} \quad \text{and} \quad V_1(pk_1,\ m, \sigma_1) = \mathsf{accept}$$

The bulk of the signing work, Steps 1 and 2, takes place before the message m is known. Step 3 used to sign m is as fast as generating a one-time signature.

A real-world application for online/offline signatures comes up in the context of web authentication at a large web site. Users who want to login to the site are first redirected to a login server. The login server asks for a username/password and then, after successful authentication, signs a special token that is sent to the user's web browser. This signed token then gives the user access to systems at the web site (perhaps only for a bounded amount of time).

At a large site the login server must sign hundreds of millions of tokens per day. But demand for these signed tokens is not uniform. It peaks at some hours of the day and ebbs at other times. During low usage times the login server can spend the time to generate many pairs (pk_1, σ_0). Then at peak times, the server can use these pairs to quickly sign actual tokens. Overall, the online/offline mechanism allows the login server to balance out demand for computing cycles throughout the day.

14.5.2 Authenticating streamed data with one-time signatures

Consider a radio transmission streamed over the Internet. The signal is sent as a stream of packets over the network. The radio station wants to authenticate the stream so that each recipient can verify that the transmission is from the station. This prevents intermediaries from messing with the broadcast, for example, replacing the station's ads with their own ads.

Recipients want to play the packets as they are received. One option is for the radio station to sign every packet using its long term signing key, but this will be quite slow. Can we can do better? Again, we can speed things up using one-time signatures. The idea is to amortize the cost of a single expensive RSA signature over many packets.

Let $\mathcal{S}_\infty = (G_\infty, S_\infty, V_\infty)$ be a long-term signature system such as $\mathcal{S}_{\text{RSA-FDH}}$. Let $\mathcal{S}_1 = (G_1, S_1, V_1)$ be a fast one-time signature system. The radio station already has a long term key pair (pk_0, sk_0) generated using G_∞. It generates a chain of one-time key pairs $\{(pk_i, sk_i)\}$ for $i = 1, \ldots, \ell$. Then key sk_i will be used to authenticate both pk_{i+1} and packet number i. More precisely, the station does the following:

```
input: $(pk_0, sk_0)$ and packets $m_0, m_1, \ldots$
    $(pk_1, sk_1) \xleftarrow{R} G_1()$
    $\sigma_0 \xleftarrow{R} S_\infty(sk,\ (m_0, pk_1))$   // sign the first one-time key using the long-term key
    send $(m_0, pk_1, \sigma_0)$
    For $i = 1, 2, \ldots$ do:
        $(pk_{i+1}, sk_{i+1}) \xleftarrow{R} G_1()$
        $\sigma_i \xleftarrow{R} S_1(sk_i,\ (m_i, pk_{i+1})\ )$   // sign key $pk_{i+1}$ using $sk_i$
        Send $(m_i, pk_{i+1}, \sigma_i)$
```

The recipient verifies this stream by using the public-key in packet i to verify packet $i+1$, starting with the first packet. Overall, the station signs the first one-time key using the slow long-term signature and signs the remaining keys using a fast one-time signature. Thus, the cost of the slow signatures is amortizes across many packets. Note also that no buffering of packets at either the sender or the receiver is needed.

Of course, this approach adds additional network traffic to send the sequence of public keys to the recipients. It should only be used in settings where the additional traffic is cheaper than signing every packet with the long-term key.

14.6 From one-time signatures to many-time signatures

We now turn to constructing a many-time signature scheme from a one-time signature. This will show that a many-time signature scheme can be built with nothing more than one-way and collision resistant functions. The resulting scheme is post-quantum secure. Here we focus on building **stateless signatures**. That is, the signer does not maintain any internal state between invocations of the signing algorithm. Stateless signatures are much easier to use than stateful ones, especially in a distributed environment where many machines issue signatures using the same secret key.

14.6.1 Indexed signatures

We will need a simple variation of q-time signatures. A **q-indexed signature** is a q-time signature scheme $\mathcal{S} = (S, G, V)$ where the message space is $\mathcal{M}' := \{1, \ldots, q\} \times \mathcal{M}$. We also require that the signing algorithm S be deterministic. We show in Exercise 13.6 that the signing algorithm of any signature scheme can be easily de-randomized using a secure PRF, so this does not limit the choice of signature scheme.

Security of a q-indexed signature is defined using the standard signature attack game (Attack Game 13.1) with one restriction — the adversary can issue up to q signature queries for messages (u_i, m_i), but $u_1, \ldots, u_q$ must all be distinct. In other words, once the adversary issues a signature query for (u, m) no other signature query can use the same u. As usual, the adversary wins this game if it is able to produce an existential forgery for $\mathcal{S}$, namely a valid message-signature pair $((u, m),\ \sigma)$ for some new message (u, m). We let $\mathrm{iSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ denote $\mathcal{A}$'s advantage in winning this game.

Definition 14.6. *A **q-indexed signature system** is a signature system $\mathcal{S} = (G, S, V)$ where the message space is $\mathcal{M}' = \{1, \ldots, q\} \times \mathcal{M}$ and the signing algorithm S is deterministic. We say that $\mathcal{S}$ is a secure q-indexed signature if for all efficient q-query signature adversaries $\mathcal{A}$, the quantity $\mathrm{iSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ is negligible.*

Any one-time signature gives a q-indexed signature. Let $\mathcal{S}_1 = (G_1, S_1, V_1)$ be a one-time signature. The derived q-indexed signature $\mathcal{S} = (G, S, V)$ works by generating q one-time public/private key pairs and signing a message (u, m) using key number u. More precisely, algorithms (G, S, V) work as follows:

```
Algorithm $G()$:
For $i = 1, \ldots, q$ :
    $(pk_i, sk_i) \xleftarrow{R} G_1()$
Output:
    $pk = (pk_1, \ldots, pk_q)$
    $sk = (sk_1, \ldots, sk_q)$
```

$$
\begin{aligned}
&S\big(sk,\ (u,m)\big) := S_1(sk_u,\ m) \\
&V\big(sk,\ (u,m), \sigma\big) := V_1(pk_u,\ m,\ \sigma)
\end{aligned}
\tag{14.4}
$$

Security of this construction follows immediately from the security of the underlying one-time signature. The proof of security uses the same "plug-and-pray" argument as in Exercise 13.2.

Shrinking the public-key. The size of the public-key in the brute-force construction (14.4) is linear in q. This can be greatly reduced using the Merkle tree approach we used in Fig. 14.4 to shrink a HORS public-key. Place the q one-time public keys at the leaves of a Merkle tree and compute the corresponding hash at the root. This single hash value at the root is the public key for the q-indexed scheme. Exercise 14.19 shows how to efficiently compute it.

A signature on a message (u, m) contains the Merkle proof needed to authenticate the one-time public key pk_u, along with the one-time signature on m using sk_u. Signature size is then

$$T + t \cdot \log_2 q \tag{14.5}$$

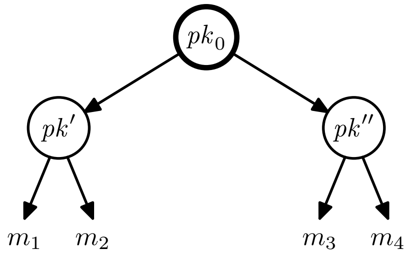


Figure 14.5: Using a 2-indexed signature to sign four messages

where T is the combined length of a single one-time signature and a single one-time public-key, and t is the output size of the hash function H used in the Merkle tree.

14.6.2 A many-time signature scheme from an indexed signature

Let $\mathcal{S} = (G_q, S_q, V_q)$ be a q-indexed signature. We build a many-time signature system $\mathcal{S}_{\text{Merkle}} = (G, S, V)$. The system uses an implicit q-ary tree of depth d. Internal tree nodes contain public-keys generated by $G_q()$. Messages to be signed are placed at the leaves of this tree. Each leaf is used to sign at most one message enabling us, in principal, to sign up to q^d messages.

Let $(pk_0, sk_0) \overset{\text{R}}{\leftarrow} G_q()$. To keep things simple for now, let us assume $q = 2$ so that the key sk_0 is only good for signing two messages. We let pk_0 be the public-key. Fig. 14.5 shows how to amplify this system to sign *four* messages. First we generate two more key pairs (pk', sk'), (pk'', sk'') and sign pk' and pk'' with sk_0:

$$\sigma' \leftarrow S_q\big(sk_0,\ (1, pk')\ \big) \quad \text{and} \quad \sigma'' \leftarrow S_q\big(sk_0,\ (2, pk'')\ \big)$$

The pairs (pk', σ') and (pk'', σ'') prove that pk' and pk'' were certified by sk_0. Now, sk' and sk'' can each sign two messages giving a total of four messages that can be signed. For example, the signature on m_2 is:

$$\Big((pk', \sigma'),\quad S_q\big(sk',\ (2, m_2)\big)\Big)$$

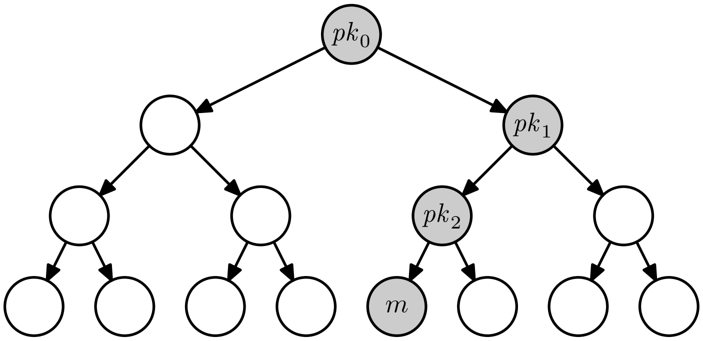
To verify the signature, first check that pk' is properly signed with respect to the public-key pk_0. Second, check that m_2 is properly signed with respect to pk'. If both sub-signatures verify then the signature is said to be valid.

We can repeat this process to obtain greater amplification — pk' and pk'' can each sign two new public-keys to obtain a total of four certified public-keys. Each of these in turn can sign two messages, thus enabling us to sign a total of eight messages. By repeating this process d times we increase the number of messages that can be signed to 2^d.

Fig. 14.6 illustrates this idea (for $q = 2$) using a tree of depth $d = 3$. The public-key pk_0 is generated by $G_q()$ and lives at the root of the tree. The secret key is sk_0. To sign a message m do:

1. First, pick a random leaf. In Fig. 14.6 we use leaf $(2, 1, 1)$ — namely we go right from the root and then left twice.

2. Next, generate two public/private key pairs (pk_1, sk_1) and (pk_2, sk_2) using $G_q()$ for internal nodes on the path. Every node on the path signs its child and the location of the child. The last node pk_2 signs the message, as shown on the right of Fig. 14.6.



$\sigma_1 \leftarrow S(sk_0,\ (2, pk_1)\ )$

$\sigma_2 \leftarrow S(sk_1,\ (1, pk_2)\ )$

$\sigma_3 \leftarrow S(sk_2,\ (1, m)\ \ \ )$

Figure 14.6: Merkle signatures with a tree of depth $d = 3$

The final signature is $\Big((2, 1, 1),\ (pk_1, \sigma_1),\ (pk_2, \sigma_2),\ \sigma_3\Big)$ which includes the intermediate public-keys and signatures as well as the location of the leaf $(2, 1, 1)$. To verify this signature simply check that all sub-signatures in this tuple are valid.

The key management problem. For this system to be secure it is essential that once the signer generates a public/private key pair for an internal node, that same key pair is used for all future signatures — we cannot ever generate a new key pair for that internal node. To see why, consider an internal node just below the root. Suppose that when signing message m the signer generates a key pair (pk_1, sk_1) for the left child and then signs $(1, pk_1)$ with sk_0, as required. Later, when signing message $m' \neq m$ the signer generates a new pair (pk'_1, sk'_1) for that node and signs $(1, pk'_1)$ with sk_0. An observer in this case sees signatures for both $(1, pk_1)$ and $(1, pk'_1)$ under sk_0, which can completely compromise security of the underlying q-indexed signature. In fact, when building a 2-indexed signature from Lamport one-time signatures, such usage will result in an insecure system. Hence, key pairs in this tree, once generated, must be kept forever.

For exactly the same reason, every leaf node can only be used to sign a single message — using a leaf to sign two distinct messages would completely compromise security of the q-indexed private key at the parent of that leaf.

To make the signature stateless, we make the signer pick a random leaf for every message and hope that he never picks the same leaf twice. For this to work, the number of leaf nodes must be large, say 2^{160}, so that the probability of a collision after issuing many signatures is small. But then the number of internal nodes is large and we cannot possibly store all internal key pairs in the tree. Again, since the signature is stateless we cannot generate internal key pairs "on the fly" and then store them for future invocations of the signing algorithm. Fortunately, this key management problem has a simple and elegant solution.

Generating internal keys using a PRF. To address the key management problem raised in the previous paragraph, our plan to is to generate all internal key pairs using a secure PRF. Consider a q-ary tree of depth d. Every node in the tree is identified by the path from the root to that node. That is, a node v at depth e is identified by a vector $(a_1, \ldots, a_e) \in \{1, \ldots, q\}^e$. This vector indicates that v is child number a_e of its parent, the parent is child number a_{e-1} of its parent, and so on all the way to the root. We refer to $(a_1, \ldots, a_e)$ as the ID of node v.

Let F be a PRF that maps node IDs in $\{1, \ldots, q\}^{\leq d}$ to bit strings in $\{0,1\}^w$ for some w. The output of F will be used as the random bits given to algorithm G_q. Therefore, we need w to be greater than the maximum number of random bits consumed by G_q. We will write $G_q(r)$, where $r \in \{0,1\}^w$, to denote the output of G_q using random bits r. Clearly once the bits r are specified, algorithm $G_q(r)$ is deterministic. The PRF F assigns a public/private key pair to every internal node in the q-ary tree. The key pair at node $\vec{a} := (a_1, \ldots, a_e) \in \{1, \ldots, q\}^{\leq d}$ is simply $(pk_{\vec{a}}, sk_{\vec{a}}) := G_q(F(k, \vec{a}))$ where k is the PRF secret key and $1 \leq e \leq d$.

Recall that we required the q-indexed signing algorithm to be deterministic. Hence, signing the same message twice with the same private key always results in the same signature. This is needed so that every time we sign an internal node, the resulting signature is identical to the one obtained during prior invocations of the signing algorithm. If this were not the case, then an observer who sees multiple Merkle signatures would obtain more than q distinct signatures for a particular internal public-key.

14.6.3 The complete Merkle stateless signature system

Let (G_q, S_q, V_q) be a q-indexed signature and let F be a PRF defined over $(\mathcal{K},\ \{1, \ldots, q\}^{\leq d},\ \{0,1\}^w)$. We use F to assign key pairs to internal tree nodes as discussed in the preceding paragraph. The Merkle signature $\mathcal{S}_{\text{Merkle}} = (G, S, V)$ system works as follows. To generate (pk, sk) algorithm G does:

```
Algorithm $G()$ :  $k \xleftarrow{R} \mathcal{K}$   // Pick a random PRF key
                   $(pk_0, sk_0) \xleftarrow{R} G_q()$
                   output $sk \leftarrow (k, sk_0)$ and $pk \leftarrow pk_0$
```

The signature generation and signature verification algorithms are described in Fig. 14.7.

Security. Next we turn to proving that this construction is secure assuming the underlying q-indexed signature is secure. Suppose we use a tree of depth d and use the system $\mathcal{S}_{\text{Merkle}}$ to generate a total of Q signatures. We show that the signature is secure as long as $Q^2/(2q^d)$ is negligible.

Theorem 14.6. *Let d, q be poly-bounded integers such that q^d is super-poly. Let $\mathcal{S}_q$ be a secure q-indexed signature. Then the derived Merkle signature $\mathcal{S}_{\text{Merkle}}$ is a secure signature.*

> *In particular, suppose $\mathcal{A}$ is a Q-query signature adversary attacking $\mathcal{S}_{\text{Merkle}}$. Then there exist an efficient q-query adversary $\mathcal{B}$ and a PRF adversary $\mathcal{B}_F$, where $\mathcal{B}$ and $\mathcal{B}_F$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + Qd \cdot \text{iSIG}\mathsf{adv}[\mathcal{B}, \mathcal{S}_q] + \frac{Q^2}{2q^d}$$

Proof idea. As usual, we first replace the PRF F with a random function. Now the Merkle signature system $\mathcal{S}_{\text{Merkle}}$ contains q^d independent instances of the $\mathcal{S}_q$ system. The adversary $\mathcal{A}$ issues at most Q queries for $\mathcal{S}_{\text{Merkle}}$ signatures. Each signature uses d instances of $\mathcal{S}_q$. Hence, throughout the game $\mathcal{A}$ interacts with at most Qd instances of $\mathcal{S}_q$. Let $\ell := Qd$.

We construct adversary $\mathcal{B}$ to break $\mathcal{S}_q$ using a basic "plug-and-pray" argument. $\mathcal{B}$ is given a $\mathcal{S}_q$ public-key pk and its goal is to forge a pk signature. It starts by generating $\ell = Qd$ public/private key pairs of $\mathcal{S}_q$ denoted $pk_0, \ldots, pk_{\ell-1}$. It then replaces one of these public-keys by the challenge

```
Algorithm $S(sk, m)$: where $sk = (k, sk_0)$

// Choose a random leaf node:
$\vec{a} := (a_1, \ldots, a_d) \xleftarrow{R} \big(\{1, \ldots, q\}\big)^d$

// Sign public-keys along path to leaf
For $i = 1$ to $d - 1$:
    $r_i \leftarrow F(k,\ (a_1, \ldots, a_i)\ )$
    $(pk_i, sk_i) \leftarrow G_q(r_i)$
    $\sigma_i \xleftarrow{R} S_q(sk_{i-1},\ (a_i, pk_i)\ )$

// Sign m using leaf key:
$\sigma_d \xleftarrow{R} S_q(sk_{d-1},\ (a_d, m)\ )$

// Output signature:
$\sigma \leftarrow \big(\ \vec{a},\ (pk_1, \sigma_1),\ \ldots,\ (pk_{d-1}, \sigma_{d-1}),\ \sigma_d\ \big)$
output $\sigma$
```

```
Algorithm $V(pk_0, m, \sigma)$:

// Parse signature components:
$\sigma \leftarrow \big(\vec{a},\ (pk_1, \sigma_1),\ \ldots,\ (pk_{d-1}, \sigma_{d-1}),\ \sigma_d\big)$

// Verify public-keys along path to leaf:
for $i = 1$ to $d - 1$:
    if $V_q(pk_{i-1},\ (a_i, pk_i),\ \sigma_i) =$ reject:
        output reject and stop

// Verify signature on m:
if $V_q(pk_{d-1},\ (a_d, m),\ \sigma_d) =$ reject:
    output reject and stop

output accept
```

Figure 14.7: The Merkle signing and verification algorithms

pk. Now $\mathcal{B}$ knows the private keys for all ℓ instances of $\mathcal{S}_q$ except for one. It has a signing oracle that it can query to generate up to q (indexed) signatures for this pk.

Next, $\mathcal{B}$ assigns pk_0 to the root of the q-ary tree and sends pk_0 to $\mathcal{A}$ as the $\mathcal{S}_{\text{Merkle}}$ public-key to attack. Adversary $\mathcal{A}$ issues signature queries $m_1, \ldots, m_q$ for $\mathcal{S}_{\text{Merkle}}$. For the ith query m_i, adversary $\mathcal{B}$ picks a random leaf v_i and assigns public-keys in $pk_0, \ldots, pk_{\ell-1}$ to internal nodes on the path from this leaf to the root. $\mathcal{B}$ does this assignment consistently, namely, once some public-key pk_i is assigned to an internal node this assignment will remain in effect for the remainder of the game.

Next, $\mathcal{B}$ uses the secret keys at its disposal to generate the necessary $\mathcal{S}_q$ signatures to obtain a valid $\mathcal{S}_{\text{Merkle}}$ signature for m_i. This requires generating signatures with respect to all the public-keys on the path from the leaf v_i to the root. It sends the resulting $\mathcal{S}_{\text{Merkle}}$ signature to $\mathcal{A}$.

In the event that one of the public-keys on the path from the leaf v_i to the root is pk, our $\mathcal{B}$ generates the required pk signature by issuing a signature query to its challenger. This works fine as long as $\mathcal{B}$ never queries its challenger for pk signatures on distinct messages $(u, \hat{m}_0)$ and $(u, \hat{m}_1)$ that have the same u. Such queries are not allowed in the q-indexed attack game. Observe that this failure event can only happen if two messages m_i, m_j from $\mathcal{A}$ happen to get mapped to the same leaf node. Since there are q^d leaves and each of the Q messages is assigned to a random leaf, this happens with probability at most $Q^2/(2q^d)$.

Now, suppose all queries from $\mathcal{A}$ are mapped to distinct leaves. Then we just said that $\mathcal{B}$ correctly answers all signature queries from $\mathcal{A}$. Eventually, $\mathcal{A}$ produces a $\mathcal{S}_{\text{Merkle}}$ signature forgery (m, σ), where σ is a vector containing d signatures. This σ uses some leaf v. Visualize the path from v to the root of the tree. Similarly, for each of the Q signatures given to $\mathcal{A}$, visualize the

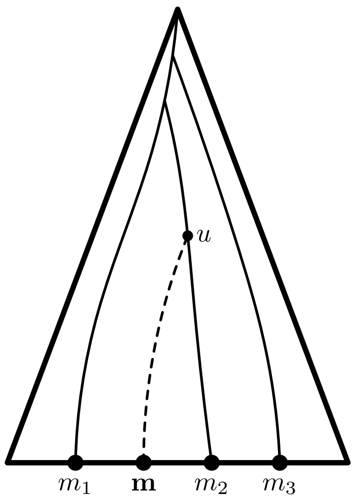


Figure 14.8: Merkle signatures: proof of security

corresponding Q paths to the root, as shown in Fig. 14.8. Let u be the lowest tree node at which the path from v intersects one of these Q paths. pk_u is the public key at that node. Suppose the leaf v is a descendant of the ith child of u.

The main point is that σ must contain an existential forgery for the public-key pk_u. This is because throughout the interaction with $\mathcal{A}$, adversary $\mathcal{B}$ never generated a signature with index i with respect to pk_u. If this node u happens to be the node to which pk is assigned then $\mathcal{B}$ just obtained a forgery on pk that lets it win the q-indexed forgery game. Since pk is placed randomly in one of the $\ell = Qd$ key pairs, this happens with probability $1/Qd$, as required. □

14.6.4 Nonce-based Merkle signatures

Up until now we only considered stateless signatures — the signer did not maintain state between invocations of the signing algorithm. Several signature systems, including Merkle signatures, become more efficient when the signing algorithm is allowed to maintain state. We observed a similar phenomenon in Section 7.5 where stateful MACs were occasionally more efficient than their stateless counterparts.

A **nonce-based signature** is a tuple of three algorithms (G, S, V) as in the case of stateless signatures. Algorithms G and V have the same inputs and outputs as in the stateless case. The signing algorithm S, however, takes an additional input $\mathcal{N}$ called a nonce that lies in some **nonce-space $\mathcal{N}$**. The system remains secure as long as algorithm S is never activated twice using the same nonce $\mathcal{N}$. That is, the system is existentially unforgeable, as long as the adversary does not obtain two signatures $S(sk, m, \mathcal{N})$ and $S(sk, m', \mathcal{N}')$ where $\mathcal{N} = \mathcal{N}'$.

Stateless signatures are preferable to nonce-based ones, especially in an environment where multiple entities can issue signatures for a particular private key. For example, a heavily loaded certificate authority is often implemented using several machines, each of which issues signatures using the authority's private key. A nonce-based signature in these settings would be harder to use since all these machines would have to somehow synchronize their state to ensure that the signing algorithm is never called twice with the same nonce. While this is certainly feasible, one typically prefers stateless signatures so that synchronization is a non-issue.

Nonce-based Merkle signatures. When nonce-based signatures are adequate, the nonce can greatly improve the efficiency of the Merkle signature system. Recall that the stateless Merkle signing algorithm chose a random leaf in the q-ary tree and signed the message using that leaf. The number of leaves had to be sufficiently large so that the probability of choosing the same leaf twice is negligible. In the nonce-based settings, we can simply make the nonce indicate what leaf to use. The uniqueness of the nonce ensures that every signature uses a different leaf. This lets us greatly shrink the Merkle signing tree leading to much shorter and more efficient signatures.

Specifically, the nonce-based Merkle signing algorithm takes as input a tuple $(sk, m, \mathcal{N})$, where $\mathcal{N}$ is a nonce, and outputs a signature. It signs m using leaf number $\mathcal{N}$. The only modification to Fig. 14.7 is that leaf number $\mathcal{N}$ is used instead of a random leaf. The nonce space $\mathcal{N}$ is simply the integers between 1 and the number of leaves in the tree, namely $\mathcal{N} := \{1, \ldots, q^d\}$. The verification algorithm is unchanged from Fig. 14.7.

If we wish to support 2^{40} signatures per public key, it suffices to choose q and d so that $q^d \geq 2^{40}$. This gives much shorter signatures than in the stateless scheme where we needed q^d to be much larger than 2^{40} to ensure that no two messages are ever randomly assigned to the same leaf.

Comparing signature sizes. Stateless Merkle signatures are much longer than nonce-based ones. Consider, for example, nonce-based Merkle signatures supporting 2^{40} signatures per public key. Using $q = 1024$ and $d = 4$, and using the q-indexed signature from Section 14.6.1, a nonce-based signature contains only four one-time signatures plus 40 hashes: 10 hashes for each of the Merkle trees used in the q-indexed signature. Using the 2.1 KB Winternitz signature scheme, this comes to about 9.6 KB per signature. Stateless signatures, where $q^d = 2^{160}$, $n = 1024$, and $d = 16$ are four times longer. In comparison, RSA signatures are far shorter, only 256 bytes per signature, but are under threat from progress in quantum computing.

We conclude by pointing out that in the nonce-based settings, the extreme parameters $q = 2^{40}$ and $d = 1$ can be quite useful for signing software updates. This setup corresponds to a very wide tree of depth 1. Key generation is slow, but signature verification is super fast: only a single one-time signature verification plus 40 hash operations for the Merkle tree, as explained in (14.5). Signature generation can also be done efficiently: if the nonce is a counter, counting from 1 to q, then an efficient **Merkle tree traversal** algorithm can be used to quickly generate the Merkle tree nodes needed for each signature. See Exercise 14.20.

14.7 A fun application: the TESLA broadcast MAC

Alice is the head engineer at the Galileo project, a **global navigation satellite system**, or **GNSS**. GNSS is the general name for navigation systems that use a constellation of satellites. Prominent examples include the American GPS, the European Galileo, the Russian Glonass, and the Chinese Beidou. Alice's group is in charge of managing dozens of satellites that form the backbone of the Galileo system. Each satellite broadcasts a short message to earth every second, and these messages are processed by GNSS receivers in phones, cars, airplanes, ships, and many other devices.

Alice is worried about a spoofing attack on Galileo. A spoofer is a device that jams the true signal from the satellites, and transmits a fake signal in its place. When the spoofer is active, every receiver close by is fooled into reporting the wrong location. A GPS spoofer is believed to have been the cause of a 2017 maritime incident in which twenty ships in the Black Sea reported that their navigation equipment placed them 32km away from their true location — they knew the system was wrong because it placed the ships near an inland airport.

Alice wants to secure Galileo against spoofing attacks. She knows that there are many other possible attacks on GNSS, but she is specifically concerned about spoofing.

Alice's first idea — after reading the last two chapters — is to use digital signatures: every message from the satellites will be signed using a secret key in the satellite, and all GNSS receivers will have the corresponding public verification key. This prevents spoofing because GNSS receivers will ignore incoming messages that are not properly signed.

However, there is a problem. The data rate in GNSS systems is extremely low; only 20 bits per second can be used for data authentication. At this slow data rate, sending a long 2048 bit RSA signature is out of the question. In Chapters 19 and 15 we will construct shorter digital signatures, where each signature can be as short as 384 bits. However, at 20 bits per second, even a 384 bit signature will take twenty seconds to transmit. This means that, at best, location data can be verified twenty seconds after it is transmitted. Alice would like to do better.

The problem we are trying to solve is called **broadcast authentication**: a single source, a satellite, is broadcasting data that needs to be verified by many recipients. Ideally, Alice would like to use a MAC to solve this problem because a MAC tag is much shorter than a digital signature. However, to use a MAC, the sender and all the recipients need the MAC key, and this is insecure. If all the recipients have the MAC key, then the spoofing device can also have it, and this key lets the spoofer compute a valid tag for its attack messages (see also Exercise 6.4).

A broadcast authentication system should be secure against an adversary who controls many recipients, possibly all the recipients except one. Moreover, the adversary can mount a chosen message attack on the source. The adversary's goal is to produce a new message-tag pair that will be accepted by a recipient that the adversary does not control. The system is secure if no efficient adversary can do so with non-negligible probability.

Timed broadcast authentication. Let us see a clever way to use a secure MAC for broadcast authentication. The approach, called TESLA, is based on hash chains developed in Section 14.3. First, let us specify the problem more precisely.

The satellite needs to broadcast a stream of messages $m_1, m_2, \ldots, m_n \in \mathcal{M}$, one message every fixed time period, say every T seconds for some small value of T. Messages are not known ahead of time: every message is known only shortly before it is transmitted. The TESLA system can be used whenever the following two assumptions hold:

- *Synchronized clocks:* The satellite and all the receivers have synchronized clocks, and the clock skew between the satellite and the receivers is at most $T/2$ seconds. Moreover, every message m_i, for $i = 1, \ldots, n$, contains a data field that indicates the time slot for which the message is intended. A recipient will reject a message that is received outside of its indicated time slot.

- *Delayed authentication:* The system can function properly even if an incoming message can only be validated T seconds after it is received. During the T seconds gap, the receiver cannot tell if an incoming message is authentic or forged. Exercise 14.21 explores an alternate design that supports *immediate authentication* at the cost of a moderate increase to the broadcast size.

Both assumptions hold for GNSS receivers, although there is some debate as to whether a receiver should display an updated location during the T seconds gap when the received data has not yet been authenticated.

With this setup, we are ready to describe the TESLA system.

The TESLA broadcast authentication system. Let (S, V) be a secure MAC defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, and let $f : \mathcal{K} \to \mathcal{K}$ be a function. The broadcaster sets up the scheme by creating a hash chain of MAC keys. It chooses a random key $k_0 \overset{\text{R}}{\leftarrow} \mathcal{K}$ and computes

$$k_1 := f(k_0), \quad k_2 := f(k_1), \quad \ldots, \quad k_n = f(k_{n-1})$$

for some n that will be determined later. Pictorially this chain of keys can be described as:

$$\begin{tikzcd} k_0 \arrow[r, "f"] & k_1 \arrow[r, "f"] & k_2 \arrow[r, "f"] & \cdots \arrow[r, "f"] & k_{n-1} \arrow[r, "f"] & k_n \end{tikzcd}$$

For now, let us assume that all the recipients somehow have k_n, the last key in the chain. We will discuss how to distribute k_n at the end of the section.

We assume that f is one-way on n iterates as in Definition 14.5. This means that an adversary that is given $k_i \in \mathcal{K}$, for some $i > 0$, cannot compute k_{i-1}. In fact, we will need something slightly stronger: we need that for all $i = 0, \ldots, n-1$, the MAC system (S, V) is secure with respect to the key k_i (as in Attack Game 6.1) even if the adversary is given $k_{i+1} = f(k_i)$.

To explain how the TESLA system works, let us first see how the satellite authenticates the first message m_1. The first two broadcasts that the satellite sends are defined as follows:

- *message 1:* use key k_{n-1} to compute $t_1 := S(k_{n-1}, m_1)$ and broadcast (m_1, t_1).
- *message 2:* use key k_{n-2} to compute $t_2 := S(k_{n-2}, m_2)$ and broadcast (m_2, t_2, k_{n-1}).

Notice that the tag t_1 in message 1 is computed using the key k_{n-1}. This key is included in the body of message 2.

A recipient first receives message 1, but cannot validate the tag t_1 because it does not yet have the key k_{n-1}. Instead, the recipient waits. Message 2, which arrives T seconds later, includes k_{n-1}. Now the recipient can verify m_1 by checking that (i) the key k_{n-1} is correct, namely $f(k_{n-1}) = k_n$, and (ii) m_1 is valid, namely $V(k_{n-1}, m_1, t_1) = \mathsf{accept}$. If so, then m_1 is accepted. We see that there is a delay of T seconds between the time that the message m_1 is received, and the time when it is marked as authentic.

Why is this secure? Suppose the attacker wants to replace message m_1 with a spoofed message $m_1' \neq m_1$ that has the same time slot number as m_1. Then:

- Before message 2 is broadcast, the attacker does not have the key k_{n-1}, and cannot compute a valid tag for m_1'. In fact, because of our security assumption about the MAC system (S, V), the attacker cannot create any new valid pair (m_1', t_1') with respect to the key k_{n-1}.
- Once message 2 is broadcast, the attacker obtains k_{n-1}, and can now compute a valid tag for m_1' as $t_1' := S(k_{n-1}, m_1')$. It can then broadcast (m_1', t_1'). However, this is too late: the broadcast (m_1', t_1') will be rejected by the recipients because at this point the recipients are expecting a message for time slot number two. They reject messages for a lower time slot.

As a result, the attacker cannot replace m_1 with a forgery m_1'.

The remaining messages are authenticated in the same way. In particular, the sequence of broadcasts from left to right is as follows: (one broadcast every T seconds)

$$\begin{tikzcd} \boxed{\begin{matrix} m_1 \\ t_1 \end{matrix}} \arrow[r] & \boxed{\begin{matrix} m_2 \\ t_2 \\ k_{n-1} \end{matrix}} \arrow[r] & \boxed{\begin{matrix} m_3 \\ t_3 \\ k_{n-2} \end{matrix}} \arrow[r] & \cdots \arrow[r] & \boxed{\begin{matrix} m_{n-1} \\ t_{n-1} \\ k_2 \end{matrix}} \arrow[r] & \boxed{\begin{matrix} m_n \\ t_n \\ k_1 \end{matrix}} \arrow[r] & \boxed{k_0} \end{tikzcd} \tag{14.6}$$

where $t_i := S(k_{n-i}, m_i)$ for $i = 1, \ldots, n$. When a recipient receives the broadcast (m_i, t_i, k_{n-i+1}), for $i = 2, \ldots, n$, it checks that (i) $f(k_{n-i+1}) = k_{n-i+2}$, and (ii) $V(k_{n-i+1}, m_{i-1}, t_{i-1}) = \textsf{accept}$. If so, it accepts the message m_{i-1} as authentic. Again, message m_{i-1} is accepted T seconds after it is received. The security argument is the same as the argument for the message m_1.

Data size in each broadcast. Let's ignore the first and last broadcasts (which are special). The authentication data in all other broadcasts is a pair $(t, k) \in \mathcal{T} \times \mathcal{K}$. In our settings, the tag t can be as short as 32 bits because a forgery probability of $1/2^{32}$ is acceptable. However, the key k needs to be at least 128 bits to ensure that the function f is one-way. Together, this is 160 bits per broadcast, which is much shorter than 384 bits when using a digital signature. Assuming a rate of 20 bits per second, the system will take eight seconds to broadcast one (t, k) pair, resulting in an authentication delay of eight seconds for received messages. If the keys are reduced to 100 bits and the tags are reduced to 20 bits, then the delay is only six seconds, at the cost of reduced security.

How to distribute the top of the chain. We still need to explain how to securely distribute k_n to all the recipients. If the hash chain of keys is very long, say long enough to support several months or even years worth of broadcasts, then the top of the chain can be distributed by an offline process. For example, k_n can be posted on a trusted web site and all the recipients get it from there. We will not go into more detail here, but note that in GNSS, the distribution of k_n is a complicated process.

Remark 14.1 (a word of caution). Our description of the hash chain is a bit oversimplified. In practice one needs to defend against pre-processing attacks such as the ones discussed in Section 18.3.1.3. To do so, the hash chain function f must take two additional inputs. First, there is a random nonce $\mathcal{N}$ generated when the chain is first created. This "randomizes" the function f and makes it harder to carry out an offline analysis of f. This nonce is distributed to recipients along with the key k_n. Second, the function takes its position in the chain as input. In other words, for $i > 0$, the key k_i is computed as $k_i := f(k_{i-1}, \mathcal{N}, i)$. □

14.8 Notes

Citations to the literature to be added.

14.9 Exercises

14.1 (Shortening the public-key). Let (G, S, V) be a signature scheme, and suppose that algorithm G generates public-keys in some set $\mathcal{X}$. We show a generic transformation that gives a new signatures scheme (G', S', V') where the public-key is short, only 16 bytes.

(a) Let H be hash function defined over $(\mathcal{X}, \mathcal{Y})$. Algorithm G' now works as follows: it runs algorithm G to obtain pk and sk, and outputs

$$pk' := H(pk), \qquad sk' := (pk, sk).$$

Explain how algorithms S' and V' work.

(b) Prove that (G', S', V') is a secure signature scheme, assuming (G, S, V) is secure, and H satisfies the following collision resistance property, which is a variation of 2nd-preimage collision resistance: namely, given (pk, sk) as generated by G, it is hard find $pk^* \neq pk$ such that $H(pk^*) = H(pk)$.

Note: If H is modeled as a random oracle, then $|\mathcal{Y}| \approx 2^{128}$ is large enough to ensure reasonable (non-quantum) security.

(c) Show that when this transformation is applied to the basic Lamport signature scheme $\mathcal{S}_{\mathrm{L}}$ discussed in Section 14.1, the signature size need only be twice as long as in $\mathcal{S}_{\mathrm{L}}$.

14.2 (Attacking Lamport multi-key security). In our description of the various Lamport signature schemes there is a fixed one-way function $f : \mathcal{X} \to \mathcal{Y}$ that all users in the system use. This can cause a problem.

(a) Consider the multi-key signature game from Exercise 13.2 played against the basic Lamport signature scheme. Show that after seeing $\approx |\mathcal{Y}|^{1/2}$ public keys, and one signature under each of these keys, an adversary can forge the signature for one of the given public keys with probability 1/2. This gives the adversary advantage 1/2 in winning the multi-key security game.

(b) When $\mathcal{Y} := \{0,1\}^{256}$ the attack from part (a) is not a concern. However, when the range is smaller, say $\mathcal{Y} := \{0,1\}^{128}$, this can lead to a real-world attack. A simple solution is to expand the domain of f to $\mathcal{R} \times \mathcal{X}$ and modify the key generation algorithm to include a fresh random nonce $r \in \mathcal{R}$ in the public and secret keys. The r associated with a key pair (pk, sk) will always be prepended to the input of f when operating with pk or sk. Explain why this prevents the attack from part (a) when $|\mathcal{R}| = |\mathcal{Y}|$.

14.3 (An injective mapping to ℓ-size subsets). Recall that $\mathrm{Sets}[n, \ell]$ is the set of all ℓ-size subsets of $\{1, \ldots, n\}$. In Section 14.4 we needed an injective mapping $P_{\mathrm{hors}} : \{0,1\}^v \to \mathrm{Sets}[n, \ell]$ where $2^v \leq \binom{n}{\ell}$, that is efficiently computable. The following algorithm provides such a mapping. In fact, it injectively maps any integer in $\big[0, \binom{n}{\ell}\big)$ to an element of $\mathrm{Sets}[n, \ell]$.

```
input: $0 \leq m < \binom{n}{\ell}$
output: $s \subseteq \{1, \ldots, n\}$ where $|s| = \ell$
$s \leftarrow \emptyset$,  $t \leftarrow \ell$
for $k = n$ down to 1 until $t = 0$:
    if $m < \binom{k-1}{t-1}$:
        $s \leftarrow s \cup \{k\}$,    $t \leftarrow t - 1$
    else:
        $m \leftarrow m - \binom{k-1}{t-1}$
output $s$
```

Prove that the function computed by this algorithm always outputs a set in Sets$[n, \ell]$ and is injective.

Hint: Use the identity $\binom{k}{t} = \binom{k-1}{t-1} + \binom{k-1}{t}$. This identity corresponds to a partition of Sets$[k, t]$ into two types of sets: sets that contain the element k and sets that do not.

Discussion: The $n \times \ell$ binomial coefficients used in the algorithm can be pre-computed so that the online running time is quite fast. If that table is too large to store, the algorithm can pre-compute a single value, namely $\binom{n-1}{\ell-1}$, and quickly derive from it the n binomial coefficients needed for a run of the algorithm. For example, $\binom{n-2}{\ell-2} = \binom{n-1}{\ell-1} \cdot \frac{\ell-1}{n-1}$, when $n, \ell > 1$. This takes one integer multiplication and one integer division per iteration.

14.4 (Another injective mapping to ℓ-size subsets). Let us see another injective function $P_{\text{hors}} : \{0,1\}^v \to$ Sets$[n, \ell]$ that is designed for the case when the input is uniform in $\{0,1\}^v$. Suppose that $n = 2^t$ and $v = t(\ell + c)$ for some $c \geq 0$. This lets us treat an element of $\{0,1\}^v$ as a sequence of $\ell + c$ elements in $\{1, \ldots, n\}$. For a random $m \in \{0,1\}^v$ define $P_{\text{hors}}(x)$ as:

```
parse $m$ as a sequence $u_1, u_2, \ldots, u_{\ell+c} \in \{1, \ldots, n\}$
$i \leftarrow 0$ ,   $s \leftarrow \emptyset$
repeat:
    $i \leftarrow i + 1$,      $s \leftarrow s \cup \{u_i\}$
until $|s| = t$ or $i = \ell + c$
if $|s| = t$ output the set $s$; otherwise output fail.
```

(a) Show that for $m \xleftarrow{\text{R}} \{0,1\}^v$, if $P_{\text{hors}}(m) \neq \mathsf{fail}$ then $P_{\text{hors}}(m)$ is uniformly distributed in Sets$[n, \ell]$.

(b) Show that for $m \xleftarrow{\text{R}} \{0,1\}^v$, the probability that $P_{\text{hors}}(m) = \mathsf{fail}$ is bounded by $e^{t-1} \cdot (t/n)^{c+1}$, where $e \approx 2.71$.

Discussion: We can assume that the input m to P_{hors} is uniform because m is typically the output of a random oracle applied to the message to be signed plus a random nonce (as in Section 14.1.1). The function P_{hors} built here is more efficient that the one in Exercise 14.3, but has a failure probability which can occasionally force a re-try with a fresh nonce.

14.5 (Lamport q-time stateless signatures). Let P be a function mapping $\mathcal{M}$ to subsets of $\{1, \ldots, n\}$. We say that P is **q-containment free** if for every $x, y_1, \ldots, y_q \in \mathcal{M}$, we have $P(x) \nsubseteq P(y_1) \cup \cdots \cup P(y_q)$ whenever $x \notin \{y_1, \ldots, y_q\}$.

(a) Generalize Theorem 14.2 to show that if the function P is q-containment free then the general Lamport framework (Section 14.2) is a q-time secure signature scheme.

(b) Show that if P is q-containment free then $n = \Omega(q^2 v)$, where $|\mathcal{M}| = 2^v$. This shows that the public-key or the signature size must grow quadratically in q.

14.6 (q-time HORST stateless signatures). Let $P : \mathcal{R} \times \mathcal{M} \to$ Sets$[n, \ell]$ be a function. Let $\mathcal{A}$ be an adversary that takes as input sets $s_1, \ldots, s_q$ in Sets$[n, \ell]$ and outputs a pair $(r, x) \in \mathcal{R} \times \mathcal{M}$ such that $P(r, x) \subseteq s_1 \cup \cdots \cup s_q$.

(a) Show that if P is modeled as a random oracle, and $\mathcal{A}$ makes at most Q_{ro} queries to P then $\mathcal{A}$ succeeds with probability at most $Q_{\text{ro}} \cdot \binom{q\ell}{\ell} / \binom{n}{\ell}$. Therefore, for a given q, one can choose the parameters n, ℓ so that a bounded adversary succeeds with only negligible probability.

(b) Explain how to use the function P as an enhanced TCR in the HORST system. Use part (a) to show that the resulting signature scheme is q-time secure when n, ℓ are chosen so that $Q_{\text{ro}} \cdot \binom{q\ell}{\ell} / \binom{n}{\ell}$ is negligible.

(c) Continuing with part (b) and setting $n := 2048$, what is the smallest value of ℓ needed if we want the adversary's advantage in defeating the HORST 2-time signature to be at most $Q_{\text{ro}}/2^{256}$? What is the smallest ℓ for a 3-time signature under the same conditions?

Discussion: Assuming $\mathcal{X} = \hat{\mathcal{Y}}$, the resulting combined size of a 2-time HORST signature and public-key is $347 \cdot \log_2(|\mathcal{X}|)$ bits. The 3-time HORST combined size is $433 \cdot \log_2(|\mathcal{X}|)$ bits. This is much shorter than the corresponding sizes for 2-time and 3-time Lamport signatures from Exercise 14.5 using the same n and $\mathcal{X}$.

14.7 (Insecure two-time signatures). Let $\mathcal{S} = (G, S, V)$ be a secure (many-time) signature scheme. Show how to construct from $\mathcal{S}$ a new signature scheme $\mathcal{S}'$ that is one-time secure, but two-time insecure: if the signer uses a single signing key to sign two messages, then the secret key is revealed publicly.

Hint: Try embedding in the public-key an encryption of the secret key under some symmetric key k. Every signature must include a share of k, otherwise the signature is rejected.

14.8 (Lamport is strongly secure). Prove that the general Lamport framework in Section 14.2 gives is a strongly secure one-time signature scheme in the sense of Definition 14.2, assuming the one-way function f is also 2nd-preimage collision resistant (as in Definition 8.6).

14.9 (Winternitz is strongly secure). As in the precious exercise, one can also show that the Winternitz construction in Section 14.3 gives a strongly secure one-time signature, under an appropriate assumption on the function f. State the assumption and prove the result.

14.10 (A many-time strongly secure signature). Consider the online-offline signature construction in Section 14.5.1. Suppose we modify the signing algorithm so that σ_1 is computed as $\sigma_1 \xleftarrow{\text{R}} S_1(sk_1, (m, \sigma_0))$, and modify the verification algorithm accordingly. Prove that this modified scheme is strongly secure (in the sense of Definition 13.3) assuming that $\mathcal{S}_\infty$ is secure and $\mathcal{S}_1$ is strongly one-time secure.

14.11 (A strongly secure one-time signature from discrete log (I)). Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $H : \mathcal{M} \to \mathbb{Z}_q$ be a hash function. We define a signature scheme (G, S, V) with message space $\mathcal{M}$ as follows.

- The key generation algorithm G computes $\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q,\ u \leftarrow g^\alpha \in \mathbb{G},\ v \leftarrow g^\beta \in \mathbb{G}$, and outputs the public-key $pk := (u, v) \in \mathbb{G}^2$ and the secret key $sk := (\alpha, \beta) \in \mathbb{Z}_q^2$.
- Given a secret key $sk = (\alpha, \beta) \in \mathbb{Z}_q^2$ and a message $m \in \mathcal{M}$, the signing algorithm S computes $c \leftarrow H(m) \in \mathbb{Z}_q$ and outputs the signature $\sigma \leftarrow c\alpha + \beta \in \mathbb{Z}_q$.
- Given a public-key $pk = (u, v) \in \mathbb{G}^2$, a message $m \in \mathcal{M}$, and a signature $\sigma \in \mathbb{Z}_q$, the verification algorithm V computes $c \leftarrow H(m)$ and accepts if $g^\sigma = v \cdot u^c$.

Discussion: Notice that for every message and public key there is a *unique* signature that will be accepted by the verifier. Moreover, signing is quite fast, only one multiplication and one addition in $\mathbb{Z}_q$. As such, this scheme is particularly well suited for online/offline signatures as discussed in Section 14.5.1.

(a) Show that when H is modeled as a random oracle, an adversary that breaks the strong one-time security of the scheme (as in Definition 14.2) can be used to solve the discrete logarithm problem in $\mathbb{G}$.

(b) The construction from part (a) is trivially not two-time secure: signing two distinct messages with the same key reveals the secret key by solving a linear system in the variables α, β. Generalize the scheme to obtain a strong two-time secure signature scheme. The public key pk is in $\mathbb{G}^3$, the signature is in $\mathbb{Z}_q$, and the hash function H is $H : \mathcal{M} \to \mathbb{Z}_q^2$. Prove security of the scheme based on the difficulty of the discrete-log problem in $\mathbb{G}$, assuming H is modeled as a random oracle.

14.12 (A strongly secure one-time signature from discrete log (II)). Like the previous exercise, this exercise develops a strongly secure one-time signature based on the discrete logarithm assumption. However, unlike in the previous exercise, this scheme does not rely on the random oracle model. The cost is that signatures are twice as long.

Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $h \in \mathbb{G}$ be a random group element, which we view as a system parameter. We can define a signature scheme (G, S, V) with message space $\mathbb{Z}_q$ as follows.

- The key generation algorithm G computes

$$\alpha, \beta, \gamma, \delta \xleftarrow{\text{R}} \mathbb{Z}_q,\ \ u \leftarrow g^{\alpha} h^{\beta} \in \mathbb{G},\ \ v \leftarrow g^{\gamma} h^{\delta},$$

  and outputs the public-key $pk := (u, v) \in \mathbb{G}^2$ and the secret key $sk := (\alpha, \beta, \gamma, \delta) \in \mathbb{Z}_q^4$.

- Given a secret key $sk = (\alpha, \beta, \gamma, \delta) \in \mathbb{Z}_q^4$ and a message $m \in \mathbb{Z}_q$, the signing algorithm S computes

$$\sigma \leftarrow \gamma + m\alpha,\ \ \tau \leftarrow \delta + m\beta,$$

  and outputs the signature $(\sigma, \tau) \in \mathbb{Z}_q^2$.

- Given a public-key $pk = (u, v) \in \mathbb{G}^2$, a message $m \in \mathbb{Z}_q$, and a signature $(\sigma, \tau) \in \mathbb{Z}_q^2$, the verification algorithm V checks if

$$g^{\sigma} h^{\tau} = v \cdot u^{m},$$

  and outputs accept if this holds, and reject otherwise.

(a) Show that an adversary that breaks the strong one-time security of the scheme (as in Definition 14.2) can be used to find two different representations of a u, relative to g and h, as defined in Section 10.6.1. Hence, by Fact 10.3, this adversary can be used to solve the discrete logarithm problem in $\mathbb{G}$.

Hint: First argue that the information contained in the public key and a single signature is independent of β. To do this, fix the random choices made by the adversary. Now, condition on any fixed value of β, and show that (u, v, τ) is uniformly distributed over $\mathbb{G} \times \mathbb{G} \times \mathbb{Z}_q$, m is determined by u and v, and σ is determined by the equation $g^{\sigma} h^{\tau} = v \cdot u^{m}$. This independence immediately implies that if we condition on any fixed value of (u, v, τ), then β is uniformly distributed over $\mathbb{Z}_q$.

(b) Consider the key generation algorithm G_0 that is the same as G, but sets $\beta := 0$. Show that (G_0, S, V) is strongly one-time secure.

Hint: Show that for any adversary, its advantage in breaking (G_0, S, V) is identical to its advantage in breaking (G, S, V). Again, use the hint from part (a): the information contained in the public key and a single signature is independent of β.

Discussion: In this modified scheme, the public key is just $(u, v) = (g^{\alpha}, g^{\gamma} h^{\delta})$, and the secret key is (α, γ, δ). A signature on m is just $(\sigma, \tau) = (\gamma + m\alpha, \delta)$, which can be computed using just one multiplication and one addition in $\mathbb{Z}_q$, like the scheme in the previous exercise. Since $\tau = \delta$, which does not depend on m, one might be tempted to modify the scheme further, and just place δ itself in the public key, and leave it out of the signature (and leave h out of the scheme altogether). However, the proof of security would no longer apply to this scheme (can you see why?). Moreover, it is not hard to see that this last scheme would be no more secure than the scheme in the previous exercise, but with no hash function, and the security of that scheme is not at all clear.

(c) Show that (G, S, V) is not two-time secure: given signatures on two distinct messages m_0 and m_1 in $\mathbb{Z}_q$, the adversary can forge the signature on every message $m \in \mathbb{Z}_q$ of its choice.

14.13 (From AD-only CCA security to CCA security). Suppose $\mathcal{E} = (G, E, D)$ is a public-key encryption scheme with message space $\mathcal{M}$, ciphertext space $\mathcal{C}$, and associated data space $\mathcal{P}$. Suppose $\mathcal{S}_1 = (G_1, S_1, V_1)$ is a signature scheme with message space $\mathcal{C} \times \mathcal{D}$, where $\mathcal{D}$ is some finite set, and where public keys are contained in the set $\mathcal{P}$ above. We define a new public-key encryption scheme $\mathcal{E}' = (G, E', D')$ with message space $\mathcal{M}$ and associated data space $\mathcal{D}$ as follows:

$$\begin{aligned}
E'(pk, m, d) \quad &:= \quad (pk_1, sk_1) \xleftarrow{\text{R}} G_1(),\ \ c \xleftarrow{\text{R}} E(pk, m, pk_1),\ \ \sigma \xleftarrow{\text{R}} S_1(sk_1, (c, d)) \\
& \quad \text{output } (pk_1, c, \sigma); \\
D'(sk, (pk_1, c, \sigma), d) \quad &:= \quad \text{if } V_1(pk_1, (c, d)) = \mathsf{reject} \\
& \quad \qquad \text{then output } \mathsf{reject} \\
& \quad \qquad \text{else \ output } D(sk, c, pk_1).
\end{aligned}$$

Show that if $\mathcal{E}$ is AD-only CCA secure (as defined in Section 12.7.1), and $\mathcal{S}_1$ is strongly one-time secure, then $\mathcal{E}'$ is CCA secure (as in Definition 12.7).

14.14 (Online/offline signatures from discrete-log). In Section 14.5.1 we showed that one-time signatures can be used to improve the online performance of any signature scheme. The one-time signature scheme in Exercise 14.12 is especially well suited for this application: signatures are relatively short and signing is fast. In this exercise we show an even better approach. Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $(G_\infty, S_\infty, V_\infty)$ be a many-time signature scheme with message space $\mathcal{M}_\infty := \mathbb{G}$. Define the following many-time signature scheme (G, S, V) with message space $\mathcal{M} := \mathbb{Z}_q$:

$$G() := \left\{ \begin{array}{l} (pk_\infty, sk_\infty) \xleftarrow{\text{R}} G_\infty(), \\ \tau \xleftarrow{\text{R}} \mathbb{Z}_q, \quad h \leftarrow g^{\tau}, \\ sk := (sk_\infty, \tau), \quad pk := (pk_\infty, h) \end{array} \right\}$$

$$S(sk,m) := \left\{ \begin{array}{l} \text{offline phase:} \\ \quad \rho \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad u \leftarrow g^{\rho}, \\ \quad \sigma \leftarrow S_{\infty}(sk,u) \\ \\ \text{online phase:} \\ \quad \alpha \leftarrow \rho - \tau m \in \mathbb{Z}_q \\ \quad \text{output } (\sigma,\alpha) \end{array} \right\} \qquad V\big(pk, m, (\sigma,\alpha)\big) := \left\{ \begin{array}{l} u \leftarrow g^{\alpha} h^{m}, \\ \text{output } V_{\infty}(pk_{\infty}, u) \end{array} \right\}$$

Show that (G, S, V) is secure assuming that $(G_{\infty}, S_{\infty}, V_{\infty})$ is secure and the discrete-log assumption holds for $\mathbb{G}$.

Discussion: Note that the online signing phase is only one multiplication and one addition in $\mathbb{Z}_q$. This construction is based on the trapdoor hash $H_{\mathrm{dl}}(\alpha,\beta) := g^{\alpha} h^{\beta}$ discussed in Exercise 10.30, where $\tau = \mathsf{Dlog}_g(h)$ is the trapdoor.

14.15 (Repeated d-iterates of a one-way function). In the proof of Winternitz's scheme (Theorem 14.4) we needed a generalization of Lemma 13.5 that applies to iterated one-way functions. Consider the following generalization of Attack Game 13.3 for given parameters n and d and adversary $\mathcal{A}$:

- The challenger computes $x_1, \ldots, x_n \xleftarrow{\mathrm{R}} \mathcal{X}$ and $y_1 \leftarrow f^{(d)}(x_1), \ldots, y_n \leftarrow f^{(d)}(x_n)$. It sends $(y_1, \ldots, y_n)$ to $\mathcal{A}$.

- $\mathcal{A}$ makes a sequence of *reveal queries* (i, j) where $1 \le i \le n$ and $0 \le j \le d$. The challenger responds with $x_{i,j} := f^{(j)}(x_i)$.

- Eventually, $\mathcal{A}$ outputs (a, b, x), where a, b are positive integers and $x \in \mathcal{X}$.

We say that $\mathcal{A}$ wins the game if $f^{(b)}(x) = y_a$ and there was no reveal query (a, b') with $b' \le b$. Let riOW$\mathsf{adv}[\mathcal{A}, f, t, d]$ be the probability that $\mathcal{A}$ wins the game. Prove that for every adversary $\mathcal{A}$ in this game there exists a (single instance) iterated one-way adversary $\mathcal{B}$ such that

$$\mathrm{riOW}\mathsf{adv}[\mathcal{A}, f, n, d] \le nd \cdot \mathrm{iOW}\mathsf{adv}[\mathcal{B}, f, d]$$

14.16 (Iterated one-way functions). Let $f : \mathcal{X} \to \mathcal{X}$ be a function.

(a) Suppose f is one-way on d iterates, as in Definition 14.5. Show that the function $f^{(d)}$ is one-way.

(b) Let f be a one-way function. Construct a function $\hat{f}$ using f such that $\hat{f}$ is one-way, but $\hat{f}^{(2)}(x) := \hat{f}(\hat{f}(x))$ is not. By part (a) this $\hat{f}$ is also not one-way on a 2-iterate.

(c) Suppose f is a one-way permutation. Show that f is one-way on d iterates for all bounded d.

(d) Show that if $|\mathcal{X}|$ is large and f is a random oracle then f is one-way on d iterates for all bounded d. In particular, an adversary that makes Q_{ro} queries to the random oracle has advantage at most $O(dQ_{\mathrm{ro}}/|\mathcal{X}|)$ in winning the random oracle variant of Attack Game 14.1. Use Exercise 14.17.

14.17 (Iterations shrink the range). Let $f : \mathcal{X} \to \mathcal{X}$ be a random function. Show that for $d \ll |\mathcal{X}|^{1/2}$, the size of the image of $f^{(d)}$ behaves approximately as $\frac{2}{d+1}|\mathcal{X}|$.

Discussion: This means that inverting $f^{(d)}$ by exhaustive search takes about a factor of $(d+1)/2$ fewer attempts than inverting f. Of course, evaluating $f^{(d)}$ takes d times longer, and therefore the overall time to invert $f^{(d)}$ by exhaustive search is about the same as the time to invert f. Exercise 14.18 gives a better algorithm for inverting $f^{(d)}$.

14.18 (Inverting an iterated function). Let $f : \mathcal{X} \to \mathcal{X}$ be a random function, where $N := |\mathcal{X}|$. Let $f^{(d)}$ be its d-th iterate, for some $0 < d < \sqrt{N}/\log_2 N$. Give an algorithm $\mathcal{A}$ that makes Q queries to H, where $0 \leq Q < N/d$, and wins the one-way inversion game (Definition 8.6) against $f^{(d)}$ with advantage at least $\frac{1}{2}dQ/N$. In particular, for $x \overset{\text{R}}{\leftarrow} \mathcal{X}$, your algorithm $\mathcal{A}$ finds a preimage of $f^{(d)}(x)$ with probability $1/2$, after only about N/d queries to f. This shows that inverting $f^{(d)}$ is about d times easier than inverting f.

Hint: On input $y \leftarrow f^{(d)}(x)$, try choosing a random $x_0 \overset{\text{R}}{\leftarrow} \mathcal{X}$ and computing the sequence $x_0, f(x_0), f^{(2)}(x_0), f^{(3)}(x_0), \ldots$. If the sequence hits y after more than d steps, then a preimage of y is found. If the sequence loops on itself, choose a new random $x_0 \overset{\text{R}}{\leftarrow} \mathcal{X}$ and try again. Show that this approach has the claimed success rate.

Discussion: This method does not generalize to invert a composition of d independent random functions, $h(x) := f_1\big(f_2(\cdots f_d(x)\cdots)\big)$ where $f_1, \ldots, f_d : \mathcal{X} \to \mathcal{X}$. In fact, one can show that inverting h is as hard as inverting a random function $f : \mathcal{X} \to \mathcal{X}$. This observation can be used to strengthen the iterated hash function in the Winternitz signature scheme.

14.19 (Tree hash). Key generation in the q-indexed signature scheme of Section 14.6.1 requires building a Merkle tree over q leaves using a hash function $H : \mathcal{Y}^2 \to \mathcal{Y}$. Recall that each leaf contains the hash of a fresh public key of a one-time signature scheme. Let *LeafCalc* be a function that takes as input an integer $1 \leq i \leq q$ and returns the contents of leaf number i. Suppose that a call to *LeafCalc* takes one time unit as does one evaluation of H. Construct an algorithm that computes the hash value at the Merkle tree root in time $O(q)$ using only enough space needed to store $O(\log q)$ elements of $\mathcal{Y}$. This algorithm is called the **treehash** algorithm.

14.20 (Merkle tree traversal). Consider a Merkle tree with q leaves, where q is a power of two. Let $H : \mathcal{Y}^2 \to \mathcal{Y}$ be a hash function used to build the Merkle tree. As in the previous exercise, let *LeafCalc* be a function that takes as input an integer $1 \leq i \leq q$ and returns the contents $h_i \in \mathcal{Y}$ of leaf number i. Assume that evaluating each of *LeafCalc* and H takes one time unit. As usual, for every leaf $1 \leq i \leq q$ there is a set of $\log_2 q$ nodes in the Merkle tree that authenticate leaf i relative to the hash value at the Merkle root. This set of nodes is called the **Merkle proof** for leaf i. Let *Merkle*(i) be a function that outputs the Merkle proof for leaf number i along with the contents h_i of that leaf. The **Merkle tree traversal** problem is to compute the q items

$$\mathit{Merkle}(1),\ \mathit{Merkle}(2),\ \ldots\ \mathit{Merkle}(q)$$

sequentially one after the other. Show an algorithm for the Merkle tree traversal problem that runs in *amortized* time $\log_2 q$ per item, and only needs enough space to store $\log_2 q$ elements of $\mathcal{Y}$.

Discussion: Merkle tree traversal can speed up the signing algorithm of the *nonce-based* q-indexed signature scheme from Section 14.6.1, where the nonce is a counter that indicates which leaf in the Merkle tree to use. The counter is incremented after every invocation of the signing algorithm. In addition to the nonce, the signer maintains the necessary $O(\log q)$-size state needed for the tree traversal algorithm. Better Merkle tree travesal algorithms [152, 39] run in *worst-case* time $\log_2 q$ per output and use space $O(\log_2 q)$.

14.21 (Broadcast authentication with no delay). In Section 14.7 we presented the TESLA broadcast authentication system built from a secure MAC system (S, V) and a function f. One down side of this system is delayed authentication: an incoming message can only be verified one time slot after it is received. Let's change the system to enable immediate authentication by changing the broadcast data in (14.6) to:

$$\begin{tikzcd}
\boxed{t_1} \arrow[r] & \boxed{\begin{matrix} m_1 \\ t_2 \\ k_{n-1} \end{matrix}} \arrow[r] & \boxed{\begin{matrix} m_2 \\ t_3 \\ k_{n-2} \end{matrix}} \arrow[r] & \cdots \arrow[r] & \boxed{\begin{matrix} m_{n-2} \\ t_{n-1} \\ k_2 \end{matrix}} \arrow[r] & \boxed{\begin{matrix} m_{n-1} \\ t_n \\ k_1 \end{matrix}} \arrow[r] & \boxed{\begin{matrix} m_n \\ k_0 \end{matrix}}
\end{tikzcd}$$

where $t_i := S(k_{n-i}, m_i)$ for $i = 1, \ldots, n$. When a recipient receives the broadcast (m_i, t_{i+1}, k_{n-i}), for $i = 1, \ldots, n-1$, it checks that (i) $f(k_{n-i}) = k_{n-i+1}$, and (ii) $V(k_{n-i}, m_i, t_i) = \textsf{accept}$. If so, it immediately accepts the message m_i, without any delay. Note, however, that the sender needs to know the contents of m_i one time slot before m_i is transmitted; otherwise it cannot compute t_i at the right time.

(a) What is the security property that the MAC system (S, V) must satisfy for this construction to be secure? Write out the explicit security game, and argue that if a MAC system satisfies your definition then an attacker cannot replace the message m_i in time slot i with a fake message $m_i' \neq m_i$ for all $i = 1, \ldots, n$.

(b) Show that if H is a hash function defined over $(\mathcal{K} \times \mathcal{M},\ \mathcal{T})$ that is modeled as a random oracle, and f is one way on n iterates, then the derived MAC system $\mathcal{I}_H$ satisfies your definition from part (a) assuming $1/|\mathcal{K}|$ and $1/|\mathcal{T}|$ are negligible.

Discussion: Assuming that $\mathcal{K}$ is large, the best attack on the system from part (b) requires about $|\mathcal{T}|$ queries to the hash function H to succeed with constant probability. In GNSS the attacker only has a few seconds to mount this attack, so that it may be sufficient to set $\mathcal{T} := \{0, 1\}^{80}$. Then each tag is 80 bits, instead of 32 bits as in Section 14.6. Hence, authentication data is increased from 160 bits to 208 bits per time slot. The benefit is immediate authentication.

Chapter 15

Elliptic curve cryptography and pairings

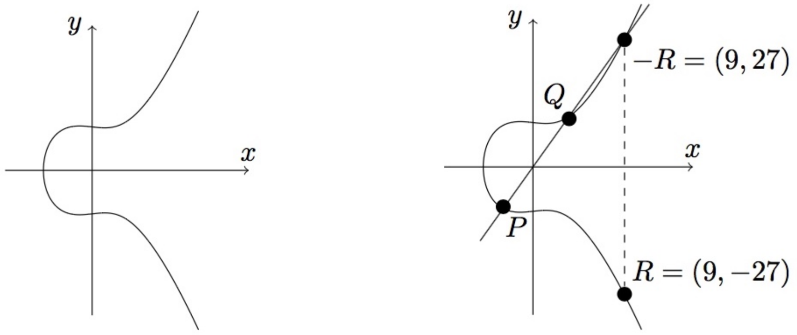
In previous chapters we saw many applications of the discrete log, CDH, and DDH assumptions in a finite cyclic group $\mathbb{G}$. Our primary example for the group $\mathbb{G}$ was the multiplicative group (or subgroup) of integers modulo a sufficiently large prime p. This group is problematic for a number of reasons, most notably because the discrete log problem in this group is not sufficiently difficult. The best known algorithm, called **the general number field sieve** (GNFS), discussed in Chapter 16, runs in time $\exp(\tilde{O}((\log p)^{1/3}))$. It was used in 2019 to solve a discrete log problem modulo a general 795-bit prime. This algorithm is the reason why, in practice, we must use a prime p whose size is at least 2048 bits. High security applications must use even larger primes. Arithmetic modulo such large primes is slow and greatly increases the cost of deploying cryptosystems that use this group.

Several other families of finite cyclic groups with an apparent hard discrete log have been proposed. Of all these proposals, the group of points of an elliptic curve over a prime finite field is the most suitable for practice, and is widely used on the Internet today. The best known discrete log algorithm in an elliptic curve group of size q runs in time $O(\sqrt{q})$. This means that to provide security comparable to AES-128, it suffices to use a group of size $q \approx 2^{256}$ so that the time to compute discrete log is $\sqrt{q} \approx 2^{128}$. The group operation uses a small number of arithmetic operations modulo a 256-bit prime, which is considerably faster than arithmetic modulo a 2048-bit prime.

Additional structure. As we will see, certain elliptic curve groups have an additional structure, called a **pairing**, that is enormously useful in cryptography. We will see many examples of encryption and signature schemes built using pairings. These systems exhibit powerful properties that are beyond what can be built using the multiplicative group of the integers modulo a prime. Some examples include aggregate signatures, broadcast encryption, functional encryption, and many others. The bulk of the chapter is devoted to exploring the world of pairing-based cryptography.

15.1 The group of points of an elliptic curve

Elliptic curves come up naturally in several branches of mathematics. Here we will follow their development as a branch of arithmetic (the study of rational numbers). Our story begins with Diophantus, a greek mathematician who lived in Alexandria in the third century AD. Diophantus



(a) The curve (b) Adding $P = (-1, -3)$ and $Q = (1, 3)$

Figure 15.1: The curve $y^2 = x^3 - x + 9$ over the reals (not drawn to scale)

was interested in the following problem: given a bivariate polynomial equation, $f(x, y) = 0$, find rational points satisfying the equation. A rational point is one where both coordinates are rational, such as $(1/2,\ 1/3)$, but not $(1, \sqrt{2}\,)$. Diophantus wrote a series of influential books on this subject, called the *Arithmetica*, of which six survived. Fourteen centuries later, Fermat scribbled his famous conjectures in the margins of a latin translation of the Arithmetica. An insightful short book by Bashmakova [11] describes Diophantus' ideas in a modern mathematical language.

Much of the Arithmetica studies integer and rational solutions of quadratic equations. However, in a few places Diophantus considers problems of higher degree. Problem 24 of book 4, which is the first occurrence of elliptic curves in mathematics, looks at a cubic equation. The problem is equivalent to the following question: find rational points $(x, y) \in \mathbb{Q}^2$ satisfying the equation

$$y^2 = x^3 - x + 9. \tag{15.1}$$

Fig. 15.1 shows a plot of this curve over the real numbers. We do not know what compelled Diophantus to ask this question, but it is a good guess that he would be shocked to learn that the method he invented to answer it now secures Internet traffic for billions of people worldwide.

One can easily verify that the six integer points $(0, \pm 3)$, $(1, \pm 3)$, $(-1, \pm 3)$ are on the curve (15.1). Diophantus wanted to find more rational points on this curve.

He proceeded to derive new rational points from the six he already had. Here is one way to do it, which is slightly different from what Diophantus did. Let $P := (-1, -3)$ and $Q := (1, 3)$, both satisfying (15.1). Let's look at the line passing through P and Q, as shown in Fig. 15.1b. One can easily verify that this line is simply $y = 3x$. It must intersect the curve $y^2 = x^3 - x + 9$ at exactly three points. To see why, observe that if we substitute $3x$ for y in (15.1) we obtain the univariate cubic equation $(3x)^2 = x^3 - x + 9$. We already know two rational roots of this cubic equation: $x_1 = -1$ from the point P, and $x_2 = 1$ from the point Q. It is not difficult to show that a cubic with rational coefficients that has two rational roots, must also have a third rational root x_3. In our case, this third rational root happens to be $x_3 = 9$. Setting $y_3 = 3x_3$ we obtain a new point on the curve (15.1), namely $(9, 27)$. For reasons that will become clear in a minute, we denote this point by $-R$. We get another point for free, $(9, -27)$, which we call R. More generally, for a point $T = (x, y)$ on the curve, we let $-T$ be the point $-T := (x, -y)$.

This technique for building rational points is called the **chord method**. It is quite general: given two distinct rational points U and V on the curve, where $U \neq -V$, we can pass a line through them, and this line must intersect the curve at a third rational point W. For example, applying this to the points P and R gives two new points $(-\frac{56}{25}, \frac{3}{125})$ and $(-\frac{56}{25}, -\frac{3}{125})$.

The chord method was re-discovered several times over the centuries, but it finally stuck with the work of Poincaré on algebraic curves [130]. Poincaré likened the process of constructing a new rational point from two known rational points to an addition operation in a group. Specifically, for distinct points U and V on the curve, with $U \neq -V$, let W be the point on the curve obtained by passing a line through U and V and finding its third point of intersection with the curve. Then Poincaré defines the sum of U and V, denoted $U \boxplus V$, as

$$U \boxplus V := -W. \tag{15.2}$$

Fig. 15.1b shows this addition rule applied to the points P and Q. Their sum $P \boxplus Q$ is the point $R = (9, -27)$. Defining addition as in (15.2) makes this operation associative, when it is well defined. Recall that associativity means that $(U \boxplus V) \boxplus W = U \boxplus (V \boxplus W)$.

We will show in the next section how to enhance this addition rule so that the set of points on the curve becomes a group. Some of the most beautiful results in number theory, and some of the deepest open problems, come from trying to understand the properties of the group of rational points on elliptic curves [8].

Going back to Diophantus, his approach for finding rational points on (15.1) is a variation of the method we just saw. Instead of passing a line through two distinct points, Diophantus chose to pass a tangent to the curve at one of the known points. Say we pass a tangent at the point $P = (-1, -3)$. As before, it is not difficult to show that on a cubic curve with rational coefficients, if (x_1, y_1) is a rational point with $y_1 \neq 0$, then the tangent at (x_1, y_1) must intersect the curve at exactly one more point T, and this point must also be rational. In our case, the tangent at $P = (-1, -3)$ is the line $y = -\frac{1}{3}x - \frac{10}{3}$. It intersects the curve at the point P and at the point $(\frac{19}{9}, -\frac{109}{27})$ which is indeed rational. This method, called the **tangent method**, is another way to build a new rational point from a given rational point (x_1, y_1), when $y_1 \neq 0$. As we will see, it corresponds to adding the point P to itself, namely computing $P \boxplus P$.

15.2 Elliptic curves over finite fields

The curve (15.1) is an example of an elliptic curve defined over the rationals. For cryptographic applications we are mostly interested in elliptic curves over finite fields. For simplicity, we only consider elliptic curves defined over a finite field $\mathbb{F}_p$ where $p > 3$ is a prime.

Definition 15.1. *Let $p > 3$ be a prime. An* ***elliptic curve*** *E defined over $\mathbb{F}_p$ is an equation*

$$y^2 = x^3 + ax + b, \tag{15.3}$$

where $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \neq 0$. We write $E/\mathbb{F}_p$ to denote the fact that E is defined over $\mathbb{F}_p$.

The condition $4a^3 + 27b^2 \neq 0$ ensures that the equation $x^3 + ax + b = 0$ does not have a double root. This is needed to avoid certain degeneracies.

The set of points on the curve. Let $E/\mathbb{F}_p$ be an elliptic curve, and let $e \geq 1$. We say that a point (x_1, y_1), where $x_1, y_1 \in \mathbb{F}_{p^e}$, is a point **on the curve** E if (x_1, y_1) satisfies the curve equation (15.3). When $e = 1$ the point (x_1, y_1) is defined over the base field $\mathbb{F}_p$. When $e > 1$ the point is defined over an extension of $\mathbb{F}_p$.

The curve includes an additional "special" point $\mathcal{O}$ called **the point at infinity**. Its purpose will become clear in a minute. We write $E(\mathbb{F}_{p^e})$ to denote the set of all points on the curve E that are defined over $\mathbb{F}_{p^e}$, including the point $\mathcal{O}$.

For example, consider the curve $E:\ y^2 = x^3 + 1$ defined over $\mathbb{F}_{11}$. Then

$$E(\mathbb{F}_{11}) = \Big\{\mathcal{O},\ \ (-1,0),\ \ (0,\pm1),\ \ (2,\pm3),\ \ (5,\pm4),\ \ (7,\pm5),\ \ (9,\pm2)\Big\} \tag{15.4}$$

This curve has 12 points in $\mathbb{F}_{11}$ and we write $|E(\mathbb{F}_{11})| = 12$.

A classic result of Hasse shows that $|E(\mathbb{F}_{p^e})| = p^e + 1 - t$ for some integer t in the interval $|t| \le 2\sqrt{p^e}$. This shows that the number of points on $E(\mathbb{F}_{p^e})$ is close to $p^e + 1$. The set $E(\mathbb{F}_p)$ in example (15.4) has exactly $p + 1$ points (so that $t = 0$).

Remark 15.1 (Point counting). A beautiful algorithm due to Schoof [141] can be used to compute the number of points in $E(\mathbb{F}_{p^e})$ in time polynomial in $\log(p^e)$. Hence, $|E(\mathbb{F}_{p^e})|$ can be computed efficiently even for a large prime p. Elkies and Atkin showed how to reduce the running time, and the resulting point counting method is called the Schoof-Elkies-Atkin algorithm, or simply, the **SEA algorithm**. □

The addition law. As we discussed in the previous section, there is a natural group law defined on the points of an elliptic curve. The group operation is written additively using the symbol "$\boxplus$" to denote point addition. We define the point at infinity $\mathcal{O}$ to be the identity element: for all $P \in E(\mathbb{F}_{p^e})$ we define $P \boxplus \mathcal{O} = \mathcal{O} \boxplus P = P$.

Now, let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points in $E(\mathbb{F}_{p^e})$. The sum $P \boxplus Q = (x_3, y_3)$ is defined using one of the following three rules:

- if $x_1 \ne x_2$ we use the chord method. Let $s_{\mathrm{c}} := \frac{y_1 - y_2}{x_1 - x_2}$ be the slope of the chord through the points P and Q. Define
$$x_3 := s_{\mathrm{c}}^2 - x_1 - x_2 \qquad \text{and} \qquad y_3 := s_{\mathrm{c}}(x_1 - x_3) - y_1.$$
- if $x_1 = x_2$ and $y_1 = y_2$ (i.e., $P = Q$), but $y_1 \ne 0$, we use the tangent method. Let $s_{\mathrm{t}} := \frac{3x_1^2 + a}{2y_1}$ be the slope of the tangent at P. Define
$$x_3 := s_{\mathrm{t}}^2 - 2x_1 \qquad \text{and} \qquad y_3 := s_{\mathrm{t}}(x_1 - x_3) - y_1.$$
- if $x_1 = x_2$ and $y_1 = -y_2$ then define $P \boxplus Q := \mathcal{O}$.

This addition law makes the set $E(\mathbb{F}_{p^e})$ into a group. The identity element is the point at infinity. Every point $\mathcal{O} \ne P = (x_1, y_1) \in E(\mathbb{F}_{p^e})$ has an additive inverse, namely $-P = (x_1, -y_1)$. Finally, it can be shown that this addition law is associative. The group law is clearly commutative, $P \boxplus Q = Q \boxplus P$ for all $P, Q \in E(\mathbb{F}_{p^e})$, making this an abelian group.

As in any group, for a point $P \in E(\mathbb{F}_{p^e})$ we write $2P := P \boxplus P$, $\ 3P := P \boxplus P \boxplus P$, and more generally, $\alpha P := (\alpha - 1)P \boxplus P$ for any positive integer α. Note that αP can be computed using at most $2 \log_2 \alpha$ group operations using the repeated squaring algorithm (Appendix A).

15.2.1 Montgomery and Edwards curves

Equation (15.3) for an elliptic curve is called the **Weierstrass form** of the curve. There are many equivalent ways of describing an elliptic curve and some are better suited for computation than the Weierstrass form. We give two examples.

Montgomery curves. A **Montgomery curve** $E/\mathbb{F}_p$ in the variables u and v is written as

$$Bv^2 = u^3 + Au^2 + u$$

for some $A, B \in \mathbb{F}_p$ where $B(A^2-4) \neq 0$. This curve equation can be easily changed into Weierstrass form via the change of variables $u := Bx - A/3$ and $v := By$. The number of points on a Montgomery curve, $|E(\mathbb{F}_{p^e})|$, is always divisible by four. Exercise 15.4 explores the computational benefit of Montgomery curves. They will also come up in the next section.

Some Weierstrass curves defined over $\mathbb{F}_p$ cannot be put into Montgomery form over $\mathbb{F}_p$ by a change of variables. For example, if the Weierstrass curve has an odd number of points over $\mathbb{F}_p$, then it has no Montgomery form over $\mathbb{F}_p$. Such Weierstrass curves cannot benefit from the speedup associated with Montgomery curves. The curve P256, discussed in the next section, is an example of such a curve.

Edwards curves. Another way to describe an elliptic curve $E/\mathbb{F}_p$ is in Edwards form, which is

$$x^2 + y^2 = 1 + dx^2y^2$$

where $d \in \mathbb{F}_p$ satisfies $d \neq 0, 1$. Again, this curve can be put into Weierstrass form via a simple rational change of variable. The beauty of the Edwards form is that the chord and tangent addition law is extremely easy to describe. For points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ in $E(\mathbb{F}_{p^e})$, we define

$$P \boxplus Q := \left(\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2},\ \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2} \right).$$

That's it. There is no need for three separate rules. The identity is the point $\mathcal{O} = (0, 1)$ and the inverse of a point (x_1, y_1) is $(-x_1, y_1)$. The points $(\pm 1, 0)$ have order four, which means that the number of points on an Edwards curve, $|E(\mathbb{F}_{p^e})|$, is always divisible by four.

The simplicity of the addition law on an Edwards curve makes it easier to resist timing attacks on the implementation. It also leads to a very fast implementation.

15.3 Elliptic curve cryptography

Let $E/\mathbb{F}_p$ be an elliptic curve and let $E(\mathbb{F}_p)$ be the group of points on this curve that are defined over $\mathbb{F}_p$. We know that $E(\mathbb{F}_p)$ is a finite abelian group. Suppose that it also happens to be cyclic, meaning that $E(\mathbb{F}_p)$ is generated by some point $P \in E(\mathbb{F}_p)$. We can now ask about the complexity of problems like discrete log, computational Diffie-Hellman (CDH), and decision Diffie-Hellman (DDH) in the group $E(\mathbb{F}_p)$. Once we establish that discrete log, CDH, and DDH are hard in this group, we can instantiate all the constructions we covered in the previous several chapters using the group $E(\mathbb{F}_p)$. The resulting systems are called **elliptic curve cryptosystems**.

Let us review the discrete log problem in $E(\mathbb{F}_p)$. Let P be a point in $E(\mathbb{F}_p)$ of order q, meaning that $qP = \mathcal{O}$. The discrete log problem in $E(\mathbb{F}_p)$ is as follows: We are given P and q, along with another point $Q := \alpha P$, for some $\alpha \in \mathbb{Z}_q$. Here αP is the point obtained by adding P to itself α times using the addition law. We want to compute α, the discrete log of Q base P in $E(\mathbb{F}_p)$.

In Chapter 16 we will show that there is a generic discrete log algorithm that computes discrete log in every cyclic group of order q using at most $O(\sqrt{q})$ group operations. We want a curve $E/\mathbb{F}_p$, or a set of curves, where the best known discrete log algorithm in $E(\mathbb{F}_p)$ runs in about the same time as this generic algorithm. That is, there is no known algorithm that can compute discrete log in $E(\mathbb{F}_p)$ much faster than $\sqrt{q}$ time, where $q := \big|E(\mathbb{F}_p)\big|$.

Insecure curves. For many elliptic curves $E/\mathbb{F}_p$ over a prime field $\mathbb{F}_p$, the best known discrete log algorithm runs in time $O(\sqrt{q})$, where $q := \big|E(\mathbb{F}_p)\big|$. However, there are several notable exceptions where discrete log is much easier. Three examples are:

- In Section 16.1.2.2 we will show that if $\big|E(\mathbb{F}_p)\big|$ is composite, and all its prime factors are less than some bound $q_{\max}$, then there is an algorithm to compute discrete log in $E(\mathbb{F}_p)$ in time $\tilde{O}(\sqrt{q_{\max}})$. In particular, if all the prime factors of $\big|E(\mathbb{F}_p)\big|$ are small, say less than 2^{80}, then the discrete log problem in $E(\mathbb{F}_p)$ is easy in practice. For this reason we will only use a curve $E/\mathbb{F}_p$ if $\big|E(\mathbb{F}_p)\big|$ is equal to either q, $4q$, or $8q$ for some prime number q. This will ensure that the algorithm from Section 16.1.2.2 runs in time $O(\sqrt{q})$.

- When $\big|E(\mathbb{F}_p)\big| = p$, the discrete log problem in $E(\mathbb{F}_p)$ is solvable in polynomial time. These curves are called *anomalous curves* and should never be used in cryptographic applications.

- Suppose there is a small integer $\tau > 0$ such that $|E(\mathbb{F}_p)|$ divides $p^\tau - 1$. Then discrete log on $E(\mathbb{F}_p)$ reduces to discrete log in the finite field $\mathbb{F}_{p^\tau}$, as explained in Section 15.4. Discrete log in $\mathbb{F}_{p^\tau}$ can be solved using variants of the general number field sieve (GNFS) discrete log algorithm discussed in Section 16.2. For example, if τ is small, say $\tau = 2$, and p is a 256-bit prime, then discrete log in $E(\mathbb{F}_p)$ can be solved efficiently: first reduce a given discrete log problem to $\mathbb{F}_{p^2}$, and then apply GNFS in $\mathbb{F}_{p^2}$. Since $\mathbb{F}_{p^2}$ is a finite field of size about 2^{512}, the discrete log problem in $\mathbb{F}_{p^2}$ can be solved in a reasonable amount of time (several hours). To prevent this attack we need to ensure that p^τ is sufficiently large so that GNFS in $\mathbb{F}_{p^\tau}$ is infeasible.

To avoid these pitfalls, many implementations use a curve from a fixed collection of curves that have been vetted and shown to not be vulnerable to the attacks listed above. The three most widely used curves are called **secp256r1**, **secp256k1**, and **Curve25519**. We discuss these curves in the next two subsections.

Remark 15.2 (Discrete log in $E(\mathbb{F}_{p^e})$***).*** Let $E/\mathbb{F}_p$ be an elliptic curve defined over the prime finite field $\mathbb{F}_p$. Asymptotically, discrete log in the group $E(\mathbb{F}_{p^e})$, for $e \geq 3$, is not as hard as one would like. For $e \geq 2$ there is a discrete log algorithm in the group $E(\mathbb{F}_{p^e})$ that runs in conjectured time $\tilde{O}(p^{2-2/e})$ [70]. For $e = 3$ this evaluates to $\tilde{O}(p^{1.33})$, for $e = 4$ it evaluates to $\tilde{O}(p^{1.5})$, and so on. This is asymptotically faster than the generic discrete log algorithm that, for $e = 3$ and $e = 4$, runs in time $O(p^{1.5})$ and $O(p^2)$, respectively. For this reason, the group $E(\mathbb{F}_{p^e})$ can only be used if p is sufficiently large so as to make this algorithm impractical. Taking $p \geq 2^{256}$ is sufficient. □

15.3.1 The curves secp256r1 and secp256k1

Two widely used elliptic curves, called **secp256r1** and **secp256k1**, are specified in a standard called SEC2, where SEC is an acronym for "standards for efficient cryptography." Both curves are defined over a 256-bit prime field, hence the "256" in their names. The 'r' in secp256r1 signifies that the curve is a random curve, meaning that it was generated by a certain sampling procedure. The 'k' in secp256k1 signifies that the curve is a *Koblitz curve* as explained below. The curve secp256r1 is widely used in Internet protocols, while secp256k1 is widely used in blockchain systems.

The curve secp256r1. This curve was approved by the U.S. National Institute of Standards (NIST) for federal government use in a standard published in 1999. The NIST standard refers to this curve as **Curve P256**. All implementations of TLS 1.3 are required to support this curve for Diffie-Hellman key exchange. It is the only mandatory curve in the TLS 1.3 standard, as discussed in Section 21.10.

The curve secp256r1 is defined as follows:

- The curve is defined over the prime $p_{\mathrm{r}} := 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. The special structure of this prime is meant to improve the performance of arithmetic modulo p_{r}.
- The curve has the Weierstrass form $y^2 = x^3 - 3x + b$ where $b \in \mathbb{F}_{p_{\mathrm{r}}}$ written as a 255-bit number in hexadecimal is:

  $b :=$ `5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b`.
- The number of points on this curve is a prime number. Recall that it must be close to p_{r}.
- The standard also specifies a point G_{r} that generates the entire group $E(\mathbb{F}_{p_{\mathrm{r}}})$, where E is the curve secp256r1.

How was the odd looking parameter b selected? The reality is that we do not really know. The standard lists an unexplained constant called a **seed** S. This seed was provided as input to a public deterministic algorithm, that generated the parameter b. This process was designed to select a random curve that resists the known discrete log attacks. The problem is that we do not know for sure how the seed S was selected. An organization that wants to use secp256r1 might worry that S was chosen adversarially so that discrete log on the resulting curve is easy. Currently we do not know how to select such a seed even if we wanted to, so this concern is just an intriguing speculation. As far as we can tell, secp256r1 is a fine curve to use. It is widely used in Internet protocols.

The curve secp256k1. The second curve in the SEC2 standard is called secp256k1. This curve was selected for the digital signature scheme in the Bitcoin blockchain. Subsequent blockchains inherited the same curve. One reason for this choice is the concern over the choice of seed in the curve secp256r1 (also known as Curve P256) discussed in the previous paragraph.

The curve secp256k1 is defined as follows:

- The curve is defined over the prime $p_{\mathrm{k}} := 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$. Again, the special structure of this prime is meant to improve the performance of arithmetic modulo p_{k}.
- The curve has the Weierstrass form $y^2 = x^3 + 7$. Note that there are no unexplained large constants in the Weierstrass form.
- The number of points on this curve is a prime number. Recall that it must be close to p_{k}.
- The standard also specifies a point G_{k} that generates the entire group $E(\mathbb{F}_{p_{\mathrm{k}}})$, where E is the curve $y^2 = x^3 + 7$.

Properties of secp256k1. The curve secp256k1 has two interesting properties. First, it is a Koblitz curve, which means that there is a useful map defined on it. To explain what that means, let E be the curve $y^2 = x^3 + 7$, and let $p := p_{\mathrm{k}}$ be the prime over which secp256k1 is defined. Let q be the size of $E(\mathbb{F}_p)$. It so happens that $p \equiv 1 \pmod 3$ and therefore there exists $1 \neq \omega \in \mathbb{F}_p$ such that $\omega^3 = 1$.

Now, consider the map

$$\phi : \mathbb{F}_p^2 \to \mathbb{F}_p^2 \quad \text{defined by} \quad \phi(x, y) := (\omega x, y). \tag{15.5}$$

It is easy to verify that if $(x, y) \in \mathbb{F}_p^2$ is a point in $E(\mathbb{F}_p)$ then $\phi(x, y) = (\omega x, y)$ is also in $E(\mathbb{F}_p)$. Indeed, if (x, y) satisfies $y^2 = x^3 + 7$ in $\mathbb{F}_p$ then so does $(\omega x, y)$ because $\omega^3 = 1$. Let us also define $\phi(\mathcal{O}) := \mathcal{O}$ so that ϕ is a map from $E(\mathbb{F}_p)$ to $E(\mathbb{F}_p)$.

The general theory of elliptic curves tells us that the map ϕ in (15.5) must be a group homomorphism. Therefore, since $E(\mathbb{F}_p)$ is a cyclic group, there must be a constant λ in $\mathbb{Z}_q$ such that for all P in $E(\mathbb{F}_p)$ we have

$$\phi(P) = \lambda \cdot P,$$

where $\lambda \cdot P$ is the point in $E(\mathbb{F}_p)$ obtained by adding P to itself λ times.

We can determine the constant λ as follows. We know that for all non-zero $P \in E(\mathbb{F}_p)$, the three points P, $\phi(P)$, and $\phi^2(P) := \phi(\phi(P))$ all have the same y-coordinate, and therefore they are colinear — they all lie on the horizontal line through P. By definition of the addition law, this means that their sum must be zero. In other words, for all $P \in E(\mathbb{F}_p)$ we have

$$\mathcal{O} = P + \phi(P) + \phi^2(P) = (1 + \lambda + \lambda^2) \cdot P.$$

We can therefore conclude that $1 + \lambda + \lambda^2 = 0$ in $\mathbb{Z}_q$. Hence, λ must be one of the two non-trivial cube roots of unity in $\mathbb{Z}_q$. Indeed, $q \equiv 1 \pmod 3$ so that a non-trivial cube root exists in $\mathbb{Z}_q$. Thus, we know the value of λ.

How is this map useful? The fact that the map ϕ is so easy to calculate can be used to speed up multiplication on the curve secp256k1. Let $\alpha \in \mathbb{Z}_q$ and suppose that we want to calculate $\alpha \cdot P$ for some $P \in E(\mathbb{F}_p)$. For most α in $\mathbb{Z}_q$ it is possible to find integers τ_0, τ_1, τ_2 such that

$$\alpha = \tau_0 + \tau_1\lambda + \tau_2\lambda^2 \quad \text{and } \left|\tau_i\right| \leq 2q^{1/3} \text{ for } i = 0, 1, 2.$$

Then to compute αP it suffices to compute

$$\alpha P = \tau_0 \cdot P + \tau_1 \cdot \phi(P) + \tau_2 \cdot \phi^2(P). \tag{15.6}$$

This equality converts a multiplication by α into three multiplications where the multipliers τ_0, τ_1, τ_2 are much smaller than α. The expression on the right hand side of (15.6) can computed about two to three times faster than computing αP directly. This is done using a fast multi-exponentiation algorithm discussed in Appendix A. The end result is that the map ϕ can be used to speed up multiplication on secp256k1.

Finally, we note that the curve secp256k1 has another useful property that is a bit magical. If E is the curve $y^2 = x^3 + 7$ and $p := p_{\mathrm{k}}$ is the prime over which secp256k1 is defined, then

$$\text{the group } E(\mathbb{F}_p) \text{ has size } q \qquad \text{and} \qquad \text{the group } E(\mathbb{F}_q) \text{ has size } p.$$

This property is useful in certain zero knowledge proof systems, but we will not make use of it here.

Security of discrete log on secp256r1 and secp256k1. Because the primes p_{r} and p_{k} are close to 2^{256}, the number of points on both curves is also close to 2^{256}. Therefore, computing discrete log on these curves using a generic discrete log algorithm takes approximately 2^{128} group operations. We assume that no algorithm can compute discrete log much faster than that. The intent is that discrete log on both curves (as well as CDH and DDH on both curves) should be at least as hard as breaking AES-128. Consequently, if one is aiming for the level of security provided by AES-128, then either curve can be used for Diffie-Hellman key exchange, public-key encryption, and digital signatures.

Curves with higher security. Some high-security applications use AES-256 to encrypt plaintext data. In these cases, one should use an elliptic curve with a higher security parameter. One option is another curve from SEC2 called **secp521r1**, whose size is approximately 2^{521}. It is defined over the Mersenne prime $p = 2^{521} - 1$. Discrete log on this curve is believed to require at least 2^{256} group operations. This curve is also approved by the U.S. national institute of standards (NIST) for federal government use.

15.3.2 A security twist

Every elliptic curve $E/\mathbb{F}_p$ has a related curve $\tilde{E}/\mathbb{F}_p$ called the **twist** of E. Let $c \in \mathbb{F}_p$ be some quadratic non-residue in $\mathbb{F}_p$. If E is the curve $y^2 = x^3 + ax + b$ then its twist $\tilde{E}$ is the curve $cy^2 = x^3 + ax + b$. It is a simple exercise to show that $\left|E(\mathbb{F}_p)\right| + \left|\tilde{E}(\mathbb{F}_p)\right| = 2p + 2$. Since the number of points on $E(\mathbb{F}_p)$ is $p + 1 - t$, it follows that the number of points on $\tilde{E}(\mathbb{F}_p)$ must be $\tilde{n} := p + 1 + t$.

We say that a curve $E/\mathbb{F}_p$ is **twist secure** if discrete log is intractable on both $E(\mathbb{F}_p)$ and $\tilde{E}(\mathbb{F}_p)$. For $E/\mathbb{F}_p$ to be twist secure we need, at the very least, that both $n = |E(\mathbb{F}_p)|$ and $\tilde{n} = |\tilde{E}(\mathbb{F}_p)|$ are prime numbers, or are small multiples of large primes.

Why do we need twist security? Consider a system where Bob has a secret key $\alpha \in \mathbb{Z}_q$. Under normal operation, anyone can send Bob a point $P \in E(\mathbb{F}_p)$ and Bob will respond with the point αP. One system that operates this way is the oblivious PRF in Section 11.6.2. Before responding, Bob had better check that the given point P is in $E(\mathbb{F}_p)$; otherwise, the response that Bob sends back could compromise his secret key α, as discussed in Exercise 15.1 (see also Remark 12.1 where a similar issue came up). Checking that a point $P = (x_1, y_1)$ satisfies the curve equation is quite simple and efficient. However some implementations use the optimizations outlined in Exercises 15.2 and 15.4, where Bob is only sent the x-coordinate of P. The y-coordinate is not needed and is never sent. In this case, checking that the given $x_1 \in \mathbb{F}_p$ is valid requires a full exponentiation to confirm that $x_1^3 + ax_1 + b$ is a quadratic residue in $\mathbb{F}_p$ (see Appendix A.2.3). Suppose Bob skips this expensive check. Then an attacker could send Bob an $x_1 \in \mathbb{F}_p$ that is the x-coordinate of a point $\tilde{P}$ on the twist $\tilde{E}(\mathbb{F}_p)$. Bob would then respond with the x-coordinate of $\alpha\tilde{P}$ in $\tilde{E}(\mathbb{F}_p)$. If discrete log in $\tilde{E}(\mathbb{F}_p)$ were easy, this response would expose Bob's secret key α. Hence, if Bob skips the group membership check, we must ensure, at the very least, that discrete log in $\tilde{E}(\mathbb{F}_p)$ is intractable so that $\alpha\tilde{P}$ does not expose α. Twist security is meant to ensure exactly that.

The curves secp256r1 and secp256k1 were not designed to be twist secure. The size of the twist of secp256r1 is divisible by $34905 = 3 \times 5 \times 13 \times 179$. Consequently, discrete log on the twist is $\sqrt{34905} \approx 187$ times easier than on secp256r1 (see Section 16.1.2.2). Similarly, for secp256k1 the size of the twist is divisible by $3^2 \times 13^2 \times 3319 \times 22639$, and consequently, discrete log on the twist is $\approx 2^{18}$ times easier than on `secp256k1`. These are important facts to remember, but not a significant enough concern to disqualify secp256r1 or secp256k1.

15.3.3 Curve25519

Curve25519 is designed to support an optimized group operation and to be twist secure. The curve is defined over the prime $p := 2^{255} - 19$, which is the reason for its name. This p is the largest prime less than 2^{255} and this enables fast arithmetic in $\mathbb{F}_p$.

It is easiest to describe Curve25519 as a Montgomery curve, namely a curve in the form $E : By^2 = x^3 + Ax^2 + x$ for some $A, B \in \mathbb{F}_p$ where $p > 3$. Exercise 15.4 shows that these curves support a fast multiplication algorithm to compute αP from P where $P \in E(\mathbb{F}_p)$ and $\alpha \in \mathbb{Z}$. We noted earlier that the number of points $|E(\mathbb{F}_p)|$ on a Montgomery curve is always a multiple of four.

Curve25519 presented as a Montgomery curve is simply

$$y^2 = x^3 + 486662 \cdot x^2 + x.$$

The number of points on this curve is eight times a prime. We say that the curve has **cofactor** eight. The curve is generated by a point $P = (x_1, y_1)$ where $x_1 = 9$. For completeness, we note that Curve25519 can also be presented as the Edwards curve $x^2 + y^2 = 1 + (121665/121666)x^2y^2$.

Why the constant 486662? When defining a Montgomery curve, the smaller A is, the faster the group operation becomes, as explained in Exercise 15.4. For the best performance we need $(A - 2)/4$ to be small [17]. Dan Bernstein, who designed this curve, chose the smallest possible A so that the curve is secure against the known discrete log attacks. He also made sure that the order of the curve and the order of its twist are either four times a prime or eight times a prime. Dan Bernstein writes [18]:

> The smallest positive choices for A are 358990, 464586, and 486662. I rejected $A = 358990$ because one of its primes is slightly smaller than 2^{252}, raising the question of how standards and implementations should handle the theoretical possibility of a user's secret key matching the prime; discussing this question is more difficult than switching to another A. I rejected 464586 for the same reason. So I ended up with $A = 486662$.

This explanation is a bit more satisfying than the unexplained constants in the random curve secp256r1, and the unexplained prime in secp256k1.

15.4 Pairing based cryptography

Up until now, we used elliptic curves as an efficient group where the discrete log problem and its variants, CDH and DDH, are believed to be difficult. This group makes it possible to instantiate efficiently many of the schemes described in earlier chapters. We now show that certain elliptic curves have an additional structure, called a **pairing**. The pairing enables a world of new schemes that could not be built from discrete log groups without this additional structure. The resulting schemes make up an important area called **pairing based cryptography**.

To present pairing based schemes we focus on the new capabilities enabled by a pairing and abstract away the details of the elliptic curve group. As in earlier chapters, we will write the group operation multiplicatively. This is a little different from the first part of this chapter where we used additive notation for the group operation to be consistent with traditional elliptic curve mathematical notation.

Definition 15.2. *Let $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_T$ be three cyclic groups of prime order q where $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. A **pairing** is an efficiently computable function $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ satisfying the following properties:*

1. *bilinear: for all $u, u' \in \mathbb{G}_0$ and $v, v' \in \mathbb{G}_1$ we have*

$$e(u \cdot u',\ v) = e(u,v) \cdot e(u',v) \qquad \textit{and} \qquad e(u,\ v \cdot v') = e(u,v) \cdot e(u,v'),$$

2. *non-degenerate: $g_T := e(g_0, g_1)$ is a generator of $\mathbb{G}_T$.*

*When $\mathbb{G}_0 = \mathbb{G}_1$ we say that the pairing is a **symmetric pairing**. We refer to $\mathbb{G}_0$ and $\mathbb{G}_1$ as the **pairing groups** or source groups, and refer to $\mathbb{G}_T$ as the **target group**.*

Bilinearity implies the following central property of pairings that will be used in all our constructions: for all $\alpha, \beta \in \mathbb{Z}_q$ we have

$$e(g_0^\alpha,\ g_1^\beta) = e(g_0, g_1)^{\alpha \cdot \beta} = e(g_0^\beta,\ g_1^\alpha). \tag{15.7}$$

This equality follows from the useful equalities $e(g_0^\alpha,\ g_1^\beta) = e(g_0, g_1^\beta)^\alpha = e(g_0^\alpha, g_1)^\beta$ which are themselves a direct consequence of bilinearity. We note that in cyclic groups such as $\mathbb{G}_0$ and $\mathbb{G}_1$, the relation (15.7) is equivalent to the bilinearity property stated in Definition 15.2.

The non-degeneracy requirement in Definition 15.2 ensures that the pairing e does not always output $1 \in \mathbb{G}_\mathrm{T}$ for all inputs. Such a pairing is bilinear, but not very useful. In addition, we will want the discrete log problem to be difficult in the groups $\mathbb{G}_0$ and $\mathbb{G}_1$. Many of the constructions will require additional problems to be hard, such as CDH and certain variants of DDH.

Direct consequences. A pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_\mathrm{T}$ has a number of consequences for the groups $\mathbb{G}_0$ and $\mathbb{G}_1$. First, when $\mathbb{G}_0 = \mathbb{G}_1$ the decision Diffie-Hellman (DDH) problem in $\mathbb{G}_0$ is easy. To see why, observe that given a triple $(u, v, w) = (g_0^\alpha, g_0^\beta, g_0^\gamma) \in \mathbb{G}^3$, we can test if $\gamma = \alpha \cdot \beta$ in $\mathbb{Z}_q$ by checking if

$$e(u, v) = e(g_0, w). \tag{15.8}$$

By (15.7) this equality holds if and only if $e(g_0, g_0)^{\alpha\beta} = e(g_0, g_0)^\gamma$, which holds if and only if $\gamma = \alpha \cdot \beta$. This is precisely when (u, v, w) is a DH-triple. Hence, DDH is easy.

One implication of this is that the multiplicative ElGamal encryption scheme from Exercise 11.5 is not semantically secure in a symmetric pairing group, and should not be instantiated in such a group. Similarly, the full ElGamal encryption scheme $\mathcal{E}_\mathrm{EG}$ from Section 11.5 cannot be proved secure using the analysis in Section 11.5.2, which is based on DDH. Nevertheless, $\mathcal{E}_\mathrm{EG}$ can still be proved secure based on CDH in the random oracle model using the analysis in Section 11.5.1.

For an **asymmetric pairing**, where $\mathbb{G}_0 \neq \mathbb{G}_1$, it may still be possible that the DDH assumption holds in both $\mathbb{G}_0$ and $\mathbb{G}_1$. In fact, for the most efficient asymmetric pairings from elliptic curves, the DDH assumption is believed to hold in both $\mathbb{G}_0$ and $\mathbb{G}_1$.

Second, computing discrete log in either $\mathbb{G}_0$ or $\mathbb{G}_1$ is no harder than computing discrete log in the target group $\mathbb{G}_\mathrm{T}$. To see why, suppose we are given $u_0 = g_0^\alpha \in \mathbb{G}_0$ and are asked to compute its discrete log $\alpha \in \mathbb{Z}_q$. To do so, we compute $u := e(u_0, g_1)$ and set $g_\mathrm{T} := e(g_0, g_1)$. Observe that $u \in \mathbb{G}_\mathrm{T}$ satisfies $u = (g_\mathrm{T})^\alpha$, and we can find α by computing the discrete log of u base g_T in the group $\mathbb{G}_\mathrm{T}$. Hence, if the discrete log problem in $\mathbb{G}_\mathrm{T}$ is easy, then so is the discrete log problem in $\mathbb{G}_0$. A similar argument applies to $\mathbb{G}_1$. Consequently, for discrete log to be hard in either $\mathbb{G}_0$ or $\mathbb{G}_1$, we must choose the groups so that discrete log in the target group $\mathbb{G}_\mathrm{T}$ is also hard.

Constructing pairings from elliptic curves. A pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_\mathrm{T}$ is typically constructed from an elliptic curve $E/\mathbb{F}_p$. The most natural and efficient pairings from elliptic curves are asymmetric, where $\mathbb{G}_0 \neq \mathbb{G}_1$. For this reason we will describe all the pairing-based systems in this chapter using asymmetric pairings.

A pairing constructed from an elliptic curve $E/\mathbb{F}_p$, has groups $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_\mathrm{T}$ with the following properties:

- $\mathbb{G}_0$ is an order q subgroup of $E(\mathbb{F}_p)$, for some prime q,
- $\mathbb{G}_1$ is an order q subgroup of $E(\mathbb{F}_{p^d})$, for some $d > 0$, where $\mathbb{G}_1 \cap \mathbb{G}_0 = \{\mathcal{O}\}$,
- $\mathbb{G}_\mathrm{T}$ is an order q multiplicative subgroup of the finite field $\mathbb{F}_{p^d}$.

The integer $d > 0$ is called the **embedding degree** of the curve. Clearly d needs to be small so that the elements in $\mathbb{G}_1$ and $\mathbb{G}_\mathrm{T}$ can be represented efficiently. If d were large we could not write down elements in $\mathbb{G}_1$ and $\mathbb{G}_\mathrm{T}$. Elliptic curves where d is small, say $d \leq 16$, are said to be **pairing friendly elliptic curves**.

Because $\mathbb{G}_0$ is defined over the base field $\mathbb{F}_p$ and $\mathbb{G}_1$ is defined over a larger field $\mathbb{F}_{p^d}$, the representation of elements in $\mathbb{G}_0$ is typically much shorter than elements in $\mathbb{G}_1$. This plays an important role in choosing what to put in each group. For example, to minimize ciphertext length we will prefer that group elements that appear in the ciphertext live in $\mathbb{G}_0$.

The pairing function e on an elliptic curve comes from an algebraic pairing called the **Weil pairing**. This pairing can be efficiently evaluated using an algorithm due to Victor Miller, and called Miller's algorithm. In practice one uses variants of the Weil pairing called the Tate and Ate pairings, for which Miller's algorithm is more efficient. We refer to [67] for more information about the definition of these pairings and how to evaluate them.

The most commonly used curves in practice are called bn256 and bls381. Both curves have embedding degree $d = 12$ and the group order q is a 256-bit prime. Both curves are defined over a prime field: bn256 is defined over a 256-bits prime field, while bls381 is defined over a 381-bits prime field. The curve bls381 is believed to provide better security because discrete log in $\mathbb{G}_\mathrm{T}$ is harder, but arithmetic in bls381 is slower because of the larger prime.

Pairing performance. Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_\mathrm{T}$ be a pairing derived from a pairing friendly elliptic curve $E/\mathbb{F}_p$. All three groups have prime order q. Computing the pairing takes $O(\log q)$ arithmetic operations in $\mathbb{F}_p$, similar to exponentiation in $\mathbb{G}_0$ or $\mathbb{G}_1$. However, in practice, computing the pairing takes longer than an exponentiation (the exact overhead depends on the specific implementation). We therefore try to minimize the number of pairing computations in the systems we design. To give a sense for the relative costs, Table 15.1 gives running times for exponentiation and pairing on the curve bn256. It shows that a pairing is about ten times slower than an exponentiation in $\mathbb{G}_0$.

Many optimizations are possible when implementing a pairing.

- First, when one of the pairing inputs is fixed, it is possible to speed up the pairing computation significantly using pre-computation. Specifically, suppose one needs to compute many pairings $e(u, v_i)$ for $i = 1, \ldots, n$ where $u \in \mathbb{G}_0$ is fixed. Then it is possible to build a table that depends only on u. Once the table is constructed, computing each pairing is much faster than computing it without the pre-computed table. This speed-up is analogous to the pre-computation speed-up for fixed-base exponentiation discussed in Appendix A.2.4.

		time (milliseconds)
exponentiation:	in $\mathbb{G}_0$	0.22
	in $\mathbb{G}_1$	0.44
	in $\mathbb{G}_\mathrm{T}$	0.95
pairing:	$\mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_\mathrm{T}$	2.32

Table 15.1: Benchmarks for pairings on the curve `bn256`[1].

- Second, a product of pairings $\prod_{i=1}^{n} e(u_i, v_i) \in \mathbb{G}_\mathrm{T}$ can be computed faster than computing the n pairings one by one. For example, the final step in a pairing computation is raising the computed value to a fixed large power. This final exponentiation can be aggregated across all the pairings in the product and done only once for the entire product. Other optimizations are also possible.

Often these two optimizations combine to give a significant speed-up [143, 144].

15.5 Signature schemes from pairings

Pairings enable many new advanced encryption and signature schemes, as well as other primitives. In this section, we present several new signature schemes, based on pairings, with interesting properties.

In Section 15.5.1, we introduce the *BLS signature scheme*. One nice feature of BLS signatures is compactness — a signature is a single short group element. Another nice feature of the scheme is that it supports *signature aggregation*, which is a means by which many signatures can be publicly compressed into a single, short *aggregate* signature. This is a new feature for a signature scheme that we have not seen before. It is discussed in Sections 15.5.2 and 15.5.3. Finally, in Section 15.5.4, we present two signature schemes based on pairings that can be proved secure without relying on the random oracle model. Note that except for the hash-based signature schemes presented in Chapter 14 (specifically, in Section 14.6), all of the other signatures schemes we have seen so far rely for their security on the random oracle model. The pairing based schemes we present in Section 15.5.4 produce much shorter signatures than those produced by the hash-based schemes.

15.5.1 The BLS signature scheme

Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_\mathrm{T}$ be a pairing where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_\mathrm{T}$ are cyclic groups of prime order q, and where $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. We will also need a hash function H that maps messages in a finite set $\mathcal{M}$ to elements in $\mathbb{G}_0$.

The BLS signature scheme, denoted $\mathcal{S}_{\text{BLS}} = (G, S, V)$, has message space $\mathcal{M}$ and works as follows:

- $G()$: The key generation algorithm runs as follows

$$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g_1^{\alpha} \in \mathbb{G}_1.$$

  The public key is $pk := u$, and the secret key is $sk := \alpha$.

[1] Benchmarked on a 2.4 GHz Intel i5 520M using the Miracl library. Numbers are from the Miracl documentation.

- $S(sk, m)$: To sign a message $m \in \mathcal{M}$ using a secret key $sk = \alpha \in \mathbb{Z}_q$, do:

$$\sigma \leftarrow H(m)^\alpha \in \mathbb{G}_0, \quad \text{output } \sigma.$$

- $V(pk, m, \sigma)$: To verify a signature $\sigma \in \mathbb{G}_0$ on a message $m \in \mathcal{M}$, using the public key $pk = u \in \mathbb{G}_1$, output accept if

$$e\big(H(m),\ u\big) = e(\sigma,\ g_1).$$

It is easy to verify that a valid signature is always accepted: for all public keys $u \leftarrow g_1^\alpha$ and messages $m \in \mathcal{M}$, a valid signature $\sigma \leftarrow H(m)^\alpha$ will be accepted by the verifier because

$$e\big(H(m),\ u\big) = e\big(H(m),\ g_1^\alpha\big) = e\big(H(m)^\alpha,\ g_1\big) = e(\sigma, g_1).$$

As presented, signatures live in $\mathbb{G}_0$ and public keys live in $\mathbb{G}_1$. In the previous section we mentioned that elements in $\mathbb{G}_0$ have a shorter representation compared to elements in $\mathbb{G}_1$. Therefore, as described, the scheme is optimized for short signatures — a signature is a single element in $\mathbb{G}_0$. One can equally optimize for short public keys by defining a dual scheme where the roles of $\mathbb{G}_0$ and $\mathbb{G}_1$ are reversed: signatures live in $\mathbb{G}_1$ and public keys live in $\mathbb{G}_0$. The security analysis below applies equally well to this dual scheme.

Unique signatures. The $\mathcal{S}_{\text{BLS}}$ scheme is a **unique signature scheme**: for a given public key, every message $m \in \mathcal{M}$ has a *unique* signature $\sigma \in \mathbb{G}_0$ that will be accepted as valid for m by the verification algorithm. This means that if $\mathcal{S}_{\text{BLS}}$ is secure, it must also be strongly secure in the sense of Definition 13.3.

Security. We can prove that $\mathcal{S}_{\text{BLS}}$ is secure when $H : \mathcal{M} \to \mathbb{G}_0$ is modeled as a random oracle. We will need a minor variation of the computational Diffie-Hellman (CDH) assumption which adapts the standard CDH assumption to the case when two groups are used.

Attack Game 15.1 (co-CDH). For a given adversary $\mathcal{A}$, the attack game runs as follows.

- The challenger computes

$$\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u_0 \leftarrow g_0^\alpha, \quad u_1 \leftarrow g_1^\alpha, \quad v_0 \leftarrow g_0^\beta, \quad z_0 \leftarrow g_0^{\alpha\beta}$$

  and gives the tuple (u_0, u_1, v_0) to the adversary. Note that α is used twice, once in $\mathbb{G}_0$ and once $\mathbb{G}_1$.

- The adversary outputs some $\hat{z}_0 \in \mathbb{G}_0$.

We define $\mathcal{A}$'s **advantage in solving the co-CDH problem for** e, denoted $\text{coCDH}\mathsf{adv}[\mathcal{A}, e]$, as the probability that $\hat{z}_0 = z_0$. $\square$

Here we used z_0 to denote the Diffie-Hellman secret $z_0 := g_0^{\alpha\beta}$. More generally, in attack games in this chapter we will always use the variable z for the unknown value that the adversary is trying to compute or distinguish from random.

Definition 15.3 (co-CDH assumption). *We say that the **co-CDH** assumption holds for the pairing e if for all efficient adversaries $\mathcal{A}$ the quantity* $\text{coCDH}\mathsf{adv}[\mathcal{A}, e]$ *is negligible.*

When e is a symmetric pairing we have $\mathbb{G}_0 = \mathbb{G}_1$ and $g_0 = g_1$, in which case the co-CDH assumption is identical to the standard CDH assumption. We can now prove security of $\mathcal{S}_{\text{BLS}}$.

Theorem 15.1. *Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a pairing and let $H : \mathcal{M} \to \mathbb{G}_0$ be a hash function. Then the derived BLS signature scheme $\mathcal{S}_{\text{BLS}}$ is a secure signature scheme assuming co-CDH holds for e, and H is modeled as a random oracle.*

> *In particular, let $\mathcal{A}$ be an efficient adversary attacking $\mathcal{S}_{\text{BLS}}$ in the random oracle version of Attack Game 13.1. Moreover, assume that $\mathcal{A}$ issues at most Q_{s} signing queries. Then there exists an efficient co-CDH adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{SIG}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{BLS}}] \leq 2.72 \cdot (Q_{\text{s}} + 1) \cdot \text{coCDH}\mathsf{adv}[\mathcal{B}, e]. \tag{15.9}$$

Proof idea. The proof mimics the proof of security for the RSA-FDH signature scheme (Theorem 13.4). Adversary $\mathcal{B}$ is given a tuple $(u_0 = g_0^{\alpha},\ u_1 = g_1^{\alpha},\ v_0 = g_0^{\beta})$ where $\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q$, as in the co-CDH attack game. It needs to compute $z_0 := g_0^{\alpha\beta} = v_0^{\alpha}$. The adversary begins by sending the BLS public key $pk := u_1$ to the signature forger $\mathcal{A}$.

Next, the forger makes a sequence of Q_{s} signing queries, and Q_{ro} queries to H. For $j = 1, 2, \ldots$ our adversary $\mathcal{B}$ responds to hash query number j for $H(m_j)$ by choosing $\rho_j \xleftarrow{\text{R}} \mathbb{Z}_q$ and setting $H(m_j) := g_0^{\rho_j}$. This enables $\mathcal{B}$ to answer all of $\mathcal{A}$'s signing queries. The signature on m_j is simply $\sigma_j = u_0^{\rho_j}$ because $\sigma_j := H(m_j)^{\alpha} = g_0^{\rho_j \alpha} = u_0^{\rho_j}$. Our $\mathcal{B}$ can compute this σ_j using the given u_0 in the co-CDH challenge. This is the reason we include u_0 in the challenge.

Eventually $\mathcal{A}$ outputs a valid signature forgery (m, σ) where $\sigma = H(m)^{\alpha}$. We know that $H(m)$ was defined either when $\mathcal{A}$ issued an explicit query for $H(m)$, or when $\mathcal{A}$ output the final forgery (note that m cannot appear in a signature query from $\mathcal{A}$). Thus, $H(m)$ was defined in one of the $(Q_{\text{ro}}+1)$ queries to H. Suppose that $\mathcal{A}$ issued a query for $H(m)$ in hash query number ν, for some $1 \leq \nu \leq Q_{\text{ro}} + 1$. One proof strategy is to have $\mathcal{B}$ try to guess ν at the beginning of the game. $\mathcal{B}$ chooses a random ω in $\{1, \ldots, Q_{\text{ro}} + 1\}$ at the beginning of the game, and responds to hash query number ω with $H(m_{\omega}) := v_0$. If $\mathcal{B}$ was lucky and it chose ω correctly, so that $\omega = \nu$, then the signature σ on $m = m_{\omega}$ output by $\mathcal{A}$ satisfies $\sigma = H(m)^{\alpha} = H(m_{\omega})^{\alpha} = v_0^{\alpha} = z_0$, which is the value $\mathcal{B}$ needs to win the co-CDH game.

This proof strategy for computing z_0 works fine, but incurs a factor $(Q_{\text{ro}}+1)$ loss in $\mathcal{B}$'s success probability compared to $\mathcal{A}$'s, because $\mathcal{B}$ needs to guess ν correctly at the beginning of the game. This loss can be reduced to $2.72 \cdot (Q_{\text{s}} + 1)$, as claimed in (15.9), using exactly the same argument used in Lemma 13.6 for the RSA function. See Exercise 15.5 for the CDH analogue of Lemma 13.6. Using Exercise 15.5 the proof of security is identical to the proof of security for RSA-FDH in Section 13.4.2 and gives the bounds stated in (15.9). □

15.5.2 Signature aggregation

The BLS signature scheme is remarkably agile. It is quite easy to build a threshold signature scheme from it (Section 22.2.2), a blind signature scheme (Exercise 15.7), and even generate the secret key distributively among several parties. In this section we focus on a feature of BLS called signature aggregation, which is a new capability that we have not seen before.

Signature aggregation refers to the ability to compress signatures on different messages, possibly issued using different signing keys, into a single short aggregate signature. The short aggregate convinces the verifier that all the input messages were properly signed.

Definition 15.4. *An aggregate signature scheme $\mathcal{SA}$ is a signature scheme with two additional efficient algorithms A and VA:*

- *A signature aggregation algorithm $A(\boldsymbol{pk}, \boldsymbol{\sigma})$: takes as input two equal length vectors, a vector of public keys $\boldsymbol{pk} = (pk_1, \ldots, pk_n)$ and a vector of signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$. It outputs an aggregate signature σ_{ag}.*
- *A deterministic aggregate verification algorithm $VA\big(\boldsymbol{pk}, \boldsymbol{m}, \sigma_{\text{ag}}\big)$: takes as input two equal length vectors, a vector of public keys $\boldsymbol{pk} = (pk_1, \ldots, pk_n)$, a vector of messages $\boldsymbol{m} = (m_1, \ldots, m_n)$, and an aggregate signature σ_{ag}. It outputs either* accept *or* reject*.*
- *The scheme is correct if for all $\boldsymbol{pk} = (pk_1, \ldots, pk_n)$, $\boldsymbol{m} = (m_1, \ldots, m_n)$, and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$, if $V(pk_i, m_i, \sigma_i) = \mathsf{accept}$ for $i = 1, \ldots, n$ then $\Pr\big[VA\big(\boldsymbol{pk}, \boldsymbol{m}, A(\boldsymbol{pk}, \boldsymbol{\sigma})\big) = \mathsf{accept}\big] = 1$.*

The length n of all the input vectors is less than some upper-bound parameter N.

The aggregate signature σ_{ag} output by algorithm A should be short. Its length should be about the same as the length of a single signature, no matter how many signatures are being aggregated. Notice that anyone can aggregate a given set of signatures using the aggregation algorithm A. Aggregation requires no knowledge of the secret signing keys and requires no interaction with the signers. In particular, aggregation can take place long after the signers issued the signatures.

Signature aggregation comes up in many real-world scenarios. For example, in a certificate chain containing multiple certificates issued by different authorities, one can aggregate all the signatures in the chain into a single short aggregate. This shrinks the overall length of the certificate chain. Another example is a distributed system where multiple parties periodically sign and publish their view of the state of the system. These signatures are recorded on a long-term log. The log manager can compress the log by aggregating all these signatures into a single short aggregate. This efficiency improvement is especially useful in blockchain systems [60].

Before we discuss aggregation security let us first look at a simple attempt at aggregating BLS signatures. Recall that the scheme uses a hash function $H : \mathcal{M} \to \mathbb{G}_0$. A BLS public key is a group element $pk = g_1^{\alpha} \in \mathbb{G}_1$, and a signature is a group element $\sigma \in \mathbb{G}_0$. Consider the following simple aggregation procedure A that aggregates n signatures by simply computing their product. More precisely, the scheme works as follows.

The aggregate signature scheme $\mathcal{SA}_{\text{BLS}} = (\mathcal{S}_{\text{BLS}}, A, VA)$:

- $A\big(\boldsymbol{pk} \in \mathbb{G}_1^n,\ \boldsymbol{\sigma} \in \mathbb{G}_0^n\big) := \big\{\sigma_{\text{ag}} \leftarrow \sigma_1 \cdot \sigma_2 \cdots \sigma_n \in \mathbb{G}_0, \quad \text{output } \sigma_{\text{ag}} \in \mathbb{G}_0\ \big\}$.
- $VA\big(\boldsymbol{pk} \in \mathbb{G}_1^n,\ \boldsymbol{m} \in \mathcal{M}^n,\ \sigma_{\text{ag}}\big)$: accept if
$$e(\sigma_{\text{ag}}, g_1) = e\big(H(m_1), pk_1\big) \cdots e\big(H(m_n), pk_n\big). \tag{15.10}$$

To see why the verification algorithm accepts an aggregate σ_{ag} of valid signatures observe that

$$e(\sigma_{\text{ag}}, g_1) = e(\sigma_1 \cdots \sigma_n,\ g_1) = \prod_{i=1}^{n} e(\sigma_i, g_1) = \prod_{i=1}^{n} e\big(H(m_i)^{\alpha_i}, g_1\big) =$$
$$= \prod_{i=1}^{n} e\big(H(m_i), g_1^{\alpha_i}\big) = \prod_{i=1}^{n} e\big(H(m_i), pk_i\big)$$

so that the equality in (15.10) holds.

Remark 15.3. Although we described the aggregation procedure as a one-time process, the BLS aggregation mechanism can be done incrementally. One can aggregate a number of signatures to obtain an aggregate. Later aggregate more signatures into the aggregate, and so on. Moreover, given an aggregate σ_{ag} and some signature σ that was aggregated into σ_{ag}, one can remove σ from the aggregate by computing ${\sigma_{\text{ag}}}' \leftarrow \sigma_{\text{ag}}/\sigma$. □

Remark 15.4. In some cases we can end up with an aggregate signature σ_{ag} that contains multiple copies of a signature σ. For example, suppose server S_1 aggregates signatures from a number of users and sends the aggregate ${\sigma_{\text{ag}}}^{(1)}$ to server S_3. Server S_2 aggregates signatures from another set of users and also sends the aggregate ${\sigma_{\text{ag}}}^{(2)}$ to server S_3. Server S_3 aggregates the two aggregates ${\sigma_{\text{ag}}}^{(1)}$ and ${\sigma_{\text{ag}}}^{(2)}$ to obtain the final aggregate $\sigma_{\text{ag}} \leftarrow {\sigma_{\text{ag}}}^{(1)} \cdot {\sigma_{\text{ag}}}^{(2)}$. If Alice sends its signature σ to both S_1 and S_2 then σ will be included twice in the final aggregate σ_{ag}. This does not hurt performance or security, however, a verifier needs to know the multiplicity of every signature in the aggregate to correctly verify σ_{ag} using (15.10). □

Remark 15.5 (Faster verification). Verifying an aggregate of n signatures in (15.10) requires computing $n + 1$ pairings. This can be simplified when all the signed messages in the aggregate are the same, namely $m_1 = m_2 = \cdots = m_n = m$. This happens when multiple parties sign the same message m, and there is a need to aggregate all these signatures into a single aggregate σ_{ag}. In this case the verification equation (15.10) for verifying the aggregate σ_{ag} simplifies to:

$$e(\sigma_{\text{ag}}, g_1) \stackrel{?}{=} e\big(H(m), apk\big) \qquad \text{where} \qquad apk := pk_1 \cdots pk_n \in \mathbb{G}_1. \tag{15.11}$$

This requires only two pairings, independent of n. The quantity apk is called an **aggregate public key**. If the set of signers in the aggregate is fixed in advance, then this short apk can be pre-computed, and the verifier does not need to store the keys $pk_1, \ldots, pk_n$. □

15.5.2.1 The rogue public key attack

While the BLS aggregation scheme $\mathcal{SA}_{\text{BLS}}$ works correctly, it is completely insecure as is. To explain the attack, consider a user Bob whose public key is $pk_{\text{B}} := u_{\text{B}} \in \mathbb{G}_1$. The adversary wants to make it seem as if Bob signed some message $m \in \mathcal{M}$ which Bob did not sign. To do so, the adversary creates a public key pk_{adv} by choosing a random $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$ and computing

$$u \leftarrow g_1^{\alpha}, \quad pk_{\text{adv}} \leftarrow u/u_{\text{B}} \in \mathbb{G}_1. \tag{15.12}$$

The private key corresponding to this public key is $\alpha_{\text{adv}} := \alpha - \alpha_{\text{B}}$, where α_{B} is Bob's secret key. The adversary does not know this secret key, but it can still claim that pk_{adv} is its public key. We say that pk_{adv} is a **rogue public key**.

The aggregate public key for pk_{B} and pk_{adv} is $apk := pk_{\text{B}} \cdot pk_{\text{adv}} = u_{\text{B}} \cdot (u/u_{\text{B}}) = u = g_1^{\alpha}$, and the adversary knows $\alpha \in \mathbb{Z}_q$. Therefore it can compute $\sigma_{\text{ag}} := H(m)^{\alpha} \in \mathbb{G}_0$. This σ_{ag} is a valid aggregate signature on the message m issued by the public keys pk_{B} and pk_{adv}. Indeed, the verification equation (15.11) holds for σ_{ag}:

$$e(\sigma_{\text{ag}}, g_1) = e\big(H(m)^{\alpha}, g_1\big) = e\big(H(m), g_1^{\alpha}\big) = e\big(H(m), u\big) = e\big(H(m), apk\big).$$

Hence, the adversary created an aggregate signature σ_{ag} that makes it look as if Bob signed m. This is a complete break of the scheme.

15.5.2.2 Definition of secure aggregation

Our security definition must properly model rogue public key attacks as above. In the security game the adversary is asked to create a valid aggregate signature over a number of public keys, where the adversary controls all but one of the public keys. Before outputting the aggregate forgery, the adversary can issue signing queries for the one public key that it does not control.

Attack Game 15.2 (Aggregate signature security). For a given aggregate signature scheme $\mathcal{SA} = (G, S, V, A, VA)$ with message space $\mathcal{M}$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger runs $(pk, sk) \xleftarrow{R} G()$ and sends pk to $\mathcal{A}$.
- $\mathcal{A}$ queries the challenger. For $i = 1, 2, \ldots$, the ith *signing query* is a message $m^{(i)} \in \mathcal{M}$. The challenger computes $\sigma_i \xleftarrow{R} S(sk, m^{(i)})$, and then gives σ_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs a candidate aggregate forgery $(\boldsymbol{pk}, \boldsymbol{m}, \sigma_{\text{ag}})$ where $\boldsymbol{pk} = (pk_1, \ldots, pk_n)$ and $\boldsymbol{m} = (m_1, \ldots, m_n) \in \mathcal{M}^n$.

We say that the adversary wins the game if the following conditions hold:

- $VA(\boldsymbol{pk}, \boldsymbol{m}, \sigma_{\text{ag}}) = \mathsf{accept}$,
- there is at least one $1 \leq j \leq n$ such that (1) $pk_j = pk$, and (2) $\mathcal{A}$ did not issue a signing query for m_j, meaning that $m_j \notin \{m^{(1)}, m^{(2)}, \ldots\}$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{SA}$, denoted $\text{ASIG}\mathsf{adv}[\mathcal{A}, \mathcal{SA}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 15.5. *We say that an aggregate signature scheme $\mathcal{SA}$ is secure if for all efficient adversaries $\mathcal{A}$, the quantity* $\text{ASIG}\mathsf{adv}[\mathcal{A}, \mathcal{SA}]$ *is negligible.*

15.5.2.3 Who signed? Strongly binding aggregation

Some aggregation schemes that are secure against forgery attacks (Definition 15.5), can be vulnerable to an issue caused by a group of colluding malicious signers. Let Alice, Bob, Carol, and David be a group of four signers. Suppose that by working together they can construct four distinct key pairs (pk_i, sk_i) for $i = 1, \ldots, 4$, a message m, and an aggregate signature σ_{ag} such that

$$VA\big((pk_1, pk_2),\ (m, m),\ \sigma_{\text{ag}}\big) = VA\big((pk_3, pk_4),\ (m, m),\ \sigma_{\text{ag}}\big) = \mathsf{accept}.$$

In other words, by working together the four of them caused a collision: the aggregate signature σ_{ag} is a valid signature on m with respect to both $\boldsymbol{pk} = (pk_1, pk_2)$ and $\boldsymbol{pk'} = (pk_3, pk_4)$. This can cause confusion over who signed m: was it Alice and Bob or was it Carol and David? We saw a similar issue in Section 13.1.1.1 where we discussed signer confusion.

Let us define a strong form of message and public key binding for signature aggregation.

Attack Game 15.3 (Strong binding). Let $\mathcal{SA} = (G, S, V, A, VA)$ be an aggregate signature scheme. A strong binding adversary $\mathcal{A}$ takes no input, and outputs $(\boldsymbol{pk}, \boldsymbol{m}, \boldsymbol{pk'}, \boldsymbol{m'}, \sigma_{\text{ag}})$. We say that $\mathcal{A}$ is successful if

$$(\boldsymbol{pk}, \boldsymbol{m}) \neq (\boldsymbol{pk'}, \boldsymbol{m'}) \qquad \text{and} \qquad VA(\boldsymbol{pk}, \boldsymbol{m}, \sigma_{\text{ag}}) = VA(\boldsymbol{pk'}, \boldsymbol{m'}, \sigma_{\text{ag}}) = \mathsf{accept}.$$

Define $\mathcal{A}$'s advantage with respect to $\mathcal{SA}$, denoted sbASIGadv$[\mathcal{A}, \mathcal{SA}]$, as the probability that $\mathcal{A}$ is successful. □

Definition 15.6. *We say that an aggregate signature scheme $\mathcal{SA}$ is **strongly binding** if for all efficient adversaries $\mathcal{A}$, the quantity* sbASIGadv$[\mathcal{A}, \mathcal{SA}]$ *is negligible.*

The definition ensures that one cannot find an aggregate signature σ_{ag} that is valid for two distinct pairs $(\boldsymbol{pk}, \boldsymbol{m})$ and $(\boldsymbol{pk}', \boldsymbol{m}')$.

Exercise 15.11(c) shows that every secure aggregate signature scheme can be made strongly binding. The cost is a slight increase in the size of the aggregate signature. As we will see, some secure aggregation schemes are strongly binding without modification. For others, if strong binding is needed, one can apply the augmentation from Exercise 15.11(c).

15.5.3 Secure BLS aggregation

Now that we understand the security requirements let's see how to enhance the simple aggregation procedure in (15.10) to properly aggregate BLS signatures. There are two primary methods to prevent rogue public keys, each designed for a different use case. We first describe both methods and then prove their security.

15.5.3.1 Method 1: message augmentation

The simplest approach to securely aggregating BLS signatures is to prepend the signing public key to every message being signed. Specifically, the modified aggregation scheme, denoted $\mathcal{SA}_{\text{BLS}}^{(1)}$, is the same as $\mathcal{SA}_{\text{BLS}}$ in (15.10) except that the signing algorithm now uses a hash function $H : \mathbb{G}_1 \times \mathcal{M} \to \mathbb{G}_0$ and is defined as

$$S(sk, m) := H(pk, m)^{\alpha} \quad \text{where} \quad sk = \alpha \in \mathbb{Z}_q \;\; \text{and} \;\; pk \leftarrow g_1^{\alpha}.$$

In effect, the message being signed is the pair $(pk, m) \in \mathbb{G}_1 \times \mathcal{M}$. The verification and aggregate verification algorithms are equally modified to hash the pairs (pk, m). Specifically, aggregate verification works as

$$\begin{aligned}
&VA\big(\boldsymbol{pk} \in \mathbb{G}_1^n,\ \boldsymbol{m} \in \mathcal{M}^n,\ \sigma_{\text{ag}}\big): \\
&\qquad\qquad \text{accept if} \quad e(\sigma_{\text{ag}}, g_1) = e\big(H(pk_1, m_1), pk_1\big) \ \cdots \ e\big(H(pk_n, m_n), pk_n\big) \\
&\qquad\qquad\qquad\qquad\;\; \text{and } pk_i \neq 1 \text{ for all } i = 1, \ldots, n.
\end{aligned}$$

We prove that this method is secure against forgery attacks in Section 15.5.3.3 below. Exercise 15.10 shows that this method is also strongly binding (Definition 15.6). The last line in algorithm VA is needed to ensure strong binding.

15.5.3.2 Method 2: proof of possession of the secret key

A downside of Method 1, where the public key is prepended to every message, is that when aggregating multiple signatures on the same message we cannot take advantage of the verification optimization in Remark 15.5. After the public keys are prepended, the messages being signed are no longer the same, and this prevents the verifier from taking advantage of the optimized verification test in (15.11). Here we show a different aggregation method that preserves this optimization, when the set of public keys is known in advance.

Recall that in the rogue public key attack (15.12), the adversary did not know the secret key corresponding to its rogue public key pk_{adv}. We can therefore prevent the attack by requiring every signer to prove possession of the secret key corresponding to its public key.

The modified aggregation scheme, denoted $\mathcal{SA}^{(2)}_{\text{BLS}}$, is the same as $\mathcal{SA}_{\text{BLS}}$ defined in (15.10) except that the key generation algorithm also generates a proof π to show that the signer has possession of the secret key. We attach this proof π to the public key, and it is checked during aggregate verification. In particular, the key generation and aggregate verification algorithms use an auxiliary hash function $H' : \mathbb{G}_1 \to \mathbb{G}_0$, and operate as follows:

- $G() := \left\{ \begin{array}{l} \alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g_1^\alpha \in \mathbb{G}_1, \quad \pi \leftarrow H'(u)^\alpha \in \mathbb{G}_0 \\ \text{output } pk := (u, \pi) \in \mathbb{G}_1 \times \mathbb{G}_0 \text{ and } sk := \alpha \in \mathbb{Z}_q \end{array} \right\}.$
- $VA\big(\boldsymbol{pk},\ \boldsymbol{m} \in \mathcal{M}^n,\ \sigma_{\text{ag}}\big)$: Let $\boldsymbol{pk} = (pk_1, \dots, pk_n) = \Big((u_1, \pi_1), \dots, (u_n, \pi_n)\Big)$ be n public keys, and let $\boldsymbol{m} = (m_1, \dots, m_n)$. Accept if
  - valid proofs: $e\big(\pi_i, g_1\big) = e\big(H'(u_i), u_i\big)$ for all $i = 1, \dots, n$, and
  - valid aggregate: $e(\sigma_{\text{ag}}, g_1) = e\big(H(m_1), u_1\big) \cdots e\big(H(m_n), u_n\big)$.

The new term $\pi = H'(u)^\alpha \in \mathbb{G}_0$ in the public key is used to prove that the public key owner is in possession of the secret key α. This π is a BLS signature on the public key $u \in \mathbb{G}_1$, but using the hash function H' instead of H. The aggregate verification algorithm first checks that all the terms $\pi_1, \dots, \pi_n \in \mathbb{G}_0$ in the given public keys are valid, and then verifies that the aggregate signature σ_{ag} is valid exactly as in $\mathcal{SA}_{\text{BLS}}$.

We prove security of this method against forgery attacks in Section 15.5.3.3 below. When using this scheme, it is important that the hash function H' be independent from the hash function H used to sign messages, otherwise the scheme is insecure (see Exercise 15.8). Of course, in practice, both H and H' can be derived from a single hash function like SHA3 using domain separation, where $H(x) := \text{SHA3}(0 \parallel x)$ and $H'(x) := \text{SHA3}(1 \parallel x)$.

Remark 15.6. The proof of possession π does not depend on the message being signed. Therefore, if the same public key pk is used to sign many messages, the verifier need only verify correctness of π once and record the fact that pk has been validated. There is no need to check π on every signature verification. □

Remark 15.7. One can slightly modify algorithm G to generate the proof of possession π as $\pi \leftarrow H'(u, id)^\alpha$, where H' takes as input $u \in \mathbb{G}_1$ along with the identity id of the user who owns this public key. Algorithm VA is similarly modified when verifying π. This prevents an attacker from claiming Alice's public key $pk = (u, \pi)$ as its own public key. The attacker does not know Alice's secret key, so there is limited harm that can be caused by this public key theft. However, since including id as an input to H' is cheap, we might as well do it. □

The benefit of $\mathcal{SA}^{(2)}_{\text{BLS}}$ is that when all the messages are the same, namely $m_1 = m_2 = \cdots = m_n$, we can take advantage of the verification optimization in (15.11) and replace the verification of σ_{ag} with an equality check that requires only two pairings. Of course, this assumes that all the proofs of possession have been previously verified.

To summarize, if one expects to mostly verify aggregate signatures on distinct messages or from fresh public keys, then $\mathcal{SA}^{(1)}_{\text{BLS}}$ is preferable. If the signatures being aggregated are usually signatures on the same message, and the set of public keys is known in advance, then $\mathcal{SA}^{(2)}_{\text{BLS}}$ is preferable.

Remark 15.8 (aggregating proofs of possession). A verifier often needs to store all the public keys and signatures it encounters to enable future auditing of its actions. Let $\big\{pk_i = (u_i, \pi_i)\big\}_{i=1}^{n}$ be a collection of n public keys. Since the proofs of possession $\pi_1, \ldots, \pi_n \in \mathbb{G}_0$ included in the public keys are themselves BLS signatures, it is tempting to aggregate all of them into a single value $\pi := \pi_1 \cdots \pi_n \in \mathbb{G}_0$ as in (15.10) and only store $(u_1, \ldots, u_n, \pi)$. This roughly halves the space needed to store the public keys $pk_1, \ldots, pk_n$. Anyone can verify the aggregate proof of possession π by checking that $e(\pi, g_1) = \prod_{i=1}^{n} e\big(H'(u_i), u_i\big)$. We would like to claim that if this equality holds, then an aggregate signature by a subset of the public keys can be trusted. Unfortunately, aggregating proofs of possession this way invalidates the proof of security for $\mathcal{SA}_{\text{BLS}}^{(2)}$ given in Theorem 15.2 below. In the proof we need every public key to include an explicit proof of possession in non-aggregated form. Nevertheless, we show in Exercise 15.9 that a slightly modified argument can prove that aggregating the proofs of possession $\pi_1, \ldots, \pi_n$ is secure in certain settings. □

Remark 15.9 (Strong binding). While the scheme $\mathcal{SA}_{\text{BLS}}^{(2)}$ is secure against forgery attacks, it is not strongly binding in the sense of Definition 15.6. We explore this in Exercise 15.11 where we also present a simple fix. Exercise 15.12 gives a very different aggregation scheme that is strongly binding without modification, and retains some of the performance benefits of $\mathcal{SA}_{\text{BLS}}^{(2)}$. □

15.5.3.3 Proving security of both aggregation methods

Let's now prove that the two aggregation methods, $\mathcal{SA}_{\text{BLS}}^{(1)}$ and $\mathcal{SA}_{\text{BLS}}^{(2)}$, presented in the previous two sections, are secure against forgery attacks as in Definition 15.5. The two schemes have very similar security proofs, but the proofs differ crucially at a few points. We prove security of each scheme, highlighting the places where the proofs differ.

To prove security we need a bit more structure on the pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\text{T}}$. Specifically, let $\psi : \mathbb{G}_1 \to \mathbb{G}_0$ be the unique group homomorphism that satisfies $\psi(g_1) = g_0$. To prove security we will need ψ to be efficiently computable. When e is a symmetric pairing, so that $\mathbb{G}_0 = \mathbb{G}_1$ and $g_0 = g_1$, the group homomorphism ψ is the identity function, which is trivially efficiently computable. Other pairing instantiations support an efficiently computable group homomorphism ψ from $\mathbb{G}_1$ to $\mathbb{G}_0$ called the *trace map*. However, for some optimized pairings, no efficiently computable homomorphism is known from $\mathbb{G}_1$ to $\mathbb{G}_0$. We will discuss this case in Remark 15.10, after we prove security.

We are now ready to prove security of $\mathcal{SA}_{\text{BLS}}^{(1)}$ and $\mathcal{SA}_{\text{BLS}}^{(2)}$.

Theorem 15.2 (Security of $\mathcal{SA}_{\text{BLS}}^{(1)}$ and $\mathcal{SA}_{\text{BLS}}^{(2)}$). *Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a pairing and let $\psi : \mathbb{G}_1 \to \mathbb{G}_0$ be an efficiently computable group homomorphism satisfying $\psi(g_1) = g_0$.*

(a) Let $H : \mathbb{G}_1 \times \mathcal{M} \to \mathbb{G}_0$ be a hash function. Then the derived $\mathcal{SA}_{\text{BLS}}^{(1)}$ signature aggregation scheme is secure, assuming co-CDH holds for e, and H is modeled as a random oracle.

(b) Let $H : \mathcal{M} \to \mathbb{G}_0$ and $H' : \mathbb{G}_1 \to \mathbb{G}_0$ be hash functions. Then the derived $\mathcal{SA}_{\text{BLS}}^{(2)}$ signature aggregation scheme is secure, assuming co-CDH holds for e, and H and H' are modeled as random oracles.

In particular, let $\mathcal{A}_1$ be an efficient adversary attacking $\mathcal{SA}_{\text{BLS}}^{(1)}$, and let $\mathcal{A}_2$ be an efficient adversary attacking $\mathcal{SA}_{\text{BLS}}^{(2)}$, in the random oracle version of Attack Game 15.2. Moreover, assume that $\mathcal{A}_1$ and $\mathcal{A}_2$ issue at most Q_{s} signing queries. Then there exist an efficient co-CDH adversary $\mathcal{B}$, *where $\mathcal{B}$ is an elementary wrapper around one of $\mathcal{A}_1$ or $\mathcal{A}_2$, such that for both $i = 1, 2$*

$$\mathrm{ASIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}_i, \mathcal{SA}^{(i)}_{\mathrm{BLS}}] \le 2.72 \cdot (Q_{\mathrm{s}} + 1) \cdot \mathrm{coCDH}\mathsf{adv}[\mathcal{B}, e]. \tag{15.13}$$

Proof idea of Theorem 15.2 part (a). First, we sketch the security proof for $\mathcal{SA}^{(1)}_{\mathrm{BLS}}$. Adversary $\mathcal{B}$ is given a tuple $(u_0 = g_0^{\alpha},\ u_1 = g_1^{\alpha},\ v_0 = g_0^{\beta})$ where $\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, as in the co-CDH attack game. It needs to compute $z_0 := g_0^{\alpha\beta} = v_0^{\alpha}$, which is equivalent to finding a $z_0 \in \mathbb{G}_0$ satisfying $e(z_0, g_1) = e(v_0, u_1)$. Adversary $\mathcal{B}$ computes z_0 by interacting with an aggregate forger $\mathcal{A}_1$ for $\mathcal{SA}^{(1)}_{\mathrm{BLS}}$.

Adversary $\mathcal{B}$ begins by sending the public key $pk := u_1 = g_1^{\alpha}$ to the aggregate forger $\mathcal{A}_1$. Next, $\mathcal{A}_1$ makes a sequence of queries: Q_{ro} hash queries to H, and Q_{s} signature queries. $\mathcal{B}$ responds to hash queries as in the proof of Theorem 15.1. It first chooses a random ω in $\{1, \dots, Q_{\mathrm{ro}} + 1\}$. Then for $j = 1, 2, \dots$, when $\mathcal{A}_1$ issues hash query number j for $H(pk^{(j)}, m^{(j)})$, adversary $\mathcal{B}$ responds as follows:

– if $j \ne \omega$ then $\mathcal{B}$ chooses $\rho_j \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ and sets $H(pk^{(j)}, m^{(j)}) := g_0^{\rho_j}$,

– if $j = \omega$ then $\mathcal{B}$ sets $H(pk^{(\omega)}, m^{(\omega)}) := v_0$.

These responses to hash queries enable $\mathcal{B}$ to respond to $\mathcal{A}_1$'s signature queries as in the proof of Theorem 15.1. Note that if $pk^{(\omega)} = u_1$ then $\mathcal{B}$ cannot respond to a signature query for $m^{(\omega)}$. However, if $\mathcal{B}$ guesses ω correctly then $\mathcal{A}_1$ will never issue a signature for $m^{(\omega)}$, as explained next.

Eventually $\mathcal{A}_1$ outputs a valid aggregate forgery:

$$\boldsymbol{pk} = (\tilde{u}_1, \dots, \tilde{u}_n), \quad \boldsymbol{m} = (m_1, \dots, m_n) \in \mathcal{M}^n, \quad \sigma_{\mathrm{ag}} \in \mathbb{G}_0 \tag{15.14}$$

where $\sigma_{\mathrm{ag}} \in \mathbb{G}_0$ is a valid aggregate forgery so that

$$e(\sigma_{\mathrm{ag}}, g_1) = e\big(H(\tilde{u}_1, m_1), \tilde{u}_1\big) \ \cdots \ e\big(H(\tilde{u}_n, m_n), \tilde{u}_n\big). \tag{15.15}$$

Moreover, the challenge public key $pk = u_1 = g_1^{\alpha}$ must appear at least once on the right hand side of (15.15), along with some message m, and $\mathcal{A}_1$ never issued a signature query for m.

We know that $H(u_1, m)$ was defined either when $\mathcal{A}_1$ issued an explicit hash query for $H(u_1, m)$, say at hash query number $1 \le \nu \le Q_{\mathrm{ro}}$, or it was defined when $\mathcal{A}_1$ output the final forgery, which we treat at hash query number $\nu = Q_{\mathrm{ro}} + 1$. If $\mathcal{B}$ guessed ν correctly at the beginning of the game, so that $\omega = \nu$, then $H(u_1, m) = H(pk^{(\omega)}, m^{(\omega)}) = v_0$. Moreover, $\mathcal{A}_1$ never issued a signature query for $m^{(\omega)}$, and therefore $\mathcal{B}$ can correctly answer all of $\mathcal{A}_1$'s signature queries.

So, suppose $H(u_1, m) = v_0$. Adversary $\mathcal{B}$ needs to compute the signature $\sigma := H(u_1, m)^{\alpha} = v_0^{\alpha} = z_0$. This signature σ is part of the aggregate forgery σ_{ag}, and $\mathcal{B}$ needs to extract it from σ_{ag}. It does so by dividing out all the signatures in σ_{ag} until only σ remains. To see how, let us separate the signatures aggregated into σ_{ag} into two types. Specifically, for $t = 1, \dots, n$ each pair $(\tilde{u}_t, m_t)$ in (15.14) is one of:

- Type I: $\tilde{u}_t = u_1$ and $m_t = m$. In this case, the signature on m_t with respect to the public key $\tilde{u}_t$ is $\sigma = z_0$, which is the signature that $\mathcal{B}$ needs to compute.

- Type II: $\tilde{u}_t \ne u_1$ or $m_t \ne m$. In this case $\mathcal{B}$ can compute itself the signature σ_t on m_t with respect to the public key $\tilde{u}_t$. To see why, recall that $H(\tilde{u}_t, m_t) = g_0^{\rho_t}$ and $\mathcal{B}$ knows $\rho_t \in \mathbb{Z}_q$. We claim that $\sigma_t = \psi(\tilde{u}_t)^{\rho_t}$. Indeed, if $\tilde{u}_t = g_1^{\alpha_t}$ for some $\alpha_t \in \mathbb{Z}_q$ then

$$\sigma_t = \psi(\tilde{u}_t)^{\rho_t} = \psi\big(g_1^{\alpha_t}\big)^{\rho_t} = \psi(g_1)^{\alpha_t \cdot \rho_t} = g_0^{\alpha_t \cdot \rho_t} = H(\tilde{u}_t, m_t)^{\alpha_t}, \tag{15.16}$$

  and hence σ_t is the unique signature on m_t with respect to $\tilde{u}_t$. Moreover, $\mathcal{B}$ can compute σ_t because ψ is efficiently computable.

This shows that $\mathcal{B}$ can compute itself all the signatures that were aggregated into σ_{ag}, except for the challenge signature σ.

Let $d \geq 1$ be the number of times that the pair (u_1, m) appears in (15.14), that is, d is the number pairs of type I. Let $s \in \mathbb{G}_0$ be the product of all the computed signatures σ_t of type II above, and note that $\mathcal{B}$ can efficiently compute s. Each σ_t of type II is a valid signature, and therefore $e(\sigma_t, g_1) = e\big(H(\tilde{u}_t, m_t), \tilde{u}_t\big)$. Then, by re-ordering terms we can re-write (15.15) as

$$e(\sigma_{\mathrm{ag}}, g_1) = e\big(H(u_1, m), u_1\big)^d \cdot e(s, g_1) = e(v_0, u_1)^d \cdot e(s, g_1) = e(z_0, g_1)^d \cdot e(s, g_1) = e(z_0^d s, g_1).$$

This implies that $\sigma_{\mathrm{ag}} = z_0^d \cdot s$ and therefore $z_0 = (\sigma_{\mathrm{ag}}/s)^{1/d}$. Our $\mathcal{B}$ outputs $(\sigma_{\mathrm{ag}}/s)^{1/d}$ as the required co-CDH solution.

This proof strategy works fine, but incurs a factor $(Q_{\mathrm{ro}} + 1)$ loss in $\mathcal{B}$'s success probability, because $\mathcal{B}$ needs to guess ν correctly at the beginning of the game. This loss can be reduced to $2.72 \cdot (Q_{\mathrm{s}} + 1)$ as claimed in (15.13) using Exercise 15.5, as we did in Theorem 15.1. □

Proof idea of Theorem 15.2 part (b). Next, we sketch the proof of security for $\mathcal{SA}_{\mathrm{BLS}}^{(2)}$. Adversary $\mathcal{B}$ is given a tuple $(u_0 = g_0^{\alpha},\ u_1 = g_1^{\alpha},\ v_0 = g_0^{\beta})$ where $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$, as in the co-CDH attack game. It needs to compute $z_0 := g_0^{\alpha\beta} = v_0^{\alpha}$ by interacting with an aggregate forger $\mathcal{A}_2$ for $\mathcal{SA}_{\mathrm{BLS}}^{(2)}$.

Adversary $\mathcal{B}$ begins by sending the public key $pk := (u_1, \pi)$ to the forger $\mathcal{A}_2$, where $\pi := H'(u_1)^{\alpha}$. To compute this π, $\mathcal{B}$ defines $H'(u_1) := g_0^{\tau}$ for $\tau \xleftarrow{R} \mathbb{Z}_q$, and sets $\pi \leftarrow u_0^{\tau}$. Then $\pi = H'(u_1)^{\alpha}$.

Next, the forger makes a sequence of signature queries and hash queries to H and H'.

- H query: $\mathcal{B}$ responds to H queries as in the proof of Theorem 15.1. First, $\mathcal{B}$ chooses a random ω in $\{1, \ldots, Q_{\mathrm{ro}} + 1\}$. Next, for $j = 1, 2, \ldots$ it responds to query number j for $H(m^{(j)})$ by choosing $\rho_j \xleftarrow{R} \mathbb{Z}_q$ and setting $H(m^{(j)}) := g_0^{\rho_j}$. For query number ω, it behaves differently and instead $\mathcal{B}$ sets $H(m^{(\omega)}) := v_0$. These responses to hash queries enables $\mathcal{B}$ to respond to $\mathcal{A}_2$'s signature queries as in the proof of Theorem 15.1.

- H' query: $\mathcal{B}$ responds to a query for $H'(u)$, where $u \neq u_1$, by choosing a random $\zeta \xleftarrow{R} \mathbb{Z}_q$ and setting $H'(u) := v_0 \cdot g_0^{\zeta}$. Recall that $H'(u_1)$ has already been defined as $H'(u_1) := g_0^{\tau}$.

Eventually $\mathcal{A}_2$ outputs a valid aggregate forgery:

$$\boldsymbol{pk} = \Big((\tilde{u}_1, \tilde{\pi}_1), \ldots, (\tilde{u}_n, \tilde{\pi}_n)\Big), \quad \boldsymbol{m} = (m_1, \ldots, m_n) \in \mathcal{M}^n, \quad \sigma_{\mathrm{ag}} \in \mathbb{G}_0 \tag{15.17}$$

where

$$e(\tilde{\pi}_t, g_1) = e\big(H'(\tilde{u}_t), \tilde{u}_t\big) \quad \text{for all } t = 1, \ldots, n, \text{ and} \tag{15.18}$$

$$e(\sigma_{\mathrm{ag}}, g_1) = e\big(H(m_1), \tilde{u}_1\big) \ \cdots \ e\big(H(m_n), \tilde{u}_n\big). \tag{15.19}$$

Moreover, the challenge public key $pk = (u_1, \pi)$ is one of the public keys in $\boldsymbol{pk}$. Let m be the corresponding message in the tuple $\boldsymbol{m}$, then $\mathcal{A}_2$ never issued a signature query for m.

As in the proof of part (a), suppose that $H(m)$ was defined in hash query number ν, where $1 \leq \nu \leq Q_{\mathrm{ro}} + 1$, and that $\mathcal{B}$ guessed ν correctly at the beginning of the game, so that $\omega = \nu$. Then $H(m) = H(m^{(\omega)}) = v_0$.

Adversary $\mathcal{B}$ now needs to extract a signature $\sigma := H(m)^{\alpha} = v_0^{\alpha} = z_0$ from the aggregate forgery σ_{ag}. It does so by dividing out all the signatures in σ_{ag} until only σ remains. For $t = 1, \ldots, n$ each pair $(pk_t, m_t) = \big((\tilde{u}_t, \tilde{\pi}_t), m_t\big)$ in (15.17) is one of three types:

- type I: $m_t = m$ and $\tilde{u}_t = u_1 = g_1^{\alpha}$. In this case, the signature on m_t with respect to the public key $(\tilde{u}_t, \tilde{\pi}_t)$ is $\sigma = z_0$, which is the signature that $\mathcal{B}$ needs to compute.
- type II: $m_t \neq m$. In this case $\mathcal{B}$ can compute the signature σ_t on m_t with respect to the public key pk_t. To see why, recall that $H(m_t) = g_0^{\rho_t}$ and $\mathcal{B}$ knows $\rho_t \in \mathbb{Z}_q$. Then $\mathcal{B}$ computes $\sigma_t := \psi(\tilde{u}_t)^{\rho_t}$. This $\sigma_t \in \mathbb{G}_0$ is the signature on m_t under pk_t, as shown in (15.16).
- type III: $m_t = m$ but $\tilde{u}_t \neq u_1$. We show that in this case $\mathcal{B}$ can also compute the signature σ_t on m_t with respect to the public key $pk_t = (\tilde{u}_t, \tilde{\pi}_t)$. This is where the proof $\tilde{\pi}_t$ is used, and is the main difference from the proof of security for $\mathcal{SA}_{\mathrm{BLS}}^{(1)}$. Recall that $H'(\tilde{u}_t) = v_0 \cdot g_0^{\zeta_t}$ and $\mathcal{B}$ knows $\zeta_t \in \mathbb{Z}_q$. Adversary $\mathcal{B}$ computes σ_t as $\sigma_t := \tilde{\pi}_t / \psi(\tilde{u}_t)^{\zeta_t} \in \mathbb{G}_0$.

  We claim that this σ_t is the signature on m_t with respect to pk_t. Indeed, suppose $\tilde{u}_t = g_1^{\alpha_t}$ for some $\alpha_t \in \mathbb{Z}_q$. Then by (15.18) we know that $\tilde{\pi}_t = H'(\tilde{u}_t)^{\alpha_t}$, and therefore

  $$\sigma_t = \frac{\tilde{\pi}_t}{\psi(\tilde{u}_t)^{\zeta_t}} = \frac{H'(\tilde{u}_t)^{\alpha_t}}{\psi(g_1^{\alpha_t})^{\zeta_t}} = \frac{\big(v_0 \cdot g_0^{\zeta_t}\big)^{\alpha_t}}{g_0^{\alpha_t \cdot \zeta_t}} = v_0^{\alpha_t} = H(m)^{\alpha_t} = H(m_t)^{\alpha_t}.$$

  Hence σ_t is the unique signature on m_t with respect to $\tilde{u}_t$.

This shows that $\mathcal{B}$ can compute itself all the signatures that were aggregated into σ_{ag}, except for the challenge signature σ.

Let $d \geq 1$ be the number of times that the pair (u_1, m) appears in (15.17), that is, d is the number pairs of type I. Let $s \in \mathbb{G}_0$ be the product of all the computed signatures σ_t of types II and III above, and note that $\mathcal{B}$ can efficiently compute s. Each σ_t of type II or III is a valid signature, and therefore $e(\sigma_t, g_1) = e\big(H(m_t), \tilde{u}_t\big)$. Then, by re-ordering terms we can re-write (15.19) as

$$e(\sigma_{\mathrm{ag}}, g_1) = e\big(H(m), u_1\big)^d \cdot e(s, g_1) = e(v_0, u_1)^d \cdot e(s, g_1) = e(z_0, g_1)^d \cdot e(s, g_1) = e(z_0^d s, g_1).$$

From this it follows that $\sigma_{\mathrm{ag}} = z_0^d \cdot s$, and therefore $z_0 = (\sigma_{\mathrm{ag}}/s)^{1/d}$. Then $\mathcal{B}$ outputs $(\sigma_{\mathrm{ag}}/s)^{1/d}$ as the required co-CDH solution.

This proof strategy works fine, but incurs a factor $(Q_{\mathrm{ro}} + 1)$ loss in $\mathcal{B}$'s success probability, because $\mathcal{B}$ needs to guess ν correctly at the beginning of the game. This loss can be reduced to $2.72 \cdot (Q_{\mathrm{s}} + 1)$ as claimed in (15.13) using Exercise 15.5, as we did in Theorem 15.1. □

Remark 15.10. The proof of Theorem 15.2 uses the homomorphism $\psi : \mathbb{G}_1 \to \mathbb{G}_0$ to extract a signature forgery from the adversary. As mentioned in the discussion before the theorem, some pairing instantiations naturally support the required ψ. For other instantiations, including the ones used in practice, there is no known efficiently computable ψ. We can still prove security, but we need a stronger version of the co-CDH problem. In particular, we prove security under the, so called, ψ-co-CDH assumption, which states that the co-CDH problem is difficult even for an efficient adversary that is given access to an oracle for the function ψ. In the proof of Theorem 15.2, adversary $\mathcal{B}$ would call this oracle every time it needs to evaluate ψ. □

Remark 15.11. Theorem 15.2 shows that an aggregate signature will not verify unless all the honest signers contributed a valid signature. However, this does not mean that all the inputs to the aggregation algorithm A were valid signatures. Here is an example. Let (pk_1, m_1, σ_1) and (pk_2, m_2, σ_2) be valid signature triples, where $\sigma_1, \sigma_2 \in \mathbb{G}_0$. Then the aggregation algorithm $A\big((pk_1, pk_2), (\sigma_1, \sigma_2)\big)$ outputs the aggregate signature $\sigma = \sigma_1 \cdot \sigma_2$. Clearly the algorithm will output exactly the same σ if the pair (σ_1, σ_2) were replaced by the pair $(\delta\sigma_1, \delta^{-1}\sigma_2)$ for some $\delta \in \mathbb{G}_0$. When $\delta \neq 1$, the triples $(pk_1,\ m_1,\ \delta\sigma_1)$ and $(pk_2,\ m_2,\ \delta^{-1}\sigma_2)$ are not valid signature triples, yet their aggregation gives a valid aggregate signature. □

15.5.4 Signature schemes secure without random oracles

To conclude our discussion of pairing-based signatures, we show how to use pairings to construct signature schemes whose security does not rely on the random oracle model. In particular, when the message being signed is short, there is no need to hash it before signing or verifying, and this can be useful in certain settings (e.g., when proving knowledge of a message-signature pair in zero knowledge, as discussed in Chapter 20).

There are several ways to use pairings to construct signatures schemes based on co-CDH and related assumptions, without the random oracle model. Here we give two constructions that rely on a stronger assumption than co-CDH, but whose security proof is short and direct. The first scheme requires the signer to use a PRF. The second scheme requires no symmetric primitives when signing short messages.

As usual, let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\mathrm{T}}$ be a pairing where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_{\mathrm{T}}$ are cyclic groups of prime order q with generators $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$. Let F be a PRF defined over $(\mathcal{K}, \mathbb{Z}_q, \mathbb{Z}_q)$. We construct a signature scheme $\mathcal{S}_{\mathrm{G}} = (G, S, V)$ with message space $\mathcal{M} := \mathbb{Z}_q$.

The signature scheme $\mathcal{S}_{\mathrm{G}} = (G, S, V)$:

- $G()$: key generation runs as follows

$$\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad k \xleftarrow{\mathrm{R}} \mathcal{K}, \quad u_1 \leftarrow g_1^{\alpha}, \quad v_0 \leftarrow g_0^{\beta}, \quad h_{\mathrm{T}} \leftarrow e(v_0, g_1) \in \mathbb{G}_{\mathrm{T}}.$$

  The public key is $pk := (u_1, h_{\mathrm{T}}) \in \mathbb{G}_1 \times \mathbb{G}_{\mathrm{T}}$. The secret key is $sk := (\alpha, \beta, k) \in \mathbb{Z}_q^2 \times \mathcal{K}$.

- $S(sk, m)$: To sign a message $m \in \mathbb{Z}_q$ using a secret key $sk = (\alpha, \beta, k)$, do:

$$r \xleftarrow{\mathrm{R}} F(k, m) \in \mathbb{Z}_q, \quad w_0 \leftarrow g_0^{(\beta - r)/(\alpha - m)}, \quad \text{output } \sigma \leftarrow (r, w_0) \ \in \mathbb{Z}_q \times \mathbb{G}_0.$$

  Note that w_0 is undefined when $m = \alpha$. In that case we set $w_0 \leftarrow 1$ and $r \leftarrow \beta$.

- $V(pk, m, \sigma)$: To verify a signature $\sigma = (r, w_0) \in \mathbb{Z}_q \times \mathbb{G}_0$ on a message $m \in \mathbb{Z}_q$, using the public key $pk = (u_1, h_{\mathrm{T}})$, output accept if

$$e\big(w_0,\ u_1 g_1^{-m}\big) \cdot e(g_0, g_1)^r = h_{\mathrm{T}}. \tag{15.20}$$

A valid signature (r, w_0) is always accepted because

$$e\big(w_0,\ u_1 g_1^{-m}\big) = e\Big(g_0^{(\beta - r)/(\alpha - m)},\ g_1^{\alpha - m}\Big) = e\big(g_0, g_1\big)^{\beta - r} = h_{\mathrm{T}} \cdot e(g_0, g_1)^{-r},$$

which is what (15.20) checks for. The PRF used during signing is needed to ensure that the signer returns the same signature when asked to sign the same message multiple times. This is needed in the proof of security, and also for security of the scheme (see Exercise 15.15). Verification does not use the PRF.

To prove security we will need an assumption called d-CDH, or power CDH, which is much stronger than the basic CDH. We define this assumption next.

Attack Game 15.4 (d-CDH). For a given adversary $\mathcal{A}$ and a positive integer d, the attack game runs as follows.

- The challenger chooses $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$ and $h_1 \xleftarrow{\text{R}} \mathbb{G}_1$, and gives the adversary the tuple
$$\left(g_0^{\alpha},\ g_0^{(\alpha^2)},\ \ldots,\ g_0^{(\alpha^d)},\quad g_1^{\alpha},\quad h_1,\ h_1^{(\alpha^{d+2})}\right). \tag{15.21}$$
Let $z := e(g_0, h_1)^{(\alpha^{d+1})} \in \mathbb{G}_{\text{T}}$.
- The adversary outputs some $\hat{z} \in \mathbb{G}_{\text{T}}$.

We define $\mathcal{A}$'s **advantage in solving the d-CDH problem for** e, denoted PCDH$\mathsf{adv}[\mathcal{A}, e, d]$, as the probability that $\hat{z} = z$. □

Definition 15.7 (d-CDH assumption). *Let d be a positive integer. We say that the **d-CDH** assumption holds for e if for all efficient adversaries $\mathcal{A}$ the quantity* PCDH$\mathsf{adv}[\mathcal{A}, e, d]$ *is negligible.*

This assumption is valid as long as d is much smaller than q, the order of the groups. In Exercise 16.3 we show that, for some q, the data provided in (15.21) can reduce the work to find α by a factor of $\sqrt{d}$, compared to the work to compute discrete log in $\mathbb{G}_0$. In our application, d is about the number of signature queries from the adversary, which is a relatively small number (say, less than 2^{40}). Out of an abundance of caution, one can slightly increase the size of q to compensate for this $\sqrt{d}$ loss. Alternatively, one can choose a pairing e so that Exercise 16.3 does not apply in the group $\mathbb{G}_0$.

The d-CDH assumption, and its variants, are used in many cryptographic constructions. The following observation plays a central role in its applications, and in particular, in proving security of the signature scheme $\mathcal{S}_{\text{G}}$. Let $P(X) := \sum_{i=0}^{d'} \gamma_i \cdot X^i \in \mathbb{Z}_q[X]$ be a polynomial of degree $d' \leq d$. Then using the data in (15.21) we can construct the term $g_0^{P(\alpha)} \in \mathbb{G}_0$ by observing that:

$$g_0^{P(\alpha)} = g_0^{\sum_{i=0}^{d'} \gamma_i \cdot \alpha^i} = \prod_{i=0}^{d'} \left(g_0^{(\alpha^i)}\right)^{\gamma_i}. \tag{15.22}$$

All the terms on the right hand side of (15.22) are provided in (15.21), and hence $g_0^{P(\alpha)}$ can be calculated using this expression. In other words, we can evaluate a low-degree polynomial P at α, without knowledge of α.

We next prove security of the signature scheme $\mathcal{S}_{\text{G}}$ using the d-CDH assumption, where $d = Q+1$ and Q is the number of signature queries from the adversary.

Theorem 15.3. *Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a pairing for which d-CDH holds for poly-bounded d. Let F be a secure PRF over $(\mathcal{K}, \mathbb{Z}_q, \mathbb{Z}_q)$. Then $\mathcal{S}_{\text{G}}$ is a strongly secure signature scheme.*

> *In particular, let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\text{G}}$ as in Attack Game 13.2. Moreover, assume that $\mathcal{A}$ issues at most Q signing queries. Then there exists a $(Q+1)$-CDH adversary $\mathcal{B}_1$ and a PRF adversary $\mathcal{B}_2$, where $\mathcal{B}_1$ and $\mathcal{B}_2$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{stSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{G}}] \leq \text{PCDH}\mathsf{adv}[\mathcal{B}_1, e, (Q+1)] + \text{PRF}\mathsf{adv}[\mathcal{B}_2, F] + (1/q). \tag{15.23}$$

Proof idea. Algorithm $\mathcal{B}_1$ will use $\mathcal{A}$ to break the $(Q+1)$-CDH assumption. However, to do so, algorithm $\mathcal{B}_1$ must answer all of $\mathcal{A}$'s signature queries, which means that $\mathcal{B}_1$ must know a signature on most messages in the message space. But then the signature forgery (m, σ) produced by $\mathcal{A}$ is of no use to $\mathcal{B}_1$ because $\mathcal{B}_1$ likely already knows a signature on m.

Here we resolve this obstacle by having $\mathcal{B}_1$ give $\mathcal{A}$ a certain public key $pk := (u_1, h_{\mathrm{T}})$, where $\mathcal{B}_1$ knows *exactly one* valid signature (r_m, w_m) for every message $m \in \mathbb{Z}_q$. It will use these known signatures to answer $\mathcal{A}$'s signature queries. If $\mathcal{A}$ repeatedly asks for a signature on the same message m', our $\mathcal{B}_1$ must respond with the same signature every time, because it only knows one signature on m'. This is the reason why we use a PRF to derandomize the signing process; it ensures that our $\mathcal{B}_1$ behaves as a real signer. Eventually, $\mathcal{A}$ outputs its signature forgery (m, σ). We will show that, with overwhelming probability, this σ is a signature on m that is different from the one that $\mathcal{B}_1$ knows for m. Hence, $\mathcal{B}_1$ learned a new signature on m that it did not previously know. We will show that this fresh σ lets $\mathcal{B}$ solve the given $(Q+1)$-CDH challenge. □

Proof sketch. We first modify the challenger in Attack Game 13.2 by replacing the PRF F by a random function $f : Z_q \to \mathbb{Z}_q$. We now construct an adversary $\mathcal{B}_1$ that plays the role of challenger to $\mathcal{A}$ and solves the given $(Q+1)$-CDH instance. For $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$, this $\mathcal{B}_1$ takes as input

$$g_0^{\alpha},\ g_0^{(\alpha^2)},\ \ldots,\ g_0^{(\alpha^{(Q+1)})}, \qquad g_1^{\alpha}, \qquad h_1,\ h_1^{(\alpha^{(Q+3)})} \tag{15.24}$$

and needs to compute $e(g_0, h_1)^{(\alpha^{(Q+2)})}$. Our $\mathcal{B}_1$ begins by preparing the following data:

- $\mathcal{B}_1$ chooses a random polynomial $P(X) = \sum_{i=0}^{Q+1} \gamma_i \cdot X^i \in \mathbb{Z}_q[X]$ of degree $(Q+1)$ by choosing its coefficients $\gamma_0, \ldots, \gamma_{Q+1}$ at random in $\mathbb{Z}_q$.

- The quantity $P(\alpha)$ will play the role of $\beta \in \mathbb{Z}_q$ in the signature scheme. $\mathcal{B}_1$ computes $v_0 \leftarrow g_0^{P(\alpha)}$ using (15.22). It sets $h_{\mathrm{T}} \leftarrow e(v_0, g_1)$ and sets $u_1 \leftarrow g_1^{\alpha}$ taken from (15.24).

Next, $\mathcal{B}_1$ runs adversary $\mathcal{A}$ giving it the public key $pk = (u_1, h_{\mathrm{T}})$ prepared above. $\mathcal{B}_1$ will emulate the random function $f : \mathbb{Z}_q \to \mathbb{Z}_q$ internally. Now $\mathcal{A}$ issues a sequence of signature queries. For $j = 1, \ldots, Q$ adversary $\mathcal{B}_1$ responds to a signature query for message $m_j \in \mathbb{Z}_q$ as follows:

- Let $r_j := P(m_j) \in \mathbb{Z}_q$. Then the polynomial $\hat{P}(X) := P(X) - r_j$ satisfies $\hat{P}(m_j) = 0$. Hence, $\hat{P}(X)$ can be written as $\hat{P}(X) = (X - m_j) \cdot p(X)$ for some polynomial $p \in \mathbb{Z}_q[X]$ of degree Q. Our $\mathcal{B}_1$ can efficiently calculate the coefficients of $p(X)$ by dividing the polynomial $\hat{P}$ by $(X - m_j)$.

- $\mathcal{B}_1$ computes $w_j \leftarrow g_0^{p(\alpha)}$ using (15.22) and sends the signature $\sigma_j := (r_j, w_j)$ to $\mathcal{A}$. This implicitly defines the random function f as $f(m_j) := r_j$.

To see that σ_j is a valid signature on m_j, first observe that the polynomial $p(X)$ satisfies $p(X) = (P(X) - r_j)/(X - m_j)$. Then $w_j = g_0^{p(\alpha)} = g_0^{(P(\alpha) - r_j)/(\alpha - m_j)}$, as required.

Second, we need to argue that r_j is uniform in $\mathbb{Z}_q$ and is independent of the adversary's current view. This follows directly from the fact that P is a random polynomial of degree $Q+1$. Evaluating P at $Q+2$ distinct points results in $Q+2$ random and independent values in $\mathbb{Z}_q$. The public key gives the adversary one point $(\alpha, P(\alpha))$ on P, and every signature reveals one more point $(m_j, P(m_j))$ on P. Since the adversary makes at most Q signing queries, it sees at most $(Q+1)$ points on P, and therefore all are uniform in $\mathbb{Z}_q$ and independent of everything else.

Eventually $\mathcal{A}$ outputs a valid forgery (m, σ) where $\sigma = (r, w)$. We know that $w = g_0^{(P(\alpha)-r)/(\alpha-m)}$. If $P(m) = r$ then $\mathcal{B}_1$ aborts and terminates. In this case the forgery (m, σ) is the signature that $\mathcal{B}_1$ already knew for m, so it learned nothing new from $\mathcal{A}$. We argue that this failure event happens with probability at most $1/q$, which is the reason for the $1/q$ term in (15.23). There are three cases. (1) If m is one of the signing queries, say $m = m_j$, then clearly $r \neq r_j = P(m)$; otherwise the forgery is invalid. (2) If $m = \alpha$ then $\mathcal{B}_1$ can trivially solve the $(Q+1)$-CDH challenge. (3) Otherwise, $P(m)$ is point number $Q+2$ on the polynomial P, and is therefore independent of the adversary's view, which means that $r = P(m)$ holds with probability at most $1/q$, as claimed.

Now, suppose $P(m) \neq r$ with $w = g_0^{(P(\alpha)-r)/(\alpha-m)}$ and $m \neq \alpha$. This is information that $\mathcal{B}_1$ could not compute itself. It can use (r, w) to calculate the required $e(g_0, h_1)^{(\alpha^{(Q+2)})}$ using a sequence of elementary polynomial manipulations. We show how in Exercise 15.14. □

15.5.4.1 A different construction

We next describe a different pairing-based signature scheme that is also secure without random oracles, but this time without relying on a PRF in the signing algorithm. We obtain a signature scheme where both signing and verification are simple algebraic algorithms that require no symmetric primitives, when signing short messages. This can be useful in certain settings (e.g., when the signing process is a distributed protocol among multiple parties, as discussed in Chapter 23). The proof of security for this scheme is quite different from the previous section, and is the topic of Exercise 15.16.

The signature scheme $\mathcal{S}_{\text{BB}} = (G, S, V)$ with message space $\mathcal{M} := \mathbb{Z}_q$:

- $G()$: key generation runs as follows

$$\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u_1 \leftarrow g_1^{\alpha}, \quad v_1 \leftarrow g_1^{\beta}, \quad h_0 \xleftarrow{\text{R}} \mathbb{G}_0, \quad h_{\text{T}} \leftarrow e(h_0, g_1) \in \mathbb{G}_{\text{T}}.$$

  The public key is $pk := (u_1, v_1, h_{\text{T}}) \in \mathbb{G}_1^2 \times \mathbb{G}_{\text{T}}$. The secret key is $sk := (\alpha, \beta, h_0) \in \mathbb{Z}_q^2 \times \mathbb{G}_0$.

- $S(sk, m)$: To sign a message $m \in \mathbb{Z}_q$ using a secret key $sk = (\alpha, \beta, h_0)$, do:

$$r \xleftarrow{\text{R}} \mathbb{Z}_q, \quad w_0 \leftarrow h_0^{1/(\alpha+r\beta+m)}, \quad \text{output } \sigma \leftarrow (r, w_0) \ \in \mathbb{Z}_q \times \mathbb{G}_0.$$

  If $\alpha + r\beta + m = 0$ then repeat this process with a fresh random $r \xleftarrow{\text{R}} \mathbb{Z}_q$. Alternatively, choose a random $r \xleftarrow{\text{R}} \mathbb{Z}_q$ where $r \neq \{\frac{m-\alpha}{\beta}\}$.

- $V(pk, m, \sigma)$: To verify a signature $\sigma = (r, w_0) \in \mathbb{Z}_q \times \mathbb{G}_0$ on a message $m \in \mathbb{Z}_q$, using the public key $pk = (u_1, v_1, h_{\text{T}})$, output accept if

$$e\big(w_0,\ u_1 v_1^r g_1^m\big) = h_{\text{T}}. \tag{15.25}$$

One can verify that a valid signature is always accepted. This is because the term $u_1 v_1^r g_1^m$ in (15.25) is equal to $g_1^{\alpha+r\beta+m}$. When paired with $w_0 = h_0^{1/(\alpha+r\beta+m)}$ from a valid signature, the result is $e(h_0, g_1) = h_{\text{T}}$, which is what (15.25) checks for.

We prove security of this signature scheme using the d-CDH assumption, where $d = Q$ is the number of signature queries from the adversary.

Theorem 15.4. *Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a pairing for which d-CDH holds for poly-bounded d. Then $\mathcal{S}_{\mathrm{BB}}$ is a strongly secure signature scheme.*

> *In particular, let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{BB}}$ as in Attack Game 13.2. Moreover, assume that $\mathcal{A}$ issues at most Q signing queries. Then there exists a Q-CDH adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{stSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{BB}}] \leq 2 \cdot \mathrm{PCDH}\mathsf{adv}[\mathcal{B}, e, Q] + (Q/q). \tag{15.26}$$

Proof idea. We construct $\mathcal{B}$ that runs $\mathcal{A}$ giving it a certain public key $pk := (u_1, v_1, h_{\mathrm{T}})$. Our $\mathcal{B}$ will know Q valid signatures for every message $m \in \mathcal{M}$, and this enables it to answer all of $\mathcal{A}$'s signature queries. If $\mathcal{A}$ issues a repeated query for the same message, our $\mathcal{B}$ answers with a different random signature every time. Eventually, $\mathcal{A}$ outputs its signature forgery (m, σ). We will show that with probability at least $1/2$, this σ is a signature on m that $\mathcal{B}$ did not previously know. Moreover, this σ lets $\mathcal{B}$ solve the Q-CDH challenge. The complete proof is the topic of Exercise 15.16. □

15.6 Advanced encryption schemes from pairings

We now turn to encryption and key exchange schemes from pairings. Using pairings we construct encryption schemes with new properties that cannot be constructed efficiently in cyclic groups without a pairing. We begin by describing **identity based encryption**, and in the next section we discuss its extreme generalization called **functional encryption**.

15.6.1 Identity based encryption

When Alice wants to send an encrypted message to Bob, she must first obtain Bob's current public key. She either asks Bob for his public key certificate, or looks up Bob's public key in a public key directory. Either way, there is a communication round trip from Alice before Alice can send Bob an encrypted message. Identity based encryption, or IBE, is a method to save this initial round trip. Beyond key exchange, IBE can also be used to construct chosen ciphertext secure public key encryption, and can be used for searching on encrypted data, as we will see in Section 15.6.4.

In an IBE scheme any string can function as a public key: Bob's email address is a public key, the current date is a public key, even the numbers 1, 2, and 3 are all public keys. If Bob uses his email address as his public key, then Alice can encrypt an email to Bob without an initial round trip to look up Bob's public key. Knowledge of Bob's email address is sufficient to send Bob an encrypted email.

In more detail, in an **identity based encryption** scheme, or **IBE**, there is a trusted entity, that we call Tracy, who generates a master key pair (mpk, msk). The key mpk is called a **master public key** and is known to everyone. Tracy keeps the **master secret key** msk to herself. When Bob wants to use his email address as his public key, he must somehow obtain the private key, which is derived from msk and Bob's public key.

We refer to Bob's email address as his (public) identity id and denote the corresponding private key by sk_{id}. Bob obtains sk_{id} by contacting the trusted party Tracy. He first proves his identity id to Tracy, by proving that he owns the email address in question. This is done using a protocol that is outside of the cryptosystem. Once Tracy is convinced that Bob owns the identity id, she uses msk and id to generate sk_{id}, and sends this key to Bob over a secure channel. Now Bob can use sk_{id} to decrypt all ciphertexts encrypted to his identity id. Alice, and everyone else, can send encrypted emails to Bob without first looking up his public key. This is assuming Alice and others already have the global master public key mpk.

Before we explain how to use such a scheme, let us first define IBE more precisely.

Definition 15.8. *An* ***identity based encryption scheme*** $\mathcal{E}_{\mathrm{id}} = (S, G, E, D)$ *is a tuple of four efficient algorithms: a* ***setup algorithm*** S*, a* ***key generation algorithm*** G*, an* ***encryption algorithm*** E*, and a* ***decryption algorithm*** D*.*

- S *is a probabilistic algorithm invoked as* $(mpk, msk) \xleftarrow{\mathrm{R}} S()$*, where* mpk *is called the* ***master public key*** *and* msk *is called the* ***master secret key*** *for the IBE scheme.*

- G *is a probabilistic algorithm invoked as* $sk_{id} \xleftarrow{\mathrm{R}} G(msk, id)$*, where* msk *is the master secret key (as output by* S*),* $id \in \mathcal{ID}$ *is an identity, and* sk_{id} *is a secret key for* id*.*

- E *is a probabilistic algorithm invoked as* $c \xleftarrow{\mathrm{R}} E(mpk, id, m)$*.*

- D *is a deterministic algorithm invoked as* $m \leftarrow D(sk_{id}, c)$*. Here* m *is either a message, or a special* reject *value (distinct from all messages).*

- *As usual, we require that decryption undoes encryption; specifically, for all possible outputs* (mpk, msk) *of* S*, all identities* $id \in \mathcal{ID}$*, and all messages* m*, we have*

$$\Pr\Big[D\big(G(msk, id),\ E(mpk, id, m)\big) = m\Big] = 1.$$

- *Identities are assumed to lie in some finite* ***identity space*** $\mathcal{ID}$*, messages in some finite* ***message space*** $\mathcal{M}$*, and ciphertexts in some finite* ***ciphertext space*** $\mathcal{C}$*. We say that the IBE scheme* $\mathcal{E}_{\mathrm{id}}$ *is defined over* $(\mathcal{ID}, \mathcal{M}, \mathcal{C})$*.*

We can think of IBE as a special public key encryption scheme where the messages to be encrypted are pairs $(id, m) \in \mathcal{ID} \times \mathcal{M}$. The master secret key msk enables one to decrypt all well formed ciphertexts. However, there are weaker secret keys, such as sk_{id}, that can only decrypt *some* well formed ciphertexts: sk_{id} can decrypt a ciphertext $c \xleftarrow{\mathrm{R}} E(mpk, id', m)$ only if $id = id'$.

Another way to think of IBE is as a public key system where exponentially many public keys have been generated. If the identity space is $\mathcal{ID} := \{1, 2, \ldots, N\}$ for some large N, say $N = 2^{128}$, then the N public keys can be written as $(mpk, 1),\ (mpk, 2), \ldots, (mpk, N)$. In a regular public key system a list of N public keys cannot be compressed, and will take space $O(N)$ to store. In an IBE, these N public keys can be compressed so that only a short mpk needs to be stored.

If a company decides to use IBE for its internal encrypted email system, most likely an employee Bob will not use his email address, id, as his IBE public key. The reason is that if Bob's secret key sk_{id} were stolen, Bob would need to change his email address to ensure that sk_{id} cannot decrypt future incoming emails. This is undesirable.

Instead, Bob could use the identity string $(id,\ \textit{today's date})$ as his IBE public key. Anyone who knows Bob's id, and the current date, can send encrypted emails to Bob without first looking up his public key. In effect, Bob's public key changes every day. Every morning Bob would talk to the trusted entity Tracy to obtain his daily secret key. He can then decrypt all incoming emails for that day, without contacting Tracy. This setup has two interesting properties:

- First, if Bob leaves the company and his access needs to be revoked, Tracy could be instructed to stop issuing new keys for Bob. This prevents Bob from decrypting messages sent after his last day in the office.

- Second, Alice can encrypt an email to Bob using the public key (id, *today's date + one year*). Since Tracy will not release the corresponding secret key for a year, Bob will only get to read the email a year from today. In effect, Alice can send mail to the future.

With some engineering, these capabilities can also be implemented using regular public key encryption. However, it either requires that Tracy participate in every decryption, or requires an additional sender round-trip to lookup a daily public key for Bob. Saving a round trip in a key exchange protocol is quite desirable, as we will see when we discuss real-world key exchange in Section 21.10.

It is important to point out that using IBE for key management results in a different trust model than a traditional certificate authority (CA). When using an IBE, the trusted entity Tracy, who holds msk, can decrypt everyone's messages. An honest CA in a standard public key system cannot do that. As such, IBE is not always appropriate. It is well suited for a corporate environment where operational requirements already mandate a trusted entity that is capable of accessing all data. IBE is not appropriate in settings that require end-to-end security where only the end points should have access to the plaintext. Nevertheless, Tracy's power can be greatly reduced by secret sharing her msk across multiple parties. All, or most, of the parties would need to collude in order to inappropriately use msk, as discussed in Exercise 22.13.

IBE has several important applications beyond public key management. We will explore these in Section 15.6.4.

Secure identity based encryption. We next define the notion of semantic security for an IBE scheme. We will discuss several variations on this basic notion, including chosen-ciphertext security, after the basic definition.

The basic security definition considers an adversary who obtains the secret keys for a number of identities of its choice. The adversary should not be able to break semantic security for some other identity of its choice for which it does not have the secret key.

Attack Game 15.5 (semantic security). For a given IBE scheme $\mathcal{E}_{\text{id}} = (S, G, E, D)$, defined over $(\mathcal{ID}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $\quad (b = 0, 1)$:

- The challenger computes $(mpk, msk) \overset{\text{R}}{\leftarrow} S()$, and sends mpk to the adversary $\mathcal{A}$.
- $\mathcal{A}$ then makes a series of queries to the challenger. Each query can be one of two types:
  - *Key query:* for $j = 1, 2, \ldots$, the jth key query consists of an identity $id'_j \in \mathcal{ID}$. The challenger computes the secret key $sk'_j \leftarrow G(msk, id'_j)$, and sends sk'_j to $\mathcal{A}$.
  - *Encryption query:* a query consists of a triple $(id, m_0, m_1) \in \mathcal{ID} \times \mathcal{M}^2$, where m_0 and m_1 are messages of the same length. The challenger computes $c \overset{\text{R}}{\leftarrow} E(mpk, id, m_b)$ and sends c to $\mathcal{A}$.

  We restrict the adversary by requiring that it issue at most one encryption query (id, m_0, m_1), but this can be relaxed as explained in Remark 15.12 below. Moreover, we require that the adversary never issue a key query for the identity id used in its encryption query.
- At the end of the game, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. Define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}_{\text{id}}$ as

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{E}_{\text{id}}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 15.9 (semantic security). *An IBE scheme $\mathcal{E}_{\text{id}}$ is* ***semantically secure*** *if for all efficient adversaries $\mathcal{A}$, the value* $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}_{\text{id}}]$ *is negligible.*

This notion of semantic security is sometimes called **adaptively secure IBE**, because the adversary can choose the challenge identity *id* adaptively, after seeing many secret keys of its choice.

Remark 15.12. We can allow the adversary in Attack Game 15.5 to issue multiple encryption queries, but this does not give the adversary more power. An IBE scheme that is secure against an adversary that can only issue a single encryption query is also secure against an adversary that can issue multiple encryption queries. The proof of this statement is a simple hybrid argument that is almost identical to the proof of Theorem 11.1. □

15.6.2 Related security notions

In addition to semantic security, as in Definition 15.9, there are a number of related security requirements for IBE. We describe three requirements here: private IBE, selectively secure IBE, and chosen ciphertext secure IBE.

15.6.2.1 Private IBE

A **private IBE** is a stronger requirement for IBE security. In a private IBE, an eavesdropper who has *mpk* and intercepts a ciphertext $c \xleftarrow{R} E(mpk, id, m)$, cannot learn the identity *id* of the target recipient. We will see a number of applications for this stronger notion in Section 15.6.4. Private IBE is sometimes called **anonymous IBE**. We capture this stronger IBE security property by enhancing the adversary's encryption query in Attack Game 15.5.

Attack Game 15.6 (Private IBE). For a given IBE scheme $\mathcal{E}_{\text{id}} = (S, G, E, D)$ defined over $(\mathcal{ID}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, the game proceeds as in Attack Game 15.5. The only difference is that in the single encryption query from $\mathcal{A}$, the adversary submits two pairs (id_0, m_0) and (id_1, m_1) in $\mathcal{ID} \times \mathcal{M}$, where m_0 and m_1 are equal length messages. The challenger computes $c \xleftarrow{R} E(mpk, id_b, m_b)$ and sends c to $\mathcal{A}$. As usual, we require that the adversary never issues a key query for either id_0 or id_1.

Let $\mathrm{PrSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{id}}]$ be $\mathcal{A}$'s advantage in winning this private IBE game. □

To see that this enhanced game captures the privacy property we want, observe that if the challenge ciphertext reveals the target identity, then the attacker can easily predict the bit b and win the game. Attack Game 15.5 is a special case of this enhanced game where we require that $id_0 = id_1$ in the encryption query.

Definition 15.10 (private IBE). *An IBE scheme $\mathcal{E}_{\text{id}}$ is said to be* ***semantically secure and private*** *if for all efficient adversaries $\mathcal{A}$, the value* $\mathrm{PrSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{id}}]$ *is negligible.*

15.6.2.2 Selectively secure IBE

Selective security is a weaker notion of IBE security where the adversary is restricted in how it issues its encryption query. Specifically, we modify Attack Game 15.5 so that the adversary must choose the challenge *id* at the beginning of the game, before receiving *mpk* from the challenger.

Attack Game 15.7 (selective semantic security). For a given IBE scheme $\mathcal{E}_{\text{id}} = (S, G, E, D)$ defined over $(\mathcal{ID}, \mathcal{M}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, the game begins with $\mathcal{A}$ sending $id \in \mathcal{ID}$ to the challenger. Next, the challenger computes $(mpk, msk) \xleftarrow{\text{R}} S()$ and sends *mpk* to $\mathcal{A}$. From then on the game proceeds as in Attack Game 15.5. However, the adversary's encryption query is just a pair of equal length messages $m_0, m_1 \in \mathcal{M}$, and the challenger's response is $c \xleftarrow{\text{R}} E(mpk, id, m_b)$, where *id* is from the adversary's first message at the beginning of the game. Everything else is unchanged.

For an adversary $\mathcal{A}$ attacking an IBE scheme $\mathcal{E}_{\text{id}}$, we let $\text{SelSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{id}}]$ denote $\mathcal{A}$'s advantage in winning this selective security game against $\mathcal{E}_{\text{id}}$. □

Definition 15.11 (selective security). *An IBE scheme $\mathcal{E}_{\text{id}}$ is **selectively secure** if for all efficient adversaries $\mathcal{A}$, the value $\text{SelSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{id}}]$ is negligible.*

In Attack Game 15.7 the adversary must choose the challenge identity before it sees the IBE master public key *mpk*. Because we restrict the adversary's power in this way, it can be easier to construct a selectively secure IBE than an adaptively secure IBE. As we will see, selective security is sufficient for some applications (see Section 15.6.4).

An IBE scheme that is selectively secure can be enhanced into one that is adaptively secure as in Definition 15.9. To do so we need a hash function $H : \mathcal{ID}' \to \mathcal{ID}$. Let $\mathcal{E}_{\text{id}} = (S, G, E, D)$ be a selectively secure IBE scheme with identity space $\mathcal{ID}$. Construct an enhanced IBE scheme $\mathcal{E}'_{\text{id}} = (S, G', E', D)$ with identity space $\mathcal{ID}'$ where G' and E' operate as follows:

$$G'(msk, id) := G\big(msk, H(id)\big), \quad E'(sk_{\text{p}}, id, m) := E\big(sk_{\text{p}}, H(id), m\big). \tag{15.27}$$

Algorithms S and D are unchanged. The following theorem shows that $\mathcal{E}'_{\text{id}}$ satisfies adaptive security as in Definition 15.9. The theorem shows that to construct a semantically secure IBE, in the random oracle model, it suffices to first construct a selectively secure scheme and then apply (15.27).

Theorem 15.5. *Let $\mathcal{E}_{\text{id}} = (S, G, E, D)$ be a selectively secure IBE where $|\mathcal{ID}|$ is super-poly. Then $\mathcal{E}'_{\text{id}} = (S, G', E', D)$ is a secure IBE as in Definition 15.9, when H is modeled as a random oracle.*

> *In particular, let $\mathcal{A}$ be an adversary attacking $\mathcal{E}'_{\text{id}}$ as in Attack Game 15.9. Moreover, assume that $\mathcal{A}$ issues at most Q_{s} key queries and at most Q_{ro} queries to H. Then there exists a selective IBE adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*

$$\text{SS}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}'_{\text{id}}] \le (Q_{\text{ro}} + 1) \cdot \text{SelSS}\mathsf{adv}[\mathcal{B}, \mathcal{E}_{\text{id}}] + \big(Q_{\text{s}}/|\mathcal{ID}|\big). \tag{15.28}$$

Proof sketch. The selective IBE adversary $\mathcal{B}$ begins by choosing a random $\omega \xleftarrow{\text{R}} \{1, \ldots, Q_{\text{ro}} + 1\}$ and a random $id_{\text{ch}} \xleftarrow{\text{R}} \mathcal{ID}$. It sends id_{ch} as the challenge identity to its challenger and gets back an *mpk* for $\mathcal{E}'_{\text{id}}$. Next, $\mathcal{B}$ plays the role of an IBE challenger to $\mathcal{A}$, and starts by giving it *mpk*. Now, $\mathcal{A}$ issues three types of queries: H queries, key queries, and a single encryption query.

- When $\mathcal{A}$ issues query number j for $H(id'_j)$, our $\mathcal{B}$ responds by (consistently) mapping $H(id'_j)$ to a random element in $\mathcal{ID}$. The only exception is query number ω where $\mathcal{B}$ instead defines $H(id'_\omega) := id_{\text{ch}}$.

- When $\mathcal{A}$ issues key query number j for $id'_j \in \mathcal{ID}'$, our $\mathcal{B}$ sets $id_j \leftarrow H(id'_j)$. If $H(id'_j)$ is not yet defined then $\mathcal{B}$ defines $H(id'_j) := id_j$ for a random $id_j \xleftarrow{\text{R}} \mathcal{ID}$. If $id_j \neq id_{\text{ch}}$ then $\mathcal{B}$ issues a key query to its own selective challenger for id_j, and sends the response back to $\mathcal{A}$. However, if $id_j = id_{\text{ch}}$ then $\mathcal{B}$ is not allowed to query its challenger for this key. In this case $\mathcal{B}$ aborts and terminates. This failure event happens with probability $1/|\mathcal{ID}|$ per query.

- When $\mathcal{A}$ issues its single encryption query for (id', m_0, m_1), our $\mathcal{B}$ first issues an internal query for $id \leftarrow H(id')$. If $H(id')$ was defined in hash query number ω then $id = id_{\text{ch}}$. Then $\mathcal{B}$ forwards (m_0, m_1) to its selective IBE challenger, and sends the response $c \xleftarrow{\text{R}} E(mpk, id, m_b)$ to $\mathcal{A}$. Here $b \in \{0, 1\}$ is the challenger's challenge bit.

Eventually $\mathcal{B}$ outputs the bit $\hat{b} \in \{0, 1\}$ that $\mathcal{A}$ outputs when it terminates. By a union bound, the probability that $\mathcal{B}$ aborts as a result of a key query is at most $Q_{\text{s}}/|\mathcal{ID}|$, which is the reason for the additive error term in (15.28). $\mathcal{B}$ correctly guesses the H query that $\mathcal{A}$ will use for its encryption query with probability $1/(Q_{\text{ro}} + 1)$, which is the reason for the multiplicative term in (15.28). Note that an identity that is used in a key query cannot be used in the encryption query, which is why the multiplicative term is not $1/(Q_{\text{ro}} + Q_{\text{s}} + 1)$. □

15.6.2.3 Chosen-ciphertext secure IBE

The third IBE security definition we consider is chosen-ciphertext secure IBE. This is a stronger notion of security where we allow the adversary in Attack Game 15.5 to also issue decryption queries. A decryption query is a pair (id', c'). The challenger responds by computing $sk_{id} \xleftarrow{\text{R}} G(msk, id')$ and $m' \leftarrow D(sk_{id}, c')$, and sends m' to the adversary. As usual, the adversary can issue a decryption query for all pairs in $\mathcal{ID} \times \mathcal{C}$ except for the pair (id, c) associated with the adversary's encryption query.

There are a number of generic transformations that transform a semantically secure IBE into a chosen ciphertext secure IBE [16]. These transformations are similar to the generic method described in Section 12.6 for transforming a semantically secure public key encryption scheme into one that is chosen ciphertext secure. We describe one such transformation in Exercise 15.18.

15.6.3 Identity based encryption from pairings

We now turn to constructing IBE schemes. There are a number of constructions for secure IBE using pairings. Here we give two simple constructions that show different ways of using pairings. The first construction gives a simple private IBE in the random oracle model. The second construction has some performance benefits over the first, and does not use random oracles, but is only selectively secure. It can be made adaptively secure using Theorem 15.5. A third construction is presented in Exercise 15.19.

We will need the following components:

- As usual, let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\text{T}}$ be a pairing where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_{\text{T}}$ are cyclic groups of prime order q with generators $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$,

- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$,

- hash functions $H_0 : \mathcal{ID} \to \mathbb{G}_0$ and $H_1 : \mathbb{G}_1 \times \mathbb{G}_{\text{T}} \to \mathcal{K}$.

15.6.3.1 Construction 1

Let $\mathcal{E}_{\text{BF}} = (S, G, E, D)$ be the following IBE scheme with identity space $\mathcal{ID}$ and message space $\mathcal{M}$:

- $S()$: the setup algorithm runs as follows:

$$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u_1 \leftarrow g_1^\alpha, \quad mpk \leftarrow u_1, \quad msk \leftarrow \alpha, \quad \text{output } (mpk, msk).$$

- $G(msk, id)$: key generation using $msk = \alpha$ runs as:

$$sk_{id} \leftarrow H_0(id)^\alpha \in \mathbb{G}_0, \quad \text{output } sk_{id}.$$

- $E(mpk, id, m)$: encryption using the public parameters $mpk = u_1$ runs as:

$$\beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad w_1 \leftarrow g_1^\beta, \quad z \leftarrow e\big(H_0(id),\ u_1^\beta\big) \in \mathbb{G}_{\text{T}},$$
$$k \leftarrow H_1(w_1, z), \quad c \xleftarrow{\text{R}} E_{\text{s}}(k, m), \quad \text{output } (w_1, c).$$

- $D\big(sk_{id},\ (w_1, c)\big)$: decryption using secret key sk_{id} of ciphertext (w_1, c) runs as follows:

$$z \leftarrow e(sk_{id}, w_1), \quad k \leftarrow H_1(w_1, z), \quad m \leftarrow D_{\text{s}}(k, c), \quad \text{output } m.$$

To see that the scheme is correct it suffices to show that k computed during encryption is the same as k obtained during decryption. Since $k = H_1(w_1, z)$ it suffices to show that decryption recovers the correct z by computing $z \leftarrow e(sk_{id}, w_1)$. This follows from:

$$e\big(sk_{id}, w_1\big) = e\big(H_0(id)^\alpha,\ g_1^\beta\big) = e\big(H_0(id),\ g_1^{\alpha\beta}\big) = e\big(H_0(id),\ u_1^\beta\big).$$

The quantity on the right is the value of z used in encryption.

15.6.3.2 The decision-BDH assumption

Security of $\mathcal{E}_{\text{BF}}$ follows from an assumption called the **bilinear Diffie-Hellman** assumption, or BDH. Here we will use the decisional version of the assumption. The assumption says that given random elements $g_0^\alpha, g_0^\beta, g_0^\gamma \in \mathbb{G}_0$, plus a few additional terms, the quantity $e(g_0, g_1)^{\alpha\beta\gamma} \in \mathbb{G}_{\text{T}}$ is indistinguishable from a random element in $\mathbb{G}_{\text{T}}$.

Attack Game 15.8 (Decision bilinear Diffie-Hellman). let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\text{T}}$ be a pairing where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_{\text{T}}$ are cyclic groups of prime order q with generators $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$. For a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $\quad (b = 0, 1)$:

- The challenger computes

$$\alpha, \beta, \gamma, \delta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u_0 \leftarrow g_0^\alpha, \quad u_1 \leftarrow g_1^\alpha, \quad v_0 \leftarrow g_0^\beta, \quad w_1 \leftarrow g_1^\gamma,$$
$$z^{(0)} \leftarrow e(g_0, g_1)^{\alpha\beta\gamma} \in \mathbb{G}_{\text{T}}, \quad z^{(1)} \leftarrow e(g_0, g_1)^\delta \in \mathbb{G}_{\text{T}}$$

  and gives $(u_0, u_1, v_0, w_1, z^{(b)})$ to the adversary.

- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. Define $\mathcal{A}$'s **advantage in solving the decision bilinear Diffie-Hellman problem for** e as

$$\mathrm{DBDH}\mathsf{adv}[\mathcal{A}, e] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 15.12 (Decision BDH assumption). *We say that the* ***decision bilinear Diffie-Hellman (DBDH)*** *assumption holds for the pairing e if for all efficient adversaries $\mathcal{A}$ the quantity* $\mathrm{DBDH}\mathsf{adv}[\mathcal{A}, e]$ *is negligible.*

15.6.3.3 Security of the IBE scheme $\mathcal{E}_{\mathrm{BF}}$

We use decision BDH to prove that $\mathcal{E}_{\mathrm{BF}}$ is a semantically secure and private IBE in the random oracle model. We model $H_0 : \mathcal{ID} \to \mathbb{G}_0$ as a random oracle, and model $H_1 : \mathbb{G}_1 \times \mathbb{G}_{\mathrm{T}} \to \mathcal{K}$ as a secure key derivation function (KDF) as in Definition 11.5.

Theorem 15.6. *If decision BDH holds for e, H_0 is modeled as a random oracle, H_1 is a secure KDF, and $\mathcal{E}_{\mathrm{s}}$ is semantically secure, then $\mathcal{E}_{\mathrm{BF}}$ (from Section 15.6.3.1) is a semantically secure and private IBE.*

> *In particular, let $\mathcal{A}$ be an adversary attacking $\mathcal{E}_{\mathrm{BF}}$ as in Attack Game 15.6. Moreover, assume that $\mathcal{A}$ issues at most Q_{s} key queries. Then there exist a decision BDH adversary $\mathcal{B}_{\mathrm{e}}$, a KDF adversary $\mathcal{B}_{\mathrm{kdf}}$ that plays Attack Game 11.3 with respect to H_1, and an SS adversary $\mathcal{B}_{\mathrm{s}}$ that plays Attack Game 2.1 with respect to $\mathcal{E}_{\mathrm{s}}$, where $\mathcal{B}_{\mathrm{e}}, \mathcal{B}_{\mathrm{kdf}}$, and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that*

$$\mathrm{SS}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{BF}}] \le 2 \cdot 2.72 \cdot (Q_{\mathrm{s}} + 1) \cdot \mathrm{DBDH}\mathsf{adv}[\mathcal{B}_{\mathrm{e}}, e] + 2 \cdot \mathrm{KDF}\mathsf{adv}[\mathcal{B}_{\mathrm{kdf}}, H_1] + \mathrm{SS}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}]. \qquad (15.29)$$

Proof sketch. Adversary $\mathcal{B}_{\mathrm{e}}$ is given a decision BDH challenge

$$(u_0 = g_0^{\alpha}, \quad u_1 = g_1^{\alpha}, \quad v_0 = g_0^{\tau}, \quad w_1 = g_1^{\beta}, \quad z)$$

where $\alpha, \beta, \tau \xleftarrow{\text{R}} \mathbb{Z}_q$. It needs to decide if $z = e(g_0, g_1)^{\alpha\beta\tau}$ or if z is uniform in $\mathbb{G}_{\mathrm{T}}$. The adversary begins by sending the IBE public parameters $mpk := u_1$ to the IBE adversary $\mathcal{A}$.

Next, $\mathcal{A}$ makes a sequence of Q_{s} key queries and Q_{ro} queries to H_0. For $j = 1, 2, \ldots$ our adversary $\mathcal{B}_{\mathrm{e}}$ responds to hash query number j for $H_0(id^{(j)})$ by choosing $\rho_j \xleftarrow{\text{R}} \mathbb{Z}_q$ and setting $H_0(id^{(j)}) := g_0^{\rho_j}$. This enables $\mathcal{B}_{\mathrm{e}}$ to answer all of $\mathcal{A}$'s key queries. The key for $id^{(j)}$ is simply

$$sk_j := H_0\big(id^{(j)}\big)^{\alpha} = g_0^{\rho_j \alpha} = u_0^{\rho_j},$$

which $\mathcal{B}_{\mathrm{e}}$ can compute itself using the given u_0. This is the reason we include u_0 in the decision BDH assumption.

One exception to the above is that at the beginning of the game $\mathcal{B}_{\mathrm{e}}$ chooses a random ω in the range 1 to $(Q_{\mathrm{ro}} + 1)$, and responds to hash query number ω with $H_0(id^{(\omega)}) := v_0$. If $\mathcal{A}$ ever issues a key query for $id^{(\omega)}$ then $\mathcal{B}_{\mathrm{e}}$ cannot answer it, and $\mathcal{B}_{\mathrm{e}}$ must abort and fail.

At some point $\mathcal{A}$ issues a single encryption query for the pair (id_0, m_0) and (id_1, m_1), as in the private IBE security game. $\mathcal{B}_{\mathrm{e}}$ chooses a random $b \xleftarrow{\text{R}} \{0, 1\}$ and issues an internal query for $H_0(id_b)$. If $\mathcal{B}_{\mathrm{e}}$ guessed ω correctly then $id_b = id^{(\omega)}$ and hence $H_0(id_b) = v_0$. Then $\mathcal{B}_{\mathrm{e}}$ responds by computing $k \leftarrow H_1(w_1, z)$ and $c \xleftarrow{\text{R}} E_{\text{s}}(k, m_b)$, and sends the challenge ciphertext (w_1, c) to $\mathcal{A}$. The point is that if $z = e(g_0, g_1)^{\alpha\beta\tau}$ then

$$z = e(v_0, g_1^{\alpha\beta}) = e\big(H_0(id_b), u_1^{\beta}\big),$$

which is the correct value of z obtained when encrypting m_b using randomness $\beta \in \mathbb{Z}_q$. However, if z is uniform in $\mathbb{G}_{\text{T}}$ then $k := H_1(w_1, z)$ is uniform in $\mathcal{K}$, and independent of the adversary's view. Now $\mathcal{A}$ cannot tell if it was given an encryption of m_0 or m_1 thanks to the semantic security of $\mathcal{E}_{\text{s}}$.

When eventually $\mathcal{A}$ outputs a bit $\hat{b} \in \{0, 1\}$, our $\mathcal{B}_{\text{e}}$ outputs 1 if $b = \hat{b}$ and 0 otherwise. Then by an argument from (2.10), $\mathcal{B}_{\text{e}}$'s advantage against decision BDH is half $\mathcal{A}$'s advantage in distinguishing $E(mpk, id_0, m_0)$ from $E(mpk, id_1, m_1)$.

This proof strategy works, but incurs a factor $(Q_{\text{ro}} + 1)$ loss in $\mathcal{B}_{\text{e}}$'s success probability compared to $\mathcal{A}$'s, because $\mathcal{B}_{\text{e}}$ needs to guess ω correctly. This loss can be reduced to $2.72 \cdot (Q_{\text{s}} + 1)$, as claimed in (15.29) using Exercise 15.5. □

15.6.3.4 Construction 2

We next present a second IBE scheme that has a very different proof of security using a technique called a punctured master key. This scheme avoids hashing identities onto the bilinear group which can lead to efficiency improvements over construction 1. We prove selective IBE security from decision BDH, without relying on the random oracle model.

The construction uses a pairing e, and a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, as in construction 1. It also uses a hash function $H : \mathbb{G}_1^2 \times \mathbb{G}_{\text{T}} \to \mathcal{K}$.

Let $\mathcal{E}_{\text{BB}} = (S, G, E, D)$ be the following IBE scheme with identity space $\mathcal{ID} := \mathbb{Z}_q$ and message space $\mathcal{M}$:

- $S()$: the setup algorithm runs as follows:

$$\begin{aligned}
&\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \qquad u_0 \leftarrow g_0^{\alpha}, \quad u_1 \leftarrow g_1^{\alpha}, \qquad v_0 \leftarrow g_0^{\beta}, \quad v_1 \leftarrow g_1^{\beta}, \\
&w_0 \xleftarrow{\text{R}} \mathbb{G}_0, \qquad w \leftarrow e(w_0, g_1) \in \mathbb{G}_{\text{T}}, \\
&mpk \leftarrow (u_1, v_1, w), \quad msk \leftarrow (u_0, v_0, w_0), \quad \text{output } (mpk, msk).
\end{aligned}$$

- $G(msk, id)$: key generation for $id \in \mathbb{Z}_q$ using $msk = (u_0, v_0, w_0)$ runs as:

$$\gamma \xleftarrow{\text{R}} \mathbb{Z}_q, \quad a_0 \leftarrow w_0 \cdot \big(u_0^{id} \cdot v_0\big)^{\gamma}, \quad b_0 \leftarrow g_0^{\gamma}, \quad sk_{id} \leftarrow (a_0, b_0) \in \mathbb{G}_0^2, \quad \text{output } sk_{id}.$$

  Notice that $sk_{id} = (a_0, b_0)$ is a multiplicative ElGamal encryption of the master secret value $w_0 \in \mathbb{G}_0$ using the public key $(u_0^{id} \cdot v_0) \in \mathbb{G}_0$.

- $E(mpk, id, m)$: encryption using the public parameters $mpk = (u_1, v_1, w)$ runs as:

$$\begin{aligned}
&\rho \xleftarrow{\text{R}} \mathbb{Z}_q, \quad x_1 \leftarrow g_1^{\rho}, \quad y_1 \leftarrow \big(u_1^{id} \cdot v_1\big)^{\rho}, \quad z \leftarrow w^{\rho}, \\
&k \leftarrow H(x_1, y_1, z), \quad c \xleftarrow{\text{R}} E_{\text{s}}(k, m), \quad \text{output } (x_1, y_1, c).
\end{aligned}$$

- $D\big(sk_{id},\ (x_1, y_1, c)\big)$: to decrypt (x_1, y_1, c) using secret key $sk_{id} = (a_0, b_0)$ do:

$$z \leftarrow e(a_0, x_1)/e(b_0, y_1), \quad k \leftarrow H(x_1, y_1, z), \quad m \leftarrow D_{\text{s}}(k, c), \quad \text{output } m.$$

To see that the scheme is correct it suffices to show that k computed during encryption is the same as k obtained during decryption. Since $k = H(x_1, y_1, z)$ it suffices to show that decryption recovers the correct z by computing $z \leftarrow e(a_0, x_1)/e(b_0, y_1)$. This follows from:

$$\frac{e(a_0, x_1)}{e(b_0, y_1)} = \frac{e\left(w_0 \cdot \left(u_0^{id} v_0\right)^{\gamma},\ g_1^{\rho}\right)}{e\left(g_0^{\gamma},\ \left(u_1^{id} v_1\right)^{\rho}\right)} = e(w_0, g_1)^{\rho} \cdot \frac{e\left(\left(u_0^{id} v_0\right)^{\gamma},\ g_1^{\rho}\right)}{e\left(g_0^{\gamma},\ \left(u_1^{id} v_1\right)^{\rho}\right)} = e(w_0, g_1)^{\rho} = w^{\rho},$$

which is the value of z used in encryption.

Privacy. Is the scheme $\mathcal{E}_{\mathrm{BB}}$ a private IBE? When using a symmetric pairing, where $\mathbb{G}_0 = \mathbb{G}_1$, the scheme is not private, unlike Construction 1. To see why, suppose an eavesdropper intercepts a ciphertext (x_1, y_1, c) intended for either id_0 or id_1. If id is the target identity then

$$e(g_0, y_1) = e(u_0^{id} v_0,\ x_1).$$

This lets the eavesdropper distinguish a ciphertext for id_0 from one for id_1. One complication is that u_0, v_0, needed for the test, are part of the master secret *msk*, and may not be available to the adversary. When using a symmetric pairing this is not a problem because $u_0 = u_1$ and $v_0 = v_1$ and both u_1 and v_1 are public as part of *mpk*. In this case the scheme is clearly not private. However, when using an asymmetric pairing, where $\mathbb{G}_0 \neq \mathbb{G}_1$, and there is no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_0$, then u_0, v_0 are secret. In this case there is no known attack on the privacy of the scheme.

Security. We prove security of the scheme $\mathcal{E}_{\mathrm{BB}}$ using the same decision BDH used to prove security of Construction 1. The theorem below proves that the scheme is a selectively secure IBE (as in Definition 15.11) without relying on random oracles. The scheme can be made adaptively secure using Theorem 15.5, by hashing all identities using a random oracle.

Theorem 15.7. *If decision BDH holds for e, H is a secure KDF, and $\mathcal{E}_{\mathrm{s}}$ is a semantically secure cipher, then $\mathcal{E}_{\mathrm{BB}}$ is a selectively secure IBE.*

> *In particular, for every selective IBE adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\mathrm{BB}}$ as in Attack Game 15.7, there exist a decision BDH adversary $\mathcal{B}_{\mathrm{e}}$, a KDF adversary $\mathcal{B}_{\mathrm{kdf}}$ that plays Attack Game 11.3 with respect to H, and an SS adversary $\mathcal{B}_{\mathrm{s}}$ that plays Attack Game 2.1 with respect to $\mathcal{E}_{\mathrm{s}}$, where $\mathcal{B}_{\mathrm{e}}, \mathcal{B}_{\mathrm{kdf}}$, and $\mathcal{B}_{\mathrm{s}}$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{SelSS}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{BB}}] \leq 2 \cdot \mathrm{DBDH}\mathsf{adv}[\mathcal{B}_{\mathrm{e}}, e] + 2 \cdot \mathrm{KDF}\mathsf{adv}[\mathcal{B}_{\mathrm{kdf}}, H] + \mathrm{SS}\mathsf{adv}[\mathcal{B}_{\mathrm{s}}, \mathcal{E}_{\mathrm{s}}].$$

Proof idea. The core of the proof shows that an efficient adversary learns nothing about the quantity $z \in \mathbb{G}_{\mathrm{T}}$ used in the challenge ciphertext. More precisely, suppose a selective IBE adversary $\mathcal{A}$ can distinguish a challenge ciphertext (x_1, y_1, c) created using a correct z (i.e., $z = w^{\rho}$), from a ciphertext created using a random z (i.e., $z \xleftarrow{\mathrm{R}} \mathbb{G}_{\mathrm{T}}$). We use $\mathcal{A}$ to construct a $\mathcal{B}_{\mathrm{e}}$ that attacks decision BDH.

The $\mathcal{B}_{\mathrm{e}}$ we construct uses a technique called a **punctured secret key**. It operates as follows. First, $\mathcal{B}_{\mathrm{e}}$ is given a decision BDH challenge as input. Next, it runs $\mathcal{A}$ and obtains from $\mathcal{A}$ a challenge identity $id \in \mathcal{ID}$ that $\mathcal{A}$ intends to attack. $\mathcal{B}_{\mathrm{e}}$ then prepares a master key pair (mpk, msk_{p}) and gives *mpk* to $\mathcal{A}$. The master secret key msk_{p} is constructed in a special way so that $\mathcal{B}_{\mathrm{e}}$ knows the secret key $sk_{id'}$ for all identities $id' \in \mathcal{ID}$, except for id. We say that msk_{p} is a punctured key. It works correctly for all identities, except for the identity id where it was punctured. Note that the identity *id* to puncture must be known before $\mathcal{B}_{\rm e}$ generates the master key pair $(mpk, msk_{\rm p})$. This is the reason we only prove selective security.

Adversary $\mathcal{B}_{\rm e}$ uses $msk_{\rm p}$ to answer all the key queries from $\mathcal{A}$. This works because $\mathcal{A}$ cannot ask for a key for the punctured identity *id*. $\mathcal{B}_{\rm e}$ then uses the given decision BDH instance to generate the challenge ciphertext for the identity *id*. If the adversary correctly distinguishes a random z from a correct z, then $\mathcal{B}_{\rm e}$ can answer its decision BDH challenge. □

Proof sketch. Let's see how $\mathcal{B}_{\rm e}$ works. It is given a decision BDH instance

$$\Big(\hat{u}_0 = g_0^{\alpha}, \quad \hat{u}_1 = g_1^{\alpha}, \quad \hat{v}_0 = g_0^{\beta}, \quad \hat{x}_1 = g_1^{\rho}, \quad z\Big), \tag{15.30}$$

where $\alpha, \beta, \rho \overset{\rm R}{\leftarrow} \mathbb{Z}_q$ and $z = e(g_0, g_1)^{\alpha\beta\rho}$ or $z \overset{\rm R}{\leftarrow} \mathbb{G}_{\rm T}$. Our $\mathcal{B}_{\rm e}$ needs to decide whether the given z is of the former or the latter type. To do so, $\mathcal{B}_{\rm e}$ runs $\mathcal{A}$ and obtains a challenge identity $id \in \mathcal{ID}$ from $\mathcal{A}$. It creates a master public key *mpk* and a punctured master secret key $msk_{\rm p}$ as follows:

$$\begin{aligned} &\kappa \overset{\rm R}{\leftarrow} \mathbb{Z}_q, \qquad u_0 \leftarrow \hat{u}_0, \quad u_1 \leftarrow \hat{u}_1, \qquad v_0 \leftarrow u_0^{(-id)} \cdot g_0^{\kappa}, \quad v_1 \leftarrow u_1^{(-id)} \cdot g_1^{\kappa}, \\ &w \leftarrow e(\hat{v}_0, \hat{u}_1) = e(g_0, g_1)^{\alpha\beta}, \qquad mpk \leftarrow (u_1, v_1, w), \quad msk_{\rm p} \leftarrow (\kappa, u_0, v_0, \hat{v}_0). \end{aligned}$$

$\mathcal{B}_{\rm e}$ sends *mpk* to $\mathcal{A}$ and keeps the punctured $msk_{\rm p}$ to itself. We will see why this $msk_{\rm p}$ is a punctured secret key in the next paragraph. Note that the full (non-punctured) master secret key corresponding to *mpk* is $msk := (u_0,\ v_0,\ w_0)$ where $w_0 := g_0^{\alpha\beta}$. This is because w in *mpk* is $w = e(g_0^{\alpha\beta}, g_1)$. However, $\mathcal{B}_{\rm e}$ does not know *msk* because it cannot compute $w_0 = g_0^{\alpha\beta}$.

Next, $\mathcal{A}$ issues a sequence of key queries. Suppose query j is for identity $id_j \neq id$. Our $\mathcal{B}_{\rm e}$ responds by choosing $\gamma \overset{\rm R}{\leftarrow} \mathbb{Z}_q$ and computing:

$$a_j \leftarrow \hat{v}_0^{-\frac{\kappa}{id_j - id}} \cdot (u_0^{id_j} \cdot v_0)^{\gamma}, \qquad b_j \leftarrow \hat{v}_0^{-\frac{1}{id_j - id}} \cdot g_0^{\gamma}, \qquad sk_j \leftarrow (a_j, b_j) \in \mathbb{G}_0^2. \tag{15.31}$$

$\mathcal{B}_{\rm e}$ sends sk_j to $\mathcal{A}$. To see that sk_j is a valid secret key for id_j set $\gamma' := \gamma - \beta/(id_j - id) \in \mathbb{Z}_q$. Then a simple calculation shows that

$$a_j = g_0^{\alpha\beta} \cdot (u_0^{id_j} \cdot v_0)^{\gamma'} \qquad \text{and} \qquad b_j = g_0^{\gamma'}.$$

Hence, (a_j, b_j) is computed as a secret key for id_j as in the real scheme, using $\gamma' \in \mathbb{Z}_q$ as the random nonce. This shows that $msk_{\rm p}$ can be used to generate a secret key for every identity, except for *id* where (15.31) fails because of the division by zero in the exponent.

At some point $\mathcal{A}$ issues its single encryption query for messages $m_0, m_1 \in \mathcal{M}$ with respect to the challenge identity *id*. Note that $u_1^{id} \cdot v_1 = g_1$. Adversary $\mathcal{B}_{\rm e}$ chooses $b \overset{\rm R}{\leftarrow} \{0, 1\}$ and computes:

$$x_1 \leftarrow \hat{x}_1 = g_1^{\rho}, \qquad y_1 \leftarrow (\hat{x}_1)^{\kappa} = (u_1^{id} \cdot v_1)^{\rho}, \qquad k \leftarrow H(x_1, y_1, z), \qquad c \overset{\rm R}{\leftarrow} E(k, m_b).$$

It sends (x_1, y_1, c) to $\mathcal{A}$. If $z = e(g_0, g_1)^{\alpha\beta\rho} = w^{\rho}$ then this is a valid encryption of m_b. If $z \overset{\rm R}{\leftarrow} \mathbb{G}_{\rm T}$ then k is uniform in $\mathcal{K}$, in the adversary's view, and hence the adversary cannot tell if it was given an encryption of m_0 or m_1 thanks to the semantic security of $\mathcal{E}_{\rm s}$. When eventually $\mathcal{A}$ outputs a bit $\hat{b} \in \{0, 1\}$, our $\mathcal{B}_{\rm e}$ outputs 1 if $b = \hat{b}$ and 0 otherwise. Then $\mathcal{B}_{\rm e}$'s advantage against decision BDH is half $\mathcal{A}$'s advantage in distinguishing the correct z from a random z, by a standard argument from (2.10). □

15.6.4 Applications

IBE can be used as a cryptographic primitive to construct other cryptosystems. We give three examples.

15.6.4.1 Chosen ciphertext security from IBE

A selectively secure IBE scheme can be used to construct a chosen ciphertext secure public key encryption (PKE) scheme. Let $\mathcal{E}_{\text{id}} = (G, S, E, D)$ be a selectively secure IBE scheme with identity space $\mathcal{ID}$ and message space $\mathcal{M}$. Let $\mathcal{S} = (G_{\text{s}}, S_{\text{s}}, V_{\text{s}})$ be a signature scheme and suppose that all public keys output by $G_{\text{s}}()$ are contained in $\mathcal{ID}$. We construct a chosen ciphertext secure public key encryption scheme where the encryption algorithm first runs $G_{\text{s}}()$ to generate a signature key pair $(pk_{\text{s}}, sk_{\text{s}})$. It then IBE encrypts the given message m using pk_{s} as the identity, and uses sk_{s} to sign the resulting ciphertext.

In more detail, the chosen ciphertext secure public key encryption scheme $\mathcal{E} = (G_{\text{e}}, E_{\text{e}}, D_{\text{e}})$ with message space $\mathcal{M}$ works as follows:

- $G_{\text{e}}()$: key generation runs $(mpk, msk) \overset{\text{R}}{\leftarrow} G()$ and outputs (mpk, msk) as the public key pair.
- $E_{\text{e}}(mpk, m)$: the algorithm runs as

$$(pk_{\text{s}}, sk_{\text{s}}) \overset{\text{R}}{\leftarrow} G_{\text{s}}(), \quad c \overset{\text{R}}{\leftarrow} E(mpk, pk_{\text{s}}, m), \quad \sigma \overset{\text{R}}{\leftarrow} S_{\text{s}}(sk_{\text{s}}, c), \quad \text{output } (c,\ pk_{\text{s}},\ \sigma).$$

- $D_{\text{e}}\big(msk, (c, pk_{\text{s}}, \sigma)\big)$ works as follows:

$$\begin{array}{l} \text{if } V_{\text{s}}(pk_{\text{s}}, c, \sigma) = \textsf{reject}, \text{ output } \textsf{reject} \text{ and stop} \\ \text{otherwise,} \quad sk \overset{\text{R}}{\leftarrow} G(msk, pk_{\text{s}}), \quad m \leftarrow D(sk, c), \quad \text{output } m. \end{array}$$

The following theorem shows that a selectively secure IBE is sufficient to ensure that $\mathcal{E}$ is CCA secure. Note that every signing key sk_{s} in this scheme is only used to sign a single message. Therefore, a one-time secure signature scheme is sufficient for security.

Theorem 15.8. *Let $\mathcal{E}_{\text{id}}$ be a selectively secure IBE scheme as in Definition 15.11. Let $\mathcal{S}$ be a strongly secure one-time secure signature scheme. Then $\mathcal{E}$ is a chosen ciphertext secure public key encryption scheme.*

> *In particular, for every CCA adversary $\mathcal{A}$ that attacks $\mathcal{E}$ and issues at most Q_{e} encryption queries, there exist a selective IBE adversary $\mathcal{B}_1$ that attacks $\mathcal{E}_{\text{id}}$, and a strong one-time signature adversary $\mathcal{B}_2$ that attacks $\mathcal{S}$, where $\mathcal{B}_1$ and $\mathcal{B}_2$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq Q_{\text{e}} \cdot \text{SelSS}\mathsf{adv}[\mathcal{B}_1, \mathcal{E}_{\text{id}}] + Q_{\text{e}} \cdot \text{stSIG}\mathsf{adv}[\mathcal{B}_2, \mathcal{S}].$$

Proof idea. Let $\mathcal{A}$ be a CCA adversary that attacks $\mathcal{E}$ as in Attack Game 12.1. By Theorem 12.1 we can assume that $\mathcal{A}$ issues a single encryption query to its challenger.

Let $(c, pk_{\text{s}}, \sigma)$ be the challenge ciphertext given to $\mathcal{A}$ in a CCA game. If $\mathcal{A}$ ever issues a decryption query for a ciphertext $(c', pk_{\text{s}}, \sigma')$ where $(c', \sigma') \neq (c, \sigma)$, and where the response is not $\textsf{reject}$, then we immediately obtain an adversary $\mathcal{B}_2$ that attacks the one-time signature scheme $\mathcal{S}$. Hence, we can modify the CCA challenger to answer all decryption queries for ciphertexts of the form $(\cdot, pk_{\text{s}}, \cdot)$ with $\textsf{reject}$. This only negligibly affects $\mathcal{A}$'s advantage.

Now we can use $\mathcal{A}$ to build an adversary $\mathcal{B}_1$ that attacks the selectively secure IBE scheme $\mathcal{E}_{id}$ as in Attack Game 15.7. $\mathcal{B}_1$ plays the role of CCA challenger to $\mathcal{A}$. At the beginning of the IBE game, $\mathcal{B}_1$ runs $(pk_{\text{s}}, sk_{\text{s}}) \xleftarrow{\text{R}} G_{\text{s}}()$ and sends pk_{s} to its selective IBE challenger as the selected identity that it intends to attack. It next receives mpk from the IBE challenger and forwards mpk to $\mathcal{A}$ as the public key to attack.

When $\mathcal{A}$ issues its (single) encryption query for m_0, m_1, our $\mathcal{B}_1$ issues an encryption query using m_0, m_1 to its own challenger. It gets back an IBE challenge ciphertext $c \xleftarrow{\text{R}} E(mpk, pk_{\text{s}}, m_b)$. It then creates the PKE challenge ciphertext $(c, pk_{\text{s}}, \sigma)$ by computing $\sigma \xleftarrow{\text{R}} S_{\text{s}}(sk_{\text{s}}, c)$ and sends this ciphertext to $\mathcal{A}$.

Next, $\mathcal{B}_1$ responds to all of $\mathcal{A}$'s decryption queries by requesting the appropriate secret key from the IBE challenger. Note that because decryption queries from $\mathcal{A}$ of the form $(\cdot, pk_{\text{s}}, \cdot)$ are automatically answered with reject, our $\mathcal{B}_1$ will never ask its IBE challenger for a secret key for the identity pk_{s}, as required in the selective IBE game.

Eventually $\mathcal{A}$ outputs a bit b' that indicates whether c is an encryption of m_0 or m_1. Our $\mathcal{B}_1$ outputs this b and halts. It is easy to see that $\mathcal{B}_1$'s advantage in winning the IBE game is the same as $\mathcal{A}$'s advantage in winning the CCA game against $\mathcal{E}$, as required. □

Properties of this construction. This CCA construction has a number of interesting properties. First, when instantiated with Construction 2 above, and the one-time signature from Exercise 14.12, the result is a CCA secure public key encryption scheme without random oracles. The only other such system we saw was described in Section 12.5. The current construction gives a completely different approach.

Second, an observer can verify the signature σ in a given ciphertext $(c, pk_{\text{s}}, \sigma)$. If the signature verifies then the observer is convinced that the ciphertext is well formed, and decryption will not output reject. This property makes it possible to construct a simple and efficient PKE that supports threshold decryption with CCA security, as discussed in Chapter 22. In such a scheme the decryption key is shared among a number of parties, and a threshold of parties is needed to decrypt a given ciphertext. The technique in this section enables each party to ensure that the ciphertext is well formed before outputting its partial decryption. The details are provided in [30]. See also Exercise 22.14.

15.6.4.2 Signatures from IBE

A secure IBE scheme directly gives a secure signature scheme. Let $\mathcal{E}_{\text{id}} = (S, G, E, D)$ be a semantically secure IBE scheme with identity space $\mathcal{ID}$ and plaintext space $\mathcal{M}_{\text{IBE}} := \{0,1\}^n$, for some n. We construct a signature scheme for messages $m \in \mathcal{ID}$. The signature on m is the IBE secret key sk_m associated with identity m. To verify the signature, check that the key sk_m correctly decrypts ciphertexts encrypted for the identity m.

In more detail, the signature scheme $\mathcal{S} = (G', S', V')$ derived from $\mathcal{E}_{\text{id}}$ has message space $\mathcal{ID}$ and works as follows:

- $G'()$: run $(mpk, msk) \xleftarrow{\text{R}} G()$ and outputs (mpk, msk) as the signature key pair.
- $S'(msk, m)$: for $m \in \mathcal{ID}$, output $\sigma \xleftarrow{\text{R}} G(msk, m)$.
- $V'(mpk, m, \sigma)$: choose $t \xleftarrow{\text{R}} \mathcal{M}_{\text{IBE}}$, compute $c \xleftarrow{\text{R}} E(mpk, m, t)$, and accept if $D(\sigma, c) = t$.

The verifier computes the encryption of a random message $t \in \mathcal{M}_{\text{IBE}}$ under identity m, and accepts the signature σ if it correctly decrypts this ciphertext. We typically require that the verification algorithm V' be deterministic, but here V' is randomized. It can be de-randomized by deriving the required randomness from a random oracle applied to (m, σ), but we will not pursue that here.

The following theorem proves security of this scheme.

Theorem 15.9. *Let $\mathcal{E}_{\text{id}}$ be a semantically secure IBE scheme as in Definition 15.9, where the message space $|\mathcal{M}_{\text{IBE}}| = \{0,1\}^n$ is super-poly. Then the derived signature scheme $\mathcal{S}$ is secure.*

> *In particular, for every signature adversary $\mathcal{A}$ that attacks $\mathcal{S}$, there exist an IBE adversary $\mathcal{B}$ that attacks $\mathcal{E}_{\text{id}}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}] \le \mathsf{SSadv}[\mathcal{B}, \mathcal{E}_{\text{id}}] + \big(1/|\mathcal{M}_{\text{IBE}}|\big). \tag{15.32}$$

Proof sketch. Adversary $\mathcal{B}$ interacts with an IBE challenger for $\mathcal{E}_{\text{id}}$, and plays the role of a signature challenger to $\mathcal{A}$ with respect to $\mathcal{S}$. The IBE challenger sends $\mathcal{B}$ a master public key *mpk*, and $\mathcal{B}$ forwards this *mpk* to $\mathcal{A}$ as the signature public key. Next $\mathcal{A}$ issues a sequence of signing queries. For each query, our $\mathcal{B}$ responds by issuing the corresponding key query to its IBE challenger, and forwards the answer back to $\mathcal{A}$.

Eventually $\mathcal{A}$ outputs a signature forgery (m, σ), where $\mathcal{A}$ did not previously issue a signing query for m. Now $\mathcal{B}$ chooses two random messages $t_0, t_1 \xleftarrow{R} \mathcal{M}_{\text{IBE}} = \{0,1\}^n$, and issues an encryption query to its IBE challenger for identity m and messages t_0, t_1. It gets back a challenge IBE ciphertext $c \xleftarrow{R} E(mpk, m, t_b)$ for some $b \in \{0,1\}$. Then $\mathcal{B}$ computes $\hat{t} \leftarrow D(\sigma, c)$. If $\hat{t} = t_1$ it outputs $\hat{b} := 1$; otherwise, it outputs $\hat{b} := 0$. Observe that

- when $b = 1$, then $c \xleftarrow{R} E(mpk, m, t_1)$, and $\mathcal{B}$ outputs 1 with probability $\text{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$,
- when $b = 0$, then $c \xleftarrow{R} E(mpk, m, t_0)$, and $\mathcal{B}$ outputs 1 with probability $1/|\mathcal{M}_{\text{IBE}}|$.

Then $\mathsf{SSadv}[\mathcal{B}, \mathcal{E}_{\text{id}}] = \big|\text{SIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}] - 1/|\mathcal{M}_{\text{IBE}}|\big|$, from which (15.32) follows. □

To conclude this section, we note that the BLS signature scheme is closely related to the signature scheme obtained from applying Theorem 15.9 to the IBE scheme $\mathcal{E}_{\text{BF}}$ (Construction 1).

15.6.4.3 Searching on public-key encrypted data

Our third and final application for IBE shows how a server can test if an encrypted message contains a certain keyword, without the server learning anything else about the message.

In particular, consider an email server that processes emails on behalf of Alice. Alice wants the email server to implement the following policy: if the subject line in an incoming email contains a trigger word such as *urgent*, *boss*, or *Bob*, the server should send an alert to Alice's phone. Otherwise, the server should store the email in Alice's inbox for later processing. Alice can update the list of trigger words $\mathcal{W}$ at any time by notifying the mail server. In this section we assume that email senders behave honestly. We will not worry about a spam mailer that tries to incorrectly trigger text messages to Alice's phone. Spam filtering is an orthogonal problem.

Now, suppose Alice enables encrypted email by publishing a public key *pk* that senders can encrypt to. She keeps the secret decryption key to herself. The problem is that now the email server can no longer do its job, unless it has Alice's secret decryption key. Without Alice's key, the server cannot tell if the subject line contains one of the trigger words. Alice prefers not to give her decryption key to the mail server.

This raises a natural question: can Alice give the mail server a "weak" key that lets it test if the subject line contains a trigger words in $\mathcal{W}$, without learning anything else about the email.

IBE provides a solution to this problem. Let $\mathcal{E}_{\text{id}} = (S, G, E, D)$ be an IBE scheme with message space $\mathcal{M}_{\text{IBE}}$, and where the identity space $\mathcal{ID}$ contains all english words. Alice generates IBE parameters $(mpk, msk) \xleftarrow{\text{R}} S()$ and publishes mpk along with her public key pk. She sends to the mail server a list of secret IBE keys $sk_w \xleftarrow{\text{R}} G(msk, w)$, one key for each word $w \in \mathcal{W}$. Overall, the server receives $|\mathcal{W}|$ keys.

When Bob sends an email to Alice he does the following:

- First, encrypt the email with Alice's regular public key pk to obtain a ciphertext ct.
- Second, let $w_1, w_2, \ldots, w_n \in \mathcal{ID}$ be the words in the email subject line. Bob treats each word on the subject line as an IBE identity, and encrypts a random plaintext t using that identity. That is, Bob chooses $t \xleftarrow{\text{R}} \mathcal{M}_{\text{IBE}}$ and computes

$$c_i \leftarrow E(mpk, w_i, t), \quad \text{for } i = 1, \ldots, n.$$

- Third, Bob sends the email $(t, c_1, \ldots, c_n,\ ct)$ to Alice.

When the server receives an incoming encrypted email $(t, c_1, \ldots, c_n,\ ct)$ it tries to decrypt each of $c_1, \ldots, c_n$ using the list of secret keys at its disposal. This takes a total of $n \times |\mathcal{W}|$ decryption attempts. If for some $w \in \mathcal{W}$ and $i = 1, \ldots, n$ it obtains $D(sk_w, c_i) = t$, then it learns that the email subject contains the trigger word $w \in \mathcal{W}$, and can forward the (encrypted) email to Alice's phone. The reason the sender encrypts a random plaintext $t \in \mathcal{M}_{\text{IBE}}$ is to ensure that $D(sk_w, c_i) \neq t$ when c_i is not encrypted for identity w. We are assuming that the message space $\mathcal{M}_{\text{IBE}}$ is super-poly.

This way, the server learns what trigger words appear in the subject line, and how many times each word appears. It learns nothing else about the subject line, other than its length.

For this scheme to properly hide the subject line from an eavesdropper, the IBE scheme $\mathcal{E}_{id}$ must be a *private* IBE, as defined in Section 15.6.2.1. Otherwise, the ciphertexts $c_1, \ldots, c_n$ may expose the identities (words) under which they were encrypted. Hence, Construction 1 can be used here, but Construction 2 is more limited.

This approach is called **searching on encrypted data** because it lets the server search the subject line for particular words without revealing anything else about the subject line. It is closely related to a construction we discussed back in Section 6.12 for searching on data encrypted using a symmetric cipher. Here we show how to do the same for data encrypted using a public key system.

This problem raises a number of additional interesting questions:

- Better privacy: the scheme above reveals to the server what trigger words appeared in the subject line as well as their frequency. Is there a scheme that reveals nothing to the server other than the existence of a trigger word? The server should not learn what words appeared in the subject line.
- Spam filtering: Alice may want the email server to remove incoming spam emails, without learning anything about an incoming email other than its spam status. Can we run a spam checker on an encrypted email, without revealing anything about the email other than its spam status?

We will come back to these questions in the next section where we discuss functional encryption.

15.7 The functional encryption paradigm

All the encryption schemes discussed in the book so far have the following property: suppose Alice receives a ciphertext c that is the encryption of some message m. If Alice has the decryption key, she learns the entire plaintext m; if not, she learns nothing about m. We refer to this as the **binary nature of ciphers**: decryption is all or nothing. A functional encryption scheme relaxes this binary notion. The goal is to support many "weak" decryption keys, where each key only reveals specific partial information about the plaintext m, and nothing else.

To give an example, consider a mail server that processes Alice's encrypted emails. Every incoming email is encrypted under Alice's public key, which we will call mpk. The mail server needs to discard all spam emails before they reach Alice. The determination if an email is spam is done using a spam predicate $P : \mathcal{M} \to \{0,1\}$. The email is considered to be spam when $P(m) = 1$, and is benign otherwise. Alice wants to give the mail server a weak secret key sk_P that lets it determine the spam status of an email, but learn nothing else about the email. More precisely, if $c \xleftarrow{R} E(mpk, m)$ then $D(sk_P, c)$ should output $P(m) \in \{0,1\}$ and nothing else. This lets the mail server do its job, but learn nothing about the email contents beyond its spam status. Alice derives the weak key sk_P from her full decryption key, which we will call msk.

Functional encryption. Let us generalize this idea to define functional encryption more abstractly. A functional encryption scheme is designed for some function F,

$$F : \mathcal{F} \times \mathcal{M} \to \mathcal{Y},$$

where $\mathcal{F}$, $\mathcal{M}$, and $\mathcal{Y}$ are finite sets, and $\mathcal{Y}$ contains a special symbol $\perp \in \mathcal{Y}$. This F is called a **functionality**. For every $f \in \mathcal{F}$ we obtain a derived function $F(f,\cdot) : \mathcal{M} \to \mathcal{Y}$ acting on messages in $\mathcal{M}$. We call f a **function identifier** and we will sometime use f to denote the function $F(f,\cdot)$. Every secret key sk_f is associated with some $f \in \mathcal{F}$ and with the corresponding function $F(f,\cdot)$. If c is the encryption of a message $m \in \mathcal{M}$ then decrypting c using the secret key sk_f will output $f(m) := F(f,m)$, whenever $F(f,m) \neq \perp$. This key sk_f is called a **functional key** because it outputs a function of the plaintext m. When $F(f,m) = \perp$ the decryption algorithm is free to output whatever it wants.

In more detail, a functional encryption scheme operates as follows. As with IBE, one party runs the setup algorithm S to generate a master public key mpk and a master secret key msk. Anyone can use mpk to encrypt a message $m \in \mathcal{M}$ by invoking the encryption algorithm as $c \xleftarrow{R} E(mpk, m)$. The master secret key msk is used to derive functional keys: for $f \in \mathcal{F}$ there is a key generation algorithm G that is invoked as $G(msk, f)$ and outputs a functional key sk_f. Running the decryption algorithm $D(sk_f, c)$ outputs $F(f,m)$, whenever $F(f,m) \neq \perp$. The decryption algorithm may also output reject to indicate that c is an invalid ciphertext. This syntax is captured in the following definition.

Definition 15.13. *A* ***functional encryption scheme*** $\mathcal{FE} = (S, G, E, D)$ *for a functionality* $F : \mathcal{F} \times \mathcal{M} \to \mathcal{Y}$*, where* $\perp \in \mathcal{Y}$*, is a tuple of four efficient algorithms: a* ***setup algorithm*** S*, a* ***key generation algorithm*** G*, an* ***encryption algorithm*** E*, and a* ***decryption algorithm*** D*.*

- *S is a probabilistic algorithm invoked as $(mpk, msk) \xleftarrow{R} S()$, where mpk is called the **master public key** and msk is called the **master secret key**.*

- *G is a probabilistic algorithm invoked as $sk_f \xleftarrow{R} G(msk, f)$, where msk is the master secret key (as output by S), $f \in \mathcal{F}$ represents the function $F(f, \cdot) : \mathcal{M} \to \mathcal{Y}$, and sk_f is a secret key for this function.*

- *E is a probabilistic algorithm invoked as $c \xleftarrow{R} E(mpk, m)$.*

- *D is a deterministic algorithm invoked as $u \leftarrow D(sk_f, c)$. Here u is either a quantity in $\mathcal{Y}$, or a special* reject *value that is not in $\mathcal{Y}$.*

- *We require that the scheme is correct, namely that for all possible outputs (mpk, msk) of S, all messages $m \in \mathcal{M}$, and all function identifiers $f \in \mathcal{F}$, if $F(f, m) \neq \perp$ then*

$$\Pr\big[D\big(G(msk, f),\ E(mpk, m)\big) = F(f, m)\big] = 1.$$

- *We say that $\mathcal{FE}$ is defined over $(\mathcal{F}, \mathcal{M}, \mathcal{C})$, where $\mathcal{C}$ is some finite **ciphertext space** $\mathcal{C}$.*

Let's see a few examples of simple functionalities.

Example 15.1. A regular public key encryption scheme, as defined in Chapter 11, is a primitive functional encryption supporting only one function, the identity function $\boldsymbol{I}$. In particular, regular public key encryption implements the following functionality:

$$F_{\text{PKE}} : \{\boldsymbol{I}\} \times \mathcal{M} \to (\mathcal{M} \cup \{\perp\}) \qquad \text{where} \qquad F_{\text{PKE}}(\boldsymbol{I}, m) = m \text{ for all } m \in \mathcal{M}. \tag{15.33}$$

There is only one functional secret key, $sk_{\boldsymbol{I}}$, and it fully decrypts every well formed ciphertext. □

Example 15.2. An identity based encryption scheme defined over $(\mathcal{ID}, \mathcal{M}, \mathcal{C})$ is a functional encryption scheme that implements the following functionality

$$F_{\text{eq}} : \mathcal{ID} \times (\mathcal{ID} \times \mathcal{M}) \to (\mathcal{M} \cup \{\perp\})$$

where

$$F_{\text{eq}}\big(id',\ (id, m)\big) = \begin{cases} m & \text{if } id = id' \\ \perp & \text{otherwise.} \end{cases} \tag{15.34}$$

Here there is a functional secret key $sk_{id'}$ for every $id' \in \mathcal{ID}$. This key outputs m for a ciphertext $c \xleftarrow{R} E(mpk, (id, m))$ whenever $id = id'$. This captures the requirement that $sk_{id'}$ must properly decrypt ciphertexts encrypted for the identity id'. When $id \neq id'$, the functionality outputs $\perp$, and therefore the decryption algorithm can output anything. Hence, IBE implements the "equality" functionality. □

We will see a few more examples in the next subsection.

The null functional key sk_ϵ. Recall that a ciphertext c generated by a public key encryption scheme can leak the length of the encrypted plaintext to an observer. We capture this allowed leakage by including a special function identifier $\epsilon \in \mathcal{F}$ in the functionality $F : \mathcal{F} \times \mathcal{M} \to \mathcal{Y}$. For a public key scheme, we augment (15.33) by defining $F(\epsilon, m) := \text{len}(m)$. The corresponding functional key, called the **null key**, is denoted sk_ϵ. We treat the key sk_ϵ as a public value, and this lets any observer learn the length of the encrypted plaintext by running $D(sk_\epsilon, c)$ on a ciphertext c. Specifically, for a regular public-key encryption scheme, the set $\mathcal{F}$ contains two function identifiers, $\mathcal{F} := \{\boldsymbol{I}, \epsilon\}$.

For an IBE scheme, the null key sk_ϵ can have one of two roles. In a private IBE scheme, a ciphertext must not reveal the target identity, but can reveal the plaintext length. In this case, we augment (15.34) with $F\big(\epsilon, (id, m)\big) := \text{len}(m)$. In a regular IBE (i.e., a non private IBE) a ciphertext can reveal the target identity to an observer. In this case, we augment (15.34) with $F\big(\epsilon, (id, m)\big) := \big(id, \text{len}(m)\big)$. This means that everyone can learn the target identity id from a ciphertext c by running $D(sk_\epsilon, c)$.

Defining secure functional encryption. The main security requirement for functional encryption is collusion resistance: an adversary who obtains several functional secret keys should learn nothing about the contents of a challenge ciphertext, beyond what is revealed by the keys at its disposal. In particular, suppose the adversary obtains a list of keys $sk_{f_1}, \ldots, sk_{f_n}$ that includes the null key sk_ϵ. The adversary is also given a challenge ciphertext $c \xleftarrow{\text{R}} E(mpk, m)$, for some message m. Clearly the adversary can learn $f_1(m), \ldots, f_n(m)$ about m. We require that the adversary not learn anything else about m. There are several ways to capture this. Here we give a simple definition that is an adaptation of the standard definition of semantic security. We allow the adversary to choose two messages $m_0, m_1 \in \mathcal{M}$ such that $f_i(m_0) = f_i(m_1)$ for all $i = 1, \ldots, n$. It is then given the encryption of either m_0 or m_1 and should be unable to decide, with non-negligible advantage, which one it was given.

Attack Game 15.9 (semantic security). For a given functional encryption scheme $\mathcal{FE} = (S, G, E, D)$ for a functionality $F : \mathcal{F} \times \mathcal{M} \to \mathcal{Y}$, where $\epsilon \in \mathcal{F}$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $(b = 0, 1)$:

- The challenger computes $(mpk, msk) \xleftarrow{\text{R}} S()$, computes the null key $sk_\epsilon \leftarrow G(msk, \epsilon)$, and sends mpk and sk_ϵ to the adversary $\mathcal{A}$.
- $\mathcal{A}$ then makes a series of queries to the challenger. Each query can be one of two types:
  - *Key query:* the query consists of a function identifier $f \in \mathcal{F}$. The challenger computes the secret key $sk'_f \leftarrow G(msk, f)$, and sends sk'_f to $\mathcal{A}$.
  - *Encryption query:* the query consists of a pair $(m_0, m_1) \in \mathcal{M}^2$. The challenger computes $c \xleftarrow{\text{R}} E(mpk, m_b)$ and sends c to $\mathcal{A}$.

  We restrict the adversary to a single encryption query $(m_0, m_1) \in \mathcal{M}^2$.

  Query restriction: let $S \subseteq \mathcal{F}$ be the set of function identifiers in all the key queries issued by the adversary, including ϵ. We require that m_0, m_1 specified in the encryption query satisfy $f(m_0) = f(m_1)$ for all $f \in S$. This ensures that the adversary cannot trivially win the game using the keys at its disposal.

- At the end of the game, the adversary outputs a bit $\hat{b} \in \{0,1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. Define $\mathcal{A}$'s **advantage** with respect to $\mathcal{FE}$ as

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{FE}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 15.14 (semantic security). *A function encryption scheme $\mathcal{FE}$ is* ***semantically secure*** *if for all efficient adversaries $\mathcal{A}$, the value $\mathsf{SSadv}[\mathcal{A}, \mathcal{FE}]$ is negligible.*

Remark 15.13. While Definition 15.14 is sufficient for simple functionalities $F : \mathcal{F} \times \mathcal{M} \to \mathcal{Y}$, it can sometimes lead to unsatisfactory conclusions. For example, let $\pi : \mathcal{M} \to \mathcal{M}$ be a one-way permutation. Consider a public key encryption scheme where a ciphertext $c \xleftarrow{\text{R}} E(mpk, m)$ is designed to reveal $\pi(m)$ to an observer. That is, the null key sk_ϵ is such that $D(sk_\epsilon, c) := \pi(m)$. The only other supported functional key fully decrypts the ciphertext. The scheme can reveal $\pi(m)$ to an observer, but should not reveal m. Nevertheless, a trivial scheme where encryption is defined as $E(mpk, m) := m$ would be considered secure under Definition 15.14. This is because the only allowed encryption query is one where $m_0 = m_1$, in which case the adversary has no hope of getting any advantage in winning Attack Game 15.9. This example illustrates a potential difficulty with this definition. Nevertheless, for simple functionalities that do not involve computational hardness, Definition 15.14 is adequate. □

Robust functional encryption. Suppose Alice is given a secret key sk_f for one function, and Bob is given a secret key sk_g for another function. An adversary might try to create a malicious ciphertext c that causes these keys to decrypt to inconsistent values. Say, $D(sk_f, c) = a$ and $D(sk_g, c) = b$, but there is no message $m \in \mathcal{M}$ such that $f(m) = a$ and $g(m) = b$. The adversary should not be able to construct such a ciphertext c.

To give an example where this could be a problem, consider again the spam filtering settings above. The mail server is given a secret key sk_P for a spam predicate $P : \mathcal{M} \to \{0,1\}$. Alice has a full decryption key $sk_{\boldsymbol{I}}$, where $\boldsymbol{I}(m)$ is the identity function, $\boldsymbol{I}(m) = m$ for all $m \in \mathcal{M}$. The mail server only forwards to Alice emails c for which $D(sk_P, c) = 0$, so that Alice only sees non-spam email. However, suppose the adversary could create a ciphertext c such that $D(sk_P, c) = 0$, but $D(sk_{\boldsymbol{I}}, c) = m$ where m is spam, namely $P(m) = 1$. This encrypted email will reach Alice despite being spam, letting the adversary evade the spam filter.

A functional encryption scheme is said to be **robust** if there are no misleading ciphertexts as above. This is captured in the following definition.

Definition 15.15 (robust functional encryption). *Let $\mathcal{FE} = (S, G, E, D)$ be a functional encryption scheme defined over $(\mathcal{F}, \mathcal{M}, \mathcal{C})$. We say that $\mathcal{FE}$ is* ***robust*** *if for all possible outputs (mpk, msk) of S, and every ciphertext $c \in \mathcal{C}$, there is a message $m \in \mathcal{M}$ such that for all $f \in \mathcal{F}$*

$$\Pr\big[f(m) = D(sk_f, c)\big] = 1,$$

when $sk_f \xleftarrow{\text{R}} G(msk, f)$, $D(sk_f, c) \neq$ reject, and $f(m) \neq \perp$.

For the spam filtering application we need the functional encryption scheme to be robust, but that is not enough. The adversary can submit any ciphertext of its choice to the mail server, and then observe if it forwards the ciphertext to Alice. This is a type of decryption query in a chosen ciphertext security game. Therefore, to ensure confidentiality, the functional encryption scheme must also be chosen ciphertext secure. This ensures that an adversary cannot decrypt a target ciphertext by submitting a sequence of related ciphertexts to the mail server and observing the server's behavior. We refer to [120] for a discussion of chosen ciphertext secure functional encryption and its construction.

15.7.1 Sample functional encryption schemes from pairings

In this section we present a few functionalities that can be implemented efficiently using pairings. Expanding the set of functionalities that can be implemented efficiently is an active area of research. In Exercise 15.21 we describe a simple "brute force" construction that works well for functionalities $F : \mathcal{F} \times \mathcal{M} \to \mathcal{Y}$ where the set of functions $\mathcal{F}$ is small. Here we are interested in constructions for functionalities where $\mathcal{F}$ is large.

15.7.1.1 Functional encryption for low-degree polynomials

Consider a functional encryption scheme where the message space is $\mathcal{M} := \mathbb{Z}_q^n$, so that messages are n-tuples in $\mathbb{Z}_q^n$. Let $\mathcal{F}_d \subseteq \mathbb{Z}_q[X_1, \ldots, X_n]$ be the set of n-variate polynomials of total degree at most d. For $f \in \mathcal{F}_d$ and $m \in \mathcal{M}$, define the **polynomial evaluation functionality** F_d as

$$F_d(f, m) := f(m).$$

A functional encryption scheme for F_d operates as follows:

- a ciphertext c is an encryption of some n-tuple $m \in \mathcal{M}$,
- a functional secret key sk_f corresponds to a polynomial $f \in \mathcal{F}_d$,
- invoking the decryption algorithm as $D(sk_f, c)$ outputs $f(m) \in \mathbb{Z}_q$, and nothing else.

Suppose some authority holds the master secret key msk. When the authority releases a functional key sk_f, for some $f \in \mathcal{F}_d$, it is effectively giving permission to compute $f(m)$ from any encrypted tuple $c \xleftarrow{\text{R}} E(mpk, m)$. If the authority releases only a small number of functional keys, then only partial information can be obtained about an encrypted tuple $m \in \mathcal{M}$.

Linear functional encryption. Consider functional encryption for F_1, the set of n-variate linear polynomials. In this case, the polynomial f is simply a vector in $\mathbb{Z}_q^n$, and the function being computed is an inner product of f and the encrypted plaintext vector $m \in \mathbb{Z}_q^n$. For this reason functional encryption for F_1 is often called **inner product functional encryption**. If a party has n functional keys corresponding to n linearly independent vectors, then it can learn the entirety of m from $c \xleftarrow{\text{R}} E(mpk, m)$. If the party has fewer than n functional keys, then it only learns partial information about m.

As an example application, consider a set of points $(x_i, y_i) \in \mathbb{Z}_q^2$ for $i = 1, \ldots, n$. These points could represent medical data, financial data, census data, or some other sensitive data. The data owner wants to make the data available to researchers, but without revealing the data in the clear. Let $\boldsymbol{x} := (x_1, \ldots, x_n)$ and $\boldsymbol{y} := (y_1, \ldots, y_n)$. The x-coordinate values are public, but the y-coordinate values are private and need to be protected by encryption. Hence, when the data owner places the data on a public server, it publishes

$$\big(\boldsymbol{x}, \quad E(mpk, \boldsymbol{y})\big). \tag{15.35}$$

A researcher wants to study a certain subset of the points, say, corresponding to people in a certain age group or in a certain geographic region. Specifically, let $S \subseteq \{1, \ldots, n\}$ be a subset of the n points. The researcher wants to compute the least squares fit line of the points in S. Recall that the least squares fit line $y = ax + b$ is the line that minimizes the squared error $L(a, b) := \sum_{i \in S}[y_i - (ax_i + b)]^2$. This line $y = ax + b$ is computed using two inner-products with $\boldsymbol{y}$:

$$a := \langle \boldsymbol{a}, \boldsymbol{y} \rangle \quad \text{and} \quad b := \langle \boldsymbol{b}, \boldsymbol{y} \rangle$$

where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_q^n$ are explicit vectors derived from $\boldsymbol{x}$ and S. Because $\boldsymbol{x}$ is public, the researcher can calculate the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.

The researcher asks the data owner for permission to carry out the study. If approved, the data owner sends the functional keys $sk_{\boldsymbol{a}}$ and $sk_{\boldsymbol{b}}$, corresponding to the vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, to the researcher. The researcher can then use these keys, and the encrypted data, to compute the line $y = ax + b$ needed for her study. In fact, these functional keys let the researcher carry out her study on all such encrypted data sets. She can use $sk_{\boldsymbol{a}}$ and $sk_{\boldsymbol{b}}$ to calculate the least square fit line from any published data set that is encrypted as in (15.35). The researcher never needs to talk to the data owner again. Of course, if the researcher wants to study a different subset S' she must again obtain permission from the data owner.

There are a number of constructions available for this inner product functionality. In fact, one does not need pairings for this; a variant of ElGamal encryption is sufficient. See Exercise 15.22 for the details.

Quadratic functional encryption. For quadratic polynomials, when $d = 2$, a functional encryption scheme for F_2 enables the researcher to compute variance and standard deviation of subsets of the encrypted data in (15.35). Such schemes can be implemented efficiently using an encryption scheme based on pairings.

It is tempting to try and reduce the quadratic case to the linear case by asking the encryptor to encrypt all pairwise products. For example, if the data is the column vector $\boldsymbol{x} := (x_1, \ldots, x_n)^{\mathsf{T}}$, then the encryption algorithm could first compute the $n \times n$ matrix $\boldsymbol{X} := \boldsymbol{x} \cdot \boldsymbol{x}^{\mathsf{T}}$, and encrypt it using the linear functional encryption scheme discussed above. All quadratic functions on $\boldsymbol{x}$ can be expressed as linear functions on $\boldsymbol{X}$. Hence, a linear functional scheme seems to suffice. However, the ciphertext size is now quadratic in n. Moreover, this approach is not robust (as in Definition 15.15) — there is no guarantee that the encryptor carried out the expansion from $\boldsymbol{x}$ to $\boldsymbol{X}$ correctly. As mentioned above, using pairings, it is possible to construct a robust functional encryption scheme for F_2 where ciphertext size is linear in n.

For polynomials of degree three or higher one has to resort to more powerful tools that we will not discuss here. We refer to [7] for some example constructions.

15.7.1.2 Attribute based encryption

Attribute based encryption (ABE) is an important instance of functional encryption that is defined by predicates. A **predicate** is a function $P : \mathcal{X} \to \{0, 1\}$ that outputs one bit representing true or false. A **predicate family** is a set of predicates $\mathcal{P} := \big\{P : \mathcal{X} \to \{0, 1\}\big\}$. An **attribute based encryption scheme** or **ABE** defined over $(\mathcal{P}, \mathcal{X}, \mathcal{M})$ is a functional encryption scheme that operates as follows:

- a ciphertext c is an encryption of a pair $(x, m) \in \mathcal{X} \times \mathcal{M}$, where x is called the **attribute** and m is called the **data**,

- a functional secret key sk_P corresponds to a predicate $P \in \mathcal{P}$,

- invoking the decryption algorithm as $D(sk_P, c)$ outputs m if $P(x) = 1$. When $P(x) = 0$ the output of $D(sk_P, c)$ is undefined and can be anything.

We say that $\mathcal{X}$ is the **attribute space** and $\mathcal{M}$ is the **data space**.

Formally, an ABE scheme for a predicate family $\mathcal{P}$ is a functional encryption scheme for the functionality

$$F : \mathcal{P} \times (\mathcal{X} \times \mathcal{M}) \to (\mathcal{M} \cup \{\perp\})$$

defined as

$$F\big(P,\ (x, m)\big) := \begin{cases} m & \text{if } P(x) = 1, \text{ and} \\ \perp & \text{otherwise.} \end{cases}$$

We will define the null functional key sk_ϵ after we look at a few examples. An ABE scheme is secure if it is secure as a functional encryption scheme for the functionality F.

Examples. An IBE system is an ABE for the equality predicate family, namely $\mathcal{P}_{\text{eq}} := \{P_{id'}\}$, where $P_{id'}(id) = 1$ if and only if $id = id'$. We already saw this in (15.34) where we showed that IBE is a special case of functional encryption.

As another example, let $\mathcal{X} := \{1, \ldots, N\}$ for some integer N. Consider a hospital database that contains encrypted pairs (x, y), where $x \in \mathcal{X}$ is the numerical result of a blood test, and y is the patient's name. Let $\mathcal{P}_{\text{leq}} := \{P_t\}$ be the comparison predicate family: for $x, t \in \mathcal{X}$ we set $P_t(x) = 1$ if and only if $x \leq t$. A functional key sk_t for a predicate P_t reveals all patients whose test result is less than a threshold t, but keeps all other patient names hidden. A nurse could be given a key sk_t and use it to compile a list of all patients who need to be notified about the results of their test. The nurse learns nothing about the names of the remaining patients.

Another interesting predicate family comes from linear algebra. Here the attribute space is $\mathcal{X} := \mathbb{Z}_q^n$. The predicate family $\mathcal{P}$ is derived from vectors in $\mathcal{X}$. A predicate $P_{\boldsymbol{v}} \in \mathcal{P}$ corresponds to a vector $\boldsymbol{v} \in \mathcal{X}$ and is defined as: for $\boldsymbol{x} \in \mathcal{X}$ we set $P_{\boldsymbol{v}}(x) = 1$ if and only if $\boldsymbol{x}$ is orthogonal to $\boldsymbol{v}$. Then for a secret key $sk_{\boldsymbol{v}}$ and a ciphertext $c \xleftarrow{\text{R}} E(mpk, (\boldsymbol{x}, m))$, running $D(sk_{\boldsymbol{v}}, c)$ outputs m whenever $\boldsymbol{v}$ is orthogonal to $\boldsymbol{x}$. An ABE scheme for this predicate family is called an **inner product ABE**.

Private attribute based encryption. Recall that in the context of IBE we distinguished between IBE and private IBE. In an IBE, a ciphertext may reveal the intended target identity. In a private IBE a ciphertext must not reveal the target identity. Similarly, we differentiate between ABE and private ABE. In an ABE scheme, defined over $(\mathcal{P}, \mathcal{X}, \mathcal{M})$, a ciphertext may reveal the attribute $x \in \mathcal{X}$ embedded in the ciphertext. In a private ABE scheme the ciphertext must not reveal x.

Formally, we capture this difference using the null functional key sk_ϵ. Recall that sk_ϵ is used in a functional encryption scheme to identify what an observer can learn from an observed ciphertext. Let $c \xleftarrow{\text{R}} E\big(mpk, (x, m)\big)$ be an ABE ciphertext.

- In an ABE scheme, running $D(sk_\epsilon, c)$ outputs $\big(x,\ \mathrm{len}(m)\big)$, where $\mathrm{len}(m)$ is the length of the message m. This means that c may reveal both x and the length of m.
- In a private ABE scheme, running $D(sk_\epsilon, c)$ outputs only $\mathrm{len}(m)$. This means that c may reveal the length of m, but nothing else.

This distinction means that in a private ABE scheme, a ciphertext does not reveal the embedded attribute x. Private ABE is also called **predicate encryption**.

Constructions for ABE. Let $\mathcal{X} := \{0,1\}^n$ and let $\mathcal{P}$ be the family of predicates on $\mathcal{X}$ that are computed by a monotone boolean formula with at most s boolean gates. Recall that a boolean formula is a special case of a boolean circuit where the out-degree of every gate is at most one. A formula with n inputs can have at most depth $\log_2 n$. A monotone formula is a formula that uses only and and or gates (but no not gates).

There are several efficient ABE schemes for $\mathcal{P}$ using bilinear maps. The ciphertext size is linear in n and the private key size is linear in s. An example construction is given in [78].

There is also interest in constructions that go beyond formulas, and support ABE for circuits. It is not known how to construct such schemes from pairings. However, there are known constructions from multilinear maps (as defined in Section 15.8), and from lattices (as defined in Chapter 17). We refer to [34] for an example construction and a survey of the work in the area.

Constructions for private ABE. Private ABE is much harder to construct. There are efficient pairing-based constructions for inner-product private ABE [98], but not much else.

15.7.2 Variations on functional encryption

There are many fascinating variations on the basic functional encryption paradigm described above. Here we briefly survey a few variations, just to give a taste for the breadth of this topic.

Secret key functional encryption. This is a functional encryption scheme where the setup algorithm S outputs only one key sk, and that key is needed for key generation in algorithm G, as well as for encryption by E. This concept can be useful for securely storing personal data on a remote server. Alice can encrypt her data using sk before sending it to the server. Later, she wants the server to publish some function of her data. Alice can send a functional key to the server, and the server uses it to obtain and publish the result in the clear.

Delegatable functional encryption. Refers to a scheme where a functional key sk_f, for $f \in \mathcal{F}$, can be further restricted. Let $h : \mathcal{Y} \to \mathcal{Y}$ be some function and let $f' \in \mathcal{F}$ be the function index for the composition $h(f(m)) : \mathcal{M} \to \mathcal{Y}$. We say that the functional encryption scheme is delegatable, if there is an algorithm that takes as input the functional key sk_f and mpk, and outputs a restricted functional key $sk_{f'}$ for the function $h(f(\cdot))$.

Multi-input functional encryption (MIFE). Refers to a functionality that operates on multiple inputs, namely $F : \mathcal{F} \times \mathcal{M}^d \to \mathcal{Y}$. The decryption algorithm takes as input a functional key sk_f for $f \in \mathcal{F}$, and d independent ciphertexts $c_i \xleftarrow{\mathrm{R}} E(mpk, m_i)$ for $i = 1, \ldots, d$. It outputs $D(sk_f, c_1, \ldots, c_d) = F(f, m_1, \ldots, m_d) \in \mathcal{Y}$. We refer to [74] for the definition of security.

An interesting two-input MIFE is called **order revealing encryption**, where the message space is $\mathcal{M} := \{0, 1, \ldots, 2^n\}$ and the functionality implements the "less than" relation:

$$F(\mathrm{lt}, m_1, m_2) := \begin{cases} 1 & \text{if } m_1 < m_2, \\ 0 & \text{otherwise} \end{cases} .$$

The functional key sk_{lt} can be applied to two ciphertexts $c_1 \xleftarrow{\mathrm{R}} E(mpk, m_1)$ and $c_2 \xleftarrow{\mathrm{R}} E(mpk, m_2)$ by running $D(sk_{\mathrm{lt}}, c_1, c_2)$. The result reveals the relative ordering of m_1 and m_2 and nothing else about m_1 and m_2. This functionality is meaningful in the context of secret key functional encryption, and can potentially be used to carry out a binary search over an encrypted database index. Unfortunately, it has been shown to not work well in practice because it reveals too much information about the underlying data (see e.g., [59]).

Function hiding. Refers to a functional encryption scheme where a functional key sk_f does not reveal the function f. This is not always possible, but can be constructed in some settings [38, 22].

15.8 Multilinear maps

While bilinear maps lead to impressive encryption and signature schemes, there are still many problems that cannot be solved using bilinear maps, most notably efficient functional encryption for more sophisticated functionalities than the ones mentioned in Section 15.7.1.

A stronger algebraic primitive, called a cryptographic **multilinear map**, lets us solve these and many other open problems in cryptography. In this section we define multilinear maps and present some of their applications. However, we need to point out that at the time of this writing, there are no known cryptographic multilinear maps, beyond bilinear maps. This remains one of the central open problems in cryptography.

To simplify the notation, we will focus on *symmetric* multilinear maps, although one can equally define asymmetric multilinear maps.

Definition 15.16 (a multilinear map). *Let $\mathbb{G}$ and $\mathbb{G}_T$ be cyclic groups of prime order q where $g \in \mathbb{G}$ is a generator. A **d-way multilinear map**, or simply a **d-linear map**, is an efficiently computable function $e : \mathbb{G}^d \to \mathbb{G}_T$ satisfying the following properties:*

1. *multilinear: if $u_1, \ldots, u_d \in \mathbb{G}$ and $\alpha_1, \ldots, \alpha_d \in \mathbb{Z}_q$ then*

$$e\left(u_1^{\alpha_1}, \ldots, u_d^{\alpha_d}\right) = e(u_1, \ldots, u_d)^{\alpha_1 \cdots \alpha_d} \qquad \textit{and}$$

2. *non-degenerate: $g_T := e(g, \ldots, g)$ is a generator of $\mathbb{G}_T$.*

At the very least, we want the discrete log problem to be hard in the group $\mathbb{G}$. As usual, for some applications we will need the n-linear map to satisfy stronger complexity assumptions. A standard hardness assumption on multilinear maps is called the **decision multilinear Diffie-Hellman** assumption, or **decision MDH**, which is a direct generalization of decision BDH.

Attack Game 15.10 (Decision multilinear Diffie-Hellman). Let $e : \mathbb{G}^d \to \mathbb{G}_{\mathrm{T}}$ be a d-linear map. For a given adversary $\mathcal{A}$, we define two experiments:

Experiment b $\quad$ ($b = 0, 1$):

- For $i = 1, \ldots, d+1$ the challenger computes $\alpha_i \xleftarrow{R} \mathbb{Z}_q$ and $u_i \leftarrow g_i^{\alpha_i} \in \mathbb{G}$. Let $\beta \xleftarrow{R} \mathbb{Z}_q$ and set
$$z^{(0)} \leftarrow e\underbrace{(g, \ldots, g)}_{d\text{-way map}}{}^{\alpha_1 \cdots \alpha_{d+1}} \in \mathbb{G}_T, \qquad z^{(1)} \leftarrow e\underbrace{(g, \ldots, g)}_{d\text{-way map}}{}^{\beta} \in \mathbb{G}_T.$$
The challenger gives $(u_1, \ldots, u_{d+1},\ z^{(b)}) \in \mathbb{G}^{d+1} \times \mathbb{G}_T$ to the adversary.

- The adversary outputs some $\hat{b} \in \{0, 1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage in solving the d-way decision Diffie-Hellman problem for** e as

$$\mathrm{DMDH}\mathsf{adv}[\mathcal{A}, e, d] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 15.17 (Decision MDH assumption). *We say that the* d***-way multilinear Diffie-Hellman (Decision MDH)*** *assumption holds for* e *if for all efficient adversaries* $\mathcal{A}$ *the quantity* $\mathrm{DMDH}\mathsf{adv}[\mathcal{A}, e, d]$ *is negligible.*

For $d = 2$ this definition reduces to decision BDH for a symmetric bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. Currently, we have no candidate d-linear map for $d > 2$, where discrete log is hard in the group $\mathbb{G}$. One can hope that the appropriate mathematical structure needed to construct such maps will eventually be discovered. Nevertheless, we briefly discuss two applications, in the hope that one day it will be possible to instantiate them with a secure d-linear candidate.

Group key exchange. In Chapter 10 we saw how to use the Diffie-Hellman protocol for anonymous key exchange among honest parties (also called toy key exchange). The protocol can be described using a shared public bulletin board between n parties. Let $\mathbb{G}$ be a cyclic group of prime order q with generator $g \in \mathbb{G}$. At setup time, every party $i = 1, \ldots, n$ chooses a secret $\alpha_i \xleftarrow{R} \mathbb{Z}_q$ and writes $u_i \leftarrow g^{\alpha_i} \in \mathbb{G}$ to the bulletin board. Once all the parties finish uploading their u_i values, all parties download $u_1, \ldots, u_n$. At any later time, if parties i and j need a shared secret $k_{i,j}$ they can compute it as $k_{i,j} := g^{\alpha_i \cdot \alpha_j} \in \mathbb{G}$. Party i computes $k_{i,j} = (u_j)^{\alpha_i}$ and party j computes $k_{i,j} = (u_i)^{\alpha_j}$. This process is called **non-interactive key-exchange** because after the initial upload and download, there is no more interaction between the parties. Note that only $O(n)$ data is written to the bulletin board.

A natural question is whether this can be generalized beyond pair-wise keys. Suppose $(d+1) \leq n$ parties want to setup a group key so that they can converse as a group. The group key should be known to all group participants, but to no one outside the group. Again, the only communication allowed is a short upload message for setup followed by a download. Once this setup is finished, every group of $(d+1)$ parties should be able to establish a group key with no further communication.

Here is a direct generalization of Diffie-Hellman to do just that. The protocol uses a d-way multilinear map $e : \mathbb{G}^n \to \mathbb{G}_T$.

- At setup time, party i, for $i = 1, \ldots, n$, chooses $\alpha_i \xleftarrow{R} \mathbb{Z}_q$ and uploads $u_i \leftarrow g^{\alpha_i} \in \mathbb{G}$ to the bulletin board. All parties then download $u_1, \ldots, u_n \in \mathbb{G}$.

- A group secret among parties $S := \{1, \ldots, d+1\}$ is defined as $k_S := g_T^{\alpha_1 \cdot \alpha_2 \cdots \alpha_{d+1}} \in \mathbb{G}_T$ (recall that $g_T := e(g, \ldots, g)$ is a generator of $\mathbb{G}_T$). Party $i \in S$ computes this secret using the d-linear map as:
$$k_S = e(u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_d)^{\alpha_i} \in \mathbb{G}_T.$$

Of course, there is nothing special about the set $\{1, \ldots, d\}$. Every subset of d users in $\{1, \ldots, n\}$ can compute a group secret this way with no interaction among the group members.

Assuming the setup step was carried out by honest parties, it is not difficult to argue that if the d-way Decision MDH holds for e, then even if all the users outside of S get together, they cannot distinguish $k_S \in \mathbb{G}_{\mathrm{T}}$ from a random element in $\mathbb{G}_{\mathrm{T}}$.

We know how to solve the non-interactive group key exchange problem for $d = 2$ using basic Diffie-Hellman. We know how to solve this problem for $d = 3$ (i.e. groups of three users) using the protocol above and a bilinear map. For $d > 3$ there is currently no practical solution to this problem, although several polynomial time, but non-practical, methods are known [35].

Software obfuscation. We briefly mention that multilinear maps are used in several constructions for a secure program obfuscator. An obfuscator $\mathcal{O}$ for a circuit C produces a circuit $C' \xleftarrow{\mathrm{R}} \mathcal{O}(C)$ that is functionally equivalent to C, namely $C(x) = C'(x)$ for all inputs x. However, C' "hides" internal secrets embedded in the implementation of C. We refer to [139] for the definition of obfuscation and its many applications.

15.9 A fun application: fair exchange of signatures

Alice and Bob want to sign a contract $m \in \mathcal{M}$. They both need to sign the contract: let σ_{a} be Alice's signature on m, and let σ_{b} be Bob's signature on m. They want a **fair exchange protocol** for the signatures: at the end of the protocol either they both hold σ_{a} and σ_{b}, or neither holds both signatures. The protocol should prevent a situation where one of them holds both signatures and the other does not. This is also called an **atomic swap protocol**.

One solution to this problem uses a technique called *gradual release.* Here Alice and Bob take turns revealing one bit of their signature at a time: Alice sends the first bit of σ_{a} to Bob, and Bob responds with the first bit of σ_{b}. Then Alice sends the second bit of σ_{a} to Bob, and Bob responds with the second bit of σ_{b}. This continues until they exchange all the bits. Of course every revealed bit must be accompanied by a proof that convinces the peer that the transmitted bit is correct. Now, suppose Alice terminates the protocol when ℓ bits are still hidden. Then Alice can obtain σ_{b} with at most 2^{ℓ} work by trying all 2^{ℓ} suffixes until she obtains Bob's valid signature on m. Bob is at most one bit behind Alice, and can therefore obtain Alice's signature with at most $2^{\ell+1}$ work. We see that if one side aborts early, then both parties can recover the other's signature with roughly equal work.

While gradual release may seem fair, this approach is somewhat problematic. First, the protocol requires many rounds of communication between the parties, which can be slow. Second, there is often an asymmetry in the computing resources of the two parties. Alice might be a large corporation, where as Bob is an individual who only has a laptop. Alice may have a thousand times more computing resources than Bob. She can then terminate the protocol at a point where she can recover Bob's signature in one day, while Bob will require a thousand days to recover Alice's signature. For these reasons, gradual release is not typically used in practice.

Instead, Alice and Bob decide to use a different technique called **optimistic fair exchange**. Their idea is to use an *offline trusted party* Tracy. Tracy generates a key pair $(pk_{\mathrm{t}}, sk_{\mathrm{t}})$ for a public key encryption scheme (G, E, D). She publishes pk_{t} and keeps her sk_{t} secret. Tracy is otherwise uninvolved in the protocol. She is a trusted party and is assumed to behave honestly.

Protocol 15.1. As a first attempt, Alice and Bob consider using the following (broken) protocol.

- *Alice:* encrypts her signature by computing $c_{\text{a}} \xleftarrow{\text{R}} E(pk_{\text{t}}, \sigma_{\text{a}})$, and sends c_{a} to Bob;
- *Bob:* once c_{a} is received, sends σ_{b} to Alice;
- *Alice:* checks that σ_{b} is Bob's valid signature on m, and if so, sends σ_{a} to Bob.

If both parties follow the protocol honestly then they both obtain both signatures, and there is no need to contact the trusted party Tracy. However, if Alice aborts and does not send σ_{a} to Bob in the last round, then Bob can ask Tracy to help him obtain σ_{a}:

- *Bob:* sends $(c_{\text{a}}, m, \sigma_{\text{b}})$ to Tracy;
- *Tracy:* verifies that (i) $\sigma := D(sk_{\text{t}}, c_{\text{a}}, d)$ is Alice's valid signature on m, and (ii) σ_{b} is Bob's valid signature on m. If so, she sends σ to Bob, and sends σ_{b} to Alice.

Here we are assuming that Tracy already has Alice's and Bob's public keys. □

The reason Tracy gives the signatures to both Alice and Bob is to ensure that Bob is not misbehaving by contacting Tracy without sending his signature σ_{b} to Alice. By giving the signature to both parties, Tracy ensures that if Bob gets Alice signature, then Alice gets Bob's signature.

A careful reader will notice that there are two glaring problems with Protocol 15.1. The first problem is that Alice could send the encryption of junk in her first message. She is supposed to send $c_{\text{a}} \xleftarrow{\text{R}} E(pk_{\text{t}}, \sigma_{\text{a}})$ to Bob, but suppose that instead she sends $\hat{c}_{\text{a}} \xleftarrow{\text{R}} E(pk_{\text{t}}, 0)$. If the encryption scheme (G, E, D) is semantically secure, then Bob cannot tell the difference between $\hat{c}_{\text{a}}$ and c_{a}. Once Bob sends his signature σ_{b} to Alice in the second round, Alice could walk away and never send her signature to Bob. Now Bob has no recourse: sending $\hat{c}_{\text{a}}$ to Tracy does not help him obtain σ_{a}.

To prevent this, Alice must somehow convince Bob that c_{a} is an encryption of her signature on the message m. This can be done using the tools developed in Chapter 20, and in particular, in Exercise 20.12. However, when using the BLS signature scheme, there is no need for sophisticated tools. It is quite easy for Bob to check validity of c_{a}. Let us see how.

A simple instantiation using BLS signatures. Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\text{T}}$ be a pairing where all three groups have prime order q, and where $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. Let $H : \mathcal{M} \to \mathbb{G}_0$ be a hash function. Suppose Alice has a BLS key pair (pk_{a}, α), and Bob has a BLS key pair (pk_{b}, β), where $\alpha, \beta \in \mathbb{Z}_q$ are the secret keys. Tracy publishes $(u_0 := g_0^{\tau},\ u_1 := g_1^{\tau})$ and keeps $\tau \xleftarrow{\text{R}} \mathbb{Z}_q$ secret. She is otherwise uninvolved.

Protocol 15.2. Alice and Bob sign a contract $m \in \mathcal{M}$. Let $\sigma_{\text{a}} := H(m)^{\alpha} \in \mathbb{G}_0$ be Alice's signature on m, and $\sigma_{\text{b}} := H(m)^{\beta} \in \mathbb{G}_0$ be Bob's signature on m. They use the following protocol.

- *Alice:* construct a multiplicative ElGamal encryption of σ_{a} as follows: choose $\rho \xleftarrow{\text{R}} \mathbb{Z}_q$, set $v_0 \leftarrow g_0^{\rho},\ \ w_0 \leftarrow u_0^{\rho} \cdot \sigma_{\text{a}}$, and send $c_{\text{a}} := (v_0, w_0) \in \mathbb{G}_0^2$ to Bob;
- *Bob:* check validity of $c_{\text{a}} = (v_0, w_0)$ by verifying that

  $$e(w_0, g_1) = e(v_0, u_1) \cdot e(H(m), pk_{\text{a}}), \tag{15.36}$$

  and if so, send σ_{b} to Alice;

- *Alice:* check that σ_b is Bob's valid signature on m, and if so, send σ_a to Bob.

If all goes well then both parties obtain both signatures, and there is no need to contact the trusted party Tracy. However, if Alice does not send σ_a to Bob in the last round, then Bob sends to Tracy $\big(c_\mathrm{a} = (v_0, w_0),\ m,\ \sigma_\mathrm{b}\big)$. Tracy verifies that (i) σ_b is Bob's valid BLS signature on m, and (ii) $\sigma := w_0/v_0^\tau$ is Alice's valid BLS signature on m. If so, she sends σ to Bob, and sends σ_b to Alice. □

We claim that this protocol unconditionally guarantees that if Alice obtains σ_b, then Bob obtains σ_a, even if Alice is malicious and sends a malformed pair (v_0, w_0) in her first message. In particular, let us argue that the BLS signature σ that Bob receives from Tracy must be Alice's valid signature on m. Using (15.36) we obtain that

$$\begin{aligned} e(\sigma, g_1) = e(w_0/v_0^\tau,\ g_1) = \frac{e(w_0, g_1)}{e(v_0^\tau,\ g_1)} &= \frac{e(v_0, u_1) \cdot e(H(m), pk_\mathrm{a})}{e(v_0^\tau, g_1)} \\ &= \frac{e(v_0, u_1) \cdot e(H(m), pk_\mathrm{a})}{e(v_0, g_1^\tau)} = \frac{e(v_0, u_1) \cdot e(H(m), pk_\mathrm{a})}{e(v_0, u_1)} = e(H(m), pk_\mathrm{a}). \end{aligned}$$

Therefore, $\sigma = w_0/v_0^\tau$ is Alice's BLS signature on m with respect to pk_a, as required.

The reader may be wondering how is it that Bob is able to peek through the veil of ElGamal encryption and verify that the plaintext is a valid BLS signature. The reason this is possible is because DDH is false in a pairing group, as explained in (15.8), and therefore multiplicative ElGamal is not semantically secure. Bob exploits this to verify that c_a is well formed.

Nevertheless, even though multiplicative ElGamal is not semantically secure, Bob cannot extract Alice's signature σ_a from her ciphertext $c_\mathrm{a} = (v_0, w_0)$. Indeed, one can show that if Bob could compute σ_a given only $(pk_\mathrm{a}, u_0, u_1, m, v_0, w_0)$, then Bob could also break co-CDH for e.

A cheating Bob. We mentioned earlier that there are two glaring problems with Protocol 15.1. We addressed one problem, a cheating Alice, by using BLS signatures and the validity check in (15.36). The second problem is that Protocol 15.1 is vulnerable to a cheating Bob.

Consider the following attack. Suppose Alice and Bob want to sign a contract m using Protocol 15.1. The protocol begins with Alice sending her well-formed encrypted signature c_a to Bob. A cheating Bob hands c_a to his friend Carol. Carol signs m herself to obtain σ_c, and sends $(c_\mathrm{a}, m, \sigma_\mathrm{c})$ to Tracy. Tracy decrypts c_a and sends Alice's signature σ_a to Carol, which Carol forwards to Bob. Tracy also sends Carol's signature σ_c to Alice. Now Alice has a problem: Bob is holding Alice's signature on m, however Alice is holding Carol's signature on m, which is not what Alice wants. This shows that Bob and Carol working together can cheat Alice.

The problem is that Tracy cannot tell who c_a is intended for. A simple solution is to require that the contract m explicitly specify the names, addresses, and public keys of the two affected parties. Tracy checks that the signatures being released are valid under those public keys, and if they are not, she aborts. If the signatures are valid then Tracy sends them to the two addresses specified in m. Informally, this ensures that if Bob receives σ_a then Alice must receive σ_b, as required. It is sufficient to secure Protocol 15.2.

For Protocol 15.1, a general solution is to use a CCA secure encryption scheme that supports associated data, as discussed in Section 12.7. Here the encryption algorithm $c \overset{\mathrm{R}}{\leftarrow} E(pk, m, d)$ takes a third argument, the associated data. Similarly, the decryption algorithm $D(sk, c, d')$ takes a third argument, and decryption fails if $d \neq d'$. When Alice runs Protocol 15.1 with Bob, she will encrypt her signature as $c_{\mathrm{a}} \xleftarrow{\mathrm{R}} E(pk_{\mathrm{t}}, \sigma_{\mathrm{a}}, d)$ where the associated data is $d := (m, id_{\mathrm{a}}, id_{\mathrm{b}})$ where id_{a} is Alice's name, address, and public key, and id_{b} contains the same information for Bob. If Carol asks Tracy to decrypt c_{a}, Tracy will attempt to run $D(sk_{\mathrm{t}}, c_{\mathrm{a}}, d')$ where $d' := (m, id_{\mathrm{a}}, id_{\mathrm{c}})$, and decryption will fail.

15.10 Notes

Citations to the literature to be added.

15.11 Exercises

15.1 (A chosen ciphertext attack on elliptic-curve ElGamal). Let $E/\mathbb{F}_p$ be an elliptic curve where $q := |E(\mathbb{F}_p)|$ is a prime number and $P \in E(\mathbb{F}_p)$ is a generator. Assume that the ICDH assumption holds for the group $E(\mathbb{F}_p)$ and consider the ElGamal encryption scheme $\mathcal{E}_{\mathrm{EG}}$ from Section 12.4 implemented over this group. The decryption algorithm $D\big(\alpha, (V, c)\big)$ operates as in Section 12.4: it computes $W \leftarrow \alpha V,\ k \leftarrow H(V, W),\ m \leftarrow D_{\mathrm{s}}(k, c)$, and outputs m. Here H is a function $H : \mathbb{F}_p^4 \to \mathcal{K}$ (the domain is $\mathbb{F}_p^4$ because V and W are in $\mathbb{F}_p^2$). We will treat the secret key α as an integer in $[0, q)$.

In Remark 12.1 we stressed that algorithm D must check that the given point V is in $E(\mathbb{F}_p)$, which means verifying that $V = (x_0, y_0)$ satisfies the curve equation $E :\ y^2 = x^3 + ax + b$. Let's show that if D skips this check, then the scheme breaks completely under a chosen ciphertext attack. Here we assume that αV is computed using the group law as described in Section 15.2. Observe that these group law equations are independent of the constant term b. For every $V_1 = (x_1, y_1) \in \mathbb{F}_p^2$ there exists some $b_1 \in \mathbb{F}_p$ such that V_1 is a point on the curve $E_1 :\ y^2 = x^3 + ax + b_1$. Then, if the adversary issues a CCA query for the ciphertext (V_1, c), algorithm D will first compute $W_1 \leftarrow \alpha V_1 \in E_1(\mathbb{F}_p)$.

(a) Suppose that $|E_1(\mathbb{F}_p)|$ is divisible by t. Show that the adversary can learn $\alpha \bmod t$, with probability close to 1, after at most t CCA queries.

(b) Use part (a) to show an efficient CCA adversary that learns the secret key α with probability close to 1. You may assume that if b_1 is uniform in $\mathbb{F}_p$ then $|E_1(\mathbb{F}_p)|$ is approximately uniform in the interval $[p + 1 - 2\sqrt{p},\ p + 1 + 2\sqrt{p}]$. Recall that there is an efficient algorithm to compute $|E_1(\mathbb{F}_p)|$ (see Remark 15.1).

To simplify the analysis of your adversary's success probability, you may model $H : \mathbb{F}_p^4 \to \mathcal{K}$ as a random oracle and assume that the symmetric cipher provides one-time ciphertext integrity.

Discussion: This attack, called an **invalid curve attack**, illustrates the importance of Remark 12.1 for security of the ElGamal system $\mathcal{E}_{\mathrm{EG}}$. More generally, it shows that when receiving an elliptic curve point as a pair $(x, y) \in \mathbb{F}_p^2$, one should verify that the point is on the expected curve $E(\mathbb{F}_p)$.

15.2 (Multiplication without the y-coordinate). In this exercise we show that the y-coordinate of a point is not needed for many cryptographic systems. Let $E/\mathbb{F}_p$ be an elliptic curve $y^2 = x^3 + ax + b$ and let $P \neq \mathcal{O}$ be a point in $E(\mathbb{F}_p)$. We write $x(P)$ for the x-coordinate of the point P.

(a) For an integer $\alpha > 0$, let $x_\alpha := x(\alpha P)$. We leave x_α undefined if $\alpha P = \mathcal{O}$. Use the addition law to show that the following formula computes $x_{2\alpha}$ and $x_{2\alpha+1}$ from $x_\alpha, x_{\alpha+1}, x_1$:

$$\text{if } (2\alpha)P \neq \mathcal{O}: \qquad x_{2\alpha} = \frac{(x_\alpha^2 - a)^2 - 8bx_\alpha}{4(x_\alpha^3 + ax_\alpha + b)} \tag{15.37}$$

$$\text{if } (2\alpha+1)P \neq \mathcal{O} \text{ and } x_1 \neq 0: \qquad x_{2\alpha+1} = \frac{(a - x_\alpha x_{\alpha+1})^2 - 4b(x_\alpha + x_{\alpha+1})}{x_1(x_\alpha - x_{\alpha+1})^2} \tag{15.38}$$

Note that $(2\alpha)P \neq \mathcal{O}$ implies that the y-coordinate of αP is non-zero and therefore the denominator of (15.37) is non-zero. Similarly, $(2\alpha+1)P \neq \mathcal{O}$ implies that $\pm\alpha P \neq (\alpha+1)P$ and therefore $x_\alpha \neq x_{\alpha+1}$, so that the denominator of (15.38) is non-zero.

(b) Use part (a) to give an algorithm, similar to repeated squaring, for computing x_α from x_1, when $x_1 \neq 0$. Your algorithm should take $\lceil \log_2 \alpha \rceil$ steps where at every step it constructs the pair $x_\gamma, x_{\gamma+1}$ for an appropriate choice of $\gamma \in \mathbb{Z}$.

Discussion: The algorithm in part (b) is called the **Montgomery ladder**. Its running time depends on the number of bits in α, but not on the value of α. This can help defend against timing attacks.

15.3 (Group law for Montgomery curves). Recall that an elliptic curve $E/\mathbb{F}_p$ in Montgomery form is given as $By^2 = x^3 + Ax^2 + x$ for some $A, B \in \mathbb{F}_p$. Work out a formula for the group law for this curve using the chord and tangent method, as on page 618.

15.4 (Montgomery ladder on Montgomery curves). A Montgomery curve $E : By^2 = x^3 + Ax^2 + x$, where $A, B \in \mathbb{F}_p$, is well suited for x-coordinate point multiplication as in Exercise 15.2. For a point $\mathcal{O} \neq P \in E(\mathbb{F}_p)$ we write $x(P)$ for the x-coordinate of P. Consider the sequence $X_1/Z_1,\ X_2/Z_2, \ldots$ where $X_1 := x(P),\ Z_1 := 1$, and

$$\begin{aligned} X_{2\alpha} &:= (X_\alpha^2 - Z_\alpha^2)^2 & X_{2\alpha+1} &:= 4Z_1(X_\alpha X_{\alpha+1} - Z_\alpha Z_{\alpha+1})^2 \\ Z_{2\alpha} &:= 4X_\alpha Z_\alpha(X_\alpha^2 + AX_\alpha Z_\alpha + Z_\alpha^2) & Z_{2\alpha+1} &:= 4X_1(X_\alpha Z_{\alpha+1} - Z_\alpha X_{\alpha+1})^2 \end{aligned}$$

Use Exercise 15.3 to show that $x(\alpha P) = X_\alpha/Z_\alpha$ whenever $\alpha P \neq \mathcal{O}$.

Discussion: As in part (b) of Exercise 15.2, we can use these equations to compute $x(\alpha P)$ in $\log_2 \alpha$ steps. By combining like terms in these equations, each step requires only 11 multiplications in $\mathbb{F}_p$ [17]. Note that choosing A to be small further speeds up the group operation.

15.5 (Repeated CDH game). Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$ be a pairing. Consider the following t-repeated CDH game, analogous to Attack Game 13.4:

- The challenger computes

  $$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u_1 \leftarrow g_1^\alpha \in \mathbb{G}_1, \quad y_1, \ldots, y_t \xleftarrow{\text{R}} \mathbb{G}_0$$

  and sends $(u_1, y_1, \ldots, y_t)$ to $\mathcal{A}$.

- $\mathcal{A}$ makes a sequence of *reveal queries*. Each reveal query consists of an index $j \in \{1, \ldots, t\}$. Given j, the challenger sends $x_j := y_j^\alpha \in \mathbb{G}_0$ to $\mathcal{A}$.

- Eventually the adversary outputs (ν, x), where $\nu \in \{1, \ldots, t\}$ and $x \in \mathbb{G}_0$.

We say that $\mathcal{A}$ wins the game if index ν is not among $\mathcal{A}$'s reveal queries, and $x = y_\nu^\alpha$. We define $\mathcal{A}$'s advantage, denoted rCDH$\mathsf{adv}[\mathcal{A}, e, t]$, as the probability that $\mathcal{A}$ wins the game.

Show that for every t-repeated CDH adversary $\mathcal{A}$ that makes at most Q reveal queries, there exists a co-CDH adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{rCDH}\mathsf{adv}[\mathcal{A}, e, t] \leq 2.72 \cdot (Q+1) \cdot \text{coCDH}\mathsf{adv}[\mathcal{B}, e]. \tag{15.39}$$

Hint: The proof is essentially the same as the proof of Lemma 13.6.

15.6 (An insecure hash function for BLS). Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_\text{T}$ be a pairing where all three groups have prime order q. Let g_0, h_0 be two random generators of $\mathbb{G}_0$ chosen at setup and treated as system parameters. Let $\hat{H} : \mathcal{M} \to \mathbb{Z}_q^2$ be a collision resistant hash function. Define $H : \mathcal{M} \to \mathbb{G}_0$ as $H(m) := g_0^{m_0} h_0^{m_1}$, where $(m_0, m_1) \leftarrow \hat{H}(m) \in \mathbb{Z}_q^2$. Observe that H is collision resistant assuming discrete log in $\mathbb{G}_0$ is hard. Show that the BLS signature scheme from Section 15.5.1 is insecure when used with the hash function H.

15.7 (BLS blind signatures). In Exercise 13.15 we defined the concept of a blind signature scheme, and presented a construction based on the RSA signature scheme.

(a) Construct a blind signature scheme based on the BLS signature scheme. Your construction should mirror the construction in Exercise 13.15.

(b) In Exercise 13.15 we defined what it means for a blind signature scheme to be secure. Prove security of the BLS blind signature scheme from part (a) based on the 1MDH assumption from Section 11.6.3, assuming H is modeled as a random oracle.

15.8 (Insecure proofs of possession). In Section 15.5.3.2 we described an aggregate signature scheme where every public key $pk = (g_1^\alpha,\ \pi)$ includes a proof of possession $\pi := H'\big(g_1^\alpha\big)^\alpha \in \mathbb{G}_0$. Here $H' : \mathbb{G}_1 \to \mathbb{G}_0$ is a hash function.

(a) Suppose the hash function H' maps all inputs to a fixed output $w_0 \in \mathbb{G}_0$. That is, $H'(u) = w_0$ for all $u \in \mathbb{G}_1$. Show that the resulting aggregate signature scheme is insecure due to a rogue public key attack. Show how to compute a proof of possession for your rogue public key.

(b) Recall that $H : \mathcal{M} \to \mathbb{G}_0$ is the hash function used for signing messages in $\mathcal{M}$. Suppose that $\mathbb{G}_1 \subseteq \mathcal{M}$ and we set $H'(u) := H(u)$ for all $u \in \mathbb{G}_1$. That is, we use H for computing proofs of possession. Show that the resulting aggregate signature scheme is vulnerable to a rogue public key attack.
Hint: in your attack, the aggregate forger issues one query to its signing oracle to produce the proof of possession that it needs for the rogue public key.

15.9 (Aggregating proofs of possession). In Section 15.5.3.2 we saw an aggregate signature scheme where every public key $pk = (g_1^\alpha,\ \pi)$ includes a proof of possession $\pi := H'\big(g_1^\alpha\big)^\alpha \in \mathbb{G}_0$. Here $H' : \mathbb{G}_1 \to \mathbb{G}_0$ is a hash function. In Remark 15.8 we mentioned the idea of reducing public key storage by aggregating proofs of possession across many public keys. Specifically, for a collection of n public keys $\{(u_i, \pi_i)\}_{i=1}^n$, define the aggregate proof of possession π_ag as $\pi_\text{ag} := \pi_1 \cdots \pi_n \in \mathbb{G}_0$. The aggregate verification algorithm $VA(\boldsymbol{u}, \pi_\text{ag}, \boldsymbol{u}', \boldsymbol{m}', \sigma_\text{ag})$ takes as input the complete list of all public keys $\boldsymbol{u} = (u_1, \ldots, u_n) \in \mathbb{G}_1^n$ and their aggregate proof of possession $\pi_\text{ag} \in \mathbb{G}_0$. It also takes a subset $\boldsymbol{u}' = (u_1', \ldots, u_\ell') \in \mathbb{G}_1^\ell$ of the public keys in $\boldsymbol{u}$, and a vector of messages $\boldsymbol{m}' = (m_1', \ldots, m_\ell')$.

It accepts σ_{ag} if (1) π_{ag} is valid, namely $e(\pi_{\text{ag}}, g_1) = \prod_{i=1}^{n} e\big(H'(u_i), u_i\big)$, and (2) σ_{ag} is valid, namely $e(\sigma_{\text{ag}}, g_1) = \prod_{i=1}^{\ell} e\big(H(m'_i), u'_i\big)$, and (3) all the public keys in $\boldsymbol{u'}$ are also in $\boldsymbol{u}$. As usual, checking validity of π_{ag} need only be done once; it need not be repeated for every aggregate verification.

(a) Modify the aggregate signature security game (Attack Game 15.2) to account for an aggregate forger that outputs a tuple $(\boldsymbol{u}, \pi_{\text{ag}}, \boldsymbol{u'}, \boldsymbol{m'}, \sigma_{\text{ag}})$ as a forgery. Note that the forged aggregate signature can be with respect to a subset $\boldsymbol{u'}$ of the public keys in $\boldsymbol{u}$. State your attack game specifically for the scheme in this exercise.

(b) Show that if the adversary is required to output a forgery $(\boldsymbol{u}, \pi_{\text{ag}}, \boldsymbol{u'}, \boldsymbol{m'}, \sigma_{\text{ag}})$ for which $\boldsymbol{u} = \boldsymbol{u'}$ (and hence $\ell = n$) and $\boldsymbol{m'}$ satisfies $m'_1 = \ldots = m'_\ell$, then one can prove security with respect to your attack game from part (a) under the same assumptions as in Theorem 15.2. The proof requires a small modification to the proof of Theorem 15.2 part (b).

Discussion: The scheme in this exercise can be proven secure even when $\boldsymbol{u} \neq \boldsymbol{u'}$, but the proof relies on a much stronger assumption than in Theorem 15.2. This exercise shows that in the special case where every aggregate signature comes with an attached aggregate proof of possession for the same set of public keys, then we can prove security under the same assumption as in Theorem 15.2.

15.10 (Strongly binding aggregation, part I). In this exercise we show that the signature aggregation method $\mathcal{SA}^{(1)}_{\text{BLS}}$ from Section 15.5.3.1 is strongly binding as in Definition 15.6. Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\text{T}}$ be a pairing where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_{\text{T}}$ are cyclic groups of prime order q. Consider the following **pairing collision** problem: the adversary is given as input a random pair $(u_0, v_0) \xleftarrow{\text{R}} \mathbb{G}_0^2$, and needs to output $(u_1, v_1) \in \mathbb{G}_1^2$, where $(u_1, v_1) \neq (1, 1)$, such that

$$e(u_0, u_1) = e(v_0, v_1).$$

We say that pairing collision is hard for e if no efficient adversary can solve the pairing collision problem with non-negligible probability.

(a) Show that the aggregation scheme $\mathcal{SA}^{(1)}_{\text{BLS}}$ is strongly binding, as in Definition 15.6, assuming pairing collision is hard for e and the hash function $H : \mathbb{G}_1 \times \mathcal{M} \to \mathbb{G}_0$ is a random oracle.
Hint: Let $\mathcal{A}$ be a strong binding adversary for $\mathcal{SA}^{(1)}_{\text{BLS}}$. Your pairing collision algorithm $\mathcal{B}$ will respond to a query from $\mathcal{A}$ for $H(pk_i, m_i)$ by setting $H(pk_i, m_i) := u_0^{\rho_i} v_0^{\tau_i}$ where $\rho_i, \tau_i \xleftarrow{\text{R}} \mathbb{Z}_q$.

(b) For certain asymmetric pairings (i.e., $\mathbb{G}_0 \neq \mathbb{G}_1$) it is believed that DDH holds in $\mathbb{G}_0$. Show that if DDH holds in $\mathbb{G}_0$ then pairing collision is hard for e.

15.11 (Strongly binding aggregation, part II). In this exercise we show that the proof of possession signature aggregation method from Section 15.5.3.2 is not strongly binding. Suppose that two malicious signers, Carol and David, generate two valid public keys $pk_{\text{c}} = (u_{\text{c}}, \pi_{\text{c}})$ and $pk_{\text{d}} = (u_{\text{d}}, \pi_{\text{d}})$ where they deliberately ensure that $u_{\text{c}} \cdot u_{\text{d}} = 1$.

(a) Show that $\sigma_{\text{ag}} = 1 \in \mathbb{G}_0$ is a valid aggregate signature with respect to $(pk_{\text{c}}, pk_{\text{d}})$ for every $m \in \mathcal{M}$. That is, $VA\big((pk_{\text{c}}, pk_{\text{d}}),\ (m, m),\ 1\big) = \mathsf{accept}$ for all $m \in \mathcal{M}$.

(b) Part (a) suggests that Carol's and David's behavior is irrational, since now an adversary can forge their aggregate signature on any message m. However, let's see how this setup can help them cause signer confusion. Let σ'_{ag} be a valid aggregate signature for some message m with respect to valid public keys $\boldsymbol{pk} = (pk_1, \ldots, pk_n)$. Show that σ'_{ag} is also a valid aggregate signature for m with respect to $\boldsymbol{pk'} := (pk_1, \ldots, pk_n, pk_{\text{c}}, pk_{\text{d}})$. That is, show that if

$VA\big(\boldsymbol{pk}, m^n, \sigma'_{\text{ag}}\big) = \textsf{accept}$ then $VA\big(\boldsymbol{pk}', m^{n+2}, \sigma'_{\text{ag}}\big) = \textsf{accept}$, where $m^n := (m, \ldots, m) \in \mathcal{M}^n$ and $m^{n+2} := (m, \ldots, m) \in \mathcal{M}^{n+2}$.

Discussion: In effect, Carol and David caused a collision between the set of signers $\boldsymbol{pk}'$ that includes them, and a set of honest signers $\boldsymbol{pk}$ that does not: the signature σ'_{ag} could have been generated by either $\boldsymbol{pk}$ or $\boldsymbol{pk}'$. This can potentially be exploited in a setting where the parties who generated (m, σ'_{ag}) are entitled to a reward. Carol and David can inject themselves into the set of signers and claim a portion of the reward. This can be easily mitigated by the augmentation method in part (c).

(c) Let's show that any secure signature aggregation scheme $\mathcal{SA} = (G, S, V, A, VA)$ can be made strongly binding. To do so, the aggregator includes a collision resistant hash of $(\boldsymbol{pk}, \boldsymbol{m})$ in the aggregate signature. Specifically, for a collision resistant hash function H, let $\mathcal{SA}' = (G, S, V, A', VA')$ be the following modification of $\mathcal{SA}$:

- $A'(\boldsymbol{pk}, \boldsymbol{m}, \boldsymbol{\sigma})$: set $h \leftarrow H(\boldsymbol{pk}, \boldsymbol{m})$ and $\sigma_{\text{ag}} \xleftarrow{\text{R}} A(\boldsymbol{pk}, \boldsymbol{\sigma})$, and output $\sigma'_{\text{ag}} := (h, \sigma_{\text{ag}})$.
- $VA'\big(\boldsymbol{pk},\ \boldsymbol{m},\ \sigma'_{\text{ag}} = (h, \sigma_{\text{ag}})\big)$: accept if $h = H(\boldsymbol{pk}, \boldsymbol{m})$ and $VA(\boldsymbol{pk}, \boldsymbol{m}, \sigma_{\text{ag}}) = \textsf{accept}$.

Show that $\mathcal{SA}'$ is a secure aggregation scheme that is also strongly binding, as in Definition 15.6, assuming $\mathcal{SA}$ is secure and H is collision resistant. Note that the cost of this augmentation is a small increase in the size of the aggregate signature, since the aggregate signature must now include the hash h.

15.12 (Another secure aggregation method). Let us see a secure signature aggregation scheme that retains many of the benefit of proofs of possession from Section 15.5.3.2, but without expanding the public key. Let $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\text{T}}$ be a pairing where $\mathbb{G}_0, \mathbb{G}_1, \mathbb{G}_{\text{T}}$ are cyclic groups of prime order q. We will need a hash function $H_n : \mathbb{G}_1^n \to \mathcal{C}^n$ where $\mathcal{C} \subseteq \mathbb{Z}_q$ is super-poly. The aggregation scheme, denoted $\mathcal{SA}^{(3)}_{\text{BLS}}$, is the same as $\mathcal{SA}_{\text{BLS}}$ in (15.10) except that aggregation and verification now operate as follows.

- $A\big(\boldsymbol{pk},\ \boldsymbol{\sigma}\big)$: Let $\boldsymbol{pk} = (pk_1, \ldots, pk_n) \in \mathbb{G}_1^n$ and $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \mathbb{G}_0^n$.
  - compute $(\tau_1, \ldots, \tau_n) \leftarrow H_n(\boldsymbol{pk}) \in \mathcal{C}^n$ and output $\sigma_{\text{ag}} \leftarrow \sigma_1^{\tau_1} \cdots \sigma_n^{\tau_n} \in \mathbb{G}_0$.
- $VA\big(\boldsymbol{pk},\ \boldsymbol{m} \in \mathcal{M}^n,\ \sigma_{\text{ag}}\big)$: Let $\boldsymbol{pk} = (pk_1, \ldots, pk_n)$ and $\boldsymbol{m} = (m_1, \ldots, m_n)$.
  - verify that $pk_i \neq 1$ for all $i = 1, \ldots, n$; otherwise reject and stop,
  - compute $(\tau_1, \ldots, \tau_n) \leftarrow H_n(\boldsymbol{pk}) \in \mathcal{C}^n$,
  - accept if $e(\sigma_{\text{ag}}, g_1) = e\big(H(m_1), pk_1\big)^{\tau_1} \cdots e\big(H(m_n), pk_n\big)^{\tau_n}$.

The security of this scheme against a forgery attack is shown in [33]. Here we consider other properties of this scheme.

(a) Show that when all the messages in $\boldsymbol{m}$ are the same, namely $m := m_1 = \cdots m_n$, the aggregate verification algorithm can be rewritten to use only two pairings instead of n. This shows that the optimization in (15.11) applies to this scheme.

(b) Show that if pairing collision, as defined in Exercise 15.10, is hard for e, and H and H_n are modeled as random oracles, then $\mathcal{SA}^{(3)}_{\text{BLS}}$ is strongly binding as in Definition 15.6.

15.13 (One-time aggregate signatures). In Exercise 14.11 we presented a simple one-time signature scheme whose security is based on the discrete log assumption in a group $\mathbb{G}$ of order q.

(a) Show that this scheme is aggregatable. In particular, given a vector of n signatures $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n) \in \mathbb{Z}_q^n$ on n messages $\boldsymbol{m} = (m_1, \ldots, m_n) \in \mathcal{M}^n$ issued under n public keys $\boldsymbol{pk} = (pk_1, \ldots, pk_n) \in (\mathbb{G}^2)^n$, let $\sigma_{\text{ag}} := \sigma_1 + \ldots + \sigma_n \in \mathbb{Z}_q$ be the aggregate signature. Explain how the aggregate verification algorithm $\mathit{VA}(\boldsymbol{pk}, \boldsymbol{m}, \sigma_{\text{ag}})$ works.

(b) Show that there is a rogue public key attack on this basic scheme.

(c) Suppose we enhance the scheme from part (a) using the message augmentation technique from Section 15.5.3.1. Show that there is still a rogue public key attack on this scheme.

(d) Let $H_n : (\mathbb{G}^2)^n \to \mathbb{Z}_q^n$ be a hash function. Suppose we enhance the scheme from part (a) by defining the aggregate signature σ_{ag} as $\sigma_{\text{ag}} \leftarrow \sigma_1\tau_1 + \ldots + \sigma_n\tau_n \in \mathbb{Z}_q$, where $(\tau_1, \ldots, \tau_n) \leftarrow H_n(\boldsymbol{pk}) \in \mathbb{Z}_q^n$. Explain how aggregate verification works. Then prove that the resulting aggregate signature scheme is one-time secure if the discrete log assumption holds in the group $\mathbb{G}$, and the hash functions H and H_n are modeled as random oracles. Specifically, show that your enhanced scheme satisfies Definition 15.5 against an adversary that issues at most one signature query to its challenger.

15.14 (Completing the proof of Theorem 15.3). In this exercise we show how to compute the value $e(g_0, h_1)^{(\alpha^{Q+2})}$ needed to complete the proof of Theorem 15.3. We have at our disposal a polynomial $P \in \mathbb{Z}_q[X]$ of degree $Q+1$, the data from (15.24), a message $m \in \mathbb{Z}_q$, and a pair $(r, w) \in \mathbb{Z}_q \times \mathbb{G}_0$ such that $P(m) \neq r$ and $w = g_0^{(P(\alpha)-r)/(\alpha-m)}$, where $m \neq \alpha$.

(a) First, show how to derive the quantity $g_0^{1/(\alpha-m)} \in \mathbb{G}_0$ from (r, w) and the data from (15.24).
Hint: First show that $\big(P(X) - r\big)/(X - m) = p_1(X) + s/(X - m)$ for some polynomial $p_1 \in \mathbb{Z}_q[X]$ of degree Q and some $0 \neq s \in \mathbb{Z}_q$. Then use (15.22) to compute $y := g_0^{p_1(\alpha)}$.

(b) Now that we have a pair $(m,\ g_0^{1/(\alpha-m)})$, show how to use h_1 and $h_1^{(\alpha^{Q+3})}$ from (15.24) to compute the quantity $t := e(g_0, h_1)^{\left(\alpha^{Q+2}+p_2(\alpha)\right)}$, where $p_2 \in \mathbb{Z}_q[X]$ is a known polynomial of degree at most $Q+1$.

Hint: Use the fact that $(X^{Q+3} - m^{Q+3})/(X - m) = X^{Q+2} + p_2(X)$ where $\deg(p_2) \leq Q+1$.

(c) Show how to calculate $y := e(g_0, h_1)^{p_2(\alpha)}$ using (15.22). Then $t/y = e(g_0, h_1)^{(\alpha^{Q+2})}$ is the required value needed to solve $(Q+1)$-CDH and complete the proof of Theorem 15.3.

15.15 (A non-strong signature scheme). Consider again the signature scheme $\mathcal{S}_{\text{G}}$ from Section 15.5.4. Suppose that we modify the signing algorithm so that instead of generating the nonce $r \in \mathbb{Z}_q$ using a PRF, the signer simply chooses a fresh random r in $\mathbb{Z}_q$ for every signature. Show that the resulting signature scheme is not strongly secure.

15.16 (Proof of Theorem 15.4). In this exercise we prove security of the signature scheme $\mathcal{S}_{\text{BB}}$ from Section 15.5.4.1. We construct an adversary $\mathcal{B}$ that uses a signature forger $\mathcal{A}$ to solve a given Q-CDH challenge with secret parameter $\alpha \in \mathbb{Z}_q$. Adversary $\mathcal{B}$ constructs the public key $pk := (u_1, v_1, h_{\text{T}})$ to give to $\mathcal{A}$ as follows. First, it chooses random $\rho_1, \ldots, \rho_Q \xleftarrow{\text{R}} \mathbb{Z}_q$, and constructs the polynomial

$$P(X) := \prod_{i=1}^{Q}(X - \rho_i) = \sum_{i=0}^{Q} \gamma_i \cdot X^i \in \mathbb{Z}_q[X].$$

Then $\mathcal{B}$ uses (15.22) to compute $h_0 \leftarrow g_0^{P(\alpha)}$ and sets $h_\mathrm{T} \leftarrow e(h_0, g_1)$. Next, $\mathcal{B}$ chooses $\delta \xleftarrow{\mathrm{R}} \mathbb{Z}_q$ and $b \xleftarrow{\mathrm{R}} \{0,1\}$ and uses g_1^α from the Q-CDH challenge to set:

$$\text{if } b = 0, \text{ set } u_1 \leftarrow g_1^\alpha,\ v_1 \leftarrow g_1^\delta, \qquad\qquad \text{if } b = 1, \text{ set } u_1 \leftarrow g_1^\delta,\ v_1 \leftarrow g_1^\alpha.$$

It obtains the public key $pk := (u_1, v_1, h_\mathrm{T})$, which it sends to $\mathcal{A}$. Note that $\mathcal{A}$ learns nothing about b.

(a) Show that $\mathcal{B}$ can construct $h_0^{1/(\alpha-\rho_i)}$ for all $i = 1, \ldots, Q$.

(b) Use part (a) to show that in both cases, $b = 0$ and $b = 1$, our $\mathcal{B}$ can answer all of $\mathcal{A}$'s signature queries. Signature query number i is answered using $\rho_i \in \mathbb{Z}_q$ and $h_0^{1/(\alpha-\rho_i)} \in \mathbb{G}_0$.

(c) Show that with probability at least 1/2, the final forgery (m, σ) from $\mathcal{A}$ enables $\mathcal{B}$ to obtain a pair $(\rho,\ h_0^{1/(\alpha-\rho)})$, where $\rho \in \mathbb{Z}_q$ satisfies $\rho \notin \{\rho_1, \ldots, \rho_Q\}$.
Hint: When $b = 0$, $\mathcal{B}$ knows a list of signatures for m, one for each $\rho \in \{\rho_1, \ldots, \rho_Q\}$. When $b = 1$, $\mathcal{B}$ knows another list of signatures for m. Show that if the forgery σ is in exactly one of the two lists, then with probability 1/2 our $\mathcal{B}$ can construct the required pair $(\rho,\ h_0^{1/(\alpha-\rho)})$, where $\rho \notin \{\rho_1, \ldots, \rho_Q\}$. Use the fact that $\mathcal{A}$ knows nothing about b. This fails if (m, σ) is on both lists. Show that in that case, $\mathcal{B}$ can directly compute the secret $\alpha \in \mathbb{Z}_q$. If you get stuck, see [29].

(d) Show that $\mathcal{B}$ can construct $(\rho,\ g_0^{1/(\alpha-\rho)})$ from $(\rho,\ h_0^{1/(\alpha-\rho)})$. Use the fact that $\rho \notin \{\rho_1, \ldots, \rho_Q\}$.
Hint: Use the fact that $P(X)/(X - \rho) = p(X) + s/(X - \rho)$ for some polynomial p of degree at most $Q - 1$ and $0 \neq s \in \mathbb{Z}_q$.

(e) Now, use parts (b) and (c) of Exercise 15.14 to solve the given Q-CDH challenge.

15.17 (Bounded collusion IBE). Consider a secure IBE, as in Definition 15.9, but where the adversary is limited to a *single* key query. This models an environment where users do not collude, so that every user sees only one decryption key. In this restricted setting we can construct IBE directly from the ElGamal encryption scheme $\mathcal{E}_\mathrm{EG} = (G_\mathrm{EG},\ E_\mathrm{EG},\ D_\mathrm{EG})$ from Section 11.5, without any pairings. Recall that $\mathcal{E}_\mathrm{EG}$ operates in a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$. Let $H : \mathcal{ID} \to \mathbb{Z}_q$ be a hash function. The derived IBE scheme (S, G, E, D) has identity space $\mathcal{ID}$ and the same message space $\mathcal{M}$ as $\mathcal{E}_\mathrm{EG}$. The IBE scheme works as follows:

- $S()$: setup runs as follows

$$\alpha, \beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \quad u \leftarrow g^\alpha \in \mathbb{G}, \quad v \leftarrow g^\beta \in \mathbb{G},$$
$$mpk \leftarrow (u, v) \in \mathbb{G}^2, \quad msk \leftarrow (\alpha, \beta) \in \mathbb{Z}_q^2, \quad \text{output } (mpk, msk).$$

- $G(msk, id)$: key generation for $id \in \mathbb{Z}_q$ using $msk = (\alpha, \beta)$ runs as

$$r \leftarrow H(id) \in \mathbb{Z}_q, \quad \sigma \leftarrow r \cdot \alpha + \beta \in \mathbb{Z}_q, \quad \text{output } sk_{id} := \sigma.$$

- $E(mpk, id, m)$: encryption using $mpk = (u, v)$ runs as:

$$r \leftarrow H(id) \in \mathbb{Z}_q, \quad pk_{id} \leftarrow u^r \cdot v \in \mathbb{G}, \quad c \leftarrow E_\mathrm{EG}(pk_{id}, m), \quad \text{output } c.$$

- $D(sk_{id}, c)$: output $D_\mathrm{EG}(sk_{id}, c)$.

Notice that secret keys in this scheme correspond to signatures in the one-time signature scheme from Exercise 14.11.

(a) Show that this IBE is scheme is secure against an adversary that issues at most a single key query, assuming $\mathcal{E}_{\text{EG}}$ is semantically secure, and H is modeled as a random oracle.

(b) This scheme is clearly not secure against an adversary that issues two key queries. Give a construction that is secure against such adversaries, under the same assumptions as in part (a). Use Exercise 14.11 part (b).

15.18 (CCA secure IBE). In this exercise we construct a chosen ciphertext secure IBE from a semantically secure IBE, in the random oracle model. The construction uses the technique from Section 12.6. Exercise to be written.

15.19 (An IBE from the signature scheme $\mathcal{S}_{\text{G}}$). In this exercise we construct an efficient IBE, called $\mathcal{E}_{\text{G}}$, where secret keys correspond to signatures in the signature scheme $\mathcal{S}_{\text{G}}$ from Section 15.5.4. The scheme is proven secure without random oracles based on the decision d-CDH assumption, where $d = Q + 1$, and Q is the number of key queries from the adversary. The scheme $\mathcal{E}_{\text{G}}$ has identity space $\mathcal{ID} := \mathbb{Z}_q$ and message space $\mathcal{M}$. It uses a pairing e, a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$ defined over $(\mathcal{K}_{\text{E}}, \mathcal{M}, \mathcal{C})$, a hash function $H : \mathbb{G}_1 \times \mathbb{G}_{\text{T}}^2 \to \mathcal{K}_{\text{E}}$, and a PRF F defined over $(\mathcal{K}_{\text{F}}, \mathbb{Z}_q)$. The IBE scheme $\mathcal{E}_{\text{G}} = (S, G, E, D)$ works as follows:

- $S()$: setup runs as follows

$$\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_q, \quad k_{\text{F}} \xleftarrow{\text{R}} \mathcal{K}_{\text{F}}, \quad u_1 \leftarrow g_1^{\alpha}, \quad v_0 \leftarrow g_0^{\beta}, \quad h_{\text{T}} \leftarrow e(v_0, g_1) \in \mathbb{G}_{\text{T}},$$
$$mpk \leftarrow (u_1, h_{\text{T}}), \quad msk \leftarrow (\alpha, \beta, k_{\text{F}}), \quad \text{output } (mpk, msk).$$

- $G(msk, id)$: key generation for $id \in \mathbb{Z}_q$ using $msk = (\alpha, \beta, k_{\text{F}})$ runs as:

$$r \xleftarrow{\text{R}} F(k_{\text{F}}, id) \in \mathbb{Z}_q, \quad w_0 \leftarrow g_0^{(\beta - r)/(\alpha - id)}, \quad \text{output } sk_{id} \leftarrow (r, w_0) \ \in \mathbb{Z}_q \times \mathbb{G}_0.$$

  Note that w_0 is undefined when $id = \alpha$. In that case we set $w_0 \leftarrow 1$ and $r \leftarrow \beta$.

- $E(mpk, id, m)$: encryption using $mpk = (u_1, h_{\text{T}})$ and $g_{\text{T}} := e(g_0, g_1)$ runs as:

$$\rho \xleftarrow{\text{R}} \mathbb{Z}_q, \quad x_1 \leftarrow (u_1 \cdot g_1^{-id})^{\rho}, \quad y \leftarrow g_{\text{T}}^{\rho}, \quad t \leftarrow h_{\text{T}}^{\rho},$$
$$k_{\text{E}} \leftarrow H(x_1, y, t), \quad c \xleftarrow{\text{R}} E_{\text{s}}(k_{\text{E}}, m), \quad \text{output } (x_1, y, c).$$

- $D\big(sk_{id},\ (x_1, y, c)\big)$: to decrypt (x_1, y, c) using secret key $sk_{id} = (r, w_0)$ do:

$$t \leftarrow e(w_0, x_1) \cdot y^r, \quad k_{\text{E}} \leftarrow H(x_1, y, t), \quad m \leftarrow D_{\text{s}}(k_{\text{E}}, c), \quad \text{output } m.$$

(a) Show that $D(sk_{id}, c)$ properly decrypts $c \xleftarrow{\text{R}} E(mpk, id, m)$.

(b) Security of this scheme relies on a decision version of the d-CDH assumption. Formulate a decision version of the d-CDH assumption from Definition 15.7. The goal is to distinguish $z := e(g_0, h_1)^{(\alpha^{d+1})} \in \mathbb{G}_{\text{T}}$ from a random element in $\mathbb{G}_{\text{T}}$, given the data in (15.21).

(c) Prove that $\mathcal{E}_{\text{G}}$ is a secure private IBE scheme, as in Definition 15.10, assuming decision d-CDH holds for e for poly-bounded d, and assuming F is a secure PRF, H is a secure KDF, and $\mathcal{E}_{\text{s}}$ is semantically secure. The proof proceeds along the same lines as the proof of Theorem 15.3.

- The Q key queries from the adversary are answered as signatures queries in Theorem 15.3, using the data provided in the decision d-CDH challenge, where $d = Q + 1$.
- When the adversary issues its encryption query $\big((id_0, m_0), (id_1, m_1)\big)$, first choose a random $b \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \{0, 1\}$ and compute the secret key (r, w_0) for the identity id_b as in the previous bullet. The challenge ciphertext sent back to the adversary is constructed using the d-CDH challenge data in (15.24) as:
  $$x_1 \leftarrow h_1^{(\alpha^{Q+3})} \cdot h_1^{-id_b^{Q+3}}, \quad y \leftarrow e\big(g_0^{T(\alpha)},\ h_1\big) \cdot z, \quad t \leftarrow e(w_0, x_1) \cdot y^r,$$
  $$k_{\mathrm{E}} \leftarrow H(x_1, y, t), \quad c \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} E_{\mathrm{S}}(k_{\mathrm{E}}, m_b), \quad \text{output } (x_1, y, c).$$
  Here $z \in \mathbb{G}_{\mathrm{T}}$ is the $(Q + 1)$-CDH challenge, and T is the degree $(Q + 1)$ polynomial $T(X) := (X^{Q+3} - id_b^{Q+3})/(X - id_b) - X^{Q+2}$.

Show that when $z = e(g_0, h_1)^{(\alpha^{Q+1})}$ then (x_1, y, c) is a proper encryption of m_b under id_b. When z is uniform in $\mathbb{G}_{\mathrm{T}}$ then (x_1, y, c) is independent of b. Use this to complete the proof of security.

15.20 (Identity based key exchange). An identity based non-interactive key exchange (ID-NIKE) lets two parties, who know each other's identity, generate a shared secret without exchanging any messages. More precisely, an ID-NIKE is a triple of algorithms (S, G, K) where S is invoked as $(mpk, msk) \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} S()$. As usual, mpk is given to all participants in the system, while msk is kept secret at a trusted entity Tracy. A user with identity id can request from Tracy the secret key $sk_{id} \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} G(msk, id)$. Say Alice has $sk_{\mathrm{a}} \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} G(msk, id_{\mathrm{a}})$ and Bob has $sk_{\mathrm{b}} \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} G(msk, id_{\mathrm{b}})$. When Alice and Bob want to generate a shared secret, Alice locally runs $k_{\mathrm{a}} \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} K(sk_{\mathrm{a}}, id_{\mathrm{b}})$ and Bob locally runs $k_{\mathrm{b}} \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} K(sk_{\mathrm{b}}, id_{\mathrm{a}})$. The scheme is correct if $k_{\mathrm{a}} = k_{\mathrm{b}}$, so that they both obtain the same secret, denoted $k_{\mathrm{a,b}}$.

(a) We say that an ID-NIKE scheme is secure if an efficient adversary who can request secret keys sk_{id} for identities id of his choice, cannot compute the secret k_{id_1,id_2} between id_1 and id_2, assuming it did not request the secret keys for either id_1 or id_2. Formulate this security property as a game between a challenger and an adversary.

(b) Let $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_{\mathrm{T}}$ be a symmetric pairing. Consider the following ID-NIKE scheme (S, G, K): algorithms S and G are the same as in the IBE scheme $\mathcal{E}_{\mathrm{BF}}$ from Section 15.6.3.1. Recall that these algorithms use a hash function $H : \mathcal{ID} \to \mathbb{G}$, and a secret key for identity id_{x} is defined as $sk_{\mathrm{x}} := H(id_{\mathrm{x}})^{\alpha} \in \mathbb{G}$, where $\alpha \in \mathbb{Z}_q$ is the master secret. For identities id_{x} and id_{y}, algorithm $K\big(sk_{\mathrm{x}}, id_{\mathrm{y}}\big)$ is defined as $K\big(sk_{\mathrm{x}}, id_{\mathrm{y}}\big) := e\big(sk_{\mathrm{x}},\ H(id_{\mathrm{y}})\big)$. Show that this scheme is correct and secure, assuming decision BDH holds for e, and H is modeled as a random oracle.

(c) Explain how to adapt the scheme from part (b) to use an asymmetric pairing $e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_{\mathrm{T}}$ where $\mathbb{G}_0 \neq \mathbb{G}_1$.

15.21 (The trivial functional encryption scheme). Let $F : \mathcal{F} \times \mathcal{M} \to \mathcal{Y}$ be a functionality where $\mathcal{F} = \{f_1, \ldots, f_n\}$. Suppose $n = |\mathcal{F}|$ is poly-bounded. Let $\mathcal{E} = (G_{\mathrm{PKE}}, E_{\mathrm{PKE}}, D_{\mathrm{PKE}})$ be a public key encryption scheme defined over $(\mathcal{Y}, \mathcal{C})$. Define the following functional encryption scheme $\mathcal{FE} = (S, G, E, D)$ for the functionality F:

- $S()$: setup runs as follows

for $i = 1, \ldots, n$ do: $\quad (pk_i, sk_i) \xleftarrow{\text{R}} G_{\text{PKE}}()$

$mpk \leftarrow (pk_1, \ldots, pk_n), \quad msk \leftarrow (sk_1, \ldots, sk_n), \quad$ output (mpk, msk).

- $E(mpk, m) := \Big\{ \boldsymbol{c} \xleftarrow{\text{R}} \Big(E_{\text{PKE}}\big(pk_1, f_1(m)\big), \ldots, E_{\text{PKE}}\big(pk_n, f_n(m)\big) \Big) \in \mathcal{C}^n, \quad \text{output } \boldsymbol{c} \Big\}$.
- $G(msk, f_i) := sk_i, \qquad D(sk_i, \boldsymbol{c}) := D_{\text{PKE}}(sk_i, c_i)$.

(a) Show that if $\mathcal{E}$ is a semantically secure public key encryption scheme, then $\mathcal{FE}$ is a semantically secure functional encryption scheme. In particular, for every $\mathcal{A}$ there exists a $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\mathsf{SSadv}[\mathcal{A}, \mathcal{FE}] \leq n \cdot \mathsf{SSadv}[\mathcal{B}, \mathcal{E}].$$

Use a hybrid argument across n hybrids.

(b) Show that there are functionalities F for which $\mathcal{FE}$ is not robust (as in Definition 15.15).

Discussion: Ciphertext size in this scheme is linear in $n = |\mathcal{F}|$, which is fine when n is small. The main challenge in constructing functional encryption schemes is supporting functionalities where n is super-poly.

15.22 (Inner-product functional encryption). In this exercise we construct a functional encryption scheme for the inner-product functionality discussed in Section 15.7.1.1. Let $\mathbb{G}$ be a group of prime order q generated by $g \in \mathbb{G}$, and let $\mathcal{M} := \{1, \ldots, \ell\}^n \subseteq \mathbb{Z}_q^n$. The scheme works as follows

- $S()$: setup runs as follows

  for $i = 1, \ldots, n$ do: $\quad \alpha_i \xleftarrow{\text{R}} \mathbb{Z}_q, \quad h_i \leftarrow g^{\alpha_i} \in \mathbb{G}$

  $mpk \leftarrow (h_1, \ldots, h_n) \in \mathbb{G}^n, \quad msk \leftarrow (\alpha_1, \ldots, \alpha_n) \in \mathcal{M}, \quad$ output (mpk, msk)

- $G(msk, \boldsymbol{f})$: key generation for $\boldsymbol{f} \in \mathcal{M}$ runs as

  $\beta \leftarrow \sum_{i=1}^{n} \alpha_i \cdot f_i \in \mathbb{Z}_q, \quad$ output $sk_{\boldsymbol{f}} := (\boldsymbol{f},\ \beta) \in \mathcal{M} \times \mathbb{Z}_q$

- $E\big(mpk, \boldsymbol{m}\big)$: encryption using $mpk = (h_1, \ldots, h_n)$ of $\boldsymbol{m} \in \mathcal{M}$ runs as

  $\rho \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g^{\rho}, \quad v_1 \leftarrow g^{m_1} h_1^{\rho}, \quad \ldots, \quad v_n \leftarrow g^{m_n} h_n^{\rho},$

  output $c \leftarrow (u, v_1, \ldots, v_n) \in \mathbb{G}^{n+1}$

- $D(sk_{\boldsymbol{f}},\ c)$: to decrypt $c = (u, v_1, \ldots, v_n)$ using $sk_{\boldsymbol{f}} = (\boldsymbol{f}, \beta)$ do:

  $w \leftarrow \big(\prod_{i=1}^{n} v_i^{f_i}\big)/u^{\beta} \in \mathbb{G}, \quad \hat{m} \leftarrow \text{dlog}_g(w), \quad$ output $\hat{m} \in \mathbb{Z}_q$.

  Here $\text{dlog}_g(w)$ refers to the discrete log of w base g. Since $0 \leq \hat{m} \leq \ell^2 n$, this step takes time $O(\ell n^{1/2})$ using the Pollard lambda algorithm, which is efficient assuming n and ℓ are poly-bounded. The algorithm outputs reject if it is not able to compute discrete log in this time bound.

(a) Show that this scheme correctly implements an inner product functional encryption.

(b) Prove that the scheme is a selectively secure inner product functional encryption, assuming DDH holds in $\mathbb{G}$. Selective security means that the adversary must commit to the challenge messages $\boldsymbol{m}_0, \boldsymbol{m}_1 \in \mathcal{M}$ before seeing mpk. For completeness, we define decryption using the null key sk_ϵ as $D(sk_\epsilon, c) := n$ for all c.

Chapter 16

Attacks on number theoretic assumptions

In the last few chapters we presented several cryptosystems whose security depends on the difficulty of number theoretic and algebraic problems. In this chapter we analyze these assumptions in more detail. We also describe common mistakes that lead to attacks against real-world implementations. These attacks teach us how to correctly implement these systems.

16.1 Analyzing the DL, CDH, and DDH assumptions

In Section 10.5, we introduced the discrete logarithm (DL) and related assumptions, such as the computational Diffie-Hellman (CDH) and decisional Diffie-Hellman (DDH) assumptions. We discussed these assumptions in the context of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. In Section 10.5, we specified that $\mathbb{G}$ was a subgroup of $\mathbb{Z}_p^*$ for some large prime p. Later, in Chapter 15, we introduced the notion of an elliptic curve, and discussed how $\mathbb{G}$ could be a prime-order subgroup of such a curve.

In this section we present discrete log algorithms that work in every finite cyclic group $\mathbb{G}$. We will consider cyclic groups whose order is not necessarily prime. In so doing, we will gain some insight into why we restricted our attention to prime-order groups in the first place.

16.1.1 Square root time algorithms for discrete log

Suppose that $\mathbb{G}$ is a cyclic group of order q generated by $g \in \mathbb{G}$. In this section, q may or may not be prime. Given $u \in \mathbb{G}$, we wish to find an $\alpha \in \mathbb{Z}_q$ such that $g^\alpha = u$. Recall that α is called the discrete log of u base g.

The simplest algorithm to solve this problem is **brute-force search**: try all $\beta = 1, 2, \ldots, q$ until we find the discrete log of u. Here is the algorithm:

```
$v \leftarrow 1 \in \mathbb{G}$
$\beta \leftarrow 0 \in \mathbb{Z}_q$
while $v \neq u$ do
    $v \leftarrow v \cdot g, \quad \beta \leftarrow \beta + 1$
output $\beta$
```

This algorithm is clearly correct, and its worst-case running time requires q multiplications in $\mathbb{G}$.

A faster algorithm. A much faster algorithm is the **baby step/giant step method**. The algorithm uses only $O(q^{1/2})$ group operations in the worst case.

Set $m := \lceil q^{1/2} \rceil$. The algorithm makes use of the fact that we can write the (unknown) discrete log $\alpha \in \mathbb{Z}_q$ as

$$\alpha = \gamma \cdot m + \beta, \qquad \text{for some } 0 \le \beta, \gamma < m. \tag{16.1}$$

Then, to find α, it suffices to find β and γ. Exponentiating both sides of (16.1) gives:

$$u = g^{\alpha} = g^{\gamma m + \beta} = (g^m)^{\gamma} \cdot g^{\beta}. \tag{16.2}$$

We can isolate β and γ to different sides of the equality by re-writing (16.2) as

$$u \cdot (g^{-m})^{\gamma} = g^{\beta}. \tag{16.3}$$

Now we can find β and γ in time $O(q^{1/2})$ using the *meet in the middle* technique from Section 4.2.3.1. The algorithm works in two steps. First, we build a lookup table T that records all m possible values for the right hand side of (16.3), along with the corresponding value of β. We can build T using the following procedure (called "baby steps"):

```
initialize an empty associative array $T : \mathbb{G} \to \mathbb{Z}_q$
$v \leftarrow 1 \in \mathbb{G}$
for $\beta$ from 0 to $m - 1$ do
    $T[v] \leftarrow \beta, \quad v \leftarrow v \cdot g$
```

This algorithm takes $m \approx q^{1/2}$ multiplications in $\mathbb{G}$ to build T.

Second, after building the array T, we compute all the values of the left hand side of (16.3), and check them one by one to see if they are in the table T. If so, then we found a pair (β, γ) that satisfies (16.3). This is done using the following procedure (called the "giant steps"):

```
        $g' \leftarrow g^{-m}, \quad v \leftarrow u, \quad \gamma \leftarrow 0$
$(*)$     while $v \notin \text{Domain}(T)$ do
            $v \leftarrow v \cdot g', \quad \gamma \leftarrow \gamma + 1$       // $v = u \cdot (g')^{\gamma}$
        $\beta \leftarrow T[v]$                        // now $\beta$ satisfies $g^{\beta} = v = u \cdot (g')^{\gamma} = u \cdot g^{-m\gamma}$
        $\alpha' \leftarrow \gamma m + \beta$                    // So $u = g^{\gamma m + \beta} = g^{\alpha'}$
        output $\alpha'$
```

Observe that the loop on line $(*)$ will repeat until a γ is found such that $v = u \cdot (g^{-m})^{\gamma}$ is contained in the table T. Then if $\beta = T[v]$, the pair (β, γ) must satisfy (16.3), from which we recover α as $\alpha \leftarrow \gamma m + \beta$. Because γ is less than $q^{1/2}$, this procedure will terminate after at most $q^{1/2}$ iterations. Overall, the entire baby step/giant step procedure uses about $2q^{1/2}$ group operations in the worse case.

While this algorithm is much faster than brute-force search, it has one drawback: storing the table T requires a lot of memory, enough to store about $q^{1/2}$ group elements. One solution is a "time/space trade-off". By choosing a smaller m, we get a smaller table of size $O(m)$, but the running time is now proportional to q/m, which gets worse the smaller m is.

Remark 16.1. The baby-step/giant-step algorithm can be easily adapted to a situation where we know that $\mathsf{Dlog}_g u$ happens to be in an interval $[A, B]$ of length ℓ. The algorithm takes time and space $O(\ell^{1/2})$. □

Other square root time algorithms. Other (heuristic) algorithms have the same running time as baby step/giant step, but require only *constant* space. Some examples include **Pollard's Rho** method and **Pollard's lambda** method [148, Sec. 11.2.5]. These algorithms can also be modified to take advantage of hardware parallelism or run in a distributed system [154].

All these algorithm use very different techniques, but all of them require $O(\sqrt{q})$ group operations to compute discrete log with probability one half. In Section 16.3 we will show that this is not a coincidence. Every discrete log algorithm that uses the group as a "black box" and succeeds with probability one half, must make at least $\sqrt{q}/3$ group operations.

16.1.2 Discrete log in groups of composite order

In this section we look at algorithms for discrete log in a cyclic group $\mathbb{G}$ of order n, where n is composite. As we will see, discrete log is only as hard as discrete log in the largest prime order subgroup of $\mathbb{G}$. Moreover, the decision Diffie-Hellman (DDH) problem is false whenever n has a small prime factor. These two reasons, among others, explain why throughout the book, we always focus on groups of prime order.

16.1.2.1 Groups of order q^e

Let $\mathbb{G}$ be a cyclic group of order q^e generated by g in $\mathbb{G}$, where $q > 1$ (not necessarily prime) and $e \geq 1$. Observe that for every $f = 0, \ldots, e$

$$g_f := g^{(q^f)} \in \mathbb{G} \quad \text{generates a subgroup of } \mathbb{G} \text{ order } q^{(e-f)}.$$

In particular, $g_0 = g$ and $g_e = 1$. The element g_{e-1} generates a subgroup of order q.

The discrete log problem in $\mathbb{G}$ is as follows: we are given q, e, g, along with an element $u \in \mathbb{G}$, and we want to compute the discrete log of u base g. In particular, if $u = g^\alpha$ for some $0 \leq \alpha < q^e$, then we want to find α.

There is a simple algorithm that allows one to reduce this problem to the problem of computing discrete logarithms in the subgroup of order q generated by g_{e-1}. It is easiest to describe the algorithm recursively. The base case is when $e = 1$, in which case the reduction does nothing because u already lives in a subgroup of order q.

Suppose now that $e > 1$. Let f be some integer in the range $1, \ldots, e-1$. If $u = g^\alpha$, where $0 \leq \alpha < q^e$, then we can write $\alpha = q^f\gamma + \beta$, where β and γ are (unknown) integers with $0 \leq \beta < q^f$ and $0 \leq \gamma < q^{e-f}$. Therefore,

$$u = g^\alpha = g^{(q^f)\gamma+\beta} = g_f^\gamma \cdot g^\beta. \tag{16.4}$$

Since g_f has order $q^{(e-f)}$, raising both sides to the power of q^{e-f} leads to

$$u^{(q^{e-f})} = (g_{e-f})^\beta.$$

Note that g_{e-f} has order q^f. Therefore, we can recursively compute the discrete logarithm of $u^{(q^{e-f})}$ to the base g_{e-f}, and eventually obtain β.

Having obtained β, observe that by (16.4) we know that $u/g^\beta = g_f^\gamma$. Moreover, g_f has order q^{e-f}, and so if we recursively compute the discrete logarithm of u/g^β to the base g_f, we obtain γ. We then find α by computing $\alpha = q^f\gamma + \beta$.

It turns out that to get the best running time, one should choose $f \approx e/2$. Let us put together the above ideas succinctly in a recursive discrete log (RDL) procedure:

Algorithm RDL. On input q, e, g, u as above, do the following:

```
if $e = 1$ then
    return $\log_g u$    // base case: use a different algorithm
else
    set $f \leftarrow \lfloor e/2 \rceil$
    $\beta \leftarrow \mathrm{RDL}\big(q,\ f,\ g_{e-f},\ u^{(q^{e-f})}\big)$    // $0 \le \beta < q^f$
    $\gamma \leftarrow \mathrm{RDL}\big(q,\ e-f,\ g_f,\ u/g^{\beta}\big)$    // $0 \le \gamma < q^{e-f}$
    return $q^f \gamma + \beta$
```

To analyze the running time of this recursive algorithm, note that the body of one invocation (not counting the recursive calls it makes) performs $O(e \log(q))$ group operations. To calculate the total running time, we have to sum up the running times of all the recursive calls plus the running times of all the base cases.

The total number of base case invocations is exactly e, and all the base cases compute discrete logarithms to the base g_{e-1}. If each base case takes T_{base} group operations, then the total time spent on all base cases is $O(e \cdot T_{\text{base}})$.

To analyze the running time of the recursion, let's round e up to the closest power of two. Then, the running time of the recursion is governed by the recurrence relation:

$$T(e) = 2T(e/2) + O(e \log(q)).$$

If $T(1) = 1$ then this evaluates to $T(e) = O(e \log e \log q)$. In our case, $T(1) = T_{\text{base}}$, and hence the total running time of Algorithm RDL is

$$O\big(e \cdot T_{\text{base}} + e \log e \log q\big).$$

group operations. Usually, T_{base} is the dominant term in this expression, and the other quantities are small (poly-bounded). We conclude that computing discrete log in $\mathbb{G}$ is only as hard as computing discrete log in the subgroup of prime order q.

16.1.2.2 Groups of composite order: the Pohlig-Hellman algorithm

Now suppose we have a group $\mathbb{G}$ of order n generated by $g \in \mathbb{G}$. Suppose that

$$n = q_1^{e_1} \cdots q_r^{e_r}$$

is the factorization of n into distinct primes. We assume that this factorization is known. We are given $u \in \mathbb{G}$ and we want to compute the discrete log of u base g.

Let $u = g^{\alpha}$, where $0 \le \alpha < n$. For $i = 1, \ldots, r$, let us define

$$q_i^* := n/q_i^{e_i} \in \mathbb{Z}.$$

Then for each $i = 1, \ldots, r$, we have

$$u^{q_i^*} = \big(g^{q_i^*}\big)^{\alpha}.$$

Note that $g^{q_i^*}$ has order $q_i^{e_i}$ in $\mathbb{G}$. Hence, if α_i is the discrete logarithm of $u^{q_i^*}$ to the base $g^{q_i^*}$, then we have $0 \leq \alpha_i < q_i^{e_i}$ and $\alpha \equiv \alpha_i \pmod{q_i^{e_i}}$. Thus, if we compute the values $\alpha_1, \ldots, \alpha_r \in \mathbb{Z}$, using Algorithm RDL in Section 16.1.2.1, we can obtain α using the Chinese Remainder Theorem (CRT), discussed in Appendix A. We summarize this in the following algorithm, called the Pohlig-Hellman algorithm:

Algorithm PH. On input $n, (q_1, e_1), \ldots, (q_r, e_r), g, u$ as above, do the following:

```
for $i = 1$ to $r$
    $\alpha_i \leftarrow \mathrm{RDL}\big(q_i,\ e_i,\ g^{q_i^*},\ u^{q_i^*}\big)$   //  compute DL in a group of order $q_i^{e_i}$
$\alpha' \leftarrow \mathrm{CRT}\big((\alpha_1, q_1^{e_1}), \ldots, (\alpha_r, q_r^{e_r})\big)$
output $\alpha'$
```

To analyze the running of this algorithm, let $T(w)$ be the number of group operations needed to compute discrete log in a subgroup of $\mathbb{G}$ of prime order w. We assume that $T(w)$ is monotonically increasing in w. If we define $q_{\max} := \max\{q_1, \ldots, q_r\}$, then the running time of Algorithm PH is bounded by

$$\sum_{i=1}^{r} O\big(e_i T(q_i) + e_i \log e_i \log q_i\big) = O\big(T(q_{\max}) \log(n) + \log n \log\log n\big)$$

group operations. We conclude that

> *the difficulty of computing discrete logarithms in a cyclic group of order n is determined by the size of $q_{\max}$, the largest prime dividing n.*

Let us look at some important consequences of this fact. First, suppose $\mathbb{G}$ is a group of order $n = 2^\ell$. Since the largest prime factor of n is 2, the Pohlig-Hellman algorithm will compute discrete log in $\mathbb{G}$ using $O(\log n \cdot \log\log n)$ group operations. Consequently, the DL assumption is false in all such groups. In particular, when p is a prime of the form $p = 2^\ell + 1$ for some integer ℓ, the group $\mathbb{Z}_p^*$ has order $p - 1 = 2^\ell$. The discrete log problem in $\mathbb{Z}_p^*$ for all such p is easy to compute. More generally, discrete log is easy in every group $\mathbb{G}$ whose order is a product of small primes. Again, we conclude that

> *For DL to hold in $\mathbb{G}$, the order of $\mathbb{G}$ must have at least one large prime factor.*

Another way to interpret the Pohlig-Hellman algorithm is as follows. Let $\mathbb{G}$ be a cyclic group of order n. Let q be the largest prime factor of n, and let $\mathbb{G}_q$ be the subgroup of $\mathbb{G}$ of order q. The Pohlig-Hellman algorithm suggests that running the Diffie-Hellman protocol in the entire group $\mathbb{G}$ is wasteful. We might as well run the protocol in the subgroup $\mathbb{G}_q$ where it is more efficient: in $\mathbb{G}_q$ the parties need only choose random exponents in $\{0, \ldots, q-1\}$ as opposed to $\{0, \ldots, n-1\}$ in $\mathbb{G}$. The Pohlig-Hellman algorithm suggests that restricting Diffie-Hellman to the subgroup $\mathbb{G}_q$ does not affect security; discrete log in $\mathbb{G}$ is only marginally harder than discrete log in $\mathbb{G}_q$. This explains why in practice, cryptosystems based on discrete log use a prime order group.

16.1.3 Information leakage in composite order groups

In some cases the Pohlig-Hellman algorithm in Section 16.1.2.2 can be used to compute partial information about the discrete log, and this can lead to attacks. Suppose $\mathbb{G}$ is a cyclic group of order n where $n = n_1 n_2$ and n_1 has only small prime factors. We assume that both n_1 and n_2 are known. Let g be a generator of $\mathbb{G}$.

We are given $(\mathbb{G}, g, n_1, n_2)$ as input, along with an element $u \in \mathbb{G}$ where $u = g^\alpha$. The Pohlig-Hellman algorithm can be used to learn partial information about α. In particular, an adversary can compute $\alpha_1 := (\alpha \bmod n_1)$. To learn α_1, run Pohlig-Hellman to compute the discrete log of u^{n_2} to the base g^{n_2}, and observe that the result is α_1. We say that discrete log in such groups is not very discreet.

As a concrete example, suppose $\mathbb{G}$ is a group of order n where n is even, so that $n = 2n_2$ for some n_2. Then given g and $u := g^\alpha$, one can easily compute $\alpha \bmod 2$. The entire Pohlig-Hellman algorithm in this case degenerates to the following simple observation:

Lemma 16.1. *With $\mathbb{G}$, g, and $u = g^\alpha$ as above,*

$$\alpha \textit{ is even} \quad \Longleftrightarrow \quad u^{n/2} = 1$$

Proof. If α is even then $u^{n/2} = g^{\alpha n/2} = (g^{\alpha/2})^n = 1$. Conversely, if $u^{n/2} = 1$ then $g^{\alpha n/2} = 1$, but then n must divide $\alpha n/2$ and hence α must be even. □

This lemma gives a simple test that tells the adversary whether $\mathsf{Dlog}_g(u)$ is even or not. This test is very damaging when applied to the DDH problem. Suppose the adversary is given an instance of DDH, namely a tuple $(g^\alpha, g^\beta, w) \in \mathbb{G}^3$, and it needs to decide if $w = g^{\alpha\beta}$. First, we observe that the adversary can learn the parity of $\alpha\beta$ using the following lemma.

Lemma 16.2. *With $\mathbb{G}$, g, and $u = g^\alpha,\ v = g^\beta \in \mathbb{G}$ as above*

$$\alpha\beta \in \mathbb{Z}_n \textit{ is even} \quad \Longleftrightarrow \quad (u^{n/2} = 1 \quad \textit{or} \quad v^{n/2} = 1)$$

Proof. By Lemma 16.1 we know that if $(u^{n/2} = 1$ or $v^{n/2} = 1)$ then either α or β is even. But in this case $\alpha \cdot \beta \in \mathbb{Z}_n$ is even. The converse follows in the same way. □

The lemma shows that the adversary can learn one bit of information about the Diffie-Hellman secret $g^{\alpha\beta}$. Now, to decide if $(g^\alpha, g^\beta, w = g^\gamma)$ is a DDH tuple, the adversary does:

step 1: use Lemma 16.1 to compute $b_0 \leftarrow \text{parity}(\gamma) \in \{\textit{even}, \textit{odd}\}$
step 2: use Lemma 16.2 to compute $b_1 \leftarrow \text{parity}(\alpha\beta) \in \{\textit{even}, \textit{odd}\}$
step 3: output accept if $b_0 = b_1$, and output reject otherwise

Observe that the adversary always accepts when $w = g^{\alpha\beta}$, but only accepts with probability 1/2 when w is uniform in $\mathbb{G}$. Consequently, this algorithm has advantage 1/2 in breaking DDH in the group $\mathbb{G}$.

This attack on DDH in groups of even order, and more generally, in groups whose order has a small prime factor (Exercise 16.2), is another reason why we typically use groups of prime order. Note that the group $\mathbb{Z}_p^*$ for some odd prime p has order $p-1$, which is an even number. One should keep in mind that the leakage described above always applies in these groups, and as a result, DDH does not hold in $\mathbb{Z}_p^*$. Instead, we can use a prime order subgroup of $\mathbb{Z}_p^*$, or a prime order elliptic curve group, where DDH is believed to hold.

16.1.4 An attack on static Diffie-Hellman

Static Diffie-Hellman is a situation that comes up in some systems, and can weaken the security of the discrete log problem. To describe the static Diffie-Hellman setup, let $\mathbb{G}$ be a cyclic group of prime order q, and let α be some random secret element in $\mathbb{Z}_q$. The adversary is given access to an oracle that takes an arbitrary $v \in \mathbb{G}$ as input and returns v^α. The adversary's goal is to recover the secret α by querying the oracle. More precisely, the static Diffie-Hellman problem, or SDH, is captured in the following game.

Attack Game 16.1 (Static Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q. For a given adversary $\mathcal{A}$, the game runs as follows.

- The challenger selects $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$
- The adversary makes a sequence of **SDH queries** to the challenger. Each query is of the form $v \in \mathbb{G}$, to which the challenger responds with $w := v^\alpha \in \mathbb{G}$.
- Finally, the adversary outputs some $\hat{\alpha} \in \mathbb{Z}_q$.

We say that $\mathcal{A}$ wins the game if $\hat{\alpha} = \alpha$. □

As usual, we say that **static Diffie-Hellman**, or **SDH**, holds in $\mathbb{G}$ if no efficient adversary $\mathcal{A}$ can win the static Diffie-Hellman game in $\mathbb{G}$ with non-negligible probability.

We had previously encountered a closely related problem in Section 11.6.3 where we constructed an oblivious PRF using the one-more Diffie-Hellman assumption. An adversary $\mathcal{A}$ that can win the static Diffie-Hellman game in $\mathbb{G}$ can trivially also break the one-more Diffie-Hellman assumption in $\mathbb{G}$. This adversary $\mathcal{A}$ can also be used to break the oblivious PRF we constructed in Section 11.6.3. Hence, for the oblivious PRF to be secure, at the very least we need SDH to hold in $\mathbb{G}$.

Static Diffie-Hellman also comes up when analyzing the CCA security of *multiplicative ElGamal*, introduced in Exercise 11.5. This public key encryption scheme has a multiplicative homomorphism, as explained in part (c) of the exercise, and it is therefore not CCA secure: a CCA adversary can break semantic security. However, if SDH were easy in $\mathbb{G}$, then a CCA adversary can go a step further and recover the complete secret key. To see how, let $\alpha \in \mathbb{Z}_q$ be the secret key in an instance of multiplicative ElGamal. We construct a key recovery CCA adversary $\mathcal{B}$. Our $\mathcal{B}$ plays the role of challenger to an SDH adversary $\mathcal{A}$. Whenever $\mathcal{A}$ issues an SDH query for $v \in \mathbb{G}$, our adversary $\mathcal{B}$ does: (i) it issues a decryption query to its challenger for the ciphertext $c := (1/v, 1)$, (ii) it receives back the decryption $w := v^\alpha$, and (iii) it sends w as a response to $\mathcal{A}$'s query. Eventually the SDH adversary $\mathcal{A}$ outputs α, which is the desired secret key.

An attack on static Diffie-Hellman. The oblivious PRF discussed above is an example of a setting where we need SDH to hold in $\mathbb{G}$, otherwise the oblivious PRF construction is insecure. In particular, winning Attack Game 16.1 should be as hard as breaking discrete log in $\mathbb{G}$. However, in some groups, SDH can be a bit easier than computing discrete log. To see why, let's construct an SDH adversary $\mathcal{A}$. The SDH adversary operates as follows:

- *step 1*: let g be a generator of $\mathbb{G}$, and set $g_0 := g$.
- *step 2:* for $i = 1, \ldots, d$: issue an SDH query for $g_{i-1} \in \mathbb{G}$ and get back $g_i := g_{i-1}^\alpha$.

- *step 3:* when step 2 completes the adversary has

$$\boldsymbol{v}_\alpha := \Big(g,\ g^\alpha,\ g^{(\alpha^2)},\ g^{(\alpha^3)},\ \ldots,\ g^{(\alpha^d)}\Big) \in \mathbb{G}^{d+1}.$$

  Now, use the algorithm in Exercise 16.3 to find α.

Exercise 16.3 shows that

- if $q-1$ is divisible by d, then α can be found using only $\tilde{O}\big(\sqrt{q/d}+\sqrt{d}\big)$ group operations;
- if $q+1$ is divisible by d, then α can be found using only $\tilde{O}\big(\sqrt{q/d}+d\big)$ group operations.

For moderate size d, namely $d < q^{1/3}$, these algorithms find α in time $\tilde{O}(\sqrt{q/d})$. This is about a factor of $\sqrt{d}$ faster than the discrete log algorithm from Section 16.1.1, which runs in time $O(\sqrt{q})$. Note that during the attack, the SDH adversary issues d SDH queries. One can show that in a generic group (as in Section 16.3), an SDH adversary that makes d SDH queries and has success probability one half, must take time at least $\Omega(\sqrt{q/d})$. Hence, a $\sqrt{d}$ speed-up is the best possible in a generic group.

So what does this mean? For systems that rely on static Diffie-Hellman for security, one could choose a group $\mathbb{G}$ so that $q-1$ and $q+1$ do not have moderate size factors. For example, consider the Curve25519 from Section 15.3.3, and let q be the order of its large prime order subgroup. Then

$$q-1 = 132 \times p_{107} \times p_{137} \qquad \text{and} \qquad q+1 = 7210 \times p_{59} \times p_{180},$$

where p_n is an n-bit prime. Hence, SDH in Curve25519 is safe from the attack above if the adversary can issue at most 2^{58} SDH queries. In particular, the oblivious PRF in Curve25519 is safe from the attack above as long as the server chooses a fresh PRF key after every 2^{58} PRF evaluation requests.

Curve P256 from Section 15.3.1, also known as secp256r1, is more vulnerable to this attack. Let q' be the order of this curve, then

$$q'-1 = 48 \times 71 \times 131 \times 373 \times 3407 \times 17449 \times 38189 \times p_{27} \times p_{29} \times p_{40} \times p_{91} \qquad \text{and}$$
$$q'+1 = 10 \times 1879 \times p_{27} \times p_{214}.$$

The fact that $q'-1$ has so many small prime factors means that in the oblivious PRF scheme, after 2^{32} PRF evaluation requests the difficulty of computing the secret key drops by about a factor of $\sqrt{2^{32}}$ which is 2^{16}. Curve secp256k1 from Section 15.3.1 is similarly vulnerable. The order of the curve minus one has a number of small factors that multiply to a 24-bit factor. The order of the curve plus one has many small factors.

Remark 16.2. Consider a variation of the SDH game above, called **hash SDH**, where the adversary can issue queries for an arbitrary $v \in \mathbb{G}$, but receives back $H(v, v^\alpha)$ instead of v^α. Here H is some hash function $H : \mathbb{G}^2 \to \mathcal{K}$. This small variation prevents the attack described above. In fact, one can show that when H is modeled as a random oracle, this hash SDH problem in a group $\mathbb{G}$ is as difficult as solving the interactive Diffie-Hellman (ICDH) problem in $\mathbb{G}$ (see Exercise 12.31). □

16.1.5 The relation between DL, CDH, and DDH

The Diffie-Hellman protocol relies on the difficulty of solving the CDH problem, which is quite different from discrete log. How hard is CDH?

Let $\mathbb{G}$ be a finite cyclic group with generator $g \in \mathbb{G}$. The best known general purpose algorithm for solving CDH in $\mathbb{G}$ is the following:

input: $u := g^{\alpha}, \quad v := g^{\beta} \in \mathbb{G}$; output: $g^{\alpha\beta} \in \mathbb{G}$

step 1: compute the discrete log of u to the base g to obtain α,
step 2: output $v^{\alpha} = g^{\alpha\beta}$

This suggests that if discrete log in $\mathbb{G}$ is difficult then so is CDH. However, perhaps there is a shortcut, namely a faster algorithm for CDH that computes the answer $g^{\alpha\beta}$ without first computing the discrete log of u or v. Can we rule out such a shortcut?

As a first step in exploring the relation between CDH and discrete log we could look for a finite cyclic group $\mathbb{G}$ of prime order q where CDH is easy, but discrete log is difficult. Clearly the Diffie-Hellman protocol will be insecure in such a group. However, as it turns out, such a group would be quite useful. It would mean that Alice could choose a random $\alpha \xleftarrow{R} \mathbb{Z}_q$ and hand $u := g^{\alpha}$ over to Bob. Bob would not learn α, however for any polynomial $f \in \mathbb{Z}_q[X]$, Bob could compute $v := g^{f(\alpha)}$ just given u. Computing v would be done by a repeated application of the algorithm to compute the CDH problem. In effect, Bob is evaluating $f(\alpha)$ without explicit knowledge of α. This property is related to the concept of *fully homomorphic encryption*, and has quite a few applications.

Currently, no one knows how to construct a finite cyclic group where CDH is easy, but discrete log is hard. We encourage the reader to try to construct such a group. However, it is quite likely that no such group exists. To prove that, we need to show that in every finite cyclic group $\mathbb{G}$ where CDH is easy, discrete log is also easy. That is, we need to construct a discrete log algorithm base g from a subroutine to compute CDH base g.

Although this is still an open problem, there is a partial answer. A result of Maurer and Wolf [110] shows that there is a short advice string S, that only depends on q, with the following property: once S is known, there is an efficient algorithm to compute discrete log base g for any $u \in \mathbb{G}$, using a CDH subroutine base g. Although the string S is just two elements in $\mathbb{Z}_q$, there is unfortunately no known efficient algorithm to find it. Nevertheless, for several standard cryptographic groups such as P256 and Curve25519 discussed in Section 15.3, this string S was computed explicitly. Hence, for these groups we now know that an efficient algorithm to compute CDH gives an efficient algorithm to compute discrete log. In other words, in these groups, there is no algorithmic shortcut to compute CDH without also computing discrete log.

Discrete log and DDH. The situation is quite different with respect to the DDH problem. In Section 15.4 we constructed a group where DDH is easy, but discrete log is believed to be hard. Recall that the algebraic tool that made this possible is called a *pairing*. The algorithm to decide DDH is described in Section 15.4. This suggests that DDH can be an easier problem than discrete log. Nevertheless, we have many candidate groups where DDH is believed to be hard.

16.2 Discrete log in $\mathbb{Z}_p^*$: the general number field sieve

The special structure of the group $\mathbb{Z}_p^*$ gives rise to several discrete log algorithms that are much faster than the generic techniques discussed in Section 16.1.1. The fastest discrete log algorithm in

$\mathbb{Z}_p^*$, called **the general number field sieve** or **GNFS**, runs in conjectured time:

$$\exp\Big(\big(\alpha + o(1)\big)\ (\ln p)^{1/3}\ (\ln\ln p)^{2/3}\Big) \tag{16.5}$$

where $\alpha := (64/9)^{1/3} \approx 1.92$. The dominant term in this expression is $\exp\big((\ln p)^{1/3}\big)$. We will describe the algorithm in a bit more detail in Section 16.2.2 below.

So what does this mean? To understand (16.5) let us simplify things by only focusing on the dominant term. The GNFS running time is dominated by the expression $\exp(\sqrt[3]{\ln p})$. If p is an n-bit prime this function behaves as $\exp(\sqrt[3]{n})$, ignoring constants. Now, suppose that for a fixed n GNFS computes discrete log in $\mathbb{Z}_p^*$ in time T. If we double the size of our prime and use a $2n$-bit prime, the GNFS running time will grow to about

$$\exp(\sqrt[3]{2n}) \approx \exp(1.26\sqrt[3]{n}) = T^{1.26}$$

This is very different from what we know for block ciphers — doubling the size of a block cipher key *squares* the amount of work to do an exhaustive key search attack. In contrast, doubling the security parameter for discrete log in $\mathbb{Z}_p^*$ increases security by far less. As a result, the GNFS forces us to use relatively large primes when using $\mathbb{Z}_p^*$.

So how large should the primes be? Concretely, one wants security of the public-key system to be comparable to the security of the symmetric-key system being used. Hence, what size prime to use depends on what size symmetric key is being used. The national institute of standards (NIST) [127] recommends certain size primes. A more conservative study by Lenstra [103] recommends slightly larger primes. These recommendations are summarized in the following table:

Symmetric key length (bits)	Size of prime (bits) [NIST]	Size of prime (bits) [Lenstra]
80	1024	1329
128	3072	4440
256	15360	26268

Note that for a 256-bit AES key one has to use an extremely large prime, over 15000 bits. Computing with such large primes can be very slow — about a hundred times slower than computing with a 1024-bit prime. As a result, it is likely that the group $\mathbb{Z}_p^*$ will never be used when a 256-bit AES key is needed. This is a direct result of the GNFS.

Another group where discrete log is believed to be hard is the group of points of an elliptic curve over a prime finite field, discussed in Chapter 15. Discrete log in this group appears to be much harder than in $\mathbb{Z}_p^*$. In particular, the best known algorithm takes time $O(\sqrt{q})$ where q is the order of the group. Consequently, these groups scale much better with the security parameter. For example, when using a 256-bit AES key one need only use a 512-bit elliptic curve group.

16.2.1 Discrete log records in $\mathbb{Z}_p^*$

Let us look at some concrete records for computing discrete log in $\mathbb{Z}_p^*$. To set up a discrete log challenge, one has to choose the prime p, the base g, and the challenge y. The goal is to find an integer α such that $y = g^\alpha$ in $\mathbb{Z}_p$. Typically, one sets $g = 2$ and chooses p and y in some random fashion.

The following table shows the progress in computing discrete log over the years. The entries refer to a discrete log computation in $\mathbb{Z}_p^*$ for a generic prime p. All these records were obtained using variants of the GNFS discrete log algorithm.

Year	Bits, $\lceil \log_2 p \rceil$	Team members
2014	**596**	Bouvier, Gaudry, Imbert, Jeljeli, Thomé
2016	**768**	Kleinjung, Diem, Lenstra, Priplata, and Stahlke
2019	**795**	Boudot, Gaudry, Guillevic, Heninger, Thomé, Zimmermann

16.2.2 A preprocessing attack on discrete log in $\mathbb{Z}_p^*$

The GNFS discrete log algorithm in $\mathbb{Z}_p^*$ has an important feature that is often overlooked: the bulk of the work to compute the discrete log of an element $u \in \mathbb{Z}_p^*$ depends on the prime p, but is independent of the specific element u. This means that one can pre-process the prime p by carrying out the bulk of the discrete log work ahead of time, and then quickly compute the discrete log of challenge values in $\mathbb{Z}_p^*$.

For example, an experiment carried out in 2017 showed that for a 512-bit prime p, the pre-processing phase runs in about two days on sufficiently many machines. However, once the one-time pre-processing phase is complete, computing the discrete log of a given $u \in \mathbb{Z}_p^*$ can be done in about a minute [2]. For a larger prime, say for a 1024-bits prime p, the estimate is that once the one-time pre-processing phase is complete, computing discrete log for a given $u \in \mathbb{Z}_p^*$ can be done in about 30 days on a single core. These times can be greatly reduced by using multiple cores. Of course, for a 1024-bits prime, the one-time pre-processing phase requires considerable effort.

This aspect of the GNFS was used in the **logjam attack** [2] to exploit the fact that some cryptographic standards that use Diffie-Hellman key exchange fix the prime p used in the standard. A prominent example is the set of Oakley groups, which is a list of "safe" primes of length 768 (Oakley Group 1), 1024 (Oakley Group 2), and 1536 (Oakley Group 5). These groups were published in 1998 and have been used for many standards based on Diffie-Hellman, including IPsec-IKE, SSH, Tor, and Off-the-Record Messaging (OTR). A nation state could carry out the expensive pre-computation for the Oakley primes, and then quickly break any key exchange it wants.

To avoid these pre-processing attacks, deployed systems that use $\mathbb{Z}_p^*$ should use a much larger prime p, at least 3072 bits long, as suggested by the table in the previous section. Alternatively, one could move to an elliptic curve group, where there is no known practical pre-processing attack.

16.3 A lower bound on discrete log in generic prime order groups

Because so much of cryptography relies on the difficulty of discrete log, we would like to prove a lower bound on the time to compute discrete log in some specific family of groups. Presently, proving such a lower bound appears to be far beyond our reach. However, if we focus our attention only on "generic" discrete log algorithms, namely algorithms that treat group elements as opaque strings and use the group operation as a black box, then it is possible to prove a lower bound. In this section we prove such a lower bound. The technique is quite general and can be used to prove a lower bound for many computational tasks in finite cyclic groups. We will see some examples in Exercise 16.5. Note that these lower bounds only apply to classical (i.e., non quantum) algorithms (see Section 16.5).

To explain what is a generic algorithm we first need to define a generic group.

Generic groups. A **generic group** is an idealized model where group elements are represented as opaque binary strings. In formulating an attack game in the generic group model, both the challenger and the adversary access the group through an "oracle". To formally model a generic cyclic group of order q, we fix a set $\mathcal{S}$ of bit strings, where $|\mathcal{S}| \geq q$. The oracle chooses a random injective function $L : \mathbb{Z}_q \to \mathcal{S}$, which we call a *labeling function*. The oracle responds to queries of the following type:

Labeling queries: The oracle receives $\beta \in \mathbb{Z}_q$ and responds with $L(\beta)$.

Group operations: The oracle receives $(\ell_1, \ell_2, \kappa_1, \kappa_2) \in \mathcal{S} \times \mathcal{S} \times \mathbb{Z}_q \times \mathbb{Z}_q$. If there exist $\beta_1, \beta_2 \in \mathbb{Z}_q$ such that $L(\beta_1) = \ell_1$ and $L(\beta_2) = \ell_2$, the oracle responds with $L(\kappa_1\beta_1 + \kappa_2\beta_2)$; otherwise, the oracle responds with reject.

We think of $L(\beta)$ as corresponding to g^β, where g is a fixed generator for the group. Therefore, if ℓ_1, ℓ_2 correspond to g^{β_1}, g^{β_2}, the group operation query computes

$$g^{\kappa_1\beta_1+\kappa_2\beta_2} = (g^{\beta_1})^{\kappa_1} \cdot (g^{\beta_2})^{\kappa_2}.$$

The baby-step/giant-step algorithm from Section 16.1.1 is an example of a discrete log algorithm in a generic group. The algorithm takes as input $g \leftarrow L(1)$ and $u \leftarrow L(\alpha)$, for some $\alpha \in \mathbb{Z}_q$. It makes a sequence of calls to the group operation oracle, and outputs α.

The following theorem gives a lower bound on the time to compute discrete log in a generic group.

Theorem 16.3. *Let $\mathcal{A}$ be an adversary that attacks the discrete logarithm problem for a generic group of prime order q. The challenger chooses $\alpha \in \mathbb{Z}_q$ at random, obtains $L(1)$ and $L(\alpha)$ from the group oracle via a labeling query, and gives $L(1)$ and $L(\alpha)$ to $\mathcal{A}$. Suppose $\mathcal{A}$ then makes a series of at most T queries to the group oracle and then outputs $\hat{\alpha} \in \mathbb{Z}_q$. Then*

$$\Pr[\alpha = \hat{\alpha}] \leq \frac{(3T+2)^2/2+1}{q}. \tag{16.6}$$

Proof. We begin by describing the discrete log attack game, which we call Game 0, in more detail. We describe the challenger's logic as follows:

```
Initialization:
    choose a random injective function $L : \mathbb{Z}_q \to \mathcal{S}$ in Funs$[\mathbb{Z}_q, \mathcal{S}]$
    $\alpha \xleftarrow{R} \mathbb{Z}_q$
    invoke (label, 1) to obtain $g \in \mathcal{S}$        // $g$ is the label of $1 \in \mathbb{Z}_q$
    invoke (label, $\alpha$) to obtain $u \in \mathcal{S}$        // $u$ the label of $\alpha \in \mathbb{Z}_q$
    return $(g, u)$                                  // send $(g, u)$ to the adversary

To process an oracle query (label, $\beta$):        // return the label of $\beta \in \mathbb{Z}_q$
    return $L(\beta)$

To process an oracle query (group-op, $\ell_1, \ell_2, \kappa_1, \kappa_2$):
    if $\ell_1 = L(\beta_1)$ and $\ell_2 = L(\beta_2)$ for some $\beta_1, \beta_2$
        then invoke (label, $\kappa_1\beta_1 + \kappa_2\beta_2$) and return the result
        else  return reject
```

At the beginning of the game, a random injective function $L : \mathbb{Z}_q \to \mathcal{S}$ is chosen, and then the initialization step is executed, where result (g, u) is given to the adversary. After this, the adversary makes a series of `label` and `group-op` queries, and then outputs $\hat{\alpha} \in \mathbb{Z}_q$. We let W_0 be the event that the adversary wins the game, that is, $\alpha = \hat{\alpha}$.

Game 1. We modify Game 0 so that the challenger performs a "lazy" simulation of the random function L. To this end, the challenger maintains an associative array $Map : \mathbb{Z}_q \to \mathcal{S}$, initially empty, which partially represents L. One messy detail is that we have to deal with group operation queries $(\texttt{group-op}, \ell_1, \ell_2, \kappa_1, \kappa_2)$ where either ℓ_1 or ℓ_2 are not Range(Map), and so we have to probabilistically decide if they belong to Range(L). If we decide that ℓ_j does not belong to Range(L), we have to remember that decision. To keep track of these decisions, the challenger also maintains a set $\mathcal{S}' \subseteq \mathcal{S}$, initially empty, which contains the elements of $\mathcal{S}$ that are known to not be in Range(L).

We describe the challenger's logic as follows:

```
Initialization:
          $\alpha \xleftarrow{R} \mathbb{Z}_q$
          invoke $(\texttt{label}, 1)$ to obtain $g \in \mathcal{S}$
    (1)   invoke $(\texttt{label}, \alpha)$ to obtain $u \in \mathcal{S}$
          return $(g, u)$

To process a group oracle query $(\texttt{label}, \beta)$:
          if $\beta \notin$ Domain($Map$) then
              $\ell \xleftarrow{R} \mathcal{S} \setminus (\text{Range}(Map) \cup \mathcal{S}')$
              $Map[\beta] \leftarrow \ell$
          return $Map[\beta]$

To process a group oracle query $(\texttt{group-op}, \ell_1, \ell_2, \kappa_1, \kappa_2)$:
          for $j = 1, 2$ do
              if $\ell_j \notin \text{Range}(Map) \cup \mathcal{S}'$ then
                  // probabilistically decide if $\ell_j$ is in Range($L$)
                  flip a biased coin that is "heads" with probability
    (2)               $\big(q - |\text{Range}(Map)|\big)/\big(|\mathcal{S}| - |\text{Range}(Map)| - |\mathcal{S}'|\big) \in [0, 1]$
                  if the coin is "heads"        // make $\ell_j$ be in Range($L$)
                      then $\beta_j \xleftarrow{R} \mathbb{Z}_q$
                          while $\beta_j \in$ Domain($Map$) do $\beta_j \xleftarrow{R} \mathbb{Z}_q$
                          $Map[\beta_j] \leftarrow \ell_j$
                      else  add $\ell_j$ to $\mathcal{S}'$        // mark $\ell_j$ as not in Range($L$)
          if $\ell_1 = Map[\beta_1]$ and $\ell_2 = Map[\beta_2]$ for some $\beta_1, \beta_2$
              then invoke $(\texttt{label}, \kappa_1\beta_1 + \kappa_2\beta_2)$ and return the result
              else  return reject
```

To understand the ratio in line (2), consider a randomly chosen injective function $L : \mathbb{Z}_q \to \mathcal{S}$. At any point in time, we have the following partial information about L:

(i) $L(\beta) = Map[\beta]$ for all $\beta \in$ Domain(Map), and

(ii) Range(L) $\cap\, \mathcal{S}' = \emptyset$.

The numerator in line (2) is the size of Range$(L) \setminus$ Range(Map). The denominator in line (2) is the size of $\mathcal{S} \setminus (\text{Range}(\mathit{Map}) \cup \mathcal{S}')$, which contains Range$(L) \setminus$ Range(Map). If follows that for $\ell_j \in \mathcal{S} \setminus (\text{Range}(\mathit{Map}) \cup \mathcal{S}')$, the conditional probability that $\ell_j \in$ Range(L) given the partial information (i) and (ii) is precisely the ratio in line (2).

From this, it is easy to verify that the challenger in Game 1 perfectly simulates the challenger in Game 0, and in particular,

$$\Pr[W_0] = \Pr[W_1]. \tag{16.7}$$

Game 2. We modify Game 1 so that the challenger now performs a "symbolic" simulation of the function L. Specifically, we introduce an indeterminate X, and the associative array is now of the form $\mathit{Map} : \mathbb{Z}_q[X] \to \mathcal{S}$. The domain of *Map* will consist of linear polynomials of the form $P(X) = \gamma X + \delta$ for some $\gamma, \delta \in \mathbb{Z}_q$. The *only* difference between Games 1 and 2 is that we replace line (1) above by

(1′) invoke $(\texttt{label}, X)$ to obtain $u \in \mathcal{S}$

Note that while the challenger invokes `label` with a non-constant polynomial in the initialization step and indirectly in the implementation of `group-op`, the adversary is only allowed to make direct `label` queries with constant polynomials (as in Game 1). We let W_2 be the event that $\alpha = \hat{\alpha}$ in Game 2.

Let us define the event Z in Game 2 as the event that for two distinct polynomials $P_1(X), P_2(X) \in$ Domain(Map), we have $P_1(\alpha) = P_2(\alpha)$. It is straightforward to see that Games 1 and 2 proceed identically unless the event Z occurs. In Game 2, the value of α is independent of the adversary's view and of all of the polynomials in Domain(Map). It follows that for any two distinct polynomials $P_1(X), P_2(X) \in$ Domain(Map), we have $P_1(\alpha) = P_2(\alpha)$ with probability at most $1/q$. Moreover, the size of Domain(Map) is at most $3T + 2$ (each `group-op` query can add up to 3 elements to Domain(Map)). It follows that $\Pr[Z] \leq (3T+2)^2/2q$, and hence (by the Difference Lemma) we have

$$\bigl|\Pr[W_2] - \Pr[W_1]\bigr| \leq (3T + 2)^2/2q. \tag{16.8}$$

Finally, we observe that

$$\Pr[W_2] = 1/q, \tag{16.9}$$

since α is independent of the adversary's view in Game 2. Combining (16.7)–(16.9), we obtain

$$\Pr[W_0] \leq (3T + 2)^2/2q + 1/q,$$

which proves the theorem. □

The theorem shows that an adversary that makes at most T group oracle queries (even an otherwise computationally unbounded adversary) computes the discrete log with probability at most $O(T^2/q)$. If we want a generic discrete log algorithm that succeeds with probability one half, then we must have $T \geq \Omega(\sqrt{q})$ — concretely, T must be greater than about $\sqrt{q}/3$. This suggests that the $O(\sqrt{q})$-time algorithms discussed in the beginning of the chapter are the best possible, if one does not use any special properties of the underlying group.

One should be careful not to read too much into this lower bound. The bound says nothing about the difficulty of discrete log in any *specific* group, where an algorithm might take advantage of specific properties of the group. A good example is a prime order subgroup of $\mathbb{Z}_p^*$, where one can take advantage of the structure of the group to give a much faster discrete log algorithm, as discussed in Section 16.2.

Remark 16.3. One exception to the generic $\Omega(\sqrt{q})$ lower bound is a discrete log attack that uses a one-time pre-processing phase to analyze the generic group $\mathbb{G}$ and produce an "advice" string of size $O(q^{1/3})$. Using this advice string, there is an algorithm that computes discrete log in time $O(q^{1/3})$ for every $u \in \mathbb{G}$. This algorithm is presented in Exercise 18.19. Computing the advice string takes time $O(q^{2/3})$, but this is a one time cost that then enables us to compute discrete log in time $O(q^{1/3})$ in the worst case, for all inputs. For $q \approx 2^{256}$, a pre-processing computation that takes time about $q^{2/3}$ is beyond the present capabilities of humanity. There is also a lower bound that shows that in a generic group, a discrete log algorithm with advice of size $O(q^{1/3})$ that succeeds with probability one half, must take time at least $\Omega(q^{1/3})$. Hence, in a generic group, the algorithm with advice in Exercise 18.19 is the best possible. □

16.4 Analyzing the factoring and RSA assumptions

We now turn to the security of the RSA problem and the associated problem of factoring integers. We will also present several elegant real-world attacks that apply when RSA is implemented incorrectly. As we will see, RSA is a bit fragile. Seemingly benign deviations from the specified algorithms can lead to devastating attacks.

Let $pk := (n, e)$ be an RSA public key, and $sk := (n, d)$ be the corresponding private key. Here $n = p \cdot q$ is the product of two random ℓ-bit primes p and q, and e is relatively prime to $(p-1)(q-1)$. Recall that the RSA trapdoor function is defined as $F(pk, x) := x^e \in \mathbb{Z}_n$ and its inverse is $I(sk, y) := y^{1/e} = y^d \in \mathbb{Z}_n$. We say that an algorithm $\mathcal{A}$ solves the RSA problem if it can compute $y^{1/e} \in \mathbb{Z}_n$ given only (n, e) and $y \in \mathbb{Z}_n$ as input. More precisely, $\mathcal{A}$ solves the RSA problem if $\mathrm{RSA}\mathsf{adv}[\mathcal{A}, \ell, e]$ is non-negligible, as in Definition 10.5. The RSA assumption, which is the basis for RSA encryption and signatures, says that there is no efficient algorithm for the RSA problem.

How quickly can we solve the RSA problem? The best known algorithm that takes (n, e, y) as input and computes $y^{1/e} \in \mathbb{Z}_n$ works as follows:

step 1: factor n to obtain p and q,
step 2: compute $d \leftarrow e^{-1} \bmod (p-1)(q-1)$,
step 3: output $y^d \in \mathbb{Z}_n$

Steps 2 and 3 are easy. The only difficult step is step 1, factoring n. Therefore, for the remainder of this section we will study the difficulty of factoring integers.

A natural question is whether there is a shortcut that lets us solve the RSA problem without factoring n. While there is likely no shortcut, and the three step algorithm above is the best possible, proving that the RSA problem is as hard as factoring remains an open problem. This is discussed further in Exercise 16.9.

16.4.1 Factoring algorithms

In this section we look at the difficulty for factoring the large integers used in the RSA trapdoor function. Recall that an RSA integer (also called an RSA modulus) is a product of two random equal size primes.

The current fastest algorithm for factoring RSA integers is called the **general number field sieve**, or **GNFS**. The asymptotic running time for factoring an integer n using GNFS is conjectured to be

$$\exp\Big(\big(\alpha + o(1)\big)\ (\ln n)^{1/3}\ (\ln\ln n)^{2/3}\Big) \tag{16.10}$$

where $\alpha := (64/9)^{1/3} \approx 1.92$. The dominant term in this expression is $\exp\big((\ln n)^{1/3}\big)$.

The algorithm works by constructing two "random" integers x and y such that $x^2 \equiv y^2 \bmod n$. Then $(x-y)(x+y) = x^2 - y^2$ is a multiple of n, and with a reasonable probability over the choice of x, y, either $\gcd(x-y, n)$ or $\gcd(x+y, n)$ is a non-trivial factor of n.

To test our real-world ability to factor RSA integers, there is a need for a list of challenge numbers to factor. Such a list, called the **RSA challenge numbers**, was produced by RSA Laboratories in 1991. Every number on the list is a product of two equal size primes. The list contains 54 numbers of varying sizes; the smallest is 330 bits and the largest is 2048 bits. Every number has a designation that identifies its size. For example, RSA250 refers to the challenge number whose size is 250 decimal digits, while RSA768 refers to the challenge number whose size is 768 bits.

The following table shows the progress in factoring the RSA challenge numbers over the years. All these records were obtained using variants of the GNFS factoring algorithm.

Year	Designation	Bits, $\lceil \log_2 n \rceil$	Team members
2005	RSA200	**663**	Bahr, Boehm, Franke, Kleinjung
2010	RSA768	**768**	Kleinjung, Aoki, et al.
2019	RSA240	**795**	Boudot, Gaudry, Guillevic, Heninger, Thomé, Zimmermann
2020	RSA250	**829**	Boudot, Gaudry, Guillevic, Heninger, Thomé, Zimmermann

Every one of these factoring projects took several months to complete, running on thousands of machines working in parallel.

The ECM factoring algorithm. Another important factoring algorithm is called the **elliptic curve method** or **ECM**. The interesting fact about ECM is that its running time depends primarily on the size of the *smallest* prime factor of the given integer. The ECM is often used when one needs to factor a large integer that has a relatively small prime factor. Its asymptotic running time for factoring an integer n whose smallest prime factor is p is conjectured to be

$$\exp\Big(c\sqrt{\ln p \cdot \ln\ln p}\Big) \cdot M(n)$$

for some fixed constant $c > 0$, where $M(n)$ is the time to multiply two elements in $\mathbb{Z}_n$. The ECM has been used to find a 274-bit prime factor of a number that is almost 1000 bits long.

The dominant term in the running time of ECM is $\exp(\sqrt{\ln p})$. Suppose one tries to use ECM to factor an RSA integer n, where the size of the smallest prime factor is about half the size of n. Then, ignoring constants, the running time of ECM would be dominated by the expression $\exp(\sqrt{\ln n})$, which is much worse than the running time for GNFS, which is dominated by $\exp(\sqrt[3]{\ln n})$. This explains why all the factoring records were obtained using GNFS. Nevertheless, the ECM can be useful in factoring an *asymmetric RSA integer* $n = p \cdot q$, where p is chosen to be much smaller than q.

16.4.2 Attacks on RSA resulting from poor key generation

Recall that the RSA key generation procedure from Section 10.3 generates two random *independent* ℓ-bit primes p and q and then sets $n \leftarrow p \cdot q$. While this may seem relatively straightforward, there are many real-world difficulties in implementing this correctly. We give three examples.

Let us drill a little deeper into how a cryptographic library might implement this procedure. In particular, where do the random bits needed to generate p and q come from? Typically, the software uses a random number generator (RNG) of the type discussed in Section 3.10. The generator is initialized with a random seed, and then the system periodically adds more randomness to the generator.

Now, consider the following natural RSA modulus generation procedure:

```
rng.initialize(seed)                  //  initialize RNG with a random seed
$p \leftarrow$ generate_random_prime(rng)      //  generate $p$ using bits from the RNG
rng.add_randomness(bits)              //  add more randomness to the RNG
$q \leftarrow$ generate_random_prime(rng)      //  generate $q$ using bits from the RNG
$n \leftarrow p \cdot q$
```

The random values *seed* and *bits* are generated by random events on the system, such as the time at which interrupts occur. Some systems include a true random number generator that contributes additional entropy to these values.

The question is what happens on a cheap electronic device, like a home WiFi router. Suppose the appliance needs to generate a cryptographic key shortly after it is powered up for the first time at the customer's home. The problem is that on first power up the device has no entropy. Then

- *seed* is completely determined by the initial configuration of the device, but
- *bits* is sampled after the first prime p is generated, and by then enough entropy may be generated by random events on the device so that *bits* is properly random.

We end up with an RSA modulus $n = p \cdot q$ where p is not random, but q is properly random. Is this a secure RSA key that the device can use?

Suppose that a batch of b devices all have the same initial configuration. Each device generates one RSA key pair. We end up with b RSA public keys with moduli $n_1, \ldots, n_b$ that all have the same prime factor p, but different q. An attacker who observes these public moduli can factor all of them: simply compute $\gcd(n_i, n_j) = p$ for some $1 \leq i \neq j \leq b$. Once these moduli are factored, the attacker can compute the private keys of all b devices.

This experiment was carried out in 2012 [86, 104]. Researchers downloaded several million RSA public keys used on the Web. They computed the gcd of all pairs, and to their surprise, this factored about 0.2% of the moduli, or about 10,000 keys.

The lesson is that cryptographic key generation must always be done with proper randomness. In the case of RSA, "half randomness" as in the example above, leads to RSA keys that are easily broken.

16.4.2.1 Attacks caused by dependent primes

Recall that the prime factors p and q of an RSA modulus are generated as random independent ℓ-bit primes. Let's see what goes wrong when they are not independent. Consider the following broken RSA modulus generation procedure that tries to minimize the number of random bits needed:

```
RSAModGen($\ell$) :=
    choose a random $\ell$-bit integer $r$
    $p \leftarrow$ NextPrime($r$)            // find the smallest prime greater than $r$
    choose a random integer $0 < s \leq 2^{(\ell/2)-1}$
    $q \leftarrow$ NextPrime($p+s$)          // find the smallest prime greater than $p+s$
    output $n \leftarrow p \cdot q$
```

This procedure generates a prime p that is close to being uniformly sampled from the set of ℓ-bit primes. The same holds for q. However the primes p and q are not independent; they are close to one another. We can assume that $p + s \leq q < p + 2s$ and therefore

$$\big|p - q\big| \leq 2s \leq 2^{\ell/2} < 2n^{1/4}. \tag{16.11}$$

An integer $n = p \cdot q$ that satisfies (16.11) can be factored quite easily. To see how, we first need a simple lemma.

Lemma 16.4. *Let $n = p \cdot q$ and $A = (p+q)/2$. If $\big|p - q\big| < 2n^{1/4}$ then $\big|A - \sqrt{n}\big| < 1/2$.*

The proof is a simple calculation and we leave it as an exercise. Now, to factor an integer $n = p \cdot q$ that satisfies (16.11) do:

- *step 1:* Since p and q are odd primes, we know that $p + q$ is an even integer, and therefore $A = (p+q)/2$ is an integer. Moreover, by the arithmetic mean-geometric mean (AMGM) inequality we know that $A \geq \sqrt{n}$. Then, Lemma 16.4 shows that

  $$\big|p - q\big| < 2n^{1/4} \qquad \text{implies that} \qquad A = \lceil \sqrt{n}\, \rceil.$$

  This lets us compute A efficiently from n.

- *step 2:* Once A is computed, we obtain a system of two equations in two variables, p and q, over the reals:

  $$\begin{cases} p + q &= 2A \\ p \cdot q &= n \end{cases}$$

  This system is easily solved over the reals and the solution gives the factors p and q.

This factoring method can be generalized to factor integers $n = p \cdot q$ even when p and q are not close; all we need is a known dependence between p and q. For example, in Exercise 16.14 we generalize the algorithm above to the case $|3p - 2q| < n^{1/4}$. Here p and q can be quite far apart, and yet $n = p \cdot q$ can be efficiently factored. In fact, this can be further generalized to other dependencies of the form $\big|ap + bq\big| < n^{1/4}$, where $a, b \in \mathbb{R}$ are known values.

The lesson from these attacks is that the primes that make up the RSA modulus must be generated independently of one another. A known relation between them could lead to a factoring algorithm. These mistakes occasionally happen in real-world cryptography libraries (for example, see CVE-2022-26320).

16.4.2.2 Attacks caused by non-uniform primes

Our third example of poor RSA key generation is a result of an ill-advised optimization. One way to generate a random ℓ-bit prime is as follows: (i) generate a random ℓ-bit integer t, (ii) run a *primality testing* algorithm to check if t is a prime, and (iii) repeat steps (i) and (ii) until a prime is found. One can show that this procedure outputs a random prime in reasonable time.

Clearly, if a random ℓ-bit candidate t happens to be even, or divisible by three, then there is no reason to run the primality test on t since we already know that t is not a prime. More generally, we can reduce the number of primality tests in the procedure above by first ensuring that the candidate t is not divisible by a small prime. This step is called *sieving* because it sieves out all the candidates that are a multiple of 2, 3, 5, etc. So, to generate a random ℓ-bit prime one repeats the following two steps until a prime is found:

- *sieve:* generate an ℓ-bit candidate prime t such that t has no factors less than some bound B (e.g., $B = 1000$),

- *test:* run a primality test on t.

While the primality test is the time consuming step, some implementations further optimize the sieving step.

Here is one way to speed up sieving that is quite dangerous. Let S be the product of the first s primes, namely $S := 2 \cdot 3 \cdot 5 \cdot 7 \cdots p_s \in \mathbb{Z}$. To generate an ℓ-bit candidate prime t do:

$$\text{choose a random } (\ell - \lceil \log_2 S \rceil)\text{-bit integer } k, \text{ and set } t \leftarrow k \cdot S + 1.$$

This way t is an ℓ-bit integer (or possibly $(\ell - 1)$ bits), and this t is guaranteed to not be divisible by the first s primes. Now, repeatedly choose a random k until the resulting t passes the primality test. Finally, output the RSA modulus n as a product of two such primes p and q.

For example, let S be the product of the first 80 primes. This S is a 552-bit number, and therefore, when generating a 1024-bit prime, one would choose k as a random 472-bit integer. This ensures that $t := kS + 1$ is a 1024-bit integer (or possibly a 1023 bits). Hence, there is ample entropy in the primes being generated thanks to the 472-bit random k.

The problem is that the p and q generated this way are not random ℓ-bit primes. In particular $p \equiv q \equiv 1 \bmod S$ for some known S. As it turns out, if $S > \Omega(n^{1/4})$ then such n can be factored efficiently. The factoring algorithm, due to Don Coppersmith [45, 90], takes as input n and S, and outputs the factors p and q of n. To make matters worse, an attacker can easily identify all vulnerable RSA moduli that are vulnerable to the attack: simply look for moduli that satisfy $n \equiv 1 \bmod S$.

We note that the factoring algorithm works equally well even if p were generated as in the previous paragraph, but q were generated as a truly random ℓ-bit prime. Hence, the problem is not the relation between the primes. The attack is possible because p is not a uniform prime.

A real-world example. A sieving optimization of this nature, though somewhat more complicated, was found in a widely used crypto library used by the Infineon hardware to generate RSA keys. This was discovered in 2017 and the resulting attack, called the **ROCA attack** [124], factors the resulting RSA moduli. Because the Infineon key generation procedure was implemented on many consumer smartcards, all of them had to be recalled due to the attack.

16.4.3 A fault injection attack on optimized RSA

We now turn to another category of attacks called **fault injection** attacks. Let (n, e) be an RSA public key, and (n, d) be the corresponding private key, where $n = p \cdot q$, and $p > q$. Recall that the RSA trapdoor function is defined as $F(pk, x) := x^e \in \mathbb{Z}_n$ and its inverse is $I(sk, y) := y^d \in \mathbb{Z}_n$. The inverse function I is used in RSA signature generation and RSA decryption. Its running time is dominated by a d-th power exponentiation in $\mathbb{Z}_n$.

The inverse function $I(sk, y)$ can be implemented more efficiently using the Chinese Remainder Theorem (CRT), discussed in Appendix A. The optimized implementation runs about four times faster than the naive implementation and is called **RSA-CRT**. It uses a slightly modified secret key that contains five integers $sk' := (p, q, d_p, d_q, q_{\text{inv}})$, where p and q are the factors of n, and

$$d_p := d \bmod (p-1), \qquad d_q := d \bmod (q-1), \qquad q_{\text{inv}} := (1/q) \bmod p.$$

The optimized $I(sk', y)$ works as follows:

- *step 1:* $x_p \leftarrow (y^{d_p} \bmod p)$,
- *step 2:* $x_q \leftarrow (y^{d_q} \bmod q)$,
- *step 3:* find the unique $x \in \mathbb{Z}_n$ such that $x \equiv x_p \bmod p$ and $x \equiv x_q \bmod q$. This is done as:

  $$\text{(i) } x'_p \leftarrow q_{\text{inv}} \cdot (x_p - x_q) \bmod p, \qquad \text{and} \qquad \text{(ii) } x \leftarrow x_q + x'_p \cdot q.$$

  The reader can verify that indeed $x \equiv x_p \bmod p$ and $x \equiv x_q \bmod q$.
  The final output is $x \in \mathbb{Z}_n$.

By definition of d_p and d_q we know that $x_p = (y^d \bmod p)$ and $x_q = (y^d \bmod q)$. Therefore, by the Chinese Remainder Theorem, the final x satisfies $x = (y^d \bmod n)$ as required.

This optimized implementation is about four times faster than directly computing $x \leftarrow y^d$ in $\mathbb{Z}_n$. The source of the speed-up comes from the fact that p and q are about half the size of n, and the fact that the exponents d_p and d_q are half the size of d. When using a quadratic-time multiplication algorithm, arithmetic modulo p is four times faster than arithmetic modulo n. Moreover, because d_p is half the size of d, the overall time to compute $x_p \leftarrow (y^{d_p} \bmod p)$, is eight times faster than computing y^d in $\mathbb{Z}_n$. Since we compute both x_p and x_q, the overall running time is four times faster than the naive implementation.

A fault injection attack on RSA-CRT. The RSA-CRT optimization discussed above is commonly used in RSA implementations. These implementations can be vulnerable to a dangerous class of attacks called **fault injection attacks**.

Fault attacks are based on the fact that computers are not perfect. In particular, environmental interference can cause a computing device to make a calculation error. For example, a bit stored in a register can spontaneously flip due to external electromagnetic interference. If this happens even once to a register holding some intermediate value in a long calculation, then the final computed value will likely be incorrect. Under normal operating conditions, a spontaneous bit flip fault is rare, but it can and does happen from time to time. Even worse, in some settings, an attacker can induce faults using a fault injection process.

Let's see how this affects the RSA inversion function when it is used to sign a message m. Suppose that (n, e) is an RSA public key, and sk' is the corresponding RSA-CRT private key. Recall that the FDH-RSA signature is defined as $\sigma := I(sk', H(m))$ for some hash function $H : \mathcal{M} \to \mathbb{Z}_n$. To compute $I(sk', H(m))$ using RSA-CRT, the algorithm begins by first computing the integers x_p and x_q. Suppose that a fault occurs during the computation of x_p, but the computation of x_q completes correctly with no faults. Then x_p and x_q satisfy the following relations:

$$x_p \neq (H(m)^d \bmod p) \qquad \text{and} \qquad x_q = (H(m)^d \bmod q).$$

Let $\hat{\sigma} \in \mathbb{Z}_n$ be the final computed (faulty) signature derived from x_p and x_q, then

$$\hat{\sigma} \equiv x_p \not\equiv H(m)^d \pmod{p} \qquad \text{and} \qquad \hat{\sigma} \equiv x_q \equiv H(m)^d \pmod{q}.$$

Raising both sides to the power of e gives then

$$\hat{\sigma}^e \not\equiv H(m) \pmod{p} \qquad \text{and} \qquad \hat{\sigma}^e \equiv H(m) \pmod{q}.$$

To complete the attack, observe that if we treat $\hat{\sigma}^e - H(m) \in \mathbb{Z}_n$ as an integer in $[0, n)$, then this integer is divisible by q but is not divisible by p. Hence,

$$\gcd\big(\hat{\sigma}^e - H(m),\ n\big) = q.$$

In summary, knowledge of the public key (n, e), the message m, and the faulty signature $\hat{\sigma} \in \mathbb{Z}_n$, reveals the factorization of n. Once the factorization is revealed, anyone can calculate *sk* and sign arbitrary messages.

This fault attack shows that a *single* faulty signature on a known message will compromise the secret signing key. For this reason, many RSA-CRT implementations verify that the computed signature is valid before outputting it to the outside world. The same applies to RSA decryption; RSA inversion needs to be verified before the plaintext is released from the crypto library.

While faults can happen organically because of hardware failure, they can also be induced by an attacker. For example, suppose the RSA secret key is stored on a small secure device, like a smart card or a credit card, that is used to sign messages. The device is designed to make it difficult to extract *sk* from the device. However, an attacker that obtains the device can subject it to a harsh environment that is deliberately designed to cause a low rate of faults. The attacker then simply runs the device over an over to sign some fixed message m until finally it receives a faulty signature that exposes the secret key. In practice, the attack is executed using specialized equipment that heats up a particular part of the processor using a laser. The localized increase in temperature causes the required low rate faults to occur.

Fault injection attacks apply to many other algorithms, including ElGamal decryption, and even AES. The RSA-CRT example is a crisp example that shows why calculation errors are so damaging to cryptographic implementations.

16.4.4 An attack on low secret exponent RSA

Our last example is another natural optimization of RSA that can harm security. Let $pk = (n, e)$ be an RSA public key and let $sk = (n, d)$ be the corresponding private key. Recall that in RSA decryption and RSA signing, the time consuming step is the exponentiation $y \leftarrow x^d$, for some $x \in \mathbb{Z}_n$.

A natural (but not very good) idea to speed-up RSA decryption and signing is to choose $d \in \mathbb{Z}$ as an integer that is much smaller than n, but sufficiently large that it cannot be found by a brute force search. For example, one could choose d as a 128-bit random integer, and then calculate the corresponding public e. In particular, one could modify the RSA key generation procedure from Section 10.3 to work as follows:

```
$\text{RSAGen}(\ell, s) :=$          // for example, invoked as $\text{RSAGen}(2048, 128)$
    generate a random $\ell$-bit prime $p$
    generate a random $\ell$-bit prime $q$
    $n \leftarrow p \cdot q$
    choose a random $s$-bit integer $d$ where $\gcd(d, p-1) = \gcd(d, q-1) = 1$
    $e \leftarrow d^{-1} \bmod (p-1)(q-1)$
    output $pk := (n, e)$ and $sk := (n, d)$.
```

This is the reverse of how an RSA key is generated in Section 10.3, where e is chosen first, and then a matching d is calculated. The benefit of the procedure above is that the resulting d is much smaller, and therefore RSA decryption and signing will be significantly faster. For example, for a 2048-bit modulus n, the secret key d is normally a 2048 bits, but with the procedure above it could be as low as 128 bits. Decryption and signing are then $2048/128 = 16$ times faster because of the smaller d (or about four times faster than RSA-CRT).

Of course, encryption and signature verification will be slower because e is much larger than normal. In some settings, this is a worthwhile tradeoff, for example, when a low power signing device, such as a smartcard, is connected to a high power verifier.

Wiener's attack. The key generation procedure described above is insecure in the worst possible way: an eavesdropper who sees the public key (n, e) can quickly calculate the private key (n, d). The attack works whenever $d < n^{1/4}/3$. For example, if $n > 2^{2048}$ then the attack works whenever $d < 2^{512}/3$. Hence, choosing d as a 128 bit integer is clearly insecure. We state this in the following theorem.

Theorem 16.5 (Wiener, 1989). *Let (n, e) be two positive integers, where $n = pq$ is the product of two ℓ-bit primes. Moreover, suppose $d := e^{-1} \bmod (p-1)(q-1)$ exists and is less than $n^{1/4}/3$. Then there is an efficient algorithm $\mathcal{A}$ that takes (n, e) as input, and outputs d.*

Proof. Let $\varphi := (p-1)(q-1)$. First, observe that $d = e^{-1} \bmod \varphi$ implies that $ed = 1 \bmod \varphi$, and therefore there is an integer k such that $e \cdot d = k \cdot \varphi + 1$. Notice that this equality implies that $\gcd(k, d) = 1$. By dividing both sides by $d \cdot \varphi$ and rearranging terms, we obtain

$$\left| \frac{e}{\varphi} - \frac{k}{d} \right| = \frac{1}{d\varphi} < \frac{1}{\sqrt{n}}. \tag{16.12}$$

The rightmost inequality holds because $\varphi > \sqrt{n}$. We know nothing about the fraction (k/d). However, we have a very good approximation to the fraction (e/φ). In particular, $\varphi = n - p - q + 1$, so that $|n - \varphi| < p + q < 3\sqrt{n}$. In other words, n is a very good approximation to φ. Therefore, we can replace the fraction (e/φ) in (16.12) by the fraction (e/n) without affecting the inequality much. To do so, we will need the following simple fact:

$$\left| \frac{e}{n} - \frac{e}{\varphi} \right| = \left| \frac{e(n-\varphi)}{n\varphi} \right| < \frac{e}{\varphi} \cdot \frac{3\sqrt{n}}{n} < \frac{3}{\sqrt{n}}.$$

Now, writing (e/n) in (16.12) in place of (e/φ) gives:

$$\left|\frac{e}{n} - \frac{k}{d}\right| \leq \left|\frac{e}{n} - \frac{e}{\varphi}\right| + \left|\frac{e}{\varphi} - \frac{k}{d}\right| < \frac{3}{\sqrt{n}} + \frac{1}{\sqrt{n}} = \frac{4}{\sqrt{n}} < \frac{1}{2d^2}, \tag{16.13}$$

The last inequality follows from the assumption that $d < n^{1/4}/3$ which implies that $2d^2 < \sqrt{n}/4$.

The inequality (16.13) says that the rational number (e/n), which we have, is close to the fraction (k/d), which we want. The two fractions are so close, that their difference is less than $1/(2d^2)$. Such a strong approximation of one fraction by another is rare: there are very few fractions that satisfy (16.13) and all of them can be found efficiently. We state this in the following lemma.

Lemma 16.6. *Let $\alpha := e/n$. Let S be the set of all rational numbers (a/b) that satisfy*

$$\left|\alpha - \frac{a}{b}\right| < \frac{1}{2b^2} \ . \tag{16.14}$$

Then (i) the size of S is at most $2\log_2 n$ and (ii) there is an efficient algorithm that takes α as input, and outputs the set S using $O(\log n)$ arithmetic operations.

The proof of the lemma is based on a theorem of Legendre from the theory of continued fractions [84, Ch. 10]. The algorithm in part (ii) is an application of the extended Euclidean algorithm. We prove the lemma in Exercise 16.11.

To complete the proof of Theorem 16.5, algorithm $\mathcal{A}$ takes as input a public key (n, e) and works as follows. It runs the algorithm from part (ii) of Lemma 16.6 to obtain the set of all $2\log_2 n$ rational numbers satisfying (16.14). By (16.13), one of these rational numbers must be equal to k/d. Since $\gcd(k, d) = 1$, the target d must be the denominator of one of the numbers in the set. The algorithm tries all $2\log_2 n$ denominators until it finds one that works. □

The attack algorithm described in the proof of Theorem 16.5 is guaranteed to work up to the bound $d < n^{1/4}/3$, but may not work when d exceeds this bound. One might be tempted to set d to be slightly bigger than $n^{1/4}$ and still benefit from the speed-up obtained from a small d. However, this is also insecure. More powerful techniques are able to recover d from (n, e) even when d is about $n^{0.3}$. It is conjectured that d can be efficiently found whenever $d < n^{0.5}$, however, as of this writing, this is still an open problem. Taking d on the order of $n^{0.5}$ gives no speed-up over RSA-CRT, so we might as well use the standard RSA key generation procedure and benefit from a small e.

A different approach: small CRT exponents. In Section 16.4.3 we explained that RSA decryption and signing is typically implemented using the Chinese Remainder Theorem (CRT). Here the signer stores $d_p := (d \bmod p - 1)$ and $d_q := (d \bmod q - 1)$ and computes x^d in $\mathbb{Z}_n$ by first computing x^{d_p} in $\mathbb{Z}_p$ and x^{d_q} in $\mathbb{Z}_q$. The same applies to RSA decryption.

To speed up signing and decryption, we could slightly modify RSAGen(ℓ, s) described at the beginning of this section to choose d_p and d_q as random (small) s-bit integers, and then set d as the smallest integer d satisfying $d \equiv d_p \bmod (p-1)$ and $d \equiv d_q \bmod (q-1)$. This way, d is unlikely to be a small integer, so that the attack of Theorem 16.5 does not apply. However we still obtain a fast RSA-CRT signing algorithm because d_p and d_q are small. The resulting exponent d is called a *small CRT exponent*.

Unfortunately, this approach is also insecure. In the notes we refer to a number of attacks on this setup. For example, one can recover d from (n, e) whenever $s < 0.244 \cdot \ell$.

This completes our discussion of attacks on RSA. This section should convince the reader that it is best to not deviate from the RSA key generation procedure described in Section 10.3.

16.5 Quantum attacks on factoring and discrete log

To complete our review of attacks on number theoretic assumptions we must also describe the looming threat of a quantum computer. The story begins with physicists who tried to simulate quantum experiments on a digital computer. They quickly came to realize that the required computing resources are enormous, even for relatively simple experiments. This lead Richard Feynman to suggest that perhaps quantum experiments implement a new computing model that can be used to solve certain computational problems faster than what can be done on a classical digital computer. Astonishingly, this prediction turned out to be true. There are now numerous computing tasks for which a quantum algorithm vastly outperforms the best known classical algorithm.

One of the best examples is a quantum algorithm, due to Peter Shor, that solves both problems discussed in this chapter in polynomial time: the algorithm can efficiently factor integers, and efficiently compute discrete logarithm in every group. The details of Shor's algorithm require a detailed introduction to quantum computing. We refer the reader to [125] for the full details. Here we give a brief overview to explain the main idea.

Shor's algorithm. The core observation in Shor's algorithm is that a quantum computer is very good at detecting periodicity. Specifically, let $f : \mathbb{Z} \to S$ be some function. We say that f is **periodic** if there exists a $0 \neq \pi \in \mathbb{Z}$ such that $f(x) = f(x + \pi)$ for all $x \in \mathbb{Z}$. This π is called a *period* of f. If π is a period, then so is $k \cdot \pi$ for every $k \in \mathbb{Z}$. Let $\ell := |\pi_0|$, where π_0 is the smallest non-zero period of f.

Suppose that we are only given black-box access to the function f. We can evaluate f at any input x, but do not have access to its inner workings. Moreover, suppose that evaluating f at an input x in $\{0, 1, \ldots, \ell^2\}$ takes time $T(f)$ in the worst case. It is not difficult to show that on a classical computer, the best algorithm for finding a period needs to evaluate f at $\Omega(\ell)$ inputs, in the worst case. The magic of Shor's algorithm is that, on a quantum computer, it can find a period of f (i.e., a multiple of ℓ) in time $O(T(f) + \log \ell)$. This is an exponential speed-up over a classical computer: the linear dependence on ℓ on a classical computer is replaced by a logarithmic dependence on ℓ on a quantum computer.

We next show that factoring integers can be formulated as a period finding problem.

Quantum factoring. Let n be an integer, and let g be some non-zero element in $\mathbb{Z}_n$. Let $f_g : \mathbb{Z} \to \mathbb{Z}_n$ be the function

$$f_g(\alpha) := g^\alpha.$$

This function is periodic. Let π be a positive integer such that $g^\pi = 1$, then π is a period of f_g. Indeed, $f_g(\alpha) = g^\alpha = g^{\alpha+\pi} = f_g(\alpha + \pi)$ for all $\alpha \in \mathbb{Z}$. In fact, every period π of f_g must be an integer multiple of the order of g in the multiplicative group $\mathbb{Z}_n$.

To factor an RSA modulus $n = pq$, choose a random g in $\mathbb{Z}_n$, and use Shor's algorithm to find a period π of the function $f_g : \mathbb{Z} \to \mathbb{Z}_n$. We know that $g^\pi = 1$. Let t be the smallest positive integer such that $g^{\pi/2^t} \neq 1$. In Exercise 16.10 we show that $\gcd(n,\ g^{\pi/2^t} - 1)$ is a non-trivial factor of n with probability about one half over the choice of g. Hence, if we can find a period of f_g for a random $g \in \mathbb{Z}_n$, we can factor n.

Shor's algorithm finds some period π of f_g in time $O(T(f_g) + \log n)$, where $T(f_g)$ is the time to compute $f(\alpha) = g^\alpha$, for an integer α is in $[0, n^2]$. The naive exponentiation algorithm runs in cubic time, and therefore the total running time of this factoring algorithm is $O(\log^3 n)$. Hence, Shor's algorithm factors n in polynomial time in $\log_2 n$.

Quantum discrete log. A similar period finding approach can be used to compute discrete log. Let $\mathbb{G}$ be a finite cyclic group of order q with generator $g \in \mathbb{G}$. We want an algorithm that takes $u \in \mathbb{G}$ as input, and outputs an $\alpha \in \mathbb{Z}$ such that $u = g^\alpha$. We can reformulate this as a period finding problem, but this time for a function $f : \mathbb{Z}^2 \to S$ defined over the two-dimensional domain $\mathbb{Z}^2$. We say that $\boldsymbol{\pi} = (\pi_1, \pi_2) \in \mathbb{Z}^2$ is a period of f if $f(\boldsymbol{x}) = f(\boldsymbol{x} + \boldsymbol{\pi})$ for all $\boldsymbol{x} \in \mathbb{Z}^2$.

Now, let $u = g^\alpha$. To compute the discrete log of u base g, consider the function $f : \mathbb{Z}^2 \to \mathbb{G}$ defined as

$$f(\gamma, \delta) := g^\gamma \cdot u^\delta.$$

It is easy to see that $(0, q)$ and $(q, 0)$ are periods of this function, but these periods reveal nothing about the discrete log α. However, there is another period, namely $(\alpha, -1)$, that is more useful. To see that $(\alpha, -1)$ is a period observe that for all $(\gamma, \delta) \in \mathbb{Z}^2$

$$f(\gamma + \alpha, \delta - 1) = g^{\gamma+\alpha} \cdot u^{\delta-1} = (g^\gamma u^\delta) \cdot (g^\alpha / u) = f(\gamma, \delta).$$

It is not difficult to show that the set of periods of the function f is the set of all integer linear combinations of the three vectors $\big\{(q, 0),\ (0, q),\ (\alpha, -1)\big\}$. This set forms a two dimensional lattice L in $\mathbb{Z}^2$.

Shor's quantum algorithm samples a random period (π_1, π_2) from L whose length is at most q^2. Then $(\pi_1, \pi_2) = a \cdot (q, 0) + b \cdot (0, q) + c \cdot (\alpha, -1)$ for some integers $a, b, c \in \mathbb{Z}$. Therefore,

$$\big((\pi_1, \pi_2) \bmod q\big) = (\hat{c}\alpha, -\hat{c}) \quad \in \mathbb{Z}_q^2$$

for some $\hat{c} \in \mathbb{Z}_q$, and with high probability $\hat{c} \neq 0$. Therefore, computing $\hat{\alpha} := -\pi_1/\pi_2$ in $\mathbb{Z}_q$ reveals the discrete log of u to the base g, as required. The algorithm's running time is dominated by the time to compute the function $f(\gamma, \delta)$ for $\gamma, \delta \in \{0, \ldots, q^2\}$. Using naive exponentiation, the algorithm runs in time $O(\log^3 q)$.

The remarkable thing about this algorithm is that it works in all groups of prime order q, and only uses the group operation as a black box. On a classical computer, one can show that the best algorithm in this setting runs in time $\Omega(\sqrt{q})$ (see, e.g., [145]). Hence, again, a quantum algorithm gives an exponential improvement over the best classical method.

What does this mean? If the postulates of quantum mechanics are correct, then building a quantum computer large enough to carry out Shor's algorithm is mainly an engineering problem, although a very challenging one. Steady progress is being made, and it is anticipated that a large enough quantum computer will be built sometime this century. Because of this, the National Institute of Standards (NIST) is standardizing a family of key exchange protocols and signature schemes that remain secure against an adversary that has access to a large scale quantum computer. These systems are collectively called *post-quantum cryptography*.

Currently, the most efficient post-quantum key exchange protocols result in considerably more traffic than Diffie-Hellman key exchange. Hence, there is some cost to deploying these protocols in practice. Nevertheless, for traffic that is encrypted today and needs to remain secret for more than thirty years, it is important to use a post-quantum cryptosystem.

16.6 Notes

To be written.

16.7 Exercises

16.1 (Multi instance discrete log). Let $\mathbb{G}$ be a cyclic group of order q generated by $g \in \mathbb{G}$. Let $\alpha_1, \ldots, \alpha_n \xleftarrow{\text{R}} \mathbb{Z}_q$, and set $u_i := g^{\alpha_i}$ for $i = 1, \ldots, n$. Suppose we are given the n group elements $u_1, \ldots, u_n \in \mathbb{G}$ and wish to compute the discrete log of all of them to the base g. Naively, this requires running a discrete log algorithm n times, which takes time $O(n\sqrt{q})$ using the baby step/giant step algorithm from Section 16.1.1. Show how to modify the baby step/giant step algorithm so that computing the discrete log of all $u_1, \ldots, u_n \in \mathbb{G}$ to the base g only takes time $O(\sqrt{nq})$. This is much better than the naive method.
Hint: try running the baby step for $\sqrt{nq}$ steps instead of $\sqrt{q}$ steps.

16.2 (An attack on DDH in composite order groups). In Section 16.1.3 we presented an algorithm that has advantage 1/2 in deciding DDH in a cyclic group $\mathbb{G}$ whose order is even. Let $\mathbb{G}$ be a cyclic group of order q, and let ℓ be the smallest prime factor of q.

(a) Generalize the DDH algorithm from Section 16.1.3 to give an algorithm that has advantage about $1/\ell$ in deciding DDH in $\mathbb{G}$. Your algorithm should use only $O(\log q)$ group operations.

(b) Give a an algorithm to decide DDH in $\mathbb{G}$ with advantage $1 - (1/\ell)$. Your algorithm can use $O(\sqrt{\ell} + \log q)$ group operations.

Your algorithms show that if the smallest prime factor of q is small, then DDH is false in $\mathbb{G}$.

16.3 (The Brown-Gallant-Cheon algorithm). Let $\mathbb{G}$ be a cyclic group of order q with generator $g \in \mathbb{G}$. For $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$, let

$$\boldsymbol{v}_\alpha := \left(g^\alpha, g^{(\alpha^2)}, \ldots, g^{(\alpha^d)}\right).$$

Suppose that the best discrete log algorithm in $\mathbb{G}$ runs in time $\Omega(\sqrt{q})$. We show that in some groups $\mathbb{G}$, the information provided in $\boldsymbol{v}_\alpha$ can lead to a faster algorithm to find α.

(a) Suppose $q-1$ is divisible by an integer $s \leq d$. Design an algorithm $\mathcal{A}_1$ that takes $\boldsymbol{v}_\alpha$ as input, and outputs α using only $\tilde{O}\big(\sqrt{q/s} + \sqrt{s}\big)$ group operations.
Hint: Observe that α^s lies in a subgroup of $\mathbb{Z}_q^*$ of size $(q-1)/s$. Let $\gamma \in \mathbb{Z}_q^*$ be a generator of this subgroup and write $\alpha^s = \gamma^u$ for some (unknown) integer $0 \leq u < (q-1)/s$. First, show how to find u in time $O\big(\sqrt{q/s} \cdot \log q\big)$. using the baby-step-giant-step algorithm from Section 16.1.1. Then show how to find α in additional time $O\big(\sqrt{s} \cdot \log q\big)$.

(b) Suppose $q+1$ is divisible by some integer $s \leq d$. Design an algorithm $\mathcal{A}_2$ such that $\mathcal{A}_2(\boldsymbol{v}_\alpha) = \alpha$, and $\mathcal{A}_2$ uses only $\tilde{O}(\sqrt{q/s} + s)$ group operations.

Hint: Use the fact that a quadratic extension of $\mathbb{F}_q$ has a subgroup of size $q+1$. Then use the same approach as in part (a). If you get stuck, see [42].

Discussion: Algorithms $\mathcal{A}_1$ and $\mathcal{A}_2$ show that if either $q+1$ or $q-1$ are divisible by d, then the data provided in $\boldsymbol{v}_\alpha$ can be used to speed up the recovery of α by about a factor of $\sqrt{d}$ over a generic discrete log algorithm. When d is small, this does not impact security much. Nevertheless, if needed, one can compensate by either using a group $\mathbb{G}$ with a slightly larger q, or by ensuring that $q+1$ and $q-1$ do not have moderate-size factors. See the discussion in Section 16.1.4, which also points to several places in the book where this algorithm is relevant.

16.4 (An expected time discrete log algorithm). Let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$. Construct a randomized algorithm that has *expected constant running time* and succeeds in computing discrete log in $\mathbb{G}$ with probability at least $1/\sqrt{q}$. The probability is over the random bits of the algorithm. Your algorithm should be generic: it should treat group elements as opaque strings, and treat the group operation as a black box.

Hint: try an algorithm that tosses a biased coin that comes up heads with probability $1/\sqrt{q}$. When the coin comes up heads the algorithm runs for $\sqrt{q}$ steps. Otherwise, the algorithm outputs 0 and stops.

Discussion: Theorem 16.3 shows that a generic algorithm that has *worst-case* constant running time can succeed with at most probability $1/q$ in computing discrete log. Your algorithm shows that a generic algorithm that has *expected* constant running time can do much better: it succeeds with probability $1/\sqrt{q}$. How much better can an expected running time algorithm do? One can adapt Theorem 16.3 and show that a generic discrete log algorithm that has *expected* running time T succeeds in computing discrete log with probability $O(T/\sqrt{q})$. See [94].

16.5 (A generic lower bound on CDH). In Section 16.3 we proved a lower bound on the time to compute discrete log in a generic group $\mathbb{G}$ of prime order q.

(a) Adapt the proof of Theorem 16.3 to prove a similar lower bound on the time to compute CDH in $\mathbb{G}$ with probability one half.

(b) Generic lower bounds can also be proved for interactive games. Adapt the proof of Theorem 16.3 to prove a lower bound on the time to solve the interactive computational Diffie-Hellman (ICDH) problem from Definition 12.4 with probability one half.

16.6 (A multiple of $\varphi(n)$ reveals the factorization of n). Let n be a composite integer and let k be a multiple of $\varphi(n) = (p-1)(q-1)$, that is, $k = c \cdot \varphi(n)$ for some (unknown) integer c. Show an efficient algorithm that takes n and k as input, and outputs a non-trivial factor of n.

16.7 (An insecure variant of RSA). Consider again the RSA-based public-key encryption scheme $\mathcal{E}_{\text{RSA}}$ from Section 11.4.1. Suppose we change the scheme so that all users make use of the same RSA modulus n, but every user is assigned its own encryption exponent e and a corresponding decryption exponent d. For example, Alice is assigned (e_1, d_1) and Bob is assigned (e_2, d_2). Show that this scheme is insecure: Alice can decrypt all messages sent to Bob.

16.8 (Discrete log in $\mathbb{Z}_n$ is as hard as factoring). Let $n = pq$ be an RSA modulus with an unknown factorization. Let $g \in \mathbb{Z}_n^*$ be an element of order at least n/k, for some k, and let $\mathbb{G}$ be the subgroup of $\mathbb{Z}_n^*$ generated by g. Show that an oracle for computing discrete log to the base g in

$\mathbb{G}$ can be used to find the factorization of n using an expected $O(k)$ calls to the oracle. This shows that for small k, computing discrete log in $\mathbb{G}$ is at least as hard as factoring n.

16.9 (Computing some roots in $\mathbb{Z}_n$ is as hard as factoring). Let $n = pq$ be an RSA modulus with an unknown factorization.

(a) Let $\mathcal{O}_2$ be an oracle that takes as input $u \in \mathbb{Z}_n$ and returns a square root of u in $\mathbb{Z}_n$ if one exists, and $\perp$ otherwise. Show an efficient algorithm that uses the oracle $\mathcal{O}_2$ to factor n.

(b) Suppose that a poly-bounded integer e divides $p - 1$, but e may or may not divide $q - 1$. Let $\mathcal{O}_e$ be an oracle that takes as input $u \in \mathbb{Z}_n$ and returns an e-th root of u in $\mathbb{Z}_n$ if one exists, and $\perp$ otherwise. Show an efficient algorithm that uses the oracle $\mathcal{O}_e$ to factor n.

Discussion: These results do not prove that breaking RSA is as hard as factoring. To do that, one has to show that an oracle $\mathcal{O}_e$, for some e where $\gcd(e, \varphi(n)) = 1$, can be used to efficiently factor n. Neither parts (a) or (b) apply to such e, and in fact, this is still an open problem.

16.10 (Support for quantum factoring). In this exercise we prove some basic facts that are needed for the quantum factoring algorithm from Section 16.5. Let $n = pq$ be an RSA modulus.

(a) Use the Chinese Remainder Theorem to show that there are four square roots of unity in $\mathbb{Z}_n$. That is, there are four distinct elements $r_1, r_2, r_3, r_4 \in \mathbb{Z}_n$ such that $r_1^2 = r_2^2 = r_3^2 = r_4^2 = 1$. Moreover, $\gcd(n,\ r_i - 1)$ is a non-trivial factor of n for exactly two of these four square roots.

(b) Let $g \in \mathbb{Z}_n^*$, and let π be an integer multiple of the order of g in $\mathbb{Z}_n^*$. Then $g^\pi = 1$. Let t be the smallest positive integer such that $h := g^{\pi/2^t} \neq 1$. By definition $h^2 = 1$. Show that if g is chosen uniformly in $\mathbb{Z}_n^*$, then h is uniformly distributed among the four square roots of unity.

Combining parts (a) and (b) we conclude that $\gcd(n, g^{\pi/2^t} - 1)$ is a non-trivial factor of n with probability 1/2 over the choice of g in $\mathbb{Z}_n^*$.

16.11 (Proof of Lemma 16.6). Let e, n be two integers. Let S be the set of all rational numbers (a/b) such that $\big|(e/n) - (a/b)\big| < 1/(2b^2)$.

(a) Show that if $(a/b) \in S$, where $\gcd(a, b) = 1$ and $b > 0$, then $b \leq n$. This proves that the set S is finite.
Hint: use the fact that if $(a/b) \neq (e/n)$ then $\big|eb - an\big|$ is a non-zero integer, and therefore must be at least 1.

(b) Let $S = \{(a_1/b_1), \ldots, (a_k/b_k)\}$ where $b_1 \leq b_2 \leq \ldots \leq b_n$. Show that $b_{i+2} > 2b_i$ for all $i = 1, \ldots, k - 2$. Deduce from this and from part (a) that $|S| \leq 2\log_2 n$, as stated in part (i) of Lemma 16.6.
Hint: for every three consecutive numbers (a_i/b_i), (a_{i+1}/b_{i+1}), (a_{i+2}/b_{i+2}) in the set S, at least two are on the same side of (e/n). Say, (a_i/b_i) and (a_{i+2}, b_{i+2}) are on the same side. Then $\big|(a_i/b_i) - (a_{i+2}/b_{i+2})\big| < 1/(2b_i^2)$. Show that this implies that $b_{i+2} > 2b_i$.

(c) Show how to efficiently construct all the numbers in S using the extended Euclidean algorithm. If you get stuck, see [148, Ex. 4.18(d)].

16.12 (Factoring challenge #1). The following integer n is a product of two primes p and q such that $\big|p - q\big| < 2n^{1/4}$. Use the factoring algorithm from Section 16.4.2.1 to find the factors of

```
n = 17976931348623159077293051907890247336179769789423065727343008115 \
    77326758055056206869853794492129829595855013875371640157101398586 \
    47833778606925583497541085196591615128057575940752635007475935288 \
    71082364994994077189561705436114947486504671101510156394068052754 \
    0071584560878577663743040086340742855278549092581
```

16.13 (Factoring challenge #2). The following integer n is a product of two primes p and q such that $\left|p-q\right| < 2^{11} \cdot n^{1/4}$. Find the factors of

```
n = 6484558428080716696628242653467722787263437207069762630604390703787 \
    9730861808111646271401527606141756919558732184025452065542490671989 \
    2428844841839353281972988531310511738648965962582821502504990264452 \
    1008852816733037111422964210278402893076574586452336833570778346897 \
    15838646088239640236866252211790085787877
```

Hint: Let $A = (p+q)/2$. Show that $A - \sqrt{n} < 2^{20}$. Then try scanning for A from $\sqrt{n}$ upwards, until you succeed in factoring n.

16.14 (Factoring challenge #3). The following integer n is a product of two primes p and q such that $\left|3p-2q\right| < n^{1/4}$. Find the factors of

```
n = 72006226374735042527956443552558373833808445147399984182665305798191 \
    63556901883377904234086641876639384851752649940178970835240791356868 \
    77441155132015188279331812309091996246361896836573643119174094961348 \
    52463970788523879939683923036467667022162701835329944324119217381272 \
    9276147530748597302192751375739387929
```

Hint: Generalize Lemma 16.4 to show that $\sqrt{6n}$ is close to $\frac{3p+2q}{2}$. Then adapt the factoring algorithm from Section 16.4.2.1 to factor n.

16.15 (Speeding up RSA inversion). In Section 16.4.3 we saw that inverting the RSA function using RSA-CRT is about four times faster than the naive implementations of RSA inversion.

(a) Show that if the RSA modulus n is a product of three equal size primes, $n = pqr$, then inverting the RSA function using RSA-CRT is about nine times faster than the naive implementation.

(b) For the next part of the question we will need the following fact called **Hensel lifting**. Let p be a prime, let $c \in \mathbb{Z}_p^*$, and let $x \in \mathbb{Z}$ satisfy $x^e \equiv c \pmod{p}$. Set

$$x' := x - \frac{x^e - c}{ex^{e-1}} \bmod p^2. \tag{16.15}$$

Use the binomial theorem to show that $x' \in \mathbb{Z}$ satisfies $(x')^e \equiv c \pmod{p^2}$. Equation (16.15) lets us quickly calculate the e'th root of c modulo p^2 from the e'th root of c modulo p. The equation *lifts* the e'th root of c from p to p^2.

(c) Suppose the RSA modulus n is chosen so that $n = p^2q$, for two equal size primes p and q. Further, suppose that the RSA public exponent e is small, say $e = 3$. Let d be the corresponding private exponent. Show that inverting the RSA function can be done about thirteen times faster than the naive implementation. In particular, for a given $y \in \mathbb{Z}_n^*$, use the following procedure to compute $y^d \bmod n$: (i) compute $x_p := y^d \bmod p$ and $x_q := x^d \bmod q$, (ii) use (16.15) to compute $x_p' := y^d \bmod p^2$, and (iii) apply the CRT to x_p' and x_q to obtain $x \in \mathbb{Z}_n$ such that $x = y^d \bmod n$. Analyze the running time of the resulting algorithm, and show that it is about thirteen times faster than naively computing y^d in $\mathbb{Z}_n$.

Part III

Protocols

Chapter 18

Protocols for identification and login

We now turn our attention to the identification problem, also known as the login problem. Party A wishes to identify itself to party B to gain access to resources available at B. She does so using an identification protocol, which is one of the fundamental tools provided by cryptography. We give a few illustrative applications that will be used as motivation throughout the chapter.

Opening a door lock. Alice wants to identify herself to a digital door lock to gain access to a building. Alice can use a simple password system: she inserts her key into the door lock and the door lock opens if Alice's key provides a valid password. A closely related scenario is a local login screen on a computer or a mobile phone. Alice wants to identify herself to the computer to gain access. Again, she can use a password to unlock the computer or mobile phone.

Unlocking a car. Alice wants to unlock her car using a *wireless* hardware key, called a **key fob**, that interacts with the car. An adversary could eavesdrop on the radio channel and observe one or more conversations between the wireless key fob and the car. Nevertheless, this *eavesdropping adversary* should not be able to unlock the car itself.

Login at a bank's automated teller machine (ATM). Alice wants to withdraw cash from her account using a bank ATM. The problem is that she may be interacting with a fake ATM. A report from a large ATM equipment manufacturer explains that fake ATM's are a big concern for the banking industry [129]:

> The first recorded instance of using fake ATMs dates back to 1993 when a criminal gang installed a fake ATM at a shopping mall in Manchester. Like most fake equipment it was not designed to steal money. Instead, the fake ATM appeared to customers as if it did not work, all the while stealing card data from everyone who attempted to use it.

Using a fake ATM, the adversary can interact with Alice in an attempt to steal her credential, and later use the credential to authenticate as Alice. We call this an *active* adversary. We aim to design identification protocols that ensure that even this active adversary cannot succeed.

Login to an online bank account. Our final example is remote login, where Alice wants to access her online bank account. Her web browser first sets up a secure channel with the bank. Alice then runs an identification protocol over the secure channel to identify herself to the bank, say using a password. As in the ATM example, an adversary can clone the bank's web site and fool Alice into identifying herself to the adversary's site. This attack, called **phishing**, is another example where the adversary can play an *active* role while interacting with Alice. The adversary tries to steal her credential so that it can later sell the credential to anyone who wishes to impersonate Alice to the real bank. Again, we aim to ensure that even a phishing adversary cannot learn a working credential for Alice. We discuss phishing attacks in more detail in Section 21.11.1 where we also discuss a potential cryptographic defense.

Identification (ID) protocols. Identification protocols are used in all the scenarios above. Abstractly, the identification problem involves two parties, a **prover** and a **verifier**. In our ATM example, Alice plays the role of prover while the ATM machine plays the role of verifier. The prover has a **secret key** *sk* that it uses to convince the verifier of its identity. The verifier has a corresponding **verification key** *vk* that it uses to confirm the prover's claim. We will occasionally refer to the prover as a human user and refer to the verifier as a computer or a server.

The motivating examples above suggest three attack models for ID protocols, ordered from weakest to strongest. We will discuss these models in detail in the coming sections.

- **Direct attacks**: The door lock and local login examples describe interactions between a prover and a verifier that are in close physical proximity. Suppose that the adversary cannot eavesdrop on this conversation. Then using no information other than what is publicly available, the adversary must somehow impersonate the prover to the verifier. A simple password protocol is sufficient to defend against such direct attacks.

- **Eavesdropping attacks**: In the wireless car key example the adversary can eavesdrop on the radio channel and obtain the transcript of several interactions between the prover and verifier. In this case the simple password protocol is insecure. However, a slightly more sophisticated protocol based on one-time passwords is secure.

- **Active attacks**: The last two examples, a fake ATM and online banking, illustrate an active adversary that interacts with the prover. The adversary uses the interaction to try and learn something that will let it later impersonate the prover to the verifier. Identification protocols secure against such active attacks require interaction between the prover and verifier. They use a technique called challenge-response.

Active attacks also come up when Alice tries to log in to a local infected computer. The malware infecting the computer could display a fake login screen and fool Alice into interacting with it, thus mounting an active attack. Malware that steals user passwords this way is called a **Trojan horse**. The stolen password can then be used to impersonate Alice to other machines.

Secret vs public verification keys. In some ID protocols the verifier must keep its verification key *vk* secret, while in other protocols *vk* can be public. We will see examples of both types of protocols. Clearly protocols where *vk* can be public are preferable since no damage is caused if the verifier (e.g., the ATM) is compromised.

Stateless vs stateful protocols. Ideally, *vk* and *sk* should not change after they are chosen at setup time. In some protocols, however, *vk* and *sk* are updated every time the protocol executes: the prover updates *sk* and the verifier updates *vk*. Protocols where *vk* and *sk* are fixed forever are called **stateless**. Protocols where *vk* and *sk* are updated are called **stateful**. Some stateful protocols provide higher levels of security at lower cost than their stateless counterparts. However, stateful protocols can be harder to use because the prover and verifier must remain properly synchronized.

One-sided vs mutual identification. In this chapter we only study the **one-sided identification** problem, namely Bob wishes to verify Alice's identity. **Mutual identification**, where Bob also identifies itself to Alice, is a related problem and is explored in Exercise 18.1. We will come back to this question in Chapter 21, where we construct mutual identification protocols that also generate a shared secret key.

Security and limitations of identification protocols. Identification protocols are designed to prevent an adversary from impersonating Alice without Alice's assistance. When defining the security of identification protocols, we may allow the adversary to eavesdrop and possibly interact with Alice; however, when it comes time to impersonate Alice, the adversary must do so without communicating with Alice. The examples above, such as opening a door lock, give a few settings where the primary goal is to prevent impersonation when Alice is not present.

ID protocols, however, are not sufficient for establishing a secure session between Alice and a remote party such as Alice's bank. The problem is that ID protocols can be vulnerable to a man in the middle (MiTM) attack. Suppose Alice runs an identification protocol with her bank over an insecure channel: the adversary controls the channel and can block or inject messages at will. The adversary waits for Alice to run the identification protocol with her bank and relays all protocol messages from one side to the other. Once the identification protocol completes successfully, the adversary sends requests to the bank that appear to be originating from Alice's computer. The bank honors these requests, thinking that they came from Alice. In effect, the adversary uses Alice to authenticate to the bank and then "hijacks" the session to send his own messages to the bank.

To defeat MiTM attacks, one can combine an identification protocol with a session key exchange protocol, as discussed in Chapter 21. The shared session key between Alice and her bank prevents the adversary from injecting messages on behalf of Alice.

18.1 Interactive protocols: general notions

Before getting into the specifics of identification protocols, we make a bit more precise what we mean by an **interactive protocol** in general.

An interactive protocol can be carried out among any number of parties, but in this text, we will focus almost exclusively on two-party protocols. Regardless of the number of parties, a protocol may be run many times. Each such protocol run is called a **protocol instance**.

In any one protocol instance, each party starts off in some initial configuration. As the protocol instance runs, parties will send and receive messages, and update their local configurations. While the precise details will vary from protocol to protocol, we can model the computation of each party in a protocol instance in terms of an **interactive protocol algorithm**, which is an efficient probabilistic algorithm I that takes as input a pair $(\mathit{config}_{\text{old}}, \mathit{data}_{\text{in}})$ and outputs a pair $(\mathit{config}_{\text{new}}, \mathit{data}_{\text{out}})$. When a party executes a protocol instance, it starts by supplying an **input value**, which defines the **initial configuration** of the protocol instance for that party. When the party receives a message over the network (presumably, from one of its peers), algorithm I is invoked on input $(\mathit{config}_{\text{old}}, \mathit{data}_{\text{in}})$, where $\mathit{config}_{\text{old}}$ is an encoding of the current configuration, and $\mathit{data}_{\text{in}}$ is an encoding of the incoming message; if the output of I is $(\mathit{config}_{\text{new}}, \mathit{data}_{\text{out}})$, then $\mathit{config}_{\text{new}}$ is an encoding of the new configuration, and $\mathit{data}_{\text{out}}$ encodes an outgoing message. The party sends this outgoing message over the network (presumably, again, to one of its peers). The party iterates this as many times as required by the protocol, until some **terminal configuration**

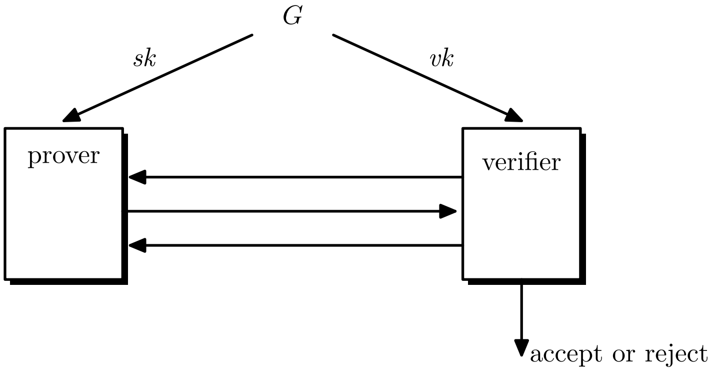


Figure 18.1: Prover and verifier in an ID protocol

is reached. This terminal configuration may specify an **output value**, which may be used by the party, presumably in some higher-level protocol.

In general, a given party may run many protocols, and even several instances of the same protocol, possibly concurrently. The configurations of all of these different protocol instances are separately maintained.

18.1.1 Mathematical details

As usual, one can define things more precisely using the terminology defined in Section 2.3. This is quite straightforward: along with the inputs described above, an interactive protocol algorithm I also takes as input a security parameter λ and a system parameter Λ. There are, however, a couple of details that deserve discussion.

For simplicity, we shall insist that the configuration size of a running protocol instance is poly-bounded — that is, the configuration can be encoded as a bit string whose length is always bounded by some fixed polynomial in λ. This allows us to apply Definition 2.8 to algorithm I. That definition assumes that the length of any input to an efficient algorithm is poly-bounded. So the requirement is that for every poly-bounded input to I, the output produced by I is poly-bounded.

The problem we are trying to grapple with here is the following. Suppose that after each round, the configuration size doubles. After a few rounds, this would lead to an exponential explosion in the configuration size, even though at every round, the computation runs in time polynomial in the current configuration size. By insisting that configuration sizes remain poly-bounded, we avoid this problematic situation.

For simplicity, we will also insist that the "round complexity" of a protocol is also poly-bounded. We will mainly be interested here in protocols that run in a *constant* number of rounds. More generally, we allow for protocols whose round complexity is bounded by some fixed polynomial in λ. This can be reasonably enforced by requiring that starting from any initial configuration, after a polynomial number of iterations of I, a terminal configuration is reached.

18.2 ID protocols: definitions

We start by defining the algorithms shown in Fig. 18.1 that comprise an ID protocol.

Definition 18.1. *An **identification protocol** is a triple $\mathcal{I} = (G, P, V)$.*

- *G is a probabilistic, **key generation** algorithm, that takes no input, and outputs a pair (vk, sk), where vk is called the **verification key** and sk is called the **secret key**.*
- *P is an interactive protocol algorithm called the **prover**, which takes as input a secret key sk, as output by G.*
- *V an interactive protocol algorithm called the **verifier**, which takes as input a verification key vk, as output by G, and which outputs* accept *or* reject*.*

We require that when $P(sk)$ and $V(vk)$ interact with one another, $V(vk)$ always outputs accept*. That is, for all possible outputs (vk, sk) of G, if P is initialized with sk, and V is initialized with vk, then with probability 1, at the end of the interaction between P and V, V outputs* accept*.*

18.3 Password protocols: security against direct attacks

In the **basic password protocol**, the prover's secret key is a **password** pw. In this protocol, the prover sends pw to the verifier, who checks that pw is the correct password. Thus, the secret key sk is simply $sk := pw$. Clearly this protocol should only be used if the adversary cannot eavesdrop on the interaction between prover and verifier. To complete the description of the basic password protocol, it remains to specify how the verifier checks that the given password is correct.

The first thing that comes to mind is to define the verifier's verification key as $vk := pw$. The verifier then simply checks that the password it receives from the prover is equal to vk. This naive password protocol is problematic and should never be used. The problem is that a compromise of the verifier (the server) will expose all passwords stored at the verifier in the clear.

Fortunately, we can easily avoid this problem by giving the verifier a hash of the password, instead of the password itself. We refer to the modified protocol as **version 1**. We describe this protocol in a rather idealized way, with passwords chosen uniformly at random from some finite password space; in practice, this may not be the case.

Password protocol (version 1). The prover's secret key sk is a password pw, chosen at random from some finite password space $\mathcal{P}$, while the verifier's key vk is $H(pw)$ for some hash function $H : \mathcal{P} \to \mathcal{Y}$. Formally, the **password ID protocol** $\mathcal{I}_{\text{pwd}} = (G, P, V)$ is defined as follows:

- G: set $pw \overset{\text{R}}{\leftarrow} \mathcal{P}$ and output $sk := pw$ and $vk := H(pw)$.
- Algorithm P, on input $sk = pw$, and algorithm V, in input $vk = H(pw)$, interact as follows:
  1. P sends pw to V;
  2. V outputs accept if the received pw satisfies $H(pw) = vk$;
     it outputs reject otherwise.

In a multi-user system the verifier (server) stores a password file that abstractly looks like Fig. 18.2. Consequently, an attack on the server does not directly expose any passwords.

To analyze the security of this protocol we formally define the notion of security against direct attacks, and then explain why this protocol satisfies this definition.

id_1	$H(pw_1)$
id_2	$H(pw_2)$
id_3	$H(pw_3)$
$\vdots$	$\vdots$

Figure 18.2: The password file stored on the server (version 1)

Attack Game 18.1 (Secure identification: direct attacks). For a given identification protocol $\mathcal{I} = (G, P, V)$ and a given adversary $\mathcal{A}$, the attack game runs as follows:

- *Key generation phase.* The challenger runs $(vk, sk) \overset{R}{\leftarrow} G()$, and sends vk to $\mathcal{A}$.
- *Impersonation attempt.* The challenger and $\mathcal{A}$ now interact, with the challenger following the verifier's algorithm V (with input vk), and with $\mathcal{A}$ playing the role of a prover, but not necessarily following the prover's algorithm P (indeed, $\mathcal{A}$ does not receive the secret key sk).

We say that the adversary wins the game if V outputs accept at the end of the interaction. We define $\mathcal{A}$'s advantage with respect to $\mathcal{I}$, denoted ID1adv$[\mathcal{A}, \mathcal{I}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 18.2. *We say that an identification protocol $\mathcal{I}$ is **secure against direct attacks** if for all efficient adversaries $\mathcal{A}$, the quantity* ID1adv$[\mathcal{A}, \mathcal{I}]$ *is negligible.*

Note that the adversary in Attack Game 18.1 is given the verifier's key vk. As a result, a naive password protocol where cleartext passwords are stored on the server does not satisfy Definition 18.2. The following simple theorem shows that the version 1 protocol is secure.

Theorem 18.1. *Suppose that hash function $H : \mathcal{P} \to \mathcal{Y}$ is one-way (as in Definition 8.6). Then the ID protocol $\mathcal{I}_{pwd}$ is secure against direct attacks.*

Proof sketch. To attack the protocol the adversary must come up with a password pw' such that $H(pw') = H(pw)$. Note that pw' may be different from pw. An adversary who can come up with such a pw' obviously breaks the one-wayness of H. □

We note that security against direct attacks (Attack Game 18.1) is a very weak notion of security. For example, although $\mathcal{I}_{\text{pwd}}$ is secure against direct attacks, it is clearly insecure if the adversary can eavesdrop on just a single instance of the protocol.

18.3.1 Password cracking using a dictionary attack

The password protocol $\mathcal{I}_{\text{pwd}}$ is widely used in practice because it is so easy to use. Anyone can memorize a password pw and participate in the protocol, playing the role of prover, without any additional hardware. The problem is that humans are terrible at generating and memorizing random passwords. In practice, passwords are typically very short. Even worse, passwords are usually not generated at random, but rather, selected by humans in rather predictable ways.

Figure 18.3 summarizes the results of a study [50] conducted in 2016 that examined five million leaked passwords that were mostly held by users in North America and Western Europe. The

```
123456,   password,     12345,     12345678,  football,
qwerty,   1234567890,   1234567,   princess,  1234,
login,    welcome,      solo,      abc123,    admin
```

Figure 18.3: The 15 most common passwords in 2016 listed in order

data shows that the passwords are not at all close to uniformly distributed over some large set, and in particular, a good percentage of passwords belong to a relative small dictionary of common passwords. About 4% of people use the password "123456" and about 10% use one of the passwords in the list of top 25 most common passwords. The list of passwords in Figure 18.3 is remarkably stable over time. It changes very little from year to year.

From now on, we will use the term **strong password** to mean a password that is chosen uniformly at random from a large password space $\mathcal{P}$. Theorem 18.1 applies only if passwords are strong. A **weak password** is one that is chosen (with some arbitrary distribution) from some small dictionary of common passwords, which we will denote by $\mathcal{D}$, where $\mathcal{D} \subseteq \mathcal{P}$.

18.3.1.1 Online dictionary attacks

Suppose an adversary suspects that a certain user's password is weak, and belongs to some small dictionary $\mathcal{D}$ of common passwords. Then the adversary can mount an **online dictionary attack** by simply trying to log in with all words in $\mathcal{D}$ one after the other, until a valid password is found. To speed things up, the attacker can sort $\mathcal{D}$ by popularity and try the most popular passwords first.

A common defense against online dictionary attacks is to double the server's response time after every 2 failed login attempts for a specific user ID or from a specific IP address. Thus, after 10 failed login attempts the time to reject the next attempt is 32 times the normal response time. This approach does not lock out an honest user who has a vague recollection of his own password. However, trying many password guesses for a single user becomes difficult.

Attackers adapt to this strategy by trying a single common password, such as `123456`, across many different usernames. These repeated attempts leverage client machines, called bots, located at different IP addresses to defeat defenses that limit the number of login attempts from a single IP address. Eventually they find a username for which the password is valid. Because every targeted username is subjected to a single login attempt, these attempts may not trigger the delay defense. Compromising random accounts this way is often good enough for an attacker. The compromised credentials can be sold on underground forums that trade in such information.

Non-cryptographic defenses are fairly effective at blocking these online attacks. However, a more devastating attack is much harder to block. We discuss this attack next.

18.3.1.2 Offline dictionary attacks

An attacker that compromises a login server can steal the password database stored at the server. This gives the attacker a large list of hashed passwords, one password for each user registered with that system. There are many other ways to obtain password files besides a direct compromise of a server. One study, for example, showed that used hard drives purchased on eBay can contain a lot of interesting, unerased data, including password files [68].

So, suppose an adversary manages to obtain a verification key $vk = H(pw)$ for some user. If the password pw is weak, and belongs to a small dictionary $\mathcal{D}$ of common passwords, then the adversary can mount an **offline dictionary attack**, by performing the following computation:

```
for each $w \in \mathcal{D}$:
    if $H(w) = vk$:
        output $w$ and halt
```
(18.1)

If pw belongs to $\mathcal{D}$, then using this procedure the adversary will obtain pw, or possibly some pw' with $H(pw) = H(pw')$.

The running time of this offline dictionary attack is $O(|\mathcal{D}|)$, assuming the time to evaluate H at one input counts as one time unit. This computation can be carried out entirely *offline*, with no interaction with the prover or the verifier.

Password statistics. In 2016, a password cracking service called `CrackStation` released a dictionary $\mathcal{D}$ of about 1.5 billion passwords. Empirical evidence suggests that a significant fraction of human generated passwords, close to 50%, are on this list. This means that after about 1.5 billion offline hashes, one in two passwords can be cracked. If the hash function H is SHA256 then this takes less than a minute on a modern GPU. There is only one conclusion to draw from this: simply hashing passwords using SHA256 is the wrong way to protect a password database.

As another way to illustrate the problem, observe that the total number of 8-character passwords containing only printable characters is about $95^8 \approx 2^{52}$ (using the 95 characters on a US keyboard). Running SHA256 on all words in this set using a modern array of GPUs can be done in a few days. This puts *all* passwords of 8 characters or less at risk in case of a server compromise.

Quantum offline password attacks. To make matters worse, the exhaustive search attack in the previous paragraph will be much faster once a large-scale quantum computer becomes available. We explained in Section 4.3.4 that a quantum computer can search a space of size n in time $\sqrt{n}$. Thus, a quantum search through the space of 8 character passwords will only take $\sqrt{2^{52}} = 2^{26}$ evaluations of the hash function. This takes a few seconds on a modern (classical) computer. Put differently, because 8 character passwords are insecure due to *classical* exhaustive search, 16 character passwords will be insecure once we have a quantum computer that is comparable in speed and size to a current classical computer. We discuss some defenses in Section 18.4.3.

18.3.1.3 Offline dictionary attacks with preprocessing

The offline dictionary attack discussed above can be made even better for the adversary by preprocessing the dictionary $\mathcal{D}$ before the attack begins. Then once a hashed password vk is obtained, the attacker will be able to quickly find the cleartext password pw. Specifically, we partition the dictionary attack into two phases: a **preprocessing phase** that is carried out before any hashed passwords are known, and an **attack phase** that cracks a given hashed password vk. Our goal is to minimize the time needed for the attack phase to crack a specific vk.

A simple dictionary attack with preprocessing works as follows:

```
Preprocessing phase:
    build a list $L$ of pairs $\big(pw, H(pw)\big)$, one pair for each $pw \in \mathcal{D}$
Attack phase on an input $vk$:                                              (18.2)
    if there is an entry $(pw, vk)$ in $L$, output $pw$
    otherwise, output fail
```

Let's assume that hashing a password using H counts as one time unit. Then the preprocessing phase takes $O(|\mathcal{D}|)$ time. If the list L is stored in a hash table that supports a constant time look up (such as Cuckoo hashing), then the attack phase is super fast, taking only constant time.

Batch offline dictionary attacks. Once the preprocessing phase is done, the attacker can use it to quickly crack many hashed passwords. Specifically, suppose an attacker obtains a large database F of hashed passwords from a compromised login server. Then cracking the hashed passwords in F using the dictionary $\mathcal{D}$ now takes only

$$\text{preprocessing time: } O(|\mathcal{D}|) \quad ; \quad \text{attack time: } O(|F|) \tag{18.3}$$

where $|F|$ is the number of hashed passwords in F. The total work of this batch dictionary attack is $O(|\mathcal{D}| + |F|)$. This is much faster than running a separate offline dictionary attack as in (18.1) against every element of F separately, which would take time $O(|\mathcal{D}| \times |F|)$.

Recall that the password statistics cited in Section 18.3.1.2 suggest that an adversary can find the passwords of about *half* the users in F using the `CrackStation` dictionary. This only takes time $O(|F|)$ once preprocessing is done. Effectively, this attack can expose millions of cracked passwords with very little work.

A time-space tradeoff. The simple preprocessing method presented in (18.2) requires the attacker to build and store a list L of all hashed dictionary words. When $\mathcal{D}$ is the set of all 2^{52} passwords of eight characters, the table L can be quite large and storing it can be difficult. In Section 18.7, we present a method called **rainbow tables** that quickly cracks passwords using a much smaller table L constructed during the preprocessing phase. For example, with $n := |\mathcal{D}|$ the method achieves the following parameters:

$$\begin{array}{l} \text{table size: } O(n^{2/3}) \quad ; \quad \text{preprocessing time: } O(n) \\ \text{attack time: } O(n^{2/3}). \end{array}$$

The table size is reduced from $O(n)$ to $O(n^{2/3})$, as promised. However, the time to attack one hashed password is increased from $O(1)$ to $O(n^{2/3})$. In other words, we traded a smaller table L for increased attack time. For this reason this approach is called a **time-space tradeoff**. We usually ignore the preprocessing time: it is a one-time cost invested before the attack even begins.

This time-space tradeoff further demonstrates why simply storing hashed passwords is the wrong thing to do. We discuss defenses against this in the next section.

id_1	$salt_1$	$H\big(pw_1, salt_1\big)$
id_2	$salt_2$	$H\big(pw_2, salt_2\big)$
id_3	$salt_3$	$H\big(pw_3, salt_3\big)$
$\vdots$	$\vdots$	$\vdots$

Figure 18.4: Password file (version 2)

18.4 Making dictionary attacks harder

Offline dictionary attacks, especially with preprocessing, are a real threat when storing hashes of weak passwords on a server. In this section we discuss a number of techniques that can make these attacks much harder for the adversary.

18.4.1 Public salts

In the previous section we showed how an attacker can preprocess the dictionary $\mathcal{D}$ to build a data structure L that then lets the attacker quickly crack one or more hashed passwords. A simple defense called **salting** can prevent these preprocessing attacks. Salting ensures that cracking a file F of hashed passwords takes time

$$\Omega\big(|\mathcal{D}| \times |F|\big)$$

even if the attacker is allowed infinite time to preprocess $\mathcal{D}$. In other words, salting ensures that the exhaustive search approach in (18.1) is the best possible attack.

Salting works by generating a random string, called a **salt**, when registering a new password. Every user in the system is assigned a fresh salt chosen at random from a set $\mathcal{S}$. As we will see, taking $|\mathcal{S}| = 2^{128}$ is sufficient in practice. This salt is hashed along with the password to derive the verification key vk. This salt must be stored in the password file in the clear, as shown in Fig. 18.4. Only the server needs to know the salt; the user is not aware that salts are being used.

Now, the modified password protocol, called **password protocol version 2**, runs as follows:

- G: set $pw \xleftarrow{\text{R}} \mathcal{P}$, $salt \xleftarrow{\text{R}} \mathcal{S}$, $y \leftarrow H(pw, salt)$,
  output $sk := pw$ and $vk := (salt, y)$.
- Algorithm P, on input $sk = pw$, and algorithm V, on input $vk = (salt, y)$, interact as follows:
  1. P sends pw to V;
  2. V outputs accept if the received pw satisfies $H(pw, salt) = y$;
     it outputs reject otherwise.

As in the description of version 1, the description of version 2 is rather idealized, in that passwords are chosen uniformly at random from a password space $\mathcal{P}$; in practice, this may not be the case.

With salts in place, the adversary has two strategies for attacking hashed passwords in a password file F:

- The first strategy is to adapt the batch offline dictionary attack. The problem is that the preprocessing phase must now be applied to a large list of possible inputs to H: any element in the set $\mathcal{D} \times \mathcal{S}$ is a possible input. Using the preprocessing algorithm in (18.2) this would require generating a data structure L of size $|\mathcal{D}| \times |\mathcal{S}|$ which is too large to generate, let alone store. Hence, the preprocessing approach of (18.2) is no longer feasible.

- The second strategy is to run an exhaustive password search as in (18.1) for every password in F. We already explained that this take time
$$O(|\mathcal{D}| \times |F|).$$

The salt space $\mathcal{S}$ needs to be sufficiently large so that the second strategy is always better than the first. This should hold even if the adversary uses a time-space tradeoff to preprocess $\mathcal{D} \times \mathcal{S}$. To derive the required bound on the size of $\mathcal{S}$, we first define more precisely what it means to invert a salted function in the preprocessing model.

Salted one-way functions with preprocessing. To define this properly we need to split the usual inversion adversary $\mathcal{A}$ into two separate adversaries $\mathcal{A}_0$ and $\mathcal{A}_1$. Adversary $\mathcal{A}_0$ has unbounded running time and implements the preprocessing phase. Adversary $\mathcal{A}_1$ is efficient and does the inversion attack. The only communication allowed between them is an exchange of an ℓ-bit string L that is the result of the preprocessing phase. This is captured in the following definition, which models H as a random oracle.

Definition 18.3. *Let H be a hash function defined over $(\mathcal{D} \times \mathcal{S},\ \mathcal{Y})$. We define the advantage* $\mathrm{OWsp}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, H]$ *of an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ in defeating the one-wayness of H in the preprocessing model as the probability of winning the following game:*

- *$\mathcal{A}_0$ issues queries to H and outputs an advice string L;*
- *the challenger chooses $(pw, s) \xleftarrow{\mathrm{R}} \mathcal{D} \times \mathcal{S}$, sets $y := H(pw, s)$, and sends (L, y, s) to $\mathcal{A}_1$;*
- *$\mathcal{A}_1$ issues queries to H and outputs $pw' \in \mathcal{D}$; it wins the game if $H(pw', s) = y$.*

Note that the adversary $\mathcal{A}_1$ is given both L and the salt s. It needs to find a pre-image of y with salt s. The following theorem gives a bound on the time to invert a salted function H in the preprocessing model, when H is modeled as a random oracle.

Theorem 18.2 ([54]). *Let H be a hash function defined over $(\mathcal{D} \times \mathcal{S},\ \mathcal{Y})$ where H is modeled as random oracle and where $|\mathcal{D}| \leq |\mathcal{Y}|$. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be an adversary as in Definition 18.3, where $\mathcal{A}_0$ outputs an ℓ-bit advice string L, and $\mathcal{A}_1$ makes at most Q_{ro} queries to H. Then*
$$\mathrm{OWsp}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, H] \leq O\left(\frac{\ell \cdot Q_{\mathrm{ro}}}{|\mathcal{S}| \cdot |\mathcal{D}|} + \frac{Q_{\mathrm{ro}}}{|\mathcal{D}|}\right). \tag{18.4}$$

The theorem shows that if $\mathcal{A}$ has constant success probability, say $1/2$, in inverting $vk := y \in \mathcal{Y}$, then the attack phase must take at least $Q_{\mathrm{ro}} \geq \Omega(|\mathcal{D}| \cdot |\mathcal{S}|/\ell)$ time. Therefore, to prevent any speed-up from preprocessing we should set $|\mathcal{S}| \geq \Omega(\ell)$. This will ensure that exhaustive search is the best attack. For example, if we assume maximum storage space of 2^{80} for the advice string L then the salt space $\mathcal{S}$ should be at least $\{0,1\}^{80}$. In practice one typically sets $\mathcal{S} := \{0,1\}^{128}$.

Technically, Theorem 18.2 bounds the time to crack a *single* password. It does not bound the time for a *batch* offline dictionary attack where the attacker tries to crack t passwords at once, for some $t > 1$. One expects, however, that the theorem can be generalized to the batch settings so that the bound $|\mathcal{S}| \geq \Omega(\ell)$ is sufficient to prevent any benefit from preprocessing even for cracking a batch of passwords.

id_1	$salt_1$	$H\big(password_1, salt_1, pepper_1\big)$
id_2	$salt_2$	$H\big(password_2, salt_2, pepper_2\big)$
id_3	$salt_3$	$H\big(password_3, salt_3, pepper_3\big)$
$\vdots$	$\vdots$	$\vdots$

Figure 18.5: Password file (version 3)

Limits of salting. While salts defend against preprocessing attacks, they do not prevent other attacks. For example, a user who chooses a weak password will still be vulnerable to the basic offline dictionary attack (18.1), even if a salt is used. In the next two sections we show how to provide further protection against offline dictionary attacks.

18.4.2 Secret salts

We can make the adversary's task harder by adding artificial entropy to human passwords. The idea is to pick a random short string, called a **secret salt**, or **pepper**, in a small space $\mathcal{S}_\text{p}$ and include it in the hash computation, but *not* include it in the password file. The resulting password file is shown in Fig. 18.5.

To verify a password, the server simply tries all possible values of the secret salt until it finds one that hashes to the stored hash value. The modified password protocol, **password protocol version 3**, is as follows:

- G: set $pw \stackrel{\text{R}}{\leftarrow} \mathcal{P}$, $salt \stackrel{\text{R}}{\leftarrow} \mathcal{S}$, $pepper \stackrel{\text{R}}{\leftarrow} \mathcal{S}_\text{p}$, $y \leftarrow H\big(pw, salt, pepper\big)$,
  output $sk := pw$ and $vk := (salt, y)$.
- Algorithm P, on input $sk = pw$, and algorithm V, on input $vk = (salt, y)$, interact as follows:
  1. P sends pw to V;
  2. V outputs accept if the received pw satisfies $H(pw, salt, p) = y$ for some $p \in \mathcal{S}_\text{p}$; it outputs reject otherwise.

A typical choice for the secret salt space is $\mathcal{S}_\text{p} := \{0,1\}^{12}$ which slows down password verification on the server by a factor of 4096 compared with protocol version 2. This still takes less than a hundredth of a second and is unnoticeable by the user. More importantly, the adversary has to do 4096 times more work to find weak passwords in the password file.

The secret salt makes an offline dictionary attack harder because now the adversary has to search through the space $\mathcal{D} \times \mathcal{S}_\text{p}$ instead of just $\mathcal{D}$. Yet this technique has minimal impact on the user experience. The secret salt increases the entropy of the user's password, without forcing the user to remember a more complicated password.

18.4.3 Slow hash functions

A different approach to protecting weak passwords is to use a slow hash function. Recall that hashing a password with a hash function such as SHA256 is fast. The speed of SHA256 is what makes an offline dictionary attack possible; the attacker can quickly evaluate the hash function on many dictionary words.

Suppose that the server hashes passwords using a *slow* hash function. Say, the function takes a hundredth of a second to evaluate on a single input, 10,000 times slower than SHA256. The user experience is hardly affected since users generally do not notice a hundredth of a second delay. However, the adversary's work to hash all words in the dictionary is increased by a factor of 10,000.

How do we build a slow hash function? The first idea that comes to mind is to iterate a fast hash function sufficiently many times until it becomes sufficiently slow. Specifically, for a hash function H defined over $(\mathcal{X}, \mathcal{X})$, define

$$H^{(d)}(x) := H(H(H(\cdots(x)\cdots))) \tag{18.5}$$

where H is iterated d times (see also Section 14.3). If $d = 10,000$ then evaluating $H^{(d)}(x)$ takes 10,000 times longer than evaluating $H(x)$. This approach, however, is problematic and should not be used. One reason is that the function $H^{(d)}$ is about d times easier to invert than H (see Exercise 14.18). We will see a better function below. First, let's define what we mean by a slow hash function.

Definition 18.4. *A **password-based key derivation function**, or **PBKDF**, is a function H that takes as input a password $pw \in \mathcal{P}$, a salt in $\mathcal{S}$, and a difficulty $d \in \mathbb{Z}^{>0}$. It outputs a value $y \in \mathcal{Y}$. We require that H is computable by an algorithm that runs in time proportional to d. As usual, we say that the PBKDF is defined over $(\mathcal{P}, \mathcal{S}, \mathcal{Y})$.*

We discuss the security requirements for a PBKDF in Exercise 18.3. Our first example PBKDF, called **PBKDF1**, is based on (18.5) and defined as:

$$\text{PBKDF1}_H(pw, \mathit{salt}, d) := H^{(d)}(pw, \mathit{salt}).$$

For a hash function H defined over $(\mathcal{X}, \mathcal{X})$, this PBKDF is defined over $(\mathcal{P}, \mathcal{S}, \mathcal{X})$, where $\mathcal{X} = \mathcal{P} \times \mathcal{S}$. It is not used in practice because of the attack discussed in Exercise 14.18.

18.4.3.1 The function PBKDF2

A widely used method to construct a slow hash function is called **PBKDF2**, which stands for password based key derivation function version 2. Let F be a PRF defined over $(\mathcal{P}, \mathcal{X}, \mathcal{X})$ where $\mathcal{X} := \{0,1\}^n$. The derived PBKDF, denoted PBKDF2_F, is defined over $(\mathcal{P}, \mathcal{X}, \mathcal{X})$ and works as follows:

$$\text{PBKDF2}_F(pw, \mathit{salt}, d) := \left\{ \begin{array}{l} x_0 \leftarrow F(pw,\ \mathit{salt}) \\ \text{for } i = 1, \ldots, d-1: \\ \qquad x_i \leftarrow F(pw, x_{i-1}) \\ \text{output } y \leftarrow x_0 \oplus x_1 \oplus \cdots \oplus x_{d-1} \in \mathcal{X} \end{array} \right\} \tag{18.6}$$

While (18.6) describes the basic PBKDF2, a simple extension outputs more bits if more are needed. In particular, PKBDF2 can output an element in $\mathcal{X}^b$ for some $1 < b < 2^{32}$ by computing:

$$\text{PBKDF2}_F^{(b)}(pw, \mathit{salt}, d) := \big(\ \text{PBKDF2}_F(pw, \mathit{salt}_1, d), \ldots, \text{PBKDF2}_F(pw, \mathit{salt}_b, d)\ \big) \in \mathcal{X}^b \tag{18.7}$$

where all b salts are derived from the provided salt by setting $\mathit{salt}_i \leftarrow \mathit{salt} \parallel \text{bin}(i)$. Here $\text{bin}(i)$ is the binary representation of $i \in \{1, \ldots, b\}$ as a 32-bit string.

```
input: $x_0 \in \mathcal{X}$,  difficulty $d \in \mathbb{Z}^{>0}$
1.  for $i = 1, \ldots, d$:    $x_i \leftarrow h(x_{i-1})$          // Then $x_i = h^{(i)}(x_0)$
2.  $y_0 \leftarrow x_d$
3.  for $i = 1, \ldots, d$:
4.          $j \leftarrow \text{int}(y_{i-1}) \bmod (d+1)$          // $\text{int}(y_{i-1})$ converts $y_{i-1} \in \mathcal{X}$ to an integer
5.          $y_i \leftarrow h(y_{i-1} \oplus x_j)$                  // read random location in the array $(x_0, \ldots, x_d)$
output $y_d \in \mathcal{X}$
```

Figure 18.6: The function $\text{Scrypt}_h(x_0, d)$

In practice, PBKDF2 is often implemented using HMAC-SHA256 as the underlying PRF. The difficultly d is set depending on the project needs and hardware speeds. For example, backup keybags in iOS 10 are protected using PBKDF2 with d set to ten million. In Windows 10, the data protection API (DPAPI) uses $d = 8000$ by default, but using HMAC-SHA512 as the PRF.

We discuss the security of PBKDF2 in more detail in Exercises 18.2 and 18.3.

18.4.4 Slow memory-hard hash functions

A significant problem with PBKDF2 is that it is vulnerable to parallel hardware attacks. To explain the problem, recall that the bulk of a modern processor is devoted to cache memory. The computing unit is a tiny fraction of the overall processor area. Consequently, a commercial processor cannot evaluate PBKDF2 on many inputs in parallel and is not well suited for an offline dictionary attacks.

A sophisticated attacker will usually run an offline dictionary attack on dedicated hardware that supports a high degree of parallelism, such as GPUs, FPGAs, or even custom chips. A single custom chip can pack over a million SHA256 engines. If each engine can do a million SHA256 evaluations per second, then the adversary can try 10^{12} passwords per second per chip. Even if the PBKDF2 difficulty is set to $d = 10,000$, a bank of about 500 such chips will run through all 2^{52} eight character passwords in less than a day. This attack is possible because the hardware implementation of SHA256 is relatively compact, making it possible to pack a large number of SHA256 engines into a single chip.

This suggests that instead of SHA256 we should use a hash function H whose hardware implementation requires a large amount of on-chip area. Then only a few copies of H can be packed into a single chip, greatly reducing the performance benefits of custom hardware.

How do we build a hash function H that has a large hardware footprint? One way is to ensure that evaluating H requires a lot of memory at every step of the computation. This forces the attacker to allocate most of the area on the chip to the memory needed for a single hash evaluation, which ensures that every chip can only contain a small number of hash engines.

Hash functions that require a lot of memory are called **memory-hard functions**. Several such functions have been proposed and shown to be provably memory-hard in the random oracle model. Before we discuss security let us first see a popular construction called **Scrypt**. Scrypt is built from a (memory-easy) hash function $h : \mathcal{X} \to \mathcal{X}$ where $\mathcal{X} := \{0,1\}^n$. The resulting function, denoted Scrypt_h, is shown in Fig. 18.6.

In our security analysis, we will treat the underlying hash function h as a random oracle. In practice, the function h is derived from the Salsa 20/8 permutation (Section 3.6). The difficulty d is set based on the performance needs of the system. For example, one could set d so that evaluating Scrypt fills the entire on-chip cache. This will ensure that evaluating Scrypt is not too slow, but needs a lot of memory.

Fig. 18.6 is a description of Scrypt as a hash function. The Scrypt PBKDF, defined over $(\mathcal{P}, \mathcal{X}, \mathcal{X})$, is built from the Scrypt hash and works as follows:

$$\text{ScryptPBKDF}_h(pw, salt, d) := \left\{ \begin{array}{l} x_0 \leftarrow \text{PBKDF2}_F(pw,\ salt,\ 1) \\ y \leftarrow \text{Scrypt}_h(x_0, d) \\ \text{output PBKDF2}_F(pw,\ y,\ 1) \end{array} \right\} \tag{18.8}$$

where F is a PRF defined over $(\mathcal{P}, \mathcal{X}, \mathcal{X})$. In practice one uses HMAC-SHA256 for F. If needed, Scrypt can be iterated several times to make it slower without increasing the required memory. Similarly, it can output an element in $\mathcal{X}^b$ for $b > 1$ by adjusting the application of PBKDF2 on the last line as in (18.7).

Is Scrypt memory-hard? The Scrypt function can be evaluated in time $O(d)$ by storing $(d+1)$ elements of $\mathcal{X}$. Step (1) in Fig. 18.6 creates an array $(x_0, \ldots, x_d)$ of size $(d+1)$. Then Step (5) repeatedly reads data from random locations in this array. Because of Step (5) it seems plausible that an algorithm that evaluates the function in time $O(d)$ must keep the entire array $(x_0, \ldots, x_d)$ in memory. Clearly this intuition needs a proof.

The danger is that a time-space tradeoff might enable one to evaluate Scrypt in a bit more time, but with far less memory. That would be devastating because the reduced memory would allow an attacker to pack many Scrypt engines into a single chip without paying much in running time per engine. This is exactly what we want to avoid.

In Exercise 18.6 we develop a simple time-space tradeoff on Scrypt. For any $1 < \alpha < d/2$ it shows that Scrypt can be evaluated in time $O(\alpha d)$ by storing only $\lceil d/\alpha \rceil$ elements of $\mathcal{X}$. In particular, Scrypt can be evaluated in time $O(d^2)$ using *constant* space. However, this type of time-space tradeoff does not help the adversary. It lets the adversary pack α times as many Scrypt engines into a single chip, but each engine must work α times harder. Therefore, the overall throughput of a single chip is unchanged compared to an implementation of Scrypt as in Fig. 18.6. Nevertheless, we need to prove that there is no better time-space tradeoff against Scrypt.

Pipelining is another threat to memory hardness. Suppose it were possible to evaluate Scrypt using an algorithm that uses $O(d)$ memory, but only in a few steps in the algorithm. If in the remaining time the algorithm used only constant space then it would be possible to share a single array of size $O(d)$ among multiple Scrypt engines arranged in a pipeline. Each engine would use the array in the few steps where it needs $O(d)$ memory, and then release the array for another engine to use. This again would enable the adversary to pack many Scrypt engines into a single chip, all sharing a single array of size $O(d)$. To prevent this form of pipelining we need to prove that every implementation of Scrypt that runs in time $O(d)$ must use $O(d)$ memory *in many steps* throughout the computation.

Scrypt is memory-hard. To prove that Scrypt is memory-hard we first define a security model that captures the hurdles discussed above and then state the security theorem for Scrypt. We begin by defining an abstract parallel random oracle model, where an algorithm $\mathcal{A}$ can query a random oracle $h : \mathcal{Y} \to \mathcal{Z}$ at multiple inputs in parallel.

A **parallel random oracle algorithm** $\mathcal{A}$ takes as input an $x \in \mathcal{X}$ and runs through a sequence of states. At each state the algorithm issues a set of queries to the random oracle h. The algorithm is given the responses to all its queries and it then moves to the next state. This process is repeated until the algorithm terminates, at which point the final state contains the output. We record all the intermediate states to keep track of their size.

Formally, the algorithm $\mathcal{A}$ implements a deterministic mapping:

$$\mathcal{A} : \mathcal{X} \times \mathcal{S} \times \mathcal{Z}^{\leq p} \to \mathcal{S} \times \mathcal{Y}^{\leq p}$$

for some positive integer p, and operates as follows:

- $\mathcal{A}$ is first invoked as $\mathcal{A}(x, \varepsilon, \varepsilon)$ and outputs a pair $\big(s_1, \bar{y}_1\big)$ in $\mathcal{S} \times \mathcal{Y}^{\leq p}$. Here s_1 is $\mathcal{A}$'s current state and $\bar{y} = (y_1, \ldots, y_r)$ is its first set of parallel queries to the random oracle $h : \mathcal{Y} \to \mathcal{Z}$.
- For $i = 1, \ldots, t$, when $\mathcal{A}$ outputs $(s_i, \bar{y}_i)$ with $\bar{y}_i = (y_1, \ldots, y_r) \in \mathcal{Y}^{\leq p}$, we do the following:
  - evaluate the oracle h in parallel by setting $\bar{z}_i \leftarrow \big(h(y_1), \ldots, h(y_r)\big)$, and
  - re-invoke $\mathcal{A}$ as $(s_{i+1}, \bar{y}_{i+1}) \leftarrow \mathcal{A}(x, s_i, \bar{z}_i)$.
- Eventually $\mathcal{A}$ outputs (s, ε) indicating that it is done and that the output is s.

The running time of $\mathcal{A}$ on input $x \in \mathcal{X}$ is the number of times that $\mathcal{A}$ is invoked until it terminates. Measuring running time this way captures the fact that a hardware implementation can evaluate the hash function h at many points in parallel.

We record the data given to $\mathcal{A}$ in step i as $st_i := (s_i, \bar{z}_i)$. We call st_i the input state at time i. For $s \in \mathcal{S}$ we let $|s|$ denote the length of s in bits, and similarly we let $|z|$ denote the length of $z \in \mathcal{Z}$. For $\bar{z} = (z_1, \ldots, z_r) \in \mathcal{Z}^{\leq p}$, we let $|\bar{z}| := \sum_{j=1}^{r} |z_i|$. When $\mathcal{Z} = \{0,1\}^n$ we have $|\bar{z}| = rn$. Finally, the bit length of an input state $st = (s, \bar{z})$ is defined as $|st| := |s| + |\bar{z}|$.

Definition 18.5. *Let $\mathcal{A}$ be a parallel random oracle algorithm taking inputs in $\mathcal{X}$. The cumulative memory complexity of $\mathcal{A}$ with respect to $h : \mathcal{Y} \to \mathcal{Z}$ and $x \in \mathcal{X}$, denoted* $\text{mem}[\mathcal{A}, h, x]$*, is defined as*

$$\text{mem}[\mathcal{A}, h, x] := \sum_{i=1}^{t} |st_i|.$$

The algorithm in Fig. 18.6 for computing $\text{Scrypt}_h(x, d)$ with respect to an oracle $h : \mathcal{X} \to \mathcal{X}$, where $\mathcal{X} = \{0,1\}^n$, has cumulative memory complexity of $O(nd^2)$. The following theorem shows that this is the best possible.

Theorem 18.3 ([6]). *Let $\mathcal{X} := \{0,1\}^n$ be such that $|\mathcal{X}|$ is super-poly and let d be chosen so that 2^{-d} is negligible. The for all parallel random oracle algorithms $\mathcal{A}$ and all $x \in \mathcal{X}$,*

$$\Pr\Big[\mathcal{A}(x, d) = \text{Scrypt}_h(x, d)\Big] \leq \Pr\Big[\text{mem}[\mathcal{A}, h, (x, d)] \geq \Omega(d^2 n)\Big] + \delta$$

for some negligible δ. Both probabilities are over the choice of random oracle $h : \mathcal{X} \to \mathcal{X}$.

The theorem shows that if $\mathcal{A}(x, d)$ outputs $\text{Scrypt}_h(x, d)$ with probability close to 1 then the cumulative memory complexity of $\mathcal{A}$ must be $\Omega(d^2 n)$ for almost all choices of h. This shows that there cannot be a time-space tradeoff against Scrypt that is significantly better than Exercise 18.6.

If an algorithm evaluates Scrypt with maximum space dn/α, for some $\alpha > 1$, then its running time must be $\Omega(d\alpha)$. Otherwise its cumulative memory complexity would violate the lower bound.

Similarly, there cannot be a pipelining attack on Scrypt. Any viable algorithm for computing Scrypt that runs in time $O(d)$ must use $\Omega(dn)$ memory throughout the algorithm. Otherwise, again, its cumulative memory complexity would violate the lower bound.

Technically, Theorem 18.3 bounds the time and space needed to evaluate Scrypt at a *single* input. It does not bound the time for a *batch* offline dictionary attack where the attacker tries to evaluate Scrypt at p passwords at once, for some $p > 1$. One expects, however, that the theorem can be generalized to the batch settings: if an algorithm $\mathcal{A}$ evaluates Scrypt correctly at p inputs with probability close to 1, then the cumulative memory complexity of $\mathcal{A}$ must be $\Omega(d^2np)$. This would show that there is no time-space tradeoff or pipelining attack against Scrypt when evaluating Scrypt at p points.

18.4.4.1 Password oblivious memory-hard functions

While Scrypt is a sound memory-hard password hashing function, it is vulnerable to a side-channel attack of the type discussed in Section 4.3.2.

Consider a login server where a running process P validates user passwords by hashing them with Scrypt. Suppose the adversary gains low-privilege access to this server; the adversary can run user-level programs on the server, but cannot compromise process P and cannot observe user passwords in the clear. However, using its foothold it can mount a clever attack, called a **cache timing attack**, that lets it learn the order in which P accesses pages in memory. It learns nothing about the contents of these pages, just the order in which they are read by P.

Now, suppose the adversary captures a hash value y which is the result of applying the Scrypt PBKDF in (18.8) to some password pw with a public salt. Normally the adversary would need to mount a dictionary attack where each attempt takes a large amount of time *and memory.* However, if the adversary also has the memory access pattern of process P as it was computing the Scrypt hash of pw, then the adversary can mount a dictionary attack on pw with very little memory.

To see how, look back at the implementation of Scrypt in Fig. 18.6. The very first time the algorithm executes Step (5) it reads cell number j from the array $(x_0, \ldots, x_d)$, where $j = \text{int}(y_0) \bmod (d+1)$. By observing P's accesses to memory, the adversary can see what memory page was read when Step (5) was first executed. This gives the adversary an approximate value j_{a} for j. The adversary does not learn the exact value of j because a single memory page may contain multiple array cells. Nevertheless, this j_{a} is sufficient to test a candidate password pw' with little memory. Here is how:

1. compute $x'_0 \leftarrow \text{PBKDF2}_F(pw',\ salt,\ 1)$ as in (18.8),
2. compute y'_0 as in Step (1) of Fig. 18.6, but without storing any intermediate values, and
3. test if $j' \leftarrow \text{int}(y'_0) \bmod (d+1)$ is close to j_{a}.

If the test fails then pw' is not the correct password. This procedure lets the adversary discard most candidate passwords in the dictionary with very little memory. Consequently, the user's password is again vulnerable to a hardware password attack.

A solution. This attack works because Scrypt's memory access pattern depends on the user's password. It would be better if we had a provably secure memory-hard hash function whose memory access pattern is independent of the user's password. It can still depend on the user's salt because the salt is not secret. Such functions are called **data-oblivious memory-hard functions**. **Argon2i-B** is an example of such a function. It is closely related to Scrypt, but the memory access pattern in its first part is independent of the password. This defeats the side-channel attack described above.

Slow hashing vs secret salts. To conclude this section we observe that both the secret salt method and the slow hashing method increase the adversary's work load. One should use one method or the other, but not both. The main benefit of the slow memory-hard hashing method is that it makes it difficult to mount a custom hardware attack. A secret salt used with a fast hash function does not prevent a parallel hardware attack. Consequently, slow memory-hard hash functions are preferable to secret salts.

18.4.5 More password management issues

The common password problem. Users frequently have accounts on multiple machines and at multiple web sites. Ideally, all of these servers take proper precaution to prevent an adversary from obtaining a password file, and also properly salt and hash passwords, to limit the damage should the adversary obtain this file. Unfortunately, the designers of low-security servers (e.g., a conference registration web site) may not take the same security precautions as are taken for high-security servers (e.g., a bank's web site). Such a low-security server may be easier to break in to. Moreover, such a low-security server may store hashes of passwords without salt, enabling a batch dictionary attack, which will retrieve all the weak passwords; even worse, such a server may store passwords in the clear, and the adversary retrieves *all* the passwords, even strong ones. Consequently, an adversary can break in to a low-security server and retrieve some, or even all, user ID/passwords at the server, and it is very likely that some of these passwords will also work at a high-security server. Thus, despite all the precautions taken at the high-security server, the security of that server can be compromised by the poor security of some completely unrelated, low-security server. This issue is known as the **common password problem**.

A standard solution to the common password problem is to install client-side software that converts a common password into unique site passwords — essentially "client-side salt." Let H be a hash function. When a user, whose login ID is id, types in a password pw that is to be sent to a server, whose identity is id_{server}, the user's machine (e.g. the user's web browser) automatically converts this password to $\widehat{pw} := H(pw, id, id_{\text{server}})$, and sends $\widehat{pw}$ to the server. Thus, from the server's point of view, the password is $\widehat{pw}$, although from the user's point of view, the password is still just pw. This technique will protect a user from servers that do not properly salt and hash passwords, even if that user uses the same password on many servers.

Biometrics. The biggest difficultly with password-based authentication is that users tend to forget their passwords. A large fraction of all support calls have to do with password related problems. As a result, several deployed systems attempt to replace passwords by human biometrics, such as fingerprints, retina scans, facial recognition, and many others. One can even use keystroke dynamics, namely the length of time between keystrokes and the length of time a key is pressed, as a biometric [115]. The idea is to use (features of) the biometric as the user's password.

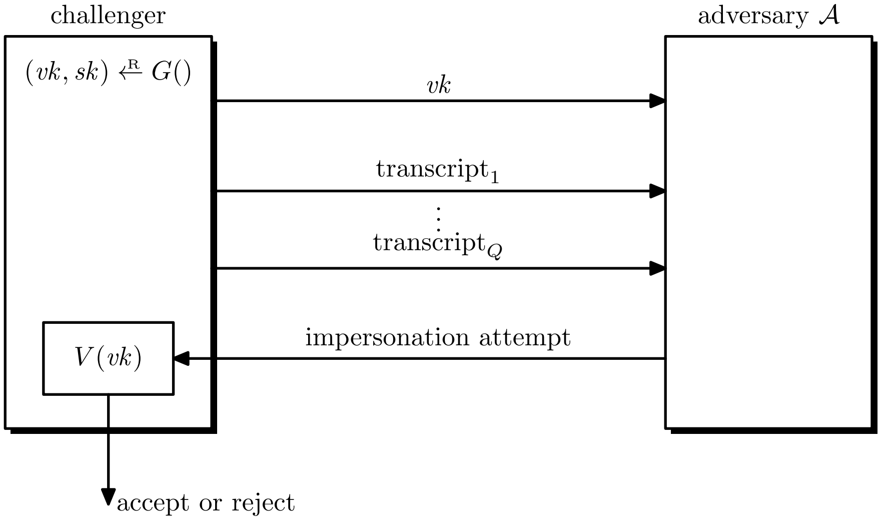


Figure 18.7: Attack Game 18.2

While biometrics offer clear benefits over passwords (e.g., the user cannot forget his fingerprint) they have two significant disadvantages:

- biometrics are not generally secret — people leave their fingerprints on almost anything they touch, and
- unlike passwords, biometrics are irrevocable — once a biometric is stolen the user has no recourse.

Consequently, biometrics should not be used as the only means of identifying users. Biometrics can be used as additional identification (sometimes referred to as *second-factor authentication*) for increased security.

18.5 One time passwords: security against eavesdropping

The password protocols in the previous section are easily compromised if an adversary can eavesdrop on a single interaction between the prover and verifier. Our goal for this section is to develop ID protocols secure against eavesdropping. We start by defining security for ID protocols in the presence of an eavesdropper. We enhance Attack Game 18.1 by introducing a new, “eavesdropping phase” in which the adversary is allowed to request a number of transcripts of the interaction between the real prover and the real verifier. The updated game is shown in Fig. 18.7.

Attack Game 18.2 (Secure identification: eavesdropping attack). For a given identification protocol $\mathcal{I} = (G, P, V)$ and a given adversary $\mathcal{A}$, the attack game runs as follows:

- *Key generation phase.* The challenger runs $(vk, sk) \xleftarrow{R} G()$, and sends vk to $\mathcal{A}$.
- *Eavesdropping phase.* The adversary requests some number, say Q, of transcripts of conversations between P and V. The challenger complies by running the interaction between P and

V a total of Q times, each time with P initialized with input sk and V initialized with vk. The challenger sends these transcripts $T_1, \ldots, T_Q$ to the adversary.

- *Impersonation attempt.* As in Attack Game 18.1: the challenger and $\mathcal{A}$ interact, with the challenger following the verifier's algorithm V (with input vk), and with $\mathcal{A}$ playing the role of a prover, but not necessarily following the prover's algorithm P.

We say that the adversary wins the game if V outputs accept at the end of the interaction. We define $\mathcal{A}$'s advantage with respect to $\mathcal{I}$, denoted ID2adv$[\mathcal{A}, \mathcal{I}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 18.6. *We say that an identification protocol $\mathcal{I}$ is* ***secure against eavesdropping attacks*** *if for all efficient adversaries $\mathcal{A}$, the quantity* ID2adv$[\mathcal{A}, \mathcal{I}]$ *is negligible.*

Keeping *vk* secret. The adversary in Attack Game 18.2 is given the verification key vk, meaning that vk can be treated as public information. However, the first eavesdropping-secure ID protocol we present requires the verifier to keep vk secret. This motivates a weaker version of Attack Game 18.2 where the challenger does not send vk to the adversary. A small complication when vk is kept secret is that we must now allow the adversary to make multiple impersonation attempts. One may insist that these impersonation attempts proceed sequentially, or allow them to proceed concurrently. In this chapter, we shall insist that they proceed sequentially. The adversary wins the game if at least one of its impersonation attempts is accepted by the verifier.

The reason we need to allow multiple impersonation attempts is that now, when vk is secret, interactions with the verifier could potentially leak some information about vk. This stronger definition of security rules out some trivially insecure protocols, as discussed in Exercise 18.10. We note that multiple attempts were not necessary in Attack Game 18.2 where vk is public, since the adversary could emulate the verifier itself.

Other than these two changes, the remainder of Attack Game 18.2 is unchanged. We let wID2adv$[\mathcal{A}, \mathcal{I}]$ denote the adversary's advantage in winning this weaker version of Attack Game 18.2. ID protocols secure in these settings are said to be **weakly secure**.

Definition 18.7. *We say that an identification protocol $\mathcal{I}$ is* ***weakly secure against eavesdropping attacks*** *if for all efficient adversaries $\mathcal{A}$, the quantity* wID2adv$[\mathcal{A}, \mathcal{I}]$ *is negligible.*

Stateful protocols. The password protocols in the previous section were all stateless — the verifier and prover did not maintain state between different invocations of the protocol. In this section, however, both protocols we present are stateful.

In a stateful protocol, after each invocation of the protocol the pair (vk, sk) changes: the prover P updates sk and the verifier V updates vk. However, we shall assume that V only updates vk if it outputs accept.

We now consider how to modify Attack Game 18.2 to deal with stateful protocols. As before, we allow the adversary to eavesdrop on several conversations between P and V. Also, we allow the adversary to make several impersonation attempts (although, if vk is not kept secret, then it suffices to just consider a single impersonation attempt). But there is another wrinkle. In the stateless case, we could assume without loss of generality that the adversary obtained all of the transcripts before making any impersonation attempts. However, with stateful protocols, this is no longer the case, and we have to allow the adversary to interleave eavesdropping and impersonation attempts. That is, the attack game proceeds in rounds. In each round the adversary can choose to either

- *eavesdrop:* obtain a transcript between P and V, after which P updates sk and V updates vk, or
- *impersonate:* make an impersonation attempt, interacting with V.

Furthermore, we also assume that the attack game ends as soon as one of the impersonation attempts succeeds (in which case the adversary wins the game). Recall that we are assuming that V does not update vk on a failed impersonation attempt, which ensures that in the eavesdropping rounds, P and V remain properly synchronized.

18.5.1 PRF-based one-time passwords: HOTP and TOTP

The simplest ID protocols secure against eavesdropping attacks are called **one-time password** protocols. These are similar to the basic password protocol of Section 18.3, except that the password changes after every invocation of the protocol.

We begin by describing a weakly secure protocol called **HOTP**, which stands for hash-based one-time password. Let F be a PRF defined over $(\mathcal{K}, \mathbb{Z}_N, \mathcal{Y})$ for some large integer N, say $N = 2^{128}$. This F is used to update the password after every successful invocation. The HOTP protocol $\texttt{HOTP} = (G, P, V)$ works as follows:

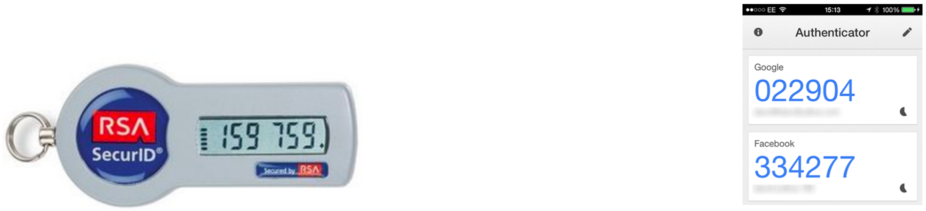
- G: choose a random $k \xleftarrow{\text{R}} \mathcal{K}$ and output $sk := (k, 0)$ and $vk := (k, 0)$.
- Algorithm P given sk, and algorithm V given vk, interact as follows:
  1. $P\big(sk = (k, i)\big)$: send $r := F(k, i)$ to V and set $sk \leftarrow \big(k,\ i + 1\big)$,
  2. $V\big(vk = (k, i)\big)$: if the received r from P satisfies $r = F(k, i)$ output accept and set $vk \leftarrow (k,\ i + 1)$. Otherwise, output reject.

Here both vk and sk must be kept secret, and therefore HOTP is only *weakly* secure against eavesdropping. Note that the integer N is chosen to be so large that, in practice, the counter i will never wrap around. Implementations of HOTP typically use HMAC-SHA256 as the underlying PRF, where the output is truncated to the desired size, typically six decimal digits, as shown in Fig. 18.8.

Theorem 18.4. *Let F be a secure PRF defined over $(\mathcal{K}, \mathbb{Z}_N, \mathcal{Y})$, where N and $|\mathcal{Y}|$ are both super-poly. Then the ID protocol $\texttt{HOTP}$ is weakly secure against eavesdropping.*

Proof sketch. Since F is a secure PRF, the adversary cannot distinguish between a challenger who uses the PRF F in Attack Game 18.2 and a challenger who uses a random function $f : \mathbb{Z}_N \to \mathcal{Y}$. Moreover, when the challenger uses a random function f, an impersonation attempt succeeds with probability at most $1/|\mathcal{Y}|$, which is negligible, since $|\mathcal{Y}|$ is super-poly. Moreover, since N is large, the counter values will not "wrap around" in any feasible attack. □

HOTP can be used in a car key fob system to wirelessly unlock a car, as discussed at the beginning of the chapter. The secret PRF key k is stored on the key fob and at the car. Every time the user presses a button on the key fob, the key fob increments the internal counter i by one, and sends the derived one-time password to the car, along with the counter i. The car maintains



(a) RSA SecurID token

(b) Google authenticator

Figure 18.8: One-time password implementations

its own counter and verifies the received one-time password and counter value. Note that the car must ensure that the recieved counter value is greater than the car's current counter value.

HOTP can also be used to authenticate a human user to a remote web server. The user is given a security token that looks something like the token in Fig. 18.8a and displays a 6-digit one-time password. The user authenticates to the remote server by typing this password into her web browser. The one-time password is then sent to the remote server to be validated. The next time the user wants to authenticate to the server she first presses a button on the token to increment the counter i by one. This advances the token to the next one-time password and updates the 6-digit value displayed on the screen.

HOTP systems are problematic for a number of reasons. First, in the remote web server settings we want to minimize the number of characters that the user needs to enter. In particular, we do not want to require the user to type in the current counter value in addition to the 6-digit password. Yet, the counter value is needed to synchronize the token and the remote server in case they go out of sync. It would be better if we could use an implicit counter that is known to both sides. The current time could serve as an implicit counter, as discussed below.

Second, there is a security problem. In HOTP the one-time password is only updated when the user initiates the protocol. If the user authenticates infrequently, say once a month, then every one-time password will be valid for an entire month. An attacker who somehow obtains the user's current one-time password, can sell it to anyone who wants to impersonate the user. The buyer can use the purchased password at anytime, as long as it is done before the next time the user authenticates to the server.

18.5.1.1 Time-based one-time passwords

A better one-time password scheme is called **time-based one-time passwords**, or **TOTP**. In TOTP the counter i is incremented by one every 30 seconds, whether the user authenticates or not. This means that every one-time password is only valid for a short time. When using a hardware token as in Fig. 18.8a, the display changes every 30 seconds to present the latest one-time password to the user. There is no button on the token.

Whenever the user authenticates to the remote server, the server uses the current time to determine the value of the counter i. It then verifies that the correct $r := F(k, i)$ was supplied by the user. To account for clock skew between the server and the token, the server will accept any of $\{F(k, (i - c)), \dots, F(k, (i + c))\}$ as valid passwords, for a small value of c such as $c = 5$. Within the $2c + 1$ clock-skew window, the server prevents replay attacks by rejecting passwords that have been used before.

Fig. 18.8a is a hardware token implementation of TOTP. The token is loaded with a secret PRF key at token setup time and uses that key to derive the 6-digit one-time passwords. The server has the same PRF key. The hardware token has an internal battery that can power the device for several years. When the battery runs out the token is dead.

Fig. 18.8b is a TOTP implemented as an app on a modern phone. The user loads the secret PRF key into the app by typing it in or by scanning a QR code. The app manages the user's one-time password with multiple systems, as shown in the figure, where the app manages one-time passwords for Google and Facebook.

18.5.2 The S/key system

TOTP requires that the verification key vk stored on the server remains secret. If an adversary steals vk without being detected then all security is lost. This actually happened in a number of well publicized cases.

The next system, called S/key, removes this limitation. The system, however, can only be used a bounded number of times before the pair (vk, sk) must be regenerated. We let n be a preset poly-bounded number, say $n = 10^6$, that indicates the maximum number of times that a (vk, sk) pair can be used.

In Section 14.3 we defined the concept of a hash chain, which will be used here too. To review, let $H : \mathcal{X} \to \mathcal{X}$ be a function. For $j \in \mathbb{Z}^{>0}$ we use $H^{(j)}(x)$ to denote the **jth iterate of** H, namely $H^{(j)}(x) := H(H(H(\cdots(x)\cdots)))$ where H is repeated j times. We let $H^{(0)}(x) := x$.

The S/key protocol. The protocol $\texttt{Skey}_n = (G, P, V)$, designed for n invocations, works as follows:

- G: choose a random $k \xleftarrow{R} \mathcal{X}$. Output $sk := (k, n)$ and $vk := H^{(n+1)}(k)$,
- Algorithm P given sk, and algorithm V given vk, interact as follows:
  1. $P\big(sk = (k, i)\big)$: send $t := H^{(i)}(k)$ to V and set $sk \leftarrow (k,\ i - 1)$,
  2. $V(vk)$: if the received t from P satisfies $vk = H(t)$ output accept and set $vk \leftarrow t$. Otherwise, output reject.

The protocol is illustrated in Fig. 18.9. In the first invocation the prover sends to the verifier the password $H^{(n)}(k)$. In the second invocation the prover sends the password $H^{(n-1)}(k)$, and so on. Each password is only used once. Clearly after n invocations, the prover runs out of one time passwords, at which point the prover can no longer authenticate to the verifier, and a new (vk, sk) pair must be generated.

Security. We show that S/key remains secure even if vk is made public. Hence, S/key is fully secure against eavesdropping, while HOTP is only weakly secure.

The analysis of S/key requires that $H : \mathcal{X} \to \mathcal{X}$ be a one-way function on n iterates as in Definition 14.5. To review, this means that for all $j = 1, \ldots, n$, given $y \leftarrow H^{(j)}(k)$ as input, where $k \xleftarrow{R} \mathcal{X}$, it is hard to find an element in $H^{-1}(y)$. Recall that Exercise 14.16 shows that a one-way function H need not be one-way on n iterates, even when $n = 2$. Nevertheless, standard cryptographic functions such as SHA256 are believed to be one-way on n-iterates for reasonable values of n, say $n \leq 10^6$.

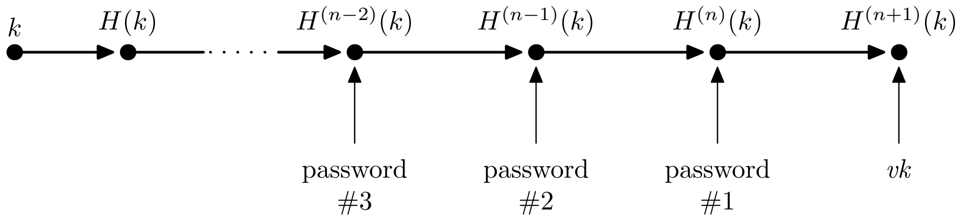


Figure 18.9: The S/key protocol

Theorem 18.5. *Let $H : \mathcal{X} \to \mathcal{X}$ be a one-way function on n iterates. Then the ID protocol Skey_n is secure against eavesdropping.*

Proof sketch. Since vk is public, we can assume that the adversary eavesdrops on, say, Q conversations, and then makes a single impersonation attempt. We do not know in advance what Q will be, but we can guess. We request $y \leftarrow H^{(n-Q+1)}(k)$ from the iterated one-way challenger and use y to generate Q valid conversations with respect to the initial verification key $vk = H^{(n+1)}(k)$. If our guess for Q is correct, and the adversary succeeds in its impersonation attempt, the adversary will find for us a pre-image of y. Thus, if the adversary impersonates with probability ϵ, we win Attack Game 14.1 with probability ϵ/n. □

Remark 18.1. To defend against preprocessing attacks on H, of the type discussed in Section 18.7, algorithm G could choose a public salt at setup time and prepend this salt to the input on every application of H. Moreover, to avoid the attack of Exercise 14.18 it is recommended to use a different hash function at every step in the chain. This has been analyzed in [100]. □

The trouble with S/key. In every authentication attempt, the prover P must send to V an element $t \in \mathcal{X}$. For H to be one-way, the set $\mathcal{X}$ must be large and therefore t cannot be a 6-digit number as in the TOTP system. In practice, t needs to be at least 128 bits to ensure that H is one-way. This makes it inconvenient to use S/key as a one-time password scheme where the user needs to type in a password. Encoding a 128-bit t as printable characters requires at least 22 characters.

18.6 Challenge-response: security against active attacks

We now consider a more powerful attack in which the adversary actively impersonates a legitimate verifier. For example, the adversary may clone a banking site and wait for a user (i.e., prover) to visit the site and run the ID protocol with the adversary. As a result, the adversary gets to repeatedly interact with the prover and send the prover arbitrary messages of its choice. The adversary's goal is to gain information about the prover's key sk. After several such interactions, the adversary turns around and attempts to authenticate as the prover to a legitimate verifier. We say that the ID protocol is secure against active attacks if the adversary still cannot fool the verifier.

The one-time password protocols HOTP and Skey in Section 18.5 are clearly insecure against active attacks. By impersonating a verifier, the adversary will learn a fresh one-time password

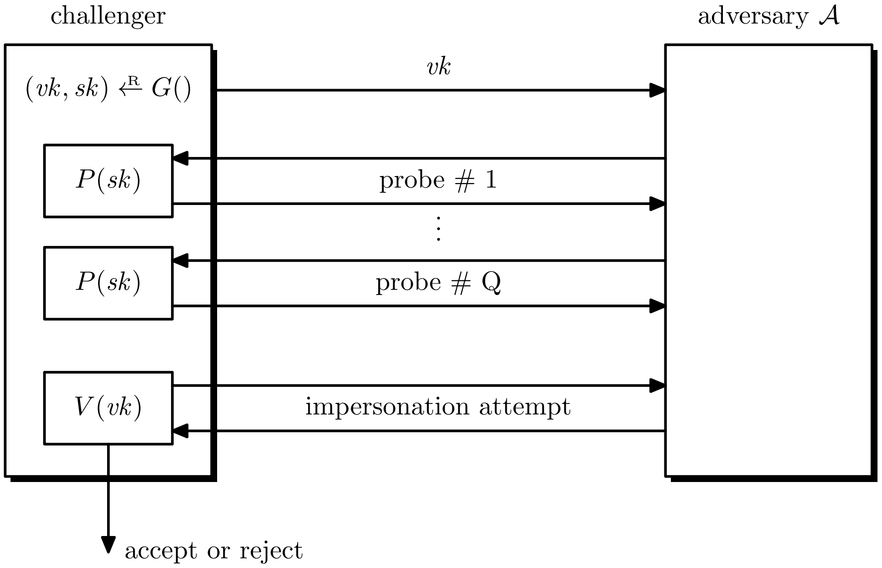


Figure 18.10: An example active attack as in Attack Game 18.3

from the prover that the adversary can then use to authenticate to the verifier. In fact, a moments reflection shows that no single flow protocol is secure against active attacks.

We first define active attacks and then construct a simple two flow protocol that is secure against active attacks. For simplicity, in this section we only consider protocols where both the prover and verifier are stateless.

Attack Game 18.3 (Secure identification: active attacks). For a given identification protocol $\mathcal{I} = (G, P, V)$ and a given adversary $\mathcal{A}$, the attack game, shown in Fig. 18.10, runs as follows:

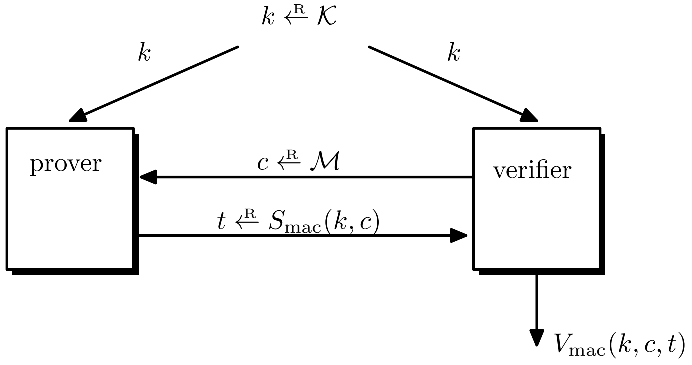
- *Key generation phase.* The challenger runs $(vk, sk) \xleftarrow{\text{R}} G()$, and sends vk to $\mathcal{A}$.

- *Active probing phase.* The adversary requests to interact with the prover. The challenger complies by interacting with the adversary in an ID protocol with the challenger playing the role of the prover by running algorithm P initialized with sk. The adversary plays the role of verifier, but not necessarily following the verifier's algorithm V. The adversary may interact concurrently with many instances of the prover — these interactions may be arbitrarily interleaved with one another.

- *Impersonation attempt.* As in Attack Game 18.1: the challenger and $\mathcal{A}$ interact, with the challenger following the verifier's algorithm V (with input vk), and with $\mathcal{A}$ playing the role of a prover, but not necessarily following the prover's algorithm P.

We say that the adversary wins the game if the verification protocol V outputs accept at the end of the interaction. We define $\mathcal{A}$'s advantage with respect to $\mathcal{I}$, denoted $\text{ID3adv}[\mathcal{A}, \mathcal{I}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 18.8. *We say that an identification protocol $\mathcal{I}$ is secure against active attacks if for all efficient adversaries $\mathcal{A}$, the quantity* $\text{ID3adv}[\mathcal{A}, \mathcal{I}]$ *is negligible.*



$k \xleftarrow{R} \mathcal{K}$
k
k
prover
verifier
$c \xleftarrow{R} \mathcal{M}$
$t \xleftarrow{R} S_{\text{mac}}(k, c)$
$V_{\text{mac}}(k, c, t)$

Figure 18.11: MAC based Challenge-Response identification

Concurrent vs sequential attacks. Note that in the active probing phase of the attack game, we allow the adversary to interact concurrently with many instances of the prover. One could consider a weaker attack model in which these interactions must be run sequentially, as shown in Fig. 18.10. However, all of the protocols we consider achieve security in this stronger, concurrent attack model.

Keeping *vk* secret. Some protocols that satisfy Definition 18.8 do not require the verifier to keep any secrets. However, one of the protocols we present in this section does require *vk* to be secret. This motivates a weaker version of Attack Game 18.3 where the challenger does not send *vk* to the adversary. Just as in Section 18.5, if *vk* is kept secret, then we must now allow the adversary to interact with the verifier, since such interactions could potentially leak information about *vk*. Therefore, in the active probing phase, we allow the adversary to interact concurrently with multiple instances of both the prover and the verifier. When interacting with an instance of the verifier, the adversary learns if the verifier outputs accept or reject. In addition, during the impersonation attempt, we let the adversary interact concurrently with several verifiers, and the adversary wins the game if at least one of these verifiers accepts.

We let $\text{wID3adv}[\mathcal{A}, \mathcal{I}]$ denote the adversary's advantage in winning this weaker version of Attack Game 18.3. ID protocols secure in these settings are said to be **weakly secure**.

Definition 18.9. *We say that an identification protocol $\mathcal{I}$ is **weakly secure against active attacks** if for all efficient adversaries $\mathcal{A}$, the quantity* $\text{wID3adv}[\mathcal{A}, \mathcal{I}]$ *is negligible.*

18.6.1 Challenge-response protocols

We present two (stateless) ID protocols, called **challenge-response**, that are secure against active attacks. The first protocol is only weakly secure, meaning that the verifier must keep the key *vk* secret. The second protocol is secure even if *vk* is public.

Let $\mathcal{I} = (S_{\text{mac}}, V_{\text{mac}})$ be a MAC defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. The challenge-response protocol $\texttt{ChalResp}_{\text{mac}} = (G, P, V)$, shown in Fig. 18.11, works as follows:

- G: choose a random $k \xleftarrow{R} \mathcal{K}$, and output $sk := k$ and $vk := k$.

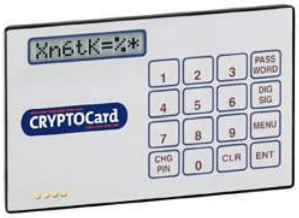


Figure 18.12: CRYPTOCard RB-1 token

- Algorithm P with input $sk = k$, and algorithm V with input $vk = k$, interact as follows:
  1. V chooses a random $c \overset{R}{\leftarrow} \mathcal{M}$, and sends c to P;
  2. P computes $t \overset{R}{\leftarrow} S_{\text{mac}}(k, c)$, and sends t to V;
  3. V outputs $V_{\text{mac}}(k, c, t)$.

The random c is called the **challenge** while t is called the **response**. Clearly vk must be kept secret for the protocol to be secure.

Theorem 18.6. *Suppose $\mathcal{I}$ is a secure MAC system, and that the size of the message space, $|\mathcal{M}|$, is super-poly. Then ID protocol $\texttt{ChalResp}_{\text{mac}}$ is weakly secure against active attacks.*

Proof sketch. The assumption that $|\mathcal{M}|$ is super-poly implies that in each impersonation attempt, the probability that the adversary receives a challenge message that it has seen before (in a previous interaction with the prover) is negligible. So either that unlikely event happens, or the adversary breaks the MAC system (in the sense of Attack Game 6.2). □

Case study: CRYPTOCard. Fig. 18.12 gives an example of a Challenge-Response token. When a user logs in to a server using his computer terminal, the server sends to the user an eight character challenge, which appears on his computer terminal screen. The user enters this challenge into the token using the keypad on the token. The token computes the response and displays this on its screen. The user then types this response into his computer terminal keyboard, and this is sent to the server to complete the protocol. The MAC is implemented as a PRF derived from either 3DES or AES.

Challenge-response using passwords. In describing protocol $\texttt{ChalResp}_{\text{mac}}$, the key k was chosen at random from the key space $\mathcal{K}$ of the underlying MAC system. In some settings it may be convenient to deploy this protocol where the key k is derived from a user generated password pw as $k \leftarrow H(pw)$ where H is a key derivation function as in Section 8.10.

This can be quite dangerous. If pw is a weak password, belonging to some relatively small dictionary $\mathcal{D}$ of common passwords, then this protocol is vulnerable to a simple offline dictionary attack. After eavesdropping on a single conversation (c, t) between prover and verifier, the adversary does the following:

```
for each $w \in \mathcal{D}$ do
    if $V_{\text{mac}}(H(w), c, t) = \mathsf{accept}$ then
        output $w$ and halt
```

In all likelihood, the output will be the password pw.

18.6.1.1 Challenge response with a public *vk*

The protocol in Fig. 18.11 is easily converted into a protocol where vk can be public. We need only replace the MAC with a signature scheme $(G, S_{\text{sig}}, V_{\text{sig}})$ defined over $(\mathcal{M}, \mathcal{T})$. The main change to Fig. 18.11 is that the prover responds to the challenge using algorithm S_{sig} and the secret signing key. The verifier checks the response using algorithm V_{sig} and the public verification key. We refer to the resulting protocol as $\texttt{ChalResp}_{\text{sig}}$.

Theorem 18.7. *Assume $\mathcal{S}$ is a secure signature scheme, and that the size of the message space, $|\mathcal{M}|$, is super-poly. Then $\texttt{ChalResp}_{\text{sig}}$ is secure against active attacks.*

Proof sketch. The idea is essentially the same as for that of Theorem 18.6, except that now, the adversary must forge a signature, rather than a MAC. □

The signature-based Challenge-Response protocol has an obvious security advantage over the MAC-based protocol, since vk need not be kept secret. However, the MAC-based protocol has the advantage that the response message can be short, which is crucial for CRYPTOCard-like applications where a person must type both the challenge and the response on a keyboard. Recall that in CRYPTOCard the response is only 48 bits long. A digital signature scheme cannot have such short signatures and still be secure. Exercise 18.13 explores a different challenge-response protocol where the response message can be short.

18.7 A fun application: rainbow tables

Let $h : \mathcal{P} \to \mathcal{Y}$ be a random function and set $N := |\mathcal{P}|$. We look at the general problem of inverting h. We will assume that $|\mathcal{Y}| \geq N$ since that is the typical situation in practice. For example, $\mathcal{P}$ might be the set of all eight character passwords while $\mathcal{Y} = \{0,1\}^{256}$.

Let $pw \xleftarrow{\text{R}} \mathcal{P}$ and let $y \leftarrow h(pw)$. Clearly an exhaustive search over all of $\mathcal{P}$ will find a preimage of y after at most N queries to h. In this section we develop a much faster algorithm to invert h using a method called **rainbow tables**. The inversion algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ proceeds in two phases:

- Preprocessing phase: algorithm $\mathcal{A}_0$ interrogates h and outputs a table L containing ℓ pairs in $\mathcal{P}^2$, for some ℓ. This preprocessing phase takes time $O(N)$, but it is done offline before the challenge y is known. The resulting table L, called a rainbow table, must be stored somewhere to be used in the second phase.

- Attack phase: once a challenge $y \in \mathcal{Y}$ is provided, algorithm $\mathcal{A}_1$ is invoked as $\mathcal{A}_1(L, y)$ and uses L to quickly find an inverse of y. It successfully outputs a preimage pw' in $h^{-1}(y)$ with probability close to 1.

Let t be the running time of the attack phase $\mathcal{A}_1$. We will show how to invert h in time t where

$$t \times \ell^2 \approx N^2. \tag{18.9}$$

For example, if we can store a table L of size $\ell = N^{2/3}$ then we can invert h in time $t \approx N^{2/3}$ with probability close to 1. This is much faster than exhaustive search over $\mathcal{P}$.

Equation (18.9) is called a **time-space tradeoff**. The more space we have for the table L, the faster we can invert h. Of course, once we have the table L, we can use it to quickly find the inverse of many elements in $\mathcal{Y}$.

Rainbow tables are commonly used to crack unsalted passwords, as discussed in Section 18.3.1.3. They can also be used to recover the secret key k in a block cipher (E, D) from a known plaintext-ciphertext pair $\big(m,\ c = E(k, m)\big)$. This is because the key k is the inverse of the function $h(k) := E(k, m)$ at the point c. If m is sufficiently long, or if multiple plaintext-ciphertexts pairs are provided, then the inverse k is unique. Applying this to AES-128 we see that a table L of size $128 \times (2^{128})^{(2/3)} \approx 128 \times 2^{85}$ bits (about a billion exabytes) can be used to break AES in time 2^{85}. This may be too much today, but could become feasible in a few decades. We discussed this threat in Section 4.2.2.1. It is partially the reason for the shift towards AES-256. Note, however, that building the table L requires significant (one-time) work; about 2^{128} evaluation of AES-128.

A careful reader will notice that the bound (18.9) is quite poor at the boundary $\ell = 1$, where it gives $t \approx N^2$. This is much worse than simple exhaustive search that only takes time N. It shows that the rainbow table algorithm is not tight for some values of ℓ. Improving the time-space tradeoff (18.9) is a long-standing open problem (see Exercise 18.7).

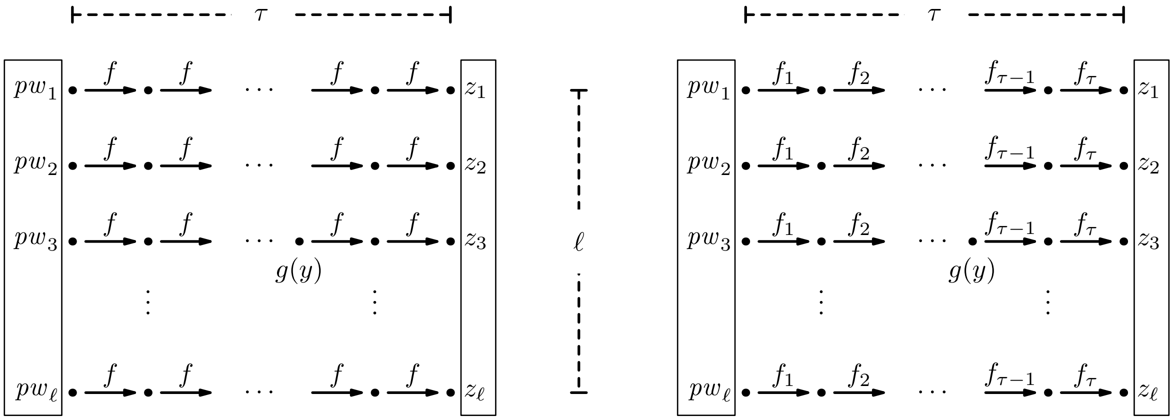
Hellman's basic time-space tradeoff. The first time-space tradeoff for inverting a random function was invented by Hellman as a criticism of the short DES key size (56-bits). Hellman's method uses an efficiently computable auxiliary function $g : \mathcal{Y} \to \mathcal{P}$ called a **reduction function**. It "reduces" an output of h in $\mathcal{Y}$ to an element of $\mathcal{P}$. For simplicity, we will assume that g is also a random function. Then the function $f(pw) := g(h(pw))$ maps $\mathcal{P}$ to itself.

The preprocessing algorithm $\mathcal{A}_0$ uses the function $f : \mathcal{P} \to \mathcal{P}$. It is parameterized by two positive constants τ and ℓ. Recall that for $\tau > 0$ the function $f^{(\tau)}$ is the τ-th iterate of f as defined in (18.5). Algorithm $\mathcal{A}_0$ works as follows, and is shown visually in Fig. 18.13a:

```
Algorithm $\mathcal{A}_0$: (preprocess $h$)
    for $i = 1, \ldots, \ell$:
        $pw_i \xleftarrow{R} \mathcal{P}$
        $z_i \leftarrow f^{(\tau)}(pw_i) \in \mathcal{P}$                    // $\textit{run through } \tau \textit{ evaluations of } f$
    output $L := \big\{(pw_1, z_1), \ldots, (pw_\ell, z_\ell)\big\} \subseteq \mathcal{P}^2$        // $\textit{output } \ell \textit{ pairs in } \mathcal{P}^2$
```

Algorithm $\mathcal{A}_0$ builds ℓ horizontal chains as shown in Fig. 18.13a. For each chain it records the starting and ending points in the table L. Its running time is proportional to $\tau \times \ell$.

Next, to invert an element $y \in \mathcal{Y}$ using L we repeatedly apply f to $g(y)$ until we hit the right edge of Fig. 18.13a. We then use L to jump to the starting point of the relevant chain and traverse it until we find a preimage of y. More precisely, to invert y do:



(a) Hellman's basic time-space tradeoff

(b) rainbow tables

Figure 18.13: Time-space tradeoff tables, the boxed items make up the table L.

```
Algorithm $\mathcal{A}_1(L, y)$:
1.  $z \leftarrow g(y) \in \mathcal{P}$
2.  for $i = 1, \ldots, \tau$:
3.      if there is a $\widetilde{pw}$ such that $(\widetilde{pw}, z) \in L$:    // if $z$ is a chain endpoint
4.          $pw \leftarrow f^{(\tau-i)}(\widetilde{pw})$                        // traverse chain from the beginning
5.          if $h(pw) = y$:                                                     // if found inverse, output it
                output $pw$ and terminate
6.      $z \leftarrow f(z) \in \mathcal{P}$                                     // move down the chain
7.  output fail                                                                 // $g(y)$ is not on any chain
```

If the picture looked liked Fig. 18.13a, then $g(y)$ would be somewhere along one of the chains, as shown in the figure. Once we find the end of that chain, the table L would give its starting point $\widetilde{pw}$. The traversal on line (4) would then give an inverse of y. The total running time to invert y would be τ evaluations of f and at most τ lookups in L.

The situation, however, is a bit more complicated. Fig. 18.13a ignores the possibility of collisions between chains, as shown in Fig. 18.14. The first and second chains in the figure collide because $f^{(4)}(pw_1) = f^{(6)}(pw_2)$. The second and third chains collide because $f^{(5)}(pw_2) = f^{(7)}(pw_3)$. The input $g(y)$ happens to lie on the top chain. As we move along the top chain, starting from $g(y)$, we first find the end of the third chain z_3, then the end of the second chain z_2, and only then do we find the end of the first chain z_1, which lets us invert y. This is why on line (5) we must check that we found an inverse of y before outputting it, to avoid a false alarm that causes us to traverse the wrong chain. In Fig. 18.14 both z_3 and z_2 will cause false alarms. A false alarm may also happen because $g(h(pw)) = g(y)$ but $h(pw) \neq y$, which is another reason for the test on line (5).

The chain merge problem. While the basic Hellman method is quite clever, it does not work as described, and will fail to invert almost all $y = h(pw)$. Let's see why. For $\mathcal{A}_1$ to succeed we need to ensure that almost all $pw \in \mathcal{P}$ are on at least one chain. The maximum number of passwords

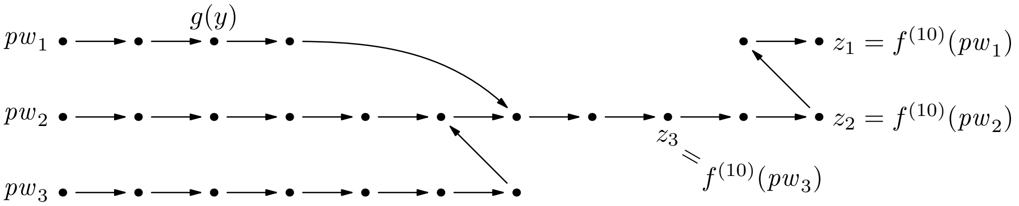


Figure 18.14: Example chain collisions, all three chains are length 10

processed by $\mathcal{A}_0$ is $\tau \times \ell$. Therefore, at the very least, we need $\tau \times \ell \geq N$. For the best performance we would like to set $\tau \times \ell = N$ and hope that most pw in $\mathcal{P}$ are on some chain.

As it turns out, this does not work. Once two chains collide, they will merge and cover the same elements, as shown in Fig. 18.14. When building a table with a large number of long chains, chain mergers are inevitable and happen frequently. To illustrate the magnitude of the problem, take $\tau = N^{1/3}$ and $\ell = N^{2/3}$ so that $\tau \times \ell = N$. Let A be the set of elements in $\mathcal{P}$ encountered during preprocessing. If we model $f : \mathcal{P} \to \mathcal{P}$ as a random function, then one can show that the set A is unlikely to contain more than $o(N)$ elements in $\mathcal{P}$. This means that $|A|/N$ tends to 0 as N goes to infinity, and algorithm $\mathcal{A}_1(L, y)$ will fail for almost all $y = h(pw)$. In fact, to capture a constant fraction of $\mathcal{P}$ we would need $\ell = \Omega(N)$ chains of length τ. This would make the table L of size $\Omega(N)$ which makes this a non interesting time-space tradeoff: with a table that big we can trivially invert h in constant time.

Hellman's solution to this problem is to build many small independent tables, where each table uses a different reduction function g. Each table contains a small number of chains of length τ ensuring that no collisions occur within a single table. Algorithm $\mathcal{A}_1$ searches every table separately and is therefore m times slower if there are m tables. This works well and achieves the bounds of (18.9). However, a different solution, called rainbow tables, is simpler and more efficient.

Rainbow tables. An elegant solution to the chain merge problem is to use an independent reduction function $g_i : \mathcal{Y} \to \mathcal{P}$ for every column $i = 1, \ldots, \tau$ of Fig. 18.13a. As before, let $f_i(pw) = g_i(h(pw))$. The preprocessing algorithm $\mathcal{A}_0$ now executes the procedure illustrated in Fig. 18.13b. It outputs the same table L as before containing the starting and ending points of every chain. If each chain were a different color, and slightly curved upwards, the picture would look like a rainbow, which explains the name.

The point of using a different function f_i in every column is that a chain collision does not necessarily cause the chains to merge. For two chains to merge they must collide at exactly the same index. This makes chain merges far less likely (see Exercise 18.18). Moreover, if a chain rooted at pw happens to merge with a chain rooted at pw', the end points z and z' of both chains will be equal. The preprocessing algorithm $\mathcal{A}_0$ can easily detect this duplicate end point and discard one of the chains. The end result is that we can set $\tau = N^{1/3}$ and $\ell = N^{2/3}$ and capture a constant fraction of $\mathcal{P}$ during preprocessing.

Now, to invert an element $y \in \mathcal{Y}$ using the table L, observe that if $g_{\tau-1}(y)$ is contained in the second to last column of Fig. 18.13b then $f_\tau(g_{\tau-1}(y))$ is a chain endpoint in L. If $g_{\tau-2}(y)$ is contained in the third to last column of the figure then $f_\tau\big(f_{\tau-1}(g_{\tau-2}(y))\big)$ is a chain endpoint in L, and so on. This suggests the following algorithm for inverting y using L:

```
Algorithm $\mathcal{A}_1(L, y)$:
1.  for $i = \tau$ downto 1:
2.      $z \leftarrow f_\tau\big(f_{\tau-1}(\cdots f_{i+1}(g_i(y))\cdots)\big) \in \mathcal{P}$     // when $i = \tau$ then $z = g_\tau(y)$,
                                                                  // when $i = \tau - 1$,   $z = f_\tau(g_{\tau-1}(y))$
3.      if there is a $\widetilde{pw}$ such that $(\widetilde{pw}, z) \in L$:      // if $z$ is a chain endpoint
4.          $pw \leftarrow f_{i-1}\big(\cdots f_2(f_1(\widetilde{pw}))\cdots\big)$       // traverse chain from the beginning
5.          if $h(pw) = y$:                                       // if found inverse, output it
                output $pw$ and terminate
6.  output fail                                                   // $y$ is not on any chain
```

The bulk of the work in this algorithm is done on line (2). In the first iteration this line evaluates f once, in the second iteration twice, and so on. Overall, the worst case work due to line (2) is $1 + 2 + \ldots + \tau = \tau(\tau + 1)/2 \approx \tau^2/2$. Hence, the maximum running time of $\mathcal{A}_1$ is $t := \tau^2/2$. To capture most of $\mathcal{P}$ we need $\ell \times \tau \geq N$, and since $\tau = (2t)^{1/2}$ we obtain

$$\ell \times (2t)^{1/2} \geq N.$$

Squaring both sides gives $\ell^2 \times t \geq N^2/2$, which is the time-space tradeoff promised in (18.9). Note also that algorithm $\mathcal{A}_1$ makes at most τ lookups into the table L.

Rainbow tables in practice. Rainbow tables for many popular hash functions are readily available. They are designed to be used with a program called **RainbowCrack**. For example, a ready-made table for SHA1 of size 460 GB is designed to find preimages in the set of all 8 character passwords over an alphabet called `ascii-32-95`. This alphabet contains all 95 characters on a standard US keyboard. The table has success rate close to 97% and is free for anyone to download. On a GPU, cracking a SHA1 hashed password of eight characters using this table takes about an hour.

Extensions. While rainbow tables are designed to invert a *random* function, a different algorithm due to Fiat and Naor [61] gives a time-space tradeoff for inverting an *arbitrary* function $h : \mathcal{P} \to \mathcal{Y}$. Their time-space tradeoff satisfies $\ell^2 t \geq \lambda N^3$, which means that to invert the function h with probability 1/2 in time t, their preprocessing algorithm must generate a table of size approximately $(\lambda N^3/t)^{1/2}$. Here λ is the collision probability of h defined as $\lambda := Pr\big[h(x) = h(y)\big]$ where $x, y \xleftarrow{R} \mathcal{P}$. When h is a random function and $|\mathcal{Y}| \gg |\mathcal{P}|$ we have $\lambda = 1/N$, which recovers the bound in (18.9).

18.8 Another fun application: hardening password storage

To be written.

18.9 Notes

Citations to the literature to be added.

18.10 Exercises

18.1 (Mutual identification). Throughout the chapter we were primarily interested in one-sided identification, where one party identifies itself to another. We can similarly develop protocols for the mutual identification that provide different levels of security. As before, the identification protocol is a triple (G, P, V), but now at setup time, algorithm G outputs (vk_1, sk_1) and (vk_2, sk_2), one pair for each side. Each participant is given the peer's verification key. The participants then run the identification protocol and each side decides whether to accept or reject the result.

(a) Security against direct attacks is defined using an attack game where the adversary is given both verification keys vk_1, vk_2, and the secret key of one side. It should be unable to successfully complete the protocol by playing the role of the other side. Give a precise security definition that extends Attack Game 18.1.

(b) Describe a password-like protocol that satisfies the security definition from part (1).

(c) Define an attack game that captures active attacks, similar to Attack Game 18.3, but applies to mutual authentication. Describe a protocol that achieves this level of security.

18.2 (An attack on PBKDF2). Let $pw \in \mathcal{P}$ be a password. Suppose the adversary obtains a $salt \in \mathcal{S}$ and three values

$$y_0 := \text{PBKDF2}_F(pw, salt, d), \quad y_1 := \text{PBKDF2}_F(pw, salt, d+1), \quad y_2 := \text{PBKDF2}_F(pw, salt, d+2)$$

for some d. Show that the adversary can recover pw in time $O(|\mathcal{P}|)$, independent of the difficulty d. You may assume that the underlying PRF F is defined over $(\mathcal{P}, \mathcal{X}, \mathcal{X})$ where $|\mathcal{X}|$ is much larger than $|\mathcal{P}|$, and that $F : \mathcal{P} \times \mathcal{X} \to \mathcal{X}$ behaves like a random function.

18.3 (Security of PBKDF2). Let H_h be a PBKDF defined over $(\mathcal{P}, \mathcal{S}, \mathcal{Y})$, and suppose that H_h is defined with respect to some underlying function $h : \mathcal{X} \to \mathcal{Z}$ that we will model as a random oracle. We say that the PBKDF is secure if no adversary that makes at most $d-1$ queries to h can distinguish H_h from a random function. In particular, define security of H_h using the following two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$ define:
Experiment b:

- The adversary $\mathcal{A}$ sends to the challenger a positive difficulty $d \in \mathbb{Z}$. The challenger chooses a random function $h : \mathcal{X} \to \mathcal{Z}$.
- The adversary then issues a sequence of queries, where for $i = 1, 2, \ldots$ query i is one of:
  - an H_h query: the adversary sends $pw_i \in \mathcal{P}$. In response, the challenger chooses $salt_i \xleftarrow{\text{R}} \mathcal{S}$ and $\tilde{y}_i \xleftarrow{\text{R}} \mathcal{Y}$. If $b = 0$ it sets $y_i \leftarrow H_h(pw_i, salt_i, d)$. If $b = 1$ it sets $y_i \leftarrow \tilde{y}_i$. The challenger sends $(y_i, salt_i)$ to the adversary.
  - an h query: the adversary sends $x_i \in \mathcal{X}$ and gets back $h(x_i)$.
- Finally, the adversary $\mathcal{A}$ outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b, and define $\mathcal{A}$'s advantage with respect to H_h as $\big|\Pr[W_0] - \Pr[W_1]\big|$. We say that H_h is a **secure PBKDF** if no adversary that makes at most $(d-1)$ queries to h has a non-negligible advantage in winning the game. Show that

PBKDF2_F is secure when the underlying PRF F is modeled as a random oracle $F : \mathcal{P} \times \mathcal{X} \to \mathcal{X}$, and $\mathcal{X}$ is super-poly.

Discussion: A security definition for a PBKDF H should require that a fast algorithm cannot distinguish the output of H from a random value. To see why, suppose there is an algorithm $\mathcal{B}(pw, salt, d)$ that quickly computes one bit of $H_h(pw, salt, d)$. When trying to crack a hashed password y, this $\mathcal{B}$ lets the adversary quickly discard about half the password candidates in the dictionary. Any candidate password that does not match y on the bit output by $\mathcal{B}$ can be quickly discarded. For this reason we require that no fast algorithm can distinguish the output of a secure PBKDF from random.

More discussion: A more complete definition would allow the adversary $\mathcal{A}$ to preprocess the function h, before it engages in the game above. Specifically, we let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ where $\mathcal{A}_0$ runs in a preprocessing phase for unbounded time, interacts with h, and outputs an ℓ-bit advice string L. Then $\mathcal{A}_1$ runs as in the game defined above, taking L as input. When $\ell < |\mathcal{S}|$, the preprocessing phase should not improve the adversary's advantage by more than a negligible amount.

The definition can be further strengthened to require that distinguishing the output of H from random at p points is p times harder than doing so at a single point. This stronger security notion was studied in [13] using a definition based on indifferentiability. They show that both PBKDF1 and PBKDF2 satisfy this stronger property.

18.4 (A stronger model for slow hash functions). Suppose we modify the security definition in Exercise 18.3 so that the adversary can specify an arbitrary difficulty d for every H_h query. That is, H_h query number i is a pair (pw_i, d_i) and both pw_i and d_i are used to compute the response. The rest of the security definition is unchanged. Exercise 18.2 shows that PBKDF2 is insecure under this stronger security definition. Show that the PBKDF H_h defined as $H_h(pw, salt, d) = h^{(d)}(pw, salt, d)$ satisfies this stronger definition. Here h is a function $h : \mathcal{X} \to \mathcal{X}$ where $\mathcal{X} = \mathcal{P} \times \mathcal{S} \times \mathbb{Z}_n$ and where n is the maximum supported difficulty.

18.5 (Broken Scrypt). Suppose line (4) of the Scrypt hash in Fig. 18.6 were changed to the following:

```
4.        $j \leftarrow \text{int}(h(i)) \bmod (d+1)$
```

where i is encoded as an element of $\mathcal{X} = \{0,1\}^n$. Show how to evaluate the resulting function using only $d/3$ memory cells without much impact to the running time. Use the fact that the order of reads from the array $(x_1, \ldots, x_d)$ is known in advance.

18.6 (A time-space tradeoff attack on Scrypt). This exercise shows how to evaluate Scrypt with little memory. Recall that for difficulty d Scrypt can be evaluated in time $O(d)$ using memory for d elements of $\mathcal{X}$.

(a) Show that Scrypt (Fig. 18.6) can be evaluated in constant space, by storing only *two* elements of $\mathcal{X}$. The running time, however, degrades to $O(d^2)$ evaluations of H instead of $O(d)$. Your attack shows that Scrypt is vulnerable to a time-space tradeoff, but one that greatly harms the running time.

(b) For $1 < t < d$, generalize part (a) to show an algorithm that evaluates Scrypt by only storing t elements of $\mathcal{X}$ and runs in time $O(d^2/t)$.

18.7 (A time-space tradeoff for one-way permutations). In Section 18.7 we saw a time-space tradeoff for one-way functions. In this exercise we develop a time-space tradeoff for one-way permutations, which is simpler and much better. Let $\pi : \mathcal{X} \to \mathcal{X}$ be a random permutation and let $N := |\mathcal{X}|$. For a given ℓ, construct an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ where $\mathcal{A}_0$ preprocesses π and outputs an advice string L containing ℓ elements of $\mathcal{X}$. Then for $y := \pi(x)$, where $x \xleftarrow{R} \mathcal{X}$, adversary $\mathcal{A}_1(L, y)$ outputs x after issuing at most $t := \lceil N/\ell \rceil$ queries to π.

Hint: Try using the cycle structure of the permutation π.

Discussion: Your solution gives a time-space tradeoff satisfying $\ell \times t \geq N$ for inverting a random permutation. This is known to be the best possible [156, 75]. For a random function we had $\ell^2 \times t \geq N^2$, which is a much worse tradeoff. To see why, try setting $\ell = N^{2/3}$ and see what is the resulting time bound t in each case. It is still an open problem if there is a better time-space tradeoff for random functions.

18.8 (A time-space tradeoff for iterated permutations). Let $\pi : \mathcal{X} \to \mathcal{X}$ be a random permutation and let $\pi^{(d)}$ be its d-th iterate, for some $d > 0$. Let $N := |\mathcal{X}|$. Give an algorithm that succeeds with probability close to 1 in inverting $\pi^{(d)}$ in time t using an advice string L of length ℓ, where $t \times \ell \geq N$. Notice that the bound on t and ℓ is independent of d, and is the same as the time-space tradeoff bound for inverting π. This means that inverting $\pi^{(d)}$ with preprocessing is no harder than inverting π.

18.9 (A batch-vulnerable one-way function). In Section 18.3.1.3 we discussed batch inversion attacks on one-way functions. Let H be a one-way function defined over $(\mathcal{X}, \mathcal{Y})$. We say that H is **batch-vulnerable** if inverting H at one random point can be done at about the same time as inverting H at t random points, for some $t > 1$. Show that the function $H(x) = x^2$ defined over $(\mathbb{Z}_n, \mathbb{Z}_n)$ is a one-way function assuming factoring is hard, but is batch-vulnerable. Here $n \xleftarrow{R} \text{RSAGen}(\ell, e)$ is an RSA modulus treated as a system parameter.

18.10 (Why multiple impersonation attempts for eavesdropping security). This exercise explains why when vk is kept secret, it is necessary to allow the adversary in Attack Game 18.2 to make multiple impersonation attempts. Describe a 3-round challenge-response protocol that is secure against eavesdropping (and even secure against active attacks) if the adversary can only make one impersonation attempt. But is completely insecure, even against direct attacks, if the adversary can make two impersonation attempts.
Solution:

- G: pick a random $k \xleftarrow{R} \mathcal{K}$ and output $sk := k$ and $vk := k$.
- Algorithm P given sk, and algorithm V given vk, interact as follows:
  (a) V sends the $c \xleftarrow{R} \mathcal{M}$ to P;
  (b) P computes $t \xleftarrow{R} S(sk, c)$, and sends t to V;
  (c) If $t = 0$ then V sends to P the secret key k;
  (d) V outputs $V(vk, c, t)$.

It should be clear that if the adversary can make two impersonation attacks then the protocol is not secure even under a direct attack. However, if only one impersonation attempt is allowed then no amount of eavesdropping will break the protocol since t is unlikely to be 0 in any of the eavesdropping transcripts.

18.11 (Why interact with the verifier for active security). In this exercise we show that when vk is kept secret, it is necessary to allow an active adversary in Attack Game 18.3 to interact with the verifier during the probing phase. We describe a protocol that is secure if the adversary cannot interact with the verifier during the probing phase, but is trivially insecure otherwise. The protocol is standard Challenge-Response except that the verifier always uses the same challenge.

- G: choose a random $k \xleftarrow{\text{R}} \mathcal{K}$ and $c \xleftarrow{\text{R}} \mathcal{M}$. Output $sk := k$ and $vk := (k, c)$.
- Algorithm P given sk, and algorithm V given vk, interact as follows:
  (a) V sends the c specified in vk to P;
  (b) P computes $t \xleftarrow{\text{R}} S(sk, c)$, and sends t to V;
  (c) V outputs $V(vk, c, t)$.

(a) Show that this ID protocol is (weakly) secure against an active adversary playing Attack Game 18.9 where the adversary cannot interact the verifier during the probing phase.

(b) Show that the protocol is insecure against an active adversary playing Attack Game 18.9 where the adversary can interact the verifier.

18.12 (Improving S/key performance). In this question we reduce the number of hash function evaluations for the prover.

(a) Suppose the prover only stores the base of the hash chain (namely, the first element in the chain). After n logins, how many times did the prover have to evaluate the hash function H? How many times did the server evaluate the hash function H?

(b) Suppose that in addition to the base of the hash chain h_0, the prover also stores the midpoint, namely $h_{n/2} = H^{(n/2)}(h_0)$ where $H^{(n/2)}(h_0)$ refers to $n/2$ repeated applications of H. Explain why this reduces the prover's total number of hash evaluations after n logins by about a factor of 2.

(c) Show that by storing the base point plus one more point (i.e. the total storage is as in part (b)) the prover can, in fact, reduce the total number of hashes after n logins to $O(n^{3/2})$. Hence, the prover does $O(\sqrt{n})$ hashes on average per login by storing only two values.

(d) Generalize part (c) — show that by storing $\log_2 n$ points along the chain the prover can reduce the total number of hashes after n logins to $O(n)$. Hence, the prover only does a constant number of hashes on average per login.

18.13 (Challenge-response by decryption). Let (G', E, D) be a public-key encryption scheme with message space $\mathcal{R}$. Consider the following challenge-response ID protocol (G, P, V):

- G: run G' to obtain a public key vk and a secret key sk.
- Algorithm P given sk, and algorithm V given vk, interact as follows:
  (a) V chooses a random nonce $r \xleftarrow{\text{R}} \mathcal{R}$, and sends $c \xleftarrow{\text{R}} E(vk, r)$ to P;
  (b) P computes $\hat{r} \leftarrow D(sk, c)$, and sends $\hat{r}$ to V;
  (c) V outputs accept only if $r = \hat{r}$.

Show that this protocol is secure against active attacks, assuming that the nonce space $\mathcal{R}$ is super-poly, and the encryption scheme is non-adaptive CCA secure, as defined in Exercise 12.28.

Discussion: This scheme is an attractive option for login to a remote web site (the verifier) from a laptop using a mobile phone (the prover) as a second factor. To login, the web site displays c as a QR code on the laptop screen and the user scans the code using the phone's camera. The phone decrypts c and displays the six least significant digits of r on the screen. The user then manually types the six digits into her web browser, and this value is sent to the remote web site to be verified.

18.14 (Insecure challenge-response by decryption). Continuing with Exercise 18.13, let's see why non-adaptive CCA is necessary for security. Give an example public-key system (G', E, D) that is semantically secure, but when used in the protocol of Exercise 18.13 leads to a protocol that is not secure against active attacks.

18.15 (Challenge-response using Diffie-Hellman). Let $\mathbb{G}$ be a cyclic group of prime order q with generator $g \in \mathbb{G}$. Let $H : \mathbb{G}^2 \to \mathcal{T}$ be a hash function. Consider the following three-round identification protocol (G, P, V) where vk is public:

- G: choose a random $\alpha \xleftarrow{R} \mathbb{Z}_q$ and set $u \leftarrow g^\alpha \in \mathbb{G}$. Output $sk := \alpha$ and $vk := u$.
- Algorithm P given sk, and algorithm V given vk, interact as follows:
  (a) V chooses a random $\beta \in \mathbb{Z}_q$, computes $v \leftarrow g^\beta \in \mathbb{G}$, and sends v to P;
  (b) P computes $w \leftarrow v^\alpha \in \mathbb{G}$, and sends $t \leftarrow H(v, w)$ to V;
  (c) V outputs accept only if $t = H(v, u^\beta)$.

Show that this protocol is secure against active attacks when $|\mathcal{T}|$ is super-poly, the ICDH assumption (Definition 12.4) holds for $\mathbb{G}$, and H is modeled as a random oracle.

Discussion: Instantiating Exercise 18.13 with CCA-secure ElGamal encryption gives a similar protocol to the one here, but the protocol in this exercise is much simpler.

18.16 (Identification using a weak PRF). Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ where $\mathcal{Y} := \{0,1\}^n$. Consider the following three-round identification protocol where vk is kept secret:

- G: choose random $k_0, k_1 \xleftarrow{R} \mathcal{K}$ and output $sk := (k_0, k_1)$ and $vk := (k_0, k_1)$.
- Algorithm P given sk, and algorithm V given vk, interact as follows:
  (a) P chooses a random $x_0 \in \mathcal{X}$ and sends it to V;
  (b) V chooses a random $x_1 \in \mathcal{X}$ and send it to P;
  (c) P computes $y \leftarrow F(k_0, x_0) \oplus F(k_1, x_1)$ and sends it to V;
  (d) V outputs accept only if $y = F(k_0, x_0) \oplus F(k_1, x_1)$.

Show that this protocol provides weak security against active attacks (Definition 18.9), assuming F is a weak PRF (as in Definition 4.3), and $|\mathcal{X}|$ and $|\mathcal{Y}|$ are super-poly. In Chapter 17 we saw an efficient weak PRF that makes this protocol computationally very cheap for the verifier and the prover.

Hint: The proof makes use of rewinding, as explained in Lemma 19.2. If you get stuck, see Section 5.2 of [55].

18.17 (Timing attacks). Consider a password system where the verifier has a stored hashed password $h \leftarrow H(pw)$. We treat the hashed password h as a string of bytes. Given a password pw' the verifier does:

```
$h' \leftarrow H(pw')$
for $i = 0, \ldots, |h|$ do:
    if $h[i] \neq h'[i]$ output reject and exit
output accept
```

(a) Show that this implementation is vulnerable to a timing attack. An attacker who can submit arbitrary queries to the verifier can recover a victim user's hashed password h with at most $256 \cdot |h|$ queries to the password checker. The attacker can mount an offline dictionary attack on h.

(b) How would you implement the verifier to prevent the timing attack from part (a)?

18.18 (The likelihood of a chain merge in rainbow tables). Consider the preprocessing phase described in Fig. 18.13b. Suppose the parameters ℓ and τ are chosen so that $\ell\tau = N$. Show that with probability at least $1/e \approx 0.37$, a chain rooted at a random starting point $pw \overset{\text{R}}{\leftarrow} \mathcal{P}$, will not merge with any of the other $\ell - 1$ chains. You may assume that every chain is a sequence of random independent elements in $\mathcal{P}$, unless the chain merges with another chain, in which case both chains share all subsequent elements.

Discussion: Because $\mathcal{A}_0$ can easily detect chain merges, it will only need to generate every chain three times, in expectation, to build a set of non-merging chains. A set of ℓ non-merging chains covers about $(1 - 1/e) \approx 0.63$ of $\mathcal{P}$ in expectation.

18.19 (A preprocessing attack on discrete-log). In Section 18.7 we saw how rainbow tables are used to invert a one-way function in the preprocessing model. This exercise and the next look at preprocessing attacks on other cryptographic primitives. We start with discrete-log.

Let $\mathbb{G}$ be a finite cyclic group of order q with generator g. In Chapter 16 we presented Pollard's algorithm for computing discrete logarithm in $\mathbb{G}$ in time $O(q^{1/2})$. Suppose the adversary can preprocess the group $\mathbb{G}$ to build an advice string L containing $q^{1/3}$ group elements. We construct an algorithm that takes L as input and computes discrete logarithm in $\mathbb{G}$ in time $O(q^{1/3})$. This is much faster than $O(q^{1/2})$.

(a) Let $H : \mathbb{G} \to \{1, \ldots, q-1\}$ be a random function. Choose $r \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$ and let $u = g^r$. Define the following chain of length τ starting at u:

$$u_1 := u, \qquad u_2 := u_1 \cdot g^{H(u_1)}, \qquad u_3 := u_2 \cdot g^{H(u_2)}, \quad \ldots, \quad u_\tau := u_{\tau-1} \cdot g^{H(u_{\tau-1})}. \quad (18.10)$$

Observe that $u_\tau = g^{r+H(u_1)+\cdots+H(u_{\tau-1})}$ and therefore it is easy to calculate the discrete logarithm of u_τ base g just given u and r. The preprocessing algorithm constructs ℓ chains as in (18.10) by choosing ℓ random starting points $u^{(1)}, \ldots, u^{(\ell)} \in \mathbb{G}$ that have a known discrete logarithm. Each chain has length τ. We can represent the set of chains as an $\ell \times \tau$ matrix. The algorithm outputs the advice string L as a table containing the last point of each chain along with its discrete logarithm base g. Thus L contains exactly ℓ pairs in $\mathbb{G} \times \mathbb{Z}_q$.

We say that preprocessing succeeds if the $\ell \times \tau$ matrix of chains contains at least $(\ell \cdot \tau)/2$ distinct elements in $\mathbb{G}$. Suppose $\ell = \tau = q^{1/3}$. Show that preprocessing succeeds with probability at least $1/2$.

(b) The algorithm to compute discrete logarithm base g takes as input the table L and an element $h \in \mathbb{G}$. It outputs the discrete logarithm of h. The algorithm builds a chain starting at h (as in (18.10), but of length possibly greater than τ). Show that, with constant probability, after at most $q/(\tau \cdot \ell)$ steps, this chain will collide with some element in the $\ell \times \tau$ matrix of chains constructed during preprocessing. Assume that preprocessing succeeded.

(c) After the collision from part (b) happens, if we continue along the chain rooted at h for at most τ more steps, the chain will collide with some element in the table L. The algorithm can detect when this happens. Show that once this collision with L happens, the algorithm can efficiently compute the discrete logarithm of h.

Hence, the overall time to compute the discrete logarithm of h, once the table L is computed, is $t = O\big(q/(\tau \cdot \ell) + \tau\big)$. Setting $\tau = \ell = q^{1/3}$ gives a running time of $O(q^{1/3})$ using a table L of size $\ell = q^{1/3}$, as promised. Note that the one-time preprocessing phase takes time $q^{2/3}$.

18.20 (A preprocessing attack on the Even-Mansour cipher). The same idea as in Exercise 18.19 can be used to attack the Even-Mansour cipher in the preprocessing model. For $\mathcal{X} := \{0,1\}^n$, recall that the Even-Mansour block cipher (E, D), built from a random permutation $\pi : \mathcal{X} \to \mathcal{X}$, is defined as:

$$E\big((k_0, k_1),\ x\big) := \pi(x \oplus k_0) \oplus k_1, \quad \text{and} \quad D\big((k_0, k_1),\ y\big) := \pi^{-1}(y \oplus k_1) \oplus k_0.$$

In Exercise 4.21 we showed how to distinguish this block cipher from a random permutation after about $2^{n/2}$ queries to π and $2^{n/2}$ queries to E. Suppose the adversary can preprocess the permutation π to build an ℓ-bit advice string L, where $\ell = 2^{n/3}$. Let's show that using this L we can break the cipher (E, D) in time $2^{n/3}$, which is much less than $2^{n/2}$.

(a) Fix some non-zero $\Delta \in \mathcal{X}$ and define the functions

$$f(x) := x \oplus \pi(x) \oplus \pi(x \oplus \Delta), \qquad g(x) := x \oplus E(x) \oplus E(x \oplus \Delta),$$

where $E(x) := E\big((k_0, k_1), x\big)$. Show that if $x = y \oplus k_0$ then

$$f(x) = g(y) \oplus k_0 \qquad \text{and} \qquad x \oplus f(x) = y \oplus g(y). \tag{18.11}$$

(b) To preprocess π the preprocessing algorithm chooses ℓ random starting points $x_1, \ldots, x_\ell \xleftarrow{R} \mathcal{X}$. For each x_i it builds a chain of length τ starting at x_i:

$$x_i, \quad f(x_i), \quad f(f(x_i)), \quad \ldots, \quad f^{\tau-1}(x_i).$$

This gives an $\ell \times \tau$ matrix of elements in $\mathcal{X}$. For every end point $z_i := f^{\tau-1}(x_i)$ it stores $z_i \oplus f(z_i)$ in the table L. Thus, L contains exactly ℓ elements in $\mathcal{X}$.

We say that preprocessing succeeds if the $\ell \times \tau$ matrix of chains contains $(\ell \cdot \tau)/2$ distinct elements in $\mathcal{X}$. Show that preprocessing succeeds with probability at least $1/2$ when $\ell = \tau = 2^{n/3}$ and π is a random permutation in $\text{Perms}[\mathcal{X}]$.

(c) To distinguish the Even-Mansour cipher (E, D) from a random permutation choose a random $y \xleftarrow{R} \mathcal{X}$ and build the chain

$$y_1 := y, \quad y_2 := g(y), \quad y_3 := g(g(y)), \quad \ldots, \quad y_\nu := g^\nu(y) \tag{18.12}$$

for $\nu := 2^n/(\tau \cdot \ell) + \tau$. At every step along the chain check if $y_j \oplus g(y_j)$ is in the table L. If so output 0 (indicating that E is an Even-Mansour cipher) and stop. If this never happens, output 1 (indicating that E is random). Show that this distinguisher has advantage close to 1. Setting $\ell = \tau = 2^{n/3}$ gives $\nu = 2^{n/3}$ so that the distinguisher runs in time $2^{n/3}$ using a table L of size $2^{n/3}$, as promised.

Hint: Show that, with constant probability, after at most $2^n/(\tau \cdot \ell)$ steps, the chain (18.12) will hit a point y_i such that $y_i \oplus k_0$ in contained in the $\ell \times \tau$ matrix of chains constructed during preprocessing (regardless of whether E implements an Evan-Mansour cipher or is a random permutation). Once this happens, use the left hand side of (18.11) to argue that this condition will continue to hold as the algorithm continues down the chain, assuming that E implements an Evan-Mansour cipher. Therefore, after at most τ more steps, use the right hand side of (18.11) to argue that the chain (18.12) will hit a point y_j such that $y_j \oplus g(y_j)$ is in L. Show that this is unlikely to happen when E is a random permutation.

Chapter 19

Identification and signatures from Sigma protocols

In the previous chapter, we studied identification protocols. In particular, in Section 18.6.1.1, we showed how one could use a secure signature scheme to build a challenge-response identification scheme that provided the highest level of security, namely, security against active attacks (Definition 18.8). In this chapter, we proceed in the opposite direction.

First, using a completely different technique, we develop a new identification protocol that achieves security against eavesdropping attacks (Definition 18.6). This protocol is of interest in its own right, because it is quite elegant, and can be proved secure under the DL assumption.

Second, we show how to transform this protocol into a very efficient signature scheme called the **Schnorr signature scheme**. The scheme is secure, under the DL assumption, in the random oracle model.

Third, we generalize these techniques, introducing the notion of a **Sigma protocol**. Using these more general techniques, we develop several new identification protocols and signature schemes. We also explore other concepts and applications, including the notion of a "proof of knowledge".

In the next chapter, we put these techniques to more advanced use, designing protocols that allow one party to prove to another that certain facts are true (without revealing unnecessary information). For example, we show how to prove that encrypted value m lies in a certain range without revealing any other information about m.

19.1 Schnorr's identification protocol

We begin by describing an identification protocol, called **Schnorr identification**, named after its inventor, C. Schnorr. This protocol can be proved secure against eavesdropping attacks, assuming the discrete logarithm problem is hard.

Let $\mathbb{G}$ be a cyclic group of prime order q with generator $g \in \mathbb{G}$. Suppose prover P has a secret key $\alpha \in \mathbb{Z}_q$, and the corresponding public verification key is $u = g^\alpha \in \mathbb{G}$. To prove his identity to a verifier V, P wants to convince V that he knows α. The simplest way to do this would be for P to simply send α to V. This protocol is essentially just the basic password protocol (version 1) discussed in Section 18.3, with the function $H(\alpha) := g^\alpha$ playing the role of the one-way function. As such, while this protocol provides security against direct attacks, it is completely insecure against eavesdropping attacks. Instead, Schnorr's protocol is a cleverly designed interactive protocol that

$$\begin{array}{ccc}
\underline{P(\alpha)} & & \underline{V(u)} \\
\alpha_{\mathrm{t}} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q,\ u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{t}}} & & \\
& \xrightarrow{\qquad\qquad u_{\mathrm{t}} \qquad\qquad} & \\
& & c \overset{\mathrm{R}}{\leftarrow} \mathcal{C} \\
& \xleftarrow{\qquad\qquad c \qquad\qquad} & \\
\alpha_{\mathrm{z}} \leftarrow \alpha_{\mathrm{t}} + \alpha c & & \\
& \xrightarrow{\qquad\qquad \alpha_{\mathrm{z}} \qquad\qquad} & \\
& & g^{\alpha_{\mathrm{z}}} \overset{?}{=} u_{\mathrm{t}} \cdot u^c
\end{array}$$

Figure 19.1: Schnorr's identification protocol

allows P to convince V that he knows the discrete logarithm of u to the base g, without actually sending this value to V.

Here is how it works. Let $\mathcal{C}$ be a subset of $\mathbb{Z}_q$. Then Schnorr's identification protocol is $\mathcal{I}_{\mathrm{sch}} = (G, P, V)$, where:

- The key generation algorithm G runs as follows:

$$\alpha \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha}.$$

  The verification key is $vk := u$, and the secret key is $sk := \alpha$.

- The protocol between P and V runs as follows, where the prover P is initialized with $sk = \alpha$, and the verifier V is initialized with $vk = u$:

  1. P computes $\alpha_{\mathrm{t}} \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q$, $u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{t}}}$, and sends u_{t} to V;
  2. V computes $c \overset{\mathrm{R}}{\leftarrow} \mathcal{C}$, and sends c to P;
  3. P computes $\alpha_{\mathrm{z}} \leftarrow \alpha_{\mathrm{t}} + \alpha c \in \mathbb{Z}_q$, and sends α_{z} to V;
  4. V checks if $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$; if so V outputs accept; otherwise, V outputs reject.

Fig. 19.1 illustrates the protocol.

An interaction between $P(\alpha)$ and $V(u)$ generates a **conversation** $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}}) \in \mathbb{G} \times \mathcal{C} \times \mathbb{Z}_q$. We call such a conversation an **accepting conversation for** u if V's check passes, i.e., if $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$. It is easy to see that an interaction between P and V always generates an accepting conversation, since if $u_{\mathrm{t}} = g^{\alpha_{\mathrm{t}}}$ and $\alpha_{\mathrm{z}} = \alpha_{\mathrm{t}} + \alpha c$, then

$$g^{\alpha_{\mathrm{z}}} = g^{\alpha_{\mathrm{t}} + \alpha c} = g^{\alpha_{\mathrm{t}}} \cdot (g^{\alpha})^c = u_{\mathrm{t}} \cdot u^c.$$

Therefore, Schnorr's protocol satisfies the basic correctness requirement that any identification protocol must satisfy.

The set $\mathcal{C}$ is called the **challenge space**. To prove security, we require that $|\mathcal{C}|$ is super-poly. Indeed, we could simply take $\mathcal{C}$ to be $\mathbb{Z}_q$, but it is technically convenient to allow somewhat smaller challenge spaces as well. Although we will eventually prove that Schnorr's protocol is secure against *eavesdropping* attacks (under the DL assumption), we begin with a simpler theorem, which proves security only against *direct* attacks (Attack Game 18.1). In proving this, we will show that any efficient adversary that can succeed in a direct impersonation attack with non-negligible probability can be turned into an algorithm that efficiently recovers the secret key α from the verification key u. For this reason, Schnorr's protocol is sometimes called a "proof of knowledge" of a discrete logarithm.

Theorem 19.1. *Under the DL assumption for $\mathbb{G}$, and assuming $N := |\mathcal{C}|$ is super-poly, Schnorr's identification protocol is secure against direct attacks.*

> *In particular, suppose $\mathcal{A}$ is an efficient impersonation adversary attacking $\mathcal{I}_{\text{sch}}$ via a direct attack as in Attack Game 18.1, with advantage $\epsilon := \text{ID1}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{sch}}]$. Then there exists an efficient DL adversary $\mathcal{B}$ (whose running time is about* twice *that of $\mathcal{A}$), with advantage $\epsilon' := \text{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}]$, such that*
>
> $$\epsilon' \geq \epsilon^2 - \epsilon/N, \tag{19.1}$$
>
> *which implies*
>
> $$\epsilon \leq \frac{1}{N} + \sqrt{\epsilon'}. \tag{19.2}$$

Proof idea. Suppose $\mathcal{A}$ has advantage ϵ in attacking $\mathcal{I}_{\text{sch}}$ as in Attack Game 18.1. In this game, the challenger generates the verification key $u = g^{\alpha}$. In his impersonation attempt, the adversary $\mathcal{A}$ generates the first flow u_{t} of the protocol using some arbitrary adversarial strategy. Now, to succeed, $\mathcal{A}$ must reply to a random challenge $c \in \mathcal{C}$ with a valid response α_{z} that satisfies $g^{\alpha_{\text{z}}} = u_{\text{t}} \cdot u^{c}$. Intuitively, if $\mathcal{A}$ can generate a valid response to one such random challenge with probability ϵ, it should be able to generate a valid response to *two* random challenges with probability ϵ^2. Making this intuition rigorous requires a somewhat technical argument that will be presented in a lemma below.

So here is how we can use $\mathcal{A}$ to compute the discrete logarithm of a random $u \in \mathbb{G}$. We use u as the verification key in $\mathcal{I}_{\text{sch}}$, and let $\mathcal{A}$ generate the first flow u_{t} of the protocol. We then supply a random challenge c to $\mathcal{A}$ and hope that $\mathcal{A}$ generates a valid response α_{z}. If this happens, we "rewind" $\mathcal{A}$'s internal state back to the point just after which it generated u_{t}, and then supply $\mathcal{A}$ with another random challenge c', and hope that $\mathcal{A}$ generates another valid response α'_{z}.

If all of this happens, then we obtain two accepting conversations $(u_{\text{t}}, c, \alpha_{\text{z}})$ and $(u_{\text{t}}, c', \alpha'_{\text{z}})$ for a given verification key u and with matching first flows u_{t}. Moreover, with overwhelming probability, we have $c' \neq c$ (this is where the assumption that $\mathcal{C}$ is super-poly comes in). Given this information, we can easily compute $\mathsf{Dlog}_g u$. Indeed, since both conversations are accepting, we have the two equations:

$$g^{\alpha_{\text{z}}} = u_{\text{t}} \cdot u^{c} \quad \text{and} \quad g^{\alpha'_{\text{z}}} = u_{\text{t}} \cdot u^{c'}.$$

Dividing the first equation by the second, the u_{t}'s cancel, and we have

$$g^{\Delta\alpha} = u^{\Delta c}, \quad \text{where } \Delta\alpha := \alpha_{\text{z}} - \alpha'_{\text{z}}, \;\; \Delta c := c - c'. \tag{19.3}$$

Since $\Delta c \neq 0$, and the group order q is prime, the inverse $1/\Delta c$ exists in $\mathbb{Z}_q$. We can now raise both sides of (19.3) to the power $1/\Delta c$, obtaining

$$g^{\Delta\alpha/\Delta c} = u.$$

Therefore, we can efficiently compute $\mathsf{Dlog}_g u$ as $\Delta\alpha/\Delta c$.

The reader should observe that the technique presented here for computing the discrete log from two accepting conversations is essentially the same idea as was used in Fact 10.3. Indeed, using the terminology introduced in Section 10.6.1, we see that $(\alpha_{\mathrm{z}}, -c)$ and $(\alpha'_{\mathrm{z}}, -c')$ are distinct representations (relative to g and u) of u_{t}, and Fact 10.3 tells us how to compute $\mathsf{Dlog}_g u$ from these two representations. □

This theorem is qualitatively different than all of the other security theorems we have presented so far in this text. Indeed, in the proof of this theorem, while we show that every adversary $\mathcal{A}$ that breaks $\mathcal{I}_{\text{sch}}$ can be converted into an adversary $\mathcal{B}$ that breaks the discrete logarithm problem, the adversary $\mathcal{B}$ that we construct is *not* an elementary wrapper around $\mathcal{A}$. Adversary $\mathcal{B}$ has to basically run $\mathcal{A}$ *twice.* In addition, this theorem is quantitatively different as well, in that the security reduction is very loose: if $\mathcal{A}$ succeeds with probability ϵ, then $\mathcal{B}$ is only guaranteed to succeed with probability $\approx \epsilon^2$.

To make the above proof idea rigorous, we need the following technical lemma:

Lemma 19.2 (Rewinding Lemma). *Let S and T be finite, non-empty sets, and let $f : S \times T \to \{0, 1\}$ be a function. Let X, Y, and Y' be mutually independent random variables, where X takes values in the set S, and Y and Y' are each uniformly distributed over T. Let $\epsilon := \Pr[f(\mathsf{X}, \mathsf{Y}) = 1]$ and $N := |T|$. Then*

$$\Pr[f(\mathsf{X}, \mathsf{Y}) = 1 \wedge f(\mathsf{X}, \mathsf{Y}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'] \geq \epsilon^2 - \epsilon/N.$$

Proof. For each $s \in S$, let $g(s) := \Pr[f(s, \mathsf{Y}) = 1]$. First, observe that $E[g(\mathsf{X})] = \epsilon$; indeed, we have

$$\begin{aligned}
E[g(\mathsf{X})] &= \sum_{s \in S} g(s) \Pr[\mathsf{X} = s] = \sum_{s \in S} \Pr[f(s, \mathsf{Y}) = 1] \Pr[\mathsf{X} = s] \\
&= \sum_{s \in S} \Pr[f(s, \mathsf{Y}) = 1 \wedge \mathsf{X} = s] \quad \text{(by independence)} \\
&= \sum_{s \in S} \Pr[f(\mathsf{X}, \mathsf{Y}) = 1 \wedge \mathsf{X} = s] \\
&= \Pr[f(\mathsf{X}, \mathsf{Y}) = 1] \quad \text{(by total probability)} \\
&= \epsilon.
\end{aligned}$$

Second, consider a fixed $s \in S$, and let $\mathcal{G}_s$ be the event that $f(s, \mathsf{Y}) = 1 \wedge f(s, \mathsf{Y}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'$. We claim that

$$\Pr[\mathcal{G}_s] = g(s)^2 - g(s)/N.$$

To see this, let N_s be the number of $t \in T$ satisfying $f(s, t) = 1$. Then there are N_s ways to choose Y satisfying $f(s, \mathsf{Y}) = 1$, and for each choice of Y, there are at least $N_s - 1$ ways to choose Y' satisfying $f(s, \mathsf{Y}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'$ (there are exactly $N_s - 1$ ways unless $N_s = 0$). Since $g(s) = N_s/N$, we therefore have

$$\Pr[\mathcal{G}_s] \geq N_s(N_s - 1)/N^2 = N_s^2/N^2 - N_s/N^2 = g(s)^2 - g(s)/N.$$

Finally, let $\mathcal{G}$ be the event that $f(\mathbf{X}, \mathbf{Y}) = 1 \wedge f(\mathbf{X}, \mathbf{Y}') = 1 \wedge \mathbf{Y} \neq \mathbf{Y}'$. We have

$$\begin{aligned}
\Pr[\mathcal{G}] &= \sum_{s \in S} \Pr[\mathcal{G} \wedge \mathbf{X} = s] \quad \text{(by total probability)} \\
&= \sum_{s \in S} \Pr[f(s, \mathbf{Y}) = 1 \wedge f(s, \mathbf{Y}') = 1 \wedge \mathbf{Y} \neq \mathbf{Y}' \wedge \mathbf{X} = s] \\
&= \sum_{s \in S} \Pr[f(s, \mathbf{Y}) = 1 \wedge f(s, \mathbf{Y}') = 1 \wedge \mathbf{Y} \neq \mathbf{Y}'] \Pr[\mathbf{X} = s] \quad \text{(by independence)} \\
&= \sum_{s \in S} \Pr[\mathcal{G}_s] \Pr[\mathbf{X} = s] \geq \sum_{s \in S} (g(s)^2 - g(s)/N) \Pr[\mathbf{X} = s] = E[g(\mathbf{X})^2] - E[g(\mathbf{X})]/N \\
&\geq E[g(\mathbf{X})]^2 - E[g(\mathbf{X})]/N = \epsilon^2 - \epsilon/N.
\end{aligned}$$

Here, we have used the general fact that $E[\mathbf{Z}^2] \geq E[\mathbf{Z}]^2$ for any random variable $\mathbf{Z}$ (in particular, for $\mathbf{Z} := g(\mathbf{X})$). This is a special case of Jensen's inequality. □

Proof of Theorem 19.1. Using the impersonation adversary $\mathcal{A}$, which has advantage ϵ, we build a DL adversary $\mathcal{B}$, with advantage ϵ', as follows. Adversary $\mathcal{B}$ is given an instance $u = g^{\alpha}$ of the DL problem from its challenger, and our goal is to make $\mathcal{B}$ compute α, with help from $\mathcal{A}$. The computation of $\mathcal{B}$ consists of two stages.

In the first stage of its computation, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$, giving $\mathcal{A}$ the value u as the verification key. The goal of $\mathcal{B}$ in this step is to compute two accepting conversations for u with different challenges, that is,

$$(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}}) \ \text{ and } \ (u_{\mathrm{t}}, c', \alpha'_{\mathrm{z}}),$$

where

$$g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^{c}, \quad g^{\alpha'_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^{c'}, \quad \text{and} \quad c \neq c'.$$

Here is how $\mathcal{B}$ does this:

1. $\mathcal{A}$ (playing the role of prover) sends u_{t} to $\mathcal{B}$ (playing the role of verifier);

2. $\mathcal{B}$ sends a random $c \in \mathcal{C}$ to $\mathcal{A}$;

3. $\mathcal{A}$ sends α_{z} to $\mathcal{B}$;

4. $\mathcal{B}$ "rewinds" $\mathcal{A}$, so that $\mathcal{A}$'s internal state is exactly the same as it was at the end of step 1; then $\mathcal{B}$ sends a random $c' \in \mathcal{C}$ to $\mathcal{A}$;

5. $\mathcal{A}$ sends α'_{z} to $\mathcal{B}$.

Now we apply the Rewinding Lemma. In that lemma, the random variable $\mathbf{Y}$ corresponds to the challenge c, $\mathbf{Y}'$ corresponds to the challenge c', and $\mathbf{X}$ corresponds to all the other random choices made by $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{B}$'s challenger (including the group $\mathbb{G}$, and group elements $g, u, u_{\mathrm{t}} \in \mathbb{G}$). The function f in the lemma is defined to be 1 if the resulting conversation is an accepting conversation for u, and 0 otherwise. So $f(\mathbf{X}, \mathbf{Y}) = 1$ if $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ is an accepting conversation for u, and $f(\mathbf{X}, \mathbf{Y}') = 1$ if $(u_{\mathrm{t}}, c', \alpha'_{\mathrm{z}})$ is an accepting conversation for u. Applying the lemma, we find that the probability that $\mathcal{B}$ gets two accepting conversations with different challenges is at least $\epsilon^2 - \epsilon/N$.

So now assume that $\mathcal{B}$ has successfully computed two such conversations $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ and $(u_{\mathrm{t}}, c', \alpha'_{\mathrm{z}})$. In the second stage of its computation, $\mathcal{B}$ uses these two conversations to compute α. Indeed, as already discussed in the "proof idea" above, we can compute $\alpha = \Delta\alpha/\Delta c$, where $\Delta\alpha := \alpha_{\mathrm{z}} - \alpha'_{\mathrm{z}}$, $\Delta c := c - c'$.

This shows (19.1). We now argue that (19.2) follows from (19.1). To do so, we may assume that $\epsilon \geq 1/N$, as otherwise, (19.2) clearly holds. So we have

$$\begin{aligned}(\epsilon - 1/N)^2 &= \epsilon^2 - 2\epsilon/N + 1/N^2 \leq \epsilon^2 - 2\epsilon/N + \epsilon/N \quad (\text{since } \epsilon \geq 1/N) \\ &= \epsilon^2 - \epsilon/N \leq \epsilon' \quad (\text{by (19.1)}),\end{aligned}$$

from which (19.2) is clear. □

To recap, we proved security against direct attacks by showing how to efficiently *extract* the secret key α from a malicious prover $\mathcal{A}$. This enabled us to use the malicious prover to solve the discrete-log problem in $\mathbb{G}$. Our "extractor" works by rewinding the prover to obtain two conversations $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ and $(u_{\mathrm{t}}, c', \alpha'_{\mathrm{z}})$ where $c \neq c'$. Rewinding the prover $\mathcal{A}$ is possible inside the proof of security, because we have full control of $\mathcal{A}$'s execution environment. In the real world, since one cannot rewind an honest prover P, an attacker cannot use this strategy to extract the secret key from P.

19.1.1 Honest verifier zero knowledge and security against eavesdropping

We have shown that Schnorr's identification protocol is secure against *direct* attacks, under the DL assumption. In fact, under the same assumption, we can show that Schnorr's identification protocol is secure against *eavesdropping* attacks as well. Now, in an eavesdropping attack, the adversary obtains *vk* and a list of transcripts — conversations between P (on input *sk*) and V (on input *vk*). The idea is to show that these conversations do not help the adversary, because the adversary could have efficiently generated these conversations by himself, given *vk* (but not *sk*). If we can show this, then we are done. Indeed, suppose $\mathcal{A}$ is an adversary whose advantage in carrying out a successful impersonation via an eavesdropping attack is non-negligible. Then we replace $\mathcal{A}$ by another adversary $\mathcal{B}$, that works the same as $\mathcal{A}$, except that $\mathcal{B}$ generates the transcripts by himself, instead of obtaining them from his challenger. Thus, $\mathcal{B}$ carries out a direct attack, but has the same advantage as $\mathcal{A}$ in carrying out a successful impersonation.

We shall develop this idea in a more general way, introducing the notion of **honest verifier zero knowledge**.

Definition 19.1. *Let $\mathcal{I} = (G, P, V)$ be an identification protocol. We say that $\mathcal{I}$ is **honest verifier zero knowledge**, or **HVZK** for short, if there exists an efficient probabilistic algorithm Sim (called a **simulator**) such that for all possible outputs (vk, sk) of G, the output distribution of Sim on input vk is* identical *to the distribution of a transcript of a conversation between P (on input sk) and V (on input vk).*

Some comments on the terminology are in order. The term "zero knowledge" is meant to suggest that an adversary learns nothing from P, because an adversary can simulate conversations on his own (using the algorithm *Sim*), without knowing *sk*. The term "honest verifier" conveys the fact this simulation only works for conversations between P and the actual, "honest" verifier V, and not some arbitrary, "dishonest" verifier, such as may arise in an *active* attack on the identification

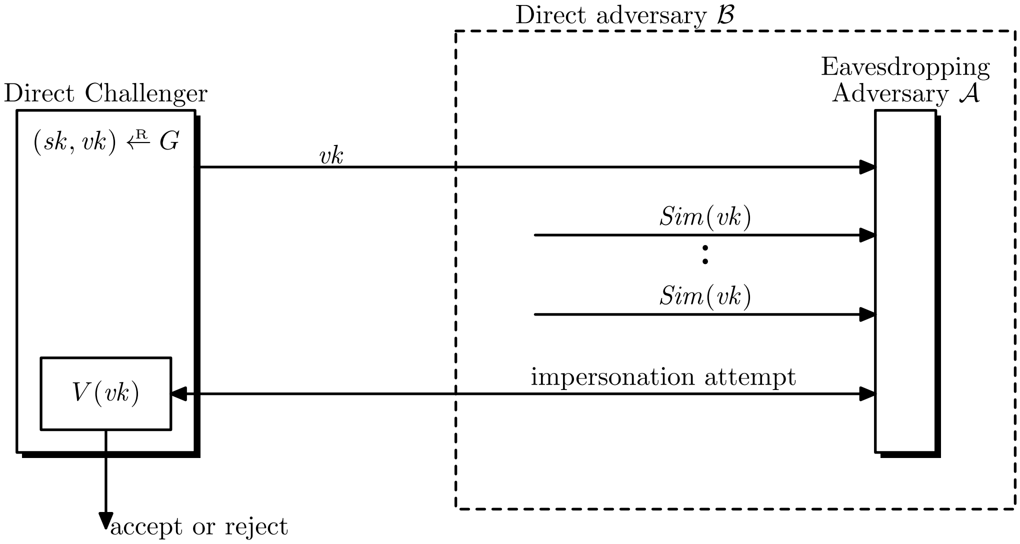

Direct adversary $\mathcal{B}$
Eavesdropping Adversary $\mathcal{A}$
Direct Challenger
$(sk, vk) \xleftarrow{R} G$
vk
$Sim(vk)$
$Sim(vk)$
impersonation attempt
$V(vk)$
accept or reject

Figure 19.2: Adversary $\mathcal{B}$ in the proof of Theorem 19.3.

protocol. The notion of zero knowledge (including honest verifier zero knowledge, and many other variants) arises in many other types of protocols besides identification protocols.

Theorem 19.3. *If an identification protocol $\mathcal{I}$ is secure against direct attacks, and is HVZK, then it is secure against eavesdropping attacks.*

> *In particular, if $\mathcal{I}$ is HVZK with simulator Sim, then for every impersonation adversary $\mathcal{A}$ that attacks $\mathcal{I}$ via an eavesdropping attack, as in Attack Game 18.2, obtaining up to Q transcripts, there is an adversary $\mathcal{B}$ that attacks $\mathcal{I}$ via a direct attack, as in Attack Game 18.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$ (and where $\mathcal{B}$ runs Sim at most Q times), such that*
>
> $$\text{ID2}\mathsf{adv}[\mathcal{A},\mathcal{I}] = \text{ID1}\mathsf{adv}[\mathcal{B},\mathcal{I}].$$

Proof. $\mathcal{B}$ works the same as $\mathcal{A}$, except that instead of obtaining transcripts from its challenger, it generates the transcripts itself using *Sim*. Adversary $\mathcal{B}$ is shown in Fig. 19.2. □

Let us now return to Schnorr's identification protocol.

Theorem 19.4. *Schnorr's identification protocol is HVZK.*

Proof. The idea is that in generating a simulated conversation (u_t, c, α_z), we do not need to generate the messages of the conversation in the given order, as in a real conversation between P and V. Indeed, our simulator *Sim* generates the messages in *reverse* order. On input $vk = u$, the simulator *Sim* computes

$$\alpha_z \xleftarrow{R} \mathbb{Z}_q, \quad c \xleftarrow{R} \mathcal{C}, \quad u_t \leftarrow g^{\alpha_z}/u^c,$$

and outputs the conversation (u_t, c, α_z).

Now we argue that the output of *Sim* on input $vk = u$ has the right distribution. The key observation is that in a real interaction, c and α_z are independent, with c uniformly distributed over $\mathcal{C}$ and α_z uniformly distributed over $\mathbb{Z}_q$; moreover, given c and α_z, the value u_t is uniquely determined by the equation $g^{\alpha_z} = u_t \cdot u^c$. It should be clear that this is the same as the output distribution of the simulator. □

As a corollary, we immediately obtain:

Theorem 19.5. *If Schnorr's identification protocol is secure against direct attacks, then it is also secure against eavesdropping attacks.*

> *In particular, for every impersonation adversary $\mathcal{A}$ that attacks $\mathcal{I}_{\text{sch}}$ via an eavesdropping attack, as in Attack Game 18.2, there is an adversary $\mathcal{B}$ that attacks $\mathcal{I}_{\text{sch}}$ via a direct attack, as in Attack Game 18.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\text{ID2}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{sch}}] = \text{ID1}\mathsf{adv}[\mathcal{B}, \mathcal{I}_{\text{sch}}].$$

At first blush, our results about Schnorr's protocol may seem counter-intuitive, or perhaps even contradictory. Namely, how can it be hard to carry out an impersonation attack, knowing only *vk*, and yet be easy to generate a conversation, also knowing only *vk*? The answer is that in carrying out an impersonation attack, the verifier V is actively involved in the conversation, and the timing and ordering of the messages is critical: the adversary (playing the role of a prover) must generate the first message u_t *before* it sees the challenge c generated by V. However, the simulator is free to generate the messages in any convenient order: our simulator in the proof of Theorem 19.4 generates c and α_z, and then computes u_t. Indeed, what these results do show is that Schnorr's identification protocol would be completely *insecure* if the challenge space were small: in its impersonation attempt, an adversary could use the simulator to prepare an accepting conversation (u_t, c, α_z), send u_t to V, and then hope that the challenge chosen by V is equal to its prepared challenge c, and if so, the adversary could then respond with α_z, and so make V accept. Thus, it is trivial to break Schnorr's identification protocol with advantage $1/|\mathcal{C}|$; therefore, the challenge space $|\mathcal{C}|$ *must* be super-poly in order to ensure security.

It is an open question as to whether Schnorr's identification protocol is secure against *active* attacks as in Attack Game 18.3: there are no known effective, active attacks, but there is also no proof that rules out such an attack under the DL assumption. Later in this chapter, we shall present a slight variation on Schnorr's identification that can be proven secure against active attacks under the DL assumption.

19.2 From identification protocols to signatures

In this section, we show how to convert Schnorr's identification protocol into a signature scheme. The signature scheme is secure in the random oracle model under the DL assumption. Later in this chapter, we will see that this construction is actually a specific instance of a more general construction.

We start with Schnorr's identification protocol $\mathcal{I}_{\text{sch}}$, which is defined in terms of a cyclic group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$, along with a challenge space $\mathcal{C} \subseteq \mathbb{Z}_q$. We also need a hash function $H : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$, which will be modeled as a random oracle in the security proof. Here, $\mathcal{M}$ will be the message space of the signature scheme.

The basic idea of the construction is that a signature on a message $m \in \mathcal{M}$ will be a pair (u_t, α_z), where (u_t, c, α_z) is an accepting conversation for the verification key u in Schnorr's identification protocol, and the challenge c is computed as $c \leftarrow H(m, u_t)$. Intuitively, the hash function H is playing the role of verifier in Schnorr's identification protocol.

In detail, the Schnorr signature scheme is $\mathcal{S}_{\text{sch}} = (G, S, V)$, where:

- The key generation algorithm G runs as follows:

$$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u \leftarrow g^{\alpha}.$$

  The public key is $pk := u$, and the secret key is $sk := \alpha$.

- To sign a message $m \in \mathcal{M}$ using a secret key $sk = \alpha$, the signing algorithm runs as follows:

$$\begin{aligned} S(\ sk,\ m\ ) := \quad & \alpha_{\text{t}} \xleftarrow{\text{R}} \mathbb{Z}_q, \quad u_{\text{t}} \leftarrow g^{\alpha_{\text{t}}}, \quad c \leftarrow H(m, u_{\text{t}}), \quad \alpha_{\text{z}} \leftarrow \alpha_{\text{t}} + \alpha c \\ & \text{output } \sigma := (u_{\text{t}}, \alpha_{\text{z}}). \end{aligned}$$

- To verify a signature $\sigma = (u_{\text{t}}, \alpha_{\text{z}})$ on a message $m \in \mathcal{M}$, using the public key $pk = u$, the signature verification algorithm V computes $c \leftarrow H(m, u_{\text{t}})$, and outputs accept if $g^{\alpha_{\text{z}}} = u_{\text{t}} \cdot u^c$, and outputs reject, otherwise.

Although we described the signing algorithm as a randomized algorithm, this is not essential. Exercise 13.6 shows how to derandomize the signing algorithm. This derandomization is important in practice, to avoid bad randomness attacks, as in Exercise 19.1.

We will show that if we model H as a random oracle, then Schnorr's signature scheme is secure if Schnorr's identification protocol is secure against eavesdropping attacks, which was already established in Theorem 19.5. It is advantageous, however, to first consider a slightly enhanced version of the eavesdropping attack game.

19.2.1 A useful abstraction: repeated impersonation attacks

We shall consider a slightly enhanced type of impersonation attack against an identification scheme, in which we allow the adversary to make many impersonation attempts (against several instances of the verifier, running concurrently, and using the same verification key). One could define this notion for either direct, eavesdropping, or active attacks, but we shall just consider eavesdropping attacks here, as that is all we need for our application. Also, we only consider identification protocols that are stateless and have a public verification key.

Here is the attack game in more detail.

Attack Game 19.1 (r-impersonation eavesdropping attack). For a given identification protocol $\mathcal{I} = (G, P, V)$, positive integer r, and adversary $\mathcal{A}$, the attack game runs as follows. The key generation and eavesdropping phase is exactly the same as in Attack Game 18.2.

The only difference is that in the impersonation phase, the adversary $\mathcal{A}$ is allowed to interact *concurrently* with up to r verifiers. The challenger, of course, plays the role of these verifiers, all of which use the same verification key as generated during the key generation phase. The adversary wins the game if it makes any of these verifiers output accept.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{I}$ and r, denoted rID2adv$[\mathcal{A}, \mathcal{I}, r]$, as the probability that $\mathcal{A}$ wins the game. □

The following lemma shows that the r-impersonation eavesdropping attack is equivalent to the ordinary eavesdropping attack. That is, winning Attack Game 19.1 is not much easier than winning Attack Game 18.2.

Lemma 19.6. *Let $\mathcal{I}$ be an identification protocol. For every r-impersonation eavesdropping adversary $\mathcal{A}$, there exists a standard eavesdropping adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*

$$\mathrm{rID2}\mathsf{adv}[\mathcal{A}, \mathcal{I}, r] \leq r \cdot \mathrm{ID2}\mathsf{adv}[\mathcal{B}, \mathcal{I}]. \tag{19.4}$$

Proof sketch. This is a simple "guessing argument". Adversary $\mathcal{B}$ simply chooses $\omega \in \{1, \ldots, r\}$ at random, and then plays the role of challenger to $\mathcal{A}$. It starts out by obtaining from its own challenger the verification key as well as the transcripts of several conversations, and passes these along to $\mathcal{A}$. During the impersonation phase, for the jth instance of the verifier, if $j \neq \omega$, our adversary $\mathcal{B}$ plays the role of verifier itself; otherwise, for $j = \omega$, it acts as a simple conduit between $\mathcal{A}$ and its own challenger in Attack Game 18.2. It should be clear that $\mathcal{A}$ makes one of the verifiers accept when playing against $\mathcal{B}$ with the same probability that it does in Attack Game 19.1. Moreover, $\mathcal{B}$ wins its attack game if it guesses the index of one of these accepting verifiers, which happens with probability at least $1/r$. □

19.2.2 Security analysis of Schnorr signatures

We now show that Schnorr's signature scheme is secure in the random oracle model, provided Schnorr's identification scheme is secure against eavesdropping attacks.

Theorem 19.7. *If H is modeled as a random oracle and Schnorr's identification scheme is secure against eavesdropping attacks, then Schnorr's signature scheme is also secure.*

> *In particular, let $\mathcal{A}$ be an adversary attacking $\mathcal{S}_{\mathrm{sch}}$ as in the random oracle version of Attack Game 13.1. Moreover, assume that $\mathcal{A}$ issues at most Q_{s} signing queries and Q_{ro} random oracle queries. Then there exists a $(Q_{\mathrm{ro}} + 1)$-impersonation adversary $\mathcal{B}$ that attacks $\mathcal{I}_{\mathrm{sch}}$ via an eavesdropping attack as in Attack Game 19.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{sch}}] \leq Q_{\mathrm{s}}(Q_{\mathrm{s}} + Q_{\mathrm{ro}} + 1)/q + \mathrm{rID2}\mathsf{adv}[\mathcal{B}, \mathcal{I}_{\mathrm{sch}}, Q_{\mathrm{ro}} + 1]. \tag{19.5}$$

Proof idea. The goal is to convert an adversary $\mathcal{A}$ that forges a signature into an adversary $\mathcal{B}$ that breaks the security of Schnorr's identification scheme in an r-impersonation eavesdropping attack, where $r := Q_{\mathrm{ro}} + 1$.

The first idea is that we have to somehow answer $\mathcal{A}$'s signing queries without using the secret key. This is done by using the transcripts from eavesdropped conversations to build the required signatures, "fixing up" the random oracle representing H to be consistent with these signatures. This "fixing up" will fail only if the random oracle needs to be queried at a point at which it has already been queried. But since the input to the random oracle includes a random group element, this is unlikely to happen. This is where the term $Q_{\mathrm{s}}(Q_{\mathrm{s}} + Q_{\mathrm{ro}} + 1)/q$ in (19.5) arises.

Once we have gotten rid of the signing queries, we argue that if the adversary successfully forges a signature, he can be effectively used in an r-impersonation attack on $\mathcal{I}_{\mathrm{sch}}$. Again, we exploit the fact that H is modeled as a random oracle. Since a signature forgery must be on a message not submitted as a signing query, the corresponding random oracle query must be at a point distinct from all those made by any of the signing queries, and so the value of the random oracle at that point essentially acts as a random challenge in a run of the identification protocol. We do not know in advance which random oracle query will correspond to the forgery, which is why we have to use the r-impersonation attack game. □

Proof. To simplify the analysis, we shall assume that when $\mathcal{A}$ outputs a forgery pair (m, σ), where $\sigma = (u_{\rm t}, \alpha_{\rm z})$, then $\mathcal{A}$ must have already explicitly queried the random oracle at the point $(m, u_{\rm t})$. If necessary, we modify $\mathcal{A}$ to ensure that this is the case, so that the total number of random oracle queries made by the modified version of $\mathcal{A}$ is at most $Q_{\rm ro} + 1$.

We define two attack games. Game 0 is essentially the original signature attack game, with H modeled as a random oracle. Game 1 is a slight modification. For $j = 0, 1$, W_j is the event that $\mathcal{A}$ wins in Game j.

Game 0. The challenger works as in the random oracle version of Attack Game 13.1. As usual, we implement the random oracle using an associative array $Map : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$. We also maintain an associative array $Explicit : \mathcal{M} \times \mathbb{G} \to \mathbb{Z}$ that keeps track of those points at which the random oracle was first queried explicitly by the adversary, rather than (implicitly) by the signing algorithm. The logic of the challenger is shown in Fig. 19.3.

To process a signing query m_i, the challenger runs the signing algorithm as usual: first it generates a random $\alpha_{{\rm t}i} \in \mathbb{Z}_q$ and computes $u_{{\rm t}i} \leftarrow g^{\alpha_{{\rm t}i}}$; it then generates a random "default" value $c_i \in \mathcal{C}$ for the value of $Map[m_i, u_{{\rm t}i}]$; if the test in the line marked (1) detects that $Map[m_i, u_{{\rm t}i}]$ was already defined, then that previously defined value is used, instead of the default value.

To process a random oracle query $(\widehat{m}_j, \widehat{u}_j)$, if the value $Map[\widehat{m}_j, \widehat{u}_j]$ has not already been defined, by either a previous signing or random oracle query, then it is defined here, and in addition, we set $Explicit[\widehat{m}_j, \widehat{u}_j] \leftarrow j$.

Suppose that the adversary submits $(m, u_{\rm t}, \alpha_{\rm z})$ as its forgery attempt, and that m is different from all the m_i's submitted as signing queries. By our simplifying assumption, the adversary must have previously submitted $(m, u_{\rm t})$ as a random oracle query, and it must be the case that $(m, u_{\rm t})$ is in Domain($Explicit$) at that point. It follows that if $(u_{\rm t}, \alpha_{\rm z})$ is a valid signature, then the challenger will output "win" and therefore

$$\mathrm{SIG^{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\rm sch}] \leq \Pr[W_0].$$

Game 1. This is the same as Game 0, except that the line marked (1) in Fig. 19.3 is deleted. By a straightforward application of the Difference Lemma, we obtain

$$|\Pr[W_1] - \Pr[W_0]| \leq Q_{\rm s}(Q_{\rm s} + Q_{\rm ro} + 1)/q.$$

Indeed, for the ith signing query, $u_{{\rm t}i}$ is uniformly distributed over $\mathbb{G}$, the union bound implies that the probability that the random oracle was previously queried at the point $(m, u_{{\rm t}i})$ (either directly by the adversary, or indirectly via a previous signing query) is at most $(Q_{\rm s} + Q_{\rm ro} + 1)/q$. Another application of the union bound gives the overall bound $Q_{\rm s}(Q_{\rm s} + Q_{\rm ro} + 1)/q$ on the probability that this occurs for any signing query.

The point of making this change is that now in Game 1, a fresh random challenge is used to process each signing query, just as an honest verifier in Schnorr's identification protocol.

At this point, it is easy to construct an adversary $\mathcal{B}$ that plays the r-impersonation eavesdropping attack game with $r = Q_{\rm ro} + 1$ against a challenger, and itself plays the role of challenger to $\mathcal{A}$ in Game 1, so that

$$\Pr[W_1] = \mathrm{rID2}\mathsf{adv}[\mathcal{B}, \mathcal{I}_{\rm sch}, r].$$

The detailed logic of $\mathcal{B}$ is shown in Fig. 19.4. Here, for $j = 1, \ldots, r$, we denote by V_j the jth verifier in the r-impersonation attack game. The theorem now follows immediately. □

```
initialization:
    $\alpha \overset{R}{\leftarrow} \mathbb{Z}_q$, $u \leftarrow g^{\alpha}$
    initialize empty associative arrays $Map : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$ and
        $Explicit : \mathcal{M} \times \mathbb{G} \to \mathbb{Z}$
    send the public key $u$ to $\mathcal{A}$;

upon receiving the $i$th signing query $m_i \in \mathcal{M}$:
    $\alpha_{\mathrm{t}i} \overset{R}{\leftarrow} \mathbb{Z}_q$, $u_{\mathrm{t}i} \leftarrow g^{\alpha_{\mathrm{t}i}}$, $c_i \overset{R}{\leftarrow} \mathcal{C}$
(1) if $(m_i, u_{\mathrm{t}i}) \in \mathrm{Domain}(Map)$ then $c_i \leftarrow Map[m_i, u_{\mathrm{t}i}]$
    $Map[m_i, u_{\mathrm{t}i}] \leftarrow c_i$
    $\alpha_{\mathrm{z}i} \leftarrow \alpha_{\mathrm{t}i} + \alpha c_i$
    send $(u_{\mathrm{t}i}, \alpha_{\mathrm{z}i})$ to $\mathcal{A}$;

upon receiving the $j$th random oracle query $(\widehat{m}_j, \widehat{u}_j) \in \mathcal{M} \times \mathbb{G}$:
    if $(\widehat{m}_j, \widehat{u}_j) \notin \mathrm{Domain}(Map)$ then
        $Map[\widehat{m}_j, \widehat{u}_j] \overset{R}{\leftarrow} \mathcal{C}$,   $Explicit[\widehat{m}_j, \widehat{u}_j] \leftarrow j$
    send $Map[\widehat{m}_j, \widehat{u}_j]$ to $\mathcal{A}$;

upon receiving a forgery attempt $(m, u_{\mathrm{t}}, \alpha_{\mathrm{z}})$:
    // By assumption, $(m, u_{\mathrm{t}}) \in Domain(Explicit)$
    if $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^{c}$ where $c = Map[m, u_{\mathrm{t}}]$
        then output "win"
        else  output "lose"
```

Figure 19.3: Game 0 challenger

```
initialization:
    obtain the verification key $u$ from challenger
    obtain eavesdropped conversations $(u_{\mathrm{t}i}, c_i, \alpha_{\mathrm{z}i})$ for $i = 1, \ldots, Q_{\mathrm{s}}$ from challenger
    initialize empty associative arrays $\mathit{Map} : \mathcal{M} \times \mathbb{G} \to \mathcal{C}$ and
        $\mathit{Explicit} : \mathcal{M} \times \mathbb{G} \to \mathbb{Z}$
    send $u$ to $\mathcal{A}$;

upon receiving the $i$th signing query $m_i \in \mathcal{M}$ from $\mathcal{A}$:
    $\mathit{Map}[m_i, u_{\mathrm{t}i}] \leftarrow c_i$
    send $(u_{\mathrm{t}i}, \alpha_{\mathrm{z}i})$ to $\mathcal{A}$;

upon receiving the $j$th random oracle query $(\widehat{m}_j, \widehat{u}_j) \in \mathcal{M} \times \mathbb{G}$:
    if $(\widehat{m}_j, \widehat{u}_j) \notin \mathrm{Domain}(\mathit{Map})$ then
        initiate an impersonation attempt with verifier $V_j$:
            send $\widehat{u}_j$ to $V_j$, who responds with a challenge $\widehat{c}_j$
        $\mathit{Map}[\widehat{m}_j, \widehat{u}_j] \leftarrow \widehat{c}_j, \quad \mathit{Explicit}[\widehat{m}_j, \widehat{u}_j] \leftarrow j$
    send $\mathit{Map}[\widehat{m}_j, \widehat{u}_j]$ to $\mathcal{A}$;

upon receiving a forgery attempt $(m, u_{\mathrm{t}}, \alpha_{\mathrm{z}})$:
    // By assumption, $(m, u_{\mathrm{t}}) \in \mathit{Domain}(\mathit{Explicit})$
    send the final message $\alpha_{\mathrm{z}}$ to $V_j$, where $j = \mathit{Explicit}[m, u_{\mathrm{t}}]$
```

Figure 19.4: Adversary $\mathcal{B}$

Putting it all together. If we string together the results of Theorem 19.7, Lemma 19.6, and Theorems 19.5 and 19.1, we get the following reduction from attacking the Schnorr signature scheme to computing discrete-log:

> *Let $\mathcal{A}$ be an efficient adversary attacking $\mathcal{S}_{\text{sch}}$ as in the random oracle version of Attack Game 13.1. Moreover, assume that $\mathcal{A}$ issues at most Q_s signing queries and Q_{ro} random oracle queries. Then there exists an efficient DL adversary $\mathcal{B}$ (whose running time is about* twice *that of $\mathcal{A}$), such that*
>
> $$\text{SIG}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{sch}}] \le \frac{Q_\text{s}(Q_\text{s} + Q_{\text{ro}} + 1)}{q} + \frac{Q_{\text{ro}} + 1}{N} + (Q_{\text{ro}} + 1)\sqrt{\text{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}]}, \qquad (19.6)$$
>
> *where N is the size of the challenge space.*

This reduction is very loose. The scalar $(Q_{\text{ro}} + 1)$ multiplying the term $\sqrt{\text{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}]}$ is the most problematic. It turns out that we can get a somewhat tighter reduction, essentially replacing $(Q_{\text{ro}} + 1)$ by $\sqrt{(Q_{\text{ro}} + 1)}$, which is much better. The trick is to combine the "guessing step" made in Lemma 19.6 and the "rewinding step" made in Theorem 19.1 into a single, direct reduction.

Lemma 19.8. *Consider Schnorr's identification protocol $\mathcal{I}_{\text{sch}}$, defined with respect to a group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$, and with a challenge space $\mathcal{C}$ of size N. For every efficient r-impersonation eavesdropping adversary $\mathcal{A}$ attacking $\mathcal{I}_{\text{sch}}$, with advantage $\epsilon := \text{rID2}\mathsf{adv}[\mathcal{A}, \mathcal{I}, r]$, there exists an efficient DL adversary $\mathcal{B}$ (whose running time is about* twice *that of $\mathcal{A}$), with advantage $\epsilon' := \text{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}]$, such that*

$$\epsilon' \ge \epsilon^2/r - \epsilon/N, \qquad (19.7)$$

which implies

$$\epsilon \le \frac{r}{N} + \sqrt{r\epsilon'}. \qquad (19.8)$$

Proof. Let us begin by reviewing how $\mathcal{A}$'s attack game works. First, the challenger in Attack Game 19.1 gives to $\mathcal{A}$ a verification key $u \in \mathbb{G}$ for Schnorr's identification protocol. Second, the challenger gives to $\mathcal{A}$ several transcripts of conversations. Third, $\mathcal{A}$ enters the impersonation phase, where it attempts to make at least one of r verifiers accept. In more detail, this works as follows. For j running from 1 to at most r, $\mathcal{A}$ sends $u_{\text{t}j}$ to the challenger, who responds with a random challenge $c_j \in \mathcal{C}$. After receiving all of these challenges, $\mathcal{A}$ either outputs fail or a pair (i, α_z) such that $(u_{\text{t}i}, c_i, \alpha_\text{z})$ is an accepting conversation for the verification key u. In the latter case, we say $\mathcal{A}$ *succeeds at verifier i*. Observe that $\mathcal{A}$'s advantage is $\epsilon = \sum_{j=1}^{r} \epsilon_j$, where ϵ_j is the probability that $\mathcal{A}$ succeeds at verifier j.

Note that we have assumed a somewhat simplified behavior for the adversary in the impersonation phase. However, since the adversary can see for himself whether a conversation is accepting or not, this is not really a restriction: any adversary can be put in the form described without changing its advantage at all, and without increasing its running time significantly. (Also, the r-impersonation adversary constructed in the proof of Theorem 19.7 is already essentially of this form.)

We now describe our DL adversary $\mathcal{B}$, which is given $u \in \mathbb{G}$, and is tasked to compute $\mathsf{Dlog}_g u$. As usual, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$. First, $\mathcal{B}$ gives u to $\mathcal{A}$ as the verification key. Second, $\mathcal{B}$ generates transcripts of conversations, using the simulator from Theorem 19.4, and gives these to $\mathcal{A}$. Third, $\mathcal{B}$ lets $\mathcal{A}$ run through the impersonation phase to completion, supplying random challenges $c_1, \ldots, c_r$. If $\mathcal{A}$ outputs a pair (i, α_z) such that $(u_{\text{t}i}, c_i, \alpha_\text{z})$ is an accepting conversation for the verification key u, then $\mathcal{B}$ rewinds $\mathcal{A}$ back to the point where it submitted $u_{\mathrm{t}i}$ to the ith verifier. Instead of the challenge c_i, our adversary $\mathcal{B}$ responds with a fresh, random challenge $c' \in \mathcal{C}$. It then lets $\mathcal{A}$ run through the remainder of the impersonation phase, using the same challenge c_j for $j = i+1, \ldots, r$. If $\mathcal{A}$ outputs a pair $(i', \alpha'_{\mathrm{z}})$ such that $i' = i$, $(u_{\mathrm{t}i}, c', \alpha'_{\mathrm{z}})$ is an accepting conversation, and $c' \neq c_i$, then $\mathcal{B}$ uses these two accepting conversations to compute $\mathsf{Dlog}_g u$, just as we did in the proof of Theorem 19.1. In this case, we say $\mathcal{B}$ *succeeds at verifier i*. Observe that $\mathcal{B}$'s advantage is $\epsilon' = \sum_{j=1}^{r} \epsilon'_j$, where ϵ'_j is the probability that $\mathcal{B}$ succeeds at verifier j.

It remains to prove (19.7) — note that (19.8) follows from (19.7) using a calculation almost identical to that used in the proof of Theorem 19.1.

We claim that for $j = 1, \ldots, r$, we have

$$\epsilon'_j \geq \epsilon_j^2 - \epsilon_j/N. \tag{19.9}$$

Indeed, for a fixed index j, this inequality follows from an application of the rewinding lemma (Lemma 19.2), where Y corresponds to the challenge c_j, Y' corresponds to the challenge c', and X corresponds to all the other random choices made by $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{B}$'s challenger. The function f in the lemma is defined to be 1 if $\mathcal{A}$ succeeds at verifier j. So $f(\mathsf{X}, \mathsf{Y}) = 1$ if $i = j$ and $(u_{\mathrm{t}j}, c_j, \alpha_{\mathrm{z}})$ is an accepting conversation; similarly, $f(\mathsf{X}, \mathsf{Y}') = 1$ if $i' = j$ and $(u_{\mathrm{t}j}, c', \alpha'_{\mathrm{z}})$ is an accepting conversation.

From (19.9), we obtain

$$\epsilon' = \sum_{j=1}^{r} \epsilon'_j \geq \sum_{j=1}^{r} \epsilon_j^2 - \sum_{j=1}^{r} \epsilon_j/N \geq \epsilon^2/r - \epsilon/N,$$

where for the last inequality, we used the fact that for any function $g : \{1, \ldots, r\} \to \mathbb{R}$, we have

$$\sum_{j=1}^{r} g(j)^2 \geq \Big(\sum_{j=1}^{r} g(j)\Big)^2 / r.$$

This follows, for example, from the fact that $E[\mathsf{Z}^2] \geq E[\mathsf{Z}]^2$ for any random variable Z, and in particular, for $\mathsf{Z} := g(\mathsf{R})$, where R is uniformly distributed over $\{1, \ldots, r\}$. □

With this result, we can replace the bound (19.6) by:

$$\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\mathrm{sch}}] \leq \frac{Q_{\mathrm{s}}(Q_{\mathrm{s}} + Q_{\mathrm{ro}} + 1)}{q} + \frac{Q_{\mathrm{ro}} + 1}{N} + \sqrt{(Q_{\mathrm{ro}} + 1)\mathrm{DL}\mathsf{adv}[\mathcal{B}, \mathbb{G}]}. \tag{19.10}$$

19.2.3 A concrete implementation and an optimization

We might take $\mathbb{G}$ to be the elliptic curve group P256 defined over a finite field $\mathbb{F}_p$ where p is a 256-bit prime (Section 15.3). It will be sufficient to work with 128-bit challenges. In this case each component of the Schnorr signature $(u_{\mathrm{t}}, \alpha_{\mathrm{z}})$ is 256 bits. Overall, a Schnorr signature is about 512 bits.

Because the length of a challenge is much shorter than the encoding length of a group element, the following "optimized" variant of Schnorr's signature scheme can be used to obtain much shorter signatures. Instead of defining a signature on m to be a pair $(u_{\mathrm{t}}, \alpha_{\mathrm{z}})$ satisfying

$$g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c,$$

where $c := H(m, u_t)$, we can define it to be a pair (c, α_z) satisfying

$$c = H(m, u_t),$$

where $u_t := g^{\alpha_z}/u^c$. The transformation $(u_t, \alpha_z) \mapsto (H(m, u_t), \alpha_z)$ maps a regular Schnorr signature on m to an optimized Schnorr signature, while the transformation $(c, \alpha_z) \mapsto (g^{\alpha_z}/u^c, \alpha_z)$ maps an optimized Schnorr signature to a regular Schnorr signature. It follows that forging an optimized Schnorr signature is equivalent to forging a regular Schnorr signature. As a further optimization, one can store u^{-1} in the public key instead of u, which will speed up verification.

With the above choices of parameters, we reduce the length of a signature from 512 bits to about $128 + 256 = 384$ bits — a 25% reduction in size.

19.3 Case study: ECDSA signatures

In 1991, when it came time to adopt a federal standard for digital signatures, the National Institute of Standards (NIST) considered a number of viable candidates. Because the Schnorr system was protected by a patent, NIST opted for a more ad-hoc signature scheme based on a prime-order subgroup of $\mathbb{Z}_p^*$ that eventually became known as the **Digital Signature Algorithm** or DSA. The standard was later updated to support elliptic curve groups defined over a finite field. The resulting signature scheme, called **ECDSA**, is used in many real-world systems. We briefly describe how ECDSA works and discuss some security issues that affect it.

The ECDSA signature scheme (G, S, V) uses the group of points $\mathbb{G}$ of an elliptic curve over a finite field $\mathbb{F}_p$. Let g be a generator of $\mathbb{G}$ and let q be the order of the group $\mathbb{G}$, which we assume is prime. We will use multiplicative notation for the group operation. We will also need a hash function H defined over $(\mathcal{M}, \mathbb{Z}_q^*)$.

The scheme works as follows:

- $G()$: Choose $\alpha \xleftarrow{R} \mathbb{Z}_q^*$ and set $u \leftarrow g^\alpha \in \mathbb{G}$. Output $sk := \alpha$ and $pk := u$.
- $S(sk, m)$: To sign a message $m \in \mathcal{M}$ with secret key $sk = \alpha$ do:

```
repeat:
    $\alpha_t \xleftarrow{R} \mathbb{Z}_q^*, \quad u_t \leftarrow g^{\alpha_t}$
    let $u_t = (x, y) \in \mathbb{G}$ where $x, y \in \mathbb{F}_p$
    treat $x$ as an integer in $[0, p)$ and set $r \leftarrow [x]_q \in \mathbb{Z}_q$   // reduce $x$ modulo $q$
    $s \leftarrow \big(H(m) + r\alpha\big)/\alpha_t \in \mathbb{Z}_q$
until $r \neq 0$ and $s \neq 0$
output $(r, s) \in \mathbb{Z}_q^2$
```

- $V(pk, m, \sigma)$: To verify a signature $\sigma = (r, s) \in \mathbb{Z}_q^2$ on $m \in \mathcal{M}$ with $pk = u \in \mathbb{G}$ do:

```
if $r = 0$ or $s = 0$ then output reject and stop
$a \leftarrow H(m)/s \in \mathbb{Z}_q, \quad b \leftarrow r/s \in \mathbb{Z}_q$
$\hat{u}_t \leftarrow g^a u^b \in \mathbb{G}$
if $\hat{u}_t$ is the point at infinity in $\mathbb{G}$ then output reject and stop
let $\hat{u}_t = (\hat{x}, \hat{y}) \in \mathbb{G}$ where $\hat{x}, \hat{y} \in \mathbb{F}_p$
treat $\hat{x}$ as an integer in $[0, p)$ and set $\hat{r} \leftarrow [\hat{x}]_q \in \mathbb{Z}_q$   // reduce $\hat{x}$ modulo $q$
if $r = \hat{r}$ output accept; else output reject
```

When using the elliptic curve P256, both p and q are 256-bit primes. An ECDSA signature $\sigma = (r, s)$ is then 512 bits long.

A straightforward calculation shows that the scheme is correct: for every key pair (pk, sk) output by G, and every message $m \in \mathcal{M}$, if $\sigma \xleftarrow{R} S(sk, m)$ then $V(pk, m, \sigma)$ outputs accept. The reason is that $\hat{u}_t$ computed by V is the same as u_t computed by S.

The security of this scheme has only been established somewhat heuristically, specifically, when we model $\mathbb{G}$ as a "generic group" (see Section 16.3).

For security, it is important that the random value α_t generated during signing be a fresh uniform value in $\mathbb{Z}_q^*$. Otherwise the scheme can become insecure in a strong sense: an attacker can learn the secret signing key α. This was used in a successful attack on the Sony PlayStation 3 because α_t was the same for all issued signatures. It has also lead to attacks on some Bitcoin wallets [48]. Because generating randomness on some hardware platforms can be difficult, a common solution is to modify the signing algorithm so that α is generated deterministically using a secure PRF, as described in Exercise 13.6. This variant is called **deterministic ECDSA**. The Schnorr signature scheme suffers from the same issue and this modification applies equally well to it.

ECDSA is not strongly secure. While the Schnorr signature scheme is strongly secure (see Exercise 19.17), the ECDSA scheme is not. Given an ECDSA signature $\sigma = (r, s)$ on a message m, anyone can generate more signatures on m. For example, $\sigma' := (r, -s) \in (\mathbb{Z}_q^*)^2$ is another valid signature on m. This σ' is valid because the x-coordinate of the elliptic curve point $u_t \in \mathbb{G}$ is the same as the x-coordinate of the point $1/u_t \in \mathbb{G}$.

19.4 Sigma protocols: basic definitions

Schnorr's identification protocol is a special case of an incredibly useful class of protocols called **Sigma protocols**. In this section, we will introduce the basic concepts associated with Sigma protocols. Later, we will consider many examples of Sigma protocols and their applications:

- We will see how we can use Sigma protocols to build new secure identification schemes and signature schemes.

- We will see how to build identification schemes that we can prove (without the random oracle heuristic) are secure against *active* attacks. Recall that for Schnorr's identification protocol we could only prove security against eavesdropping attacks.

- In the next chapter, we will also see how to use Sigma protocols for other applications that have nothing to do with identification and signatures. For example, we will see how one can encrypt a message m and then "prove" to a skeptical verifier that m satisfies certain properties, without revealing to the verifier anything else about m. We will illustrate this idea with an electronic voting protocol.

Consider again Schnorr's identification protocol. Intuitively, that protocol allows a prover P to convince a skeptical verifier V that he knows a secret that satisfies some relation, without revealing any useful information to V about the secret. For Schnorr's protocol, the prover's secret was $\alpha \in \mathbb{Z}_q$ satisfying the relation $g^\alpha = u$.

We can generalize this to more general and interesting types of relations.

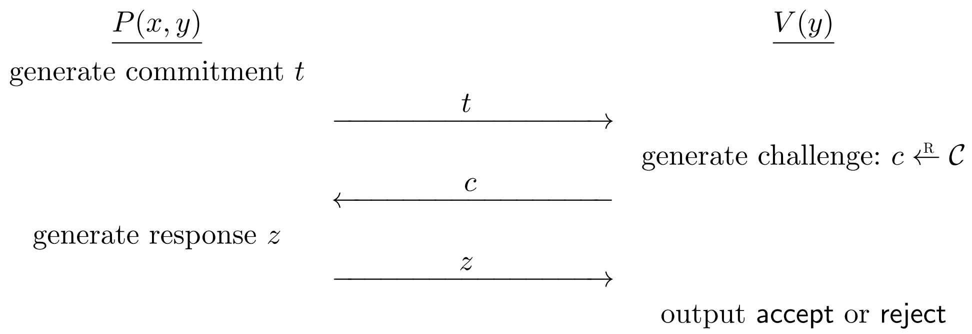


Figure 19.5: Execution of a Sigma protocol

Definition 19.2 (Effective relation). *An **effective relation** is a binary relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{R}$ are efficiently recognizable finite sets. Elements of $\mathcal{Y}$ are called **statements**. If $(x, y) \in \mathcal{R}$, then x is called a **witness for** y.*

We now define the syntax of a Sigma protocol.

Definition 19.3 (Sigma protocol). *Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be an effective relation. A **Sigma protocol** for $\mathcal{R}$ is a pair (P, V).*

- *P is an interactive protocol algorithm called the **prover**, which takes as input a witness-statement pair $(x, y) \in \mathcal{R}$.*
- *V is an interactive protocol algorithm called the **verifier**, which takes as input a statement $y \in \mathcal{Y}$, and which outputs* accept *or* reject*.*
- *P and V are structured so that an interaction between them always works as follows:*
    - *To start the protocol, P computes a message t, called the **commitment**, and sends t to V;*
    - *Upon receiving P's commitment t, V chooses a **challenge** c at random from a finite **challenge space** $\mathcal{C}$, and sends c to P;*
    - *Upon receiving V's challenge c, P computes a **response** z, and sends z to V;*
    - *Upon receiving P's response z, V outputs either* accept *or* reject*, which must be computed strictly as a function of the statement y and the **conversation** (t, c, z). In particular, V does not make any random choices other than the selection of the challenge — all other computations are completely deterministic.*

We require that for all $(x, y) \in \mathcal{R}$, when $P(x, y)$ and $V(y)$ interact with each other, $V(y)$ always outputs accept*.*

See Fig. 19.5, which illustrates the execution of a Sigma protocol. The name Sigma protocol comes from the fact that the "shape" of the message flows in such a protocol is vaguely reminiscent of the shape of the Greek letter Σ.

As stated in the definition, we require that the verifier computes its output as a function of the statement y and its conversation (t, c, z) with the prover. If the output is accept we call the conversation (t, c, z) an **accepting conversation for** y. Of course, interactions between the verifier and an honest prover only produce accepting conversations; non-accepting conversation can arise, for example, if the verifier interacts with a "dishonest" prover that is not following the protocol.

In most applications of Sigma protocols, we will require that the size of the challenge space is super-poly. To state this requirement more succinctly, we will simply say that the protocol has a **large challenge space**.

Example 19.1. It should be clear that for Schnorr's identification protocol (G, P, V), the pair (P, V) is an example of a Sigma protocol for the relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, where

$$\mathcal{X} = \mathbb{Z}_q, \quad \mathcal{Y} = \mathbb{G}, \quad \text{and} \quad \mathcal{R} = \{ \ (\alpha, u) \in \mathbb{Z}_q \times \mathbb{G} : \ g^\alpha = u \ \}.$$

The challenge space $\mathcal{C}$ is a subset of $\mathbb{Z}_q$. We call (P, V) **Schnorr's Sigma protocol**.

The reader should observe that unlike an identification protocol, a Sigma protocol itself does not specify an algorithm for generating elements of $\mathcal{R}$.

Note also that the relation $\mathcal{R}$ in this case is parameterized by a description of the group $\mathbb{G}$ (which includes its order q and the generator $g \in \mathbb{G}$). In general, we allow effective relations that are defined in terms of such "system parameters," which are assumed to be generated at system setup time, and publicly known to all parties.

A statement for Schnorr's Sigma protocol is a group element $u \in \mathbb{G}$, and a witness for u is $\alpha \in \mathbb{Z}_q$ such that $g^\alpha = u$. Thus, every statement has a unique witness. An accepting conversation for u is a triple of the form $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$, with $u_{\mathrm{t}} \in \mathbb{G}$, $c \in \mathcal{C}$, and $\alpha_{\mathrm{z}} \in \mathbb{Z}_q$, that satisfies the equation

$$g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c.$$

The reader may have noticed that, as we have defined it, the prover P from Schnorr's identification protocol takes as input just the witness α, rather than the witness/statement pair (α, u), as formally required in our definition of a Sigma protocol. In fact, in this and many other examples of Sigma protocols, the prover does not actually use the statement explicitly in its computation. □

19.4.1 Special soundness

We next define a critical security property for Sigma protocols, which is called **special soundness**.

Definition 19.4 (Special soundness). *Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. We say that (P, V) provides* ***special soundness*** *if there is an efficient deterministic algorithm Ext, called a* ***witness extractor****, with the following property: whenever Ext is given as input a statement $y \in \mathcal{Y}$, and two accepting conversations (t, c, z) and (t, c', z'), with $c \neq c'$, algorithm Ext always outputs $x \in \mathcal{X}$ such that $(x, y) \in \mathcal{R}$ (i.e., x is a witness for y).*

Example 19.2. Continuing with Example 19.1, we can easily verify that Schnorr's Sigma protocol provides special soundness. The witness extractor takes as input the statement $u \in \mathbb{G}$, along with two accepting conversations $(u_{\mathrm{t}}, c, \alpha_{\mathrm{z}})$ and $(u_{\mathrm{t}}, c', \alpha'_{\mathrm{z}})$ for u, with $c \neq c'$. Just as we did in the proof of Theorem 19.1, we can compute the corresponding witness $\alpha = \mathsf{Dlog}_g u$ from these two conversations as $\Delta\alpha / \Delta c \in \mathbb{Z}_q$, where $\Delta\alpha := \alpha_{\mathrm{z}} - \alpha'_{\mathrm{z}}$ and $\Delta c := c - c'$. □

Suppose (P, V) is a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Moreover, suppose (P, V) provides special soundness and has a large challenge space. Then in a certain sense, (P, V) acts as a "proof of knowledge." Indeed, consider an arbitrary prover P^* (even a potentially "cheating" one) that makes V accept a statement y with non-negligible probability. Then P^* must "know" a witness for y, in the following sense: just as in the proof of Theorem 19.1, we can rewind P^* to get two accepting conversations (t, c, z) and (t, c', z') for y, with $c \neq c'$, and then use the witness extractor to compute the witness x.

More generally, when a cryptographer says that P^* must "know" a witness for a statement y, what she means is that the witness can be extracted from P^* using rewinding. Although we will not formally define the notion of a "proof of knowledge," we will apply special soundness in several applications.

19.4.2 Special honest verifier zero knowledge

We introduced the notion of honest verifier zero knowledge (HVZK) in Section 19.1.1 for identification protocols. We can easily adapt this notion to the context of Sigma protocols.

Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Intuitively, what we want to say is that for $(x, y) \in \mathcal{R}$, a conversation between $P(x, y)$ and $V(y)$ should not reveal anything about the witness x. Just as in Section 19.1.1, we will formalize this intuition by saying that we can efficiently simulate conversations between $P(x, y)$ and $V(y)$ without knowing the witness x. However, we will add a few extra requirements, which will streamline some constructions and applications.

Definition 19.5 (Special HVZK). *Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ with challenge space $\mathcal{C}$. We say that (P, V) is **special honest verifier zero knowledge**, or **special HVZK**, if there exists an efficient probabilistic algorithm Sim (called a **simulator**) that takes as input $(y, c) \in \mathcal{Y} \times \mathcal{C}$, and satisfies the following properties:*

(i) for all inputs $(y, c) \in \mathcal{Y} \times \mathcal{C}$, algorithm Sim always outputs a pair (t, z) such that (t, c, z) is an accepting conversation for y;

(ii) for all $(x, y) \in \mathcal{R}$, if we compute

$$c \xleftarrow{\text{R}} \mathcal{C}, \;\; (t, z) \xleftarrow{\text{R}} \mathit{Sim}(y, c),$$

then (t, c, z) has the same distribution as that of a transcript of a conversation between $P(x, y)$ and $V(y)$.

The reader should take note of a couple of features of this definition. First, the simulator takes the challenge c as an additional input. Second, it is required that the simulator produce an accepting conversation even when the statement y does not have a witness. These two properties are the reason for the word "special" in "special HVZK."

Example 19.3. Continuing with Example 19.2, we can easily verify that Schnorr's Sigma protocol is special HVZK. Indeed, the simulator in the proof of Theorem 19.4 is easily adapted to the present setting. On input $u \in \mathbb{G}$ and $c \in \mathcal{C}$, the simulator computes

$$\alpha_{\text{z}} \xleftarrow{\text{R}} \mathbb{Z}_q, \;\; u_{\text{t}} \leftarrow g^{\alpha_{\text{z}}}/u^c,$$

and outputs the pair $(u_{\text{t}}, \alpha_{\text{z}})$. We leave it to the reader to verify that this simulator satisfies all the requirements of Definition 19.5. □

19.5 Sigma protocols: examples

So far, the only Sigma protocol we have seen is that of Schnorr, which allows a prover to convince a skeptical verifier that it "knows" the discrete logarithm of a given group element, without revealing anything about the discrete logarithm to the verifier. In this section, we present several additional examples of Sigma protocols. These examples not only serve to flesh out the general theory of Sigma protocols, they also have many practical applications, some of which we will discuss below.

19.5.1 Okamoto's protocol for representations

Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $h \in \mathbb{G}$ be some arbitrary group element. We will think of h for now as a system parameter — generated once and for all at system setup time, and publicly available to all parties. Recall (see Section 10.6.1) that for $u \in \mathbb{G}$, a representation of u (relative to g and h) is a pair $(\alpha, \beta) \in \mathbb{Z}_q^2$ such that $g^\alpha h^\beta = u$.

Okamoto's protocol allows a prover to convince a skeptical verifier that he "knows" a representation of a given $u \in \mathbb{G}$, without revealing anything about that representation to the verifier. More precisely, it is a Sigma protocol for the relation

$$\mathcal{R} = \Big\{ \ \big(\ (\alpha, \beta),\ u\ \big) \in \mathbb{Z}_q^2 \times \mathbb{G} :\ g^\alpha h^\beta = u\ \Big\}. \tag{19.11}$$

A witness for the statement $u \in \mathbb{G}$ is $(\alpha, \beta) \in \mathbb{Z}_q^2$ such that $g^\alpha h^\beta = u$, i.e., a representation of u. Thus, in this example, every statement has many witnesses (precisely q, in fact).

The challenge space $\mathcal{C}$ for Okamoto's protocol is assumed to be a subset of $\mathbb{Z}_q$. The protocol (P, V) runs as follows, where the prover P is initialized with $((\alpha, \beta), u) \in \mathcal{R}$ and the verifier V is initialized with $u \in \mathbb{G}$:

1. P computes
$$\alpha_{\rm t} \xleftarrow{\rm R} \mathbb{Z}_q,\ \ \beta_{\rm t} \xleftarrow{\rm R} \mathbb{Z}_q,\ \ u_{\rm t} \leftarrow g^{\alpha_{\rm t}} h^{\beta_{\rm t}},$$
and sends the commitment $u_{\rm t}$ to V;

2. V computes $c \xleftarrow{\rm R} \mathcal{C}$, and sends the challenge c to P;

3. P computes
$$\alpha_{\rm z} \leftarrow \alpha_{\rm t} + \alpha c \in \mathbb{Z}_q,\ \ \beta_{\rm z} \leftarrow \beta_{\rm t} + \beta c \in \mathbb{Z}_q,$$
and sends the response $(\alpha_{\rm z}, \beta_{\rm z})$ to V;

4. V checks if $g^{\alpha_{\rm z}} h^{\beta_{\rm z}} = u_{\rm t} \cdot u^c$; if so V outputs accept; otherwise, V outputs reject.

See Fig. 19.6.

Theorem 19.9. *Okamoto's protocol is a Sigma protocol for the relation $\mathcal{R}$ defined in (19.11). Moreover, it provides special soundness and is special HVZK.*

Proof. Clearly, Okamoto's protocol has the required syntactic structure of a Sigma protocol. An accepting conversation for $u \in \mathbb{G}$ is of the form

$$(u_{\rm t}, c, (\alpha_{\rm z}, \beta_{\rm z})) \quad \text{such that} \quad g^{\alpha_{\rm z}} h^{\beta_{\rm z}} = u_{\rm t} \cdot u^c.$$

$$
\begin{array}{ccc}
\underline{P((\alpha,\beta),u)} & & \underline{V(u)} \\
\alpha_{\mathrm{t}} \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_q,\ \beta_{\mathrm{t}} \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_q,\ u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{t}}} h^{\beta_{\mathrm{t}}} & & \\
 & \xrightarrow{\qquad u_{\mathrm{t}} \qquad} & \\
 & & c \stackrel{\mathrm{R}}{\leftarrow} \mathcal{C} \\
 & \xleftarrow{\qquad c \qquad} & \\
\alpha_{\mathrm{z}} \leftarrow \alpha_{\mathrm{t}} + \alpha c & & \\
\beta_{\mathrm{z}} \leftarrow \beta_{\mathrm{t}} + \beta c & & \\
 & \xrightarrow{\qquad \alpha_{\mathrm{z}},\ \beta_{\mathrm{z}} \qquad} & \\
 & & g^{\alpha_{\mathrm{z}}} h^{\beta_{\mathrm{z}}} \stackrel{?}{=} u_{\mathrm{t}} \cdot u^c
\end{array}
$$

Figure 19.6: Okamoto's protocol

Correctness. We have to verify that the protocol satisfies the basic correctness requirement that an interaction between an honest prover and an honest verifier always produces an accepting conversation. This is easy to verify, since if

$$u_{\mathrm{t}} = g^{\alpha_{\mathrm{t}}} h^{\beta_{\mathrm{t}}}, \quad \alpha_{\mathrm{z}} = \alpha_{\mathrm{t}} + \alpha c, \quad \text{and} \quad \beta_{\mathrm{z}} = \beta_{\mathrm{t}} + \beta c,$$

then we have

$$g^{\alpha_{\mathrm{z}}} h^{\beta_{\mathrm{z}}} = g^{\alpha_{\mathrm{t}} + \alpha c} h^{\beta_{\mathrm{t}} + \beta c} = g^{\alpha_{\mathrm{t}}} h^{\beta_{\mathrm{t}}} \cdot (g^{\alpha} h^{\beta})^c = u_{\mathrm{t}} \cdot u^c.$$

Special soundness. Next, we show that Okamoto's protocol provides special soundness. Suppose we have two accepting conversations

$$(u_{\mathrm{t}}, c, (\alpha_{\mathrm{z}}, \beta_{\mathrm{z}})) \quad \text{and} \quad (u_{\mathrm{t}}, c', (\alpha'_{\mathrm{z}}, \beta'_{\mathrm{z}}))$$

for the statement u, where $c \neq c'$. We have to show how to efficiently extract a representation of u from these two conversations. The computation here is very similar to that in Schnorr's protocol. Observe that

$$g^{\alpha_{\mathrm{z}}} h^{\beta_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c \quad \text{and} \quad g^{\alpha'_{\mathrm{z}}} h^{\beta'_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^{c'},$$

and dividing the first equation by the second, the u_{t}'s cancel, and we have

$$g^{\Delta\alpha} h^{\Delta\beta} = u^{\Delta c}, \ \text{ where } \Delta\alpha := \alpha_{\mathrm{z}} - \alpha'_{\mathrm{z}}, \ \ \Delta\beta := \beta_{\mathrm{z}} - \beta'_{\mathrm{z}}, \quad \Delta c := c - c'.$$

and so the witness extractor can efficiently compute a representation $(\alpha, \beta) \in \mathbb{Z}_q^2$ of u as follows:

$$\alpha \leftarrow \Delta\alpha / \Delta c, \quad \beta \leftarrow \Delta\beta / \Delta c.$$

Note that because $c \neq c'$, the value Δc is invertible in $\mathbb{Z}_q$. Here we use the fact that q is a prime.

Special HVZK. Finally, we show that Okamoto's protocol is special HVZK by exhibiting a simulator. Again, this is very similar to what we did for Schnorr's protocol. On input $u \in \mathbb{G}$ and $c \in \mathcal{C}$, the simulator computes

$$\alpha_{\mathrm{z}} \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_q, \ \beta_{\mathrm{z}} \stackrel{\mathrm{R}}{\leftarrow} \mathbb{Z}_q, \ u_{\mathrm{t}} \leftarrow g^{\alpha_{\mathrm{z}}} h^{\beta_{\mathrm{z}}} / u^c,$$

$$\begin{array}{ccc}
\underline{P(\beta, (u,v,w))} & & \underline{V(u,v,w)} \\
\beta_{\mathrm{t}} \stackrel{\scriptscriptstyle \mathrm{R}}{\leftarrow} \mathbb{Z}_q,\ v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{t}}},\ w_{\mathrm{t}} \leftarrow u^{\beta_{\mathrm{t}}} & & \\
 & \xrightarrow{\qquad v_{\mathrm{t}}, w_{\mathrm{t}} \qquad} & \\
 & & c \stackrel{\scriptscriptstyle \mathrm{R}}{\leftarrow} \mathcal{C} \\
 & \xleftarrow{\qquad c \qquad} & \\
\beta_{\mathrm{z}} \leftarrow \beta_{\mathrm{t}} + \beta c & & \\
 & \xrightarrow{\qquad \beta_{\mathrm{z}} \qquad} & \\
 & & g^{\beta_{\mathrm{z}}} \stackrel{?}{=} v_{\mathrm{t}} \cdot v^c \text{ and } u^{\beta_{\mathrm{z}}} \stackrel{?}{=} w_{\mathrm{t}} \cdot w^c
\end{array}$$

Figure 19.7: The Chaum-Pedersen protocol

and outputs $(u_{\mathrm{t}}, (\alpha_{\mathrm{z}}, \beta_{\mathrm{z}}))$. Observe that the output always yields an accepting conversation, as required.

Now we argue that when $c \in \mathcal{C}$ is chosen at random, the output of the simulator on input u, c has the right distribution. The key observation is that in a real conversation, c, α_{z}, and β_{z} are mutually independent, with c uniformly distributed over $\mathcal{C}$, and α_{z} and β_{z} both uniformly distributed over $\mathbb{Z}_q$; moreover, given c, α_{z}, and β_{z}, the value u_{t} is uniquely determined by the equation

$$g^{\alpha_{\mathrm{z}}} h^{\beta_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c.$$

It should be clear that this is the same as the output distribution of the simulator. □

19.5.2 The Chaum-Pedersen protocol for DH-triples

The Chaum-Pedersen protocol allows a prover to convince a skeptical verifier that a given triple is a DH-triple, without revealing anything else to the verifier.

Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$, as usual. Recall (see Section 10.5) that for $\alpha, \beta, \gamma \in \mathbb{Z}_q$, we say that $(g^\alpha, g^\beta, g^\gamma)$ is a DH-triple if $\gamma = \alpha\beta$. Equivalently, (u, v, w) is a DH-triple if and only if there exists $\beta \in \mathbb{Z}_q$ such that $v = g^\beta$ and $w = u^\beta$.

The Chaum-Pedersen protocol is a Sigma protocol for the relation

$$\mathcal{R} := \Big\{ \ \big(\ \beta,\ (u, v, w)\ \big) \in \mathbb{Z}_q \times \mathbb{G}^3 :\ v = g^\beta \text{ and } w = u^\beta \ \Big\}. \tag{19.12}$$

A witness for the statement $(u, v, w) \in \mathbb{G}^3$ is $\beta \in \mathbb{Z}_q$ such that $v = g^\beta$ and $w = u^\beta$. Thus, a statement has a witness if and only if it is a DH-triple. Unlike the other examples we have seen so far, not all statements have a witness.

The Chaum-Pedersen protocol (P, V) is given in Fig. 19.7. The challenge space $\mathcal{C}$ is a subset of $\mathbb{Z}_q$.

Theorem 19.10. *The Chaum-Pedersen protocol is a Sigma protocol for the relation $\mathcal{R}$ defined in (19.12). Moreover, it provides special soundness and is special HVZK.*

Proof. The protocol has the required syntactic structure of a Sigma protocol. An accepting conversation for $(u, v, w) \in \mathbb{G}^3$ is of the form

$$((v_{\mathrm{t}}, w_{\mathrm{t}}), c, \beta_{\mathrm{z}}) \quad \text{such that} \quad g^{\beta_{\mathrm{z}}} = v_{\mathrm{t}} \cdot v^c \text{ and } u^{\beta_{\mathrm{z}}} = w_{\mathrm{t}} \cdot w^c.$$

We leave it to the reader to verify that an interaction between an honest prover and an honest verifier always produces an accepting conversation.

Special soundness. Suppose we have two accepting conversations

$$((v_{\mathrm{t}}, w_{\mathrm{t}}), c, \beta_{\mathrm{z}}) \quad \text{and} \quad ((v_{\mathrm{t}}, w_{\mathrm{t}}), c', \beta'_{\mathrm{z}})$$

for the statement (u, v, w), where $c \neq c'$. The reader may verify that

$$\beta := \Delta\beta/\Delta c, \quad \text{where } \Delta\beta := \beta_{\mathrm{z}} - \beta'_{\mathrm{z}},\ \Delta c := c - c',$$

is the corresponding witness.

Special HVZK. On input $(u, v, w) \in \mathbb{G}^3$ and $c \in \mathcal{C}$, the simulator computes

$$\beta_{\mathrm{z}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q,\ \ v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{z}}}/v^c,\ \ w_{\mathrm{t}} \leftarrow u^{\beta_{\mathrm{z}}}/w^c.$$

and outputs $((v_{\mathrm{t}}, w_{\mathrm{t}}), \beta_{\mathrm{z}})$. Observe that the output always yields an accepting conversation, as required.

Now we argue that when $c \in \mathcal{C}$ is chosen at random, the output of the simulator on input $((u, v, w), c)$ has the right distribution. The key observation is that in a real conversation, c and β_{z} are independent, with c uniformly distributed over $\mathcal{C}$ and β_{z} uniformly distributed over $\mathbb{Z}_q$; moreover, given c and β_{z}, the values v_{t} and w_{t} are uniquely determined by the equations

$$g^{\beta_{\mathrm{z}}} = v_{\mathrm{t}} \cdot v^c \text{ and } u^{\beta_{\mathrm{z}}} = w_{\mathrm{t}} \cdot w^c.$$

It should be clear that this is the same as the output distribution of the simulator. □

19.5.3 A Sigma protocol for arbitrary linear relations

The reader may have noticed a certain similarity among the Schnorr, Okamoto, and Chaum-Pedersen protocols. In fact, they are all special cases of a generic Sigma protocol for proving linear relations among group elements.

As usual, let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. We shall consider boolean formulas ϕ of the following type:

$$\phi(x_1, \ldots, x_n) \quad := \quad \left\{ \prod_{j=1}^{n} g_{1j}^{x_j} = u_1 \quad \wedge \quad \cdots \quad \wedge \quad \prod_{j=1}^{n} g_{mj}^{x_j} = u_m \right\}. \tag{19.13}$$

In such a formula ϕ, the g_{ij}'s and u_i's are elements of the group $\mathbb{G}$. Some of these group elements could be system parameters or even constants, while others are specific to the formula. The x_i's are the formal variables of the formula. When we assign values in $\mathbb{Z}_q$ to the variables $x_1, \ldots, x_n$, the formula evaluates to true if all the equalities in (19.13) hold.

For a specific class $\mathcal{F}$ of such formulas, we can define the relation

$$\mathcal{R} := \Big\{ \ \big(\ (\alpha_1, \ldots, \alpha_n),\ \phi\ \big) \in \mathbb{Z}_q^n \times \mathcal{F} :\ \phi(\alpha_1, \ldots, \alpha_n) = \text{true} \ \Big\}. \tag{19.14}$$

$$\begin{array}{lcl}
\underline{P((\alpha_1,\ldots,\alpha_n),\phi)} & & \underline{V(\phi)} \\
\alpha_{\mathrm{t}j} \xleftarrow{\mathrm{R}} \mathbb{Z}_q \quad (j=1,\ldots,n) & & \\
u_{\mathrm{t}i} \leftarrow \prod_{j=1}^n g_{ij}^{\alpha_{\mathrm{t}j}} \quad (i=1,\ldots,m) & & \\
 & \xrightarrow{u_{\mathrm{t}1},\ldots,u_{\mathrm{t}m}\in\mathbb{G}} & \\
 & & c \xleftarrow{\mathrm{R}} \mathcal{C} \\
 & \xleftarrow{\quad c \quad} & \\
\alpha_{\mathrm{z}j} \leftarrow \alpha_{\mathrm{t}j} + \alpha_j c \quad (j=1,\ldots,n) & & \\
 & \xrightarrow{\alpha_{\mathrm{z}1},\ldots,\alpha_{\mathrm{z}n}\in\mathbb{Z}_q} & \\
 & & \prod_{j=1}^n g_{ij}^{\alpha_{\mathrm{z}j}} \overset{?}{=} u_{\mathrm{t}i}\cdot u_i^c \quad (i=1,\ldots,m)
\end{array}$$

Figure 19.8: The generic linear protocol

So a statement is a formula $\phi \in \mathcal{F}$, and a witness for ϕ is an assignment $(\alpha_1,\ldots,\alpha_n) \in \mathbb{Z}_q^n$ to the variables $x_1,\ldots,x_n$ that makes the formula true. The reason we call this a set of "linear" relations is because if we take discrete logarithms, (19.13) can be written as the system of linear equations

$$\mathsf{Dlog}_g(u_i) = \sum_{j=1}^n x_j \cdot \mathsf{Dlog}_g(g_{ij}) \qquad \text{for } i = 1,\ldots,m.$$

A witness is a solution to this system of equations.

The **generic linear protocol** (P,V) for such a relation $\mathcal{R}$ is given in Fig. 19.8. The prover has ϕ and a witness $(\alpha_1,\ldots,\alpha_n) \in \mathbb{Z}_q^n$. As usual, the challenge space $\mathcal{C}$ is a subset of $\mathbb{Z}_q$. All the Sigma protocols presented so far are special cases of the generic linear protocol:

- Schnorr's protocol is a special case with $\phi_1(x) := \{u = g^x\}$.
- Okamoto's protocol is a special case with $\phi_2(x,y) := \{u = g^x h^y\}$.
- The Chaum-Pedersen protocol is a special case with $\phi_3(x) := \{v = g^x \ \wedge \ w = u^x\}$.

One can prove the following theorem by mimicking the proofs of the corresponding theorems for Schnorr, Okamoto, and Chaum-Pedersen. We leave it as an exercise for the reader.

Theorem 19.11. *The generic linear protocol in Fig. 19.8 is a Sigma protocol for the relation $\mathcal{R}$ defined in (19.14). Moreover, it provides special soundness and is special HVZK.*

We can generalize the generic linear protocol even further, where we allow the various equations in (19.13) to be over different groups. The only requirement is that all groups have the same prime order q. The protocol is exactly the same. A typical situation that arises in applications is where there are two types of equations: the first type are equations over a cryptographically interesting group $\mathbb{G}$ of order q, and the second type are equations over $\mathbb{Z}_q$, which are of the form $\kappa_i = \sum_{j=1}^n \lambda_{ij} x_j$, where the κ_i's and λ_{ij}'s are elements of $\mathbb{Z}_q$.

$$\begin{array}{ccc}
\underline{P\big(\alpha \in \mathbb{H}_1,\ \psi\big)} & & \underline{V(u \in \mathbb{H}_2,\ \psi)} \\
\alpha_{\mathrm{t}} \overset{\scriptscriptstyle \mathrm{R}}{\leftarrow} \mathbb{H}_1, \quad u_{\mathrm{t}} \leftarrow \psi(\alpha_{\mathrm{t}}) & & \\
& \xrightarrow{\qquad u_{\mathrm{t}} \in \mathbb{H}_2 \qquad} & \\
& & c \overset{\scriptscriptstyle \mathrm{R}}{\leftarrow} \mathcal{C} \\
& \xleftarrow{\qquad c \in \mathbb{Z} \qquad} & \\
\alpha_{\mathrm{z}} \leftarrow \alpha_{\mathrm{t}} + \alpha \cdot c \in \mathbb{H}_1 & & \\
& \xrightarrow{\qquad \alpha_{\mathrm{z}} \in \mathbb{H}_1 \qquad} & \\
& & \psi(\alpha_{\mathrm{z}}) \overset{?}{=} u_{\mathrm{t}} \cdot u^c
\end{array}$$

Figure 19.9: A Sigma protocol for the preimage of a homomorphism

19.5.4 A Sigma protocol for the pre-image of a homomorphism

All the Sigma protocols presented so far, including the general linear protocol, can be described more clearly and succinctly using the language of group homomorphisms. Let $\mathbb{H}_1$ and $\mathbb{H}_2$ be two finite abelian groups of known order and let $\psi : \mathbb{H}_1 \to \mathbb{H}_2$ be a group homomorphism. We will write the group operation in $\mathbb{H}_1$ additively and the group operation in $\mathbb{H}_2$ multiplicatively.

Let $u \in \mathbb{H}_2$. Fig. 19.9 gives a Sigma protocol that allows a prover to convince a verifier that it "knows" a preimage of u under ψ. Specifically, the protocol is a Sigma protocol for the relation

$$\mathcal{R} := \big\{ \big(\alpha, (u, \psi)\big) \in \mathbb{H}_1 \times (\mathbb{H}_2 \times \mathcal{F}) : \ \psi(\alpha) = u \big\}. \tag{19.15}$$

Here $\alpha \in \mathbb{H}_1$ is the preimage under ψ for $u \in \mathbb{H}_2$. The prover in Fig. 19.9 has the witness $\alpha \in \mathbb{H}_1$, the verifier has the image $u \in \mathbb{H}_2$, and both parties have $(\mathbb{H}_1, \mathbb{H}_2, \psi)$. The challenge space $\mathcal{C}$ is $\{0, 1, \dots, N-1\} \subseteq \mathbb{Z}$ for some integer N.

Let us see how this protocol encompases all the example Sigma protocols so far. Let $\mathbb{G}$ be a group of prime order q with generators $g, h, u \in \mathbb{G}$.

- Okamoto's protocol is a special case with $\mathbb{H}_1 := \mathbb{Z}_q^2, \quad \mathbb{H}_2 := \mathbb{G}, \quad \text{and} \quad \psi_1(x, y) := g^x h^y$.

- The Chaum-Pedersen protocol is a special case with

  $$\mathbb{H}_1 := \mathbb{Z}_q, \quad \mathbb{H}_2 := \mathbb{G}^2, \quad \text{and} \quad \psi_2(x) := (g^x, u^x).$$

  We can even set $\mathbb{H}_2 := \mathbb{G}_1 \times \mathbb{G}_2$ with $g \in \mathbb{G}_1$, $u \in \mathbb{G}_2$, and $|\mathbb{G}_1| = |\mathbb{G}_2|$. Then for a given $(v, w) \in \mathbb{G}_1 \times \mathbb{G}_2$, proving knowledge of a ψ_2 preimage of (v, w) proves equality of discrete-logs $\mathsf{Dlog}_g(v) = \mathsf{Dlog}_u(w)$ in distinct groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

- The general linear protocol in Fig. 19.8 is a special case with

  $$\mathbb{H}_1 := (\mathbb{Z}_q)^n, \quad \mathbb{H}_2 := \mathbb{G}^m, \quad \text{and} \quad \psi_3(x_1, \dots, x_n) := \left(\prod_{j=1}^{n} g_{1j}^{x_j}, \quad \dots, \quad \prod_{j=1}^{n} g_{mj}^{x_j} \right).$$

  where $g_{ij} \in \mathbb{G}$ for all $i = 1, \dots, m$ and $j = 1, \dots, n$.

One can easily verify that the maps ψ_1, ψ_2, ψ_3 are group homomorphims. By using these homomorphims in the protocol of Fig. 19.9 we obtain all the example protocols in this section as a special case.

Theorem 19.12. *The protocol in Fig. 19.9 is a Sigma protocol for the relation $\mathcal{R}$ defined in (19.15). Moreover, it is special HVZK, and provides special soundness whenever the smallest prime factor of $|\mathbb{H}_1| \times |\mathbb{H}_2|$ is at least $|\mathcal{C}|$.*

The proof exactly mimicks the proof of the corresponding theorem for the Schnorr protocol. We require the lower bound on the smallest prime factor of $|\mathbb{H}_1| \times |\mathbb{H}_2|$ to ensure that the witness extractor can obtain a ψ preimage from two accepting conversations $(u_{\text{t}}, c, \alpha_{\text{z}})$ and $(u_{\text{t}}, c', \alpha'_{\text{z}})$. As in the witness extractor for the Schnorr protocol, we obtain a relation $\psi(\Delta\alpha) = u^{\Delta c}$ where $\Delta\alpha := (\alpha_{\text{z}} - \alpha'_{\text{z}}) \in \mathbb{H}_1$ and $\Delta c := (c - c') \in \mathbb{Z}$. The lower bound on the prime factors of $|\mathbb{H}_1|$ and $|\mathbb{H}_2|$ ensures that (1) we can divide $\Delta\alpha$ by Δc in $\mathbb{H}_1$, and (2) we can take a Δc root of the right hand side by raising it to the power of $((\Delta c)^{-1} \bmod |\mathbb{H}_2|) \in \mathbb{Z}$. We then obtain the relation $\psi(\Delta\alpha/\Delta c) = u$ so that $\Delta\alpha/\Delta c \in \mathbb{H}_1$ is a preimage of $u \in \mathbb{H}_2$ under ψ, as required.

19.5.5 A Sigma protocol for RSA

Lest the reader think that Sigma protocols are only for problems related to discrete logarithms, we present one related to RSA.

Let (n, e) be an RSA public key, where e is a prime number. We will view (n, e) as a system parameter. The Guillou-Quisquater (GQ) protocol allows a prover to convince a skeptical verifier that he "knows" an eth root of $y \in \mathbb{Z}_n^*$, without revealing anything else. More precisely, it is a Sigma protocol for the relation

$$\mathcal{R} = \left\{ (x, y) \in \mathbb{Z}_n^* \times \mathbb{Z}_n^* : \; x^e = y \right\}. \tag{19.16}$$

A witness for a statement $y \in \mathbb{Z}_n^*$ is $x \in \mathbb{Z}_n^*$ such that $x^e = y$. Since (n, e) is an RSA public key, the map that sends $x \in \mathbb{Z}_n^*$ to $y = x^e \in \mathbb{Z}_n^*$ is bijective. Therefore, every statement has a unique witness.

The GQ protocol (P, V) is given in Fig. 19.10. The challenge space $\mathcal{C}$ is a subset of $\{0, \ldots, e-1\}$. Notice that when e is small, the challenge space is small. If needed, it can be enlarged using the method of Exercise 19.6. However, when using this protocol we will typically ensure that the challenge space is large by taking e to be a large prime.

The GQ protocol in Fig. 19.10 is a special case of the protocol in Fig. 19.9 for proving knowledge of the preimage of a homomorphism. Here the homomorphism is $\psi : \mathbb{Z}_n^* \to \mathbb{Z}_n^*$ defined by $\psi(x) = x^e$. However, special soundness does not follow from Theorem 19.12 because the group $\mathbb{Z}_n^*$ has unknown order. Instead, we have to give a seperate proof of these properties. We do so in the following theorem.

Theorem 19.13. *The GQ protocol is a Sigma protocol for the relation $\mathcal{R}$ defined in (19.16). Moreover, it provides special soundness and is special HVZK.*

Proof. An accepting conversation for y is of the form $(x_{\text{t}}, c, x_{\text{z}})$, where $x_{\text{z}}^e = y_{\text{t}} \cdot y^c$. The reader may easily verify the basic correctness requirement: an interaction between an honest prover and an honest verifier always produces an accepting conversation.

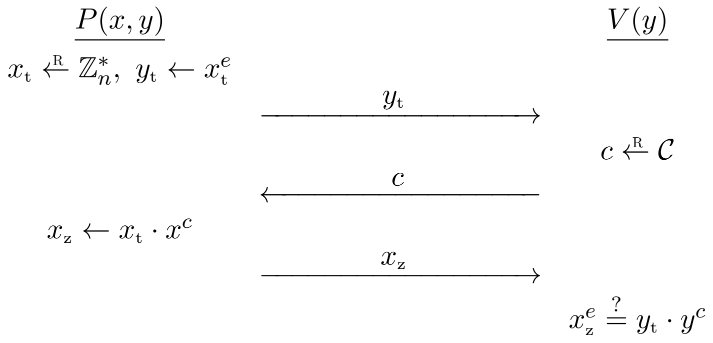

$P(x,y)$
$V(y)$
$x_{\rm t} \xleftarrow{\rm R} \mathbb{Z}_n^*,\ y_{\rm t} \leftarrow x_{\rm t}^e$
$y_{\rm t}$
$c \xleftarrow{\rm R} \mathcal{C}$
c
$x_{\rm z} \leftarrow x_{\rm t} \cdot x^c$
$x_{\rm z}$
$x_{\rm z}^e \overset{?}{=} y_{\rm t} \cdot y^c$

Figure 19.10: The GQ protocol

Special soundness. Next, we show that the GQ protocol provides special soundness. Suppose we have two accepting conversations $(x_{\rm t}, c, x_{\rm z})$ and $(x_{\rm t}, c', x'_{\rm z})$ for the statement y, where $c \neq c'$. We have to show how to efficiently compute an eth root of y. Observe that

$$x_{\rm z}^e = y_{\rm t} \cdot y^c \quad \text{and} \quad (x'_{\rm z})^e = y_{\rm t} \cdot y^{c'}.$$

Dividing the first equation by the second, we obtain

$$(\Delta x)^e = y^{\Delta c}, \quad \text{where } \Delta x := x_{\rm z}/x'_{\rm z},\ \Delta c := c - c'.$$

Observe that because $c \neq c'$ and both c and c' belong to the interval $\{0, \ldots, e-1\}$, we have $0 < |\Delta c| < e$, and so $e \nmid \Delta c$; moreover, since e is prime, it follows that $\gcd(e, \Delta c) = 1$. Thus, we may apply Theorem 10.6 (with the given e, $f := \Delta c$, and $w := \Delta x$), to obtain an eth root of y.

The reader should observe that the technique presented here for computing an RSA inverse from two accepting conversations is essentially the same idea that was used in the proof of Theorem 10.7. Indeed, the two accepting conversations yield a collision $((x_{\rm z}, -c \bmod e), (x'_{\rm z}, -c' \bmod e))$ on the hash function $H_{\rm rsa}(a, b) := a^e y^b$.

Special HVZK. Finally, we show that the GQ protocol is special HVZK by exhibiting a simulator. On input $y \in \mathbb{Z}_n^*$ and $c \in \mathcal{C}$, the simulator computes

$$x_{\rm z} \xleftarrow{\rm R} \mathbb{Z}_n^*,\ \ y_{\rm t} \leftarrow x_{\rm z}^e / y^c$$

and outputs $(y_{\rm t}, x_{\rm z})$. The key observation is that in a real conversation, c and $x_{\rm z}$ are independent, with c uniformly distributed over $\mathcal{C}$ and $x_{\rm z}$ uniformly distributed over $\mathbb{Z}_n^*$; moreover, given c and $x_{\rm z}$, the value $y_{\rm t}$ is uniquely determined by the equation $x_{\rm z}^e = y_{\rm t} \cdot y^c$. It should be clear that this is the same as the output distribution of the simulator. □

19.6 Identification and signatures from Sigma protocols

By mimicking the Schnorr constructions, we can easily convert any Sigma protocol into a corresponding identification scheme and signature scheme.

Suppose we have a Sigma protocol (P, V) for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. In addition to P and V, we need a **key generation algorithm for** $\mathcal{R}$. This is a probabilistic algorithm G that generates a public-key/secret-key pair (pk, sk), where $pk = y$ and $sk = (x, y)$ for some $(x, y) \in \mathcal{R}$.

To get secure identification and signature schemes we need the following "one-wayness" property: given a public key $pk = y \in \mathcal{Y}$ output by G, it should be hard to compute $\hat{x} \in \mathcal{X}$ such that $(\hat{x}, y) \in \mathcal{R}$. This notion is made precise by the following attack game.

Attack Game 19.2 (One-way key generation). Let G be a key generation algorithm for $R \subseteq \mathcal{X} \times \mathcal{Y}$. For a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger runs $(pk, sk) \xleftarrow{R} G()$, and sends $pk = y$ to $\mathcal{A}$;
- $\mathcal{A}$ outputs $\hat{x} \in \mathcal{X}$.

We say that the adversary wins the game if $(\hat{x}, y) \in \mathcal{R}$. We define $\mathcal{A}$'s advantage with respect to G, denoted $\text{OW}\mathsf{adv}[\mathcal{A}, G]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 19.6. *We say that a key generation algorithm G is* ***one way*** *if for all efficient adversaries $\mathcal{A}$, the quantity $\text{OW}\mathsf{adv}[\mathcal{A}, G]$ is negligible.*

Example 19.4. For the Schnorr Sigma protocol (Example 19.1), the most natural key generation algorithm computes $\alpha \xleftarrow{R} \mathbb{Z}_q$ and $u \leftarrow g^\alpha \in \mathbb{G}$, and outputs $pk := u$ and $sk := (\alpha, u)$. It is clear that this key generation algorithm is one-way under the DL assumption. □

Example 19.5. Consider the GQ protocol in Section 19.5.5. Recall that the RSA public key (n, e) is viewed here as a system parameter. The most natural key generation algorithm computes $x \xleftarrow{R} \mathbb{Z}_n^*$ and $y \leftarrow x^e \in \mathbb{Z}_n^*$. It outputs $pk := y$ and $sk := (x, y)$. It is clear that this key generation algorithm is one-way under the RSA assumption (see Theorem 10.5). □

A Sigma protocol (P, V) with a key generation algorithm G gives an identification scheme (G, P, V). The next two theorems prove that it is secure against eavesdropping attacks.

Theorem 19.14. *Let (P, V) be a Sigma protocol for an effective relation $\mathcal{R}$ with a large challenge space. Let G be a key generation algorithm for $\mathcal{R}$. If (P, V) provides special soundness and G is one-way, then the identification scheme $\mathcal{I} := (G, P, V)$ is secure against direct attacks.*

> *In particular, suppose $\mathcal{A}$ is an efficient impersonation adversary attacking $\mathcal{I}$ via a direct attack as in Attack Game 18.1, with advantage $\epsilon := \text{ID1}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$. Then there exists an efficient adversary $\mathcal{B}$ attacking G as in Attack Game 19.2 (whose running time is about* twice *that of $\mathcal{A}$), with advantage $\epsilon' := \text{OW}\mathsf{adv}[\mathcal{B}, G]$, such that*
>
> $$\epsilon' \geq \epsilon^2 - \epsilon/N, \tag{19.17}$$
>
> *where N is the size of the challenge space, which implies*
>
> $$\epsilon \leq \frac{1}{N} + \sqrt{\epsilon'}. \tag{19.18}$$

Proof. We can just mimic the proof of Theorem 19.1. Using the impersonation adversary $\mathcal{A}$, we build an adversary $\mathcal{B}$ that breaks the one-wayness of G, as follows. Adversary $\mathcal{B}$ is given a public key $pk = y$ from its challenger, and our goal is to make $\mathcal{B}$ compute $\hat{x}$ such that $(\hat{x}, y) \in \mathcal{R}$, with help from $\mathcal{A}$. The computation of $\mathcal{B}$ consists of two stages.

In the first stage of its computation, $\mathcal{B}$ plays the role of challenger to $\mathcal{A}$, giving $\mathcal{A}$ the value $pk = y$ as the verification key. Using the same rewinding argument as in the proof of Theorem 19.1, with probability at least $\epsilon^2 - \epsilon/N$, adversary $\mathcal{B}$ obtains two accepting conversations (t, c, z) and (t, c', z') for y with $c \neq c'$. In more detail, $\mathcal{B}$ awaits $\mathcal{A}$'s commitment t, gives $\mathcal{A}$ a random challenge c, and awaits $\mathcal{A}$'s response z. After this happens, $\mathcal{B}$ rewinds $\mathcal{A}$'s internal state back to the point just after which it generated t, gives $\mathcal{A}$ another random challenge c', and awaits $\mathcal{A}$'s response z'. By the Rewinding Lemma (Lemma 19.2), this procedure will yield the two required accepting conversations with probability at least $\epsilon^2 - \epsilon/N$.

In the second stage of the computation, $\mathcal{B}$ feeds these two conversations into a witness extractor (which is guaranteed by the special soundness property) to extract a witness $\hat{x}$ for y.

That proves (19.17), and (19.18) follows by the same calculation as in Theorem 19.1. □

Theorem 19.3 obviously applies to identification protocols derived from special HVZK Sigma protocols:

Theorem 19.15. *Let (P, V) be a Sigma protocol for an effective relation $\mathcal{R}$. Let G be a key generation algorithm for $\mathcal{R}$. If the identification protocol $\mathcal{I} = (G, P, V)$ is secure against direct attacks, and (P, V) is special HVZK, then $\mathcal{I}$ is also secure against eavesdropping attacks.*

> *In particular, for every impersonation adversary $\mathcal{A}$ that attacks $\mathcal{I}$ via an eavesdropping attack, as in Attack Game 18.2, there is an adversary $\mathcal{B}$ that attacks $\mathcal{I}$ via a direct attack on, as in Attack Game 18.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{ID2adv}[\mathcal{A}, \mathcal{I}] = \mathrm{ID1adv}[\mathcal{B}, \mathcal{I}].$$

Example 19.6. If we augment the GQ protocol (P, V) with the key generation algorithm G in Example 19.5, then we get an identification scheme $\mathcal{I}_{\mathrm{GQ}} = (G, P, V)$ that is secure against eavesdropping attacks under the RSA assumption (provided the challenge space is large). □

19.6.1 The Fiat-Shamir heuristic for signatures

We can convert Sigma protocols to signature schemes, using the same technique developed in Section 19.2. The general technique is originally due to Fiat and Shamir. The building blocks are as follows:

- a Sigma protocol (P, V) for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$; we assume that conversations are of the form (t, c, z), where $t \in \mathcal{T}$, $c \in \mathcal{C}$, and $z \in \mathcal{Z}$;
- a key generation algorithm G for $\mathcal{R}$;
- a hash function $H : \mathcal{M} \times \mathcal{T} \to \mathcal{C}$, which will be modeled as a random oracle; the set $\mathcal{M}$ will be the message space of the signature scheme.

The **Fiat-Shamir signature scheme** derived from G and (P, V) works as follows:

- The key generation algorithm is G, so a public key is of the form $pk = y$, where $y \in \mathcal{Y}$, and a secret key is of the form $sk = (x, y) \in \mathcal{R}$.
- To sign a message $m \in \mathcal{M}$ using a secret key $sk = (x, y)$, the signing algorithm runs as follows:
  - it starts the prover $P(x, y)$, obtaining a commitment $t \in \mathcal{T}$;

  - it computes a challenge $c \leftarrow H(m,t)$;
  - finally, it feeds c to the prover, obtaining a response z, and outputs the signature $\sigma := (t,z) \in \mathcal{T} \times \mathcal{Z}$.

- To verify a signature $\sigma = (t,z) \in \mathcal{T} \times \mathcal{Z}$ on a message $m \in \mathcal{M}$ using a public key $pk = y$, the verification algorithm computes $c \leftarrow H(m,t)$, and checks that (t,c,z) is an accepting conversation for y.

Just as we did for Schnorr, we will show that the Fiat-Shamir signature scheme is secure in the random oracle model if the corresponding identification scheme (G,P,V) is secure against eavesdropping attacks. However, we will need one more technical assumption, which essentially all Sigma protocols of interest satisfy.

Definition 19.7 (Unpredictable commitments). *Let (P,V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, and suppose that all conversations (t,c,z) lie in $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$. We say that (P,V) has δ-**unpredictable commitments** if for every $(x,y) \in \mathcal{R}$ and $\hat{t} \in \mathcal{T}$, with probability at most δ, an interaction between $P(x,y)$ and $V(y)$ produces a conversation (t,c,z) with $t = \hat{t}$. We say that (P,V) has **unpredictable commitments** if it is has δ-unpredictable commitments for negligible δ.*

Theorem 19.16. *If H is modeled as a random oracle, the identification scheme $\mathcal{I} = (G,P,V)$ is secure against eavesdropping attacks, and (P,V) has unpredictable commitments, then the Fiat-Shamir signature scheme $\mathcal{S}$ derived from G and (P,V) is secure.*

> *In particular, let $\mathcal{A}$ be an adversary attacking $\mathcal{S}$ as in the random oracle version of Attack Game 13.1. Moreover, assume that $\mathcal{A}$ issues at most Q_{s} signing queries and Q_{ro} random oracle queries, and that (P,V) has δ-unpredictable commitments. Then there exist a $(Q_{\mathrm{ro}}+1)$-impersonation adversary $\mathcal{B}$ that attacks $\mathcal{I}$ via an eavesdropping attack as in Attack Game 19.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathrm{SIG^{ro}}\mathsf{adv}[\mathcal{A},\mathcal{S}] \le Q_{\mathrm{s}}(Q_{\mathrm{s}}+Q_{\mathrm{ro}}+1)\delta + \mathrm{rID2}\mathsf{adv}[\mathcal{B},\mathcal{I},Q_{\mathrm{ro}}+1].$$

The proof of this theorem is almost identical to that of Theorem 19.7. We leave the details to the reader.

Putting everything together, suppose that we start with a Sigma protocol (P,V) that is special HVZK and provides special soundness. Further, suppose (P,V) has unpredictable commitments and a large challenge space. Then, if we combine (P,V) with a one-way key generation algorithm G, the Fiat-Shamir signature construction gives us a secure signature scheme (that is, if we model H as a random oracle). The Schnorr signature scheme is a special case of this construction.

Just as we did for Schnorr signatures, we could use Lemma 19.6 to reduce from r-impersonation to 1-impersonation; however, a tighter reduction is possible. Indeed, the proof of Lemma 19.8 goes through, essentially unchanged:

Lemma 19.17. *Let (P,V) be a special HVZK Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ that provides special soundness, let G be a key generation algorithm for $\mathcal{R}$, and consider the resulting identification protocol $\mathcal{I} = (G,P,V)$. Suppose $\mathcal{A}$ is an efficient r-impersonation eavesdropping adversary attacking $\mathcal{I}$, as in Attack Game 19.1, with advantage $\epsilon :=$ $\mathrm{rID2}\mathsf{adv}[\mathcal{A},\mathcal{I},r]$. Then there exists an efficient adversary $\mathcal{B}$ attacking G as in Attack Game 19.2 (whose running time is about* twice *that of $\mathcal{A}$), with advantage $\epsilon' :=$ $\mathrm{OW}\mathsf{adv}[\mathcal{B},G]$, such that*

$$\epsilon' \ge \epsilon^2/r - \epsilon/N, \tag{19.19}$$

where N is the size of the challenge space, which implies

$$\epsilon \leq \frac{r}{N} + \sqrt{r\epsilon'}. \tag{19.20}$$

Using this, we get the following concrete security bound for Theorem 19.16, assuming (P, V) is special HVZK and provides special soundness:

> *Let $\mathcal{A}$ be an efficient adversary attacking $\mathcal{S}$ as in the random oracle version of Attack Game 13.1. Moreover, assume that $\mathcal{A}$ issues at most $Q_{\rm s}$ signing queries and $Q_{\rm ro}$ random oracle queries. Then there exists an efficient adversary $\mathcal{B}$ attacking G as in Attack Game 19.2 (whose running time is about* twice *that of $\mathcal{A}$), such that*
>
> $$\mathrm{SIG^{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}] \leq Q_{\rm s}(Q_{\rm s} + Q_{\rm ro} + 1)\delta + (Q_{\rm ro} + 1)/N + \sqrt{(Q_{\rm ro} + 1)\mathrm{OW}\mathsf{adv}[\mathcal{B}, G]}), \tag{19.21}$$
>
> *where N is the size of the challenge space.*

19.6.1.1 The GQ signature scheme

The Fiat-Shamir signature construction above applied to the GQ Sigma protocol (Section 19.5.5) gives us a new signature scheme based on RSA. The scheme makes use of an RSA public key (n, e) as a system parameter, where the encryption exponent e is a large prime. If desired, this system parameter can be shared by many users. We need a hash function $H : \mathcal{M} \times \mathcal{T} \to \mathcal{C}$, where $\mathcal{T}$ is a set into which all elements of $\mathbb{Z}_n^*$ can be encoded, $\mathcal{M}$ is the message space of the signature scheme, and $\mathcal{C}$ is a subset of $\{0, \ldots, e-1\}$. The GQ signature scheme is $\mathcal{S}_{\rm GQ} = (G, S, V)$, where:

- The key generation algorithm G runs as follows:

  $$x \xleftarrow{\text{R}} \mathbb{Z}_n^*, \quad y \leftarrow x^e.$$

  The public key is $pk := y$, and the secret key is $sk := x$.

- To sign a message $m \in \mathcal{M}$ using a secret key $sk = x$, the signing algorithm runs as follows:

  $$\begin{aligned} S(\ sk,\ m\ ) := \quad & x_{\rm t} \xleftarrow{\text{R}} \mathbb{Z}_n^*, \quad y_{\rm t} \leftarrow x_{\rm t}^e, \quad c \leftarrow H(m, y_{\rm t}), \quad x_{\rm z} \leftarrow x_{\rm t} \cdot x^c \\ & \text{output } \sigma := (y_{\rm t}, x_{\rm z}). \end{aligned}$$

- To verify a signature $\sigma = (y_{\rm t}, x_{\rm z})$ on a message $m \in \mathcal{M}$, using the public key $pk = y$, the signature verification algorithm V computes $c := H(m, y_{\rm t})$. It outputs accept if $x_{\rm z}^e = y_{\rm t} \cdot y^c$, and outputs reject, otherwise.

As we saw in Example 19.6, the GQ identification scheme is secure against eavesdropping attacks under the RSA assumption (provided the challenge space is large). Also, we observe that the GQ Sigma protocol has $1/\phi(n)$-unpredictable commitments. It follows from Theorem 19.16 that the corresponding signature scheme is secure in the random oracle model, under the RSA assumption.

The advantage of GQ signatures over RSA signatures, such as $\mathcal{S}_{\text{RSA-FDH}}$, is that the signing algorithm is much faster. Signing with $\mathcal{S}_{\text{RSA-FDH}}$ requires a large exponantiation. Signing with GQ requires two exponentiations with exponents e and c, but both can be only 128 bits. Fast signing is important when the signer is a weak device, as in the case of a chip enabled creditcard that signs every creditcard transaction.

An optimization. The GQ signature scheme can be optimized in the same way as the Schnorr signature scheme. Instead of defining a signature on m to be a pair $(y_{\rm t}, x_{\rm z})$ satisfying

$$x_{\rm z}^e = y_{\rm t} \cdot y^c,$$

where $c := H(m, y_{\rm t})$, we can define it to be a pair $(c, x_{\rm z})$ satisfying

$$c = H(m, y_{\rm t}),$$

where $y_{\rm t} := x_{\rm z}^e / y^c$. As a further optimization, one can store y^{-1} in the public key instead of y, which will speed up verification.

It turns out that this same optimization can be applied to most instances of the Fiat-Shamir signature construction. See Exercise 19.19.

19.7 Combining Sigma protocols: AND and OR proofs

In this section we show how Sigma protocols can be combined to make new Sigma protocols. In the AND-proof construction, a prover can convince a verifier that he "knows" witnesses for a pair of statements. In the OR-proof construction, a prover can convince a verifier that he "knows" witnesses for one of two statements.

19.7.1 The AND-proof construction

Suppose we have a Sigma protocol (P_0, V_0) for $\mathcal{R}_0 \subseteq \mathcal{X}_0 \times \mathcal{Y}_0$, and a Sigma protocol (P_1, V_1) for $\mathcal{R}_1 \subseteq \mathcal{X}_1 \times \mathcal{Y}_1$. Further, let us assume that both Sigma protocols use the same challenge space $\mathcal{C}$. We can combine them to form a Sigma protocol for the relation

$$\mathcal{R}_{\rm AND} = \Big\{ \big((x_0, x_1),\ (y_0, y_1) \big) \in (\mathcal{X}_0 \times \mathcal{X}_1) \times (\mathcal{Y}_0 \times \mathcal{Y}_1) :\ (x_0, y_0) \in \mathcal{R}_0 \text{ and } (x_1, y_1) \in \mathcal{R}_1 \Big\}. \tag{19.22}$$

In other words, for a given pair of statements $y_0 \in \mathcal{Y}_0$ and $y_1 \in \mathcal{Y}_1$, this **AND protocol** allows a prover to convince a skeptical verifier that he "knows" a witness for y_0 *and* a witness for y_1. The protocol (P, V) runs as follows, where the prover P is initialized with $((x_0, x_1), (y_0, y_1)) \in \mathcal{R}_{\rm AND}$, the verifier V is initialized with $(y_0, y_1) \in \mathcal{Y}_0 \times \mathcal{Y}_1$:

1. P runs $P_0(x_0, y_0)$ to get a commitment t_0 and runs $P_1(x_1, y_1)$ to get a commitment t_1, and sends the commitment pair (t_0, t_1) to V;

2. V computes $c \xleftarrow{\rm R} \mathcal{C}$, and sends the challenge c to P;

3. P feeds the challenge c to both $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$, obtaining responses z_0 and z_1, and sends the response pair (z_0, z_1) to V;

4. V checks that (t_0, c, z_0) is an accepting conversation for y_0 and that (t_1, c, z_1) is an accepting conversation for y_1.

Theorem 19.18. *The AND protocol (P, V) is a Sigma protocol for the relation $\mathcal{R}_{\rm AND}$ defined in (19.22). If (P_0, V_0) and (P_1, V_1) provide special soundness, then so does (P, V). If (P_0, V_0) and (P_1, V_1) are special HVZK, then so is (P, V).*

Proof sketch. Correctness is clear.

For special soundness, if (P_0, V_0) has extractor Ext_0 and (P_1, V_1) has extractor Ext_1, then the extractor for (P, V) is

$$Ext\Big((y_0, y_1), \big((t_0, t_1), c, (z_0, z_1)\big),\ \big((t_0, t_1), c', (z_0', z_1')\big) \Big) :=$$
$$\Big(Ext_0\big(y_0, (t_0, c, z_0), (t_0, c', z_0')\big),\ Ext_1\big(y_1, (t_1, c, z_1), (t_1, c', z_1')\big)\Big).$$

For special HVZK, if (P_0, V_0) has simulator Sim_0 and (P_1, V_1) has simulator Sim_1, then the simulator for (P, V) is

$$Sim((y_0, y_1), c) := ((t_0, t_1), (z_0, z_1)),$$

where

$$(t_0, z_0) \xleftarrow{R} Sim_0(y_0, c) \quad \text{and} \quad (t_1, z_1) \xleftarrow{R} Sim_1(y_1, c).$$

We leave it to the reader to fill in the details. However, we point out that in our construction of *Sim*, we have exploited the fact that in our definition of special HVZK, the challenge is an input to the simulator, which we can feed to both Sim_0 and Sim_1. This is one of the main reasons for this aspect of the definition. □

19.7.2 The OR-proof construction

Suppose we have a Sigma protocol (P_0, V_0) for $\mathcal{R}_0 \subseteq \mathcal{X}_0 \times \mathcal{Y}_0$, and a Sigma protocol (P_1, V_1) for $\mathcal{R}_1 \subseteq \mathcal{X}_1 \times \mathcal{Y}_1$. We need to make some additional assumptions:

- Both Sigma protocols use the same challenge space $\mathcal{C}$, which is of the form $\mathcal{C} = \{0, 1\}^n$. (Note that in the examples we have seen where challenges are numbers, we can always encode bit strings as numbers in binary notation.)

- Both protocols are special HVZK, with simulators Sim_0 and Sim_1, respectively.

We can combine them to form a Sigma protocol for the relation

$$\mathcal{R}_{\text{OR}} = \Big\{ \big((b, x),\ (y_0, y_1)\big) \in \big(\{0, 1\} \times (\mathcal{X}_0 \cup \mathcal{X}_1)\big) \times (\mathcal{Y}_0 \times \mathcal{Y}_1) :\ (x, y_b) \in \mathcal{R}_b \Big\}. \tag{19.23}$$

In other words, for a given pair of statements $y_0 \in \mathcal{Y}_0$ and $y_1 \in \mathcal{Y}_1$, this **OR protocol** allows a prover to convince a skeptical verifier that he "knows" a witness for y_0 *or* a witness for y_1. Nothing else should be revealed. In particular the protocol should not reveal if the prover has a witness for y_0 or for y_1.

The protocol (P, V) runs as follows, where the prover P is initialized with $((b, x), (y_0, y_1)) \in \mathcal{R}_{\text{OR}}$, the verifier V is initialized with $(y_0, y_1) \in \mathcal{Y}_0 \times \mathcal{Y}_1$, and $d := 1 - b$:

1. P computes $c_d \xleftarrow{R} \mathcal{C}$, $(t_d, z_d) \xleftarrow{R} Sim_d(y_d, c_d)$.

   P also runs $P_b(x, y_b)$ to get a commitment t_b, and sends the commitment pair (t_0, t_1) to V;

2. V computes $c \xleftarrow{R} \mathcal{C}$, and sends the challenge c to P;

3. P computes $c_b \leftarrow c \oplus c_d$

   P feeds the challenge c_b to $P_b(x, y_b)$, obtaining a response z_b, and sends (c_0, z_0, z_1) to V;

4. V computes $c_1 \leftarrow c \oplus c_0$, and checks that (t_0, c_0, z_0) is an accepting conversation for y_0, and that (t_1, c_1, z_1) is an accepting conversation for y_1.

Theorem 19.19. *The OR protocol (P, V) is a special HVZK Sigma protocol for the relation $\mathcal{R}_{\text{OR}}$ defined in (19.23). If (P_0, V_0) and (P_1, V_1) provide special soundness, then so does (P, V).*

Proof sketch. Correctness is clear.

For special soundness, if (P_0, V_0) has extractor Ext_0 and (P_1, V_1) has extractor Ext_1, then the extractor Ext for (P, V) takes as input (y_0, y_1) and a pair of accepting conversations

$$\big((t_0, t_1),\ c,\ (c_0, z_0, z_1)\big) \quad \text{and} \quad \big((t_0, t_1),\ c',\ (c_0', z_0', z_1')\big).$$

Let $c_1 := c \oplus c_0$ and $c_1' := c' \oplus c_0'$. The key observation is that if $c \neq c'$, then we must have either $c_0 \neq c_0'$ or $c_1 \neq c_1'$. So Ext works as follows:

```
if $c_0 \neq c_0'$
    then output $\big(\ 0,\ \mathit{Ext}_0(y_0, (t_0, c_0, z_0), (t_0, c_0', z_0'))\ \big)$
    else  output $\big(\ 1,\ \mathit{Ext}_1(y_1, (t_1, c_1, z_1), (t_1, c_1', z_1'))\ \big)$
```

For special HVZK, the simulator for (P, V) is

$$\mathit{Sim}((y_0, y_1), c) := ((t_0, t_1), (c_0, z_0, z_1)),$$

where

$$c_0 \xleftarrow{\text{R}} \mathcal{C},\ c_1 \leftarrow c \oplus c_0,\ (t_0, z_0) \xleftarrow{\text{R}} \mathit{Sim}_0(y_0, c_0),\ (t_1, z_1) \xleftarrow{\text{R}} \mathit{Sim}_1(y_1, c_1).$$

We leave it to the reader to fill in the details. However, we point out that to guarantee correctness, we have exploited the fact that in our definition of special HVZK, the simulator always outputs an accepting conversation. This is one of the main reasons for this aspect of the definition. □

19.8 Witness independence and applications

We next study a useful property of Sigma protocols called **witness independence**.

For a given statement there may be several witnesses. Roughly speaking, witness independence means the following: if a "cheating" verifier V^* (one that need not follow the protocol) interacts with an honest prover P, then V^* cannot tell *which* witness P is using. In particular, even if V^* is very powerful and/or very clever and is able to compute a witness after interacting with P, this witness will be unrelated to P's witness. Of course, this property is only interesting if a given statement has more than one witness.

First, we define this property more precisely. Second, we show that special HVZK implies witness independence. This is perhaps a bit surprising, as HVZK is a property about *honest* verifiers, while witness independence applies to *all* verifiers (even computationally unbounded cheating verifiers). Finally, as an application, we show how to use witness independence to design identification protocols that are secure against *active* attacks, rather than just eavesdropping attacks. These identification protocols are simple and efficient, and their security can be based on either the DL or RSA assumptions (and without relying on the random oracle heuristic).

19.8.1 Definition of witness independence

We define witness independence using an attack game.

Attack Game 19.3 (Witness independence). Let $\Pi = (P, V)$ be a Sigma protocol for $R \subseteq \mathcal{X} \times \mathcal{Y}$. For a given adversary $\mathcal{A}$, we define an experiment (x, y) for each $(x, y) \in \mathcal{R}$. Experiment (x, y) runs as follows.

- Initially, the adversary is given the value y.
- The adversary then interacts with several instances of the prover $P(x, y)$ — in each of these interactions, the challenger carries out the provers' computations, while the adversary plays the role of a cheating verifier (i.e., one that need not follow V's protocol). These interactions may be concurrent (in particular, the adversary may issue challenges that depend on commitments and responses output so far by all prover instances).
- At the end of the game, the adversary outputs some value s, which belongs to a finite **output space** $\mathcal{S}$ (which may depend on $\mathcal{A}$).

For each $(x, y) \in \mathcal{R}$ and $s \in \mathcal{S}$, we define $\theta_{\mathcal{A},\Pi}(x, y, s)$ to be the probability that $\mathcal{A}$ outputs s in Experiment (x, y). □

Definition 19.8. *Let $\Pi = (P, V)$ be a Sigma protocol for $R \subseteq \mathcal{X} \times \mathcal{Y}$. We say that (P, V) is* ***witness independent*** *if for every adversary $\mathcal{A}$, for every $y \in \mathcal{Y}$, for every $x, x' \in \mathcal{X}$ such that $(x, y) \in \mathcal{R}$ and $(x', y) \in \mathcal{R}$, and for every s in the output space of $\mathcal{A}$, we have*

$$\theta_{\mathcal{A},\Pi}(x, y, s) = \theta_{\mathcal{A},\Pi}(x', y, s).$$

The definition states that for every $y \in \mathcal{Y}$ and $s \in \mathcal{S}$, the quantity $\theta_{\mathcal{A},\Pi}(x, y, s)$ is the same for all $x \in \mathcal{X}$ for which $(x, y) \in \mathcal{R}$. Note that in this definition, $\mathcal{A}$ need not be efficient. We also note that in this definition, if the Sigma protocol makes use of a system parameter, which itself may be randomly generated, we insist that the defining property should hold for every possible choice of system parameter.

This definition captures in a very strong sense the idea that the adversary's behavior depends only on the statement, but not on the particular witness that the prover is using.

In the analysis of identification schemes, it is sometimes convenient to apply the definition of witness independence as follows. Suppose (P, V) is a Sigma protocol for $R \subseteq \mathcal{X} \times \mathcal{Y}$, and that G is a key generation algorithm for $\mathcal{R}$. Suppose we run the key generation algorithm to obtain $pk = y$ and $sk = (x, y)$, and then run Experiment (x, y) in Attack Game 19.3 with an adversary $\mathcal{A}$. Let us define random variables X, Y, S, as follows:

- X represents the witness x generated by G;
- Y represents the statement y generated by G;
- S represents the adversary's output $s \in \mathcal{S}$.

Fact 19.20. *If (P, V) is witness independent, then we have*

$$\Pr[\mathsf{X} = x \wedge \mathsf{S} = s \mid \mathsf{Y} = y] = \Pr[\mathsf{X} = x \mid \mathsf{Y} = y] \cdot \Pr[\mathsf{S} = s \mid \mathsf{Y} = y] \qquad (19.24)$$

for all $(x, y) \in \mathcal{R}$ and $s \in \mathcal{S}$.

We leave the proof of Fact 19.20 as a straightforward exercise for the reader. Equation (19.24) says that conditioned on $\mathsf{Y} = y$ for any particular y, the random variables X and S are independent. One can rewrite (19.24) in a number of different ways. For example, it is equivalent to saying

$$\Pr[\mathsf{X} = x \mid \mathsf{S} = s \wedge \mathsf{Y} = y] = \Pr[\mathsf{X} = x \mid \mathsf{Y} = y]. \tag{19.25}$$

Example 19.7. Theorem 19.21 below will show that the OR-protocol (Section 19.7.2) and Okamoto's protocol (Section 19.6) are both witness independent protocols. □

19.8.2 Special HVZK implies witness independence

As promised, we now prove that special HVZK implies witness independence.

Theorem 19.21 (Special HVZK $\Longrightarrow$ WI). *If a Sigma protocol is special HVZK, then it is witness independent.*

Proof. Let (P, V) be a Sigma protocol for $R \subseteq \mathcal{X} \times \mathcal{Y}$. Suppose that all conversations (t, c, z) lie in $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$.

Let Coins be a random variable representing the possible random choices *coins* made by P. For example, in Schnorr's protocol, *coins* is the value $\alpha_{\mathrm{t}} \in \mathbb{Z}_q$, and Coins is uniformly distributed over $\mathbb{Z}_q$. The prover P's logic can be completely characterized by some function γ that maps $(x, y, c, \mathit{coins})$ to (t, z), where $(x, y) \in \mathcal{R}$ and $(t, c, z) \in \mathcal{T} \times \mathcal{C} \times \mathcal{Z}$.

Consider the probability that a real conversation between $P(x, y)$ and $V(y)$ produces a particular conversation (t, c, z). This is precisely

$$\Pr[\gamma(x, y, c, \mathsf{Coins}) = (t, z)] \;/\; |\mathcal{C}|. \tag{19.26}$$

Now consider a simulator *Sim* that is guaranteed by the special HVZK property. For all $(x, y) \in \mathcal{R}$, $c \in \mathcal{C}$, and $(t, z) \in \mathcal{T} \times \mathcal{Z}$, we define $p(y, t, c, z)$ to be the probability that $\mathit{Sim}(y, c)$ outputs (t, z). The probability that the conversation produced by running the simulator on a random challenge is equal to a particular conversation (t, c, z) is precisely

$$p(y, t, c, z) \;/\; |\mathcal{C}|. \tag{19.27}$$

As the probabilities (19.26) and (19.27) must be equal, we conclude that for all $(x, y) \in \mathcal{R}$ and $(t, c, z) \in \mathcal{T} \times \mathcal{C} \times \mathcal{Z}$, we have

$$\Pr[\gamma(x, y, c, \mathsf{Coins}) = (t, z)] = p(y, t, c, z),$$

which does not depend on x. This fact is really the crux of the proof, even if the details get a bit involved.

Now consider Experiment (x, y) of Attack Game 19.3, and assume that the adversary $\mathcal{A}$ interacts with Q copies of the prover P. The logic of the entire collection of provers can be characterized by a function γ^* that maps $(x, y, c^*, \mathit{coins}^*)$ to (t^*, z^*), where now t^*, c^*, z^*, and coins^* are corresponding vectors of length Q. Moreover, if Coins^* is a vector of Q independent copies of the random variable Coins, then for all $(x, y) \in \mathcal{R}$ and $(t^*, c^*, z^*) \in \mathcal{T}^Q \times \mathcal{C}^Q \times \mathcal{Z}^Q$, we have

$$\Pr[\gamma^*(x, y, c^*, \mathsf{Coins}^*) = (t^*, z^*)] = \prod_i p(y, t^*[i], c^*[i], z^*[i]),$$

which again, does not depend on x.

Let Coins' be a random variable representing the possible random choices *coins'* made by the adversary. The adversary's logic can be characterized by a function γ' that maps $(y, t^*, z^*, \mathit{coins}')$ to (c^*, s). Here, $(t^*, c^*, z^*) \in \mathcal{T}^Q \times \mathcal{C}^Q \times \mathcal{Z}^Q$, $s \in \mathcal{S}$ is the adversary's output, and *coins'* denotes the particular random choices made by the adversary.

Let $\mathsf{S}_{x,y}$ be a random variable that represents the output of $\mathcal{A}$ in Experiment (x, y) of the attack game. Let $\mathsf{T}_{x,y}$ be the random variable representing the possible transcripts $t = (t^*, c^*, z^*)$. For $s \in \mathcal{S}$ and $t = (t^*, c^*, z^*)$, define events $\mathbf{\Gamma}^*(x, y; t)$ and $\mathbf{\Gamma}'(y, s; t)$ as follows:

$$\mathbf{\Gamma}^*(x, y; t) : \gamma^*(x, y, c^*, \mathsf{Coins}^*) = (t^*, z^*), \qquad \mathbf{\Gamma}'(y, s; t) : \gamma'(y, t^*, z^*, \mathsf{Coins}') = (c^*, s).$$

Note that $\mathbf{\Gamma}^*(x, y; t)$ and $\mathbf{\Gamma}'(y, s; t)$ are independent events. Also, as we observed above, the probability $\Pr[\mathbf{\Gamma}^*(x, y; t)]$ does not depend on x.

For $s \in \mathcal{S}$, we calculate $\Pr[\mathsf{S}_{x,y} = s]$ by summing over all possible transcripts t, using total probability:

$$\begin{aligned}\Pr[\mathsf{S}_{x,y} = s] &= \sum_t \Pr[\mathsf{S}_{x,y} = s \wedge \mathsf{T}_{x,y} = t] \\ &= \sum_t \Pr[\mathbf{\Gamma}^*(x, y; t) \wedge \mathbf{\Gamma}'(y, s; t)] \\ &= \sum_t \Pr[\mathbf{\Gamma}^*(x, y; t)] \cdot \Pr[\mathbf{\Gamma}'(y, s; t)] \quad \text{(by independence).}\end{aligned}$$

In this last expression, we see that neither $\Pr[\mathbf{\Gamma}^*(x, y; t)]$ nor $\Pr[\mathbf{\Gamma}'(y, s; t)]$ depends on x, which proves the theorem. □

19.8.3 Actively secure identification protocols

As promised, we now show how to use witness independence to design actively secure identification protocols. The construction is quite general. The basic ingredients are a Sigma protocol, along with a one-way key generation algorithm. We also make use of the OR-proof construction in Section 19.7.2.

Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. We will assume that (P, V) is special HVZK and that its challenge space is of the form $\mathcal{C} = \{0, 1\}^n$. These assumptions will allow us to apply the OR-proof construction presented in Section 19.7.2. In the security analysis, we will also need to assume that (P, V) provides special soundness.

As we saw in Section 19.6, to build an identification protocol from (P, V), we also need a one-way key generation algorithm G for the relation $\mathcal{R}$. The identification scheme $\mathcal{I} := (G, P, V)$ is secure against *eavesdropping.* However, without too much more effort, and without making any additional assumptions, we can build an identification scheme that is secure against *active* attacks (as defined in Section 18.6).

First, we build a new Sigma protocol (P', V') by applying the OR-proof construction to $(P_0, V_0) := (P, V)$ and $(P_1, V_1) := (P, V)$. Let $\mathcal{R}' := \mathcal{R}_{\mathrm{OR}}$ be the corresponding relation: a statement for $\mathcal{R}'$ is of the form $Y = (y_0, y_1) \in \mathcal{Y}^2$, and a witness for Y is of the form $X = (b, x) \in \{0, 1\} \times \mathcal{X}$, where $(x, y_b) \in \mathcal{R}$. For a witness $X = (b, x)$, let us call the bit b its *type.*

Second, we build a new key generation algorithm G' for the relation $\mathcal{R}'$. Algorithm G' runs as follows:

```
G':  $(y_0,(x_0,y_0)) \xleftarrow{R} G(),\ (y_1,(x_1,y_1)) \xleftarrow{R} G()$
     $b \xleftarrow{R} \{0,1\}$
     $Y \leftarrow (y_0,y_1)$
     $X \leftarrow (b,x_b)$
     output $(Y,(X,Y))$
```

A key property of G' is that, as random variables, Y and b are independent. That is, after we see the statement Y, we cannot infer if X is $(0,x_0)$ or $(1,x_1)$.

We now prove that the identification protocol $\mathcal{I}' := (G',P',V')$ is secure against active attacks.

Theorem 19.22. *Let (P,V) be a Sigma protocol for an effective relation $\mathcal{R}$ with a large challenge space of the form $\{0,1\}^n$. Assume that (P,V) is special HVZK and provides special soundness. Further, assume that the key generation algorithm G for $\mathcal{R}$ is one-way. Then the identification scheme $\mathcal{I}' := (G',P',V')$ defined above is secure against active attacks.*

> *In particular, suppose $\mathcal{A}$ is an impersonation adversary attacking $\mathcal{I}'$ via an active attack as in Attack Game 18.3, with advantage $\epsilon := \mathrm{ID3}\mathsf{adv}[\mathcal{A},\mathcal{I}']$. Then there exists an efficient adversary $\mathcal{B}$ (whose running time is about* twice *that of $\mathcal{A}$), such that*
>
> $$\mathsf{OWadv}[\mathcal{B},G] \geq \frac{1}{2}(\epsilon^2 - \epsilon/N),$$
>
> *where $N := 2^n$.*

Proof. Let us begin by reviewing how an active impersonation attack against (P',V') works. There are three phases.

Key generation phase. The challenger runs the key generation algorithm G', obtaining a public key $pk' = Y$ and a secret key $sk' = (X,Y)$, and sends pk' to the adversary $\mathcal{A}$.

Active probing phase. The adversary interacts with the prover $P'(sk')$. Here, the challenger plays the role of the prover, while the adversary plays the role of a possibly "cheating" verifier. The adversary may interact concurrently with many instances of the prover.

Impersonation attempt. As in a direct attack, the adversary now interacts with the verifier $V'(pk')$, attempting to make it accept. Here, the challenger plays the role of the verifier, while the adversary plays the role of a possibly "cheating" prover. In this phase, the adversary (acting as prover) supplies a commitment, to which the challenger replies (acting as verifier) with a random challenge. The adversary wins the game if its response to the random challenge yields an accepting conversation.

So let ϵ be the probability that $\mathcal{A}$ wins this game.

We now describe our adversary $\mathcal{B}$ for breaking the one-wayness assumption for G. To start with, $\mathcal{B}$'s challenger computes $(y^*,(x^*,y^*)) \xleftarrow{R} G()$ and gives y^* to $\mathcal{B}$. The goal of $\mathcal{B}$ is to compute a witness for y^*.

Our adversary $\mathcal{B}$ begins by playing the role of challenger to $\mathcal{A}$, running $\mathcal{A}$ once through all three phases. In the key generation phase, $\mathcal{B}$ computes $(pk',sk') = (Y,(X,Y))$ as follows:

```
$b \xleftarrow{R} \{0,1\}$
$(y,(x,y)) \xleftarrow{R} G()$
if $b = 0$
    then $Y \leftarrow (y,y^*)$
    else  $Y \leftarrow (y^*,y)$
$X \leftarrow (b,x)$
```

Observe that the distribution of (pk', sk') is precisely the same as the output distribution of G'.

After running all three phases, $\mathcal{B}$ rewinds $\mathcal{A}$ back to the point in the third phase where the challenger (as verifier) gave $\mathcal{A}$ its random challenge, and gives to $\mathcal{A}$ a fresh random challenge. If this results in two accepting conversations with distinct challenges, then by special soundness, $\mathcal{B}$ can extract a witness $\widehat{X} = (\hat{b}, \hat{x})$ for Y. Moreover, if $\hat{b} \neq b$, then $\hat{x}$ is a witness for y^*, as required.

So it remains to analyze $\mathcal{B}$'s success probability. Now, $\mathcal{B}$ succeeds if it extracts a witness $\widehat{X}$ from $\mathcal{A}$, and $\widehat{X}$ and X have unequal types. By the Rewinding Lemma (Lemma 19.2), we know that $\mathcal{B}$ will extract *some* witness $\widehat{X}$ from $\mathcal{A}$ with probability at least $\epsilon^2 - \epsilon/N$. Moreover, we know that Y by itself reveals nothing about the type of X to $\mathcal{A}$, and witness independence essentially says that the active probing phase reveals nothing more about the type of X to $\mathcal{A}$. Therefore, for any *particular* witness that $\mathcal{B}$ extracts, its type will match that of X with probability 1/2. This means that $\mathcal{B}$'s overall success probability is at least $(\epsilon^2 - \epsilon/N) \times \frac{1}{2}$, as required.

We can make the above argument about $\mathcal{B}$'s success probability a bit more rigorous, if we like, using the definition of witness independence directly (in the form of (19.25)). To this end, we use the letters $\mathbf{X}, \widehat{\mathbf{X}}, \mathbf{Y}$ to denote random variables, and the letters $X, \widehat{X}, Y$ to denote particular values that these random variables might take. If $\mathcal{B}$ fails to extract a witness, we define $\widehat{\mathbf{X}} := \perp$. If σ is $\mathcal{B}$'s success probability, then we have

$$\sigma = \Pr[(\widehat{\mathbf{X}}, \mathbf{Y}) \in \mathcal{R}' \wedge \text{type}(\mathbf{X}) \neq \text{type}(\widehat{\mathbf{X}})].$$

Using total probability, we sum over all $(\widehat{X}, Y) \in \mathcal{R}'$:

$$\begin{aligned}
\sigma &= \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\text{type}(\mathbf{X}) \neq \text{type}(\widehat{\mathbf{X}}) \wedge \widehat{\mathbf{X}} = \widehat{X} \wedge \mathbf{Y} = Y] \\
&= \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\text{type}(\mathbf{X}) \neq \text{type}(\widehat{X}) \mid \widehat{\mathbf{X}} = \widehat{X} \wedge \mathbf{Y} = Y] \cdot \Pr[\widehat{\mathbf{X}} = \widehat{X} \wedge \mathbf{Y} = Y] \\
&= \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\text{type}(\mathbf{X}) \neq \text{type}(\widehat{X}) \mid \mathbf{Y} = Y] \cdot \Pr[\widehat{\mathbf{X}} = \widehat{X} \wedge \mathbf{Y} = Y] \quad \text{(witness independence)} \\
&= \frac{1}{2} \sum_{(\widehat{X},Y)\in\mathcal{R}'} \Pr[\widehat{\mathbf{X}} = \widehat{X} \wedge \mathbf{Y} = Y] \quad \text{(independence of } \mathbf{Y} \text{ and type}(\mathbf{X})) \\
&= \frac{1}{2} \Pr[(\widehat{\mathbf{X}}, \mathbf{Y}) \in \mathcal{R}'] \geq \frac{1}{2}(\epsilon^2 - \epsilon/N). \quad \Box
\end{aligned}$$

Concrete instantiations. The above construction immediately gives us two concrete identification protocols that are secure against active attacks. One, derived from Schnorr, whose security is based on the DL assumption, and the other, derived from GQ, whose security is based from the RSA assumption. These two actively secure protocols are roughly twice as expensive (in terms of computation and bandwidth) as their eavesdropping secure counterparts.

19.8.4 Okamoto's identification protocol

We just saw how to build an identification protocol whose security against active attacks is based on the DL assumption. We now look at a more efficient approach, based on Okamoto's protocol.

Recall Okamoto's protocol (P, V) in Section 19.5.1. In addition to the cyclic group $\mathbb{G}$ of order q generated by $g \in \mathbb{G}$, this protocol also makes use of a second group element $h \in \mathbb{G}$, which we view as a system parameter. The most natural key generation algorithm G for this protocol computes $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$, and outputs $pk = u$ and $sk = ((\alpha, \beta), u)$ where $u := g^\alpha h^\beta \in \mathbb{G}$. This gives us the identification protocol $\mathcal{I}_\text{O} = (G, P, V)$, which we call **Okamoto's identification protocol**. Using the concept of witness independence, it is not hard to show that $\mathcal{I}_\text{O}$ is secure against active attacks.

Theorem 19.23. *Let $\mathcal{I}_\text{O} = (G, P, V)$ be Okamoto's identification protocol, and assume that the challenge space is large. Also, assume that the system parameter h is generated uniformly over $\mathbb{G}$. Then $\mathcal{I}_\text{O}$ is secure against active attacks, assuming the DL assumption holds for $\mathbb{G}$.*

> *In particular, suppose $\mathcal{A}$ is an impersonation adversary attacking $\mathcal{I}_\text{O}$ via an active attack as in Attack Game 18.3, with advantage $\epsilon :=$ ID3adv$[\mathcal{A}, \mathcal{I}_\text{O}]$. Then there exists an efficient adversary $\mathcal{B}$ (whose running time is about* twice *that of $\mathcal{A}$), such that*
>
> $$\mathsf{DLadv}[\mathcal{B}, \mathbb{G}] \geq (1 - 1/q)(\epsilon^2 - \epsilon/N),$$
>
> *where N is the size of the challenge space.*

Proof. The proof has the same basic structure as that of Theorem 19.22.

Suppose $\mathcal{A}$ has advantage ϵ in attacking $\mathcal{I}_\text{O}$ in Attack Game 18.3. Our DL adversary $\mathcal{B}$ receives a random group element $h \in \mathbb{G}$ from its challenger. The goal of $\mathcal{B}$ is to compute $\mathsf{Dlog}_g h$, making use of $\mathcal{A}$.

Our adversary $\mathcal{B}$ begins by playing the role of challenger to $\mathcal{A}$, running $\mathcal{A}$ once through all three phases of Attack Game 18.3. Our adversary $\mathcal{B}$ uses the group element h as the system parameter for Okamoto's protocol, but otherwise follows the logic of the challenger in Attack Game 18.3 without modification:

Key generation phase. $\mathcal{B}$ computes $\alpha, \beta \xleftarrow{R} \mathbb{Z}_q$, $u \leftarrow g^\alpha h^\beta$, and sends the public key $pk := u$ to $\mathcal{A}$, keeping the secret key $sk := ((\alpha, \beta), u)$ to itself.

Active probing phase. $\mathcal{A}$ interacts (possibly concurrently) with several instances of the prover $P(sk)$. The role of these provers is played by $\mathcal{B}$.

Impersonation attempt. $\mathcal{A}$ tries to make the verifier $V(pk)$ accept. The role of the verifier is played by $\mathcal{B}$.

After running all three phases, $\mathcal{B}$ rewinds $\mathcal{A}$ back to the point in the third phase where the verifier gave $\mathcal{A}$ its random challenge, and gives to $\mathcal{A}$ a new, random challenge. If this results in two accepting conversations with distinct challenges, then by special soundness, $\mathcal{B}$ can extract a witness $(\hat{\alpha}, \hat{\beta})$ for u. Moreover, if $(\alpha, \beta) \neq (\hat{\alpha}, \hat{\beta})$, then we have two distinct representations (relative to g and h) of u, and therefore, $\mathcal{B}$ can compute $\mathsf{Dlog}_g h$ as in Fact 10.3.

Our adversary $\mathcal{B}$ succeeds if it extracts a witness from $\mathcal{A}$ that is different from (α, β). By the Rewinding Lemma (Lemma 19.2), we know that $\mathcal{B}$ will extract *some* witness from $\mathcal{A}$ with probability at least $\epsilon^2 - \epsilon/N$. Moreover, u by itself reveals nothing about which of the q possible witnesses for u that $\mathcal{B}$ is using, and witness independence says that the active probing phase reveals nothing more about this witness to $\mathcal{A}$. Therefore, for any *particular* witness that $\mathcal{B}$ extracts from $\mathcal{A}$, the probability that it is equal to (α, β) is $1/q$. This means that $\mathcal{B}$'s overall success probability is at least $(\epsilon^2 - \epsilon/N) \times (1 - 1/q)$, as required. □

19.9 Multi-extractability: another notion of "proof of knowledge"

In Section 19.4.1 we noted that Sigma protocols such as Schnorr's acts as a "proof of knowledge", meaning that a witness can be extracted from a prover using a particular rewinding technique. We used this rewinding technique to prove the security of Schnorr's identification and signature schemes (and similar schemes based on more general Sigma protocols).

The notion of a "proof of knowledge" is often used as a step towards proving security of some candidate scheme (such as security of a signature scheme). We have not attempted to give a general definition for a "proof of knowledge" because the exact definition can depend to a certain degree on the application at hand, and on the exact statement one is trying to prove.

In this section we explore an interesting "proof of knowledge" property that Sigma protocols satsify called **multi-extractability**. This property is also based on a rewinding technique, but the technique is different than the rewinding technique used to prove the security of Schnorr's identification and signature schemes. We shall discuss some applications and limitations of multi-extractability.

Intuitively, multi-extractability means the following. Suppose that during an attack, an adversary is allowed to interact concurrently with many verifiers, supplying each verifier with a statement of the adversary's choice, and at the end of the attack, the adversary makes some of these verifiers accept. Then we can "shake the adversary" (i.e., "rewind" or "re-run" the adversary) over and over again until it finally "spits out what it knows" (i.e., we extract witnesses for all statements supplied to the accepting verifiers).

19.9.1 Multi-extractable Sigma protocols

Let Π be a Sigma protcol for an effective relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. We assume that conversations are of the form (t, c, z), where $t \in \mathcal{T}$ is a "commitment", $c \in \mathcal{C}$ is a "challenge", and $z \in \mathcal{Z}$ is a "response". Suppose $\mathcal{A}$ is an adversary that makes a series of queries to a *verifier oracle*, where each query consists of a statement and a commitment, to which the verifier oracle responds with a random challenge. That is, for $i = 1, 2, \ldots$, the ith verifier oracle query is of the form $(y_i, t_i) \in \mathcal{Y} \times \mathcal{T}$, to which the verifier oracle responds with a random challenge $c_i \in \mathcal{C}$. Suppose that the adversary makes a total of r such queries. After this, $\mathcal{A}$ outputs

$$(\mathcal{J},\ \{z_j\}_{j \in \mathcal{J}}),$$

where $\mathcal{J} \subseteq \{1, \ldots, r\}$, and for each $j \in \mathcal{J}$, the conversation (t_j, c_j, z_j) is an accepting conversation for the statement y_j.

Roughly speaking, **multi-extractability** means that there is an efficient algorithm, called a **multi-extractor**, that can be used to augment the execution of $\mathcal{A}$ so that for each $j \in \mathcal{J}$, it also outputs a witness x_j for y_j. The multi-extractor does so by "rewinding" or "re-running" $\mathcal{A}$, and is allowed to fail with some *prescribed failure probability* θ. Moreover, the number of times it "rewinds" or "re-runs" the adversary should not be too large (i.e., bounded by a polynomial $1/\theta$ and the number of verifier oracle queries).

We now define multi-extractability more precisely. We say that $\mathcal{A}$ **is B-bounded** if for all executions of $\mathcal{A}$, it makes at most $r \leq B$ verifier oracle queries. We shall assume that $\mathcal{A}$ is probabilistic. However, for technical reasons, we shall treat $\mathcal{A}$ as a deterministic algorithm that explicitly requests random bits from a random bit generator. However, these random bit requests need not be made at the beginning of $\mathcal{A}$'s execution — they may be arbitrarily interleaved with

$\mathcal{A}$'s verifier oracle queries. The way a multi-extractor MEx is allowed to interact with an adversary $\mathcal{A}$ as above is described in the following experiment.

Multi-extraction experiment. *This experiment involves a B-bounded adversary $\mathcal{A}$ as above, a multi-extractor MEx, and a prescribed failure probability θ.*

Stage 1: *We initially run $\mathcal{A}$ to completion, responding to verifier oracle queries with random challenges, and fulfilling all random bit requests with random bits.*

Stage 2: *We initialize MEx with B and θ, along with a **transcript** of $\mathcal{A}$'s initial run. This transcript consists of a list of $\mathcal{A}$'s verifier oracle query/response pairs $((y_1, t_1), c_1), \ldots, ((y_r, t_r), c_r)$, as well as $\mathcal{A}$'s output $(\mathcal{J}, \{z_j\}_{j \in \mathcal{J}})$.*

Recall that $\mathcal{A}$'s output must satisfy $\mathcal{J} \subseteq \{1, \ldots, r\}$, and for each $j \in \mathcal{J}$, the conversation (t_j, c_j, z_j) is an accepting conversation for y_j. The transcript does not *include any information about $\mathcal{A}$'s random bit requests.*

The multi-extractor MEx is now allowed to re-run *$\mathcal{A}$ many times. In each such re-run, $\mathcal{A}$ specifies a **rewinding index** $j \in \mathcal{J}$, and then we do the following:*

- *we rewind $\mathcal{A}$'s execution to the state in the initial run at the point where it makes its jth verifier oracle query, and then run it forward:*
  - *responding to the jth and subsequent verifier oracle queries with fresh random challenges;*
  - *fulfilling subsequent random bit requests with fresh random bits;*
- *when $\mathcal{A}$ finishes, we give MEx the transcript from this re-run of $\mathcal{A}$.*

After performing all of these re-runs, MEx either outputs fail or $\{x_j\}_{j \in \mathcal{J}}$, where x_j is a witness for y_j for each $j \in \mathcal{J}$.

Note that if in the initial run, $\mathcal{A}$ does not output anything (i.e., $\mathcal{J} = \emptyset$), then MEx need not produce any witnesses (i.e., MEx may output $\emptyset$, rather than fail).

Definition 19.9 (Multi-extractability). *A Sigma protocol Π is called **multi-extractable** if there exists a multi-extractor MEx such that for every*

(a) poly-bounded B,

(b) prescribed failure probability θ with $1/\theta$ poly-bounded,

(c) B-bounded efficient adversary $\mathcal{A}$,

there exists a negligible value η such that in the above multi-extraction experiment, we have:

(i) the probability that MEx fails is at most $\theta + \eta$,

(ii) the number of times that MEx re-runs $\mathcal{A}$ in the worst case *is bounded by a fixed polynomial in $1/\theta$ and B (and the security parameter), and*

(iii) the running time of all other computations performed by MEx is bounded in the worst case *by a fixed polynomial in $1/\theta$ and B (and the security parameter).*

When the multi-extraction experiment is run under the constraints of Definition 19.9, the entire experiment will be executed in time polynomial in the implicit security parameter. This means that in designing *MEx*, we can make computational assumptions, if necessary. The negligible "error term" η in condition (i) of the definition can be used, for example, to model a situation where *MEx*'s failure probability is at most θ, except if some failure condition associated with the computational assumption occurs (see Exercise 19.32). It can also be used to simply model a situation where *MEx*'s failure probability is unconditionally at most θ for all sufficiently large values of the security parameter (which we will exploit below in Theorem 19.24).

The above definition of multi-extractability has a number of very particular features. These are included (a) because they can be achieved in a number of settings, and (b) because they are sufficient to prove some interesting results. As we already said, there is no one "right" definition for a "proof of knowledge", but this one is nice because it is useful in some interesting applications.

The next theorem says that under appropriate conditions, Sigma protocols are multi-extractable.

Theorem 19.24 (Sigma protocols are multi-extractable). *Let Π be Sigma protocol with a large challenge space that provides special soundness. Then Π is multi-extractable.*

> *In particular, there exists a multi-extractor, such that for every B-bounded adversary, and for every prescribed error probability θ, multi-extractor re-runs the adversary $O(B/\theta)$ times, and fails with probability at most θ provided $\theta \geq 2B/N$, where N is the size of Π's challenge space.*

Proof. We have a Sigma protocol Π for a relation consisting of pairs (x, y), where x is a witness and y is a statement. A conversation for Π is a triple (t, c, z), where c comes from a challenge space $\mathcal{C}$ of size N, where $1/N$ is negligible. We are assuming that Π provides special soundness, which means that there is an efficient algorithm that given two accepting conversations (t, c, z) and (t, c', z') for a statement y, where $c \neq c'$, outputs a witness x for y.

Assume a B-bounded adversary $\mathcal{A}$ that makes a sequence of verifier oracle queries, where the query/response pairs are

$$((y_1, t_1), c_1), \ldots, ((y_r, t_r), c_r),$$

and outputs $(\mathcal{J}, \{z_j\}_{j\in\mathcal{J}})$, where $\mathcal{J} \subseteq \{1, \ldots, r\}$, and for each $j \in \mathcal{J}$, the conversation (t_j, c_j, z_j) is an accepting conversation for y_j.

Any run of $\mathcal{A}$ is completely determined by the random bits it requests and the *challenge vector*

$$(c_1, \ldots, c_B),$$

where c_i is the response to the ith verifier oracle query, for $i = 1, \ldots, B$.

Our goal is to design a multi-extractor *MEx* that works as in the multi-extraction experiment above. At the end, *MEx* should output either fail or $\{x_j\}_{j\in\mathcal{J}}$, where x_j is a witness for y_j for each $j \in \mathcal{J}$, corresponding to the oracle query/response pairs $((y_1, t_1), c_1), \ldots, ((y_r, t_r), c_r)$ and $\mathcal{A}$'s output $(\mathcal{J}, \{z_j\}_{j\in\mathcal{J}})$ in the initial run of $\mathcal{A}$.

Our multi-extractor *MEx* will be given the bound B and a prescribed failure probability θ, and will re-run $\mathcal{A}$ at most $\lceil 2B/\theta \rceil$ times in total. Moreover, *MEx* will fail with probability at most θ assuming

$$\theta \geq \frac{2B}{N}. \tag{19.28}$$

(Note that since $1/N$ is negligible, and $1/\theta$ and B are poly-bounded, (19.28) will hold for all sufficiently large values of the security parameter.)

MEx works as follows:

- Let $(c_1, \ldots, c_B)$ be the challenge vector of $\mathcal{A}$'s initial run.
- *MEx* processes each output z_j for $j \in \mathcal{J}$ in its initial run as follows:
  - *MEx* then repeatedly re-runs $\mathcal{A}$ with rewinding index j. In each such re-run, the corresponding challenge vector is

$$(\ c_1, \ldots, c_{j-1}, \ \tilde{c}_j, \ldots, \tilde{c}_B \ ),$$

    where $c_1, \ldots, c_{j-1}$ are the same as in the initial run, and $\tilde{c}_j, \ldots, \tilde{c}_B$ are fresh random values, chosen uniformly and independently from the challenge space $\mathcal{C}$. Note also that in each re-run, the random bits obtained by $\mathcal{A}$ up through the jth verifier oracle query are the same as in the initial run, while subsequent random bits are freshly generated. In particular, in each such re-run, the jth verifier oracle query is the same as in the initial run, namely, (y_j, t_j).
  - This is repeated until a challenge $\tilde{c}_j$ is found such that $\tilde{c}_j \neq c_j$ and $\mathcal{A}$ outputs $(\mathcal{K}, \{\tilde{z}_k\}_{k \in \mathcal{K}})$ such that $\mathcal{K}$ contains j, which means that such that $(t_j, \tilde{c}_j, \tilde{z}_j)$ is an accepting conversation for the statement y_j. By special soundness, this allows *MEx* to extract a witness x_j for the statement y_j.
- The above procedure is carried out for each $j \in \mathcal{J}$ output by $\mathcal{A}$. If the *total* number of re-runs reaches $\lceil 2B/\theta \rceil$ without finding all required witnesses, *MEx* stops and outputs fail.

Claim 1. MEx fails with probability at most θ, assuming (19.28).

To prove this, let us first consider a modified version of *MEx*, which we call *MEx*$_1$. The only difference between *MEx* and *MEx*$_1$ is that *MEx*$_1$ implements the failure condition a bit differently. Instead of failing if the total number of re-runs reaches $\lceil 2B/\theta \rceil$ without finding all witnesses, it works as follows:

> *for each $j \in \mathcal{J}$, it re-runs $\mathcal{A}$ repeatedly with rewinding index j at most $\lceil 2B/\theta \rceil$ times or until it finds a witness, whichever comes first.*

The reason for considering *MEx*$_1$ is that its behavior is easier to model mathematically, as the attempt to extract a witness for one statement has no impact on the the attempt to to extract a witness for another statement.

Claim 2. The *expected value* of the total number of re-runs performed by *MEx*$_1$ is at most $2B$, assuming (19.28).

Claim 1 follows from Claim 2. Indeed, we first observe that *MEx* and *MEx*$_1$ proceed identically until such point that *MEx* fails. Moreover, if *MEx* fails, then *MEx*$_1$ performs at least $\lceil 2B/\theta \rceil$ re-runs, and by Claim 2 and Markov's inequality, this happens with probability at most θ.

We now prove Claim 2. For $i = 1, \ldots, B$, we consider the number R_i of re-runs that the ith verifier oracle query may generate. We will show that the expected value of R_i is at most 2, from which the claim will follow.

To show that this holds, we show that it holds conditioned on any fixed values of

$$c_1, \ldots, c_{i-1}, \rho$$

where $c_1, \ldots, c_{i-1}$ represent the first $i-1$ verifier oracle responses of $\mathcal{A}$'s initial run, and ρ is a bit string that represents the random bits obtained by $\mathcal{A}$ in its initial run up to its ith verifier oracle query. We may assume that the fixed values $c_1, \ldots, c_{i-1}, \rho$ lead $\mathcal{A}$ to make an ith verifier oracle query (y_i, t_i), which is completely determined by these fixed values.

Starting from the state determined by the fixed values $c_1, \ldots, c_{i-1}, \rho$, the remainder of $\mathcal{A}$'s execution is determined by the *random variables*

$$c_i, \underbrace{c_{i+1} \ldots, c_B, \rho'}_{=:\boldsymbol{d}},$$

where $c_i, c_{i+1}, \ldots, c_B$ represent the remaining verifier oracle responses, and ρ' is a bit string that represents the remaining random bits obtained by $\mathcal{A}$. We call the random variable $(c_i, \boldsymbol{d})$ a *continuation.* Define the random variable $G(c_i, \boldsymbol{d}) := 1$ if the continuation $(c_i, \boldsymbol{d})$ makes $\mathcal{A}$ generate an output of the form $(\mathcal{J}, \cdot)$ such that $\mathcal{J}$ contains i, and $G(c_i, \boldsymbol{d}) := 0$ otherwise. Let $\gamma := \Pr[G(c_i, \boldsymbol{d}) = 1]$.

Now let $(c_i, \boldsymbol{d})$ denote $\mathcal{A}$'s continuation in its initial run. Then by total expectation, we have

$$\begin{aligned} E[R_i] &= E[R_i \mid G(c_i, \boldsymbol{d}) = 1] \Pr[G(c_i, \boldsymbol{d}) = 1] + E[R_i \mid G(c_i, \boldsymbol{d}) = 0] \Pr[G(c_i, \boldsymbol{d}) = 0] \\ &= E[R_i \mid G(c_i, \boldsymbol{d}) = 1] \cdot \gamma, \end{aligned}$$

since $R_i = 0$ whenever $G(c_i, \boldsymbol{d}) = 0$.

There are two cases to consider.

Case 1: $\gamma < 2/N$.

If $G(c_i, \boldsymbol{d}) = 1$ occurs, MEx_1 will re-run $\mathcal{A}$ at most $\lceil 2B/\theta \rceil$ times. So we have

$$E[R_i] = E[R_i \mid G(c_i, \boldsymbol{d}) = 1] \cdot \gamma < \lceil 2B/\theta \rceil \cdot \frac{2}{N},$$

since $\gamma < 2/N$. Moreover,

$$\lceil 2B/\theta \rceil \cdot \frac{2}{N} \leq 2 \iff \lceil 2B/\theta \rceil \leq N \iff 2B/\theta \leq N,$$

which holds by (19.28).

Case 2: $\gamma \geq 2/N$.

In any re-run of $\mathcal{A}$, we run $\mathcal{A}$ to completion using an independent continuation $(\tilde{c}_i, \tilde{\boldsymbol{d}})$. Moreover, we have

$$\Pr[G(c_i, \boldsymbol{d}) = 1 \wedge G(\tilde{c}_i, \tilde{\boldsymbol{d}}) = 1 \wedge \tilde{c}_i \neq c_i] \geq \gamma^2 - \gamma/N.$$

This follows from Exercise 19.5(a). So we have

$$\Pr[G(\tilde{c}_i, \tilde{\boldsymbol{d}}) = 1 \wedge \tilde{c}_i \neq c_i \mid G(c_i, \boldsymbol{d}) = 1] \geq \gamma - 1/N \geq \gamma/2,$$

since $\gamma \geq 2/N$. It follows that

$$E[R_i \mid G(c_i, \boldsymbol{d}) = 1] \leq 2/\gamma.$$

Therefore

$$E[R_i] = E[R_i \mid G(c_i, \boldsymbol{d}) = 1] \cdot \gamma \leq (2/\gamma) \cdot \gamma = 2.$$

That completes the proof of the theorem. □

19.9.2 Applications and limitations

To illustrate some applications of multi-extractability, as well as its limitations, we consider a "folklore" construction for a chosen ciphertext secure encryption scheme.

Recall from Chapter 12 that a public key encryption scheme is secure against chosen ciphertext attack, or CCA secure, if an adversary cannot learn anything about an unknown message m given its encryption c, even the adversary is allowed to query an "encryption oracle" that will decrypt any ciphertexts given to it other than c itself.

The folklore construction is a technique called "encrypt then prove". The idea is to start with a semantically secure encryption scheme and augment each ciphertext with "proof of knowledge". The "proof of knowledge" is derived from an appropriate Sigma protocol using the Fiat-Shamir heuristic (see Section 19.6.1). Assuming the Sigma protocol is honest verifier zero knowledge (HVZK), the proof should not leak any information about the plaintext. The "proof of knowledge" property should, at least intuitively, ensure that interacting with a decryption oracle does not help an attacker, since the attacker must already "know" what the response from the decryption oracle will be. As we will see, in some limited attack scenarios, this is indeed the case; however, in a general CCA attack, a proof of this seems out of reach (although we know of no attack, either).

19.9.2.1 A concrete "encrypt then prove" scheme

To make things concrete, let us start with the multiplicative ElGamal scheme $\mathcal{E}_{\mathrm{MEG}}$. We have a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$.

- The secret key is a random $\alpha \in \mathbb{Z}_q$ and the public key is $u := g^\alpha \in \mathbb{G}$.
- An encryption of $m \in \mathbb{G}$ is $(v, e) \in \mathbb{G}^2$, computed as $\beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q$, $v \leftarrow g^\beta$, $w \leftarrow u^\beta$, $e \leftarrow w \cdot m$.
- A ciphertext $(v, e) \in \mathbb{G}^2$ decrypts to $m \in \mathbb{G}$, computed as $w \leftarrow v^\alpha$, $m \leftarrow e/w$.

We know that $\mathcal{E}_{\mathrm{MEG}}$ is semantically secure under the DDH (see Exercise 11.5) but is completely insecure against a chosen ciphertext attack (see Exercise 12.1).

We now augment $\mathcal{E}_{\mathrm{MEG}}$ so that a ciphertext is of the form (v, e, π), where π is a "proof of knowledge" of $\beta \in \mathbb{Z}_q$ such that $v = g^\beta$. Intuitively, if the encryptor "knows" β, then he must also "know" the plaintext $m = e/u^\beta$. When decrypting such a ciphertext, we always check that the proof is valid before computing the plaintext.

To design such a "proof of knowledge", we start with a Sigma protocol Π for the relation

$$\mathcal{R} = \{\ (\beta, (u, v, e)) \in \mathbb{Z}_q \times \mathbb{G}^3 :\ g^\beta = v\ \}.$$

For this, we can essentially use Schnorr's Sigma protocol (see Example 19.1). The challenge space $\mathcal{C}$ is a subset of $\mathbb{Z}_q$ of size N, and we will assume that $1/N$ is negligible. Π works as follows:

1. the prover computes $\beta_{\mathrm{t}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q,\ v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{t}}}$, and sends v_{t} to the verifier;
2. the verifier computes $c \xleftarrow{\mathrm{R}} \mathcal{C}$, and sends c to the prover;
3. the prover computes $\beta_{\mathrm{z}} \leftarrow \beta_{\mathrm{t}} + \alpha c \in \mathbb{Z}_q$, and sends β_{z} to the verifier;
4. the verifier checks that $g^{\beta_{\mathrm{z}}} = v_{\mathrm{t}} \cdot v^c$.

Π provides special soundness and is special HVZK (just as we observed in Section 19.4 for the Sigma protocol corresponding to Schnorr's identification protocol).

In applying the Fiat-Shamir heuristic to Π, the challenge will be computed by applying a hash function H to the statement (u, v, e) together with the first flow v_{t} of the Sigma protocol. From this, we obtain the **augmented multiplicative ElGamal encryption scheme $\mathcal{E}_{\mathrm{aMEG}}$**:

- The secret key is a random $\alpha \in \mathbb{Z}_q$ and the public key is $u := g^\alpha \in \mathbb{G}$.
- An encryption of $m \in \mathbb{G}$ is $(v, e, \pi) = (v, e,\ (v_{\mathrm{t}}, \beta_{\mathrm{z}})\ ) \in \mathbb{G} \times \mathbb{G} \times (\mathbb{G} \times \mathbb{Z}_q)$, computed as

  $$\beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q,\ v \leftarrow g^\beta,\ w \leftarrow u^\beta,\ e \leftarrow w \cdot m,$$

  and

  $$\beta_{\mathrm{t}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q,\ v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{t}}},\ c \leftarrow H(\ (u, v, e),\ v_{\mathrm{t}}\ ),\ \beta_{\mathrm{z}} \leftarrow \beta_{\mathrm{t}} + \beta c.$$

- A ciphertext $(v, e, \pi) = (v, e,\ (v_{\mathrm{t}}, \beta_{\mathrm{z}})\ ) \in \mathbb{G} \times \mathbb{G} \times (\mathbb{G} \times \mathbb{Z}_q)$ is decrypted as follows:

```
$c \leftarrow H(\ (u, v, e),\ v_{\mathrm{t}}\ )$
if $g^{\beta_{\mathrm{z}}} = v_{\mathrm{t}} \cdot v^c$
    then $w \leftarrow v^\alpha$, $m \leftarrow e/w$, output $m$
    else  output reject.
```

Note that the only place the group element $e \in \mathbb{G}$ in a ciphertext (v, e, π) is used in constructing or verifying the proof π is as an input to the hash H. This is essential, however, as otherwise the encryption scheme $\mathcal{E}_{\mathrm{aMEG}}$ would be trivially insecure against a chosen ciphertext attack. Indeed, in this case, given an encryption of (v, e, π) of m, an attacker could ask the decryption oracle for a decryption of $(v, e \cdot g, \pi)$, obtaining $m \cdot g$, from which it can compute m. Including e in the hash prevents this type of attack. But the question remains: can we actually *prove* that $\mathcal{E}_{\mathrm{aMEG}}$ is CCA secure (under the DDH assumption, modeling H as a random oracle)?

19.9.2.2 Security properties of "encrypt then prove"

We now examine in more detail the question of whether the "encrypt then prove" construction yields a CCA encryption scheme. To keep things concrete, we focus on the scheme $\mathcal{E}_{\mathrm{aMEG}}$ in Section 19.9.2.1 presented above, although all of our analysis quite generally applies to any "encrypt then prove" based encryption scheme. (See Exercise 20.28.)

The short answer to this question is that while there is no known chosen ciphertext attack on $\mathcal{E}_{\mathrm{aMEG}}$, it seems unlikely that we can prove this; however, we can prove security of $\mathcal{E}_{\mathrm{aMEG}}$ under a very restricted form of chosen ciphertext attack.

Definition of CCA security. Before going further, let us recall more precisely what it means for $\mathcal{E}_{\mathrm{aMEG}}$ to be CCA secure, adapting the notation and terminology of Chapter 12 to our particular setting. We shall work with the bit-guessing version of 1CCA security defined in Section 12.1, which is equivalent to general CCA security. In the attack game defining 1CCA security where H is modeled as a random oracle, we have an adversary $\mathcal{A}$ that is allowed to interact with a challenger as follows.

Key generation step: first, the challenger generates a random secret key $\alpha \in \mathbb{Z}_q$, computes the public key $u \leftarrow g^\alpha \in \mathbb{G}$, and sends u to $\mathcal{A}$.

Pre-challenge phase: second, $\mathcal{A}$ submits a series of *decryption queries* to the challenger; each such query is a ciphertext (v^*, e^*, π^*), to which the challenger responds with the decryption of (v^*, e^*, π^*) under the secret key α (invoking the random oracle to validate π^*).

Challenge step: third, $\mathcal{A}$ submits a single *encryption query* $(m_0, m_1) \in \mathbb{G}^2$ to the challenger; the challenger chooses a random $b \in \{0, 1\}$, encrypts m_b under the public key u, computing the "challenge ciphertext" (v, e, π) (invoking the random oracle to compute π), and sends (v, e, π) to $\mathcal{A}$.

Post-challenge phase: fourth, the adversary again submits a series of decryption queries to the challenger; each such decryption query is a ciphertext $(v^*, e^*, \pi^*) \neq (v, e, \pi)$, to which the challenger responds with the decryption of (v^*, e^*, π^*) under the secret key α (invoking the random oracle to validate π^*).

Output step: at the end of the game, the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Random oracle queries: the adversary may query the random oracle directly, any number of times, at any point in the game between the key generation and output steps.

Adversary $\mathcal{A}$'s **1CCA bit-guessing advantage** is defined to be $|\Pr[\hat{b} = b] - 1/2|$. **1CCA security for $\mathcal{E}_{\text{aMEG}}$** means that $\mathcal{A}$'s 1CCA bit-guessing advantage is negligible for all efficient adversaries $\mathcal{A}$.

Parallel 1CCA security. One can prove that $\mathcal{E}_{\text{aMEG}}$ satisfies a weaker notion of 1CCA security called **parallel 1CCA security**. This notion of security restricts the adversary's ability to query the decryption oracle. In this restricted attack game, the adversary is allowed to request the decryption of many ciphertexts, *but is required to submit all of these ciphertexts as a single batch.* So for $\mathcal{E}_{\text{aMEG}}$, this means the adversary is allowed to make only one query to the decryption oracle of the form

$$(v_1^*, e_1^*, \pi_1^*), \ \ldots, \ (v_n^*, e_n^*, \pi_n^*),$$

and the decryption oracle will decrypt all of these. This decryption request can come either before or after the challenge step in the attack game — if it comes after, and the target ciphertext is (v, e, π), then we must have $(v_k^*, e_k^*, \pi_k^*) \neq (v, e, \pi)$ for each $k = 1, \ldots, n$. Without loss of generality, we may assume that each (v_k^*, e_k^*) is distinct and that each proof π_k^* is valid.

Theorem 19.25. $\mathcal{E}_{\text{aMEG}}$ *is parallel 1CCA secure under the DDH assumption, if we model H as a random oracle.*

Proof sketch. Since the Sigma protocol Π provides special soundness and the challenge space is of size N, where $1/N$ is negligible, Theorem 19.24 applies, and so this Sigma protocol is multi-extractable. So there is a multi-extractor MEx for Π that satisfies Definition 19.9.

We know that the non-augmented scheme $\mathcal{E}_{\text{MEG}}$ is semantically secure under the DDH assumption. So to prove the theorem, we shall show that if the augmented scheme $\mathcal{E}_{\text{aMEG}}$ is not parallel 1CCA secure, then the non-augmented scheme $\mathcal{E}_{\text{MEG}}$ is not semantically secure. To this end, assume $\mathcal{E}_{\text{aMEG}}$ is not parallel 1CCA secure. This means there exists there exists an efficient adversary $\mathcal{A}$ whose bit-guessing advantage in the parallel 1CCA attack game is non-negligible. This implies there exists δ such that $1/\delta$ is poly-bounded and $\mathcal{A}$'s advantage is at least δ. (Technically speaking, $\mathcal{A}$'s advantage is at least δ for infinitely many settings of the security parameter (see Example 2.11), but this is sufficient for our purposes.) Our goal is to use $\mathcal{A}$ to build another efficient adversary $\mathcal{B}$ that breaks the semantic security of $\mathcal{E}_{\mathrm{MEG}}$.

Recall that in the parallel 1CCA attack game defined above, $\mathcal{A}$ is allowed to make one encryption query and one batch decryption query, which may come either before or after the encryption query. We shall assume that $\mathcal{A}$ makes its batch decryption query in the after its encryption query. This is the most interesting case to consider, and we leave the other case to the reader.

Let B be a bound on the number of random oracle queries that $\mathcal{A}$ makes, so that B is also poly-bounded. For simplicity, we assume that $\mathcal{A}$ queries the random oracle at most once on any given input. Without loss of generality, we assume that $\mathcal{A}$ verifies the proofs in all of the ciphertexts in its batch decryption query, and only submits ciphertexts with valid proofs. In particular, we assume that $\mathcal{A}$ has already queried the random oracle at all inputs required to validate these proofs prior to making its batch decryption query. We also assume that $\mathcal{A}$ makes all of its own random choices based on a random bit string $\rho_{\mathcal{A}}$, and is otherwise deterministic.

We begin by constructing a bounded adversary $\mathcal{A}'$ that we can use in the multi-extractability experiment defined above. Recall that such an adversary must make explicit requests for random bits, and is otherwise deterministic. The essential idea is that $\mathcal{A}'$ play the role of both $\mathcal{A}$ and the challenger in the 1CCA attack game, and will forward all random oracle queries made by $\mathcal{A}$ to its own verifier oracle – there is one exception to this, as indicated below in Step 4. In more detail, adversary $\mathcal{A}'$ runs as follows:

Step 1: $\mathcal{A}'$ requests random bit strings $\rho_{\mathcal{A}}$ and ρ_{kg}. The string $\rho_{\mathcal{A}}$ is the randomness for $\mathcal{A}$, while ρ_{kg} is the randomness for the key-generation step in the 1CCA attack game.

$\mathcal{A}'$ then runs the key generation step of the 1CCA attack game, computing the secret key α from ρ_{kg}, and sending the public key $u := g^{\alpha}$ to $\mathcal{A}$. During this time, $\mathcal{A}'$ forwards all random oracle queries to its own verifer oracle.

Step 2: $\mathcal{A}'$ then runs $\mathcal{A}$ up to the point where $\mathcal{A}'$ makes its encryption query $(m_0, m_1) \in \mathbb{G}^2$.

Step 3: At this point (and no earlier!), $\mathcal{A}'$ requests a random string ρ_{enc} used by the challenger in the 1CCA attack game to process the encryption query. Specifically, ρ_{enc} is used to derive the random bit b chosen by the challenger, as well as the random value β used in the non-augmented ciphertext (v, e).

Next, $\mathcal{A}'$ requests a random string ρ_{prv} that is used to generate a simulated proof $\pi = (v_{\mathrm{t}}, \beta_{\mathrm{z}})$ using the HVZK property of Schnorr's Sigma protocol. Specifically, ρ_{prv} is used to generate random $c \in \mathcal{C}$ and $\beta_{\mathrm{z}} \in \mathbb{Z}_q$, and effectively sets the value of the random oracle H at the point $((u, v, e), v_{\mathrm{t}})$ to c, where $v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{z}}}/v^{c}$. This step my fail (but only with negligible probability) if $\mathcal{A}$ previously queried H at this same point (in which case $\mathcal{A}'$ will abort).

$\mathcal{A}'$ then sends (v, e, π) to $\mathcal{A}$.

Step 4: Finally, $\mathcal{A}'$ runs $\mathcal{A}$ up to the point where $\mathcal{A}$ makes its batch decryption query

$$(v_1^*, e_1^*, \pi_1^*),\ \ldots,\ (v_n^*, e_n^*, \pi_n^*).$$

During this time, $\mathcal{A}'$ forwards all random oracle queries to its own verifer oracle, except that if $\mathcal{A}$ subsequently queries H at the point $((u, v, e), v_{\mathrm{t}})$ used to construct the proof in Step 3, $\mathcal{A}'$ will return the corresponding value c to $\mathcal{A}$.

Under our assumptions, and by the fact that Schnorr's Sigma protocol has unique responses (see Exercise 19.14), the following holds:

for each $k = 1, \ldots, n$, if $\pi_k^* = (v_{\mathrm{t}k}^*, \beta_{\mathrm{z}k}^*)$, then there is a unique verifier oracle query of the form $((u, v_k^*, e_k^*), v_{\mathrm{t}k}^*)$ — in particular, $((u, v_k^*, e_k^*), v_{\mathrm{t}k}^*) \neq ((u, v, e), v_{\mathrm{t}})$.

For each such verifier oracle query, $\mathcal{A}'$ outputs a corresponding value $\beta_{\mathrm{z}k}^*$. More precisely, suppose that for each $k = 1, \ldots, n$, the kth ciphertext corresponds to the $j(k)$th verifier oracle query. Then $\mathcal{A}$ outputs $(\mathcal{J}, \{z_j\}_{j \in \mathcal{J}})$, where $\mathcal{J} = \{j(k) : k = 1, \ldots, n\}$ and $z_{j(k)} = \beta_{\mathrm{z}k}^*$ for $k = 1, \ldots, n$.

Now consider the execution of the multi-extraction experiment with the B-bounded adversary $\mathcal{A}'$ and multi-extractor *MEx* with prescribed error probability $\theta := \delta/2$. In this experiment, we run $\mathcal{A}'$ to completion, and then we run *MEx*, which either outputs corresponding witnesses $\beta_1^*, \ldots, \beta_n^*$ or fails. The probability that this experiment fails is at most $\delta/2$ plus some negligible amount. If the experiment succeeds, then from these witnesses, we can readily decrypt all of the ciphertexts

$$(v_1^*, e_1^*, \pi_1^*), \ \ldots, \ (v_n^*, e_n^*, \pi_n^*).$$

Note that $\mathcal{A}'$ only makes verifier oracle queries in Steps 2 and 4. This implies that in any re-run of $\mathcal{A}'$, we always use the original string ρ_{kg} from the initial run of $\mathcal{A}$, and we either

- start from a verifier oracle query made in Step 2, running Step 3 of $\mathcal{A}'$ using a freshly generated random strings $\tilde{\rho}_{\mathrm{enc}}$ and $\tilde{\rho}_{\mathrm{prv}}$, in place of the original strings ρ_{enc} and ρ_{prv} from the initial run of $\mathcal{A}'$, or

- start from a verifier oracle query made in Step 4, and use the original strings ρ_{enc} and ρ_{prv} from the initial run of $\mathcal{A}'$.

It follows that we can convert the multi-extraction experiment with $\mathcal{A}'$ into an adversary $\mathcal{B}$ that attacks the semantic security of the non-augmented scheme $\mathcal{E}_{\mathrm{MEG}}$ as in the bit-guessing version of Attack Game 11.1. The idea is that the random strings ρ_{kg} and ρ_{enc} in the multi-extraction experiment are moved to the challenger in the the bit-guessing version of Attack Game 11.1, while the remaining logic of the multi-extraction experiment is moved to $\mathcal{B}$. Adversary $\mathcal{B}$ uses the extracted witnesses $\beta_1^*, \ldots, \beta_n^*$ to decrypt the ciphertexts submitted by $\mathcal{A}$ for decryption. If this procedure fails for any reason, $\mathcal{B}$ outputs a random bit; otherwise, $\mathcal{B}$ gives the decryptions of these ciphertexts to $\mathcal{A}$ and outputs whatever $\mathcal{A}$ outputs. Adversary $\mathcal{B}$ re-runs $\mathcal{A}$ at most $O(B/\delta)$ times and has bit-guessing advantage at least $\delta/2 - \epsilon$ for some negligible quantity ϵ. (Again, technically speaking, the advantage of $\mathcal{B}$ is at least $\delta/2 - \epsilon$ for infinitely many settings of the security parameter, which is sufficient for our purposes.) □

Note that the security reduction in the proof of Theorem 19.25 is very loose: given an adversary $\mathcal{A}$ attacking $\mathcal{E}_{\mathrm{aMEG}}$ in the parallel 1CCA attack game with advantage δ and making B random oracle queries, we get an adversary that breaks the DDH with advantage roughly $\delta/2$, but which has to re-run $\mathcal{A}$ on the order of B/δ times.

Parallel 1CCA security allows an attacker to make one encryption query and one batch decryption query. Analogous to Theorem 12.1, we can use a simple hybrid argument to show that this implies security against an attacker who can make many encryption queries and one batch decryption query (but this makes the security reduction for $\mathcal{E}_{\mathrm{aMEG}}$ even looser).

It also turns out that parallel 1CCA security is formally *equivalent* to the property of *non-malleability*, which we discussed informally in Section 12.2.1 (but to prove this, one first needs a formal definition of non-malleability, which we do not present here). In any case, we have already seen in Chapter 12 very efficient encryption schemes that provide full CCA security with a much tighter reduction under similar assumptions. Nevertheless, $\mathcal{E}_{\mathrm{aMEG}}$ has some some properties that are still useful — we will later examine a very similar encryption scheme in the context of voting protcols in Section 20.3.1. However, there are very similar encryption schemes which achieve full CCA security under the DDH assumption with a much tighter security reduction, and which are only a bit less efficient (see Exercise 20.20).

On the limitations of rewinding techniques. While we have shown that $\mathcal{E}_{\mathrm{aMEG}}$ is parallel 1CCA secure, it seems impossible to prove that it provides full CCA security, at least using the rewinding techniques that we have developed. We will illustrate the obstruction to proving full CCA security with an example. While this example is very specific, it highlights a fundamental and very general limitation of using rewinding techniques in security proofs.

In the full CCA attack game, the adversary is allowed to make a sequence of possibly dependent decryption queries. That is, each decryption query may depend on the result of the previous decryption queries. At the very least, the adversary may simply refuse to submit its next decryption query before it obtains replies to its earlier queries. So if we want to use some kind of technique to extract the plaintext from the ciphertext, we will have to extract the plaintext immediately for each decryption query — we cannot wait, as we did in the proof of Theorem 19.25, for the adversary to first submit all of its decryption queries. However, if we try to use a rewinding technique as we did in the multi-extractor presented in Theorem 19.24, the running time of the multi-extractor blows up exponentially.

As promised, here is the example that illustrates the issue. Consider an adversary that submits a sequence of ciphertexts to its decryption oracle in the CCA attack game for $\mathcal{E}_{\mathrm{aMEG}}$:

$$(v_1^*, e_1^*, \pi_1^*), \ldots, (v_n^*, e_n^*, \pi_n^*).$$

Each ciphertext encrypts a message m_k^*, so that $v_k^* = g^{\beta_k^*}$ and $e_k^* = u^{\beta_k^*} m_k^*$. Each proof π_k^* is of the form

$$(v_{\mathrm{t}k}^*, \beta_{\mathrm{z}k}^*) = (g^{\beta_{\mathrm{t}k}^*}, \beta_{\mathrm{t}k}^* + c_k \beta_k^*),$$

where the corresponding random oracle query is $((u, v_k^*, e_k^*), v_{\mathrm{t}k}^*)$, which defines the challenge $c_k = H((u, v_k^*, e_k^*), v_{\mathrm{t}k}^*)$. The adversary will prepare these ciphertexts ahead of time, *in reverse order*, so that the kth ciphertext depends on the challenges $c_{k+1}, \ldots, c_n$. In particular, the adversary will generate the kth ciphertext using the usual encryption algorithm, except that m_k^*, β_k^*, and $\beta_{\mathrm{t}k}^*$ are computed as a cryptographic hash of $c_{k+1}, \ldots, c_n$ and a secret key known only to the adversary. Fig. 19.11 shows a diagram illustrating the attack.

Now imagine how a reduction to semantic security would work based on rewinding techniques. When the adversary makes his first decryption query (v_1^*, e_1^*, π_1^*), we could rewind the adversary back to the point where he obtained the challenge c_1, feed the adversary a fresh random challenge c_1', and run him forward and hope that he submits an appropriate ciphertext that will allow us to extract β_1^* (via special soundness) and hence $m_1^* = e_1^*/u^{\beta_1^*}$. In general, we would have to be rather lucky for this to happen, but in this example we will succeed (with overwhelming probability) on the first try.

Now we run the adversary forward until he makes his second decryption query (v_2^*, e_2^*, π_2^*). As above, we rewind the adversary to the point where he obtained the challenge c_2 and run him forward with fresh random challenges c_2', c_1'. Unfortunately, the challenge c_2 was obtained *before* the adversary made his first decryption query, so when we run him forward again, he will make

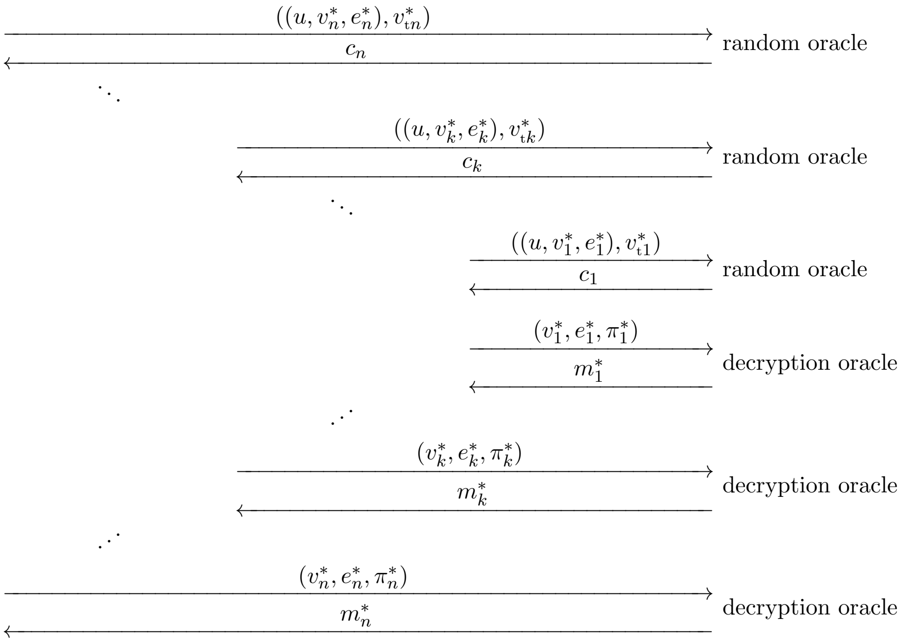

$((u, v_n^*, e_n^*), v_{tn}^*)$
c_n
random oracle
$((u, v_k^*, e_k^*), v_{tk}^*)$
c_k
random oracle
$((u, v_1^*, e_1^*), v_{t1}^*)$
c_1
random oracle
(v_1^*, e_1^*, π_1^*)
m_1^*
decryption oracle
(v_k^*, e_k^*, π_k^*)
m_k^*
decryption oracle
(v_n^*, e_n^*, π_n^*)
m_n^*
decryption oracle

Figure 19.11: Attack on $\mathcal{E}_{\text{aMEG}}$

another "first" decryption query for a ciphertext $(v_1^{**}, e_1^{**}, \pi_1^{**})$ before he ever gets to the second decryption query; therefore, we must first somehow extract the plaintext from $(v_1^{**}, e_1^{**}, \pi_1^{**})$. Moreover, $(v_1^{**}, e_1^{**}, \pi_1^{**})$ was computed by the adversary as a function of $c_2', c_3, \ldots, c_n$, and so will not be related at all to (v_1^*, e_1^*, π_1^*), which was computed as a function of $c_2, c_3, \ldots, c_n$. Given that our only tool is rewinding, we will have to recursively rewind the adversary just to extract the plaintext from $(v_1^{**}, e_1^{**}, \pi_1^{**})$, which will allow us to then extract the plaintext from (v_2^*, e_2^*, π_2^*).

It is not too hard to see that the above rewinding strategy takes time exponential in n. Indeed, let us just count the number of times we run the special-soundness witness extractor. Let $f(k)$ be the number of times we runs the extractor in order to respond to the first k decryption queries. Then we have

$$f(0) = 0 \ \text{ and } \ f(k) = 2f(k-1) + 1 \ \text{ for } k \geq 1,$$

since to respond to the kth decryption query, we first have to respond to the first $k-1$ decryption queries on the main execution path, and then we have to respond to the first $k-1$ decryption queries on a second, completely independent execution path, and then we get two accepting conversations from which we extract the witness for the kth decryption query using the special-soundness extractor. It follows that $f(n) = 2^n - 1$.

The above is by no means a proof that $\mathcal{E}_{\text{aMEG}}$ is not secure against a CCA attack. Indeed, there is no known chosen ciphertext attack against $\mathcal{E}_{\text{aMEG}}$, and in fact, one *can* prove that $\mathcal{E}_{\text{aMEG}}$ is CCA secure if we model H as a random oracle and model $\mathbb{G}$ as a "generic group" (see Section 16.3). However, it remains an open question as to whether $\mathcal{E}_{\text{aMEG}}$ is CCA secure modeling H as a random oracle, under some concrete security assumption for $\mathbb{G}$.

We saw how the rewinding technique can blow-up exponentially if we use it to try to prove that the "encrypt then prove" construction is CCA secure. The same type of exponential blow-up can occur if we try to analyze the security of a system in which we have many concurrent instances of a protocol that uses such a "proof of knowledge". As we try to rewind one instance of the protocol to extract a witness, this can lead to a cascade of rewinding just as we saw above.

Another subtle issue related to rewinding techniques in particular, and "proofs of knowledge" more generally, is that of determining who *really* knows something. For example, suppose we have an adversary $\mathcal{A}$ who is interacting with some honest party $\mathcal{Z}$. If $\mathcal{A}$ has a "proof of knowledge" π for a witness x for some statement y, it may be the case that $\mathcal{A}$ simply got that proof from $\mathcal{Z}$; therefore, while $\mathcal{Z}$ may well know x, adversary $\mathcal{A}$ itself may not have any idea of what x is. We already saw an example of this in the proof of Theorem 19.25. There, the party $\mathcal{Z}$ corresponds to the encryption oracle in the parallel 1CCA attack game. The proof of that theorem still worked because

(a) in the parallel 1CCA attack game, we were not required to extract a witness for the statement produced by the encryption oracle and so did not have to rewind the encryption oracle itself, which would have invalidated our security proof of that theorem, and

(b) although we may have had to rewind the adversary back to a point *before* it queried the encryption oracle, in those rewindings we effectively re-ran the encryption oracle in its entirety using fresh randomness (which is the reason that in our definition of multi-extractability, we explicitly modeled *when* randomness was generated and did not assume that it was all generated at the beginning).

It was precisely these types of subtleties that introduced some of the complexity in Definition 19.9.

19.10 A fun application: a two round witness independent protocol

To be written.

19.11 Notes

Citations to the literature to be added.

19.12 Exercises

19.1 (Bad randomness attack on Schnorr signatures). Let (sk, pk) be a key pair for the Schnorr signature scheme (Section 19.2). Suppose the signing algorithm is faulty and chooses *dependent* values for α_{t} in consecutively issued signatures. In particular, when signing a message m_0 the signing algorithm chooses a uniformly random $\alpha_{\text{t}0}$ in $\mathbb{Z}_q$, as required. However, when signing m_1 it choose $\alpha_{\text{t}1}$ as $\alpha_{\text{t}1} \leftarrow a \cdot \alpha_{\text{t}0} + b$ for some known $a, b \in \mathbb{Z}_q$. Show that if the adversary obtains the corresponding Schnorr message-signature pairs (m_0, σ_0) and (m_1, σ_1) and knows a, b and pk, it can learn the secret signing key sk, with high probability.

Discussion: This attack illustrates why it is important to derandomize signature schemes derived from Sigma protocols using the method of Exercise 13.6. It ensures that the signer is not dependent on the security of its entropy source.

19.2 (Batch Schnorr verification). Consider the unoptimized Schnorr signature scheme $\mathcal{S}_{\text{sch}}$ (Section 19.2). Let $\{(m_i, \sigma_i)\}_{i=1}^n$ be n message-signature pairs, signed relative to a public key u. In this exercise we show that verifying these n signatures as a batch may be faster than verifying them one by one. Recall that a signature $\sigma = (u_{\text{t}i}, \alpha_{\text{z}i})$ on message m_i is valid if $g^{\alpha_{\text{z}i}} = u^{c_i} \cdot u_{\text{t}i}$, where $c_i = H(m_i, u_{\text{t}i})$. To batch verify n signatures, the verifier does:

1. Choose random $\beta_1, \ldots, \beta_n \xleftarrow{\text{R}} \mathcal{C}$,
2. Compute $\bar{\alpha} \leftarrow \sum_{i=1}^n \beta_i \alpha_{\text{z}i} \in \mathbb{Z}_q$ and $\bar{c} \leftarrow \sum_{i=1}^n \beta_i c_i \in \mathbb{Z}_q$,
3. Accept all n signatures if $g^{\bar{\alpha}} = u^{\bar{c}} \cdot \prod_{i=1}^n u_{\text{t}i}^{\beta_i}$.

(a) Show that if one of the n signatures is invalid, then the entire batch will be rejected with probability at least $1/|\mathcal{C}|$.

Discussion: The bulk of the work is in line (3), which can be faster than verifying the signatures one by one, at least for a verifier with bounded memory. A verifier with sufficient memory can speed up single signature verification by pre-computing exponentiation tables for g and u, as discussed in Appendix A. In this case, batching may not save much time.

(b) Generalize the algorithm and show how to batch verify n triples $\{(u_i, m_i, \sigma_i)\}_{i=1}^n$ where u_i is a public key, and each (m_i, σ_i) is a message-signature pair with respect to u_i.

19.3 (Tight reduction for multi-key Schnorr signatures). In Exercise 13.2, you were asked to show that if a signature scheme is secure, it is also secure in the multi-key setting. However, the security bound degrades by a factor proportional to the number of keys.

Suppose that we modify Schnorr's signature scheme (Section 19.2) slightly, so that instead of computing the challenge as $c \leftarrow H(m, u_{\text{t}})$, we compute it as $c \leftarrow H(pk, m, u_{\text{t}})$. That is, we include the public key in the hash. Consider the security of this modified signature scheme in the multi-key setting, modeling H as a random oracle. Show that the bound (19.10) holds in the multi-key setting, independent of the number of keys, but assuming that all keys are generated using the same group $\mathbb{G}$. In this setting, the term Q_{s} in (19.10) represents the total number of signing queries performed under all the keys.

Hint: Use the random self-reducibility property of the DL problem (see Section 10.5.1).

19.4 (Schnorr signatures with related public keys). Let $u := g^\alpha \in \mathbb{G}$ be a random Schnorr public key, generated as in Section 19.2. Define n related Schnorr public keys $pk_1, \ldots, pk_n$ by setting $pk_i := u \cdot g^i \in \mathbb{G}$ for $i = 1, \ldots, n$. Consider the Schnorr signature scheme from Section 19.2, modified as in the previous exercise to include the public key as an input to the hash function H.

(a) Show that this scheme is multi-key secure with respect to the n public keys $pk_1, \ldots, pk_n$. That is, consider an adversary that is given u and can issue signing queries to all n public keys. Each query is a pair $(pk^{(j)}, m_j)$, for $j = 1, \ldots, Q$, and the response is a signature on m_j with respect to $pk^{(j)} \in \{pk_1, \ldots, pk_n\}$. Show that the adversary cannot produce, with non-negligible probability, an existential forgery (pk, m, σ) where $pk \in \{pk_1, \ldots, pk_n\}$ and

(pk, m) is not one of the signing queries. Security is based on discrete-log in $\mathbb{G}$, where the hash function used in the scheme is modeled as a random oracle.

(b) Consider again the unmodified Schnorr signature scheme from Section 19.2, where the public key is not included as part of the input to H. Show that this scheme is insecure with respect to the related public keys $pk_1, \ldots, pk_n$. In particular, a signature on message m with respect to pk_j lets the adversary construct a signature on m with respect to pk_i for some $i \neq j$.

(c) Suppose that the n related public keys are generated as $pk_i := u^i \in \mathbb{G}$ for $i = 1, \ldots, n$. Show that the Schnorr signature scheme from Section 19.2 is multi-key secure with respect to these public keys $pk_1, \ldots, pk_n$, even if we do not include the public key as part of the input to H.

Discussion: A variant of the result in part (a) is widely used in crypto-currencies such as Bitcoin, where a public-facing server generates many public keys from $u = g^\alpha$, without knowledge of the secret key α (this is called *hierarchical deterministic* (HD) key creation). The secret key α can be stored in a vault and later used to compute the secret keys for the public keys generated by the public-facing server.

19.5 (Rewinding lemma variations). This exercise looks as some variants of Lemma 19.2. Let T and U be finite, non-empty sets, and let $f : T \times U \to \{0, 1\}$ be a function.

(a) Let Y, Z, Y', and Z' be mutually independent random variables. Assume that Y and Y' are each uniformly distributed over T and that Z and Z' take values in the set U and have the same distribution. Let $\epsilon := \Pr[f(\mathsf{Y}, \mathsf{Z}) = 1]$ and $N := |T|$. Show that

$$\Pr\big[f(\mathsf{Y}, \mathsf{Z}) = 1 \wedge f(\mathsf{Y}', \mathsf{Z}') = 1 \wedge \mathsf{Y} \neq \mathsf{Y}'\big] \geq \epsilon^2 - \epsilon/N.$$

(b) Generalize part (a) as follows. Let $\mathsf{Y}_1, \ldots, \mathsf{Y}_n$ and $\mathsf{Z}_1, \ldots, \mathsf{Z}_n$ be mutually independent random variables, where each Y_i has the same distribution as Y, and each Z_i has the same distribution as Z. For $i = 1, \ldots, n$, define

$$\epsilon_i = \Pr\big[\ f(\mathsf{Y}_1, \mathsf{Z}_1) = \cdots = f(\mathsf{Y}_i, \mathsf{Z}_i) = 1 \wedge \mathsf{Y}_1, \ldots, \mathsf{Y}_i \text{ are pairwise distinct }\big].$$

Observe that $\epsilon_1 = \epsilon$. Prove that for $i = 2, \ldots, n$, we have

$$\epsilon_i \geq \epsilon_{i-1}(\epsilon - (i-1)/N).$$

19.6 (Enlarging the challenge space). Many applications of Sigma protocols require a large challenge space. This exercise shows that we can always take a Sigma protocol with a small challenge space and turn it into one with a large challenge space, essentially by parallel repetition.

Let (P, V) be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, with challenge space $\mathcal{C}$. Let k be a positive integer. Define a new Sigma protocol (P^k, V^k) as follows. Here, the prover P^k takes as input $(x, y) \in \mathcal{R}$, the verifier V^k takes as input $y \in \mathcal{Y}$, and the challenge space is $\mathcal{C}^k$.

- P^k initializes k instances of P on input (x, y), obtaining commitments $t_1, \ldots, t_k$, and sends these to V^k.
- V^k chooses $(c_1, \ldots, c_k) \in \mathcal{C}^k$ at random, and sends this to P^k.
- For $i = 1, \ldots, k$, the prover P^k feeds c_i into the ith instance of P, obtaining a response z_i. It then sends $(z_1, \ldots, z_k)$ to V^k.

- For $i = 1, \ldots, k$, the verifier V^k verifies' that (t_i, c_i, z_i) is an accepting conversation for y.

(a) Show that (P^k, V^k) is Sigma protocol for $\mathcal{R}$.

(b) Show that if (P, V) provides special soundness, then so does (P^k, V^k).

(c) Show that if (P, V) is special HVZK, then so is (P^k, V^k).

Discussion: For example, if we want to use the GQ protocol (see Section 19.5.5) to prove knowledge of an eth root of y modulo n, where e is small (say $e = 3$), then we can use this technique to increase the size of the challenge space to 3^k, which is essential to get a secure ID scheme. Of course, this blows up the complexity of the protocol by a factor of k, which is unfortunate. See Exercise 19.10 below that shows that some simple ideas to increase the challenge space more efficiently do not work. See also Exercise 19.29 for a more efficient scheme in an "amortized" setting.

19.7 (A soundness bound on Sigma protocols). Let (P, V) be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, with challenge space $\mathcal{C}$. Suppose that (P, V) is special HVZK. Show that a dishonest prover $\hat{P}$ that is initialized with a statement $y \in \mathcal{Y}$ (but is not given the corresponding witness $x \in \mathcal{X}$) can succeed in getting the verifier to accept with probability $1/|\mathcal{C}|$. This is why Sigma protocols must use a challenge space $\mathcal{C}$ where $|\mathcal{C}|$ is super-poly.

19.8 (The Schnorr protocol in composite order groups). In this exercise we explore the security of the Schnorr protocol in groups whose order is not a prime. Let $\mathbb{G}$ be a cyclic group of order $n = \ell q$ where ℓ is poly-bounded and q is super-poly prime (for example take $\ell = 2$). Let $g \in \mathbb{G}$ be a generator. The prover has a secret key $\alpha \in \mathbb{Z}_n$ and the corresponding verification key $u := g^\alpha \in \mathbb{G}$.

(a) Show that if the challenge space $\mathcal{C}$ in Schnorr's protocol is $\mathcal{C}_q := \{0, \ldots, q - 1\}$ then the protocol provides special soundness and is special HVZK.

(b) Suppose we use a larger challenge space $\mathcal{C}_B := \{0, \ldots, B\}$ for some $B \geq q$. Show that a prover that is only given $u = g^\alpha \in \mathbb{G}$ (but is not given α) can fool the verifier with probability $1/q$. Hence, the enlarged challenge space does not reduce the probability that a dishonest prover succeeds in convincing the verifier.

Discussion: One can show that when $B \geq q$ the Schnorr protocol with challenge space $\mathcal{C}_B$ does not have special soundness, assuming discrete-log in $\mathbb{G}$ is hard.

(c) Let's go back to a challenge space $\mathcal{C}_q := \{0, \ldots, q - 1\}$ and let $\hat{g} := g^\ell \in \mathbb{G}$. This $\hat{g}$ generates a strict subgroup of $\mathbb{G}$ of order q. Let $u = g^\alpha \in \mathbb{G}$ where α is relatively prime to ℓ. This u is not in the subgroup generated by $\hat{g}$ and therefore not a power of $\hat{g}$. Now consider the Schnorr identification protocol defined base $\hat{g}$ between a prover and verifier: the prover is given $\hat{g}$ and α and the verifier is given $\hat{g}$ and u. The Schnorr protocol is designed to convince the verifier that the prover knows the discrete-log of u base $\hat{g}$, and hence the verifier should reject because the discrete-log does not exist. However, show that the prover can fool the verifier with probability about $1/\ell$. In particular, the prover can cause the verifier to accept whenever the challenge c is divisible by ℓ.

Discussion: This example shows that if the base $\hat{g} \in \mathbb{G}$ is not a generator of $\mathbb{G}$, then the Schnorr protocol can lose soundness altogether if the order of $\mathbb{G}$ has small prime factors. To prevent this attack the verifier must explicitely check that u is in the group generated by $\hat{g}$.

19.9 (Idealized ECDSA signatures). In this exercise we analyze an idealized version of the ECDSA signature scheme from Section 19.3. Let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$. The secret key is $sk := \alpha$ where $\alpha \xleftarrow{\text{R}} \mathbb{Z}_q$. The public key is $pk := u \in \mathbb{G}$ where $u \leftarrow g^{\alpha}$. Let $H : \mathcal{M} \to \mathbb{Z}_q$ and $H' : \mathbb{G} \to \mathbb{Z}_q^*$ be hash functions.

- To sign a message $m \in \mathcal{M}$ using a secret key $sk = \alpha$, the signing algorithm does:

$$S(sk, m) := \left\{ \begin{array}{l} \text{repeat until } s \neq 0\text{:} \\ \qquad \alpha_{\text{t}} \xleftarrow{\text{R}} \mathbb{Z}_q^*, \quad u_{\text{t}} \leftarrow g^{\alpha_{\text{t}}}, \quad r \leftarrow H'(u_{\text{t}}), \quad s \leftarrow \big(H(m) + r\alpha\big)/\alpha_{\text{t}} \in \mathbb{Z}_q \\ \text{output } \sigma := (r, s). \end{array} \right\}.$$

- To verify a signature $\sigma = (r, s)$ on a message $m \in \mathcal{M}$, using the public key $pk = u$, the signature verification algorithm outputs accept only if $H'\big(g^{H(m)/s} \cdot u^{r/s}\big) = r$.

Prove that this signature scheme is secure assuming discrete log in $\mathbb{G}$ is hard, H is collision resistant, and H' is modeled as a random oracle.

Discussion: This scheme is identical to the actual ECDSA signature scheme from Section 19.3, except that here, we are computing $r \leftarrow H'(u_{\text{t}})$, instead of computing r as the x-coordinate of u_{t}, reduced modulo q.

19.10 (GQ security). This exercise explains why the challenge space in the GQ protocol (see Section 19.5.5) is restricted to a subset of $\{0, \ldots, e-1\}$.

(a) Suppose we set the challenge space in the GQ protocol to $\mathcal{C} := \{0, \ldots, b \cdot e\}$ for some integer $b > 1$. Show that a prover that is only given $y = x^e \in \mathbb{Z}_n^*$ (but is not given x) can fool the verifier with probability $1/e$. Hence, the enlarged challenge space does not reduce the probability that a dishonest prover succeeds in convincing the verifier.

(b) Suppose we set the challenge space in the GQ protocol to $\mathcal{C} := \{0, \ldots, e\}$. Show that the protocol no longer has special soundness. To do so, show that an efficient witness extractor *Ext* would give an efficient algorithm to compute an eth root of y in $\mathbb{Z}_n^*$. This would violate the RSA assumption.

19.11 (Okamato's RSA-based Sigma protocol). Okamoto's protocol (see Section 19.5.1) is based on the discrete logarithm problem. There is a variant of Okamoto's protocol that is based on the RSA problem. By way of analogy, Okamoto's DL-based protocol was a "proof of knowledge" of a preimage of the hash function H_{dl} in Section 10.6.1, and Okamato's RSA-based protocol is a "proof of knowledge" of a preimage of the hash function H_{rsa} in Section 10.6.2.

The setup is as follows. Let (n, e) be an RSA public key, where the encryption exponent e is a prime number. Also, let y be a random number in $\mathbb{Z}_n^*$. We shall view the values n, e, and y as systems parameters. Let $I_e := \{0, \ldots, e-1\}$.

The relation of interest is the following:

$$\mathcal{R} = \{ \ (\ (a, b), \ u\ ) \in (\mathbb{Z}_n^* \times I_e) \times \mathbb{Z}_n^* \ : \ u = a^e y^b \ \}.$$

The protocol (P, V) runs as follows, where the prover is initialized with $((a, b), u) \in \mathcal{R}$ and the verifier V is initialized with $u \in \mathbb{Z}_n^*$, and the challenge space $\mathcal{C}$ is a subset of I_e:

- P computes
$$a_{\mathrm{t}} \xleftarrow{\mathrm{R}} \mathbb{Z}_n^*, \;\; b_{\mathrm{t}} \xleftarrow{\mathrm{R}} I_e, \;\; u_{\mathrm{t}} \leftarrow a_{\mathrm{t}}^e y^{b_{\mathrm{t}}},$$
and sends the commitment u_{t} to V;

- V computes $c \xleftarrow{\mathrm{R}} \mathcal{C}$, and sends the challenge c to P;

- P computes
$$b' \leftarrow b_{\mathrm{t}} + cb, \;\; a_{\mathrm{z}} \leftarrow a_{\mathrm{t}} \cdot a^c \cdot y^{\lfloor b'/e \rfloor}, \;\; b_{\mathrm{z}} \leftarrow b' \bmod e,$$
and sends the response $(a_{\mathrm{z}}, b_{\mathrm{z}})$ to V;

- V checks if $a_{\mathrm{z}}^e y^{b_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u^c$; if so V outputs accept; otherwise, V outputs reject.

Prove that this protocol is a Sigma protocol for the relation $\mathcal{R}$ defined above, and that it provides special soundness and is special HVZK.

19.12 (An insecure variant of Fiat-Shamir signatures). Consider the signature system derived from a Sigma protocol (P, V) as in Section 19.6.1. Assume (P, V) is special HVZK. Suppose that during signing we set the challenge as $c \leftarrow H(m)$ instead of $c \leftarrow H(m, t)$. Show that the resulting signature system is insecure.

Hint: Use the HVZK simulator to forge the signature on any message of your choice.

19.13 (Computational special soundness). Let $\Pi = (P, V)$ be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, with challenge space $\mathcal{C}$. Recall that our definition of special soundness (see Definition 19.4) says that there is an efficient witness extractor algorithm *Ext* that on input $y \in \mathcal{Y}$, along with any two accepting conversations (t, c, z) and (t, c', z') with $c \neq c'$, outputs a witness x for y. We can relax this definition, by insisting only that it is computationally infeasible to find inputs to the witness extraction algorithm of the required form on which the algorithm fails to output a witness.

More precisely, for a given adversary $\mathcal{A}$, we define $\mathrm{cSS}\mathsf{adv}[\mathcal{A}, \Pi, \mathit{Ext}]$ to be the probability that $\mathcal{A}$ outputs two accepting conversations (t, c, z) and (t, c', z') with $c \neq c'$, such that $\mathit{Ext}(y, (t, c, z), (t, c', z'))$ is not a witness for y. We say Π provides **computational special soundness** if there exists an efficient witness extractor *Ext* for Π, such that for every efficient adversary $\mathcal{A}$, the value $\mathrm{cSS}\mathsf{adv}[\mathcal{A}, \Pi, \mathit{Ext}]$ is negligible.

Consider the Fiat-Shamir signature construction in Section 19.6.1 built from a Sigma protocol (P, V) and a key generation algorithm G. Assume that G is one way. Also assume that (P, V) that is special HVZK, provides special soundness, has unpredictable commitments and a large challenge space. We showed that the resulting signature scheme is secure (in the random oracle model), satisfying the bound in (19.21). Show that if (P, V) only provides computational special soundness (instead of special soundness), then the resulting signature scheme is still secure (in the random oracle model), and show how the bound (19.21) needs to be updated.

19.14 (Unique responses). Let Π be a Sigma protocol. We say that Π has **unique responses** if for every statement y and for every pair of accepting conversations (t, c, z) and (t, c, z') for y, we must have $z = z'$.

(a) Prove that Schnorr's Sigma protocol has unique responses.

(b) Prove that the Chaum-Pedersen protocol (see Section 19.5.2) has unique responses.

(c) Consider the generic linear protocol in Section 19.5.3. A particular instance of this protocol is defined in terms of a class $\mathcal{F}$ of formulas ϕ of the form (19.13). For such a formula ϕ, we can consider its *homogenized form* ϕ', which is obtained by replacing each u_i in (19.13) by the group identity 1. Prove that the generic linear protocol for formulas in $\mathcal{F}$ has unique responses if the following holds: for every $\phi \in \mathcal{F}$, its homogenized form ϕ' has a unique solution (namely, $\alpha_j = 1$ for $j = 1, \ldots, n$).

(d) Prove that the GQ protocol (see Section 19.5.5) has unique responses.

19.15 (Strong special soundness). Let Π be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Recall that our definition of special soundness (see Definition 19.4) says that there is an efficient witness extractor algorithm *Ext* that on input $y \in \mathcal{Y}$, along with any two accepting conversations (t, c, z) and (t, c', z') with $c \neq c'$, outputs a witness x for y. We can strengthen the requirement by insisting that *Ext* should output a witness for y assuming only that $(c, z) \neq (c', z')$, rather than $c \neq c'$. We say that Π provides **strong special soundness** if there exists an efficient witness extractor with this property.

(a) Prove that if Π provides special soundness and has unique responses (see previous exercise), then it provides strong special soundness.

(b) Consider the OR-proof construction in Section 19.7.2, which combines two Sigma protocols (P_0, V_0) and (P_1, V_1) into a new Sigma protocol (P, V) for the relation $\mathcal{R}_{\text{OR}}$ in (19.23). Prove that if (P_0, V_0) and (P_1, V_1) provide strong special soundness, then so does (P, V).

19.16 (Computational strong special soundness). We can relax the notion of strong special soundness, which was introduced in the previous exercise, by insisting only that it is computationally infeasible to find inputs to the witness extraction algorithm of the required form on which the algorithm fails to output a witness.

More precisely, for a given adversary $\mathcal{A}$, we define cSSS$\mathsf{adv}[\mathcal{A}, \Pi, \mathit{Ext}]$ to be the probability that $\mathcal{A}$ outputs two accepting conversations (t, c, z) and (t, c', z') with $(c, z) \neq (c', z')$, such that $\mathit{Ext}(y, (t, c, z), (t, c', z'))$ is not a witness for y. We say Π provides **computational strong special soundness** if there exists an efficient witness extractor *Ext* for Π, such that for every efficient adversary $\mathcal{A}$, the value cSSS$\mathsf{adv}[\mathcal{A}, \Pi, \mathit{Ext}]$ is negligible.

(a) Prove that computational strong special soundness implies computational special soundness (as defined in Exercise 19.13).

(b) Prove that Okamoto's protocol (see Section 19.5.1) provides computational strong special soundness, under the DL assumption. Here, we are assuming that the system parameter $h \in \mathbb{G}$ used by Okamoto's protocol is uniformly distributed over $\mathbb{G}$. You should show that an adversary that can find two accepting conversations for some statement with different responses, but with the same commitment and challenge, can compute $\mathsf{Dlog}_g h$.

(c) Prove that Okamoto's RSA-based protocol (see Exercise 19.11) provides computational strong special soundness, under the RSA assumption. You should show that an adversary that can find two accepting conversations for some statement with different responses, but with the same commitment and challenge, can compute $y^{1/e} \in \mathbb{Z}_n^*$.

(d) Generalize part (b) of the previous exercise, showing that if (P_0, V_0) and (P_1, V_1) provide computational strong special soundness, then so does (P, V).

19.17 (Strongly secure signature schemes). Consider the Fiat-Shamir signature construction in Section 19.6.1 built from a Sigma protocol (P, V) and a key generation algorithm G. Assume that (P, V) that is special HVZK, has unpredictable commitments, and a large challenge space. Also assume that G is one way.

(a) Prove that if (P, V) provides special soundness and has unique responses (see Exercise 19.14), then the resulting signature scheme is strongly secure (in the sense of Definition 13.3), modeling H as a random oracle. You should prove the same bound as in (19.21), but for $\mathrm{stSIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ instead of $\mathrm{SIG}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$.

(b) Prove that if (P, V) provides computational strong special soundness (see previous exercise), then the resulting signature scheme is strongly secure, again, modeling H as a random oracle. Derive a concrete security bound as a part of your analysis.

Discussion: As a consequence of part (a), we see that Schnorr and GQ are strongly secure signature schemes.

19.18 (Backward computable commitments). Most of the examples of Sigma protocols we have seen in this chapter have the following special structure: if the relation is $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, and if conversations (t, c, z) lie in the set $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$, then for every $(y, c, z) \in \mathcal{Y} \times \mathcal{C} \times \mathcal{Z}$, there exists a unique $t \in \mathcal{T}$ such that (t, c, z) is an accepting conversation for y; moreover, the function f mapping (y, c, z) to t is efficiently computable. Let us say that (P, V) has **backward computable commitments** in this case. (In fact, all of the special HVZK simulators essentially work by choosing z at random and computing $t = f(y, c, z)$. Note that the range proof protocol in Section 20.4.1 is an example of a Sigma protocol that does not have backward computable commitments.)

(a) Verify that the generic linear protocol (see Section 19.5.3) and the GQ protocol (see Section 19.5.5) have backward computable commitments.

(b) Show that if (P_0, V_0) and (P_1, V_1) have backward computable commitments, then so do the AND-proof and OR-proof constructions derived from (P_0, V_0) and (P_1, V_1) (see Section 19.7).

19.19 (Optimized Fiat-Shamir signatures). The optimization we made for Schnorr and GQ signatures can be applied to Fiat-Shamir signatures derived from most Sigma protocols. Consider the Fiat-Shamir signature scheme derived from a Sigma protocol (P, V) for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, and a key generation algorithm G for $\mathcal{R}$. Recall that a Fiat-Shamir signature on a message m is of the form (t, z), where $(t, c, z) \in \mathcal{T} \times \mathcal{C} \times \mathcal{Z}$ is an accepting conversation, and $c := H(m, t)$.

Assume that (P, V) has backward computable commitments, as in the previous exercise, and let $f : \mathcal{Y} \times \mathcal{C} \times \mathcal{Z} \to \mathcal{T}$ be the corresponding function that computes a commitment from a given statement, challenge, and response. Then we can optimize the Fiat-Shamir signature scheme, so that instead of using (t, z) as the signature, we use (c, z) as the signature. To verify such an optimized signature (c, z), we compute $t \leftarrow f(c, z)$, and verify that $c = H(m, t)$. Note that c is usually much smaller than t, so these optimized signatures are usually much more compact.

(a) Show that if the Fiat-Shamir signature scheme is secure, then so is the optimized Fiat-Shamir signature scheme.

(b) Show that if the Fiat-Shamir signature scheme is *strongly* secure, then so is the optimized Fiat-Shamir signature scheme.

Note: For both parts, you should show that any adversary that breaks the optimized scheme can be converted to one that is just as efficient, and breaks the unoptimized scheme with the same advantage.

19.20 (Collision resistance from Sigma protocols). Suppose (P, V) is a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Furthermore, assume that (P, V) has backward computable commitments, as in Exercise 19.18, where $f : \mathcal{Y} \times \mathcal{C} \times \mathcal{Z} \to \mathcal{T}$ is the corresponding function that computes a commitment from a given statement, challenge, and response. Also assume that (P, V) provides computational strong special soundness, as in Exercise 19.16. Finally, let G be a one-way key generation algorithm for $\mathcal{R}$.

From these components, we can build a hash function $H : \mathcal{C} \times \mathcal{Z} \to \mathcal{T}$, as follows. The hash function makes use of a system parameter $y \in \mathcal{Y}$, which is obtained by running $(y, (x, y)) \xleftarrow{R} G()$. For $(c, z) \in \mathcal{C} \times \mathcal{Z}$, and a given system parameter $y \in \mathcal{Y}$, we define $H(c, z) := f(y, c, z) \in \mathcal{T}$.

Prove that H is collision resistant.

Discussion: The hash function H_{dl} in Section 10.6.1 can be viewed as a special case of this result, applied to Schnorr's protocol. The hash function H_{rsa} in Section 10.6.2 can be viewed as a special case of this result, applied to the GQ protocol.

19.21 (Strongly secure one-time signatures from Sigma protocols (I)). Suppose (P, V) is a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ and that (P, V) has conversations in $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$. Also assume that (P, V) provides computational strong special soundness (as in Exercise 19.16) and is special HVZK. Let G be a one-way key generation algorithm for $\mathcal{R}$. Finally, let $H : \mathcal{M} \to \mathcal{C}$ be a hash function.

We can define a signature scheme (G^*, S^*, V^*) with message space $\mathcal{M}$ as folows.

- G^* computes $(y, (x, y)) \xleftarrow{R} G()$, and then initializes a prover instance $P(x, y)$, obtaining a commitment $t \in \mathcal{T}$. It outputs the public key $pk^* := (y, t)$. The secret key sk^* is the internal state of the prover instance $P(x, y)$.
- Given a secret key sk^* as above, and a message $m \in \mathcal{M}$, the signing algorithm S^* feeds $c \leftarrow H(m)$ to the prover instance $P(x, y)$, obtaining a response $z \in \mathcal{Z}$. The signature is z.
- Given a public key $pk^* = (y, t) \in \mathcal{Y} \times \mathcal{T}$, a message $m \in \mathcal{M}$, and a signature $z \in \mathcal{Z}$, the verification algorithm computes $c \leftarrow H(m)$ and checks that (t, c, z) is an accepting conversation for y.

Prove that (G^*, P^*, V^*) is a strongly secure one-time signature scheme (see Definition 14.2) if we model H as a random oracle.

Discussion: If we instantiate this with Schnorr's protocol, we get the signature scheme discussed in Exercise 14.11.

19.22 (Type hiding key generation). In Section 19.8, we introduced the notion of witness independence, and we saw that this property (which is implied by special HVZK) could be used to design actively secure identification protocols. This exercise generalizes these results, establishing more general conditions under which a Sigma-protocol based ID scheme can be proved actively secure using WI. Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Let G be a key generation algorithm for $\mathcal{R}$.

Suppose that $\mathsf{type} : \mathcal{X} \to \mathcal{T}$ is a function from $\mathcal{X}$ into some finite set $\mathcal{T}$, where $|\mathcal{T}| > 1$. We say that G is **second-preimage resistant** (relative to the function type) if it is hard for any efficient adversary $\mathcal{A}$ to win the following game:

- challenger computes $(y, (x, y)) \xleftarrow{R} G()$ and sends (x, y) to $\mathcal{A}$;
- $\mathcal{A}$ wins the game if he can compute $\hat{x} \in \mathcal{X}$ such that $(\hat{x}, y) \in \mathcal{R}$ and $\mathsf{type}(\hat{x}) \neq \mathsf{type}(x)$.

We also need an information-theoretic notion that says that G generates public keys that do not leak any information about the type of the secret key. Let $\mathbf{X}$ and $\mathbf{Y}$ be random variables representing the witness and statement output by G. We say G is **type hiding** if for all $(\hat{x}, y) \in \mathcal{R}$, we have

$$\Pr[\mathsf{type}(\mathbf{X}) = \mathsf{type}(\hat{x}) \mid \mathbf{Y} = y] = \frac{1}{|\mathcal{T}|}.$$

This is equivalent to saying that $\mathbf{Y}$ and $\mathsf{type}(\mathbf{X})$ are independent, with $\mathsf{type}(\mathbf{X})$ uniformly distributed over $\mathcal{T}$.

(a) Suppose (P, V) is a Sigma protocol for $\mathcal{R}$ that provides special soundness and is special HVZK, and has a large challenge space. Further, suppose that G is a key generation algorithm for $\mathcal{R}$ that is second-preimage resistant and type hiding for some type function. Prove that the identification protocol (G, P, V) is secure against active attacks.

(b) Show that key generation algorithm G' for the OR-proof-based Sigma protocol (P', V') in Section 19.8.3 is second-preimage resistant (under the assumption that underlying key generation algorithm G is one-way) and type hiding, using the type function $\mathsf{type}(b, x) := b \in \{0, 1\}$.

(c) Show that key generation algorithm G for Okamoto's protocol (P, V) in Section 19.8.4 is second-preimage resistant (under the DL assumption) and type hiding, using the type function $\mathsf{type}(\alpha, \beta) := \beta \in \mathbb{Z}_q$.

(d) Consider Okamoto's RSA-based Sigma protocol (P, V) in Exercise 19.11. Define the key generation G, which outputs the statement u and witness (a, b), where $a \xleftarrow{R} \mathbb{Z}_n^*$, $b \xleftarrow{R} I_e$, and $u \xleftarrow{R} a^y y^b$. Show that G is second-preimage resistant (under the RSA assumption) and type hiding, using the type function $\mathsf{type}(a, b) := b \in I_e$. Conclude that the identification scheme (G, P, V) is secure against active attacks, under the RSA assumption.

19.23 (Public-key equivalence). We can use the notion of witness independence to simplify certain schemes built from Sigma protocols.

Suppose (P, V) is Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Let G_0 and G_1 be two key generation algorithms for $\mathcal{R}$. We say that these two algorithms are **public-key equivalent** if the public keys generated by these two algorithms have the same distribution.

Consider the following attack game, which consists of two experiments. In Experiment b, where $b \in \{0, 1\}$, the challenger computes $(pk, sk) \xleftarrow{R} G_b()$, to obtain $pk = y$ and $sk = (x, y)$, and then interacts with an adversary $\mathcal{A}$ as in Experiment (x, y) of Attack Game 19.3, at the end of which the adversary outputs a bit $\hat{b} \in \{0, 1\}$.

Show that if (P, V) is witness independent and G_0 and G_1 are public-key equivalent, then the probability that $\mathcal{A}$ outputs 1 is the same in both experiments.

19.24 (Simplified identification protocols). We can use the result of the previous exercise to obtain somewhat simplified, and more efficient, identification protocols that are secure against active attacks.

(a) Suppose (P, V) is a witness independent Sigma protocol for a relation $\mathcal{R}$, G is a key generation for $\mathcal{R}$, and that the identification protocol (G, P, V) is secure against active attacks. Further, suppose that G_0 is a key generation algorithm that is public-key equivalent to G, as in the previous exercise. Show that the identification protocol (G_0, P, V) is just as secure against active attacks, in the sense that any impersonation adversary that breaks the security of (G_0, P, V) breaks (G, P, V) with the same advantage.

(b) Consider the OR-proof-based identification protocol (G', P', V') in Section 19.8.3. Argue that we can replace G' by G'_0, which always sets $b \leftarrow 0$, instead of $b \xleftarrow{\text{R}} \{0, 1\}$, and the resulting identification protocol (G'_0, P', V') is just as secure against active attacks.

(c) Consider Okamoto's identification protocol (G, P, V) in Section 19.8.4. Argue that we can replace G by G_0, which always sets $\beta \leftarrow 0$, instead of $\beta \xleftarrow{\text{R}} \mathbb{Z}_q$, and the resulting identification protocol (G_0, P, V) is just as secure against active attacks. Describe the resulting scheme in detail.

(d) Consider Okamoto's RSA-based identification protocol (G, P, V) in part (d) of Exercise 19.22. Argue that we can replace G by G_0, which always sets $b \leftarrow 0$, instead of $b \xleftarrow{\text{R}} I_e$, and the resulting identification protocol (G_0, P, V) is just as secure against active attacks. Describe the resulting scheme in detail.

19.25 (Strongly secure one-time signatures from Sigma protocols (II)). In Exercise 19.21, we saw how to build strongly secure one-time signature schemes from Sigma protocol using a random oracle. In this exercise, we do the same, without relying on random oracles; however, we require Sigma protocols with stronger properties.

Suppose (P, V) is a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, and that (P, V) has conversations in $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$. Let G_0 be a key generation algorithm for $\mathcal{R}$. We can define a signature scheme (G_0^*, S^*, V^*), with message space $\mathcal{C}$, as follows.

- G_0^* computes $(y, (x, y)) \xleftarrow{\text{R}} G_0()$, and then initializes a prover instance $P(x, y)$, obtaining a commitment $t \in \mathcal{T}$. It outputs the public key $pk^* := (y, t)$. The secret key sk^* is the internal state of the prover instance $P(x, y)$.

- Given a secret key sk^* as above, and a message $c \in \mathcal{C}$, the signing algorithm S^* feeds c to the prover instance $P(x, y)$, obtaining a response $z \in \mathcal{Z}$. The signature is z.

- Given a public key $pk^* = (y, t) \in \mathcal{Y} \times \mathcal{T}$, a message $c \in \mathcal{C}$, and a signature $z \in \mathcal{Z}$, the verification algorithm checks that (t, c, z) is an accepting conversation for y.

(a) Assume that (P, V) provides computational strong special soundness (see Exercise 19.16) and is special HVZK. Further, assume that G_0 is public-key equivalent (see Exercise 19.23) to a key generation algorithm G that is second-preimage resistant and type hiding for some type function (see Exercise 19.22). Prove that (G_0^*, P^*, V^*) is a strongly secure one-time signature scheme (see Definition 14.2).

(b) Describe in detail the signature schemes based on the Sigma protocols and key generation algorithms in parts (b)–(d) of the previous exercise, and argue that they are strongly secure one-time signature schemes.

Note: The scheme based on part (c) of the previous exercise is actually the same scheme that was presented in Exercise 14.12.

19.26 (Generalized AND-proofs and OR-proofs). Generalize the AND-proof and OR-proof constructions in Section 19.7 from two Sigma protocols to n protocols. You can view n as either a constant or a system parameter. If n is not constant, then it is perhaps simplest to assume that all the Sigma protocols are the same. State the relations for your new Sigma protocols, and argue that they provide special soundness and are special HVZK under appropriate assumptions. The computational and communication complexity of your protocols should scale linearly in n.

19.27 (Special HVZK with non-uniform challenges). Suppose (P, V) is a Sigma protocol for a relation $R \subseteq \mathcal{X} \times \mathcal{Y}$, with challenge space $\mathcal{C}$. Further, suppose (P, V) is special HVZK with simulator Sim. Now let D be an arbitrary probability distribution on $\mathcal{C}$. Consider a challenger V_D that generates its challenge according to the distribution D, rather than uniformly over $\mathcal{C}$. Show the following: *for all $(x, y) \in \mathcal{R}$, if we compute*

$$c \overset{\text{R}}{\leftarrow} D, \;\; (t, z) \overset{\text{R}}{\leftarrow} Sim(y, c),$$

then (t, c, z) has the same distribution as that of a transcript of a conversation between $P(x, y)$ and $V_D(y)$.

19.28 (Threshold proofs). The OR-proof construction in Section 19.7.2 allows a prover to convince a verifier that he knows a witness for one of two given statements. In this exercise, we develop a generalization that allows a prover to convince a verifier that he knows at least k witnesses for n given statements.

Let (P, V) be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Assume that (P, V) provides special soundness and is special HVZK, with simulator Sim. We also assume that $\mathcal{C} = \mathbb{Z}_q$ for some prime q. Let n and k be integers, with $0 < k < n < q$. We can think of n and k as being constants or system parameters.

We shall build a Sigma protocol (P', V') for the relation

$$\mathcal{R}' = \Big\{ \; \big(\, (x_1, \ldots, x_n), \; (y_1, \ldots, y_n) \, \big) \in (\mathcal{X} \cup \{\bot\})^n \times \mathcal{Y}^n : \\ \big|\{i \in \{1, \ldots, n\} : (x_i, y_i) \in \mathcal{R}\}\big| \geq k \; \Big\}.$$

Suppose the prover P' is given the witness $(x_1, \ldots, x_n)$ and the statement $(y_1, \ldots, y_n)$, and the verifier V' is given the statement $(y_1, \ldots, y_n)$. Let I denote the set of indices i such that $(x_i, y_i) \in \mathcal{R}$. We know that $|I| \geq k$. We shall assume that $|I| = k$, removing indices from I if necessary. Let $J := \{1, \ldots, n\} \setminus I$, so $|J| = n - k$. The protocol runs as follows.

1. For each $j \in J$, the prover chooses $c_j \in \mathbb{Z}_q$ at random, and runs Sim on input (y_j, c_j) to obtain (t_j, z_j). For each $i \in I$, the prover initializes an instance of P with (x_i, y_i), obtaining a commitment t_i. The prover then sends $(t_1, \ldots, t_n)$ to the verifier.

2. The verifier generates a challenge $c \in \mathbb{Z}_q$ at random, and sends c to the prover.

3. The prover computes the unique polynomial $f \in \mathbb{Z}_q[w]$ of degree at most $n - k$ such that $f(0) = c$ and $f(j) = c_j$ for all $j \in J$ using a polynomial interpolation algorithm. It then computes the challenges $c_i := f(i)$ for all $i \in I$. For each $i \in I$, the prover then feeds the challenge c_i to the instance of P it initialized with (x_i, y_i), obtaining a response z_i. The prover then sends $(f,\ z_1, \ldots, z_n)$ to the verifier.

4. First, the verifier checks that f is a polynomial of degree at most $n - k$ with constant term c. Then, for $\ell = 1, \ldots, n$, it computes $c_\ell := f(\ell)$. Finally, for $\ell = 1, \ldots, n$, it verifies that (t_ℓ, c_ℓ, z_ℓ) is an accepting conversation for y_ℓ.

Show that (P', V') is a Sigma protocol for $\mathcal{R}'$ that provides special soundness and is special HVZK.

Hint: The previous exercise may be helpful in arguing special HVZK.

Discussion: For simplicity, we presented the protocol for n identical relations $\mathcal{R}$. The protocol also works essentially "as is" even if the relations are not all the same.

19.29 (Amortized complexity of Sigma protocols). This exercise illustrates a technique that can be used to increase the challenge space size of a Sigma protocol without increasing its communication complexity, at least in an amortized sense. We illustrate the technique on the GQ protocol for proving knowledge of an eth root modulo n, where e is a small prime. However, the technique (or variations thereon) can be applied more generally.

Suppose that for $i = 1, \ldots, \ell$, the prover knows $x_i \in \mathbb{Z}_n^*$ such that $x_i^e = y_i$, and wants to convince a skeptical verifier of this. If e is small, we could use the technique of Exercise 19.6 to increase the challenge space size to e^k, and then apply the generalized AND-proof construction of Exercise 19.26. The resulting protocol would have communication complexity proportional to $k\ell$ times the cost of a single run of the GQ protocol ($O(k\ell)$ elements of $\mathbb{Z}_n^*$). In this exercise, we show how to do this with a protocol whose challenge space is of size e^ℓ and whose communication complexity is proportional to just ℓ times the cost of a single run of the GQ protocol.

Suppose $\boldsymbol{v} = (v_1, \ldots, v_\ell) \in (\mathbb{Z}_n^*)^{1\times\ell}$ is row vector of length ℓ with entries in the group $\mathbb{Z}_n^*$. Suppose $M = (m_{ij}) \in \mathbb{Z}^{\ell\times\ell}$ is an $\ell \times \ell$ matrix with integer entries. We define $\boldsymbol{v}^M$ to be the vector $(w_1, \ldots, w_\ell) \in (\mathbb{Z}_n^*)^{1\times\ell}$, where

$$w_i = v_1^{m_{1i}} \cdots v_\ell^{m_{\ell i}} \quad \text{for } i = 1, \ldots, \ell.$$

This is really just the usual rule for vector-matrix multiplication, except that the scalar "addition" operation in the group $\mathbb{Z}_n^*$ is written multiplicatively. For two vectors $\boldsymbol{v}, \boldsymbol{w} \in (\mathbb{Z}_n^*)^{1\times\ell}$, we write $\boldsymbol{v} \cdot \boldsymbol{w} \in (\mathbb{Z}_n^*)^{1\times\ell}$ for the component-wise product of $\boldsymbol{v}$ and $\boldsymbol{w}$. The usual rules of vector-matrix arithmetic carry over, for example, we have

$$(\boldsymbol{v} \cdot \boldsymbol{w})^M = \boldsymbol{v}^M \cdot \boldsymbol{w}^M, \quad \boldsymbol{v}^{M+N} = \boldsymbol{v}^M \cdot \boldsymbol{v}^N, \quad \text{and} \quad \boldsymbol{v}^{MN} = (\boldsymbol{v}^M)^N.$$

For $\boldsymbol{v} \in (\mathbb{Z}_n^*)^{1\times\ell}$ and integer f, we write $\boldsymbol{v}^f \in (\mathbb{Z}_n^*)^{1\times\ell}$ for the component-wise fth power of $\boldsymbol{v}$, that is, the vector whose ith entry is $v_i^f \in \mathbb{Z}_n^*$.

Let e be a prime, and let $I_e := \{0, \ldots, e-1\}$. The challenge space $\mathcal{C}$ for our Sigma protocol is $I_e^{1\times\ell}$. With each challenge $\boldsymbol{c} \in \mathcal{C}$, we associate an efficiently computable matrix $M_{\boldsymbol{c}} \in I_e^{\ell\times\ell}$. The essential property of these associated matrices is the following: given two distinct challenges $\boldsymbol{c}$ and $\boldsymbol{c}'$ in $\mathcal{C}$, we can efficiently compute a matrix $N \in I_e^{\ell\times\ell}$, such that $(M_{\boldsymbol{c}} - M_{\boldsymbol{c}'})N \equiv I \pmod{e}$, where I is the identity matrix. In other words, for all distinct $\boldsymbol{c}, \boldsymbol{c}' \in \mathcal{C}$, the matrix $(M_{\boldsymbol{c}} - M_{\boldsymbol{c}'}) \bmod e$ is invertible over the field $\mathbb{F}_e$.

If the statement is $\boldsymbol{y} \in (\mathbb{Z}_n^*)^{1\times\ell}$, and the witness is $\boldsymbol{x} \in (\mathbb{Z}_n^*)^{1\times\ell}$ such that $\boldsymbol{x}^e = \boldsymbol{y}$, then the protocol works as follows:

$$\begin{tikzcd}
\boldsymbol{x}_{\mathrm{t}} \xleftarrow{\mathrm{R}} (\mathbb{Z}_n^*)^{1\times\ell},\ \boldsymbol{y}_{\mathrm{t}} \leftarrow \boldsymbol{x}_{\mathrm{t}}^e & {} \arrow[r, "\boldsymbol{y}_{\mathrm{t}}"] & {} & \\
 & & & \boldsymbol{c} \xleftarrow{\mathrm{R}} \mathcal{C} \\
 & {} & {} \arrow[l, "\boldsymbol{c}"'] & \\
\boldsymbol{x}_{\mathrm{z}} \leftarrow \boldsymbol{x}_{\mathrm{t}} \cdot \boldsymbol{x}^{M_{\boldsymbol{c}}} & & & \\
 & {} \arrow[r, "\boldsymbol{x}_{\mathrm{z}}"] & {} & \\
 & & & \boldsymbol{x}_{\mathrm{z}}^e \stackrel{?}{=} \boldsymbol{y}_{\mathrm{t}} \cdot \boldsymbol{y}^{M_{\boldsymbol{c}}}
\end{tikzcd}$$

(a) Assuming the associated matrices $M_{\boldsymbol{c}}$ have the stated properties, prove that the above protocol provides special soundness and is special HVZK.

(b) Show how to define the matrix $M_{\boldsymbol{c}}$ associated with challenge $\boldsymbol{c} \in \mathcal{C}$ with the stated properties.

Hint: Use a finite field of cardinality e^ℓ.

(c) A straightfoward implementation takes $O(\ell^2 \log(e))$ multiplications in $\mathbb{Z}_n^*$ for both prover and verifier. Show how to reduce this to $O(\ell^2 \log(e)/\log(\ell))$ with precomputation.

19.30 (ℓ-special soundness and multi-extractability). We can generalize Definition 19.4 for special soundness as follows. Let (P, V) be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Let $\ell \geq 2$ be a constant or poly-bounded system parameter. We say that (P, V) provides **ℓ-special soundness** if there is an efficient witness extractor algorithm *Ext* with the following property: whenever *Ext* is given as input a statement $y \in \mathcal{Y}$, and ℓ accepting conversations of the form

$$(t, c_1, z_1), \ldots, (t, c_\ell, z_\ell),$$

where the c_i's are pairwise distinct, algorithm *Ext* always outputs $x \in \mathcal{X}$ such that $(x, y) \in \mathcal{R}$ (i.e., x is a witness for y).

Prove the following generalization of Theorem 19.24: if Π is a non-interactive proof system derived, is a Sigma protocol Π with a large challenge space that provides ℓ-special soundness, then Φ is multi-extractable.

To do this, you should design a multi-extractor that for every B-bounded adversary, and for every prescribed error probability θ, re-runs the adversary $O(B\ell/\theta)$ times, and fails with probability at most θ provided $\theta \geq 2B(\ell - 1)/N$, where N is the size of Π's challenge space.

Hint: Exercise 19.5(b) may be helpful.

19.31 (Compressed n-wise Schnorr). This exercise asks you to analyze a compressed n-wise Schnorr protocol. Let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$. Let n be a poly-bounded parameter, and let $\mathcal{R}_n \subseteq (\mathbb{Z}_q^n) \times (\mathbb{G}^n)$ be the following relation:

$$\mathcal{R}_n := \Big\{ \big((\alpha_1, \ldots, \alpha_n), (u_1, \ldots, u_n)\big) \quad : \quad u_i = g^{\alpha_i} \text{ for } i = 1, \ldots, n \Big\}.$$

$$\begin{array}{ccc}
\underline{P(\alpha_1,\dots,\alpha_n)} & & \underline{V(u_1,\dots,u_n)} \\
\alpha_0 \overset{\mathrm{R}}{\leftarrow} \mathbb{Z}_q,\ u_0 \leftarrow g^{\alpha_0} \in \mathbb{G} & \xrightarrow{\qquad u_0 \qquad} & \\
 & & c \overset{\mathrm{R}}{\leftarrow} \mathcal{C} \\
 & \xleftarrow{\qquad c \qquad} & \\
\beta \leftarrow \displaystyle\sum_{i=0}^{n} \alpha_i c^i \in \mathbb{Z}_q & \xrightarrow{\qquad \beta \qquad} & \\
 & & g^{\beta} \overset{?}{=} \displaystyle\prod_{i=0}^{n} u_i^{(c^i)}
\end{array}$$

Figure 19.12: A compressed n-wise Schnorr protocol

Our goal is to design a Sigma protocol that lets a prover P convince a verifier V who has $\boldsymbol{u} := (u_1, \dots, u_n)$ that he knows $\boldsymbol{\alpha}$ such that $(\boldsymbol{\alpha}, \boldsymbol{u}) \in \mathcal{R}_n$. One solution is to run n instances of the Schnorr protocol from Section 19.1 in parallel (see Exercise 19.26). However, the amount of traffic in the resulting protocol is linear in n. In this exercise we develop a protocol that requires only constant communication. The complete protocol is shown in Fig. 19.12.

(a) Show that protocol is special HVZK.

(b) Show that the protocol provides $(n+1)$-special soundness, as defined in Exercise 19.30.

Discussion: Exercise 20.27 shows how to apply the same technique to the Chaum-Pedersen protocol for DH-triples.

19.32 (Multi-extractability under weaker conditions). Show that Theorem 19.24 holds if Π only provides computational special soundness (as in Exercise 19.13) rather than special soundness. The concrete bounds stated in that theorem still hold, but the multi-extractor may fail with probability $\theta + \eta$, where η the advantage of an adversary in the computational special soundness attack game.

Chapter 20

Proving properties in zero-knowledge

In the previous chapter, we saw how to use Sigma protocols to construct identification and signature schemes. In these applications we used Sigma protocols as "proofs of knowledge" — using rewinding and special soundness, we could effectively extract a witness from any convincing prover.

In this chapter, we will see how to use Sigma protocols to prove that certain facts are true (without disclosing much else). In applications that use Sigma protocols in this way, security hinges on the *truth* of the alleged fact, not any notion of *knowledge*. For example, the Chaum-Pedersen protocol (Section 19.5.2) allows a prover to convince a verifier that a given triple of group elements is a DH-triple. That ability in itself is a useful tool in constructing and analyzing interesting cryptographic protocols.

In Section 20.1, we begin by defining the *language of true statements* associated with an effective relation: this is just the set of statements for which there exists a corresponding witness. Then we define a notion of *soundness* for a Sigma protocol, which just means that it is infeasible for any prover to make the verifier accept a statement that is not true (i.e., does not have a witness). This notion differs from special soundness, in that we do not require any kind of witness extractor. However, we shall see that special soundness implies soundness.

In Section 20.2, we will present a series of examples that illustrate soundness. These examples revolve around the idea of proving properties on encrypted data.

In Section 20.3, we show how to turn Sigma protocols into non-interactive proofs, using a variant of the Fiat-Shamir transform (see Section 19.6.1).

In later sections, we examine more advanced techniques for building proof systems.

20.1 Languages and soundness

We begin with a definition.

Definition 20.1 (The language of true statements). *Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be an effective relation. We say a statement $y \in \mathcal{Y}$ is a* ***true statement*** *if $(x, y) \in \mathcal{R}$ for some $x \in \mathcal{X}$; otherwise, we say $y \in \mathcal{Y}$ is a* ***false statement****. We define $L_{\mathcal{R}}$, which is called* ***language defined by*** *$\mathcal{R}$, to be the set of all true statements; that is, $L_{\mathcal{R}} := \{y \in \mathcal{Y} : (x, y) \in \mathcal{R} \text{ for some } x \in \mathcal{X}\}$.*

The term "language" comes from complexity theory. In this chapter, we will look at a number of interesting relations $\mathcal{R}$ and the languages $L_{\mathcal{R}}$ defined by them. To give an example from the previous chapter, recall that the Chaum-Pedersen protocol is a Sigma protocol for the following relation:

$$\mathcal{R} := \Big\{ \ \big(\ \beta, \ (u,v,w) \ \big) \in \mathbb{Z}_q \times \mathbb{G}^3 : \ v = g^\beta \text{ and } w = u^\beta \ \Big\}.$$

The language $L_\mathcal{R}$ defined by $\mathcal{R}$ is the set of all DH-triples $(u,v,w) \in \mathbb{G}^3$.

We can now define the notion of soundness using the following attack game:

Attack Game 20.1 (Soundness). Let $\Pi = (P,V)$ be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. For a given adversary $\mathcal{A}$, the attack game runs as follows:

- The adversary chooses a statement $y^* \in \mathcal{Y}$ and gives this to the challenger.
- The adversary now interacts with the verifier $V(y^*)$, where the challenger plays the role of verifier and the adversary plays the role of a possibly "cheating" prover.

We say that the adversary wins the game if $V(y^*)$ outputs accept but $y^* \notin L_\mathcal{R}$. We define $\mathcal{A}$'s advantage with respect to Π, denoted $\mathrm{Snd}\mathsf{adv}[\mathcal{A},\Pi]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 20.2. *We say that* Π *is* ***sound*** *if for all efficient adversaries* $\mathcal{A}$*, the quantity* $\mathrm{Snd}\mathsf{adv}[\mathcal{A},\Pi]$ *is negligible.*

Theorem 20.1 (Special soundness implies soundness). *Let* Π *be a Sigma protocol with a large challenge space. If* Π *provides special soundness, then* Π *is sound.*

In particular, for every adversary $\mathcal{A}$*, we have*

$$\mathrm{Snd}\mathsf{adv}[\mathcal{A},\Pi] \le \frac{1}{N}, \tag{20.1}$$

where N *is the size of the challenge space.*

Proof. It will suffice to show that if $\mathcal{A}$ chooses a false statement y^* and a commitment t^*, then there can be at most one challenge c for which there exists a response z that yields an accepting conversation (t^*, c, z) for y^*. Observe that if there were two such challenges, then there would be two accepting conversations (t^*, c, z) and (t^*, c', z') for y^*, with $c \neq c'$, and special soundness would imply that there exists a witness for y^*, which is not the case. □

Remark 20.1 (Statistical vs computational soundness). Note that the above theorem holds even for arbitrarily powerful adversaries. In this case we may say that the proof system is **statistically sound**. Our notion of soundness is sometimes called **computational soundness**, to explicitly distinguish it from statistical soundness. □

We put these ideas to use in the next section.

20.2 Proving properties on encrypted data

In a number of applications, the following scenario arises. Alice encrypts a message m under Bob's public key, obtaining a ciphertext c. In addition, Alice wants to prove to a third party, say Charlie (who gets to see c but not m), that the encrypted plaintext m satisfies a certain property, without revealing to Charlie anything else about m.

A Sigma protocol that is sound and special HVZK can be used to solve this type of problem. Such a protocol is not a complete solution, however. One problem is that the HVZK property only ensures that no information about m is leaked assuming that Charlie honestly follows the verification protocol. One way to address this issue is to use the same idea that we used in Section 19.6.1 to turn interactive identification protocols into signatures. That is, instead of using an actual verifier to generate the random challenge, we instead generate the challenge using a hash function. We will investigate this approach in detail in the next section. For now, let us look at a few interesting and important examples that show how we can use Sigma protocols to prove properties on encrypted data.

In our examples, it is convenient to use the multiplicative variant of the ElGamal encryption scheme, discussed in Exercise 11.5. This scheme makes use of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. The secret key is $\alpha \in \mathbb{Z}_q$ (which is chosen at random) and the public key is $u := g^\alpha \in \mathbb{G}$. The encryption of $m \in \mathbb{G}$ is $(v, e) \in \mathbb{G}^2$, where $v := g^\beta$, $e := u^\beta \cdot m$, and $\beta \in \mathbb{Z}_q$ is chosen at random. To decrypt (v, e) using the secret key α, one computes $m := e/v^\alpha$. As you were asked to show in Exercise 11.5, this scheme is semantically secure under the DDH assumption for $\mathbb{G}$.

Example 20.1 (Equal plaintexts). Suppose Alice has one ciphertext (v_0, e_0) that encrypts a message m under Bob's public key u_0, and another (v_1, e_1), that encrypts the same message m under Bill's public key u_1. She wants to convince Charlie that this is the case, without revealing anything else. For example, some protocols may require that Alice broadcast the same message to Bob and Bill. A protocol for this problem allows Alice to do this, while keeping her message encrypted, but proving that she really did encrypt the same message.

So we want a Sigma protocol for the relation

$$\mathcal{R} := \Big\{ \ (\ (\beta_0, \beta_1, m),\ (u_0, v_0, e_0,\ u_1, v_1, e_1)\ ) \ :\ v_0 = g^{\beta_0},\ e_0 = u_0^{\beta_0} \cdot m,\ v_1 = g^{\beta_1},\ e_1 = u_1^{\beta_1} \cdot m \ \Big\}.$$

The language $L_\mathcal{R}$ is precisely the set of tuples $(u_0, v_0, e_0,\ u_1, v_1, e_1)$ such that (v_0, e_0) and (v_1, e_1) encrypt the same message under the public keys u_0 and u_1.

To design an efficient Sigma protocol for $\mathcal{R}$, we observe that

$$\begin{aligned}(u_0, v_0, e_0,\ u_1, v_1, e_1) \in L_\mathcal{R} \quad \Longleftrightarrow \quad & v_0 = g^{\beta_0}, \quad v_1 = g^{\beta_1}, \quad \text{and} \quad e_0/e_1 = u_0^{\beta_0} u_1^{-\beta_1} \\ & \text{for some } \beta_0, \beta_1 \in \mathbb{Z}_q.\end{aligned}$$

Based on this observation, we can implement a Sigma protocol for $\mathcal{R}$ using the generic linear protocol from Section 19.5.3. Specifically, Alice proves to Charlie that there exist β_0, β_1 satisfying the system of equations

$$v_0 = g^{\beta_0}, \quad v_1 = g^{\beta_1}, \quad e_0/e_1 = u_0^{\beta_0} u_1^{-\beta_1}.$$

The result is a sound, special HVZK Sigma protocol for the relation $\mathcal{R}$.

Note that while Alice does not explicitly use the message m in the above protocol, she anyway needs to know it, since she needs to know both β_0 and β_1, either one of which determine m. □

Example 20.2 (Equal plaintexts, again). Consider a variation of the previous example in which Alice has two ciphertexts, (v_0, e_0) and (v_1, e_1), that encrypt the same message under Bob's public key u. The difference now is that both ciphertexts encrypt the same message under the *same* public key. Again, she wants to convince Charlie that this is the case, without revealing anything else. Observe that if (v_0, e_0) and (v_1, e_1) encrypt the same message, then

$$v_0 = g^{\beta_0}, \quad e_0 = u^{\beta_0} \cdot m, \quad v_1 = g^{\beta_1}, \quad e_1 = u^{\beta_1} \cdot m$$

for some $\beta_0, \beta_1 \in \mathbb{Z}_q$ and $m \in \mathbb{G}$. Dividing the first equation by the third, and the second by the fourth, we have

$$v_0/v_1 = g^{\beta} \quad \text{and} \quad e_0/e_1 = u^{\beta}, \tag{20.2}$$

where $\beta := \beta_0 - \beta_1$. Moreover, it is not hard to see that if (20.2) holds for some $\beta \in \mathbb{Z}_q$, then (v_0, e_0) and (v_1, e_1) encrypt the same message.

Therefore, all Alice needs to do is to convince Charlie that there exists β satisfying (20.2). She can do this using the generic linear protocol from Section 19.5.3, which in this case is really just the Chaum-Pedersen protocol (see Section 19.5.2) for proving that $(u, v_0/v_1, e_0/e_1)$ is a DH-triple.

Note that to prove that (v_0, e_0) and (v_1, e_1) encrypt the same message, Alice only needs to know the value β satisfying (20.2) — she does not need to know the message itself. In particular, Alice need not have been the party that generated these ciphertexts. In fact, she could have received the ciphertext (v_0, e_0) from another party, and then created a new encryption (v_1, e_1) of the same message by computing $v_1 := v_0 \cdot g^{\beta}$ and $e_1 := e_0 \cdot u^{\beta}$ for a value β of her choice. Some anonymity services perform precisely this type of function, creating a fresh re-encryption of an encrypted message. This protocol can be used to ensure that this was done correctly. □

Example 20.3 (Encrypted bits). To encrypt a bit $b \in \{0, 1\}$, it is convenient to encode b as the group element $g^b \in \mathbb{G}$, and then encrypt g^b using multiplicative ElGamal. So suppose Alice has encrypted a bit b in this way, under Bob's public key u, producing a ciphertext $(v, e) = (g^{\beta}, u^{\beta} \cdot g^b)$. She wants to convince Charlie that (v, e) really does encrypt a bit under Bob's public key (and not, say, g^{17}), without revealing anything else.

So we want a Sigma protocol for the relation

$$\mathcal{R} := \Big\{ \ (\ (b, \beta),\ (u, v, e)\ )\ :\ v = g^{\beta},\ \ e = u^{\beta} \cdot g^b,\ \ b \in \{0, 1\}\ \Big\}.$$

The language $L_{\mathcal{R}}$ corresponding to this relation is precisely the set of tuples (u, v, e) such that (v, e) encrypts a bit under the public key u.

Our Sigma protocol for $\mathcal{R}$ is based on the observation that

$$(u, v, e) \in L_{\mathcal{R}} \iff \text{either } (u, v, e) \text{ or } (u, v, e/g) \text{ is a DH-triple.}$$

The Chaum-Pedersen protocol in Section 19.5.2 allows a party to prove that a given triple is a DH-triple. We combine this with the OR-proof construction in Section 19.7.2. This gives us a Sigma protocol for the relation

$$\mathcal{R}' := \Big\{ \ \big(\ (b, \beta),\ ((u_0, v_0, w_0), (u_1, v_1, w_1))\ \big)\ :\ v_b = g^{\beta} \ \text{ and } \ w_b = u_b^{\beta}\ \Big\}.$$

A statement $\big((u_0, v_0, w_0), (u_1, v_1, w_1)\big)$ is in $L_{\mathcal{R}'}$ if at least one of (u_0, v_0, w_0) or (u_1, v_1, w_1) is a DH-triple. Then, we have

$$(u, v, e) \in L_{\mathcal{R}} \iff ((u, v, e), (u, v, e/g)) \in L_{\mathcal{R}'}.$$

$$\begin{array}{lcl}
\underline{P\big((b,\beta),(u,v,e)\big)} & & \underline{V(u,v,e)} \\
 & \text{set } w_0 := e,\ w_1 := e/g & \\
\beta_{\mathrm{t}b} \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathbb{Z}_q,\ v_{\mathrm{t}b} \leftarrow g^{\beta_{\mathrm{t}b}},\ w_{\mathrm{t}b} \leftarrow u^{\beta_{\mathrm{t}b}} & & \\
d \leftarrow 1-b,\ c_d \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathcal{C},\ \beta_{\mathrm{z}d} \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathbb{Z}_q & & \\
v_{\mathrm{t}d} \leftarrow g^{\beta_{\mathrm{z}d}}/v^{c_d},\ w_{\mathrm{t}d} \leftarrow u^{\beta_{\mathrm{z}d}}/w_d^{c_d} & & \\
 & \xrightarrow{\quad v_{\mathrm{t}0},\, w_{\mathrm{t}0},\, v_{\mathrm{t}1},\, w_{\mathrm{t}1}\quad} & c \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathcal{C} \\
c_b \leftarrow c \oplus c_d,\ \beta_{\mathrm{z}b} \leftarrow \beta_{\mathrm{t}b} + \beta c_b & \xleftarrow{\qquad\quad c \qquad\quad} & \\
 & \xrightarrow{\quad c_0,\, \beta_{\mathrm{z}0},\, \beta_{\mathrm{z}1}\quad} & \text{compute } c_1 \leftarrow c \oplus c_0 \text{ and verify that} \\
 & & g^{\beta_{\mathrm{z}0}} = v_{\mathrm{t}0} \cdot v^{c_0},\quad u^{\beta_{\mathrm{z}0}} = w_{\mathrm{t}0} \cdot w_0^{c_0} \\
 & & g^{\beta_{\mathrm{z}1}} = v_{\mathrm{t}1} \cdot v^{c_1},\quad u^{\beta_{\mathrm{z}1}} = w_{\mathrm{t}1} \cdot w_1^{c_1}
\end{array}$$

Figure 20.1: Sigma protocol for encrypted bits

So, for Alice to prove to Charlie that $(u, v, e) \in L_{\mathcal{R}}$, they run the Sigma protocol for $\mathcal{R}'$, using the statement $((u, v, e), (u, v, e/g))$ and the witness (b, β). For completeness, we give the entire Sigma protocol for $\mathcal{R}$ in Fig. 20.1. In the first line of the prover's logic, the prover is initiating the proof for the witness it knows, and the second and third lines are running the HVZK simulator for the witness it does not know. The resulting Sigma protocol for $\mathcal{R}$ is sound and special HVZK.

This protocol generalizes to proving that a ciphertext (v, e) encrypts a value $0 \leq b < B$ for $B > 2$, as discussed in Exercise 20.1. The protocol transcript grows linearly in B, so this can only be used for relatively small B. We will see how to handle larger B in Section 20.4.1. □

Example 20.4 (Encrypted DH-triples). Suppose Alice has a DH-triple $(g^{\gamma_1}, g^{\gamma_2}, g^{\gamma_3})$, where $\gamma_3 = \gamma_1\gamma_2$. She encrypts each element under Bob's public key u, producing three ciphertexts $(v_1, e_1), (v_2, e_2), (v_3, e_3)$, where

$$v_i = g^{\beta_i}, \quad e_i = u^{\beta_i} g^{\gamma_i} \quad \text{for } i = 1, 2, 3. \tag{20.3}$$

She presents these ciphertexts to Charlie, and wants to convince him that these ciphertexts really do encrypt a DH-triple, without revealing anything else.

So we want a Sigma protocol for the relation

$$\mathcal{R} := \Big\{ \ \big(\ (\beta_1, \beta_2, \beta_3, \gamma_1, \gamma_2, \gamma_3),\ (u,\ v_1, e_1,\ v_2, e_2,\ v_3, e_3)\ \big)\ : \\ v_i = g^{\beta_i},\ e_i = u^{\beta_i} g^{\gamma_i} \text{ for } i = 1, 2, 3 \quad \text{and} \quad \gamma_3 = \gamma_1\gamma_2 \ \Big\}.$$

The corresponding language $L_{\mathcal{R}}$ is precisely the set of tuples $(u,\ v_1, e_1,\ v_2, e_2,\ v_3, e_3)$ such that the ciphertexts $(v_1, e_1), (v_2, e_2), (v_3, e_3)$ encrypt a DH-triple under the public key u.

While the relation $\mathcal{R}$ is inherently non-linear because of the condition $\gamma_3 = \gamma_1\gamma_2$, we can nevertheless design a Sigma protocol for $\mathcal{R}$ using the generic linear protocol from Section 19.5.3.

The basic idea is that Alice proves to Charlie that there exist $\beta_1, \beta_3, \gamma_1, \tau$ satisfying the system of equations:

$$v_1 = g^{\beta_1}, \quad e_1 = u^{\beta_1} g^{\gamma_1}, \quad v_3 = g^{\beta_3}, \quad v_2^{\gamma_1} = g^{\tau}, \quad e_2^{\gamma_1} u^{\beta_3} = e_3 u^{\tau}. \tag{20.4}$$

To prove that this works, we claim that $(u,\ v_1, e_1,\ v_2, e_2,\ v_3, e_3) \in L_{\mathcal{R}}$ if and only if there exist $\beta_1, \beta_3, \gamma_1, \tau$ satisfying (20.4). Observe that the ciphertexts $(v_1, e_1), (v_2, e_2), (v_3, e_3)$ uniquely determine β_i's and the γ_i's satisfying (20.3). These values of β_1, β_3, and γ_1 are also the unique values satisfying the first three equations in (20.4). The fourth equation in (20.4) is satisfied uniquely by setting $\tau := \gamma_1 \beta_2$. So it remains to consider the last equation in (20.4). The left-hand side is

$$e_2^{\gamma_1} u^{\beta_3} = (u^{\beta_2} g^{\gamma_2})^{\gamma_1} u^{\beta_3} = u^{\beta_3 + \tau} g^{\gamma_1 \gamma_2},$$

while the right-hand side is

$$e_3 u^{\tau} = (u^{\beta_3} g^{\gamma_3}) u^{\tau} = u^{\beta_3 + \tau} g^{\gamma_3}.$$

So this equation is satisfied if and only if $\gamma_1 \gamma_2 = \gamma_3$. That proves the claim.

So this gives us a Sigma protocol for $\mathcal{R}$. To run the protocol, Alice runs the generic linear protocol for (20.4) using the witness $(\beta_1, \beta_3, \gamma_1, \tau := \gamma_1 \beta_2)$. Correctness, soundness, and special HVZK all follow from the corresponding properties for the generic linear protocol. □

Example 20.5 (Encrypted bits, again). We can use the idea from the previous example to get another Sigma protocol for the encrypted bits problem in Example 20.3.

If Alice wants to prove to Charlie that a ciphertext (v, e) is of the form $v = g^{\beta}$, $e = u^{\beta} g^{b}$, where $b \in \{0, 1\}$, it suffices for her to show that $b^2 = b$, as the only values of $b \in \mathbb{Z}_q$ that satisfy $b^2 = b$ are $b = 0$ and $b = 1$.

So, using the generic linear protocol, Alice proves to Charlie that there exist $b, \beta, \tau (= \beta b)$ satisfying the system of equations:

$$v = g^{\beta}, \quad e = u^{\beta} g^{b}, \qquad v^{b} = g^{\tau}, \quad e^{b} = u^{\tau} g^{b}.$$

We leave it to the reader to verify that this yields a sound, special HVZK Sigma protocol for the relation $\mathcal{R}$ in Example 20.3. The resulting protocol offers similar performance as the encrypted bits protocol of Example 20.3.

The protocol generalizes to prove to Charlie that a ciphertext (v, e) encrypts a value b satisfying $0 \le b < B$ for some $B > 2$. The generalization uses a Sigma protocol, presented in the next example, to convince Charlie that b satisfies the polynomial relation $b(b-1)(b-2)\cdots(b-(B-1)) = 0$. This relation implies that $0 \le b < B$. The protocol transcript grows linearly in B and therefore can only be used for small B. □

Example 20.6 (Polynomial relations). We can extend the idea from Example 20.4 even further. Suppose Alice has two ciphertexts (v, e) and (v', e') under Bob's public key u. The first ciphertext encrypts a group element g^{γ} and the second encrypts $g^{\gamma'}$. Alice wants to convince Charlie that $\gamma' = f(\gamma)$ for some specific polynomial $f(x) = \sum_{i=0}^{d} \lambda_i x^i$. We shall assume that the degree d and the coefficients $\lambda_0, \ldots, \lambda_d$ of $f(x)$ are fixed, public values (constants or system parameters).

So we want a Sigma protocol for the relation

$$\mathcal{R} = \Big\{ \ (\ (\beta, \gamma, \beta', \gamma'), (u,\ v, e,\ v', e')\ )\ :\ v = g^{\beta},\ e = u^{\beta} \cdot g^{\gamma},\ v' = g^{\beta'},\ e' = u^{\beta'} \cdot g^{\gamma'},\ \gamma' = f(\gamma)\ \Big\}.$$

To get a Sigma protocol for $\mathcal{R}$, Alice and Charlie use the generic linear protocol, where Alice proves to Charlie that there exist

$$\beta, \quad \gamma_1, \ldots, \gamma_d, \quad \tau_1, \ldots, \tau_{d-1}, \quad \beta', \gamma'$$

satisfying the system of equations:

$$\begin{aligned}
&v = g^{\beta}, \quad e = u^{\beta} g^{\gamma_1}, \quad v' = g^{\beta'}, \quad e' = u^{\beta'} g^{\gamma'}, \quad \gamma' = \lambda_0 + \lambda_1 \gamma_1 + \cdots + \lambda_d \gamma_d, \\
&v^{\gamma_i} = g^{\tau_i}, \quad e^{\gamma_i} = u^{\tau_i} g^{\gamma_{i+1}} \quad (i = 1, \ldots, d-1).
\end{aligned}$$

Note that here, we are using the generalized version of the generic linear protocol, which handles the equations over both $\mathbb{G}$ and $\mathbb{Z}_q$ (see discussion after Theorem 19.11). Alice runs the protocol using $\gamma_i := \gamma^i$ for $i = 1, \ldots, d$ and $\tau_i := \beta\gamma^i$ for $i = 1, \ldots, d-1$. The reader may verify that these are in fact the only values that satisfy this system of equations. This is easily seen by a simple induction argument. It follows that the resulting Sigma protocol is a sound, special HVZK Sigma protocol for the relation $\mathcal{R}$. □

The above examples all illustrate the notion of a *language reduction.* In general, such a reduction from $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ to $\mathcal{R}' \subseteq \mathcal{X}' \times \mathcal{Y}'$ is a pair of efficiently computable maps $f : \mathcal{X} \times \mathcal{Y} \to \mathcal{X}'$ and $g : \mathcal{Y} \to \mathcal{Y}'$, such that

(i) $(f(x, y), g(y)) \in \mathcal{R}'$ for all $(x, y) \in \mathcal{R}$, and

(ii) $g(y) \in L_{\mathcal{R}'} \implies y \in L_{\mathcal{R}}$ for all $y \in \mathcal{Y}$.

Using such a reduction, we can use a Sigma protocol Π' for $\mathcal{R}'$ to build a Sigma protocol Π for $\mathcal{R}$. The first condition ensures that Π inherits correctness and special HVZK from Π', and the second ensures that Π inherits soundness from Π'. Knowledge soundness need not always be inherited — that is, it is not required that a witness for y can be recovered from a witness for $g(y)$. In almost all of the above examples above, the relation $\mathcal{R}'$ was a special case of the generic linear relation. The only exception was Example 20.3, where the relation $\mathcal{R}'$ arose from the OR-proof construction.

20.2.1 A generic protocol for non-linear relations

In several of the examples above, we saw that we could use the generic linear protocol to prove certain non-linear relations. We now show how to do this with much greater generality. As we will see, the protocol for polynomial evaluation in Example 20.6 can be easily derived as a special case of this construction. This same general construction could also be used to derive protocols for the problems in Examples 20.4 and 20.5; however, the resulting protocols would not be quite as efficient as the ones presented in those two examples.

As usual, let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Consider the generic linear protocol in Section 19.5.3. That protocol works with formulas ϕ of the form described in (19.13). Suppose that we also allow non-linear equations of the form $x_i = x_j \cdot x_k$ in ϕ. To make this construction work, we will require that for each such non-linear equation, ϕ also contains two auxiliary equations, which are of the form

$$v = g^{x_\ell} \quad \text{and} \quad e = u^{x_\ell} g^{x_j}, \tag{20.5}$$

where u, v, and e are group elements, and x_ℓ is some variable. To keep things simple, let us assume that in the description of ϕ, there is a pointer of some kind from each non-linear equation to the corresponding auxiliary equations.

We can transform such a formula ϕ into a formula ϕ' that can be handled by the generic linear protocol, as follows. For each non-linear equation $x_i = x_j \cdot x_k$ in ϕ, with corresponding auxiliary equations as in (20.5), we introduce a new temporary variable t, and replace $x_i = x_j \cdot x_k$ by the pair of equations

$$v^{x_k} = g^t \quad \text{and} \quad e^{x_k} = u^t h^{x_i}. \tag{20.6}$$

The result of this transformation is a formula ϕ' that can be handled by the generic linear protocol. The Sigma protocol for ϕ works as follows. Both prover and verifier can transform ϕ into ϕ'. Suppose the prover has an assignment $(\alpha_1, \ldots, \alpha_n)$ to the variables $(x_1, \ldots, x_n)$ that makes the formula ϕ true. For each non-linear equation $x_i = x_j \cdot x_k$ in ϕ, the prover assigns to the temporary variable t in (20.6) the value $\alpha_k \alpha_\ell$, and then runs the generic linear protocol for ϕ' with the verifier, using this extended assignment.

We leave it to the reader to verify that this transformation yields a Sigma protocol that is special HVZK and provides special soundness for the relation (19.14), where the formulas ϕ are now allowed to have the non-linear form described above.

Polynomial evaluation, again. The protocol in Example 20.6 can be derived using this transformation. With notation as in that example, Alice proves to Charlie that there exist

$$\beta, \quad \gamma_1, \ldots, \gamma_d, \quad \beta', \gamma'$$

satisfying the system of equations:

$$\begin{aligned}
&v = g^\beta, \quad e = u^\beta g^{\gamma_1}, \quad v' = g^{\beta'}, \quad e' = u^{\beta'} g^{\gamma'}, \quad \gamma' = \lambda_0 + \lambda_1 \gamma_1 + \cdots + \lambda_d \gamma_d, \\
&\gamma_{i+1} = \gamma_1 \cdot \gamma_i \quad (i = 1, \ldots, d-1).
\end{aligned}$$

The reader should verify that the non-linear to linear transformation converts each equation $\gamma_{i+1} = \gamma_1 \cdot \gamma_i$ to the pair of equations $v^{\gamma_i} = g^{\tau_i}$ and $\gamma_{i+1} = \gamma_1 \cdot \gamma_i$.

Encrypted bits, yet again. The protocol in Example 20.5 can be derived using this transformation. Alice proves to Charlie that there exist b, β such that

$$v = g^\beta, \quad e = u^\beta g^b, \quad b = b \cdot b.$$

We leave it to the reader to show that applying the non-linear to linear transformation to this system of equations yields precisely the protocol in Example 20.5.

Encrypted DH triples, again. We could also attempt to use this technique to design a protocol for the problem in Example 20.4. The most obvious approach would be for Alice to prove to Charlie that there exist

$$\beta_1, \beta_2, \beta_3, \quad \gamma_1, \gamma_2, \gamma_3$$

such that

$$v_i = g^{\beta_i}, \; e_i = u^{\beta_i} g^{\gamma_i} \text{ for } i = 1, 2, 3 \quad \text{and} \quad \gamma_3 = \gamma_1 \gamma_2.$$

We can just plug this system of equations in the above non-linear to linear transformation. This works, but the resulting protocol would not be quite as efficient as the one in Example 20.4.

Removing constraints on the non-linear equation. While our generic transformation is quite useful, it is still somewhat constrained. Indeed, we essentially require that for each non-linear equation $x_i = x_j \cdot x_k$, the system of equations must also include equations describing the encryption of either x_j or x_k using multiplicative ElGamal. Later, in Section 20.4.3, we will see how to drop this requirement, if we are willing to work with a weaker (but still useful) form of HVZK (or a weaker form of special soundness — see Exercise 20.6).

20.3 Non-interactive proof systems

In this section, we show how to use the Fiat-Shamir transform (see Section 19.6.1) to convert any Sigma protocol into a *non-interactive proof system.*

The basic idea is very simple: instead of relying on a verifier to generate a random challenge, we use a hash function H to derive the challenge from the statement and the commitment. If we model H as a random oracle, then we can prove the following:

(i) if the Sigma protocol is sound, then so is the non-interactive proof system;

(ii) if the Sigma protocol is special HVZK, then running the non-interactive proof system does not reveal any useful information about the prover's witness.

The first property is a fairly straightforward adaptation of the notion of soundness to the non-interactive setting. The second property is a new type of "zero knowledge" property that is a bit tricky to define.

20.3.1 Example: a voting protocol

Before getting into the formalities, we illustrate the utility of non-interactive proofs by showing how they can be used in the context of voting protocols.

It takes considerable effort to properly model a voting protocol — just formulating all of the security requirements is quite challenging. We will not attempt to do this here; rather, we will just illustrate some of the essential ideas, and hint at some of the remaining issues.

Suppose we have n voters, where each voter wants to cast a vote of 0 or 1. At the end of the election, all the parties should learn the sum of the votes.

Of course, each voter could simply publish their vote. However, this is not such a great solution, as we would like to allow voters to keep their votes private. To this end, some voting protocols make use of an encryption scheme, so that each voter publishes an encryption of their vote.

A convenient scheme to use for this purpose is the multiplicative variant of the ElGamal scheme, discussed in Section 20.2. Again, the setting is that we have a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$. The secret key is $\alpha \in \mathbb{Z}_q$ and the public key is $u := g^\alpha \in \mathbb{G}$. An encryption of $m \in \mathbb{G}$ is (v, e), where $v := g^\beta$, $e := u^\beta \cdot m$.

Here is an initial attempt at a voting protocol that provides some privacy to the voters.

Suppose that we have a trusted server, called the *vote tallying center (VTC)*, that runs the key generation algorithm, obtaining a public key $pk = u$ and a secret key $sk = \alpha$. It publishes pk for all voters to see, and keeps sk to itself.

Voting stage. In the voting stage, the ith voter encrypts its vote $b_i \in \{0, 1\}$ by encoding the vote b_i as the group element $g^{b_i} \in \mathbb{G}$, and encrypting this group element under the VTC's public key, obtaining a ciphertext (v_i, e_i). Note that $v_i = g^{\beta_i}$ and $e_i = u^{\beta_i} \cdot g^{b_i}$, where $\beta_i \in \mathbb{Z}_q$ is chosen at random. All of these ciphertexts are published.

Tallying stage. The VTC takes all of the published ciphertexts and aggregates them into a single ciphertext (v_*, e_*), where

$$v_* := \prod_{i=1}^{n} v_i \quad \text{and} \quad e_* := \prod_{i=1}^{n} e_i.$$

If $\beta_* := \sum_i \beta_i$ and $\sigma := \sum_i b_i$, then we see that

$$v_* = g^{\beta_*} \quad \text{and} \quad e_* = u^{\beta_*} g^{\sigma}.$$

Thus, (v_*, e_*) is an encryption of g^{σ}. So, the VTC can decrypt (v_*, e_*) and publish the result, so all the voters can see g^{σ}. Since σ itself is a small number, it is easy to compute σ from g^{σ}, just by brute-force search or table lookup.

If all the voters and the VTC correctly follow the protocol, then, at least intuitively, the semantic security of ElGamal encryption ensures that no voter learns anyone else's vote at the end of the voting stage. Moreover, at the end of the tallying stage, the voters learn only the sum of the votes. No extra information about any of the votes is revealed.

The above protocol is not very robust, in the sense that if any of the voters or the VTC are corrupt, both the correctness of the election result and the privacy of the votes may be compromised. For the time being, let us continue to assume that the VTC is honest (some of the exercises in this chapter will develop ideas that can be used to prevent the VTC from cheating). Rather, let us focus on the possibility of a cheating voter.

One way a voter can cheat is to encrypt a vote other than 0 or 1. So, for example, instead of encrypting the group element g^0 or g^1, he might encrypt the group element g^{100}. This would be equivalent to casting 100 1-votes, which would allow the voter to unfairly influence the outcome of the election.

To prevent this, when a voter casts its vote, we might insist that he *proves* that its encrypted vote (v, e) is *valid*, in the sense that it is of the form $(g^{\beta}, u^{\beta} \cdot g^{b})$, where $b \in \{0, 1\}$. To do this, we apply the Fiat-Shamir transform to the Sigma protocol in Example 20.3. The voter (using the witness (b, β)) simply runs the prover's logic in Fig. 20.1, computing the challenge for itself by hashing the statement and the commitment, in this case, as

$$c \leftarrow H(\ (u, v, e),\ (v_{t0}, w_{t0}, v_{t1}, w_{t1})\ ). \tag{20.7}$$

The voter then publishes the proof

$$\pi = (\ (v_{t0}, w_{t0}, v_{t1}, w_{t1}),\ (c_0, \beta_{z0}, \beta_{z1})\ ), \tag{20.8}$$

along with the ciphertext (v, e). Anyone (in particular, the VTC) can verify the validity of the proof π by checking that the same conditions that the verifier would normally check in Fig. 20.1 are satisfied, where c is computed from the hash function as in (20.7).

As we shall see, if we model the hash function H as a random oracle, then the proof is sound, in the sense that it is computationally infeasible to come up with a valid proof if the encrypted vote is not valid. Moreover, the zero-knowledge property will ensure that the proof itself does not leak any additional information about the vote. Indeed, if we define a new, augmented encryption scheme where ciphertexts are of the form (v, e, π), as above, then one can show that this augmented encryption scheme is semantically secure (under the DDH assumption, with H modeled as a random oracle model). In fact, this augmented encryption scheme is very similar to the scheme $\mathcal{E}_{\text{aMEG}}$ introduced in Section 19.9.2.1. Moreover, the analysis in Section 19.9.2.2 carries over to this scheme, so that one can show that this scheme is parallel CCA secure (see Exercise 20.28) and hence non-malleable (but just as in Section 19.9.2.2, the security reduction is very loose). In the application to voting, this is typically a a sufficient security property, rather than full CCA security. However, see Exercise 20.20 for a similar encryption scheme that provides full CCA security (with a tight security reduction) that is suitable for use in voting protocols as well as other applications.

We can optimize this proof system along the same lines that we optimized Schnorr's signatures in Section 19.2.3. Namely, instead of a proof π as in (20.8), we can use a proof of the form

$$\pi^* = (c_0, c_1, \beta_{z0}, \beta_{z1}).$$

To verify such a proof, one derives the values $v_{t0}, w_{t0}, v_{t1}, w_{t1}$ from the verification equations (computing $v_{t0} \leftarrow g^{\beta_{z0}}/v^{c_0}$, and so on), and then checks that $c_0 \oplus c_1 = H((u, v, e), (v_{t0}, w_{t0}, v_{t1}, w_{t1}))$. In practice, one would use this optimized system, as the proofs are much more compact, and provide the same security properties (both soundness and zero knowledge) as the unoptimized system. See Exercise 20.14 for more general conditions under which this type of optimization is possible.

20.3.2 Non-interactive proofs: basic syntax

We now get down to the business of defining non-interactive proofs in general, their security properties, and the details of the Fiat-Shamir transform.

We begin by defining the basic syntax of a non-interactive proof.

Definition 20.3 (Non-interactive proof system). *Let $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ be an effective relation. A **non-interactive proof system for** $\mathcal{R}$ is a pair of algorithms (GenPrf, VrfyPrf), where:*

- *GenPrf is an efficient probabilistic algorithm that is invoked as $\pi \xleftarrow{R} \mathit{GenPrf}(x, y)$, where $(x, y) \in \mathcal{R}$, and π belongs to some **proof space** $\mathcal{PS}$;*
- *VrfyPrf is an efficient deterministic algorithm that is invoked as $\mathit{VrfyPrf}(y, \pi)$, where $y \in \mathcal{Y}$ and $\pi \in \mathcal{PS}$; the output of VrfyPrf is either* accept *or* reject*. If $\mathit{VrfyPrf}(y, \pi) =$* accept*, we say π **is a valid proof for** y.*

We require that for all $(x, y) \in \mathcal{R}$, the output of $\mathit{GenPrf}(x, y)$ is always a valid proof for y.

20.3.3 The Fiat-Shamir transform

We now present in detail the Fiat-Shamir transform that converts a Sigma protocol into non-interactive proof system.

Let $\Pi = (P, V)$ be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Assume that conversations (t, c, z) for Π belong to $\mathcal{T} \times \mathcal{C} \times \mathcal{Z}$. Let $H : \mathcal{Y} \times \mathcal{T} \to \mathcal{C}$ be a hash function. We define the Fiat-Shamir non-interactive proof system FS-$\Pi = (\mathit{GenPrf}, \mathit{VrfyPrf})$, with proof space $\mathcal{PS} = \mathcal{T} \times \mathcal{Z}$, as follows:

- on input (x, y) in $\mathcal{R}$, *GenPrf* first runs $P(x, y)$ to obtain a commitment $t \in \mathcal{T}$; it then feeds the challenge $c := H(y, t)$ to $P(x, y)$, obtaining a response $z \in \mathcal{Z}$; the output is $(t, z) \in \mathcal{T} \times \mathcal{Z}$;

- on input $(y, (t, z)) \in \mathcal{Y} \times (\mathcal{T} \times \mathcal{Z})$, *VrfyPrf* verifies that (t, c, z) is an accepting conversation for y, where $c := H(y, t)$.

20.3.4 Non-interactive soundness

We next adapt our definition of soundness to the non-interactive setting. Essentially, the definition says that it is hard to cook up a valid proof of a false statement.

Attack Game 20.2 (Non-interactive Soundness). Let $\Phi = (\mathit{GenPrf}, \mathit{VrfyPrf})$ be a non-interactive proof system for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ with proof space $\mathcal{PS}$. To attack Φ, an adversary $\mathcal{A}$ outputs a statement $y^* \in \mathcal{Y}$ and a proof $\pi^* \in \mathcal{PS}$.

We say that the adversary wins the game if $\mathit{VrfyPrf}(y^*, \pi^*) = \mathsf{accept}$ but $y^* \notin L_{\mathcal{R}}$. We define $\mathcal{A}$'s advantage with respect to Φ, denoted $\text{niSnd}\mathsf{adv}[\mathcal{A}, \Phi]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 20.4. *We say that Φ is **sound** if for all efficient adversaries $\mathcal{A}$, the quantity* $\text{niSnd}\mathsf{adv}[\mathcal{A}, \Phi]$ *is negligible.*

We next show that under appropriate assumptions, the Fiat-Shamir transform yields a sound non-interactive proof system, if we model the hash function as a random oracle.

Theorem 20.2 (Fiat-Shamir-based proofs are sound). *Let Π be a Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, and let $\text{FS-}\Pi$ be the Fiat-Shamir non-interactive proof system derived from Π with hash function H. If Π is sound, and if we model H as a random oracle, then $\text{FS-}\Pi$ is sound.*

In particular, let $\mathcal{A}$ be an adversary attacking the soundness of $\text{FS-}\Pi$ as in the random oracle version of Attack Game 20.2. Moreover, assume that $\mathcal{A}$ issues at most Q_{ro} random oracle queries. Then there exists an adversary $\mathcal{B}$ that attacks the soundness of Π as in Attack Game 20.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{niSnd}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \text{FS-}\Pi] \leq (Q_{\text{ro}} + 1)\text{Snd}\mathsf{adv}[\mathcal{B}, \Pi].$$

Proof sketch. The basic idea is similar to what we did in the proof of security of Schnorr's signature scheme (Theorem 19.7). Suppose that $\mathcal{A}$ produces a valid proof (t^*, z^*) on a false statement y^*. This means that (t^*, c, z^*) is a valid conversation for y^*, where c is the output of the random oracle at the point (y^*, t^*). Without loss of generality, we can assume that $\mathcal{A}$ queries the random oracle at this point — if not, we make it so, increasing the number of random oracle queries to $Q_{\text{ro}} + 1$. Our adversary $\mathcal{B}$ then starts out by guessing (in advance) which of the $\mathcal{A}$'s random oracle queries will be the relevant one. At the point when $\mathcal{A}$ makes that random oracle query, $\mathcal{B}$ initiates a proof attempt with its own challenger, supplying y^* as the statement and t^* as the commitment message; $\mathcal{B}$'s challenger responds with a random challenge c, which $\mathcal{B}$ forwards to $\mathcal{A}$ as if this were the value of the random oracle at the point (y^*, t^*). If $\mathcal{B}$'s guess was correct, then the value z^* in $\mathcal{A}$'s proof will let $\mathcal{B}$ succeed in his attack game. The factor $(Q_{\text{ro}} + 1)$ in the concrete security bound comes from the fact that $\mathcal{B}$'s guess will be correct with probability $1/(Q_{\text{ro}} + 1)$. □

20.3.5 Non-interactive zero knowledge

Let $\Phi = (\mathit{GenPrf}, \mathit{VrfyPrf})$ be a non-interactive proof system for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ with proof space $\mathcal{PS}$. We wish to define a useful notion of "zero knowledge". Intuitively, we want this notion to capture the idea that the output of *GenPrf* on input (x, y) reveals nothing more than the fact that $y \in L_{\mathcal{R}}$.

Defining such a notion is rather tricky. The approach we take is similar to the approach we took for defining HVZK — namely, we want to say that there is a *simulator* that on input $y \in L_{\mathcal{R}}$ can faithfully simulate the output distribution of $GenPrf(x, y)$. Unfortunately, it is essentially impossible to make this idea work without giving the simulator some kind of "insider advantage". Indeed, if a simulator can generate a valid proof on input $y \in L_{\mathcal{R}}$, it may very well be the case that it outputs a valid proof on input $y \notin L_{\mathcal{R}}$, which would violate soundness; moreover, if the simulator failed to output a valid proof on input $y \notin L_{\mathcal{R}}$, we could use the simulator itself to distinguish between elements of $L_{\mathcal{R}}$ and elements of $\mathcal{Y} \setminus L_{\mathcal{R}}$, which for most languages of interest is computationally infeasible.

We shall only attempt to formulate non-interactive zero knowledge in the random oracle model, and the "insider advantage" that we give to our simulator is that it is allowed to simultaneously manage both the simulated output of *GenPrf* and the access to the random oracle.

Suppose that Φ makes use of a hash function $H : \mathcal{U} \to \mathcal{C}$, and that we wish to model H as a random oracle. A **simulator for** Φ is an interactive machine *Sim*[1] that responds to a series of queries, where each query is one of two types:

- $(\textsf{sim-proof-query}, y)$, where $y \in \mathcal{Y}$, to which *Sim* replies with $\pi \in \mathcal{PS}$;
- $(\textsf{sim-oracle-query}, u)$, where $u \in \mathcal{U}$, to which *Sim* replies with $c \in \mathcal{C}$.

Our definition of non-interactive zero knowledge (niZK) says that an efficient adversary cannot distinguish between a "real world", in which it asks for real proofs of true statements and a "simulated world" in which it just gets simulated proofs as generated by *Sim*. In both worlds, the hash function H is modeled as a random oracle, and the adversary gets to make random oracle queries, but in the simulated world, *Sim* processes these queries as well.

Attack Game 20.3 (Non-interactive zero knowledge). Let $\Phi = (GenPrf, VrfyPrf)$ be a non-interactive proof system for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ with proof space $\mathcal{PS}$. Suppose that Φ makes use of a hash function $H : \mathcal{U} \to \mathcal{C}$, which is modeled as a random oracle. Let *Sim* be a simulator for Φ, as above. For a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. In both experiments, the adversary makes a series of queries to the challenger, each of which is of the form:

- a *proof query*, which is of the form $(x, y) \in \mathcal{R}$, and to which the challenger replies with $\pi \in \mathcal{PS}$;
- a *random oracle query*, which is of the form $u \in \mathcal{U}$, and to which the challenger replies with $c \in \mathcal{C}$.

In Experiment 0 (the "real world"), the challenger chooses $\mathcal{O} \in \text{Funs}[\mathcal{U}, \mathcal{C}]$ at random, answering each proof query $(x, y) \in \mathcal{R}$ by running $GenPrf(x, y)$, using $\mathcal{O}$ in place of H, and answering each random oracle query $u \in \mathcal{U}$ with $\mathcal{O}(u)$.

In Experiment 1 (the "simulated world"), the challenger answers each proof query $(x, y) \in \mathcal{R}$ by passing $(\textsf{sim-proof-query}, y)$ to *Sim*, and answers each random oracle query $u \in \mathcal{U}$ by passing $(\textsf{sim-oracle-query}, u)$ to *Sim*.

[1] Formally, a simulator should be an *efficient interface*, as in Definition 2.12.

```
initialization:
    initialize an empty associative array $Map : \mathcal{Y} \times \mathcal{T} \to \mathcal{C}$;
upon receiving (sim-proof-query, $y$):
    $c \xleftarrow{R} \mathcal{C}$, $(t, z) \xleftarrow{R} Sim_1(y, c)$
    if $(y, t) \notin \text{Domain}(Map)$
        then $Map[y, t] \leftarrow c$
            return $(t, z)$
        else return fail;
upon receiving (sim-oracle-query, $(\hat{y}, \hat{t})$):
    if $(\hat{y}, \hat{t}) \notin \text{Domain}(Map)$ then $Map[\hat{y}, \hat{t}] \xleftarrow{R} \mathcal{C}$
    return $Map[\hat{y}, \hat{t}]$
```

Figure 20.2: niZK Simulator for Fiat-Shamir

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to Φ and *Sim* as

$$\text{niZK}\mathsf{adv}[\mathcal{A}, \Phi, Sim] := \big|\Pr[W_0] - \Pr[W_1]\big|. \quad \Box$$

Definition 20.5. *We say Φ provides **non-interactive zero knowledge (niZK) in the random oracle model**, if there exists an efficient simulator Sim for Φ, such that for every efficient adversary $\mathcal{A}$, the value* $\text{niZK}\mathsf{adv}[\mathcal{A}, \Phi, Sim]$ *is negligible.*

We note that in the simulated world in Attack Game 20.3, for the proof queries, the adversary must supply a witness, even though this witness is not passed along to the simulator. Thus, the simulator only needs to generate simulated proofs for true statements. We do not require that the simulator *always* returns a valid proof, but this should happen with overwhelming probability, if Definition 20.5 is to be satisfied.

We next show that the Fiat-Shamir transform always yields niZK, provided the underlying Sigma protocol is special HVZK and has unpredictable commitments (see Definition 19.7).

Theorem 20.3 (Fiat-Shamir-based proofs are zero knowledge). *Let $\Pi = (P, V)$ be a special HVZK Sigma protocol for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$ with unpredictable commitments, and let* FS-Π *be the Fiat-Shamir non-interactive proof system derived from Π with hash function H. If we model H as a random oracle, then* FS-Π *is niZK.*

> *In particular, there exists a simulator Sim such that if $\mathcal{A}$ is an adversary that attacks* FS-Π *and Sim as in Attack Game 20.3, making at most $Q_{\rm p}$ proof queries and at most $Q_{\rm ro}$ random oracle queries, and if Π has δ-unpredictable commitments, then we have*
>
> $$\text{niZK}\mathsf{adv}[\mathcal{A}, \text{FS-}\Pi, Sim] \le Q_{\rm p}(Q_{\rm p} + Q_{\rm ro}) \cdot \delta. \qquad (20.9)$$

Proof sketch. The basic idea is similar to one we already saw in the proof of security of Schnorr's signature scheme in Theorem 19.7. Our niZK simulator is given in Fig. 20.2. Here, we assume that Sim_1 is the simulator guaranteed by the special HVZK property for Π. We leave it to the reader to verify the inequality (20.9) — the argument is very similar to that made in the proof of Theorem 19.7. □

20.3.6 An example: applying the Fiat-Shamir transform to the Chaum-Pedersen protocol

In this section, we work out the details of applying the Fiat-Shamir transform to a particular Sigma protocol, namely, the Chaum-Pedersen protocol from Section 19.5.2.

Recall that the Chaum-Pedersen protocol allows a prover to convince a skeptical verifier that a given triple is a DH-triple. Specifically, it is a Sigma protocol for the relation

$$\mathcal{R} := \Big\{ \big(\beta,\ (u, v, w) \big) \in \mathbb{Z}_q \times \mathbb{G}^3 :\ v = g^\beta \text{ and } w = u^\beta \Big\}.$$

Here, $\mathbb{G}$ is a group of prime order q generated by $g \in \mathbb{G}$. Also, a statement is $(u, v, w) \in \mathbb{G}^3$, and a witness for such a statement is $\beta \in \mathbb{Z}_q$ satsifying $v = g^\beta$ and $w = u^\beta$. The reader should review the details of the Chaum-Pedersen protocol in Fig. 19.7. The challenge space $\mathcal{C}$ is a subset of $\mathbb{Z}_q$. An accepting conversation for a statement $(u, v, w) \in \mathbb{G}^3$ is of the form

$$((v_t, w_t), c, \beta_z) \in \mathbb{G}^2 \times \mathcal{C} \times \mathbb{Z}_q \quad \text{such that} \quad g^{\beta_z} = v_t \cdot v^c \text{ and } u^{\beta_z} = w_t \cdot w^c.$$

To apply the Fiat-Shamir transform, we first need a hash function

$$H : \mathbb{G}^3 \times \mathbb{G}^2 \to \mathcal{C}.$$

The resulting non-interactive proof system $\Phi = (GenPrf, VrfyPrf)$ is defined as follows:

- on input $\big(\beta,\ (u, v, w) \big) \in \mathcal{R}$, $GenPrf$ computes

```
$\beta_t \xleftarrow{R} \mathbb{Z}_q,\ v_t \leftarrow g^{\beta_t},\ w_t \leftarrow u^{\beta_t}$
$c \leftarrow H(\ (u, v, w),\ (v_t, w_t)\ ) \in \mathcal{C}$
$\beta_z \leftarrow \beta_t + \beta c$
output $(\ (v_t, w_t),\ \beta_z\ ) \in \mathbb{G}^2 \times \mathbb{Z}_q$;
```

- on input $\big((u, v, w),\ (\,(v_t, w_t),\ \beta_z\,) \big) \in \mathbb{G}^3 \times (\mathbb{G}^2 \times \mathbb{Z}_q)$, $VrfyPrf$ computes

```
$c \leftarrow H(\ (u, v, w),\ (v_t, w_t)\ )$
if $g^{\beta_z} = v_t \cdot v^c$ and $u^{\beta_z} = w_t \cdot w^c$
    then output accept
    else output reject.
```

Here, the proof is $(\ (v_t, w_t),\ \beta_z\ ) \in \mathbb{G}^2 \times \mathbb{Z}_q$.

Soundness. Assuming $N := |\mathcal{C}|$ is super-poly, and if we model H as a random oracle, then Φ is sound (as in Definition 20.4). In particular, let $\mathcal{A}$ be an adversary attacking the soundness of Φ as in the random oracle version of Attack Game 20.2. Moreover, assume that $\mathcal{A}$ issues at most Q_{ro} random oracle queries. Then

$$\text{niSnd}^{\text{ro}}\mathsf{adv}[\mathcal{A}, \Phi] \leq (Q_{\text{ro}} + 1)/N.$$

This means that the probability that $\mathcal{A}$ can create a valid proof of a false statement is at most $(Q_{\mathrm{ro}}+1)/N$. This follows from Theorems 20.2, 20.1, and 19.10. In Attack Game 20.2, the adversary is allowed to make one attempt at creating a valid proof of a false statement. As you are asked to show in Exercise 20.18, if the adversary is allowed to make $r \geq 1$ attempts at creating a valid proof of a false statement, he will succeed with probability most $(Q_{\mathrm{ro}} + r)/N$.

Zero knowledge. If we model H as a random oracle, then Φ provides non-interactive zero knowledge (as in Definition 20.5). In particular, there exists a simulator *Sim* such that if $\mathcal{A}$ is an adversary that attacks Φ and *Sim* as in Attack Game 20.3, making at most Q_{p} proof queries and at most Q_{ro} random oracle queries, then we have

$$\mathrm{niZK}\mathsf{adv}[\mathcal{A}, \Phi, \mathit{Sim}] \leq Q_{\mathrm{p}}(Q_{\mathrm{p}} + Q_{\mathrm{ro}})/q.$$

This follows from Theorems 20.3 and 19.10, along with the fact that the Chaum-Pedersen protocol has $(1/q)$-unpredictable commitments (as in Definition 19.7).

An optimized version of the proof system. We can optimize this proof system along the same lines that we optimized Schnorr's signatures in Section 19.2.3 to obtain a system with more compact proofs. This optimized non-interactive proof system $\Phi' = (\mathit{GenPrf}', \mathit{VrfyPrf}')$ is defined as follows:

- on input $\big(\ \beta,\ (u, v, w)\ \big) \in \mathcal{R}$, GenPrf' computes

```
$\beta_{\mathrm{t}} \xleftarrow{\mathrm{R}} \mathbb{Z}_q,\ v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{t}}},\ w_{\mathrm{t}} \leftarrow u^{\beta_{\mathrm{t}}}$
$c \leftarrow H(\ (u, v, w),\ (v_{\mathrm{t}}, w_{\mathrm{t}})\ ) \in \mathcal{C}$
$\beta_{\mathrm{z}} \leftarrow \beta_{\mathrm{t}} + \beta c$
output $(c, \beta_{\mathrm{z}}) \in \mathcal{C} \times \mathbb{Z}_q$;
```

- on input $\big(\ (u, v, w),\ (c, \beta_{\mathrm{z}})\ \big) \in \mathbb{G}^3 \times (\mathbb{G}^2 \times \mathbb{Z}_q)$, $\mathit{VrfyPrf}'$ computes

```
$v_{\mathrm{t}} \leftarrow g^{\beta_{\mathrm{z}}}/v^c$
$w_{\mathrm{t}} \leftarrow u^{\beta_{\mathrm{z}}}/w^c$
if $c = H(\ (u, v, w),\ (v_{\mathrm{t}}, w_{\mathrm{t}})\ )$
    then output accept
    else  output reject.
```

The proof now is $(c, \beta_{\mathrm{z}}) \in \mathcal{C} \times \mathbb{Z}_q$. All of the results above on soundness and zero knowledge for Φ are also true for Φ' — see Exercises 20.14 and 20.18(b).

20.4 Computational zero-knowledge and applications

It turns out that for some relations, we need a more relaxed notion of zero knowledge in order to get an efficient Sigma protocol. We will motivate and illustrate the idea with an example.

20.4.1 Example: range proofs

We again use the multiplicative ElGamal encryption scheme that we used in the examples in Section 20.2. Bob's public key is $u = g^\alpha \in \mathbb{G}$ and his secret key is $\alpha \in \mathbb{Z}_q$. As usual, $\mathbb{G}$ is a cyclic group of order q with generator $g \in \mathbb{G}$.

Suppose we generalize Example 20.3, so that instead of encrypting a bit b, Alice encrypts a d-bit number x, so $x \in \{0, \dots, 2^d - 1\}$. To perform the encryption, Alice encodes x as the group element g^x, and then encrypts this group element under Bob's public key. The resulting ciphertext will be of the form (v, e), where $v = g^\beta$ and $e = u^\beta g^x$. We shall assume that $2^d < q$, so that the encoding of x is one-to-one. As usual, Alice wants to convince Charlie that (v, e) does indeed encrypt a d-bit number under Bob's public key, without revealing anything else.

So we want a Sigma protocol for the relation

$$\mathcal{R} = \left\{ \ (\ (\beta, \gamma, x),\ (u, v, e)\ )\ :\ v = g^\beta,\ e = u^\beta \cdot g^x,\ x \in \{0, \dots, 2^d - 1\} \right\}. \tag{20.10}$$

Here, we will assume that d is a fixed, public value.

A straightforward approach is just to use the same OR-proof technology that we used in Example 20.3. Namely, Alice essentially proves that $x = 0$, or $x = 1$, or $\dots$, $x = 2^d - 1$. While this idea works, the communication and computational complexity of the resulting Sigma protocol will be proportional to 2^d. It turns out that we can do much better. Namely, we can construct a Sigma protocol that scales *linearly* in d, rather than *exponentially* in d.

Here is how. Alice starts by writing x in binary, so $x = \sum_i 2^i b_i$, where $b_i \in \{0, 1\}$ for $i = 0, \dots, d - 1$. Next, next Alice encrypts each bit. To get a simpler and more efficient protocol, she uses the variation of the ElGamal encryption scheme discussed in Exercise 11.8. Specifically, Alice generates a random public key $(u_0, \dots, u_{d-1}) \in \mathbb{G}^d$; she then chooses $\beta_0 \in \mathbb{Z}_q$ at random, and computes $v_0 \leftarrow g^{\beta_0}$; finally, she computes $e_i \leftarrow u_i^{\beta_0} g^{b_i}$ for $i = 0, \dots, d-1$. So $(v_0, e_0, \dots, e_{d-1})$ is an encryption of $(b_0, \dots, b_{d-1})$ under the public key $(u_0, \dots, u_{d-1})$. Alice then sends v_0, $(u_0, \dots, u_{d-1})$, and $(e_0, \dots, e_{d-1})$ to Charlie, and proves to him that (i) each encrypted value b_i is a bit, and (ii) $\sum_i 2^i b_i = x$. To prove (i), Alice will use a technique similar to that used in Example 20.5, exploiting the fact that $b_i \in \{0, 1\} \iff b_i^2 = b_i$.

To prove (i) and (ii), Alice and Charlie can use the generic linear protocol from Section 19.5.3. So Alice proves to Charlie that there exist

$$\beta, x, \quad \beta_0, \quad b_0, \dots, b_{d-1}, \quad \tau_0, \dots, \tau_{d-1}$$

such that

$$\left.\begin{aligned} & v = g^\beta, \quad e = u^\beta g^x, \\ & v_0 = g^{\beta_0} \\ & e_i = u_i^{\beta_0} g^{b_i}, \quad v_0^{b_i} = g^{\tau_i}, \quad e_i^{b_i} = u_i^{\tau_i} g^{b_i} \qquad (i = 0, \dots, d-1), \\ & x = b_0 + 2b_1 + \cdots + 2^{d-1} b_{d-1}. \end{aligned}\right\} \tag{20.11}$$

The first line of (20.11) says that (v, e) encrypts g^x under u. The third line says that each encrypted value b_i is a bit, using a variant of the technique in Example 20.5, where $\tau_i = \beta_0 b_i$. The fourth line says that these bits are precisely the bits in the binary representation of x.

So the overall structure of the protocol is as follows:

1. Alice generates v_0, $(u_0, \ldots, u_{d-1})$, and $(e_0, \ldots, e_{d-1})$, and sends these auxiliary group elements to Charlie.

2. Alice and Charlie engage in the generic linear Sigma protocol for the system of equations (20.11).

The first observation we make is that by having Alice "piggyback" the auxiliary group elements on top of the commitment message of the generic linear Sigma protocol, the overall protocol has the basic structure of a Sigma protocol.

We leave it to the reader to verify that the protocol provides soundness.

The question of interest to us here is: in what sense is this protocol zero knowledge? The problem is that while the generic linear protocol is special HVZK, the overall protocol is not, in the sense that the encryptions of the bits of x could conceivably leak information about x to Charlie. Intuitively, under the DDH assumption, these encryptions should not leak any information. So the protocol is still zero knowledge, but only in a *computational* sense. To put this on firmer ground, we need to formulate the notion of special *computational* HVZK.

20.4.2 Special computational HVZK

We relax Definition 19.5, which defines the notion of special HVZK for a Sigma protocol, to obtain the weaker notion of special *computational HVZK*, or special *cHVZK*, for short. The idea is that instead of requiring that the distributions of the real and simulated definitions be *identical*, we only require them to be *computationally indistinguishable.*

Let $\Pi = (P, V)$ be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, with challenge space $\mathcal{C}$. As in Definition 19.5, a *simulator for* Π is an efficient probabilistic algorithm *Sim* that takes as input $(y, c) \in \mathcal{Y} \times \mathcal{C}$, and always outputs a pair (t, z) such that (t, c, z) is an accepting conversation for y.

Attack Game 20.4 (Special cHVZK). Let $\Pi = (P, V)$ be a Sigma protocol for $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$, with challenge space $\mathcal{C}$. Let *Sim* be a simulator for Π, as above. For a given adversary $\mathcal{A}$, we define two experiments, Experiment 0 and Experiment 1. In both experiments, $\mathcal{A}$ starts out by computing $(x, y) \in \mathcal{R}$ and submitting (x, y) to the challenger.

- In Experiment 0, the challenger runs the protocol between $P(x, y)$ and $V(y)$, and gives the resulting conversation (t, c, z) to $\mathcal{A}$.

- In Experiment 1, the challenger computes

$$c \xleftarrow{\text{R}} \mathcal{C}, \ \ (t, z) \xleftarrow{\text{R}} \mathit{Sim}(y, c),$$

  and gives the simulated conversation (t, c, z) to $\mathcal{A}$.

At the end of the game, $\mathcal{A}$ computes and outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. We define $\mathcal{A}$'s **advantage** with respect to Π and *Sim* as

$$\text{cHVZK}\mathsf{adv}[\mathcal{A}, \Pi, \mathit{Sim}] := \big|\Pr[W_0] - \Pr[W_1]\big|. \quad \square$$

Definition 20.6. *We say* Π *is **special computational HVZK**, or **special cHVZK**, if there exists a simulator Sim for* Π*, such that for every efficient adversary* $\mathcal{A}$*, the value* $\text{cHVZK}\mathsf{adv}[\mathcal{A}, \Pi, \mathit{Sim}]$ *is negligible.*

Many results that hold for special HVZK Sigma protocols also hold for special cHVZK Sigma protocols:

- Theorem 19.15 also holds if we use a cHVZK protocol instead of an HVZK protocol, although the concrete security bound becomes

$$\text{ID2}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq \text{ID1}\mathsf{adv}[\mathcal{B}, \mathcal{I}] + Q \cdot \text{cHVZK}\mathsf{adv}[\mathcal{B}', \Pi, \mathit{Sim}],$$

  where Q is an upper bound on the number of transcripts obtained in the eavesdropping attack. This factor of Q arises from applying a standard hybrid argument, which allows us to replace Q real conversations by Q simulated conversations.

- Lemma 19.17 can also be adapted to work with a cHVZK protocol, instead of an HVZK protocol. The security bound (19.20) becomes

$$\epsilon \leq \frac{r}{N} + \sqrt{r\epsilon'} + Q \cdot \text{cHVZK}\mathsf{adv}[\mathcal{B}', \Pi, \mathit{Sim}],$$

  where, again, Q is an upper bound on the number of transcripts obtained in the eavesdropping attack.

- Theorem 20.3 also holds if we use a cHVZK protocol instead of an HVZK protocol. Again, the concrete security bound degrades with an extra additive term of $Q_{\text{p}} \cdot \text{cHVZK}\mathsf{adv}[\mathcal{B}, \Pi, \mathit{Sim}_1]$, where Q_{p} is the number of proof queries.

We remark, however, that Theorem 19.21 (on witness independence) *does not* carry over under cHVZK.

Range proofs. We leave it as a simple exercise to the reader to prove that our protocol in Section 20.4.1 for proving that an encrypted value lies in the range $[0, 2^d)$ is special cHVZK.

20.4.3 An unconstrained generic protocol for non-linear relations

The technique used in Section 20.4.1 can be generalized, allowing us to add non-linear relations of the form $x_i = x_j \cdot x_k$ to the systems of linear equations handled by the generic linear protocol, as we did in Section 20.2.1. However, unlike in Section 20.2.1, we do not require any auxiliary equations. The price we pay for this generality is that we achieve only special cHVZK, rather than HVZK.

Again, let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$, and let ϕ be a formula as in (19.13), but with non-linear equations of the form $x_i = x_j \cdot x_k$ as well. Suppose the prover and verifier are both given ϕ, and the prover is also given an assignment $(\alpha_1, \ldots, \alpha_n)$ to the variables $(x_1, \ldots, x_n)$ that satisfies ϕ. The prover generates a new formula ϕ', as follows. The prover chooses $\beta \in \mathbb{Z}_q$ at random, sets $v \leftarrow g^\beta$, and adds the equation $v = g^y$ to ϕ, where y is a new variable. Then, for each non-linear equation $x_i = x_j \cdot x_k$ in ϕ, the prover chooses $u \in \mathbb{G}$ at random and computes $e \leftarrow u^\beta g^{\alpha_j}$, and adds the equations

$$e = u^y g^{x_j}, \quad v^{x_k} = g^t, \quad \text{and} \quad e^{x_k} = u^t g^{x_i} \tag{20.12}$$

to ϕ, where t is another new variable. This results in a new formula ϕ' that can be handled by the generic linear protocol. The prover then sends to the verifier the collection of auxiliary group elements, consisting of v, along with the group elements u and e corresponding to each non-linear equation.

Given these auxiliary group elements, the verifier can reconstruct the formula ϕ', and now both prover and verifier can run the generic linear protocol on ϕ'. The prover assigns the value β to the variable y, and the value $\tau := \beta\alpha_k$ to the variable t arising from each non-linear equation $x_i = x_j \cdot x_k$. Also, the prover can "piggy-back" the auxiliary group elements on top of the commitment message from the generic linear protocol, so that the resulting protocol has the right communication pattern.

We leave it to the reader to verify that this transformation yields a Sigma protocol that is special cHVZK (under the DDH assumption, using Exercise 11.8) and provides special soundness for the relation (19.14), where the formulas ϕ are now allowed to have the non-linear form described above.

There are a couple of obvious opportunities for efficiency improvements to the above transformation. For example, the value u and the first equation in (20.12) can be reused across all non-linear equations in which x_j appears as the first multiplicand. Similarly, the variable t and the second equation in (20.12) can be reused across all non-linear equations in which x_k appears as the second multiplicand.

Range proofs, again. It is easy to see that our range proof protocol can be derived using this transformation. Alice proves to Charlie that there exist

$$\beta, x, \quad , b_0, \ldots, b_{d-1}$$

such that

$$v = g^{\beta}, \quad e = u^{\beta} g^{x}, \quad x = \sum_{i=0}^{d-1} 2^i b_i, \quad \text{and} \quad b_i = b_i^2 \quad (i = 0, \ldots, d-1).$$

We leave it to the reader to verify that applying the above non-linear to linear transformation yields precisely the protocol in Section 20.4.1 (with the values v_0 and β_0 playing the roles of v and β in the transformation).

20.5 Bulletproofs: compressed Sigma protocols

To be written.

20.6 Succinct non-interactive zero-knowledge proofs (SNARKs)

To be written.

20.7 A fun application: everything that can be proved, can be proved in zero knowledge

To be written.

20.8 Notes

Citations to the literature to be added.

20.9 Exercises

20.1 (Generalized encrypted bits). Use the generalized OR-proof construction from Exercise 19.26 to generalize the encrypted bits protocol from Example 20.3 to give a sound, special HVZK Sigma protocol for proving that a ciphertext (v, e) encrypts a value b (encoded as g^b) satisfying $0 \leq b < B$ for some constant $B > 2$. Write out the protocol for $B = 3$.

The following two exercises ask you to design sound, HVZK Sigma protocols for proving properties on encrypted data, as in Section 20.2, using the multiplicative ElGamal encryption scheme. You should not *use the techniques introduced in Section 20.4, which yield only computational HVZK protocols. You may, however, use the techniques in Section 20.2.1.*

20.2 (A 2-input mixnet). Consider the following generalization of the scenario discussed in Example 20.2. Here, Alice is implementing a 2-input mixnet service, which can be used to help to foil traffic analysis. In this setting, Alice receives two ciphertexts (v_0, e_0) and (v_1, e_1), which encrypt messages under Bob's public key u. Alice does not know these messages, but she can re-randomize the ciphertexts, choosing β_0 and β_1 in $\mathbb{Z}_q$ at random, and computing $(v'_i, e'_i) := (v_i \cdot g^{\beta_i}, e_i \cdot u^{\beta})$ for $i = 0, 1$. Further, she chooses $b \in \{0, 1\}$ at random and sets $(v''_i, e''_i) := (v'_{i \oplus b}, e'_{i \oplus b})$ for $i = 0, 1$. Finally, she outputs the pair of ciphertexts (v''_0, e''_0) and (v''_1, e''_1). Thus, Alice re-randomizes the two ciphertexts, and with probability $1/2$ she flips their order.

Design a sound, special HVZK Sigma protocol that allows Alice to prove to Charlie that she has performed this task correctly. That is, she should prove that the output ciphertexts encrypt the same messages as the input ciphertexts, but with the ordering of the ciphertexts possibly flipped. The statement for the Sigma protocol should include Bob's public key, Alice's two input ciphertexts, and Alice's two output ciphertexts.

20.3 (Encrypted polynomial relations). Consider again the task in Example 20.6, where Alice encrypts g^{γ} and $g^{\gamma'}$ under Bob's public key, and wants to prove to Charlie that $\gamma' = f(\gamma)$ for some polynomial $f(x) = \sum_{i=0}^{d} \lambda_i x^i$. However, suppose now that the coefficients of f are also encrypted under Bob's public key. That is, each coefficient λ_i is encrypted as $(v_i, e_i) = (g^{\beta_i}, h^{\beta_i} g^{\lambda_i})$, and these $d + 1$ ciphertexts are included in the statement, along with the ciphertexts (v, e) and (v', e') encrypting g^{γ} and $g^{\gamma'}$.

Design a sound, special HVZK Sigma protocol for this problem. The complexity (computational and communication) of your protocol should grow linearly in d.

20.4 (Computational special soundness implies soundness). Prove the following generalization of Theorem 20.1. Suppose Π is a Signma protocol that provides computational special soundness (as defined in Exercise 19.13) with extractor Ext, and that Π has a large challenge space of size N. Then Π is sound.

In particular, suppose $\mathcal{A}$ is an adversary attacking the soundness of Π as in Attack Game 20.1, with advantage $\epsilon := \text{Snd}\mathsf{adv}[\mathcal{A}, \Pi]$. Then there exists an efficient adversary $\mathcal{B}$ (whose running time is about *twice* that of $\mathcal{A}$), such that $\text{cSS}\mathsf{adv}[\mathcal{B}, \Pi, Ext] \geq \epsilon^2 - \epsilon/N$.

20.5 (Zero knowledge range proofs). Consider again the range proof problem introduced in Section 20.4.1, where Alice wants to prove to Charlie that she has encrypted a d-bit integer under Bob's public key. The Sigma protocol we presented there provides *statistical soundness* (see Remark 20.1), but only *computational zero knowledge* (special cHVZK). This exercise develops an alternative Sigma protocol for the relation $\mathcal{R}$ defined in (20.10). This new Sigma protocol is *zero knowledge* (special HVZK), but provides only *computational soundness* under an intractibility assumption.

Suppose that we have a system parameter $h \in \mathbb{G}$. We assume that h is uniformly distributed over $\mathbb{G}$, and that nobody knows $\mathsf{Dlog}_g h$ (especially Alice). The protocol is the same as that in Section 20.4.1, except that instead of encrypting each bit b_i, Alice just "commits" to it, by computing $\beta_i \xleftarrow{\text{R}} \mathbb{Z}_q$ and $u_i \leftarrow g^{\beta_i} h^{b_i}$. In the protocol, Alice sends $u_0, \ldots, u_{d-1}$ to Charlie, and proves that she knows

$$\beta, x, \quad \beta_0, \ldots, \beta_{d-1}, \quad b_0, \ldots, b_{d-1}, \quad \tau_0, \ldots, \tau_{d-1}$$

such that

$$\begin{aligned}
&v = g^{\beta}, \quad e = u^{\beta} g^{x}, \\
&u_i = g^{\beta_i} h^{b_i}, \quad u_i^{b_i} = g^{\tau_i} h^{b_i} \qquad (i = 0, \ldots, d-1), \\
&x = b_0 + 2b_1 + \cdots + 2^{d-1} b_{d-1}
\end{aligned}$$

using the generic linear protocol. To run the protocol, Alice sets $\tau_i := \beta_i b_i$.

Show that this is a Sigma protocol for $\mathcal{R}$ that is *special HVZK* and that provides *computational special soundness* (defined in Exercise 19.13) under the DL assumption for $\mathbb{G}$. To prove computational special soundness, you should make use of the fact that the generic linear protocol itself provides special soundness.

Hint: If you break computational special soundness, you can compute $\mathsf{Dlog}_g h$.

20.6 (Zero knowledge protocols for non-linear relations). Design and analyze a construction for non-linear relations as in Section 20.4.3. However, the resulting protocol should be a Sigma protocol that is special HVZK and provides computational special soundness (defined in Exercise 19.13) under the DL assumption.

Hint: Generalize the technique in the previous exercise.

The following four *exercises ask you to design sound, computational HVZK Sigma protocols for proving properties on encrypted data, using the techniques developed in Section 20.4. These protocols are actually statistically sound (see Remark 20.1). Alternatively, instead of computational zero knowledge, you may apply the techniques developed in Exercise 20.6 to achieve zero knowledge, but resulting protocols are only computationally sound.*

20.7 (Generalized range proofs). Generalize the protocol in Section 20.4.1, so that instead of proving that $x \in \{0, \ldots, 2^d - 1\}$, Alice proves to Charlie that $x \in [a, b]$ for arbitrary integers a and b. For this exercise, you can assume that a and b are fixed, public values. Your protocol should have complexity proportional to $\log(b - a)$, and should be a sound, special cHVZK Sigma protocol for this problem.

20.8 (Encrypted range proofs). Generalize the previous problem, so that now, the values g^a and g^b are encrypted under Bob's public key. You may assume that $b - a < 2^d$ for some fixed, public value d.

20.9 (High-degree relations). Consider the following variation on Example 20.6. Instead of proving to Charlie that $\gamma' = f(\gamma)$, Alice proves that $\gamma' = \gamma^k$, for some specific, large, positive integer k. Assume that k is a fixed, public value. Your protocol should have complexity proportional to $\log k$, and should be a sound, special cHVZK Sigma protocol for this problem.

20.10 (Encrypted high-degree relations). Generalize the previous problem, so that now, the value g^k is encrypted under Bob's public key. You may assume that $k < 2^d$ for some fixed, public value d.

20.11 (Encrypting a discrete logarithm). Suppose Alice wants to encrypt a discrete logarithm under Bob's public key, and prove to Charlie that she has done so. Again, we are assuming that we are using the multiplicative ElGamal encryption scheme, as in Section 20.2. So Alice knows $\gamma \in \mathbb{Z}_q$ such that $h = g^\gamma \in \mathbb{G}$. She is willing to make the value h public, and wants to somehow encrypt γ under Bob's public key $u \in \mathbb{G}$, and prove to Charlie that she has done so.

One approach is the following. Alice can encrypt the bits of γ one at a time, resulting in a ciphertext containing $O(\log q)$ group elements. She can then run a Sigma protocol to convince Charlie that these bits form the binary representation of γ. Work out the details of this approach.

20.12 (Encrypting a signature). We can use the result of the previous exercise to allow Alice to verifiably encrypt a signature. In this setting, Alice has a signature on a message m under Bill's public key. Assume that Bill is using Schnorr's signature scheme with public key $u_0 \in \mathbb{G}$. So a signature on m is of the form $(u_{\mathrm{t}}, \alpha_{\mathrm{z}})$, where $g^{\alpha_{\mathrm{z}}} = u_{\mathrm{t}} \cdot u_0^{c}$ and $c = H(m, u_{\mathrm{t}})$. Suppose that Alice presents to Charlie the values m, u_{t}, and an encryption ψ of α_{z} under Bob's public key, as in the previous exercise. Suppose she also presents to Charlie a non-interactive proof π that the ciphertext ψ indeed encrypts $\mathsf{Dlog}_g(u_{\mathrm{t}} \cdot u_0^{c})$. The proof she presents is the Fiat-Shamir proof (see Section 20.3.3) derived from the Sigma protocol of the previous exercise.

(a) Work out the details of this approach.

(b) Using the soundness property of the Fiat-Shamir non-interactive proof system, argue that after seeing the values $m, u_{\mathrm{t}}, \psi, \pi$, and verifying that π is a valid proof, Charlie can be assured that ψ decrypts to a value from which a valid signature on m can be recovered.

(c) Using the zero-knowledge property of the Fiat-Shamir non-interactive proof system, argue that after seeing the values $m, u_{\mathrm{t}}, \psi, \pi$, Charlie cannot forge a signature on m under Bill's public key. Formulate this problem as an attack game, and prove that if Charlie can win this game, he can break the DDH assumption.

20.13 (Broken Fiat-Shamir proofs). In Section 20.3.3, we showed how to turn a Sigma protocol into a non-interactive proof system by computing the challenge as $c := H(y, t)$, where y is the statement and t is the commitment. The point of this exercise is to illustrate that the statement y must be included in the hash to maintain soundness. To this end, suppose that we transform the Chaum-Pedersen protocol (see Section 19.5.2) into a non-interactive proof by deriving the challenge from the hash of the commitment only, the resulting non-interactive proof system is not sound.

20.14 (Optimized Fiat-Shamir proofs). We can optimize Fiat-Shamir non-interactive proof systems (see Section 20.3.3) just as we did Fiat-Shamir signatures in Exercise 19.19. Consider the Fiat-Shamir proof system scheme derived from a Sigma protocol (P, V) for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$.

Recall that a proof π for a statement y is (t, z), where $(t, c, z) \in \mathcal{T} \times \mathcal{C} \times \mathcal{Z}$ is an accepting conversation, and $c := H(y, t)$.

Assume that (P, V) has backward computable commitments, as in Exercise 19.18, and let $f : \mathcal{Y} \times \mathcal{C} \times \mathcal{Z} \to \mathcal{T}$ be the corresponding function that computes a commitment from a given statement, challenge, and response. Then we can optimize the Fiat-Shamir proof system, so that instead of using (t, z) as the proof, we use (c, z) as the proof. To verify such an optimized proof (c, z), we compute $t \leftarrow f(c, z)$, and verify that $c = H(y, t)$.

(a) Show that Theorem 20.2 holds for the optimized Fiat-Shamir proof system.

(b) We can modify the niZK simulator in Fig. 20.2, so that in processing proof query y_i, we return (c_i, z_i), instead of (t_i, z_i). Show that Theorem 20.3 holds for the optimized Fiat-Shamir proof system, using the modified simulator.

20.15 (Verifiable decryption). In Section 20.3.1, we described a voting protocol, which required the Vote Tallying Center (VTC) to decrypt a ciphertext (v_*, e_*) and publish the result. Design a Sigma protocol that allows the VTC to prove that it performed the decryption correctly. Then covert the Sigma protocol to a corresponding non-interactive proof system using the optimized Fiat-Shamir transform from the previous exercise.

20.16 (A verifiable random function). The notion of a verifiable random function (VRF) was introduced in Exercise 13.20. This exercise develops an instantiation of this notion — actually, as we will see, it satisfies a slightly weaker property, which is still good enough for most applications.

Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let Π be the Chaum-Pedersen protocol, as discussed in Section 19.5.2, for the relation (19.12), and assume that Π has a large challenge space $\mathcal{C}$. Let Φ be the optimized Fiat-Shamir proof system derived from Π, as in Exercise 20.14, using a hash function $H' : \mathbb{G}^3 \times \mathbb{G}^2 \to \mathcal{C}$.

Let F be the PRF defined over $(\mathbb{Z}_q, \mathcal{M}, \mathbb{G})$ as in Exercise 11.2, so $F(k, m) := H(m)^k$, where $H : \mathcal{M} \to \mathbb{G}$ is a hash function. You were asked to show in that exercise that if we model H as a random oracle, then F is a PRF under the DDH (with a very tight reduction, in fact).

Our VRF is (G', F', V'), which is defined over $(\mathcal{M}, \mathbb{G})$; G' chooses $k \in \mathbb{Z}_q$ at random, k is the secret key, and g^k is the public key; $F'(k, m) := (y, \pi)$, where $y = F(k, m) = H(m)^k$ and π is a proof, generated using Φ, that $(H(m), g^k, H(m)^k)$ is a DH-triple; $V'(g^k, m, y, \pi)$ checks that $(H(m), g^k, y)$ is a DH-triple by verifying the proof π using Φ.

(a) Describe the functions F' and V' in detail.

(b) Using the zero knowledge property for Φ (in particular, Theorem 20.3 and part (b) of Exercise 20.14), show that if we model both H and H' as random oracles, then under the DDH assumption for $\mathbb{G}$, the VRF (G', F', V') satisfies the VRF security property defined in Exercise 13.20. Give a concrete security bound (which should be fairly tight).

(c) This VRF does not satisfy the uniqueness property defined in Exercise 13.20. Nevertheless, it does satisfy a weaker, but still useful property. Using the soundness property for Φ (in particular, Theorem 20.2 and part (a) of Exercise 20.14), show that it is infeasible for an adversary to come up with a triple (m, y, π) such that $V'(g^k, m, y, \pi) = \mathsf{accept}$ yet $y \neq F(k, m)$. Give a concrete security bound.

20.17 (Signatures schemes based on DDH and CDH). In the previous exercise, we saw how to construct a "quasi-VRF" (G', F', V') based on the DDH. We can build a signature scheme $\mathcal{S}$ quite easily from this. The key generation algorithm for $\mathcal{S}$ is G', a signature on a message m under secret key k is $F'(k, m) = (y, \pi)$, and the verification algorithm on a public key g^k, message m, and signature (y, π) simply runs $V'(g^k, m, y, \pi)$.

(a) Using the results of the previous exercise, show that $\mathcal{S}$ is secure under the DDH assumption in the random oracle model. Give a concrete security bound.

Discussion: We will see a simpler signature scheme based on the DDH below in Exercise 20.23.

(b) Prove that $\mathcal{S}$ is secure in the random oracle model under the *CDH assumption*. Give a concrete security bound. Can you use the ideas in the proof of Lemma 13.6 to get a better security bound?

(c) Can you use the ideas in Section 13.5 to modify $\mathcal{S}$ slightly so as to get a signature scheme with a much tighter reduction to CDH in the random oracle model?

20.18 (Multi-attempt Fiat-Shamir soundness). We can generalize Attack Game 20.2, allowing the adversary to output many attempts $(y_1^*, \pi_1^*), \ldots, (y_r^*, \pi_r^*)$, winning the game if $\mathit{VrfyPrf}(y_j^*, \pi_j^*) = \mathsf{accept}$ but $y_j^* \notin L_{\mathcal{R}}$ for some $j = 1, \ldots, r$. For such a r-attempt adversary $\mathcal{A}$, we define its advantage $\mathrm{rniSnd}\mathsf{adv}[\mathcal{A}, \Phi, r]$ to be the probability that $\mathcal{A}$ wins the game.

(a) Let Φ be an non-interactive proof system. Show that for every r-attempt adversary $\mathcal{A}$ attacking Φ as above, there exists an adversary $\mathcal{B}$ attacking Φ as in Attack Game 20.2, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that $\mathrm{rniSnd}\mathsf{adv}[\mathcal{A}, \Phi, r] \leq r \cdot \mathrm{niSnd}\mathsf{adv}[\mathcal{B}, \Phi]$.

(b) Let Φ be the non-interactive proof derived using the Fiat-Shamir transform from a Sigma protocol Π. Show that in the random oracle model of the above r-attempt attack game, if $\mathcal{A}$ makes at most Q_{ro} random oracle queries, then there exists an adversary $\mathcal{B}$ attacking Π as in Attack Game 20.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that $\mathrm{rniSnd}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \Phi, r] \leq (r + Q_{\mathrm{ro}})\mathrm{Snd}\mathsf{adv}[\mathcal{B}, \Pi]$.

Discussion: This reduction is much more efficient than applying the reduction in part (a) and then the reduction in Theorem 20.2. Note that if Π provides special soundness and the has a challenge space of size N, then this result, together with Theorem 20.1, implies that $\mathrm{rniSnd}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, \Phi, r] \leq (r + Q_{\mathrm{ro}})/N$.

(c) Show that the result of part (b) holds as well for optimized Fiat-Shamir proofs (see Exercise 20.14).

20.19 (An AD-only CCA secure scheme (II)). The AD-only CCA secure encryption scheme discussed in Exercise 12.20 relied on an algorithm $\mathcal{O}_{\mathrm{DDH}}$ for recognizing DH-triples. In this exercise, we shall see how we can replace such an algorithm by a non-interactive proof system. To this end, let $\Phi = (\mathit{GenPrf}, \mathit{VrfyPrf})$ be a non-interactive proof system for the relation

$$\mathcal{R} = \{ \ (\ \beta, \ (u, v, w)) \ ) \ \in \ \mathbb{Z}_q \times \mathbb{G}^3 \ : \ v = g^\beta, \ w = u^\beta \ \};$$

We presented the details of such a proof system in Section 20.3.6 based on the application of Fiat-Shamir transform to the Chaum-Pedersen protocol (see Section 19.5.2). As discussed in Section 20.3.6, this proof system (as well as the optimized version) provides both soundness and zero knowledge.

The encryption scheme $\mathcal{E}_{\text{GS}}^{\dagger} = (G^{\dagger}, E^{\dagger}, D^{\dagger})$ defined as follows:

- the key generation algorithm runs as follows:

```
$G^{\dagger}() :=$    $\alpha \in \mathbb{Z}_q,\ u \leftarrow g^{\alpha}$
              output $(pk, sk) \leftarrow (u, \alpha)$
```

- the encryption algorithm runs as follows:

```
$E^{\dagger}(pk := u, m, d) :=$    $\beta \xleftarrow{R} \mathbb{Z}_q,\ \ v \leftarrow g^{\beta},\ \ w \leftarrow u^{\beta},\ \ k \leftarrow H(v, w),\ \ c \xleftarrow{R} E_{\text{s}}(k, m),$
                          $\underline{u} \leftarrow H_{\mathbb{G}}(v, d),\ \ \underline{w} \leftarrow \underline{u}^{\beta},\ \ \pi \xleftarrow{R} \mathit{GenPrf}(\ \beta,\ (\underline{u}, v, \underline{w})\ )$
                          output $(v, c, \underline{w}, \pi)$
```

- the decryption algorithm runs as follows:

```
$D(sk := \alpha,\ (v, c, \underline{w}, \pi),\ d) :=$    $\underline{u} := H_{\mathbb{G}}(v, d)$
                                  if $\mathit{VrfyPrf}(\ (\underline{u}, v, \underline{w}),\ \pi\ ) = \mathsf{reject}$
                                      then output $\mathsf{reject}$
                                      else $w \leftarrow v^{\alpha},\ \ k \leftarrow H(v, w),\ \ m \leftarrow D_{\text{s}}(k, c)$
                                          output $m$
```

Just as in Exercise 12.20, your task is to prove that $\mathcal{E}_{\text{GS}}^{\dagger}$ is AD-only CCA secure in the stronger sense described there (where the adversary gets $\hat{w} := \hat{v}^{\alpha}$ rather than $\hat{m}$ in processing decryption queries). In your proof, you should assume that the CDH assumption holds for $\mathbb{G}$, that $\mathcal{E}_{\text{s}}$ is semantically secure, and that Φ provides both soundness (as in Definition 20.4) and zero knowledge (as in Definition 20.5). In particular, show that the advantage of any adversary $\mathcal{A}$ in this strengthened AD-only CCA attack game is bounded by

$$2Q_{\text{Hk}} \cdot \text{CDH}\mathsf{adv}[\mathcal{B}_{\text{cdh}}, \mathbb{G}] + 2 \cdot \text{niZK}\mathsf{adv}[\mathcal{B}_{\text{zk}}, \Phi, \mathit{Sim}] + \\ 2 \cdot \text{rniSnd}\mathsf{adv}[\mathcal{B}_{\text{es}}, \Phi, Q_d] + Q_{\text{e}} \cdot \text{SS}\mathsf{adv}[\mathcal{B}_{\text{s}}, \mathcal{E}_{\text{s}}],$$

where Q_{Hk} is a bound on the number of $H_{\mathcal{K}}$-queries made by $\mathcal{A}$, Q_{e} is a bound on the number of encryption queries made by $\mathcal{A}$, Q_{d} is a bound on the number of decryption queries made by $\mathcal{A}$, $\mathcal{B}_{\text{cdh}}$, $\mathcal{B}_{\text{zk}}$, $\mathcal{B}_{\text{es}}$, and $\mathcal{B}_{\text{s}}$ are elementary wrappers around $\mathcal{A}$, Sim is a zero knowledge simulator (as in Definition 20.5), and rniSndadv is as defined in Exercise 20.18.

Hint: Your proof should follow the same outline as that provided in Exercise 12.20, except that you will now rely on the soundness and zero knowledge properties of Φ, rather than on $\mathcal{O}_{\text{DDH}}$. Note that in Exercise 12.20, we relied on $\mathcal{O}_{\text{DDH}}$ to get a tight reduction to CDH, but here, our adversary $\mathcal{B}_{\text{cdh}}$ will simply prepare a list of at most Q_{Hk} group elements, one of which should be a solution to the given instance of the CDH problem, and then simply guess which one is correct — this is where the factor Q_{Hk} comes from.

Discussion: We remind the reader that although the above scheme only provides AD-only CCA security, it is easily transformed into an efficient scheme that provides full CCA security, as discussed in Exercise 14.13.

20.20 (An AD-only CCA secure scheme (III)). This exercise develops a multiplicative version of the encryption scheme in the previous exercise. The encryption scheme $\mathcal{E}_{\text{GS}}^{\ddagger} = (G^{\ddagger}, E^{\ddagger}, D^{\ddagger})$ with message space $\mathbb{G}$ is defined as follows:

- the key generation algorithm runs as follows:

```
$G^{\ddagger}() :=$    $\alpha \in \mathbb{Z}_q,\ u \leftarrow g^{\alpha}$
                       output $(pk, sk) \leftarrow (u, \alpha)$
```

- the encryption algorithm runs as follows:

```
$E^{\ddagger}(pk := u, m, d) :=$    $\beta \overset{\text{R}}{\leftarrow} \mathbb{Z}_q,\ \ v \leftarrow g^{\beta},\ \ w \leftarrow u^{\beta},\ \ e \leftarrow w \cdot m,$
                                    $\underline{u} \leftarrow H_{\mathbb{G}}(v, d),\ \ \underline{w} \leftarrow \underline{u}^{\beta},\ \ \pi \overset{\text{R}}{\leftarrow} \mathit{GenPrf}(\ \beta,\ (\underline{u}, v, \underline{w})\ )$
                                    output $(v, e, \underline{w}, \pi)$
```

- the decryption algorithm runs as follows:

```
$D(sk := \alpha,\ (v, e, \underline{w}, \pi),\ d) :=$    $\underline{u} := H_{\mathbb{G}}(v, d)$
                                                      if $\mathit{VrfyPrf}(\ (\underline{u}, v, \underline{w}),\ \pi\ ) = \mathsf{reject}$
                                                          then output $\mathsf{reject}$
                                                          else $w \leftarrow v^{\alpha},\ \ m \leftarrow e/w$
                                                              output $m$
```

Prove that $\mathcal{E}_{\text{GS}}^{\ddagger}$ satisfies the same security property as in the previous exercise, but under the DDH assumption rather than the CDH assumption, and specifically, with the following concrete security bound:

$$2 \cdot \Big(\ \text{DDH}\mathsf{adv}[\mathcal{B}_{\text{ddh}}, \mathbb{G}] + 1/q + \text{niZK}\mathsf{adv}[\mathcal{B}_{\text{zk}}, \Phi, \mathit{Sim}] + \text{rniSnd}\mathsf{adv}[\mathcal{B}_{\text{es}}, \Phi, Q_d]\ \Big),$$

which is independent of the number of encryption queries.

Hint: Apply the random self-reduction for DDH from Exercise 10.11 with $n = 1$.

Discussion: Again, this scheme may be easily transformed into an efficient scheme that provides full CCA security, as discussed in Exercise 14.13.

Because of the multiplicative structure of the encryption, it is easy to augment such a ciphertext, as in the voting protocol in Section 20.3.1, with a proof that the ciphertext encrypts a valid vote. In fact, as an optimzation, one could fold this proof into the proof π that is already a part of the ciphertext. To that end, one would replace the relation $\mathcal{R}$ above by the relation

$$\mathcal{R}' = \{\ (\ (\beta, b)\ (\underline{u}, v, \underline{w}, u, e)\ )\ :\ v = g^{\beta},\ \underline{w} = \underline{u}^{\beta},\ e = u^{\beta} g^{b},\ b \in \{0, 1\}\ \}.$$

In addition, the associated data can be used to associate an encrypted vote with a particular voter ID. CCA security ensures these encrypted votes are *non-malleable* (see Section 12.2.1), which means (among other things) that a malicious voter cannot copy or negate the vote of any other voter. This scheme also can be easily implemented as a threshold decryption scheme (see Section 22.3). By using threshold decryption, the VTC in the voting protocol can also be distributed among a number of servers, so that the voting protocol will still work correctly and securely even if some of these servers are compromised.

Also because of the multiplicative structure of the encryption, one can apply the ideas from Exercise 20.11 to this scheme to obtain an AD-only CCA or CCA secure encryption scheme that verifiably encrypts the discrete logarithm of a given group element. The construction in the following exercise may also be used to obtain a more efficient scheme.

20.21 (An AD-only CCA secure scheme (IV)). This exercise generalizes the scheme in the previous exercise so that instead of encrypting a single message $m \in \mathbb{G}$, it encrypts n messages $(m_1, \ldots, m_n)$, along same lines as in Exercise 11.8. In this multi-message scheme, the secret key is $(\alpha_1, \ldots, \alpha_n)$ and the public key is $(u_1, \ldots, u_n)$, where each α_i is a random element of $\mathbb{Z}_q$, and $u_i := g^{\alpha_i}$ for $i = 1, \ldots, n$. Encryption generates a ciphertext $(v, e_1, \ldots, e_n, \underline{w}, \pi)$, where β, v, $\underline{u}$, $\underline{w}$, and π are generated just as before, and $e_i \leftarrow u_i^{\beta} \cdot m_i$ for $i = 1, \ldots, n$. Decryption checks that the proof π is valid, and then computes $m_i \leftarrow e_i / v^{\alpha_i}$ for $i = 1, \ldots, n$. Note that the same ciphertext elements v, $\underline{w}$, and π are used to encrypt all n message elements $m_1, \ldots, m_n$. Prove that this multi-message scheme satisfies the same security property as in the previous exercise with the following concrete security bound:

$$2n \cdot \big(\ \mathrm{DDH}\mathsf{adv}[\mathcal{B}_{\mathrm{ddh}}, \mathbb{G}] + 1/q + \mathrm{niZK}\mathsf{adv}[\mathcal{B}_{\mathrm{zk}}, \Phi, \mathit{Sim}] + \mathrm{rniSnd}\mathsf{adv}[\mathcal{B}_{\mathrm{es}}, \Phi, Q_d]\ \big).$$

Hint: Apply a hybrid argument.

Discussion: By exploiting *simulation soundness* (see Exercise 20.22 below), the term $2n \cdot \mathrm{niZK}\mathsf{adv}[\mathcal{B}_{\mathrm{zk}}, \Phi, \mathit{Sim}]$ in the above security may be replaced by $2 \cdot \mathrm{niZK}\mathsf{adv}[\mathcal{B}_{\mathrm{zk}}, \Phi, \mathit{Sim}]$. The interested reader may wish to work out the details of this.

20.22 (Simulation soundness). This exercise develops a security notion for non-interactive proof systems (see Section 20.3) that combines the notions of soundness (see Section 20.3.4) and zero knowledge (see Section 20.3.5) in a way that is perhaps a bit unintuitive, but that has a number of useful applications, some of which will be developed in subsequent exercises. Roughly speaking, *simulation soundness* means that after seeing simulated proofs of both true and false statements, it should be hard to come up with a new valid proof of a false statement.

Let Φ be a non-interactive proof system for a relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{Y}$. Suppose that Φ makes use of a hash function $H : \mathcal{U} \to \mathcal{C}$, which we model as a random oracle. Consider a simulator Sim for Φ, as defined in Section 20.3.5, which is an interactive machine that responds to queries of the form $(\textsf{sim-proof-query}, y)$, where $y \in \mathcal{Y}$, and $(\textsf{sim-oracle-query}, u)$, where $u \in \mathcal{U}$.

Consider the following attack game played between an adversary $\mathcal{A}$ and a challenger. The adversary makes a number of queries, each of which is of the following form:

- $(\textsf{sim-proof-query}, y)$, which the challenger passes through directly to Sim;
- $(\textsf{sim-oracle-query}, u)$, which the challenger passes through directly to Sim;
- $(\textsf{attempt-query}, y^*, \pi^*)$, which the challenger processes by checking whether π^* is a valid proof for y^*, and responds with $\textsf{accept}$ if this is the case, and $\textsf{reject}$, otherwise; to make this check, the challenger may need to evaluate the random oracle at various points, and it does so by invoking Sim with queries of the form $(\textsf{sim-oracle-query}, \cdot)$, as necessary.

Note that for queries of the form $(\textsf{sim-proof-query}, y)$, the statement y may very well be a false statement, but nevertheless, Sim responds with a simulated proof π, and we say (y, π) *is a proof query/response pair.*

We say that $\mathcal{A}$ wins the game if the challenger responds with $\textsf{accept}$ to any query of the form $(\textsf{attempt-query}, y^*, \pi^*)$ where (y^*, π^*) is not a previously generated proof query/response pair and y^* is a false statement. We denote by $\mathrm{simSnd}\mathsf{adv}[\mathcal{A}, \Phi, \mathit{Sim}]$ the probability that $\mathcal{A}$ wins the game.

We say that Φ is **simulation sound ZK** if there exists a simulator Sim such that $\text{niZK}\mathsf{adv}[\mathcal{A}, \Phi, Sim]$ is negligible for all efficient adversaries $\mathcal{A}$ and $\text{simSnd}\mathsf{adv}[\mathcal{A}, \Phi, Sim]$ is negligible for all efficient adversaries $\mathcal{A}$.

Let $\Pi = (P, V)$ be a special HVZK Sigma protocol, and consider the corresponding simulator Sim in Fig. 20.2.

(a) Suppose that Π has unique responses (see Exercise 19.14). Show that for every adversary $\mathcal{A}$ that makes at most Q_{ro} random oracle queries and Q_{a} attempt queries in the above simulation soundness attack game, there exists an adversary $\mathcal{B}$ attacking Π as in Attack Game 20.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

$$\text{simSnd}\mathsf{adv}[\mathcal{A}, \text{FS-}\Pi, Sim] \leq (Q_{\text{ro}} + Q_{\text{a}})\text{Snd}\mathsf{adv}[\mathcal{B}, \Pi].$$

Discussion: In particular, if Π is sound and special HVZK, and has unique responses and unpredictable commitments, then FS-Π is simulation sound ZK.

(b) More generally, show that for every adversary $\mathcal{A}$ that makes at most Q_{ro} random oracle queries and Q_{a} attempt queries in the above simulation soundness attack game, there exists an adversary $\mathcal{B}$ attacking Π as in Attack Game 20.1 and an adversary $\mathcal{B}'$ attacking Π with extractor Ext as in Exercise 19.16, where $\mathcal{B}, \mathcal{B}'$ are elementary wrappers around $\mathcal{A}$, such that

$$\text{simSnd}\mathsf{adv}[\mathcal{A}, \text{FS-}\Pi, Sim] \leq (Q_{\text{ro}} + Q_{\text{a}})\text{Snd}\mathsf{adv}[\mathcal{B}, \Pi] + \text{cSSS}\mathsf{adv}[\mathcal{B}', \Pi, Ext].$$

Discussion: In particular, if Π provides computational strong special soundness, is special HVZK, and has unpredictable commitments, then FS-Π is simulation sound ZK.

(c) Show that if Π has backward computable commitments, then the results of part (a) and (b) also hold for the optimized Fiat-Shamir proof system discussed in Exercise 20.14, using the modified simulator in part (b) of that exercise.

20.23 (A DDH-based signature scheme from simulation soundness). This exercise develops a simple, strongly secure signature scheme with a very tight security reduction to DDH. Let $\mathbb{G}$ be a cyclic group of prime order q generated by $g \in \mathbb{G}$. Let $\mathcal{M}$ be the desired message space for the signature scheme $\mathcal{S}$. A public key for $\mathcal{S}$ is a random DH-triple $(u, v, w) \in \mathbb{G}^3$, and the secret key is $\beta \in \mathbb{Z}_q$ such that $v = g^\beta$ and $w = u^\beta$.

The signature scheme is based on a Sigma protocol Π for the relation

$$\mathcal{R} = \left\{ \ (\ \beta,\ (u, v, w, m)\ ) \in \mathbb{Z}_q \times (\mathbb{G}^3 \times \mathcal{M}) :\ v = g^\beta \text{ and } w = u^\beta \ \right\},$$

which generalizes the relation used in the Chaum-Pedersen protocol (see Section 19.5.2). In fact, the protocol Π for $\mathcal{R}$ is really just the Chaum-Pedersen protocol — both the prover and the verifier can just ignore m. We assume that Π has a large challenge space $\mathcal{C}$ of size N.

Now consider the optimized version Φ of the non-interactive proof system obtained by applying the Fiat-Shamir transform to Π (see Exercise 20.14). The proof system Φ uses a hash function $H : (\mathbb{G}^3 \times \mathcal{M}) \times \mathbb{G}^2 \to \mathcal{C}$, which we will model as a random oracle in the security analysis. Although m is ignored by Π, it is not ignored by Φ, as it is included in the hash used to derive the challenge.

A valid signature on a message m under public key (u, v, w) is simply a valid proof π for the statement (u, v, w, m) under Φ.

(a) Describe the signature scheme $\mathcal{S}$ in detail.

(b) Prove that the signature scheme $\mathcal{S}$ is strongly secure in the random oracle model under the DDH assumption. In particular, use the zero knowledge simulator and the result from Theorem 20.3, along with the result of Exercise 20.22, to prove the following: for every adversary $\mathcal{A}$ attacking $\mathcal{S}$ as in the random oracle version of Attack Game 13.2, and making at at most $Q_{\rm s}$ signing queries and $Q_{\rm ro}$ random oracle queries, there exists a DDH adversary $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\text{stSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}] \leq Q_{\rm s}(Q_{\rm s} + Q_{\rm ro} + 1)\frac{1}{q} + \text{DDH}\mathsf{adv}[\mathcal{B}, \mathbb{G}] + \frac{1}{q} + (Q_{\rm ro} + 1)/N.$$

Hint: Game 1: replace the signer by a simulator; Game 2: replace (u, v, w) by a random non-DH-triple (see Exercise 10.7), and then use simulation soundness to bound the advantage.

(c) Analyze the security of the scheme $\mathcal{S}$ in the multi-key setting (as in Exercise 13.2). Show that if at most $Q_{\rm k}$ signature keys are used in the multi-key attack, then the bound in part (a), but with an extra additive term of $Q_{\rm k}/q$, also holds in the multi-key setting.

Hint: Use Exercise 10.11.

20.24 (CCA secure encryption from simulation soundness). Let (G, E, D) be a semantically secure public-key encryption scheme with message space $\mathcal{M}$. Let us assume that E is a deterministic algorithm takes as input a public key pk, a message $m \in \mathcal{M}$, and a randomizer $s \in \mathcal{S}$, so to encrypt a message m, one computes $s \xleftarrow{\rm R} \mathcal{S}$ and $c \leftarrow E(pk, m; s)$.

Let us also assume that $\Phi = (\mathit{GenPrf}, \mathit{VrfyPrf})$ is a simulation sound ZK non-interactive proof system for the relation

$$\mathcal{R} = \{ \ (\ (m, s_0, s_1), \ (pk_0, c_0, pk_1, c_1) \ ) : \ c_0 = E(pk_0, m; s_0) \text{ and } E(pk_1, m; s_1) \ \}.$$

Thus, $(pk_0, c_0, pk_1, c_1) \in L_{\mathcal{R}}$ iff c_0 encrypts some message under the public key pk_0 and c_1 encrypts the same message under the public key pk_1.

We build a new encryption scheme (G', E', D') as follows. The key generation algorithm G' runs G twice, obtaining (pk_0, sk_0) and (pk_1, sk_1). The public key is $pk' := (pk_0, pk_1)$ and the secret key is $(sk_0, \ pk_0, pk_1)$. Given a message m, the encryption algorithm E' computes

$$s_0, s_1 \xleftarrow{\rm R} \mathcal{S}, \ \ c_0 \leftarrow E(pk_0, m; s_0), \ \ c_1 \leftarrow E(pk_1, m; s_1), \ \ \pi \leftarrow \mathit{GenPrf}((m, s_0, s_1), (pk_0, c_0, pk_1, c_1)),$$

and outputs the ciphertext $c' := (c_0, c_1, \pi)$. To decrypt such a ciphertext c', the decryption algorithm D' checks that the proof π is valid and, if so, outputs $D(sk_0, c_0)$, and otherwise, outputs reject.

Prove that (G', E', D') is CCA secure in the random oracle model.

Hint: Start with Attack Game 12.1, but define a number of different experiments parameterized by

$$\mathit{enc} \in \{0, 1\} \times \{0, 1\}, \ \ \mathit{dec} \in \{0, 1\}, \ \ \mathit{prf} \in \{0, 1\}.$$

If $\mathit{enc} = (d, e) \in \{0, 1\} \times \{0, 1\}$, then given an encryption query (m_0, m_1), the challenger encrypts m_d under pk_0 and m_e under pk_1. If $\mathit{dec} = f \in \{0, 1\}$, then given a decryption query $(\hat{c}_0, \hat{c}_1, \hat{\pi})$, the challenger decrypts $\hat{c}_f$ under sk_f. If $\mathit{prf} = 0$, then the challenger generates proofs in encryption queries as usual, but if $\mathit{prf} = 0$ it does so using simulated proofs. Analyze the following sequence of experiments:

0. $\mathit{enc} = (0, 0)$, $\mathit{dec} = 0$, $\mathit{prf} = 0$

1. $\mathit{enc} = (0, 0)$, $\mathit{dec} = 0$, $\mathit{prf} = 1$

2. $\mathit{enc} = (0, 1)$, $\mathit{dec} = 0$, $\mathit{prf} = 1$

3. $\mathit{enc} = (0, 1)$, $\mathit{dec} = 1$, $\mathit{prf} = 1$

4. $\mathit{enc} = (1, 1)$, $\mathit{dec} = 1$, $\mathit{prf} = 1$

5. $\mathit{enc} = (1, 1)$, $\mathit{dec} = 1$, $\mathit{prf} = 0$

6. $\mathit{enc} = (1, 1)$, $\mathit{dec} = 0$, $\mathit{prf} = 0$

20.25 (A concrete instantiation based on DDH). Instantiate the construction in the previous exercise with the multiplicative ElGamal encryption scheme in Section 20.2, along with the optimized Fiat-Shamir non-interactive proof system derived from the Sigma protocol of Example 20.1. Describe the encryption scheme in detail, and verify that all of the assumptions of the previous exercise are satisfied, so that the resulting encryption scheme is CCA secure under the DDH assumption in the random oracle model.

20.26 (ℓ-special soundness implies soundness). Consider again the notion of ℓ-special soundness, as defined in Exercise 19.30. Prove the following generalization of Theorem 20.1: if Π is a Sigma protocol with a large challenge space that provides ℓ-special soundness, then Π is sound (in fact, statistically sound, as in Remark 20.1).

20.27 (Compressed n-wise Chaum-Pedersen). This exercise asks you to design and analyze a compressed n-wise Chaum-Pedersen protocol (see Section 19.5.2). Let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$. Let n be a poly-bounded parameter, and let $\mathcal{R}_n \subseteq (\mathbb{Z}_q^n) \times (\mathbb{G}^{2n+1})$ be the following relation:

$$\mathcal{R}_n := \Big\{ \big((\beta_1, \ldots, \beta_n), (u, v_1, w_1, \ldots, v_n, w_n)\big) \quad : \quad v_i = g^{\beta_i} \text{ and } w_i = u^{\beta_i} \text{ for } i = 1, \ldots, n \Big\}.$$

Using the idea from Exercise 19.31, design and analyze a Sigma protocol for $\mathcal{R}_n$ that requires only constant communication (in particular, independent of n), is special HVZK, and provides $(n + 1)$-special soundness.

Discussion: Combined with Theorems 20.2 and 20.3, and the result from Exercise 20.26, this result shows that when we apply the Fiat-Shamir transform to this protocol, we get a sound zero-knowledge non-interactive proof system for n-wise Chaum-Pedersen. This result illustrates the utility of ℓ-special soundness in a setting where we only need soundness, and not a "proof of knowledge".

20.28 (Generalized "encrypt then prove"). This exercise generalizes Theorem 19.25. Let (G, E, D) be a semantically secure public-key encryption scheme with message space $\mathcal{M}$. As in Exercise 20.24, let us assume that E is a deterministic algorithm takes as input a public key pk, a message $m \in \mathcal{M}$, and a randomizer $s \in \mathcal{S}$, so to encrypt a message m, one computes $s \xleftarrow{\text{R}} \mathcal{S}$ and $c \leftarrow E(pk, m; s)$.

Let Π be a Sigma protocol for the relation

$$\mathcal{R} = \big\{ \ (\ (m, s),\ (pk, c)\ ) : \ c = E(pk, m; s)\ \big\}.$$

Assume that Π has a large challenge space, has unpredictable commitments (see Definition 19.7), is special cHVZK (see Definition 20.6), and provides computational strong special soundness (see Exercise 19.16).

Let $\Phi = (\mathit{GenPrf}, \mathit{VrfyPrf})$ be the non-interactive proof system built from Π via Fiat-Shamir (as in Section 20.3.3).

We build a new encryption scheme (G, E', D') as follows. To encrypt a message m under public key pk, the encryption algorithm G' computes

$$s \xleftarrow{R} \mathcal{S}, \;\; c \leftarrow E(pk, m; s), \;\; \pi \leftarrow \mathit{GenPrf}((m, s), (pk, c_0)),$$

and outputs the ciphertext $c' := (c, \pi)$. To decrypt such a ciphertext c' under secret key sk, the decryption algorithm E' checks that the proof π is valid and, if so, outputs $D(sk, c)$, and otherwise, outputs reject.

Show that if we model the hash in Fiat-Shamir as a random oracle, then (G, E', D') is parallel 1CCA secure, which is a security notion we introduced in Section 19.9.2.2. Recall that this notion of CCA security is defined by a restriction of the 1CCA attack game (see Definition 12.2) in which all of the adversary's decryption queries are submitted in a single batch. You may use the result of Exercise 19.32.

Chapter 21

Authenticated Key Exchange

Suppose Alice and Bob wish to communicate securely over an insecure network. Somehow, they want to use a *secure channel.* In Chapter 9, we saw how Alice and Bob could do this if they already have a shared key; in particular, we looked at real-world protocols, such as IPsec and the TLS record protocol, which provide authenticated encryption of packets, and which also guarantee that packets are delivered in order and without duplicates. However, this begs the question: how do Alice and Bob establish such a shared key to begin with? Protocols that are used for this purpose are called **authenticated key exchange (AKE)** protocols, and are the subject of this chapter.

Roughly speaking, an AKE protocol should allow two users to establish a shared key, called a **session key**. At the end of a successful run of such a protocol, a user, say P, should have a clear idea of which user, say Q, he is talking to, that is, with which user he has established a shared session key (this may be determined either before the protocol runs, or during the course of the execution of the protocol). A secure AKE protocol should ensure that P's session key is effectively a fresh, random key that is known only to Q.

Use of a TTP. Typically, to realize an AKE protocol, we shall need to assume the existence of a **trusted third party (TTP)**, whose job it is to facilitate communication between users who have no prior relationship with each other. Initially, each user of the system must perform some kind of **registration protocol** with the TTP; at the end of the registration protocol, the user has established his own *long term secret key.* If the TTP is *offline*, no further communication with the TTP is necessary, and users do not need to share any secret information with the TTP. Here we will primarily focus on protocols that make use of such an offline TTP. The role of the TTP in these protocols is that of a Certificate Authority, or CA, a notion we introduced in Section 13.8. Recall that a CA issues *certificates* that bind a real-world *identity* to a *public key.*

In Section 21.12, we discuss AKE protocols that use an *online* TTP, which is involved in every run of the AKE protocol, and which shares secret information with users. In general, an offline TTP is preferable to an online TTP. Nevertheless, the advantage of an online TTP is that such protocols can be built using only symmetric key primitives, without public-key tools. In addition, key revocation is relatively simple with an online TTP. However, there are many disadvantages to online TTP protocols, as discussed in Section 21.12.

Multiple user instances and freshness of keys. A given user may run an AKE protocol many times. We shall refer to each such run as an *instance* of that user. While a given user has only a single long-term secret key, we expect that each run of the AKE protocol produces a fresh session key.

For example, a user may wish to set up a secure channel with his bank on Monday, with his office file server on Tuesday, and again with his bank on Wednesday. Freshness guarantees that even if the user's office file server is hacked, and an adversary is able to retrieve the session key from Tuesday, this should not compromise the session key from Monday or Wednesday. The adversary should learn nothing about the Monday and Wednesday keys. Moreover, freshness guarantees that certain methods of realizing secure channels maintain their security across multiple sessions. For example, suppose that a stream cipher is used to maintain the secrecy of the data sent through the secure channel between the user and his bank. If the same key were used to encrypt two different streams, an adversary can mount a "two time pad" attack to obtain information about the encrypted data, as discussed in Section 3.3.1. Freshness ensures that keys used in different sessions are effectively independent of one another.

Security properties: an informal introduction. Secure AKE protocols turn out to be rather tricky to design: there are many subtle pitfalls to avoid. Indeed, part of the problem is that it is challenging to even formally specify what the security goals even should be.

First, let us consider the powers of the adversary. Of course, an adversary may eavesdrop on messages sent between user instances running the protocol. Typically, these messages will include certificates issued by the CA, and so we should assume that such certificates are public and freely available to any adversary. We shall also assume that an adversary may be able to modify messages, and indeed, may be able to inject and delete messages as well. So essentially, *we shall allow the adversary to have complete control over the network.* Of course, this is an overly pessimistic point of view, and a typical real-world adversary will not have this much power, but as usual, in analyzing security, we want to take the most pessimistic point of view.

In addition, some users in the system may register with the CA, but these users may be *corrupt*, and not follow the protocol. Such corrupt users may even collude with one another. For our purposes, we shall just assume that all such corrupt users are under the control of a single adversary. The remaining users of the system are *honest users*, who follow the protocol properly.

We have already hinted at some of the properties we want any secure AKE protocol to satisfy. Let us try to make these just a bit more precise.

Suppose an instance of an honest user P has successfully terminated a run of the AKE protocol, thinking he is talking to an instance of user Q, and holding a session key k. On the one hand, if Q happens to be a corrupt user, the key k is inherently vulnerable, and we might as well assume that k is known to the adversary. On the other hand, if Q is an honest user, we want the following guarantees:

authenticity: the key k, if it is shared with anyone, is shared with an instance of user Q; moreover, *this instance of user Q should think he is talking to an instance of user P*;

secrecy: from the adversary's point of view, the key k is indistinguishable from a random key; moreover, *this should hold even if the adversary sees the session keys from other user instances.*

Later in this chapter (in Section 21.9), we shall make the above security requirements much more precise. In fact, we will consider several levels of security, depending on the exact powers of the adversary. In the weakest security definition, the adversary never compromises the long-term secret key of any honest user. A stronger security notion, called "perfect forward secrecy," defends against an adversary that is able to compromise long-term keys of honest users. An even stronger notion, called "HSM security," defends against an adversary that can read the ephemeral random bits generated by honest users. The issues involved will become clearer after looking at a number of example protocols.

21.1 Identification and AKE

One can think of AKE as a combination of *identification*, discussed in Chapters 18 and 19, and *anonymous* key exchange, discussed in Section 10.1. However, it is not enough to simply run such protocols sequentially.

Consider the following protocol:

1. P identifies himself to Q;

2. Q identifies himself to P;

3. P and Q generate a shared key.

Here, steps 1 and 2 are implemented using an identification protocol (as in Chapter 18), and step 3 is implemented using an anonymous key exchange protocol (as in Section 10.1).

To attack this protocol, an adversary might wait until steps 1 and 2 are complete, and then "hijack" the session. Indeed, suppose that after step 2, the adversary steps in between P and Q, runs one anonymous key exchange protocol with P, obtaining a shared key k_1, and another anonymous key exchange protocol with Q, obtaining a shared key k_2.

If the session key is used to implement a secure channel, then after the protocol completes the adversary can easily play "man in the middle": whenever P encrypts a message under k_1, the adversary decrypts the resulting ciphertext, and then *re-encrypts* the message, possibly after modifying it in some way, under k_2; similarly, messages from Q to P can be decrypted and then re-encrypted. Thus, the adversary is able to read the entire conversation between P and Q, modifying messages at will.

To foil the above attack, one might consider the following protocol:

1. P and Q generate a shared key, and use this key to implement a secure channel;

2. P identifies himself to Q inside the channel;

3. Q identifies himself to P inside the channel.

Here, step 1 is implemented using an anonymous key exchange protocol. This key can then be used to implement a secure channel, and then steps 2 and 3 are implemented by an identification protocol, with each protocol message encrypted under the shared key a symmetric cipher that provides authenticated encryption.

However, an adversary can also easily attack this protocol by playing "man in the middle":

1. The adversary generates a shared key k_1 with P, and other shared key k_2 with Q;

2. During each run of the identification protocol, whenever P sends a message to Q, which is encrypted under k_1, the adversary decrypts the corresponding ciphertext, and then re-encrypts the message under k_2, sending the corresponding ciphertext to Q.

3. Similarly, whenever Q sends a message to P, which is encrypted under k_2, the adversary decrypts the corresponding ciphertext, and then re-encrypts the message under k_1, sending the corresponding ciphertext to P.

When the attack completes, the adversary can simply continue playing "man in the middle."

Thus, these simple-minded approaches to designing a secure AKE protocol do not work. To build a secure AKE protocol, one must carefully intertwine the processes of identification and anonymous key exchange.

21.2 An encryption-based protocol

In this section, we present an AKE protocol, called `AKE1`. As we shall eventually see (in Section 21.9.2), protocol `AKE1` does indeed satisfy our most basic notion of security, called **static security**, in which the adversary never compromises the long-term secret key of any honest user. However, it is vulnerable to more powerful attacks that will be discussed later, and which are modeled by stronger security definitions.

Certificate authority. Protocol `AKE1`, like all the protocols in the chapter up through Section 21.10, makes use of a CA, which issues certificates that bind identities to public keys. For a user P, we shall write id_P to denote P's identity, and let $Cert_P$ be a certificate that binds id_P to a public key. Here, id_P is an arbitrary bit string, unique to this user, and we assume that $Cert_P$ encodes id_P, as well as P's public key pk_P, and a signature on a message of the form "id_P's public key is pk_P", under the CA's public key. We shall assume that all users have access to the CA's public key, so that they can verify certificates.

For this particular protocol, the public key pk_P for a user P consists of the public key for a CCA-secure public key encryption scheme, and the public key for a signature scheme. The long-term secret for user P consists of the corresponding secret keys for the encryption and signature schemes. When P registers with the CA, he presents id_P and pk_P, along with any credentials needed to convince the CA that P's identity "really is" id_P (what these credentials are, and how they are checked, is outside the scope of our description of this protocol). If the CA is happy with these credentials, the CA issues a certificate $Cert_P$, which P retains.

Note that for this protocol and all the other protocols we discuss in this chapter, we do not assume that the CA does anything else besides checking a user's credentials. In particular, the CA does not do anything to ensure that the user's public key satisfies any particular property, or that the user "knows" the corresponding secret key.

Notation. To describe protocol `AKE1`, we use the following notation:

- $Cert_P$ denotes P's certificate, binding his identity id_P to his public keys for encryption and signing;
- $Enc_P(m)$ denotes an encryption of the message m under P's public encryption key;
- $Sig_P(m)$ denotes a signature on the message m under P's public verification key;
- $\mathcal{K}$ denotes the set of session keys;

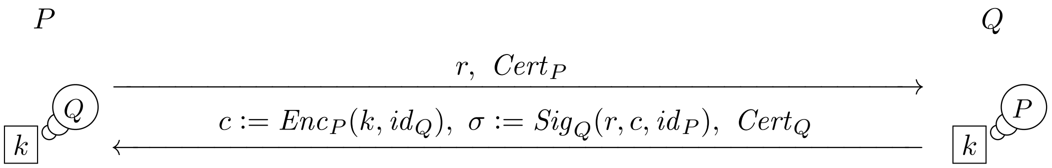
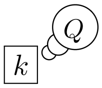

P
Q
$r,\ Cert_P$
$c := Enc_P(k, id_Q),\ \sigma := Sig_Q(r, c, id_P),\ Cert_Q$
k
Q
k
P

Figure 21.1: Protocol `AKE1`

- $\mathcal{R}$ denotes a large set, which will be used to generate random nonces.

When executed by users P and Q, protocol `AKE1` runs as described in Fig. 21.1. Here, r is chosen at random by P from the set $\mathcal{R}$, and k is chosen at random by Q from the set $\mathcal{K}$. Also, each user verifies the certificate it receives; in addition, P verifies the signature σ it receives, and also verifies that c decrypts to a message of the form (k, id_Q).

In Fig. 21.1 we have used the notation



to indicate that when the protocol finishes, a user holds the session key k, and thinks he is talking to user Q.

Here is a more detailed description of the protocol:

1. P computes $r \xleftarrow{\text{R}} \mathcal{R}$, and sends $(r, Cert_P)$ to Q;

2. Q verifies $Cert_P$; if the certificate is invalid, Q aborts; otherwise, Q extracts the identity id_P from $Cert_P$, along with P's public encryption key, and then computes

$$k \xleftarrow{\text{R}} \mathcal{K},\ c \xleftarrow{\text{R}} Enc_P(k, id_Q),\ \sigma \xleftarrow{\text{R}} Sig_Q(r, c, id_P),$$

   and sends $(c, \sigma, Cert_Q)$ to P; in addition, Q terminates successfully, and outputs the session key k, and *partner identity* id_P;

3. P verifies $Cert_Q$; if the certificate is invalid, P aborts; otherwise, P extracts the identity id_Q from $Cert_Q$, along with Q's public verification key, and then verifies that σ is a valid signature on the message (r, c, id_P) under Q's public verification key; if not, P aborts; otherwise, P decrypts the ciphertext c, and verifies that c decrypts to a message of the form (k, id_Q) for some $k \in \mathcal{K}$; if not, P aborts; otherwise, P terminates successfully, and outputs the session key k, and partner identity id_Q.

Remarks. A number of comments are in order, which apply to any AKE protocol:

1. When a user runs this protocol, it either aborts or terminates successfully; when it terminates successfully, the protocol outputs a session key k and a partner identity id. When we say "P thinks he is talking to Q", we really mean that P runs the protocol to a successful termination, and outputs the partner identity id_Q.

2. As we have described this protocol, a user can start the protocol without necessarily knowing the identity of his partner, obtaining this identity (and certificate) along the way. Of course,

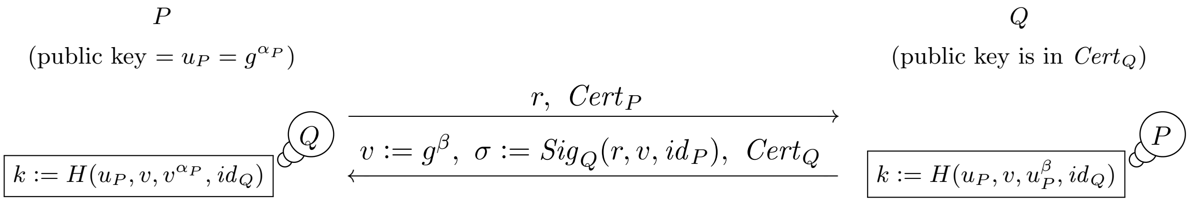

P
(public key $= u_P = g^{\alpha_P}$)
Q
(public key is in $Cert_Q$)
$r,\ Cert_P$
$v := g^{\beta},\ \sigma := Sig_Q(r, v, id_P),\ Cert_Q$
Q
$k := H(u_P, v, v^{\alpha_P}, id_Q)$
P
$k := H(u_P, v, u_P^{\beta}, id_Q)$

Figure 21.2: Protocol $\texttt{AKE1}_{\text{eg}}$

in many situations, a user might know in advance who he plans on talking to, and may abandon the protocol if the partner identity obtained during the run of the protocol does not match his expectations. A user might also abandon the protocol if it "times out."

3. The protocol is inherently asymmetric: the role played by P is quite different from that played by Q. Two users running the protocol will have to establish a convention to decide who plays which role.

4. When a single user runs multiple instances of the protocol, some mechanism is used to route protocol messages to the appropriate instance of the protocol. This routing mechanism is not required to provide any security guarantees, and our description of the protocol does not include any description of this mechanism.

Choice of encryption scheme. We will prove the static security of this protocol in Section 21.9.2. The purpose of encrypting the identity id_Q along with the session key k is to bind this identity to the ciphertext c. CCA-secure encryption is needed to ensure that this binding cannot be broken. If we wish, we could reduce the length of the ciphertext by encrypting a collision-resistant hash of id_Q instead of id_Q itself. We saw a similar usage of binding public information to a ciphertext in Section 12.2.3. In fact, instead of encrypting id_Q (or a hash thereof), we could use a CCA-secure public-key encryption scheme with associated data, as in Section 12.7, treating id_Q as the associated data in this application. Since we are just encrypting a random key with associated data, we could get by with a key encapsulation mechanism (KEM) with associated data (see Exercise 12.18).

If we use the KEM corresponding to the ElGamal encryption scheme $\mathcal{E}_{\text{EG}}$ (see Section 12.4), we get the key exchange protocol $\texttt{AKE1}_{\text{eg}}$ shown in Fig. 21.2. Here, $\mathbb{G}$ is a cyclic group of prime order q generated by $g \in \mathbb{G}$, and $H : \mathbb{G}^3 \times \mathit{IDSpace} \to \mathcal{K}$ is a hash function, where user identities belong to $\mathit{IDSpace}$. User P's public key is $u_P \in \mathbb{G}$ and secret key is $\alpha_P \in \mathbb{Z}_q$. On each run of the protocol, P generates $r \in \mathcal{R}$ at random and Q generates $\beta \in \mathbb{Z}_q$ at random. At the end of the protocol, both users compute the session key $k := H\big(g^{\alpha_P}, g^{\beta}, g^{\alpha_P \beta}, id_Q\big)$ (we have added $u_P = g^{\alpha_P}$ to the hash for a tighter security reduction). For CCA security of the KEM we model H as a random oracle, and assume that ICDH, defined in Section 12.4, holds for $\mathbb{G}$. As discussed in Remark 12.1, P should check that v is in $\mathbb{G}$. The description of the group $\mathbb{G}$, including the generator g, is viewed as a shared system parameter.

Instead of ElGamal, one could use any other CCA-secure encryption scheme, such as $\mathcal{E}_{\text{RSA}}$.

Erasing ephemeral data. We are assuming (for now) that the user's long-term keys are never compromised. However, in a secure implementation of any session key protocol, it is important that the participants securely erase any ephemeral data they generated during the protocol. If any of this data ends up stored in memory which an adversary can read at some later time, then the adversary may be able to break the system. For example, if we use ElGamal encryption as in protocol $\texttt{AKE1}_{\text{eg}}$, then it is important that Q securely erases the value β — if this leaks, the adversary can obviously recover the session key k. Similarly, any random bits that go into generating signatures should also be securely erased. It is easy to see that if the random bits that go into generating a Schnorr signature (see Section 19.2) are leaked, then the adversary can trivially compute the long-term signing key. This attack is potentially even more devastating, since instead of just obtaining a single session key, the adversary can impersonate a user at any time, as often as he likes, to any user. This is another reason to derandomize signature schemes, as discussed in Exercise 13.6, so as to avoid this problem altogether.

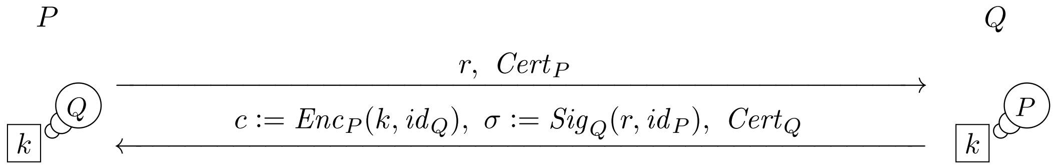
Implicit authentication. Protocol `AKE1` only provides **implicit authentication**, in the following sense. When P finishes the protocol, he can be confident that Q was "alive" during the run of the protocol (since Q must have signed the message containing P's random nonce); moreover, P can in fact be confident that some instance of Q finished the protocol and is holding a matching session key. However, when Q finishes the protocol, he has no such guarantee: not only may there not be an instance of P with a matching session key, but P may not have even been "alive" during the execution of the protocol. Nevertheless, Q can be sure of this: *if* anyone at all eventually shares his session key, *then* that someone is an instance of P (who thinks he is talking to Q).

21.2.1 Insecure variations

To appreciate why this protocol is designed the way it is, it is instructive to consider minor variations that are susceptible to various attacks, and illustrate how these attacks might be exploited in the real world. These attacks serve to illustrate the types of vulnerabilities any secure AKE should avoid, and demonstrate why each and every piece of protocol `AKE1` is essential to achieve security.

Variation 1: do not sign c — a key exposure attack

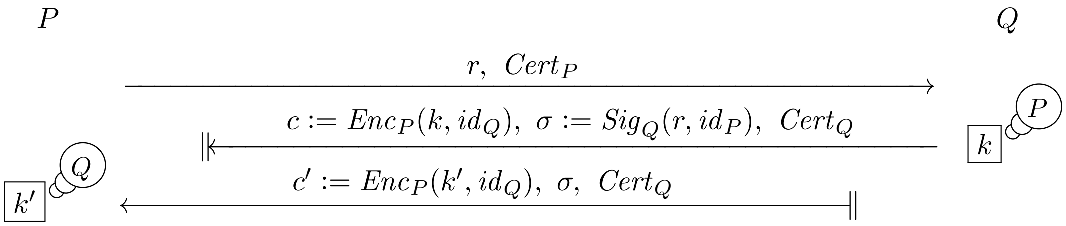
Suppose we modify protocol `AKE1` so that the message signed by Q does not include the ciphertext c; likewise, the logic of P is modified accordingly. The resulting protocol runs as follows:



This modified protocol can be attacked as follows:

- the adversary intercepts the message $(c, \sigma, Cert_Q)$ from Q to P;
- the adversary computes $c' \xleftarrow{R} Enc_P(k', id_Q)$, where k' is a session key of his choosing, and sends the message $(c', \sigma, Cert_Q)$ to P.

The following diagram illustrates the attack:


P
Q
$r,\ Cert_P$
$c := Enc_P(k, id_Q),\ \sigma := Sig_Q(r, id_P),\ Cert_Q$
$c' := Enc_P(k', id_Q),\ \sigma,\ Cert_Q$
Q
k'
P
k

In the diagram, we write $\|\!\!\leftarrow$ to indicate a message blocked by the adversary, and $\leftarrow\!\!\|$ to indicate a message generated by the adversary. At the end of the attack, Q is holding the session key k, which is unknown to the adversary; however, P is holding the session key k', which is known to the adversary.

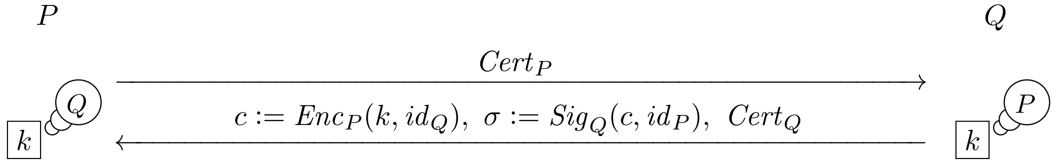
This type of attack, where the adversary is able to recover (or in this case, even choose) a session key, is called a **key exposure attack**, and certainly violates the *secrecy property*. However, let us consider a more concrete scenario to illustrate why this attack is dangerous, even though the adversary knows only k', but not k. Suppose that after the AKE protocol is run, the session key is used to secure a conversation between P and Q, using authenticated encryption. If P sends the first message in this conversation, then the adversary can obviously decrypt and read this message. Alternatively, if P receives the first message in the conversation, the adversary can make this message anything he wants.

Let us flesh out this attack scenario even further, and consider a hypothetical electronic banking application. Suppose that one user is a bank and the other a customer. Further, suppose that the conversation between the bank and customer is a sequence of request/response pairs: the customer sends a transaction request, the bank executes the transaction, and sends a response to the customer.

On the one hand, suppose P is the customer and Q is the bank. In this case, the first request made by the customer can be read by the adversary. Obviously, such a request may contain private information, such as a credit card or social security number, which the customer obviously does not want to share with an adversary. On the other hand, suppose P is the bank and Q is the customer. In this case, the adversary can send the bank a request to perform some arbitrary transaction on the customer's account, such as to transfer money from the customer's account into some bank account controlled by the adversary.

Variation 2: do not sign r — a replay attack

Suppose we modify protocol `AKE1` so that the message signed by Q does not include the random nonce r; likewise, the logic of P is modified accordingly. The resulting protocol runs as follows:

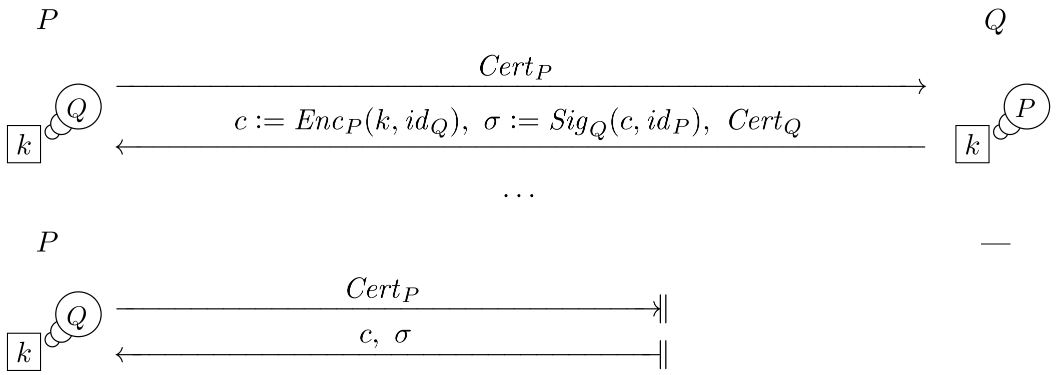


In this new protocol, r is not really used at all, so we leave it out.

This new protocol is susceptible to the following attack:

- first, the adversary eavesdrops on a conversation between P and Q; suppose P sent the message Cert_P to Q, who responds with the message $(c, \sigma, \mathit{Cert}_Q)$; these messages are recorded by the adversary;
- at some later time, the adversary, initiates a new run of the protocol with P; P sends out the message Cert_P; the adversary intercepts this message, throws it away, and sends P the message $(c, \sigma, \mathit{Cert}_Q)$, recorded from the previous run of the protocol.

The following diagram illustrates the attack:


P
Q
Cert_P
Q
k
$c := \mathit{Enc}_P(k, \mathit{id}_Q),\ \sigma := \mathit{Sig}_Q(c, \mathit{id}_P),\ \mathit{Cert}_Q$
P
k
...
P
—
Cert_P
Q
k
$c,\ \sigma$

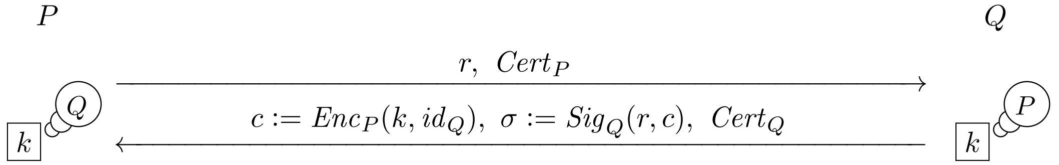
At the end of the attack, the second instance of user P thinks he is talking to Q, but the session key of the second instance of P is exactly the same as the session key k of the first instance of P. Note that the adversary does not obtain any direct information about k, nor is there a new instance of Q that shares this key. This type of attack, where the adversary is able to force a user instance to re-use an old session key, is called a **replay attack**, and it also violates the *secrecy property.*

Even though the adversary obtains no direct information about k from the attack, this attack can still be exploited. Suppose, for example, that k is used to implement a secure channel that uses a stream cipher in its implementation. In this way, the adversary might be able to get P to encrypt two different messages, using a stream cipher, under the same secret key. As discussed in Section 3.3.1, this might allow the adversary to obtain information about the encrypted data, via a "two time pad" attack.

Another way this replay attack might be exploited is to simply replay some part of the first conversation between P and Q. Indeed, returning to our bank example, suppose P is the bank and Q is the customer. Then if in the first conversation, the customer requested a certain amount of money to be transferred to a third party's account, the adversary could simply replay this request, and cause the bank to transfer the same amount of money a second time.

Variation 3: do not sign id_P — an identity misbinding attack

Suppose we modify protocol `AKE1` so that the message signed by Q does not include the identity id_P; likewise, the logic of P is modified accordingly. The resulting protocol runs as follows:


P
Q
$r,\ \mathit{Cert}_P$
Q
k
$c := \mathit{Enc}_P(k, \mathit{id}_Q),\ \sigma := \mathit{Sig}_Q(r, c),\ \mathit{Cert}_Q$
P
k

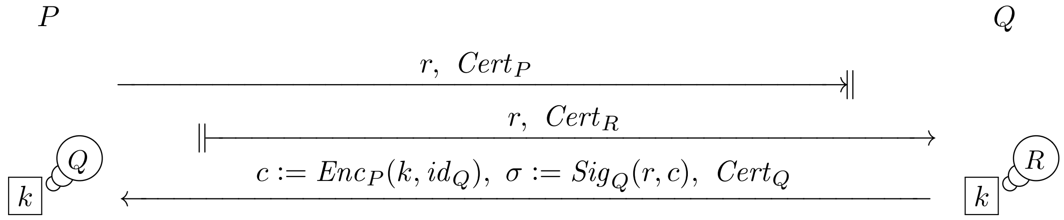
Here is a rather subtle attack on this protocol:

- after obtaining P's public key by some means, the adversary registers a new user R with the CA, obtaining a certificate $Cert_R$ that binds R's identity, id_R, to P's public key;

- at some later time, P and Q engage in the AKE protocol; when P sends the message $(r, Cert_P)$, the adversary intercepts this message, and instead delivers the message $(r, Cert_R)$ to Q;

- when Q sends the message $(c, \sigma, Cert_Q)$, the adversary delivers this message to P.

The following diagram illustrates the attack:



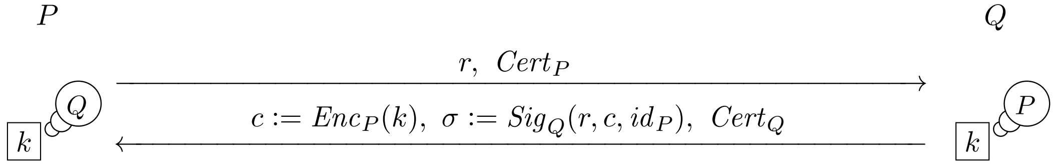
At the end of the attack, P and Q share the session key k, which is unknown to the adversary; however, P thinks he is talking to Q, while Q thinks he is talking to R. This type of attack is called an **identity misbinding attack**, and it violates the *authentication property.*

Note that to carry out the attack, R needs to "hijack" P's public key; that is, the adversary registers the user R with the CA, but using P's public key. Recall that we are assuming here that although the CA checks R's credentials (i.e., he is who he says he is), the CA does not necessarily require R to prove that he has the corresponding secret key (which he could not do in this case).

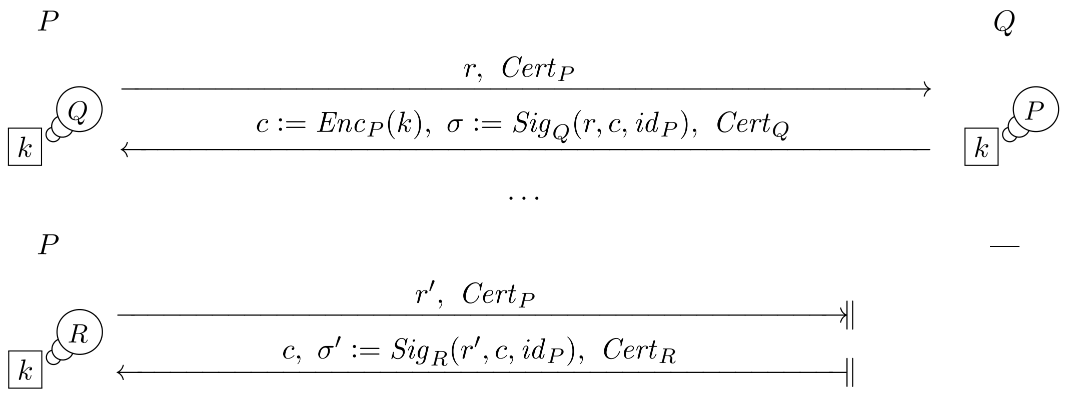
It is perhaps not so easy to exploit an identity misbinding attack, but here is one semi-plausible scenario. Nowadays, one can buy plastic "voucher cards" at stores, and these voucher cards can be redeemed on the Internet in various ways, for example, to add credit to a prepaid cell phone account. To redeem a voucher card, a customer logs in to his account, and then types in a serial number that appears on the voucher card, and the value of the voucher card is added to the customer's account. Now, suppose that the above protocol is used to allow users to log into their accounts, and that Q represents the cell phone company, and that P and R are customers. In the above misbinding attack, the phone company thinks he is talking to R, when really, he is talking to P. So when the unsuspecting customer P redeems a voucher card, Q credits the value of the voucher card to R's account, rather than to P's account.

Variation 4: do not encrypt id_Q — a replay attack

Suppose we modify protocol `AKE1`, so that Q does not encrypt his identity. The new protocol runs as follows:



The following diagram illustrates a simple replay attack on this protocol:



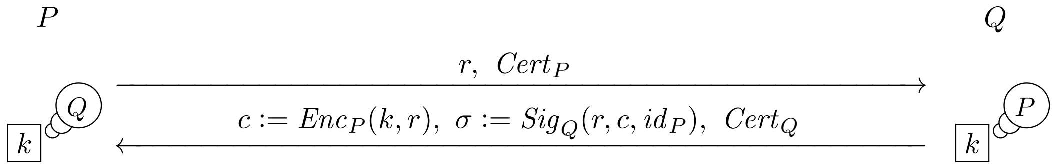
P
Q
$r,\ Cert_P$
$c := Enc_P(k),\ \sigma := Sig_Q(r, c, id_P),\ Cert_Q$
Q
k
P
k
$\ldots$
P
—
$r',\ Cert_P$
$c,\ \sigma' := Sig_R(r', c, id_P),\ Cert_R$
R
k

Here, R is a corrupt user, under control of the adversary. However, we assume that R has registered with the CA as usual, and so has a certificate that binds his identity to a public key for which he has a corresponding secret key. Thus, in the last flow, the adversary may easily generate the required signature $Sig_R(r', c, id_P)$.

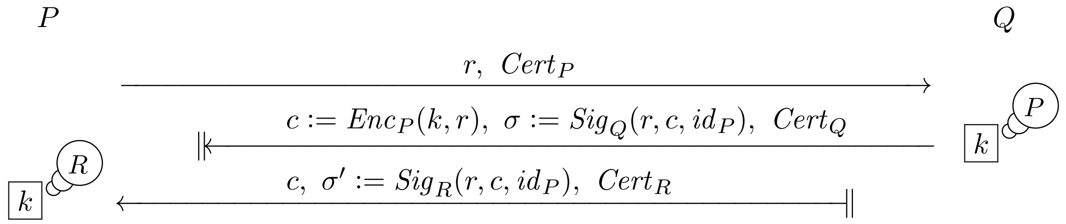
The end result of this replay attack is essentially the same as the replay attack we saw against Variation 2, except that in this case, the second instance of user P thinks he is talking to R, instead of to Q.

Variation 5: encrypt r instead of id_Q — an identity misbinding attack

Suppose that Q encrypts r instead of id_Q. The new protocol runs as follows:



As in Variation 3, this protocol is susceptible to an identity misbinding attack:



P
Q
$r,\ Cert_P$
$c := Enc_P(k, r),\ \sigma := Sig_Q(r, c, id_P),\ Cert_Q$
P
k
R
k
$c,\ \sigma' := Sig_R(r, c, id_P),\ Cert_R$

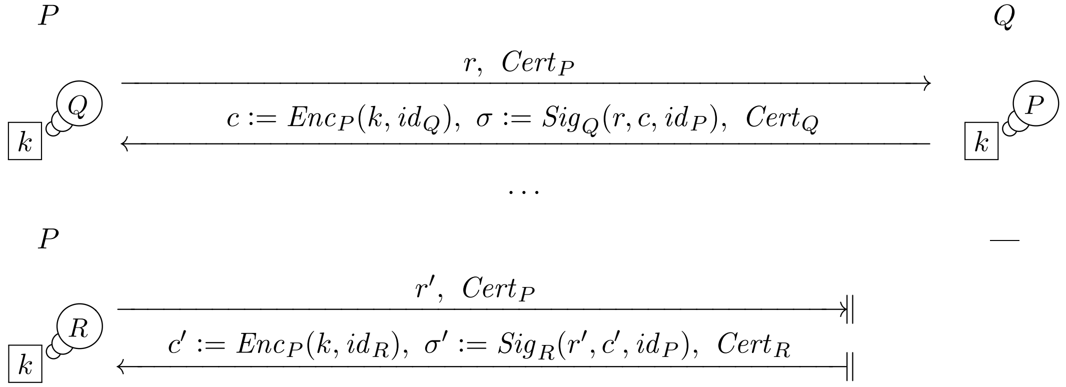
At the end of this attack, P and Q share the session key k, which is unknown to the adversary; however, P thinks he is talking to R, while Q thinks he is talking to P.

As in Variation 4, R is a corrupt user, under the control of the adversary, but we assume that R has registered with the CA as usual — unlike Variation 3, R does not need to "hijack" another user's public key.

Variation 6: The need for CCA secure encryption

Suppose we use an encryption scheme that is semantically secure, but not necessarily CCA secure. There are a number of types of attack that may be possible, depending on the scheme.

For example, suppose that we use the encryption scheme $\mathcal{E}_{\text{TDF}}$, based on a trapdoor function, as discussed in Section 11.4. This scheme makes use of a semantically secure cipher, and we shall assume that this is a stream cipher. With these assumptions, given a ciphertext c that encrypts some unknown bit string m, and given an arbitrary bit string Δ, one can easily compute a ciphertext c' that encrypts $m \oplus \Delta$ (see Section 3.3.2). Now, suppose that $m = k \parallel id_Q$ is the encoding of the pair (k, id_Q), where k is an ℓ-bit string. Then setting $\Delta := 0^\ell \parallel (id_Q \oplus id_R)$, we can easily transform an encryption c of $k \parallel id_Q$ into an encryption c' of $k \parallel id_R$, without any knowledge of k. Because of this, we can easily modify the replay attack on Variation 4, so that it works on the original protocol `AKE1`, as follows:


P
Q
$r,\ Cert_P$
$c := Enc_P(k, id_Q),\ \sigma := Sig_Q(r, c, id_P),\ Cert_Q$
Q
k
P
k
...
P
—
$r',\ Cert_P$
$c' := Enc_P(k, id_R),\ \sigma' := Sig_R(r', c', id_P),\ Cert_R$
R
k

Another avenue of attack is to send “trick” ciphertexts to P, so that the decryptions of the trick ciphertexts reveal secret information. For example, an attacker could use the Bleichenbacher attack on PKCS1 from Section 12.8.3 to recover a secret session key. The adversary could record the ciphertext c sent from Q to P during a run of the protocol between Q and P. Then, by later sending to P “trick” ciphertexts derived from c, as in Bleichenbacher’s attack, the attacker could learn the decryption of c. This will expose the secret session key between P and Q.

21.2.2 Summary

We have presented the AKE protocol `AKE1`, and have illustrated how several variants of this protocol are insecure. In particular, we illustrated three basic types of attack:

- a **key recovery attack**, in which an adversary is able to recover (or even choose) a session key;
- a **replay attack**, in which an adversary is able to force a user instance to re-use an old session key;
- an **identity misbinding attack**, in which an adversary is able to make two user instances share a key, but these two user instances have conflicting views of who is talking to whom.

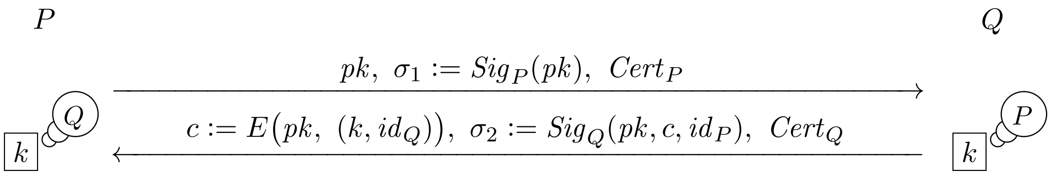

P
Q
$pk,\ \sigma_1 := Sig_P(pk),\ Cert_P$
$c := E\big(pk,\ (k, id_Q)\big),\ \sigma_2 := Sig_Q(pk, c, id_P),\ Cert_Q$
Q
P
k
k

Figure 21.3: Protocol `AKE2`

21.3 Perfect forward secrecy and a protocol based on ephemeral encryption

If an adversary obtains a user's long-term secret key, the adversary may impersonate that user going forward, and cause a great deal of damage. However, it would be nice if the damage could be limited to the time *after* which the user's key was compromised, so that at least session keys generated before the compromise remain secret. This additional security property is called **perfect forward secrecy**. If a protocol satisfies this property, we say that it is **PFS secure**.

Protocol `AKE1` in Section 21.2 certainly is not PFS secure. Indeed, if a user's long-term decryption key is obtained by an adversary, then *all* previous session keys encrypted under that user's encryption key become available to the adversary.

In this section, we present another AKE protocol, called `AKE2`, that is PFS secure. This protocol makes use of a CCA-secure public-key encryption scheme $\mathcal{E} = (G, E, D)$ along with a signature scheme. The public key for each user is a verification key for the signature scheme, and the long-term secret key is the corresponding secret signing key. A new, "ephemeral" key pair for the encryption scheme is generated with every run of the protocol.

When executed by users P and Q, protocol `AKE2` runs as described in Fig. 21.3. Here, user P generates a key pair (pk, sk) every time he runs the protocol. In addition, each user verifies the certificates and signatures it receives.

Here is a more detailed description of protocol `AKE2`:

1. P computes
$$(pk, sk) \xleftarrow{R} G(),\ \sigma_1 \xleftarrow{R} Sig_P(pk),$$
and sends $(pk, \sigma_1, Cert_P)$ to Q;

2. Q verifies $Cert_P$; if the certificate is invalid, Q aborts; otherwise, Q extracts the identity id_P from $Cert_P$, along with P's public verification key; Q verifies that σ_1 is a valid signature on pk under P's public verification key; if not, Q aborts; otherwise, Q computes
$$k \xleftarrow{R} \mathcal{K},\ c \xleftarrow{R} E(pk, (k, id_Q)),\ \sigma_2 \xleftarrow{R} Sig_Q(pk, c, id_P),$$
and sends $(c, \sigma_2, Cert_Q)$ to P; in addition, Q terminates successfully, and outputs the session key k, and partner identity id_P;

3. P verifies $Cert_Q$; if the certificate is invalid, P aborts; otherwise, P extracts the identity id_Q from $Cert_Q$, along with Q's public verification key, and then verifies that σ is a valid signature

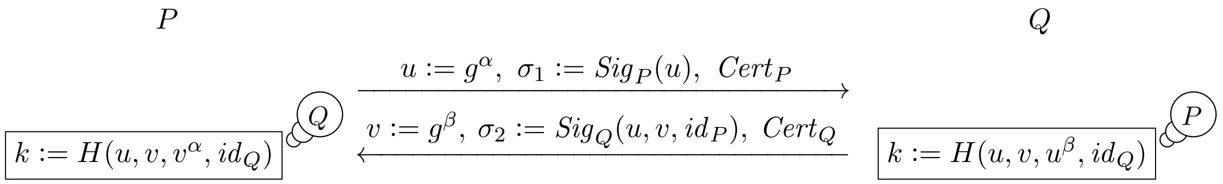

P
Q
$u := g^{\alpha},\ \sigma_1 := \mathit{Sig}_P(u),\ \mathit{Cert}_P$
$v := g^{\beta},\ \sigma_2 := \mathit{Sig}_Q(u, v, \mathit{id}_P),\ \mathit{Cert}_Q$
$k := H(u, v, v^{\alpha}, \mathit{id}_Q)$
Q
$k := H(u, v, u^{\beta}, \mathit{id}_Q)$
P

Figure 21.4: Protocol $\mathtt{AKE2}_{\mathrm{eg}}$

on the message (pk, c, id_P) under Q's public verification key; if not, P aborts; otherwise, P decrypts the ciphertext c, and verifies that c decrypts to a message of the form (k, id_Q) for some $k \in \mathcal{K}$; if not, P aborts; otherwise, P terminates successfully, and outputs the session key k, and partner identity id_Q.

Forward secrecy. Intuitively, protocol AKE2 is PFS secure because user long-term keys are used only for signing, not encrypting. So compromising a signing key should not allow the adversary to decrypt any messages.

Choice of encryption scheme. Just as we did for protocol AKE1, we can make use of an ElGamal-based KEM to implement the encryption. The resulting protocol, called $\mathtt{AKE2}_{\mathrm{eg}}$, is shown in Fig. 21.4. As before, $\mathbb{G}$ is a cyclic group of prime order q generated by $g \in \mathbb{G}$, and $H : \mathbb{G}^3 \times \mathit{IDSpace} \to \mathcal{K}$ is a hash function. User P generates $\alpha \in \mathbb{Z}_q$ at random, while user Q generates $\beta \in \mathbb{Z}_q$ at random. At the end of the protocol, both users compute the session key $k = H(g^{\alpha}, g^{\beta}, g^{\alpha\beta}, id_Q)$. Again, H is modeled as a random oracle and we assume that ICDH holds for $\mathbb{G}$ (see Section 12.4). As discussed in Remark 12.1, P should check that v is in $\mathbb{G}$. It is *not* necessary for Q to check that u is in $\mathbb{G}$.

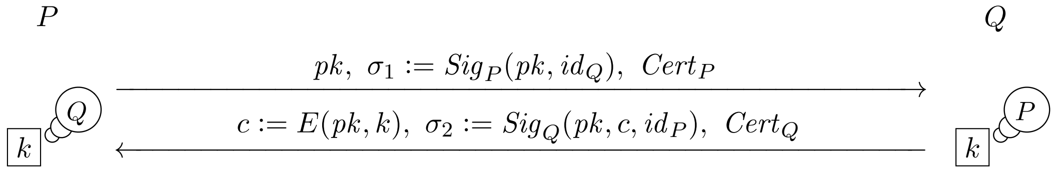
Just as for protocol AKE1, we could use $\mathcal{E}_{\mathrm{RSA}}$ instead of ElGamal. However, this is not very practical, as key generation for RSA is much slower than for ElGamal, and the key generation algorithm must be executed with every run of the protocol.

Erasing ephemeral data. As we discussed above, in a secure implementation of any session key protocol, it is important that the participants securely erase any ephemeral data they generated during the protocol. Again, if we use ElGamal encryption as in protocol $\mathtt{AKE2}_{\mathrm{eg}}$, if either Q's value β or P's value α is leaked, the adversary can obviously recover the session key. Worse, if α leaks, the adversary can even do more damage: he can impersonate P at any time, as often as he likes, to any user. This is because the adversary has both α and P's signature on $u = g^{\alpha}$. This is all the adversary needs to run the protocol and establish a shared session key with any user Q, causing Q to think it is talking to P.

Insecure variations. Because of the similarity of protocol AKE2 to protocol AKE1, most of the examples of insecure variations discussed in Section 21.2.1 can be easily adapted to protocols AKE2 and $\mathtt{AKE2}_{\mathrm{eg}}$. See also Exercise 21.2.

21.3.1 Assuming only semantically secure encryption

We briefly discuss the possibility of modifying protocol AKE2 so that it requires only a semantically secure encryption scheme. Without CCA security, the protocol is vulnerable to a similar attack as in Variation 6 of AKE1. Therefore, for this to have any chance of success, we have to assume that one of the two users knows the identity of its partner before the protocol starts. So in the following protocol, we assume P knows the identity id_Q of its partner beforehand.


P
Q
$pk,\ \sigma_1 := Sig_P(pk, id_Q),\ Cert_P$
$c := E(pk, k),\ \sigma_2 := Sig_Q(pk, c, id_P),\ Cert_Q$
Q
k
P
k

While this protocol is statically secure, it is not PFS secure if we only assume that the encryption scheme is semantically secure. It is instructive to see why this is the case. Suppose the adversary lets P send the first message $(pk, \sigma_1, Cert_P)$ to Q, and then Q responds with $(c, \sigma_2, Cert_Q)$. However, suppose the adversary blocks this last message, but that Q uses the session key k to encrypt a plaintext m_1, sending the resulting ciphertext c_1 out over the network for the adversary to see. At this point in time, neither P nor Q is compromised, and so we expect that the adversary should not be able to learn anything about m_1. Moreover, PFS security should imply that this holds *even if P or Q is compromised in the future.*

So suppose that at some later time, the adversary is able to obtain Q's signing key. This allows the adversary to send a message $(c', \sigma_2', Cert_Q)$ to P, where $c' \neq c$ and σ_2' is a valid signature on (pk, c', id_P). This means that P will accept the signature and decrypt c', obtaining some session key k' that may be different from but *related to* k. For example, the adversary may be able to make $k' \oplus k = \Delta$ for some $\Delta \neq 0$ of the adversary's choice. Now, suppose P encrypts a plaintext m_2 under k', and sends the resulting ciphertext c_2 out over the network for the adversary to see.

The adversary may now be able to carry out a *related key attack* on the symmetric cipher, analyzing the ciphertexts c_1 and c_2, and exploiting the fact that they are produced using keys whose XOR is Δ to learn something new about the plaintext m_1. Indeed, the standard definitions of security for symmetric ciphers make no security guarantees when such related keys are used.

More generally, this attack violates our informal secrecy requirement, which says that learning one session key (in this case P's) should not reveal anything about a different session key (in this case Q's).

21.4 HSM security

We emphasized a number of times that in a secure implementation of a session key protocol, it is important that the participants securely erase all ephemeral data they generated during the protocol. Consider again protocol AKE2 in Section 21.3. If the value *sk* generated by P during a run of the protocol is somehow leaked to an adversary, the consequences are devastating: using *sk* and P's signature on *pk*, the adversary can impersonate P at any time, as often as he likes, to any user.

Such ephemeral leakage could occur in a number of ways. The protocol could be poorly implemented, and fail to securely erase this data. Alternatively, the user's machine could be temporarily infected with malware that is able to observe the machine's memory while the protocol is running.

If such leakage occurs, some damage is unavoidable. It would be nice if protocols could be designed so that the damage is limited to only those sessions where the leakage occurred.

One might object to this whole line of inquiry: if an adversary is able to read this ephemeral data, what is to keep him from reading the user's long-term secret key? Indeed, in many implementations of the protocol, this objection is perfectly reasonable. However, in well-designed implementations, special care is taken to ensure that the long-term key is carefully stored and not as easily accessed as the ephemeral data or even the session key itself. In this situation, it is reasonable to demand more from a key exchange protocol.

A good way to think about the attack model is in terms of a **Hardware Security Module (HSM)**. An HSM is a specialized piece of hardware that stores a user P's long-term secret key LTS_P, and which can only be used as an "oracle" that computes a protocol-specific function $f(LTS_P, x)$. That is, given x, the HSM computes and outputs the value $f(LTS_P, x)$. During a limited time window, the adversary may have access to the HSM, and be able to evaluate $f(LTS_P, x)$ for any x of its choice, but not to extract LTS_P from the hardware. This should only compromise a limited number of sessions, and only sessions in which the adversary had access to the HSM while the key exchange protocol was running. Of course, as in PFS security, we also consider a permanent compromise where the adversary permanently steals LTS_P.

While an HSM may be implemented using special hardware, it may also be implemented as an isolated "enclave" enforced by the processor [47]. Such enclaves are becoming ubiquitous.

HSM security. Very roughly speaking, **HSM security** means that if an instance of user Q runs the protocol to completion, and thinks he shares a session key with P, then that session key should be vulnerable only if

(i) P is a corrupt user,

(ii) P is an honest user, but LTS_P was compromised at some time in the past, or

(iii) P is an honest user, but the adversary accessed P's HSM during the (presumably short) window of time that Q was running the protocol, and moreover, the number of other user instances who think they are talking to P is less than the total number of times P's HSM was accessed.

Condition *(i)* corresponds to static security, conditions *(i)–(ii)* correspond to PFS security, and conditions *(i)–(iii)* correspond to HSM security (so HSM security is at least as strong as PFS security). Essentially, condition (iii) says that the sessions damaged by a single HSM query are limited in both time and number. The formal definition is fleshed out in Section 21.9.4.

HSM security is a very strong notion of security. It can be used to model leakage of ephemeral data: if a run of the protocol leaks ephemeral data, then we treat that run of the protocol as if the adversary ran the protocol, accessing the HSM just as the protocol itself would. However, it can also model much stronger attacks, in which the adversary can actively probe the HSM with arbitrary inputs, to try to learn something more about LTS_P than could be gained by observing honest runs of the protocol.

One might argue that we should just put the entirety of protocol AKE2 in the HSM, and thereby trivially obtain HSM security. However, we would prefer the interface to the HSM be as simple as possible. Moreover, we *insist* that the HSM is just an oracle for a simple function and is completely *stateless*. This requirement rules out the possibility of encapsulating protocol AKE2 in

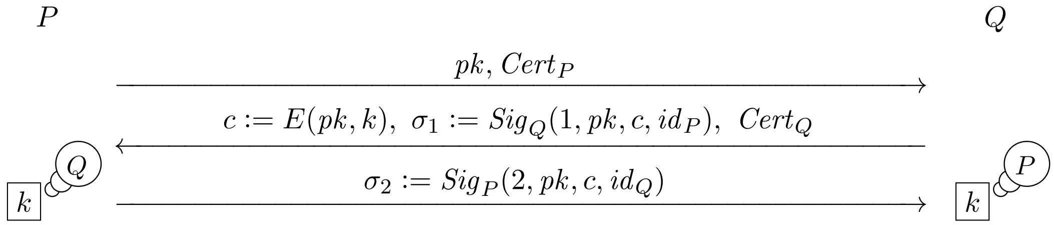

P
Q
$pk, Cert_P$
$c := E(pk,k),\ \sigma_1 := Sig_Q(1, pk, c, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, c, id_Q)$
Q
k
P
k

Figure 21.5: Protocol AKE3

an HSM. Moreover, even without this restriction, AKE2 is not HSM secure: an adversary could query P's HSM on Monday to get the first flow of the protocol, interact with Q on Tuesday, and then interact with P's HSM on Wednesday to finish the protocol. This attack would contradict the requirement that compromises based on HSM access should be limited in time. Our goal is to construct an efficient protocol that achieves HSM security, where the HSM stores a signing key, and does nothing more than act as a stateless "signing oracle."

In the following protocol, which we call AKE3, each user has a long-term public key that is a public key for a signature scheme. The corresponding long-term signing key is stored in an HSM that signs arbitrary messages. In addition, the protocol makes use of a semantically secure public-key encryption scheme $\mathcal{E} = (G, E, D)$. As in protocol AKE2, a new, ephemeral key pair for the encryption scheme is generated with every run of the protocol. On the plus side, we will only need to assume that $\mathcal{E}$ is semantically secure (instead of CCA secure). On the minus side, the protocol consists of three flows (instead of two).

When executed by users P and Q, protocol AKE3 runs as described in Fig. 21.5. Here, user P generates a key pair (pk, sk) every time he runs the protocol. In addition, each user verifies the certificates and signatures it receives.

Here is a more detailed description of protocol AKE3:

1. P computes $(pk, sk) \xleftarrow{R} G()$, and sends $(pk, Cert_P)$ to Q;

2. Q verifies $Cert_P$; if the certificate is invalid, Q aborts; otherwise, Q extracts the identity id_P from $Cert_P$, along with P's public verification key, and then computes

$$k \xleftarrow{R} \mathcal{K},\ c \xleftarrow{R} E(pk, k),\ \sigma_1 \xleftarrow{R} Sig_Q(1, pk, c, id_P),$$

   and sends $(c, \sigma_1, Cert_Q)$ to P;

3. P verifies $Cert_Q$; if the certificate is invalid, P aborts; otherwise, P extracts the identity id_Q, along with Q's public verification key, and then verifies that σ_1 is a valid signature on the message $(1, pk, c, id_P)$ under Q's public verification key; if not, P aborts; otherwise, P computes $k \leftarrow D(sk, c)$; if $k = \text{reject}$; then P aborts; otherwise, P computes $\sigma_2 \xleftarrow{R} Sig_P(2, pk, c, id_Q)$, and sends σ_2 to Q; in addition, P terminates successfully, and outputs the session key k, and partner identity id_Q;

4. Q verifies that σ_2 is a valid signature on the message $(2, pk, c, id_Q)$ under P's public verification key; if not, Q aborts; otherwise, Q terminates successfully, and outputs the session key k, and partner identity id_P.

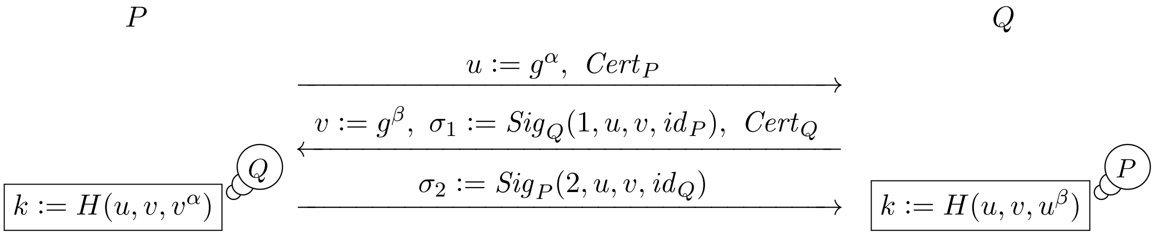

P
Q
$u := g^{\alpha},\ Cert_P$
$v := g^{\beta},\ \sigma_1 := Sig_Q(1, u, v, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, u, v, id_Q)$
$k := H(u, v, v^{\alpha})$
Q
$k := H(u, v, u^{\beta})$
P

Figure 21.6: Protocol $\mathtt{AKE3}_{\mathrm{eg}}$

Ensuring HSM security. A key property that is needed to prove HSM security is that both P and Q get their peer to sign random challenges during the protocol. This ensures that the HSM must have been accessed during the protocol to sign that particular random challenge — either indirectly, by an honest user instance, or directly, by the adversary. This is essential to achieve HSM security. It also means that every HSM secure protocol must have three flows.

Choice of encryption scheme. As we did for protocols AKE1 and AKE2, we can implement protocol AKE3 using ElGamal encryption. This is shown in Fig. 21.6. To prove security, either H is modeled as a random oracle and we assume that CDH holds for $\mathbb{G}$, or H is modeled as a secure KDF and we assume that DDH holds for $\mathbb{G}$ (or we use the HDH assumption in Exercise 11.14). Also note that since we do not require CCA security, it is not necessary for P to explicitly check that v is in $\mathbb{G}$ (or for Q to explicitly check that u is in $\mathbb{G}$).

21.4.1 A technical requirement: strongly unpredictable ciphertexts

To prove HSM security, we need to impose a non-standard, but perfectly reasonable, requirement on the public-key encryption scheme. Namely, that it is hard to predict the output of the encryption algorithm on a given public key and given message. Although semantic security implies that this holds for honestly generated public keys (this follows from the result of Exercise 5.12, which is easily adapted to the public key setting), we require that it holds even for adversarially chosen public keys.

To formulate this property, we assume that the encryption algorithm may output error if it detects that something is wrong with the public key (or the message, for that matter). We say the encryption scheme has **strongly unpredictable ciphertexts** if for all pk, m, and c, with $c \neq \mathsf{error}$, the probability that $E(pk, m) = c$ is negligible.

The reason for this technical requirement is that in protocol AKE3 (and other HSM secure protocols we will examine), the ciphertext is being used as an unpredictable challenge.

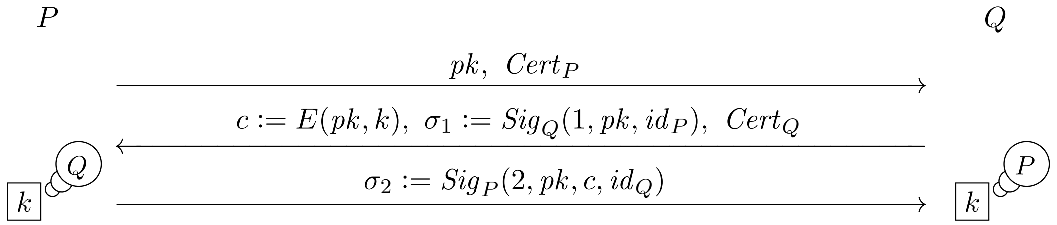
Certainly, for ElGamal-based encryption, this requirement is already met. Other encryption schemes can typically be easily adapted to ensure this requirement is met.

21.4.2 Insecure variations

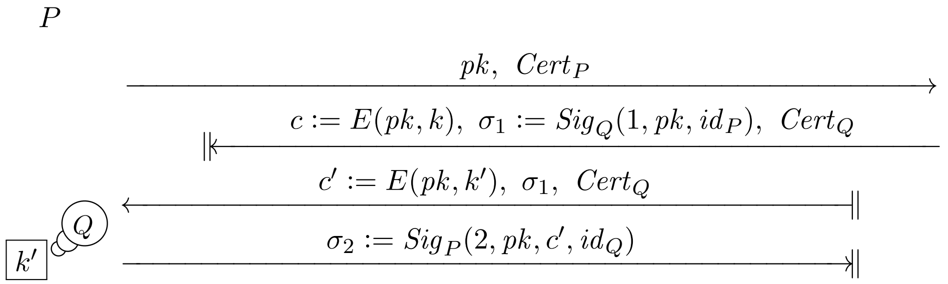
As in Section 21.2.1, we consider minor variants, showing attacks on each, and thus demonstrating that every piece of protocol AKE3 is essential. Insecure Variation 4 is the most interesting. It demonstrates an attack in the HSM model, where the adversary makes a single oracle query to the user's long-term key and can subsequently compromise many sessions.

Variation 1: do not sign c in σ_1 — a key exposure attack

Suppose Q does not sign c in σ_1. The new protocol runs as follows:


P
Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, c, id_Q)$
Q
k
P
k

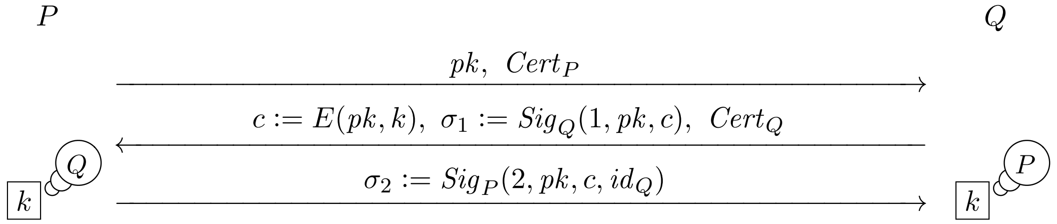
Here is a simple key exposure attack:


P
Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, id_P),\ Cert_Q$
$c' := E(pk, k'),\ \sigma_1,\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, c', id_Q)$
Q
k'

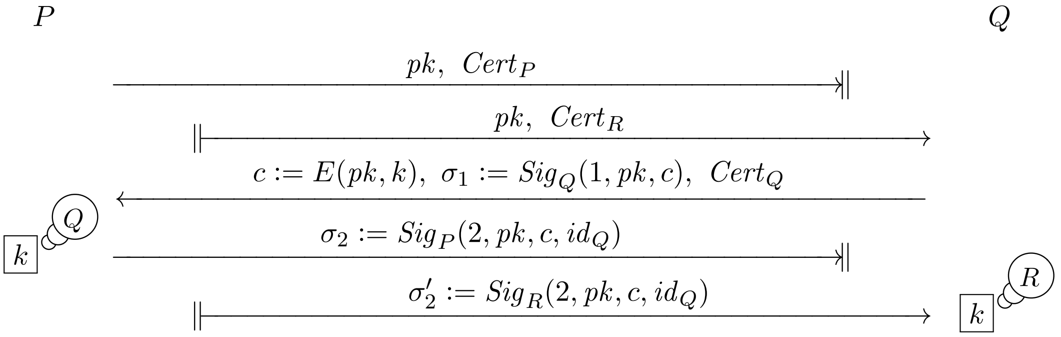
Here, the adversary generates c' by encrypting a session key k' of his choosing under pk. At the end of the protocol, P has the session key k', which is known to the adversary; however, the adversary cannot make Q terminate the protocol successfully.

Variation 2: do not sign id_P in σ_1 — an identity misbinding attack

Suppose Q does not sign id_P in σ_1. The new protocol runs as follows:


P
Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, c),\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, c, id_Q)$
Q
k
P
k

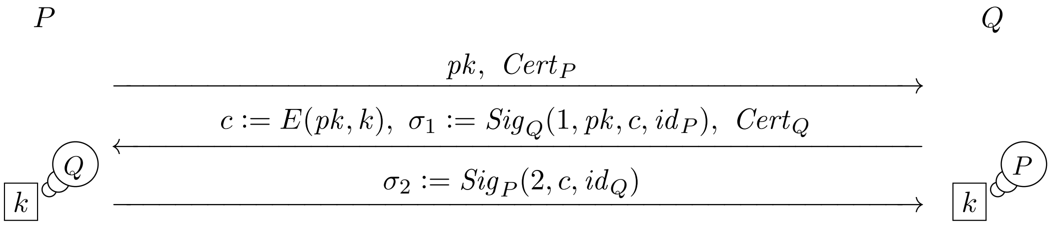
Here is an identity misbinding attack:


P
Q
$pk,\ Cert_P$
$pk,\ Cert_R$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, c),\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, c, id_Q)$
$\sigma_2' := Sig_R(2, pk, c, id_Q)$
Q
k
R
k

At the end of this attack, P and Q share the session key k, although P thinks he is talking to Q, and Q thinks he is talking to R. To carry out this attack, the adversary needs the help of a corrupt user R, who registers with the CA following the normal registration protocol.

Variation 3: do not sign pk in σ_2 — a key exposure attack

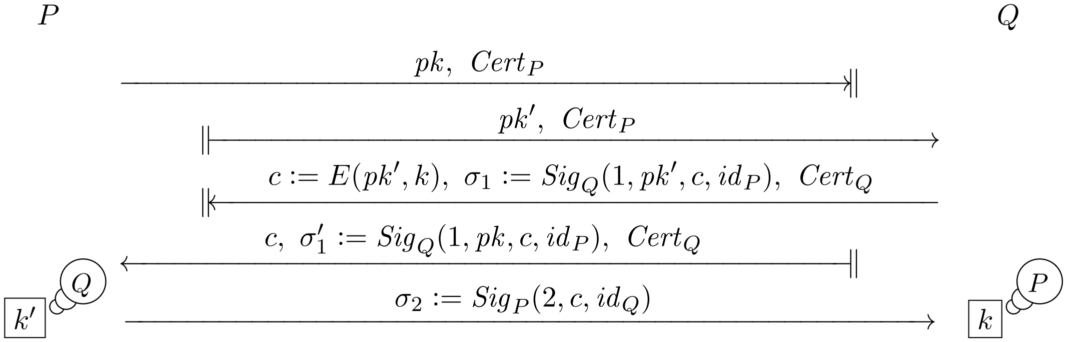
Suppose P does not sign pk in σ_2. The new protocol runs as follows:


P
Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, c, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, c, id_Q)$
Q
k
P
k

Here is a rather subtle attack. Suppose Q's signing key has been compromised and he is unaware that this has happened, and continues participating in using the session key protocol with this compromised key. Even though Q's long-term signing key is compromised, we nevertheless might expect that if Q runs the protocol with an honest user P, the session should remain secure — after all, it is Q's key that is compromised, not P's. However, in this situation, the adversary can carry out an attack as follows:

- the adversary intercepts the message $(pk, Cert_P)$ from P to Q;
- the adversary runs the key generation algorithm to obtain $(pk', sk') \overset{\text{R}}{\leftarrow} G()$, and sends the message $(pk', Cert_P)$ to Q;
- when Q responds with a message $(c, \sigma_1, Cert_P)$, where $c := E(pk', k)$ and $\sigma_1 := Sig_Q(1, pk', c, id_P)$, the adversary blocks this message and sends instead the message $(c, \sigma_1', Cert_P)$, where $\sigma_1' := Sig_Q(1, pk, c, id_P)$; it also decrypts c using sk' to obtain k;
- when P responds with a signature $\sigma_2 := Sig_P(2, c, id_Q)$, the adversary simply forwards this to Q.

The following diagram illustrates the attack:


P
Q
$pk,\ Cert_P$
$pk',\ Cert_P$
$c := E(pk', k),\ \sigma_1 := Sig_Q(1, pk', c, id_P),\ Cert_Q$
$c,\ \sigma_1' := Sig_Q(1, pk, c, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, c, id_Q)$
Q
k'
P
k

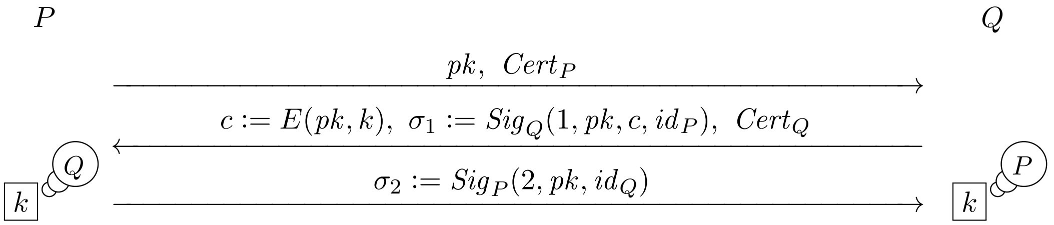
At the end of the attack, P is holding the session key k', which is unknown to the adversary; however, Q is holding the session key k, which is known to the adversary.

For this attack to work, it must be the case that even though P decrypts c with the "wrong" public key, the decryption still succeeds so that P generates the final signature σ_2. For typical semantically secure encryption schemes, this decryption will always succeed (and it still might succeed even using a CCA-secure scheme).

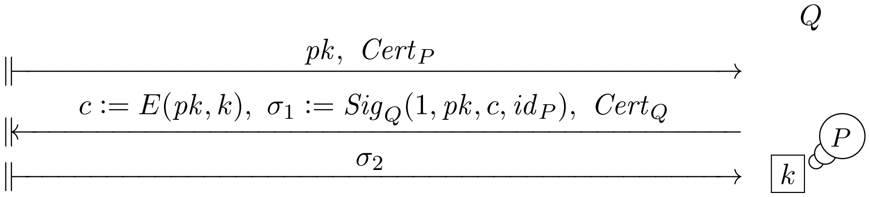
This particular type of vulnerability is called a **key compromise integrity (KCI)** vulnerability. A similar notion in a different context was discussed in Section 13.7.5.2. It is not entirely clear that one should really worry about this type of vulnerability. But some people do, and since it is easy mitigate against, it seems best to do so.

Variation 4: do not sign c in σ_2 — a key exposure attack

Suppose P does not sign c in σ_2. The new protocol runs as follows:


P
Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, c, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, id_Q)$
Q
k
P
k

Here is a key exposure attack that exploits the fact that the adversary can access P's HSM. First, the adversary runs $(pk, sk) \overset{R}{\leftarrow} G()$. It then somehow queries P's HSM to get a signature $\sigma_2 = Sig_P(2, pk, id_Q)$. In practice, all the adversary needs to be able to do is to somehow get a look at P's ephemeral secret key during an ordinary run of the protocol between P and Q. Now that the adversary has done this, he can run the following attack against Q *at any time* and *any number of times*:

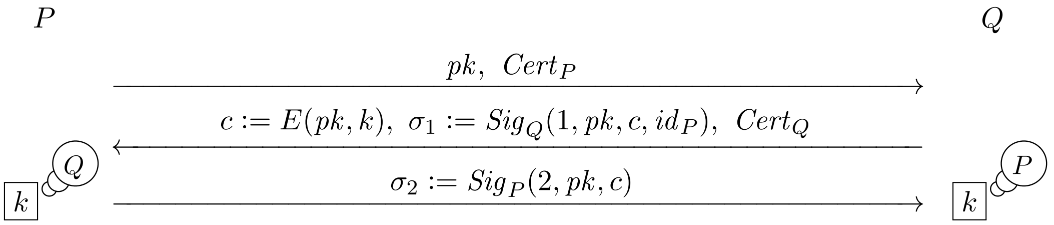

Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, c, id_P),\ Cert_Q$
σ_2
P
k

In the each run of the protocol, the adversary makes Q think he shares the key k with P, but in fact, Q shares the key k with the adversary. The adversary can get k by decrypting c using sk.

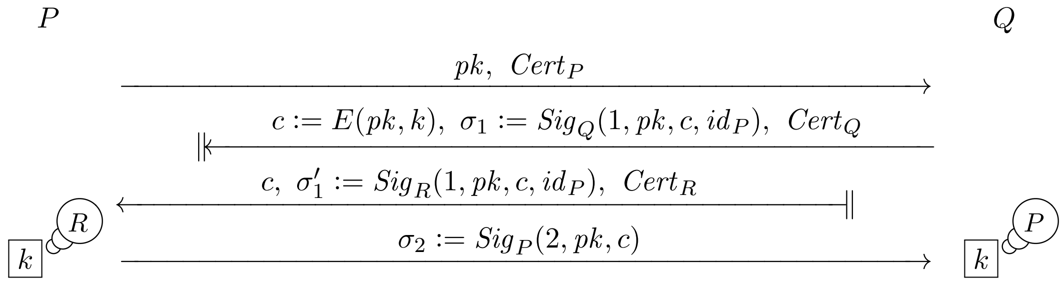
This variation is also open to a key exposure attack via a KCI vulnerability, similar to that in Variation 3. We leave this to the reader to verify.

Variation 5: do not sign id_Q in σ_2 — an identity misbinding attack

Suppose P does not sign id_Q in σ_2. The new protocol runs as follows:


P
Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, c, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, c)$
Q
k
P
k

Here is an identity misbinding attack:

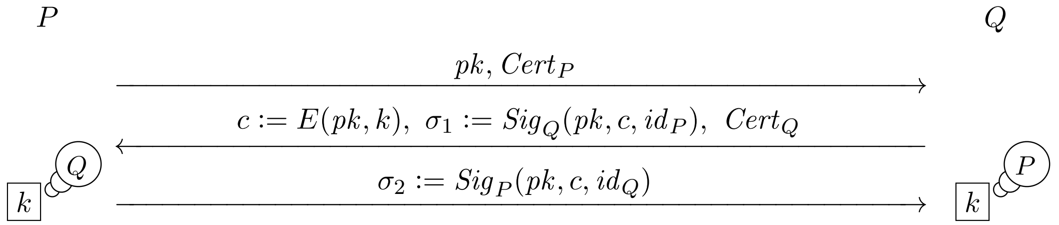


P
Q
$pk,\ Cert_P$
$c := E(pk,k),\ \sigma_1 := Sig_Q(1, pk, c, id_P),\ Cert_Q$
$c,\ \sigma_1' := Sig_R(1, pk, c, id_P),\ Cert_R$
$\sigma_2 := Sig_P(2, pk, c)$
R
k
P
k

At the end of this attack, P and Q share the session key k, although P thinks he is talking to R, and Q thinks he is talking to P. To carry out this attack, the adversary needs the help of a corrupt user R, who registers with the CA following the normal registration protocol.

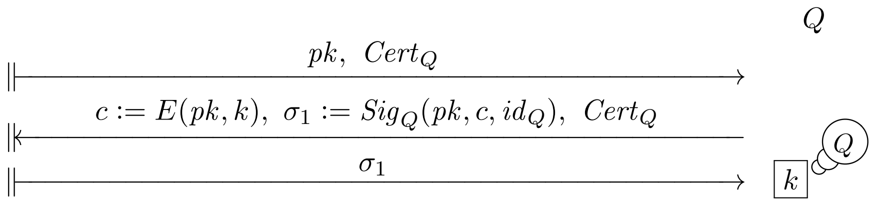
Variation 6: do not sign the 1/2 values — a key exposure attack

The reader may be wondering why we have Q include a "1" in its signed message and P include a "2" in its signed message. Suppose we leave these out, so that the protocol becomes:



P
Q
$pk, Cert_P$
$c := E(pk,k),\ \sigma_1 := Sig_Q(pk, c, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(pk, c, id_Q)$
Q
k
P
k

Here is a key exposure attack:



At the end of this attack, an instance of user Q thinks he shares a key with another instance of user Q, while in reality, he shares a key with the adversary. In some settings, it may be reasonable to assume that an instance of a user will not wish to share a key with another instance of itself, but this may not always be the case: for example, a person's phone and laptop computer may talk to each other, using the same certificate.

21.5 Identity protection

In this section, we consider an additional security requirement: **identity protection**.

Very roughly speaking, identity protection means that an adversary cannot learn the identity of either one or both the users that are running the AKE protocol. Here, the adversary could either be a passive observer, or even an active participant in the protocol.

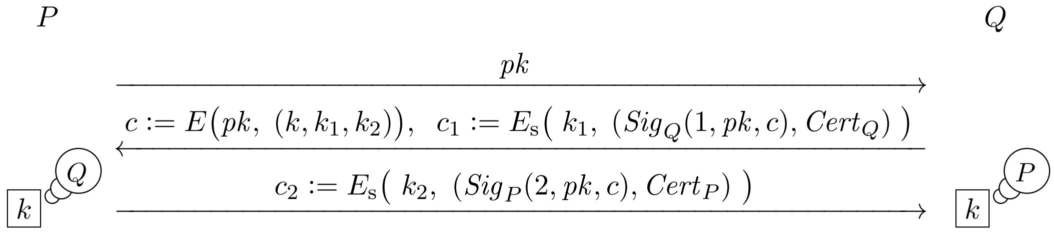

P
Q
pk
c := E(pk, (k, k₁, k₂)), c₁ := Eₛ(k₁, (Sig_Q(1, pk, c), Cert_Q))
c₂ := Eₛ(k₂, (Sig_P(2, pk, c), Cert_P))
Q
k
P
k

Figure 21.7: Protocol AKE4

In the case where the adversary is a passive observer, and the two users running the protocol are honest, the goal is to prevent the adversary from learning the identity of either one or both users. We call this **eavesdropping identity protection**. When the adversary is one of the participants, the goal is a bit more subtle: obviously, we want each user to *eventually* learn the identity of the other; however, the goal is to allow one user, say P, to withhold his identity until he is sure he is talking to someone he trusts. We say that P enjoys **full identity protection**.

As an example, consider a network of mobile devices communicating with a number of base stations. Identity protection should prevent an adversary from tracking the location of a given mobile device. Certainly, identity protection against an eavesdropping adversary will help to prevent this. However, a more aggressive adversary may try to interact with a mobile device, pretending to be a base station: although the protocol will presumably end in failure, it may have proceeded far enough for the adversary to have learned the identity of the mobile device.

In Fig. 21.7 we present a simple protocol that is HSM secure and provides identity protection, which we call protocol AKE4. This protocol makes use of a public-key encryption scheme $\mathcal{E} = (G, E, D)$ and a symmetric encryption scheme $\mathcal{E}_{\rm s} = (E_{\rm s}, D_{\rm s})$.

Here is a more detailed description of protocol AKE4:

1. P computes $(pk, sk) \xleftarrow{\mathrm{R}} G()$, and sends pk to Q;

2. Q generates a random session key k and random keys k_1, k_2 for $\mathcal{E}_{\rm s}$, and then computes

$$c \xleftarrow{\mathrm{R}} E\big(pk,\ (k, k_1, k_2)\big), \quad \sigma_1 \xleftarrow{\mathrm{R}} Sig_Q(1, pk, c), \quad c_1 \xleftarrow{\mathrm{R}} E_{\rm s}\big(\ k_1,\ (\sigma_1, Cert_Q)\ \big)$$

   and sends (c, c_1) to P;

3. P decrypts c under the key sk; if decryption fails, P aborts; otherwise, P obtains k, k_1, k_2, and decrypts c_1 under k_1; if decryption fails, P aborts; otherwise, P obtains $\sigma_1, Cert_Q$; P verifies $Cert_Q$; if the certificate is invalid, P aborts; otherwise, P extracts the identity id_Q, along with Q's public verification key, and then verifies that σ_1 is a valid signature on the message $(1, pk, c)$ under Q's public verification key; if not, P aborts; otherwise, P computes

$$\sigma_2 \xleftarrow{\mathrm{R}} Sig_P(2, pk, c), \ c_2 \xleftarrow{\mathrm{R}} E_{\rm s}\big(\ k_2,\ (\sigma_2,\ Cert_P)\ \big)$$

   and sends c_2 to Q; in addition, P terminates successfully, and outputs the session key k, and partner identity id_Q;

4. Q decrypts c_2 under the key k_2; if decryption fails, Q aborts; otherwise, Q obtains $\sigma_2, Cert_P$; Q verifies $Cert_P$; if the certificate is invalid, P aborts; otherwise, Q extracts the identity id_P,

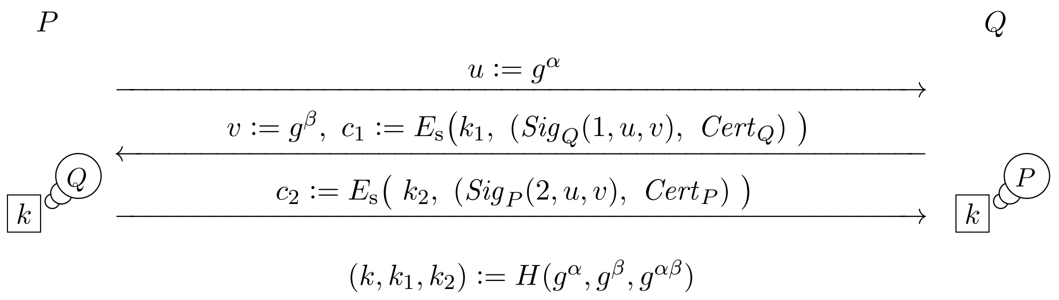

P
Q
$u := g^{\alpha}$
$v := g^{\beta},\ c_1 := E_{\mathrm{s}}\big(k_1,\ (Sig_Q(1,u,v),\ Cert_Q)\ \big)$
$c_2 := E_{\mathrm{s}}\big(\ k_2,\ (Sig_P(2,u,v),\ Cert_P)\ \big)$
Q
k
P
k
$(k, k_1, k_2) := H(g^{\alpha}, g^{\beta}, g^{\alpha\beta})$

Figure 21.8: Protocol $\texttt{AKE4}_{\mathrm{eg}}$

along with P's public verification key, and then verifies that σ_2 is a valid signature on the message $(2, pk, c)$ under P's public verification key; if not, P aborts; otherwise, Q terminates successfully, and outputs the session key k, and partner identity id_P.

Let us return to the above application to get some intuition. In using this protocol, P plays the role of a mobile device while Q plays the role of a base station. First, to an outside observer watching an interaction between P and Q, no information about the identity of either P or Q is revealed. Second, P need only reveal its identity to a base station whose identity it knows and trusts. Note that Q's identity is not protected; it is revealed to P before Q knows who P is. Hence, both parties have eavesdropping identity protection, and P has full identity protection.

HSM security. The protocol is HSM secure (where the HSM is a signing oracle), assuming $\mathcal{E}$ is semantically secure, $\mathcal{E}_{\mathrm{s}}$ provides one-time authenticated encryption, and the underlying signature schemes are secure. In fact, to prove HSM security, we only need to assume that $\mathcal{E}_{\mathrm{s}}$ provides one-time ciphertext integrity. Semantic security for $\mathcal{E}_{\mathrm{s}}$ is only needed to achieve identity protection, which is a notion that we shall not attempt to formally define.

Choice of encryption scheme. As we did for protocols AKE1–AKE3, we can implement protocol AKE4 using ElGamal encryption. This is shown in Fig. 21.8. We have streamlined the protocol somewhat, so that all of the necessary keys are derived directly from the hash function H. Again, H is modeled as a random oracle and we assume that CDH holds for $\mathbb{G}$, or H is modeled as a secure KDF and we assume that DDH holds for $\mathbb{G}$ (or we use the HDH assumption in Exercise 11.14). Just as for protocol AKE3, since we do not require CCA security, there is no need for either user to perform any explicit group membership checks.

21.6 One-sided authenticated key exchange

Up to now, we have assumed that all users must register with a CA. In fact, in many practical settings, this is too much to ask for. In this section we consider a setting in which only one of the two users running the protocol has registered with a CA.

For example, consider the situation where a customer wishes to establish a secure channel with an online bank. Here, the customer will typically not have registered with a CA, but the bank has. To be more general, let us call a user (such as a customer) without a certificate a **client**,

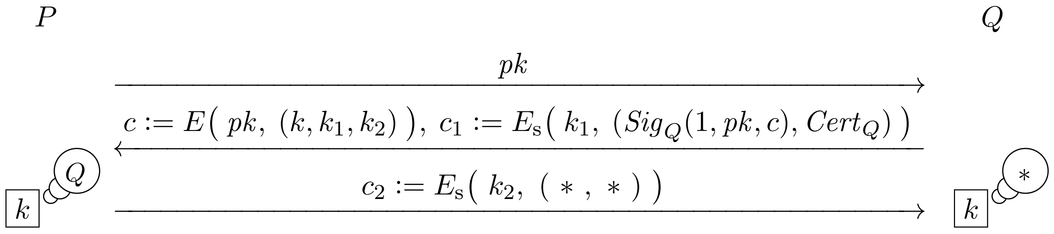

P
Q
pk
$c := E\big(\, pk,\ (k, k_1, k_2)\,\big),\ c_1 := E_{\rm s}\big(\, k_1,\ (Sig_Q(1, pk, c), Cert_Q)\,\big)$
$c_2 := E_{\rm s}\big(\, k_2,\ (\ *\ ,\ *\ )\,\big)$
Q
k
$*$
k

Figure 21.9: Protocol AKE4*

and a user (such as a bank) with a certificate a **server**. As we shall see below, one can easily construct key exchange protocols that effectively allow a client and server to establish a **one-sided authenticated secure channel**. Intuitively, when the client establishes such a channel, he effectively has a "data pipe" that connects securely to the server. For example, the client may safely transmit sensitive information (e.g., a credit card number) through the channel, confident that only the server will read this information; also, the client can be sure that any data received on this channel originated with the server. However, from the server's point of view, things are different, since a client has no certificate. When the server establishes such a channel, all he knows is that he has a "data pipe" that connects to "someone," but he has no idea who that "someone" is.

Typically, if a client wants to establish a long-term relationship with the server, he will use a one-sided authenticated secure channel to create a client account with the server, which includes, among other things, the client's user ID and password. The client can be sure that this password, and any other sensitive information, can only be read by the server. Later, in a subsequent transaction with the server, the client will set up a new one-sided authenticated secure channel with the server. To identify himself to the server, the client will transmit his user ID and password over the channel. From the client's point of view, it is safe to transmit his password over the channel, since he knows that only the server can read it. From the server's point of view, once the client's user ID and password have been verified, the server can be (relatively) confident that this "data pipe" connects securely to this client. At this point, the one-sided authenticated secure channel has been essentially upgraded to a mutually authenticated secure channel (but see Section 21.11.1). While the server may not know who the client "really is," he at least knows it is the same client that initially established a relation with the server using the given user ID.

21.6.1 A one-sided authenticated variant of AKE4

We present a one-sided authenticated variant of AKE4, which we call AKE4*, in Fig. 21.9.

Here, P is the client and Q is the server. Protocol AKE4* is to be viewed as an extension of protocol AKE4, so that some sessions provide one-sided authentication and others provide two-sided authentication. Protocol AKE4* is identical until the last flow, in which now the client sends an encryption under k_2 of a dummy message. When the server decrypts c_2 and sees this dummy message, the server assumes the client is unauthenticated.

Note that with the third flow, the security still holds under semantic security. Note that if we want a protocol that only supports one-sided authentication, the third flow is not necessary. To prove HSM security of protocol AKE4*, we need to make a stronger assumption — namely, that the public-key encryption scheme $\mathcal{E}$ is CCA secure. The reason for this is much the same as in the case of protocol AKE2 (see Section 21.3.1).

If we implement protocol AKE4* using ElGamal encryption, we get protocol AKE4$_{\text{eg}}$, but with the last flow replaced by an encryption of a dummy message, as in protocol AKE4*.

21.7 Deniability

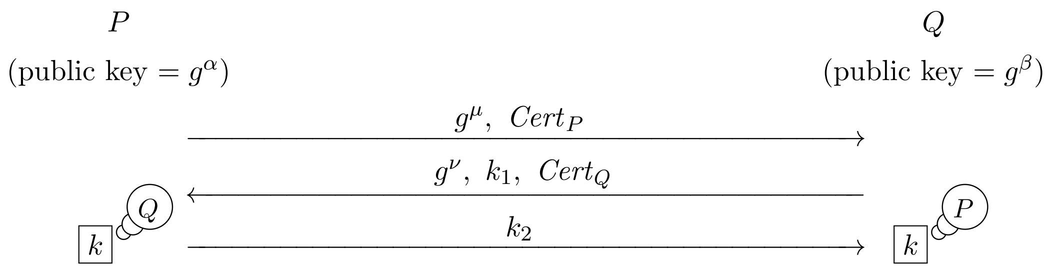
Consider protocol AKE3 in Section 21.4. In that protocol user P generates a signature $Sig_P(2, pk, c, id_Q)$. Anybody observing the protocol would see this signature, and could prove to another party that P ran the key exchange protocol with Q. For example, suppose P is a mobile device that communicates with a base station Q. From this signature, one could "prove" to a judge that the mobile device was near the base station at some point in time. As discussed at the beginning of Chapter 13, this "proof" might still be challenged in a court of law, as there are other ways this signature could have been created — for example, P's signing key could have been leaked. The same observations apply to Q in protocol AKE3, since Q generates a signature $Sig_Q(1, pk, c, id_P)$.

It would be nice if key exchange protocols would provide some form of "deniability", so that no information obtained from the execution of the protocol could be used to prove to a third party that either one or both of the users involved actually participated in the protocol.

Now consider protocol AKE4. Since all messages in the protocol are encrypted, an outsider observing the execution gets no information about the users involved, and in particular, no information that could implicate either of them. However, one still has to consider the possibility that one of the participants can implicate the other. In this protocol, neither P nor Q explicitly sign a message that contains the other's identity. In fact, it does indeed seem that this protocol provides *some* level deniability to both users. However, we know of no way to argue this in any rigorous way. Indeed, in this protocol, Q can implicate P to a certain degree, as follows. After running the protocol with P, user Q can save all of the data it generated and collected during the protocol, including the random bits r that Q used to generate the ciphertext c. At this point, Q can prove to a third party that P signed the message $(2, pk, c)$, where Q knows all of the inputs (including r) used in the computation of c. Does this by itself prove that P ran the protocol with Q? Not really, but to make Q's evidence stronger, Q could compute r, say, as a hash of $Sig_Q(pk)$, which is something that only Q can compute. The fact that P signed a message that includes a ciphertext c computed using this special r seems like strong evidence that P ran the AKE protocol with Q.

Now P could defend himself against this evidence by claiming that what actually happened is that he ran the protocol with another user R, but Q collaborated with R to make it look like P ran the protocol with Q. In particular, P could argue that R generated the ciphertext c using randomness r that was supplied by Q. Hence, Q's evidence that implicates P is not unassailable, but it is still perhaps stronger than we would like.

Deniability. In this section, we will briefly present a couple of protocols that provide a strong form of deniability for one of the two participants of the protocol. Deniability for a user P is ensured by a proof (in a fairly reasonable heuristic model) that when P engages in the protocol with Q, whatever evidence user Q is able to gather that might implicate user P, user Q could have generated on its own, without ever talking to P. This ensures that Q's evidence is unconvincing. This property holds even for a malicious Q that does not follow the protocol.



$$(k, k_1, k_2) := H\big(g^{(\alpha+\mu)(\beta+\nu)},\ g^{\mu\nu},\ g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu},\ id_P, id_Q\big)$$

Figure 21.10: Protocol AKE5

The first protocol we present provides deniability for P, but no identity protection. The second protocol additionally provides identity protection where P only reveals its identity to Q after it knows Q's identity.

21.7.1 Deniability without identity protection

Our first protocol that provides deniability for one of the two users is called protocol AKE5, and is presented in Fig. 21.10. The protocol makes use of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$, along with a hash function H. The hash function takes as input several group elements along with two user identities, and outputs $(k, k_1, k_2) \in \mathcal{K} \times \mathcal{R} \times \mathcal{R}$. Here, $\mathcal{K}$ is the set of session keys, and $\mathcal{R}$ is any super-poly-sized set. In the security analysis, we will model H as a random oracle. We will also need to assume a variant of the ICDH assumption (see Section 12.4) for $\mathbb{G}$.

Here is a more detailed description of protocol AKE5.

1. P chooses $\mu \in \mathbb{Z}_q$ at random and sends $(g^{\mu}, Cert_P)$ to Q;

2. Q verifies $Cert_P$; if the certificate is invalid, Q aborts; otherwise, Q extracts the identity id_P from $Cert_P$, along with P's public key g^{α}; Q chooses $\nu \in \mathbb{Z}_q$ at random and computes

$$(k, k_1, k_2) \leftarrow H((g^{\alpha} g^{\mu})^{(\beta+\nu)},\ (g^{\mu})^{\nu},\ g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu},\ id_P, id_Q), \tag{21.1}$$

   where g^{β} is Q's public key, and sends $(g^{\nu}, k_1, Cert_Q)$ to P;

3. P verifies $Cert_Q$; if the certificate is invalid, P aborts; otherwise, P extracts the identity id_Q from $Cert_Q$, along with Q's public key g^{β}; then P computes

$$(k, k_1, k_2) \leftarrow H((g^{\beta} g^{\nu})^{(\alpha+\mu)},\ (g^{\nu})^{\mu},\ g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu},\ id_P, id_Q); \tag{21.2}$$

   then P compares its computed value of k_1 to the value it received from Q; if these do not match, P aborts; otherwise, P sends k_2 to Q; in addition, P terminates successfully, and outputs the session key k, and partner identity id_Q;

4. Q compares its computed value of k_2 to the value it received from P; if these do not match, Q aborts; otherwise, Q terminates successfully, and outputs the session key k, and partner identity id_P.

To completely specify the protocol, we have to specify the interface for the HSM. P's HSM stores α and id_P, takes as input μ, g^β, g^ν, and id_Q, and outputs the hash value computed in (21.2). Similarly, Q's HSM stores β and id_Q, takes as input ν, g^α, g^μ, and id_P, and outputs the hash value computed in (21.1). In fact, we assume that any given user may have user instances playing both the role of P and the role of Q, so the HSM also takes as input the specified role and computes the hash accordingly.

HSM security. The HSM security of protocol AKE5 can be proved under a variant of the ICDH assumption. We will sketch some of the details later in Section 21.9.4.3.

Deniability. Since neither party signs anything, protocol AKE5 seems to provide some level of deniability for both P and Q. However, we can make an even stronger case for P's deniability. The idea is that we can efficiently simulate everything that Q sees in its interaction with P, without using P at all — this means that whatever evidence user Q is able to gather that might implicate user P, it could have generated on its own, without ever talking to user P. To build such a simulator, we need to assume that it is easy to recognize DH-triples in $\mathbb{G}$ — we can achieve this by using an elliptic curve with a pairing as in Section 15.4. Our simulator will also work by modeling H as a random oracle — in particular, our simulator must have the ability to observe Q's random oracle queries. The simulator works as follows. It begins by choosing $\mu \in \mathbb{Z}_q$ at random and sending $(g^\mu, Cert_P)$ to Q. Next, when Q sends $(g^\nu, k_1, Cert_Q)$ to P, the simulator looks at Q's random oracle queries and sees if any of these output k_1. If none exists, we can safely say that P would abort (note that if some other user R in the system made the relevant query, the input to the hash would contain id_R rather than id_Q, and so P would also abort in this case). If it does exist, the simulator checks that its input is of the right form (this is where we need the ability to recognize DH-triples). If not, we can again safely say that P would abort. Otherwise, the simulator knows the input to the hash function and therefore knows the output (k, k_1, k_2). Therefore, the simulator can generate the last flow k_1 as well as the session key k.

A few remarks about this simulator are in order:

- While it works in the random oracle model, it does not actively manipulate the random oracle as in many of our security proofs, but rather, simply observes Q's random oracle queries. This is essential in order to achieve a meaningful notion of deniability — we are trying to argue that Q could generate this view on its own, and Q does not have the ability to manipulate the random oracle.

- The simulator must be able to simulate not only the conversation but also the session key. This is because after completing the protocol, P might start using the session key. Any usage of this key, together with the conversation, could potentially be used by Q to implicate P.

- The simulation proceeds in an "online" fashion, and works even in a concurrent, multi-user environment where Q might also be interacting with other users who are completely honest and are not collaborating with Q, as well as with many instances of P itself.

21.7.2 Deniability with identity protection

We now add identity protection to protocol AKE5. This new protocol is presented in Fig. 21.11, and is called protocol AKE6. The main idea is that instead of sending g^α in the clear, user P sends

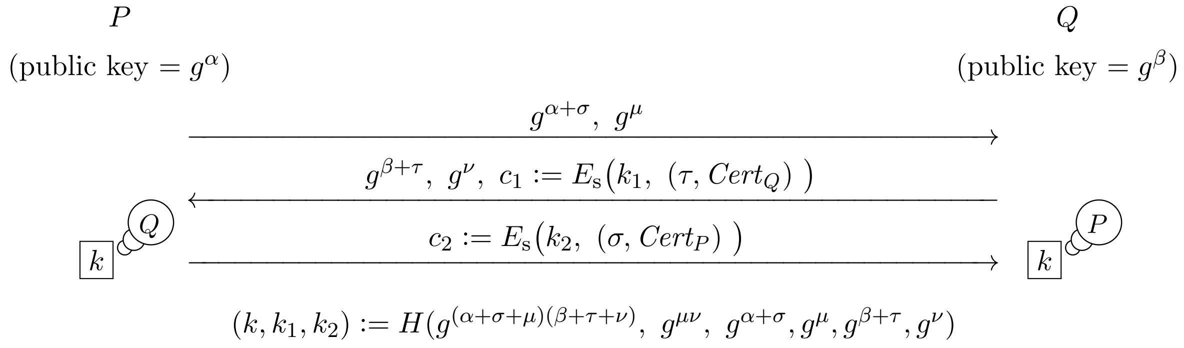

P
(public key $= g^{\alpha}$)
Q
(public key $= g^{\beta}$)
$g^{\alpha+\sigma},\ g^{\mu}$
$g^{\beta+\tau},\ g^{\nu},\ c_1 := E_{\mathrm{s}}\big(k_1,\ (\tau, \mathit{Cert}_Q)\ \big)$
$c_2 := E_{\mathrm{s}}\big(k_2,\ (\sigma, \mathit{Cert}_P)\ \big)$
Q
k
P
k
$(k, k_1, k_2) := H(g^{(\alpha+\sigma+\mu)(\beta+\tau+\nu)},\ g^{\mu\nu},\ g^{\alpha+\sigma}, g^{\mu}, g^{\beta+\tau}, g^{\nu})$

Figure 21.11: Protocol `AKE6`

a "blinded" value $g^{\alpha'}$, where $\alpha' = \alpha + \sigma$, and then later sends the exponent σ along with Cert_P (which contains P's public key g^{α}) in encrypted form; Q can then verify the blinding by checking that $g^{\alpha} \cdot g^{\sigma} = g^{\alpha'}$. User Q carries out a symmetric strategy.

Here is a more detailed description of protocol `AKE6`. It makes use of a symmetric encryption scheme $\mathcal{E}_{\mathrm{s}} = (E_{\mathrm{s}}, D_{\mathrm{s}})$.

1. P chooses $\sigma, \mu \in \mathbb{Z}_q$ at random, sets $\alpha' := \alpha + \sigma$, and sends $(g^{\alpha'}, g^{\mu})$ to Q;

2. Q chooses $\tau, \nu \in \mathbb{Z}_q$ at random, sets $\beta' := \beta + \tau$, and computes

$$\begin{aligned} (k, k_1, k_2) &\leftarrow H((g^{\alpha'} g^{\mu})^{(\beta'+\nu)},\ (g^{\mu})^{\nu},\ g^{\alpha'}, g^{\mu}, g^{\beta'}, g^{\nu}), \\ c_1 &\xleftarrow{\mathrm{R}} E_{\mathrm{s}}\big(k_1,\ (\tau, \mathit{Cert}_Q)\ \big), \end{aligned} \tag{21.3}$$

   and sends $(g^{\beta'}, g^{\nu}, c_1)$ to P;

3. P computes

$$\begin{aligned} (k, k_1, k_2) &\leftarrow H((g^{\beta'} g^{\nu})^{(\alpha'+\mu)},\ (g^{\nu})^{\mu},\ g^{\alpha'}, g^{\mu}, g^{\beta'}, g^{\nu}), \\ c_2 &\xleftarrow{\mathrm{R}} E_{\mathrm{s}}\big(k_2,\ (\sigma, \mathit{Cert}_P)\ \big); \end{aligned} \tag{21.4}$$

   P decrypts c_1 using the key k_1; if decryption fails, P aborts; otherwise, P obtains (τ, Cert_Q) and verifies Cert_Q; if the certificate is invalid, P aborts; otherwise, P extracts the identity id_Q from Cert_Q, along with Q's public key g^{β}; P verifies that $g^{\beta} \cdot g^{\tau} = g^{\beta'}$; if this fails, P aborts; otherwise, P sends c_2 to Q; in addition, P terminates successfully, and outputs the session key k, and partner identity id_Q;

4. Q decrypts c_2 using the key k_2; if decryption fails, Q aborts; otherwise, Q obtains $(\sigma, \mathit{Cert}_P)$ and verifies Cert_P; if the certificate is invalid, Q aborts; otherwise, Q extracts the identity id_P from Cert_P, along with P's public key g^{α}; Q verifies that $g^{\alpha} \cdot g^{\sigma} = g^{\alpha'}$; if this fails, Q aborts; otherwise, Q terminates successfully, and outputs the session key k, and partner identity id_P.

To completely specify the protocol, we have to specify the interface for the HSM. P's HSM stores α, takes as input σ, μ, $g^{\beta'}$, and g^{ν}, and outputs the hash value in (21.4). Similarly, Q's HSM stores β, takes as input τ, ν, $g^{\alpha'}$, and g^{μ}, and outputs the hash value in (21.3).

HSM security. The HSM security of protocol AKE6 can be proved under a variant of the I^2CDH assumption (see Section 13.7.4), and assuming $\mathcal{E}_{\mathrm{s}}$ provides one-time authenticated encryption. We will sketch some of the details later in Section 21.9.4.4.

Deniability. We can also argue for P's deniability using simulation strategy similar to that which we used for protocol AKE5. Let us return to our example of a mobile device communicating with a number of base stations, which we discussed in the context of identity protection in Section 21.5. If a mobile device plays the role of P and the base station plays the role of Q, then protocol AKE6 provides very strong privacy guarantees for the mobile device:

- the mobile device's identity is not visible to an outside observer;
- the mobile device only reveals its identity to the base station after the base station reveals and authenticates its own identity;
- the mobile device can deny that it interacted with any particular base station.

On the limits of deniability. Deniability is a very slippery concept. In reality, many steps in the conversation between P and Q may provide Q with evidence that it interacted with P. For example, Q might ask P to supply information that is not publicly available, such as P's bank account number or birth date. Q could later use this information to argue to a third party that it interacted with P. The point of protocols AKE5 and AKE6 is to show that the AKE protocol itself need not give Q evidence that it interacted with P.

21.8 Channel bindings

Sometimes, it is helpful if a higher-level application using a session can refer to a session by a globally unique name. In key exchange protocols, this is called a **channel binding**, although some authors also call this a "session ID".

To add this feature to a key exchange protocol, we require when instances of users P and Q finish a successful run of a session key protocol, in addition to a session key k and partner IDs, the key exchange protocol provides them with a *channel binding.*

The security property that a key exchange protocol with channel bindings should provide can be roughly stated as follows:

> *Two user instances securely share a key if and only if they share the same channel binding.*

We can easily add secure channel bindings to all of the protocols discussed so far:

- Protocol AKE1: (id_P, id_Q, r, c)
  - Protocol $\mathtt{AKE1}_{\mathrm{eg}}$: (id_P, id_Q, r, v)
- Protocol AKE2: (id_P, id_Q, pk, c)
  - Protocol $\mathtt{AKE2}_{\mathrm{eg}}$: (id_P, id_Q, u, v)
- Protocol AKE3: (id_P, id_Q, pk, c)

  - Protocol $\texttt{AKE3}_{\text{eg}}$: (id_P, id_Q, u, v)

- Protocols $\texttt{AKE4}$ and $\texttt{AKE4}^*$: (pk, c)
  - Protocols $\texttt{AKE4}_{\text{eg}}$ and $\texttt{AKE4}^*_{\text{eg}}$: (u, v)
- Protocol $\texttt{AKE5}$: $(id_P, id_Q, g^{\mu}, g^{\nu})$
- Protocol $\texttt{AKE6}$: $(g^{\alpha+\sigma}, g^{\mu}, g^{\beta+\tau}, g^{\nu})$

We will briefly discuss an application of channel bindings later in Section 21.11.1.

21.9 Formal definitions

Defining security for AKE protocols is not so easy. In fact, there is currently no widely accepted standard definition of security. Nevertheless, in this section, we present a definition of security that captures the most basic elements of secure AKE in a reasonable way, and which is consistent in the most essential aspects with various definitions in the literature. We start with the formal definition of static security, which does not model either PFS or HSM security. Later, we discuss how to modify the definition to model these notions.

The definition presented here works with either an offline TTP (e.g., a CA), or an online TTP. Intuitively, our definition of security captures the idea that each instance of a user should obtain a fresh session key, which from an adversary's point of view, should essentially appear to be uniformly distributed over some specified key space $\mathcal{K}$, and independent of all other session keys obtained by other honest user instances (belonging to either this or other users). However, there are some wrinkles which complicate things:

- The whole point of an AKE protocol is to generate a session key that is *shared* between two user instances; therefore, the goal that *every* session key should be fresh is not quite right: some pairs of session keys can and should be equal.
- A user may establish a session key directly with a corrupt user, in which case, this key cannot possibly be expected to be fresh.

Syntactically, an AKE protocol specifies a set $\mathcal{K}$ of session keys, and three algorithms:

- The **TTP algorithm**, which dictates the logic of the TTP over the lifetime of the system.[1]
- The **user registration algorithm**, which is an interactive protocol algorithm (see Section 18.1) that takes as input a user ID. This algorithm specifies an interactive subprotocol that registers the named user with the TTP, and which establishes that user's *long-term secret key*.
- The **session key establishment algorithm**, which is an interactive protocol algorithm (see Section 18.1) that takes as input a user's ID and long-term secret key (as initialized by the *user registration algorithm*). This algorithm specifies an interactive subprotocol that is used to establish a session key with another user. To break symmetry, this algorithm also takes as input a value $role \in \{\mathsf{left}, \mathsf{right}\}$. Upon termination, this subprotocol outputs either abort, or outputs (pid, k), where pid is a partner ID and $k \in \mathcal{K}$ is a session key.

[1] Formally, the TTP should be an *efficient interface*, as in Definition 2.12.

Our goal is to present an attack game consisting of two experiments. Experiment 0 represents a real attack, while Experiment 1 represents an idealization of an attack. As usual, we want these two experiments to be indistinguishable from the point of view of an adversary. In each experiment, the adversary is interacting with a challenger, which is slightly different in each experiment.

The challenger plays the roles of the TTP and all the honest users. Formally speaking, the adversary is completely arbitrary; however, one can think of the adversary as really playing three distinct roles at once:

- the network,
- a higher level protocol, such as an email system, being run by honest users, and which makes use of the session keys obtained by instances of honest users, and
- a truly malicious attacker, coordinating with corrupt users.

Because our formal adversary also plays the role of higher level protocols that use session keys, we allow the adversary free access to session keys obtained by honest users, which may at first seem counter-intuitive, since one normally thinks of session keys as being hidden from the adversary. See Section 21.9.1 for more about how to understand and use the definition.

Experiment 0. At the beginning of the attack, the challenger initializes the internal state of the TTP. Now the adversary can make a number of queries to the challenger:

Register honest user: This query constructs a new honest user U, with an identity $U.\mathsf{id}$ specified by the adversary. Behind the scenes, the challenger runs an instance of the *registration protocol* with the TTP. This protocol is run in a secure fashion: the adversary cannot see or influence any messages sent between the honest user and the TTP. The TTP will update its internal state, if necessary, and the challenger sets $U.\mathsf{ltk}$ to the user's long-term secret key.

Register corrupt user: Here, the adversary essentially is allowed to run the *registration protocol* directly with the TTP, using an identity of his choice.

Initialize honest user instance: This query constructs a new user instance I, which is associated with a previously registered honest user $I.\mathsf{user} = U$, which is specified by the adversary. The adversary also supplies a role $I.\mathsf{role} \in \{\mathsf{left}, \mathsf{right}\}$. The challenger initializes the internal state of an honest user instance, using the ID $I.\mathsf{user.id}$, the long-term secret key $I.\mathsf{user.ltk}$, and the given role $I.\mathsf{role}$.

Deliver protocol message: The adversary specifies a *running* honest user instance I along with an *incoming message* m_{in} that is to be processed by that instance. The challenger processes the message, updating the internal state of the instance, producing an *outgoing message* m_{out} along with a *status value*, which is one of

- $(\mathsf{finished}, I.\mathsf{pid}, I.\mathsf{sk})$, indicating successful termination with partner ID $I.\mathsf{pid}$ and session key $I.\mathsf{sk}$,
- $\mathsf{aborted}$, indicating unsuccessful termination,
- $\mathsf{running}$, indicating not yet terminated.

Both m_{out} and the status value — including the partner ID and session key, in the case of a finished status — are handed to the adversary.

Deliver TTP message: This is only used in the online TTP setting. The adversary gives a message m_{in} that is intended for the TTP. The challenger processes the message according to the logic of the TTP. Any resulting message m_{out} is given to the adversary.

There is one further restriction: the adversary is never allowed to register an honest user's ID as a corrupt user, and is never allowed to register an honest user ID more than once.

That completes the formal description of Experiment 0. Thus, the challenger maintains the internal state of the TTP, the honest users, and all the honest user instances. The challenger does not maintain any state information for corrupt users: this is the adversary's responsibility. However, the challenger does maintain a list of user IDs registered as corrupt users, and refuses any registration requests that would register a given ID as both an honest and corrupt user.

Note that the adversary is never allowed to obtain the long-term secret key of an honest user or the internal state of an honest user instance. Because of this restriction, this definition of security does not capture the notions of PFS or HSM security. Later, we will show how to tweak the definition to model these notions.

Before defining Experiment 1, we have to introduce the notion of a **partner function**, which will be required to establish security. Basically, a partner function is a mechanism which establishes which user instances actually share a key, and which user instances hold session keys that must be treated as inherently vulnerable. To be as flexible as possible, the partner function may depend on the protocol itself, but it must be efficiently computable as a function of the *network communication log*.

For technical reasons relating to protocol composability, this log does not include everything the adversary sees. We shall define the log as a sequence of entries, generated as follows.

- For a *register corrupt user* query, the entry $(\mathsf{corruptUser}, id)$, where id is the ID of the corrupt user.

- For an *initialize honest user instance* query, the entry $(\mathsf{init}, I, I.\mathsf{role})$ is appended to the log. Note that the log entry *does not* include $I.\mathsf{pid}$, as this value is not a part of the normal network traffic.

- For a *deliver protocol message* query, the entry $(\mathsf{deliver}, I, m_{\text{in}}, m_{\text{out}}, \mathit{status})$ is appended to the log, where $\mathit{status} \in \{\mathsf{finished}, \mathsf{aborted}, \mathsf{running}\}$. Note that the log entry *does not* include $I.\mathsf{pid}$ or $I.\mathsf{sk}$ when $\mathit{status} = \mathsf{finished}$, as these values are not a part of the normal network traffic.

- For a *deliver TTP message* query, the entry $(\mathsf{deliverTTP}, m_{\text{in}}, m_{\text{out}})$ is added to the log. Recall that in the offline TTP setting of this chapter, there are no *deliver TTP message* queries, and hence no log entries of this form.

These are the only entries in the log.

The partner function will be computed by the challenger in Experiment 1 each time a *deliver protocol message* query to a running user instance I results in successful termination with a status of (finished, I.pid, I.sk). The input to the partner function consists the communication log in the attack game up to that point in time. The output of the partner function classifies the user instance as either

- *vulnerable*,
- *fresh*, or
- *connected to* J, where J is some finished honest user instance.

The meaning of this classification will become clear momentarily.

Before defining Experiment 1, we need one other concept. We call two finished honest user instances I and J **compatible** if

- $I.\mathsf{pid} = J.\mathsf{user.id}$,
- $J.\mathsf{pid} = I.\mathsf{user.id}$, and
- $I.\mathsf{role} \neq J.\mathsf{role}$.

Recall that our intuitive notion of authenticity translates into saying that if two users share a key, they should be compatible.

Experiment 1. The challenger's actions are precisely the same as in Experiment 0, except that when a user instance I finishes with a session key $I.\mathsf{sk}$, instead of giving the adversary $I.\mathsf{sk}$, the challenger instead gives the adversary an *effective session key* $I.\mathsf{esk}$, which is determined (in part) by the classification of I by the partner function.

vulnerable: If $I.\mathsf{pid}$ belongs to a corrupt user, then

$$I.\mathsf{esk} \leftarrow I.\mathsf{sk};$$

that is, the effective session key is set to the actual session key. Otherwise, $I.\mathsf{esk} \leftarrow \mathsf{error}$.

fresh: If $I.\mathsf{pid}$ belongs to some registered user (honest or corrupt), then

$$I.\mathsf{esk} \xleftarrow{\text{R}} \mathcal{K};$$

that is, the effective session key is chosen at random. Otherwise, $I.\mathsf{esk} \leftarrow \mathsf{error}$.

connected to J: If J is compatible with I, and J is fresh and no other user instance previously connected to it, then

$$I.\mathsf{esk} \leftarrow J.\mathsf{esk};$$

that is, the effective session key is set to that of this honest user instance's "partner." Otherwise, $I.\mathsf{esk} \leftarrow \mathsf{error}$.

That finishes the description of Experiments 0 and 1. If W_b is the event that an adversary $\mathcal{A}$ outputs 1 in Experiment b, we define $\mathcal{A}$'s **advantage** with respect to a given AKE protocol Π and partner function pf to be

$$\text{sKE}\mathsf{adv}[\mathcal{A}, \Pi, \mathsf{pf}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|.$$

Definition 21.1 (statically secure authenticated key exchange). *An AKE protocol* Π *is* ***statically secure*** *if there exists an efficiently computable partner function* pf *such that for all efficient adversaries* $\mathcal{A}$*, the value* $\text{sKE}\mathsf{adv}[\mathcal{A}, \Pi, \mathsf{pf}]$ *is negligible.*

Remark 21.1. Note that in Experiment 1, the effective session key is set to error if certain validity conditions do not hold. However, since keys never take the value error in Experiment 0, security implies that these validity conditions must hold with overwhelming probability in both experiments. Also, for many protocols, these validity conditions are easily computable as a function of the communication log. However, this is not always the case — for example, protocols that provide identity protection, such as protocol AKE4 in Section 21.5. □

Remark 21.2. For a secure protocol, there is typically very little, if any, choice in the definition of a partner function. In the literature, this partnering is sometimes achieved by other means, whereby a specific partner function is defined that must work for all secure protocols. For example, some authors use the notion of "matching conversations", which roughly means that two user instances are partners if their conversations match up bit-by-bit. This can sometimes be overly restrictive, as it may require the use of strongly secure signatures to ensure that conversations are highly non-malleable. Instead of matching conversations, some authors use a notion of "session IDs" to specify a partner function. This can also be problematic, especially when defining security of protocols that provide only one-sided authentication, as in Section 21.6.1. □

A correctness requirement. To be complete, in addition to defining the *security* of an AKE protocol Π with respect to a partner function pf, we should also define a *correctness* requirement. Roughly speaking, such a requirement says that if an adversary interacts with the challenger as in Experiment 0 above, then for any pair of honest user instances, if the adversary faithfully transmits all protocol messages between these two instances (and the TTP, if necessary), then (with overwhelming probability) these two honest user instances will both terminate the protocol successfully, and one will be connected to the other via the partner function. Note that this correctness requirement, together with the security requirement, guarantees that (with overwhelming probability) these two honest user instances must share the same session key.

21.9.1 Understanding the definition

Our formal security definition may seem a bit unintuitive at first. For example, one might ask, why is the adversary given the session keys when the goal of the protocol is supposedly to protect the session keys?

To gain a better understanding of the definition, it is useful to see how to use the definition to analyze the security of a higher-level protocol that uses a secure AKE protocol. We focus here on the most important application of AKE protocols, namely, to establish secure channels.

So suppose that we use a secure AKE protocol as follows. Once a user instance finishes the key exchange protocol, it uses its session key to implement a secure channel using authenticated encryption, as in Chapter 9. If we want this channel to be bi-directional, we will need an authenticated encryption for each one-directional channel. We can derive all of the necessary keys from the session key by using the session key as a seed for a PRG or a key for a PRF. The user instance may now send and receive messages on its bi-directional channel, using these keys.

To analyze the security of this "secure session protocol", we can proceed as follows. We can think of each user instance as using an abstract interface, similar to that in Section 9.3. After the user instance starts the protocol, if the AKE protocol terminates successfully, the user instance obtains a partner ID. Next, the user instance can place messages in its out-box and retrieve messages from its in-box, as in Section 9.3, but where now, the channel is bi-directional, so this user instance is both a sender and a receiver. In this implementation of the abstract interface, the logic of the out-box and in-box is implemented using an authenticated encryption scheme and the keys derived from the session key.

An attacker in this setting has complete control of the network, and can attempt to interfere with the protocol messages used to implement the AKE protocol as well as the protocol messages used to implement the secure channel.

Now, starting with this "real" implementation, we can work towards a more "ideal" implementation.

The first step is to use the security property of the AKE protocol, which allows us to replace real session keys with effective session keys, according to the classification of user instances. Some user instances will be "vulnerable" if they attempt to communicate with a corrupt user. Each remaining user instance will have a truly random session key, which is shared with its partner user instance (if any). Let us call these user instances "safe". In our classification system, "safe" user instances are either "fresh" or "connected".

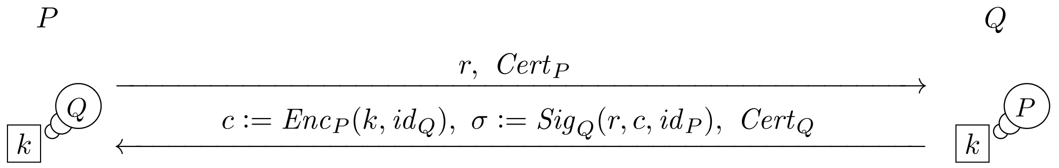
To justify this step, we need to apply our definition of a secure AKE. In this analysis, the adversary $\mathcal{B}$ attacking the AKE protocol comprises not only our original attacker $\mathcal{A}$, but also the logic of the honest users, outside of the internals of the AKE protocol itself. This adversary $\mathcal{B}$ does see and use the session keys, but it is only an artifact of the proof, and does not correspond to any "real world" attacker. (Also note that the original attacker $\mathcal{A}$ can compute the partner function based on the network communication log, so $\mathcal{A}$ "knows" who is talking to whom.)

The second step is to replace the real implementation of each channel connecting two "safe" user instances by the ideal implementation discussed in Section 9.3. In this ideal implementation, ciphertexts are just handles and messages magically jump from sender to receiver.

21.9.2 Security of protocol AKE1

We now consider the security of the AKE protocol AKE1.

Recall protocol AKE1:


P
Q
$r,\ Cert_P$
$c := Enc_P(k, id_Q),\ \sigma := Sig_Q(r, c, id_P),\ Cert_Q$
Q
k
P
k

An instance of P playing on the left-hand side has the left role, and an instance of Q playing on the right-hand side has the right role. We will adopt this convention in analyzing all the protocols in this chapter. When this protocol terminates, the instance of P has session key k and partner ID id_Q, and the instance of Q has session key k and partner ID id_P.

Theorem 21.1. *Protocol* AKE1 *is a statically secure authenticated key exchange protocol, assuming: the size of the nonce space $\mathcal{R}$ is super-poly, the underlying public-key encryption scheme is CCA secure, and the underlying signature schemes used by the users and CA are secure.*

Proof sketch. The first step in the proof is to specify a partner function. In this and other proofs in this chapter, it is convenient to define a "loosely matching" relation for cryptographic objects. We say two certificates are loosely matching if their IDs are the same. We say two signatures are always loosely matching. We say two nonces, public keys, or ciphertexts are loosely matching if they are identical. Two tuples of cryptographic objects are loosely matching if each of their components are loosely matching.

When a right instance J finishes, we look at the ID of the certificate it receives. If it belongs to a corrupt user, we classify J as *vulnerable.* Otherwise, we classify J as *fresh.* (Note that this is the only time in this chapter we need to use the $\mathsf{corruptUser}$ entries in the communication log.)

When a left instance I finishes, if the two flows it sees loosely match the two flows seen by some right instance J, we classify I as *connected to J.* Otherwise, we classify I as *vulnerable.*

We now sketch why this partner function works.

First, observe that the classification of a right instance J as *vulnerable* is always valid, since by definition, J's partner ID belongs to a corrupt user.

Next, consider a left instance I that successfully finishes the protocol. We consider two cases.

Case 1: I has a partner ID belonging to some honest user. We claim that there is a unique right instance J who sees two flows that loosely match those seen by I, and that I and J are compatible; this follows from the security of the signature schemes and the fact that ciphertexts do not repeat (all this happens with overwhelming probability, of course). This I will be classified as *connected to J*, and this classification will be valid; this follows from the fact that I and J are compatible, and nonces do not repeat.

Case 2: Otherwise. We claim that flows seen by I cannot loosely match the two flows seen by any right instance J — otherwise, I's partner ID would match the ID of J, and we would be back in Case 1. Thus, I is classified as *vulnerable*, and this is a valid classification. We also argue that the ciphertext decrypted by I could not have been generated by any *fresh* right instance J under I's public key. Indeed, if it were, then J's ID would be embedded in the ciphertext, and since that ID belongs to an honest user, by the logic of the protocol, we must be back in Case 1. This last assertion, together with the CCA security of encryption, implies that fresh session keys can be replaced by random keys without detection. □

21.9.3 Modeling perfect forward secrecy

We now show how to modify our static security definition to model perfect forward secrecy, that is, PFS security. The changes are actually quite minimal.

First, we add a new type of query:

Compromise user: The adversary specifies an honest user U. The challenger gives the long-term secret key $U.\mathsf{ltk}$ to the adversary. Although we still say that U is an honest user, we say U is **compromised** from this point on.

This query models the compromise of an honest user's long-term secret key.

The second change is to the computation of effective session keys in Experiment 1. Specifically, we change the rule for computing the effective session key for a *vulnerable* user instance I as follows:

vulnerable: If $I.\mathsf{pid}$ belongs to a corrupt user *or a compromised honest user,* then

$$I.\mathsf{esk} \leftarrow I.\mathsf{sk}.$$

Otherwise, $I.\mathsf{esk} \leftarrow \mathsf{error}$.

These are the only changes. We denote by $\mathrm{pfsKE}\mathsf{adv}[\mathcal{A}, \Pi, \mathsf{pf}]$ an adversary $\mathcal{A}$'s advantage against a protocol Π in this modified attack game, with respect to a given partner function pf.

Definition 21.2 (PFS secure key exchange). *An AKE protocol* Π *is* ***PFS secure*** *if there exists an efficiently computable partner function* pf *such that for all efficient adversaries* $\mathcal{A}$*, the value* $\text{pfsKE}\mathsf{adv}[\mathcal{A}, \Pi, \mathsf{pf}]$ *is negligible.*

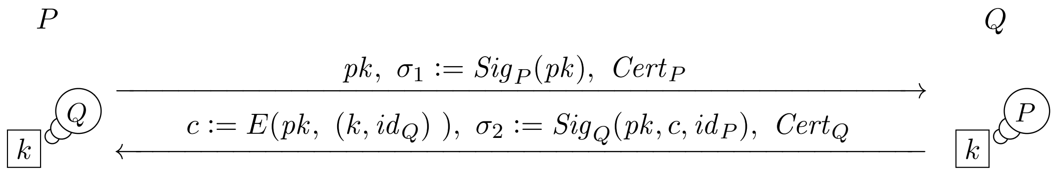
Remark 21.3. Even after an honest user is compromised, the adversary may continue delivering messages to user instances belonging to that user. We must allow this, as the adversary does not obtain the internal state of these user instances, and so cannot "take over" the execution of these user instances. For consistency and simplicity, we also allow the adversary to continue to initialize user instances belonging to a compromised user. □

Remark 21.4. Observe that the *vulnerable* classification of I is valid only if $I.\mathsf{pid}$ belongs to a corrupt user or a compromised honest user. It is *not* valid if only $I.\mathsf{user}$ itself is compromised. This means our definition of security implies security against so-called KCI (key compromise impersonation) attacks (see Variation 3 on p. 874 in Section 21.4.2). □

21.9.3.1 Security of protocol AKE2

We now consider the security of protocol AKE2. We shall prove that this protocol satisfies the definition of security with forward secrecy presented in Section 21.9.3.

Recall protocol AKE2:


P
Q
$pk, \ \sigma_1 := Sig_P(pk), \ \ Cert_P$
$c := E(pk, \ (k, id_Q) \ ), \ \sigma_2 := Sig_Q(pk, c, id_P), \ \ Cert_Q$
Q
k
P
k

Theorem 21.2. *Protocol* AKE2 *is a PFS secure authenticated key exchange protocol assuming: the underlying public-key encryption scheme is CCA secure, and the underlying signature schemes used by the users and CA are secure.*

Proof sketch. The first step in the proof is to specify a partner function.

When a right instance J finishes, we classify it as *fresh* if the first flow it sees loosely matches the first flow sent by some left instance (see proof of Theorem 21.1). Otherwise, we classify J as *vulnerable.*

When a left instance I finishes, we see if the two flows it sees loosely match the two flows seen by some right instance J. If so, we classify I as *connected to* J. Otherwise, we classify I as *vulnerable.*

To prove that this works, one has to prove two claims.

1. When some right instance J finishes with a partner ID that belongs to an uncompromised honest user, then the first flow it sees loosely matches the flow sent by some left instance I.

2. When some left instance I finishes with a partner ID that belongs to an uncompromised honest user, then there is a unique right instance J for which the two flows seen by both loosely match.

Proving that both of these hold (with overwhelming probability) follows from the security of the signature schemes. The rest of the proof is very similar to that of Theorem 21.1. □

21.9.4 Modeling HSM security

We now show how to modify our PFS security definition so as to model HFS security. Again, the changes are actually quite minimal.

Starting with the PFS security model, we add a new type of query to the PFS that models adversarial access to the HSM:

Access HSM: The adversary specifies an honest user U and a value x. The challenger responds with $f(U.\mathsf{ltk}, x)$. Here f is the function defining the interface to the HSM and $U.\mathsf{ltk}$ is the long-term secret key of user U.

The second change is to the computation of effective session keys in Experiment 1. Specifically, we change the rule for computing the effective session key for a *vulnerable* user instance I as follows:

vulnerable: If $I.\mathsf{pid}$ belongs to a corrupt user or a compromised honest user, or both of the following conditions hold:

(i) $I.\mathsf{pid}$ belongs to an honest user U whose HSM was accessed at some point in time between when I was activated and when I finished, and

(ii) the total number of adversarial HSM accesses on user U is greater than the number of other vulnerable user instances J with $J.\mathsf{pid} = I.\mathsf{pid}$,

then

$$I.\mathsf{esk} \leftarrow I.\mathsf{sk}.$$

Otherwise, $I.\mathsf{esk} \leftarrow \mathsf{error}$.

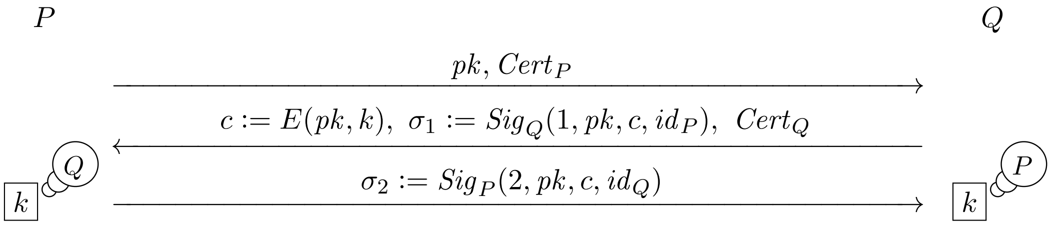
Conditions (i) and (ii) above correspond to high-level security goals for HSM security we introduced in Section 21.4. Together, they say that a single HSM query can be used to classify only a single user instance as vulnerable, and the query must happen while that user instance is running.

We denote by hsmKE$\mathsf{adv}[\mathcal{A}, \Pi, \mathsf{pf}]$ an adversary $\mathcal{A}$'s advantage against a protocol Π in this modified attack game, with respect to a given partner function pf.

Definition 21.3 (HSM secure authenticated key exchange). *An AKE protocol* Π *is* ***HSM secure*** *if there exists an efficiently computable partner function* pf *such that for all efficient adversaries* $\mathcal{A}$*, the value* hsmKE$\mathsf{adv}[\mathcal{A}, \Pi, \mathsf{pf}]$ *is negligible.*

21.9.4.1 Security of protocol `AKE3`

Recall protocol `AKE3`:


P
Q
$pk,\ Cert_P$
$c := E(pk, k),\ \sigma_1 := Sig_Q(1, pk, c, id_P),\ Cert_Q$
$\sigma_2 := Sig_P(2, pk, c, id_Q)$
k
Q
k
P

Theorem 21.3. *Protocol* `AKE3` *is an HSM secure authenticated key exchange protocol assuming: the underlying public-key encryption scheme is semantically secure and has strongly unpredictable ciphertexts (see Section 21.4.1), and the underlying signature schemes used by the users and CA are secure.*

Proof sketch. We first define the partner function.

When a left instance I finishes, we classify it as *fresh* if the first two flows it sees loosely match the first two flows seen by some right instance (see proof of Theorem 21.1). Otherwise, we classify I as *vulnerable.*

When a right instance J finishes, we see if the first two flows it sees loosely match the first two flows seen by some left instance I. If so, we classify J as *connected to* I. Otherwise, we classify J as *vulnerable.*

To prove that this works, one has to prove two claims.

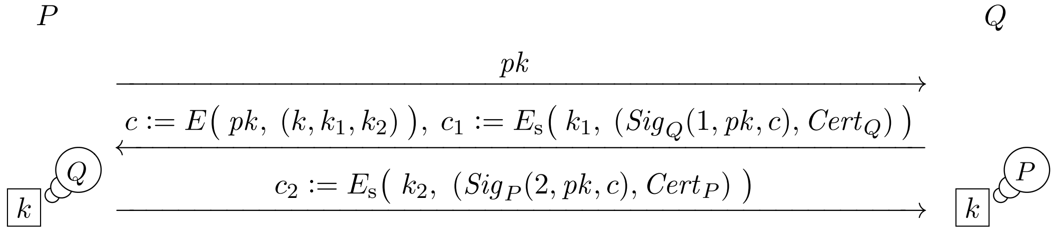
1. When some left instance I finishes with a partner ID that belongs to an uncompromised honest user whose HSM has not been queried during the protocol execution to sign the relevant message, then there is a unique right instance J for which the two flows seen by both loosely match.

2. When some right instance J finishes with a partner ID that belongs to an uncompromised honest user whose HSM has not been queried during the protocol execution to sign the relevant message, then there is a unique left instance I for which the two flows seen by both loosely match.

Proving that both of these hold (with overwhelming probability) follows from the security of the signature schemes. From these two claims, the rest of the proof follows fairly easily. □

21.9.4.2 Security of protocol AKE4

Recall protocol AKE4:



P
Q
pk
$c := E\big(\, pk,\ (k, k_1, k_2)\,\big),\ c_1 := E_{\mathrm{s}}\big(\, k_1,\ (Sig_Q(1, pk, c), Cert_Q)\,\big)$
$c_2 := E_{\mathrm{s}}\big(\, k_2,\ (Sig_P(2, pk, c), Cert_P)\,\big)$
Q
k
P
k

Theorem 21.4. *Protocol* AKE4 *is an HSM secure authenticated key exchange protocol assuming: the underlying public-key encryption scheme is semantically secure and has strongly unpredictable ciphertexts (see Section 21.4.1), the underlying symmetric encryption scheme provides one-time ciphertext integrity, and the underlying signature schemes used by the users and CA are secure.*

Proof sketch. We first define the partner function.

When a left instance I finishes, we classify it as *fresh* if the first two flows it sees *exactly* match the first two flows seen by some right instance. Otherwise, we classify I as *vulnerable.*

When a right instance J finishes, we see if the first two flows it sees *exactly* match the first two flows seen by some left instance I. If so, we classify J as *connected to* I. Otherwise, we classify J as *vulnerable.*

To prove that this works, one has to prove two claims.

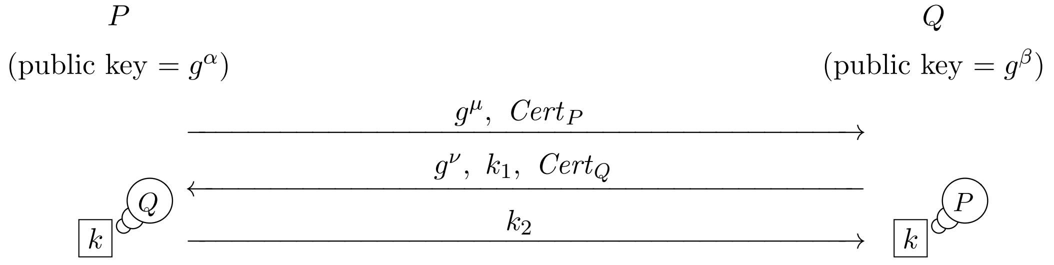
1. When some left instance I finishes with a partner ID that belongs to an uncompromised honest user whose HSM has not been queried during the protocol execution to sign the relevant message, then there is a unique right instance J for which the two flows seen by both exactly match.

2. When some right instance J finishes with a partner ID that belongs to an uncompromised honest user whose HSM has not been queried during the protocol execution to sign the relevant message, then there is a unique left instance I for which the two flows seen by both exactly match, and moreover, the third flow seen by both exactly match as well.

From these two claims, the rest of the proof follows fairly easily. □

21.9.4.3 Security of protocol AKE5

Recall protocol AKE5:



P
(public key = g^{α})
Q
(public key = g^{β})
g^{μ}, $Cert_P$
g^{ν}, k_1, $Cert_Q$
k_2
Q
k
P
k

$$(k, k_1, k_2) := H(g^{(\alpha+\mu)(\beta+\nu)},\ g^{\mu\nu},\ g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu},\ id_P, id_Q)$$

To prove security of AKE5, we need the ICDH assumption for $\mathbb{G}$ (see Section 12.4). Also, if an instance of user P can establish a session key with another instance of user P (which is something that we do allow in general), then we need an additional assumption. We can define this assumption using a slight modification of Attack Game 12.3 — namely, instead of choosing $\alpha, \beta \in \mathbb{Z}_q$ at random, the challenger chooses $\alpha \in \mathbb{Z}_q$ at random and sets $\beta \leftarrow \alpha$. We call this the ICDH+ assumption. Intuitively, it means that it is hard to compute g^{α^2}, given g^{α} and access to "DH-decision oracle" that recognizes DH-triples of the form $(g^{\alpha}, \cdot, \cdot)$.

Theorem 21.5. *Protocol* AKE5 *is an HSM secure authenticated key exchange protocol under the ICDH and ICDH+ assumptions, if we model H as a random oracle (and if the set $\mathcal{R}$ in which k_1 and k_2 lie is super-poly-sized).*

Proof sketch. Unlike all of the other AKE protocols presented so far, protocol AKE5 does not use a signature scheme for authentication.

We sketch why this authentication mechanism works. Suppose that a left user instance I belonging to a user P that terminates successfully with a partner ID that belongs to an uncompromised honest user Q whose HSM has not been queried during the execution of the protocol with the value g^{μ} as an input. We want to show that the first two flows seen by I loosely match the first two flows seen by a right instance J. If there is no such instance J, then the adversary must have himself queried the random oracle at the relevant point

$$(g^{(\alpha+\mu)(\beta+\nu)},\ g^{\mu\nu},\ g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu},\ id_P, id_Q). \tag{21.5}$$

We show how this adversary can be used to solve the CDH problem for the problem instance (g^{β}, g^{μ}). To this end, we run the attack game knowing α. Dividing the first component of (21.5) by the second, we can compute

$$g^{\alpha\beta+\alpha\nu+\mu\beta}.$$

Since α is known, we can divide out the terms involving α, which allows us to compute $g^{\mu\beta}$. However, we have to take into account the fact that the right user's HSM may be accessed throughout this attack (directly by the adversary, as well as by honest user instances). We can still use the adversary to solve the CDH problem for the problem instance (g^{β}, g^{μ}), provided we also are given access to an oracle that recognizes DH-triples of the form $(g^{\beta}, \cdot, \cdot)$ — using this DH-decision oracle, we can manage the random oracle much as in the proof of Theorem 12.4. This is why we need the ICDH assumption.

We have to take into account that we could have $Q = P$, in which case the above argument has to be modified. One can make a similar argument as above, but now we use the adversary to compute g^{α^2} given g^{α} as well as access to an oracle that recognizes DH-triples of the form $(g^{\alpha}, \cdot, \cdot)$. This is why we need the ICDH+ assumption. We leave the details of this to the reader.

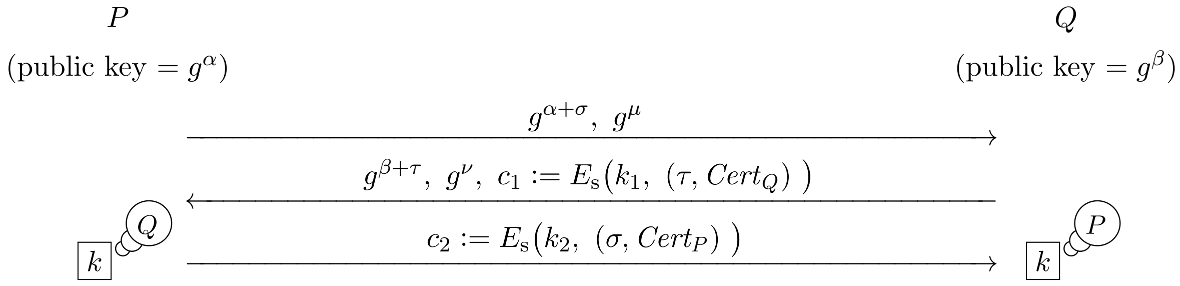
The above argument shows that this mechanism ensures authenticity for P. A similar argument shows that it provides authenticity for Q.

Once we have established these authenticity properties, we can also argue that replacing a real session key by a random session key is not detectable, unless the adversary can compute $g^{\mu\nu}$ given g^{μ} and g^{ν}. This makes use of the ordinary CDH assumption.

□

21.9.4.4 Security of protocol AKE6

Recall protocol AKE6:


P
(public key = g^{α})
Q
(public key = g^{β})
$g^{\alpha+\sigma},\ g^{\mu}$
$g^{\beta+\tau},\ g^{\nu},\ c_1 := E_{\mathrm{s}}\big(k_1,\ (\tau, Cert_Q)\big)$
$c_2 := E_{\mathrm{s}}\big(k_2,\ (\sigma, Cert_P)\big)$
Q
k
P
k

$$(k, k_1, k_2) := H(g^{(\alpha+\sigma+\mu)(\beta+\tau+\nu)},\ g^{\mu\nu},\ g^{\alpha+\sigma}, g^{\mu}, g^{\beta+\tau}, g^{\nu})$$

Theorem 21.6. *Protocol* AKE6 *is an HSM secure authenticated key exchange protocol under the* I^{2}CDH *and ICDH+ assumptions, if we model H as a random oracle, and if $\mathcal{E}_{\mathrm{s}}$ provides one-time authenticated encryption.*

Proof sketch. The main ideas of the proof are the same as in the proof of Theorem 21.5. We need the stronger I^{2}CDH assumption to prove that the protocol provides authenticity for Q. In the analysis, in using the adversary to compute $g^{\alpha\nu}$ from g^{α} and g^{ν}, we will have to be able to recognize DH-triples of the form $(g^{\nu}, \cdot, \cdot)$, in addition to DH-triples of the form $(g^{\alpha}, \cdot, \cdot)$. □

21.9.5 Modeling one-sided authentication

We briefly sketch how we can modify our various security definitions to accommodate one-sided authentication. Formally speaking, there is a special honest user with the special user ID `anonymous`. Any unauthenticated user instance is considered to be an instance belonging to this user. In addition, for any of the models (static, PFS, or HSM), we always allow a user instance I to be classified as *vulnerable* if $I.\mathsf{pid} = \mathsf{anonymous}$. That is, we change the rule for computing the effective session key for a *vulnerable* user instance I as follows:

vulnerable: If $I.\mathsf{pid} = \texttt{anonymous}$ or ..., then

$$I.\mathsf{esk} \leftarrow I.\mathsf{sk}.$$

Otherwise, $I.\mathsf{esk} \leftarrow \mathsf{error}$.

Here, "..." is the corresponding condition, which depends on the model (static, PFS, or HSM).

Theorem 21.4 holds for protocol $\texttt{AKE4}^*$ as well. The partner function is identical, and the proof outline is basically the same.

21.9.6 Modeling channel bindings

We introduced the notion of *channel bindings* in Section 21.8. All of our security models can be easily accommodated to model this feature.

In Experiment 1 of the attack game, when computing an effective session key for a user instance, the challenger checks if the channel binding for this user instance would violate the following global constraint:

> *Two user instances are classified as connected to each other if and only if they share the same channel binding.*

This is just a restatement of the informal constraint given in Section 21.9.6 in the language of our formal model. If this constraint is violated, the effective session key is set to error. Otherwise, it is set using the normal rules.

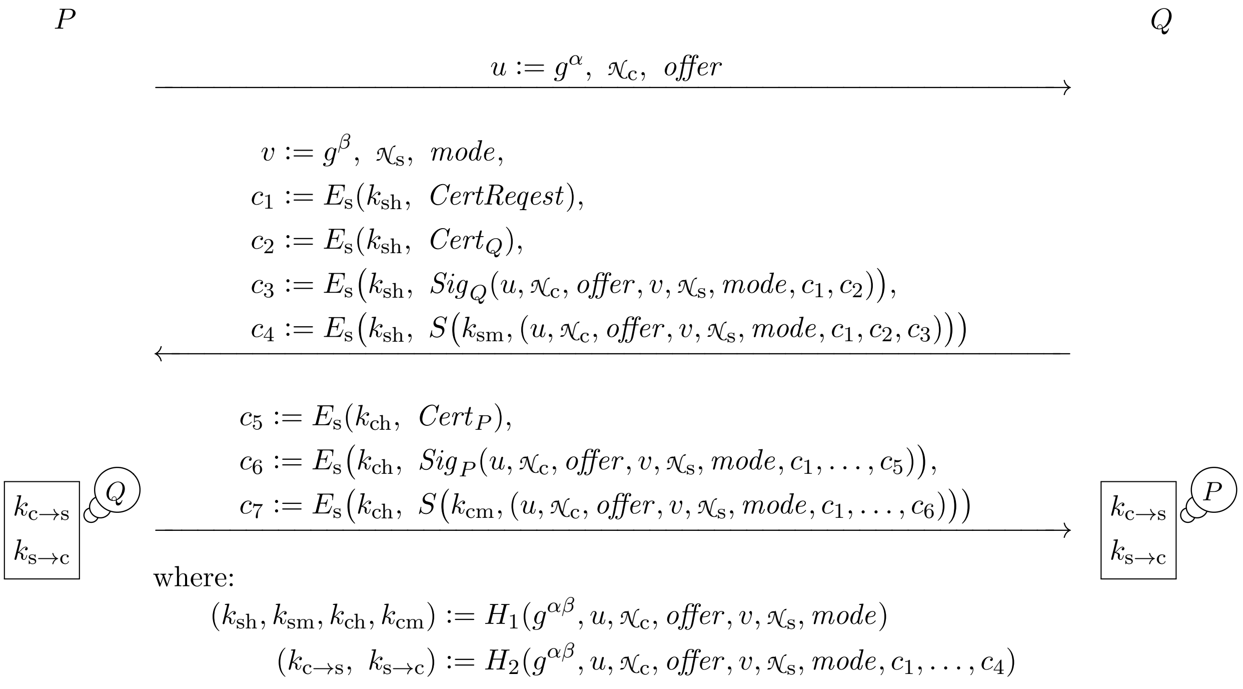
The security theorems for all the protocols we have studied in this chapter carry over unchanged if we use the channel bindings defined in Section 21.8. Note that for all of the schemes that use a public-key encryption scheme, we require that the scheme has strongly unpredictable ciphertexts (see Section 21.4.1). This property ensures that the probability that multiple invocations of the encryption algorithm output the same ciphertext twice is negligible, even if public keys are adversarially chosen.

For technical reasons relating to protocol composability, one other technical requirement for channel bindings is that a user instance should compute its channel binding based only on its role, and on the the information it sends and receives over the network during the execution of the key exchange protocol (all of this information is present in the network communication log). All of the channel bindings defined in Section 21.8 satisfy this requirement.

21.10 Case study: TLS session setup

In Section 9.8 we saw the TLS record protocol which is used to encrypt traffic between two parties after they set up a secure session by generating shared secret keys. In this section we describe the authenticated key exchange protocol used in TLS to set up a secure session. We only look at the key exchange protocol used in TLS 1.3 which was introduced in 2017.

For consistency with the notation in this chapter, we let P play the role of the client and Q play the role of the server. P and Q wish to set up a secure session.



$$u := g^{\alpha},\ \mathcal{N}_{\mathrm{c}},\ \mathit{offer}$$

$$\begin{aligned}
v &:= g^{\beta},\ \mathcal{N}_{\mathrm{s}},\ \mathit{mode},\\
c_1 &:= E_{\mathrm{s}}(k_{\mathrm{sh}},\ \mathit{CertReqest}),\\
c_2 &:= E_{\mathrm{s}}(k_{\mathrm{sh}},\ \mathit{Cert}_Q),\\
c_3 &:= E_{\mathrm{s}}\big(k_{\mathrm{sh}},\ \mathit{Sig}_Q(u, \mathcal{N}_{\mathrm{c}}, \mathit{offer}, v, \mathcal{N}_{\mathrm{s}}, \mathit{mode}, c_1, c_2)\big),\\
c_4 &:= E_{\mathrm{s}}\big(k_{\mathrm{sh}},\ S\big(k_{\mathrm{sm}}, (u, \mathcal{N}_{\mathrm{c}}, \mathit{offer}, v, \mathcal{N}_{\mathrm{s}}, \mathit{mode}, c_1, c_2, c_3)\big)\big)
\end{aligned}$$

$$\begin{aligned}
c_5 &:= E_{\mathrm{s}}(k_{\mathrm{ch}},\ \mathit{Cert}_P),\\
c_6 &:= E_{\mathrm{s}}\big(k_{\mathrm{ch}},\ \mathit{Sig}_P(u, \mathcal{N}_{\mathrm{c}}, \mathit{offer}, v, \mathcal{N}_{\mathrm{s}}, \mathit{mode}, c_1, \ldots, c_5)\big),\\
c_7 &:= E_{\mathrm{s}}\big(k_{\mathrm{ch}},\ S\big(k_{\mathrm{cm}}, (u, \mathcal{N}_{\mathrm{c}}, \mathit{offer}, v, \mathcal{N}_{\mathrm{s}}, \mathit{mode}, c_1, \ldots, c_6)\big)\big)
\end{aligned}$$

$$\boxed{\begin{matrix} k_{\mathrm{c}\to\mathrm{s}} \\ k_{\mathrm{s}\to\mathrm{c}} \end{matrix}} \qquad \boxed{\begin{matrix} k_{\mathrm{c}\to\mathrm{s}} \\ k_{\mathrm{s}\to\mathrm{c}} \end{matrix}}$$

where:

$$\begin{aligned}
(k_{\mathrm{sh}}, k_{\mathrm{sm}}, k_{\mathrm{ch}}, k_{\mathrm{cm}}) &:= H_1(g^{\alpha\beta}, u, \mathcal{N}_{\mathrm{c}}, \mathit{offer}, v, \mathcal{N}_{\mathrm{s}}, \mathit{mode})\\
(k_{\mathrm{c}\to\mathrm{s}},\ k_{\mathrm{s}\to\mathrm{c}}) &:= H_2(g^{\alpha\beta}, u, \mathcal{N}_{\mathrm{c}}, \mathit{offer}, v, \mathcal{N}_{\mathrm{s}}, \mathit{mode}, c_1, \ldots, c_4)
\end{aligned}$$

Figure 21.12: The TLS 1.3 key exchange protocol

TLS 1.3. The essence of the TLS key exchange protocol is shown in Fig. 21.12. The protocol supports both one-sided authentication and mutual authentication. The figure shows TLS mutual authentication. In the figure, $(E_{\mathrm{s}}, D_{\mathrm{s}})$ is a symmetric encryption scheme that provides authenticated encryption, such as AES-128 in GCM mode. Algorithm S refers to a MAC signing algorithm, such as HMAC-SHA256. Algorithms $\mathit{Sig}_P(\cdot)$ and $\mathit{Sig}_Q(\cdot)$ sign the provided data using P's or Q's signing keys, respectively. Finally, the hash functions H_1, H_2 are used to derive symmetric keys. They are built from HKDF (Section 8.10.5) with a hash function such as SHA256.

The symmetric encryption scheme $(E_{\mathrm{s}}, D_{\mathrm{s}})$ and the hash function in HMAC and HKDF to use is determined by a negotiation in the first two messages of the protocol. TLS negotiates these algorithms, rather than hard code a specific choice, because some countries and organizations may prefer to use different algorithms. Some may not trust the algorithms standardized by the US National Institute of Standards (NIST). Nevertheless, all implementations are required, at a minimum, to support AES-128 in GCM mode and SHA256, as well as a few other common ciphers.

The protocol. The TLS key exchange in Fig. 21.12 works as follows. In the first flow, the client P sends to the server Q a group element $u := g^{\alpha}$, a nonce $\mathcal{N}_{\mathrm{c}}$, and an "offer". The offer is a message that specifies the group and the symmetric encryption and hash schemes that the client is willing and able to use. In fact, the client can provide several groups in his offer, providing corresponding group elements for each group. In TLS 1.3, the groups that may be used are constrained to be one of several pre-defined groups, which include both elliptic curves and subgroups of finite fields. The symmetric encryption and hash scheme are also constrained to be one of a small number of pre-defined schemes. The offer includes a list of encryption/hash scheme pairs, in order of preference.

After receiving the first flow, the server Q examines the "offer" sent by the client. It verifies that the group (or groups) preferred by the client coincides with the group (or groups) that the server is able and willing to use. It also selects an encryption scheme $(E_{\mathrm{s}}, D_{\mathrm{s}})$ and a hash function from the offer that it is willing and able to use, if any. If the server is unable to find a compatible group and encryption/hash schemes, the server may send a special "retry" request to the client, but we will not discuss this aspect of the protocol here. Otherwise, the server responds to the client with a flow that consists of a group element $v := g^{\beta}$, a nonce $\mathcal{N}_{\mathrm{s}}$, and a "mode" message which indicates the parameter choices (group, encryption/hash scheme) made by the server. This flow also contains several encrypted messages:

- A special "certificate request" message, which is only sent if the server wishes to authenticate the client. If present, this message specifies the type of certificates the server will accept.
- The server's certificate (which includes the server's signature verification key).
- A signature (under server's signing key) on the conversation so far.
- A tag computed using HMAC (see Section 8.7) on the conversation so far.

The key k_{sh} used to encrypt these messages and the key k_{sm} used in applying HMAC are derived from the data

$$g^{\alpha\beta},\ u = g^{\alpha},\ \mathcal{N}_{\mathrm{c}},\ \mathit{offer},\ v = g^{\beta},\ \mathcal{N}_{\mathrm{s}},\ \mathit{mode} \tag{21.6}$$

using HKDF.

After receiving the second flow, the client responds with a flow that consists of several encrypted messages:

- The client's certificate (which includes the client's signature verification key). This message is only sent if the server requested client authentication.
- A signature (under client's signing key) on the conversation so far. This message is only sent if the server requested client authentication.
- A tag computed using HMAC (see Section 8.7) on the conversation so far.

The key k_{ch} used to encrypt these messages and the key k_{cm} used in applying HMAC are derived from (21.6) using HKDF.

The session computed by both client and server consists of two keys, $k_{\mathrm{c}\to\mathrm{s}}$ and $k_{\mathrm{s}\to\mathrm{c}}$, which are derived from

$$g^{\alpha\beta},\ u,\ \mathcal{N}_{\mathrm{c}},\ \mathit{offer},\ v,\ \mathcal{N}_{\mathrm{s}},\ \mathit{mode},\ c_1, \dots, c_4$$

using HKDF. In the record protocol, $k_{\mathrm{c}\to\mathrm{s}}$ is used to encrypt messages sent from the client to the server, and $k_{\mathrm{s}\to\mathrm{c}}$ is used to encrypt messages sent from the server to the client, as discussed in Section 9.8. The client may "piggyback" record protocol messages along with the third flow of the key exchange protocol.

The TLS 1.3 protocol also allows the server to "piggyback" record protocol messages along with the second flow of the key exchange protocol. This is why c_5, c_6, c_7 are not included in the computation of the session keys $k_{\mathrm{c}\to\mathrm{s}}$ and $k_{\mathrm{s}\to\mathrm{c}}$. Of course, any record protocol messages piggybacked on the second flow are being sent to an unauthenticated client. This mode of operation falls out of the scope of our formal models of secure key exchange.

Identity protection. Notice that P's and Q's identities (contained in $Cert_P$ and $Cert_Q$) are encrypted and not visible to an eavesdropper. Moreover, P does not transmit its identity until it verifies Q's identity. This is done to ensure eavesdropping identity protection for both parties and full identity protection for P, as discussed in Section 21.5. Identity protection is a feature of TLS 1.3 that was not present in earlier versions of the protocol.

We note, however, that identity protection is only of limited value in a Web environment. The reason is that modern web browsers include Q's identity (the DNS domain name of the web server) in the clear in the first message to the server. This data is included in a field called **server name indication** or **SNI**. It is needed when multiple domains are hosted on a single server. The SNI field tells the server what certificate $Cert_Q$ to use in its response. Without the SNI field, the server would not know what domain the client is requesting and the connection will fail.

Security. TLS 1.3, as described in Fig. 21.12, is very similar to our protocol AKE4 (specifically, its instantiation using ElGamal encryption, protocol $\mathtt{AKE4}_{\mathrm{eg}}$) and its one-sided variant, protocol $\mathtt{AKE4}^*$ (see Figures 21.7, 21.8, and 21.9). It can be proved to be HSM secure in the random oracle model under roughly the same assumptions as we make for protocol AKE4 (in particular, the CDH assumption), provided we restrict the protocol so that Q does not employ the special mode of operation in which it starts using the session key before it receives the third flow. To securely support that mode of operation, instead of the CDH assumption, we need to invoke the stronger ICDH assumption (for much the same reason as we need CCA security and the ICDH assumption in the analysis of protocol $\mathtt{AKE2}_{\mathrm{eg}}$).

Additional features. TLS 1.3 provides several features beyond basic key exchange. One feature, called **exported keys**, is used by applications that need additional key material for encrypting application data outside of the secure session. The TLS 1.3 key exchange provides an *exporter master secret* that is known to both P and Q. It can be given to a higher-level application for its own use. This key is effectively independent of all the keys used to secure the TLS session, but provides the same security guarantees.

Another feature is the ability to **update traffic keys**. In a long lived TLS session it may be desirable to "roll" the session keys $k_{\mathrm{s}\to\mathrm{c}}$ and $k_{\mathrm{c}\to\mathrm{s}}$ forward, so that a compromise of these keys does not expose earlier session data. At any time after the key exchange completes, either P or Q can request a key update by sending a *KeyUpdate* message. Roughly speaking, this causes the following update to take place

$$k'_{\mathrm{s}\to\mathrm{c}} := H(k_{\mathrm{s}\to\mathrm{c}}) \qquad \text{and} \qquad k'_{\mathrm{c}\to\mathrm{s}} := H(k_{\mathrm{c}\to\mathrm{s}})$$

where H is implemented using HKDF. Both sides then delete the earlier keys $k_{\mathrm{s}\to\mathrm{c}}$ and $k_{\mathrm{c}\to\mathrm{s}}$ and encrypt all subsequent traffic using $k'_{\mathrm{s}\to\mathrm{c}}$ and $k'_{\mathrm{c}\to\mathrm{s}}$. This mechanism provides some degree of forward secrecy within the session.

21.10.1 Authenticated key exchange with preshared keys

TLS provides support for an abbreviated session setup protocol if P and Q already share a secret key. This abbreviated key exchange, called a **pre-shared key** handshake, is more efficient than a full key-exchange. It skips some of the computationally expensive key exchange steps in Fig. 21.12.

Usually, a pre-shared key is the result of a previous TLS key exchange between P and Q. A pre-shared key can be generated at anytime after a TLS key exchange completes. Server Q sends a *new-session-ticket* message to client P, over the secure channel, and this message tells P to compute and store a pre-shared key *psk*. Using the notation in Fig. 21.12, this pre-shared key is computed as:

$$psk := H_3(g^{\alpha\beta}, u, \mathcal{N}_{\rm c}, \mathit{offer}, v, \mathcal{N}_{\rm s}, \mathit{mode}, c_1, \ldots, c_7, \mathcal{N}_{\rm t}),$$

where H_3 is a key derivation function based on HKDF, and $\mathcal{N}_{\rm t}$ is a random nonce from Q provided in the *new-session-ticket* message. The server can send multiple *new-session-ticket* messages, each with a fresh nonce $\mathcal{N}_{\rm t}$, causing the client to calculate and store multiple pre-shared keys.

A pre-shared key can also be generated by an out-of-band mechanism, such as manually loading a random key into the client and server. For example, if the client is a mobile device, say a car, this key can be loaded into the car at the time of sale, and also loaded into the cloud service that the car connects to.

Every pre-shared key has an **identity**, which is a bit string that identifies the key. This identity is either provided manually, in case the pre-shared key is loaded manually, or it is provided as part of the *new-session-ticket* message from Q.

When a client P wants to establish a TLS session with Q using a pre-shared key, it includes the key's identity in the *offer* that it sends (in the clear) in the first flow in Fig. 21.12. In fact, the client can include several identities corresponding to multiple pre-shared keys that it is willing to use. Server Q can choose to reject all them and do a full handshake as in Fig. 21.12, or it can choose one of the provided identities and include it in the *mode* that it sends in its response to the client.

If the server selects one of the provided identities, then P and Q do an abbreviated key exchange. The key exchange proceeds as in Fig. 21.12, except that the ciphertexts c_1, c_2, c_3, c_5, c_6 are not computed, not sent, and not included in the HMAC and HKDF computations. Instead, the derived keys are computed as

$$\begin{aligned}(k_{\rm sh}, k_{\rm sm}, k_{\rm ch}, k_{\rm cm}) &:= H_4(psk,\ g^{\alpha\beta}, u, \mathcal{N}_{\rm c}, \mathit{offer}, v, \mathcal{N}_{\rm s}, \mathit{mode}) \\ (k_{\rm c\to s},\ k_{\rm s\to c}) &:= H_5(psk,\ g^{\alpha\beta}, u, \mathcal{N}_{\rm c}, \mathit{offer}, v, \mathcal{N}_{\rm s}, \mathit{mode}, c_4)\end{aligned} \tag{21.7}$$

where H_4 and H_5 are key derivation functions based on HKDF. Notice that this abbreviated key exchange is much faster than a full key exchange because no signatures are computed by either party. This is a significant performance savings for a busy server.

Forward secrecy. A key exchange based on a pre-shared key (optionally) includes the quantities $u := g^{\alpha}$ and $v := g^{\beta}$, as in Fig. 21.12. They are used along with $g^{\alpha\beta}$ in the symmetric key derivation steps in (21.7). This ensures forward secrecy in case the pre-shared key is leaked to an adversary at some future time. This optional forward secrecy for session resumption is a feature of TLS 1.3 that was not present in earlier versions of TLS.

Tickets. To establish a secure session using a pre-shared key, both P and Q need to remember the key and its identity. This can be difficult on a busy server that interacts with many clients. Instead, the server can offload storage of the pre-shared key to the client. To do so, the server computes an authenticated encryption c of the server's TLS state with the client, which includes all the computed pre-shared keys. This c is called a **ticket** and the encryption is done using a secret key known only to the server. Then, in the *new-session-ticket* message sent to the client, the server sets the key's identity to c. When the client later suggests to use this pre-shared key in a TLS key exchange, it sends the key's identity c to the server as part of the *offer*. The server decrypts c and obtains the pre-shared key. This way the server does not store any long-term state per client. It greatly simplifies the use of pre-shared keys in a large system.

Zero round trip data. TLS 1.3 introduced a dangerous mechanism called **0-RTT** that is designed to improve performance. 0-RTT allows the client to send encrypted application data in the very first flow of Fig. 21.12. In a Web context, 0-RTT lets the client send an HTTP request in the first flow. This can greatly improve page load time, because the server can send the response in the second flow of Fig. 21.12. Hence, 0-RTT saves a full round trip when loading a web page.

This mechanism is only allowed when the client requests a key exchange using a pre-shared key. The client appends the encrypted application data to the first flow, encrypted with a key k_{ce} that is derived from

$$psk,\ u,\ \mathcal{N}_{\text{c}},\ \mathit{offer}.$$

The server derives the same key k_{ce} from the received data and uses it to decrypt the ciphertext appended to the first flow.

The trouble with 0-RTT is that the encrypted application data is vulnerable to replay. An adversary can record the first flow from the client to the server and replay it at a later time. Unless the server takes extra steps to prevent a replay attack, the result can be very harmful to the client P. For example, suppose the first flow from P contains a query for P's bank account balance. The adversary can replay this first flow at any time and count the number of bytes in the server's response. Because the response length is correlated with the number of digits in P's account balance, the adversary can monitor P's account balance by repeatedly replaying the request from P. Even worse, if the request from P is a bill payment, the adversary can replay the request and cause the same bill to be paid multiple times.

The reason that 0-RTT data is not protected from a replay attack is that the encryption key k_{ce} cannot depend on the server nonce $\mathcal{N}_{\text{s}}$. This key must be derived before the client sees $\mathcal{N}_{\text{s}}$. All other keys in TLS depend on $\mathcal{N}_{\text{s}}$ and this prevents replay attacks on data encrypted with those keys. Data encrypted with k_{ce} is not protected this way.

With some effort, server Q can defend against replay attacks on 0-RTT data. As a first line of defense, every pre-shared key has a limited lifetime, specified in the *new-session-ticket* message when the pre-shared key is first created. The maximum lifetime is seven days. The server will reject any connection attempt using an expired pre-shared key. This limits the replay window, but does not fully prevent replays.

To prevent replays the server can store the client nonce from every 0-RTT request it receives. If the server ever sees a request with a duplicate client nonce, it can safely reject that request. Note that the client nonce only needs to be stored for a limited amount of time; it can be deleted once the corresponding pre-shared key expires. In practice, this defense is not easy to implement in a large distributed web application. The adversary can record a client request in North America and replay it in Asia. Since large systems do not typically synchronize state across geographic regions in real time, the repeated client nonce will not be detected in Asia and the replay request will be accepted.

The end result is that if a server chooses to support 0-RTT, clients can benefit from faster page load times, but they are also at risk due to replay attacks. If the benefit is not worth the risk, the server can signal to the client that it is choosing to ignore the 0-RTT data, in which case the client will retransmit the data in the third flow, after the secure session is properly established.

21.11 Password authenticated key exchange

In Section 21.6 we discussed one-sided authenticated key exchange protocols, where the server has a certificate and the client does not.

As we discussed there, a client can establish a one-sided authenticated secure channel with a server, and then identify himself to the server within the channel, using, perhaps, some simple, password-based identification protocol. This approach is widely used today. However, this approach has some serious security problems. In this section, we explore these problems in some detail, and then examine a new type of key exchange protocol, called **password based key exchange**, which mitigates these problems to some degree.

21.11.1 Phishing attacks

Suppose a client has an account with a server, and that an adversary wants to discover the client's password. Typically, the client logs into his account using a web browser, entering his user ID and password into some fields on a special "secure login page" belonging to the server. Normally, this is all done using a one-sided authenticated secure channel, as discussed above. However, in a **phishing attack**, an adversary bypasses the secure channel by simply tricking the client into entering his user ID and password on a fake login page that belongs to the adversary, rather than the secure login page.

In practice, phishing attacks are not so hard to mount. There are two phases to a phishing attack: first, to trick the client into visiting his fake login page, rather than the secure login page, and second, to make the fake login page look and feel like the secure login page, so that the client enters his user ID and password.

- One common approach used to trick a client into visiting the fake login page is to send the client an email, telling the client that there is some compelling reason that he should log in to his account at the server ("verify your account" or "confirm billing information"). The email is designed very nicely, perhaps with an official-looking logo, and for the client's "convenience," contains an embedded link that will take the client's web browser to the secure login page. However, the embedded link is really a link to the fake login page. This approach, though fairly crude, actually works with a good number of unsuspecting clients.

  Because of such attacks, careful clients know better than to follow any links in such email messages. However, there are more sophisticated strategies that trick even the most careful clients: using attacks that exploit security weaknesses in the Internet routing mechanism, it is possible for a client to directly enter one web address in the address bar of their browser, but end up at a web site controlled by the adversary.

- Once the phisher has brought the client to his fake login page, he has to make sure that his fake login page is a convincing replica of the secure login page. This is typically not too hard to do. Of course, the adversary has to design the page so that the content displayed is very similar to the content displayed on the secure login page. This is usually trivial to do. There might be other clues that indicate the client is at the wrong web page, but many clients may not notice these clues:

  - the address of the web page may not be that of the server; however many clients may not even look carefully at this address, or even know what it really means; moreover, even a more discerning client may be easily fooled if, for example, the adversary controls a domain called `somesite.com`, and then directs the client to `http://www.yourbank.com.somesite.com`, instead of `http://www.yourbank.com`;

- the web browser may not display the usual signal (e.g., a little padlock) that is used to indicate a "secure web page," but again, a casual client may not notice;
- the web browser may indeed display a "secure web page" signal, but almost no client will bother checking the details of the certificate, which in this case, may be a perfectly valid certificate that was issued by the CA to the adversary, rather than to the server; in fact, unless the client has taken a course in security or cryptography, he probably has no idea what a certificate even is.

To attempt to foil a phishing attack, instead of using a simple password identification protocol, one might use a challenge-response identification protocol, such as the following:

$$\begin{array}{ccc} P & & Q \\ & & r \overset{\mathrm{R}}{\leftarrow} \mathcal{R} \\ & \xleftarrow{\qquad\quad r \qquad\quad} & \\ v \leftarrow H(pw,r) & & \\ & \xrightarrow{\qquad\quad v \qquad\quad} & v \overset{?}{=} H(pw,r) \end{array}$$

Here, P is the client, Q is the server, pw is the password. Also, r is a random nonce, and H is a hash function (which we model as a random oracle). The server Q sends P the random nonce r, P computes v as $H(pw,r)$, and sends v to Q, and Q verifies that $v = H(pw,r)$. (Note that this protocol is a variant of the password-based challenge-response protocol discussed in Section 18.6.1, with the MAC and key derivation all rolled in to the hash function.)

If the client uses this protocol, then at least a phishing attack will not lead directly to the complete exposure of the client's password. However, there are other ways a phishing attack can be exploited.

First of all, the client identifies himself to the server, but not vice versa, so if the client is tricked into visiting the fake login page, the adversary may not get his password, but may be able to cause other trouble, since the client thinks he is securely logged in to the server, and so may be tricked into revealing other sensitive information.

Worse, the adversary could mount a man-in-the-middle attack. Again, the adversary sets up a channel with the client, via phishing, and simultaneously sets up a perfectly normal one-sided authenticated secure channel with the server. Now the adversary simply plays "man in the middle," forwarding the messages in the identification protocol from the server to client, and vice versa. Now, not only can the adversary try to obtain sensitive information from the client, as above, but since the adversary is now logged into the server under the client's user ID, he can also cause damage on the server side. For example, if the server is a bank, and the client is a customer, the adversary can transfer money from the customer's account to a bank account controlled by the adversary.

One might also consider using the following mutual challenge-response identification protocol:

$$\begin{array}{ccc}
P & & Q \\
 & & r \overset{\text{R}}{\leftarrow} \mathcal{R} \\
 & \xleftarrow{\qquad\qquad r \qquad\qquad} & \\
s \overset{\text{R}}{\leftarrow} \mathcal{R}, v \leftarrow H(pw, 0, r, s) & & \\
 & \xrightarrow{\qquad\qquad s, v \qquad\qquad} & \\
 & & v \overset{?}{=} H(pw, 0, r, s), w \leftarrow H(pw, 1, r, s) \\
 & \xleftarrow{\qquad\qquad w \qquad\qquad} & \\
w \overset{?}{=} H(pw, 1, r, s) & &
\end{array}$$

Unfortunately, this protocol is subject to the same man-in-the-middle phishing attack as above.

This type of man-in-the-middle attack can be avoided, however, if we use the *channel binding* feature that can be provided by key exchange protocols. We briefly introduced this notion in Section 21.8. The security property for channel bindings guarantees that the client and server have the same channel bindings if and only if they share a key. So to protect against a man-in-the-middle attack, we can modify the above mutual authentication protocol so that the channel binding is included in the hash. That is, the hashes are computed as $H(pw, chb, 0, r, s)$ and $H(pw, chb, 1, r, s)$, where chb is the channel binding. Now, the man-in-the-middle attacks fails, because the two participants will be computing the hashes with different channel binding inputs, so it does no good to forward a hash from one participant to the other. This even protects against phishing attacks, but also provides some security even when the server's long-term key secret used in the key exchange has been compromised (as in the PFS or HSM attack models).

Unfortunately, even with this patch, this challenge-response protocol is subject to an *offline dictionary attack* (see Section 18.3.1). Indeed, suppose that the client's password is weak, and belongs to a relatively small dictionary $\mathcal{D}$ of common passwords. Also suppose that the adversary establishes a channel with the client, via phishing. Playing the role of server, the adversary sends a nonce r to the client, who responds with $s, v := H(pw, chb, 0, r, s)$. At this point, the adversary quits the protocol. Having obtained v, the adversary now performs a brute-force search for the client's password, as follows: for each $pw' \in \mathcal{D}$, the adversary computes $H(pw', r)$, and tests if this is equal to v. If he finds such a pw', it is very likely that $pw' = pw$, and so the adversary has obtained the client's password.

By the way, reversing the roles of client and server in the above mutual identification protocol makes matters even worse: now the adversary can simply set up a normal one-sided authenticated secure channel with the server, and the first thing the server does is to send the value $H(pw, chb, 0, r, s)$ to the adversary. Now the adversary can carry out an offline dictionary attack without even having to first do any phishing.

Finally, we remark that even without phishing, the adversary can always perform an *online* dictionary attack, by simply attempting to log in to the server many times, using passwords chosen from some dictionary of common passwords. As discussed in Section 18.3.1, a server can usually take simple countermeasures that limit the number of failed login attempts, so that such online dictionary attacks are not nearly as dangerous as an offline attack.

21.11.2 PAKE: an introduction

We have discussed one-sided authenticated key exchange protocols, and how these can be combined with simple password-based identification protocols to establish a secure channel between a client and a server, where the server has a certificate, and the client has no certificate but shares a password with the server. We also discussed how this approach to establishing a secure channel is not very secure in practice: via a phishing attack, an adversary can trick a client into divulging his password to the adversary; moreover, we saw that even if a challenge-response identification protocol is used, a phisher can still obtain enough information from the client so that the adversary can still obtain the client's password using an offline dictionary attack.

These security problems are the motivation for **password authenticated key exchange (PAKE)** protocols. Here is the basic idea of a PAKE protocol. We assume that every pair of users that wish to establish a shared session key have a shared password. We make no other assumptions: there are no certificates, and there is no CA or any other type of TTP.

Ideally, passwords are strong, that is, chosen at random from a large set. In this case, the security goals for a PAKE protocol are essentially the same as for an AKE protocol; indeed, although we do not spell out a formal security model, it is essentially the same as in Section 21.9, except that now, shared, strong passwords are used for authentication purposes, in place of a TTP.

Unfortunately, in practice, a PAKE protocol may very well be used with weak passwords, and we have to relax our security expectations accordingly. Indeed, with weak passwords, any PAKE protocol is inherently vulnerable to an *online dictionary attack*: an adversary can always guess a password, and engage a user in the protocol and see if its guess is correct. Typically, the guess is incorrect if and only if either the key exchange protocol itself fails, or some higher level protocol that uses the session key fails. However, we might at least hope that this is the worst type of attack possible; in particular, we might hope that an adversary cannot mount an *offline dictionary attack*.

21.11.3 Protocol $\mathtt{PAKE}_0$

Consider the following protocol $\mathtt{PAKE}_0$, which is described in Fig. 21.13. Here, P and Q are users with a shared password pw, and H is a hash function, which we model as a random oracle. In this protocol, P and Q exchange random nonces r and s, and then compute the session key as $k = H(pw, id_P, id_Q, r, s)$. In describing this, and other, PAKE protocols, we leave out the details of how P and Q communicate their identities id_P and id_Q to one another, and how they retrieve the corresponding password.

Suppose that pw is a strong password. Then in this case, protocol $\mathtt{PAKE}_0$ is quite secure (in particular, it would satisfy the definition of security in Section 21.9, appropriately modified, modeling H as a random oracle). Notice that this protocol does not provide mutual, or even one-sided, identification: an instance of P may run the protocol, and not share a session key with anyone; however, if he shares a session key with someone, he shares it with an instance of Q.

Unfortunately, if pw is a weak password, then an eavesdropping adversary can easily carry out an *offline dictionary attack*, as follows.

Assume that pw belongs to some relatively small dictionary $\mathcal{D}$ of common passwords. Also assume that after P runs the protocol, it encrypts a publicly known plaintext m under the session key, using a symmetric cipher $\mathcal{E} = (E, D)$, and sends the resulting ciphertext out on the network.

Our adversary eavesdrops on a run of the protocol between P and Q, obtaining the values r and s. At this point, P computes the session key as $k = H(pw, id_P, id_Q, r, s)$, and sends out an

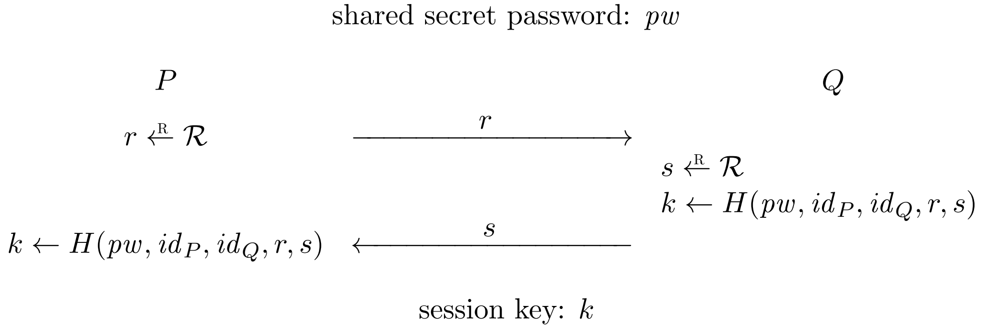

shared secret password: pw
P
Q
$r \xleftarrow{\text{R}} \mathcal{R}$
r
$s \xleftarrow{\text{R}} \mathcal{R}$
$k \leftarrow H(pw, id_P, id_Q, r, s)$
s
$k \leftarrow H(pw, id_P, id_Q, r, s)$
session key: k

Figure 21.13: Protocol $\mathtt{PAKE}_0$

encryption c of m under the key k. The adversary intercepts c, and then does the following:

```
for all $pw' \in \mathcal{D}$ do
    $k' \leftarrow H(pw', id_P, id_Q, r, s)$
    $m' \leftarrow D(k', c)$
    if $m' = m$ then
        output $pw'$ and halt
```

In all likelihood, the output pw' is equal to the password pw.

Of course, the above attack will work with many other types of partial information about the session key that may be leaked to the adversary, besides a plaintext/ciphertext pair. For example, the key may be used as a MAC key, and used to authenticate publicly known messages.

21.11.4 Protocol $\mathtt{PAKE}_1$

As we saw, if weak passwords are used, then protocol $\mathtt{PAKE}_0$ is vulnerable to an offline dictionary attack by an eavesdropping adversary. We next present a PAKE protocol that does not suffer from this vulnerability.

This protocol, which we call $\mathtt{PAKE}_1$, makes use of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$, and a hash function H, which we model as a random oracle. The protocol is described in Fig. 21.14. Here, both users compute the value $w = g^{\alpha\beta}$, and then compute the session key as $k = H(pw, id_P, id_Q, u, v, w)$.

If the password pw is strong, then this protocol is quite secure. The interesting case is what happens when the password pw is weak. First, we claim that under the CDH assumption for $\mathbb{G}$, and modeling H as a random oracle, then protocol $\mathtt{PAKE}_1$ is not vulnerable to a dictionary attack by an *eavesdropping* adversary.

We shall give an intuitive argument for this. But first, we introduce some notation, and we recall the CDH assumption. For $s, t \in \mathbb{G}$, if $s = g^{\mu}$ and $t = g^{\nu}$, then we define

$$[s, t] := g^{\mu\nu}.$$

The CDH problem is this: given random $s, t \in \mathbb{G}$, compute $[s, t]$. The CDH assumption asserts that there is no efficient algorithm that can solve the CDH problem with non-negligible probability.

shared secret password: pw

$$\begin{array}{lcl}
\qquad P & & \qquad Q \\
\alpha \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_q, u \leftarrow g^{\alpha} & \xrightarrow{\quad u \quad} & \\
 & & \beta \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_q,\ v \leftarrow g^{\beta},\ w \leftarrow u^{\beta} \\
 & & k \leftarrow H(pw, id_P, id_Q, u, v, w) \\
w \leftarrow v^{\alpha} & \xleftarrow{\quad v \quad} & \\
k \leftarrow H(pw, id_P, id_Q, u, v, w) & &
\end{array}$$

session key: k

Figure 21.14: Protocol $\texttt{PAKE}_1$

Suppose an adversary eavesdrops on a conversation between P and Q. He obtains random group elements u and v, while P and Q compute the session key as $k = H(pw, id_P, id_Q, u, v, [u, v])$. Intuitively, for a dictionary attack to succeed, the adversary will have to query the random oracle H at points of the form $(pw', id_P, id_Q, u, v, [u, v])$ for various values of pw'. Let us call such a point *relevant*. Indeed, it is only by querying the random oracle at a relevant point for some pw' that the adversary can tell if $pw' = pw$: for example, by using the value k' of the oracle at that point to decrypt a given encryption of a known plaintext under k.

We claim that under the CDH assumption, the probability that he queries the random oracle at *any* relevant point is negligible. Indeed, if an adversary can make a relevant query with non-negligible probability, then we could use this adversary to solve the CDH problem with non-negligible probability, as follows. Given a challenge instance (s, t) of the CDH problem, set $u := s$ and $v := t$, and give u and v to our eavesdropping adversary. Now, the adversary will make a number of random oracle queries. As usual, we process random oracle queries using a lookup table, and collect a list of all queries of the form $(pw', id_P, id_Q, u, v, w')$, where pw' is an arbitrary password, and w' is an arbitrary group element. Finally, we select one of the queries in this list at random, and output the corresponding w'. If our selected query is relevant, then w' is a solution to the CDH problem. Note that because recognizing solutions to the CDH problem is in general hard (this is the DDH assumption), we cannot easily recognize relevant queries, and so we are forced to employ this guessing strategy; nevertheless, if the adversary has a non-negligible chance of making a relevant query, we have non-negligible (though smaller) chance of solving the CDH problem.

Thus, we have shown that protocol $\texttt{PAKE}_1$ provides security against an offline dictionary attack by an *eavesdropping* adversary. However, as we now illustrate, protocol $\texttt{PAKE}_1$ does not provide security against a dictionary attack by a *active* adversary, that is, an adversary that participates directly in the protocol.

Assume that pw belongs to some relatively small dictionary $\mathcal{D}$ of common passwords. Also assume that after P runs the protocol, it encrypts a publicly known plaintext m under the session key, using a symmetric cipher $\mathcal{E} = (E, D)$, and sends the resulting ciphertext out on the network.

Our adversary works as follows. First, he plays the role of Q in $\texttt{PAKE}_1$. The honest user P sends u to the adversary, who simply follows the protocol, computing

$$\beta \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_q, \quad v \leftarrow g^{\beta}, \quad w \leftarrow u^{\beta},$$

public system parameters: $a, b \in \mathbb{G}$
shared secret password: pw

$$
\begin{array}{lcl}
\qquad\qquad P & & \qquad\qquad Q \\
\alpha \xleftarrow{\text{R}} \mathbb{Z}_q,\, u \leftarrow g^{\alpha} a^{pw} & \xrightarrow{\quad u \quad} & \\
 & & \beta \xleftarrow{\text{R}} \mathbb{Z}_q,\ v \leftarrow g^{\beta} b^{pw} \\
 & & w \leftarrow (u/a^{pw})^{\beta} \\
 & & k \leftarrow H(pw, id_P, id_Q, u, v, w) \\
w \leftarrow (v/b^{pw})^{\alpha} & \xleftarrow{\quad v \quad} & \\
k \leftarrow H(pw, id_P, id_Q, u, v, w) & & \\
 & \text{session key: } k &
\end{array}
$$

Figure 21.15: Protocol $\texttt{PAKE}_2$

and sending v to P. At this point, P computes the session key as $k = H(pw, id_P, id_Q, u, v, w)$, and sends out an encryption c of m under the key k. The adversary intercepts c, and then does the following:

```
for all $pw' \in \mathcal{D}$ do
    $k' \leftarrow H(pw', id_P, id_Q, u, v, w)$
    $m' \leftarrow D(k', c)$
    if $m' = m$ then
        output $pw'$ and halt
```

In all likelihood, the output pw' is equal to the password pw.

21.11.5 Protocol $\texttt{PAKE}_2$

As we saw, if weak passwords are used, then while protocol $\texttt{PAKE}_1$ provides security against an offline dictionary attack by an *eavesdropping* adversary, it is vulnerable to an offline dictionary attack by an *active* adversary.

We now present a protocol, $\texttt{PAKE}_2$, which *does* provide security against an offline dictionary attack, by both passive and active adversaries. Like $\texttt{PAKE}_1$, protocol $\texttt{PAKE}_2$ makes use of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in G$, and a hash function H, which we model as a random oracle. The protocol has additional system parameters a and b, which are randomly chosen elements of $\mathbb{G}$. Furthermore, passwords are viewed as elements of $\mathbb{Z}_q$. This is not a limitation since, if needed, passwords can be hashed into $\mathbb{Z}_q$ using an appropriate hash function.

Protocol $\texttt{PAKE}_2$ is described in Fig. 21.15. Just as in protocol $\texttt{PAKE}_1$, both users compute the value $w = g^{\alpha\beta}$, and then compute the session key as $k = H(pw, id_P, id_Q, u, v, w)$. The only difference is that now P "blinds" the value g^{α} by multiplying it by a^{pw}, and Q blinds the value g^{β} by multiplying it by b^{pw}.

We now give an informal argument that protocol $\texttt{PAKE}_2$ provides security against dictionary attacks by either an eavesdropping or active adversary, under the CDH assumption, and modeling H as a random oracle.

First, consider an adversary that eavesdrops on a run of the protocol between honest user P and honest user Q. He obtains a conversation (u, v). The session key computed by P and Q is

$$k = H(pw, id_P, id_Q, u, v, [u/a^{pw}, v/b^{pw}]). \tag{21.8}$$

Intuitively, the adversary's goal is to query the random oracle at as many *relevant* points as possible, where here, a relevant point is one of the form

$$(pw', id_P, id_Q, u, v, [u/a^{pw'}, v/b^{pw'}]), \tag{21.9}$$

where $pw' \in \mathbb{Z}_q$. The following lemma shows that under the CDH assumption, he is unable to make even a single relevant query:

Lemma 21.7. *Under the CDH assumption, the following problem is hard: given random $a, b, u, v \in \mathbb{G}$, compute $\gamma \in \mathbb{Z}_q$ and $w \in \mathbb{G}$ such that $w = [u/a^\gamma, v/b^\gamma]$.*

Proof. We first make some simple observations about the “Diffie-Hellman operator” $[\cdot,\cdot]$. Namely, for all $x, y, z \in \mathbb{G}$ and all $\mu, \nu \in \mathbb{Z}_q$, we have

$$[x, y] = [y, x], \quad [xy, z] = [x, z][y, z], \quad \text{and} \quad [x^\mu, y^\nu] = [x, y]^{\mu\nu}.$$

Also, note that $[x, g^\mu] = x^\mu$, so given any two group elements x and y, if we know the discrete logarithm of either one, we can efficiently compute $[x, y]$.

Now suppose we have an adversary that can efficiently solve the problem in the statement of the lemma with non-negligible probability. We show how to use this adversary to solve the CDH problem with non-negligible probability. Given a challenge instance (s, t) for the CDH problem, we compute

$$\mu \xleftarrow{\text{R}} \mathbb{Z}_q, \quad a \leftarrow g^\mu, \quad \nu \xleftarrow{\text{R}} \mathbb{Z}_q, \quad b \leftarrow g^\nu,$$

and then we give the adversary

$$a, \quad b, \quad u := s, \quad v := t.$$

Suppose now that the adversary computes for us $\gamma \in \mathbb{Z}_q$ and $w \in \mathbb{G}$ such that $w = [u/a^\gamma, v/b^\gamma]$. Then we have

$$w = [u, v][u, b]^{-\gamma}[a, v]^{-\gamma}[a, b]^{\gamma^2}. \tag{21.10}$$

Since we know the discrete logarithms of a and b, each of the quantities

$$w, \quad [u, b], \quad [a, v], \quad [a, b], \quad \gamma$$

appearing in (21.10) are either known or easily computed from known values, and so we can easily solve (21.10) for $[u, v]$, which is the same as $[s, t]$. □

Next, consider an active adversary that engages in the protocol with an honest user. We consider the case where the adversary plays the role of Q, and the honest user is P — the argument in the other case is similar.

Now, in the adversary's attack, he obtains the first message u from P, which is just a random group element. Next, the adversary computes a group element v in some way, and sends this to P — the adversary may compute v in any way he likes, possibly in some devious way that depends on u. As usual, P now computes the session key as in (21.8), and the adversary's goal is to evaluate the random oracle H at as many relevant points, as in (21.9), as possible. Of course, an adversary that simply follows the protocol using some guess pw' for password can always make one relevant query. What we want to show is that it is infeasible to make more than one relevant query. This is implied by the following lemma:

Lemma 21.8. *Under the CDH assumption, the following problem is hard: given random $a, b, u \in \mathbb{G}$, compute $\gamma_1, \gamma_2 \in \mathbb{Z}_q$ and $v, w_1, w_2 \in \mathbb{G}$ such that $\gamma_1 \neq \gamma_2$ and $w_i = [u/a^{\gamma_i}, v/b^{\gamma_i}]$ for $i = 1, 2$.*

Proof. Suppose that we are given an instance (s, t) of the CDH problem. Then we compute

$$\mu \xleftarrow{\text{R}} \mathbb{Z}_q, \quad a \leftarrow g^{\mu},$$

and give the adversary

$$a, \quad b := s, \quad u := t.$$

The adversary computes for us γ_1, γ_2 and w_1, w_2 such that $\gamma_1 \neq \gamma_2$, and

$$w_i = [u/a^{\gamma_i}, v/b^{\gamma_i}] = [u, v][u, b]^{-\gamma_i}[a, v]^{-\gamma_i}[a, b]^{\gamma_i^2} \quad (i = 1, 2).$$

Then we have

$$w_2/w_1 = [u, b]^{\gamma_1 - \gamma_2}[a, v]^{\gamma_1 - \gamma_2}[a, b]^{\gamma_2^2 - \gamma_1^2}. \tag{21.11}$$

Since we know the discrete logarithm of a, each of the quantities

$$w_1, \quad w_2, \quad [a, v], \quad [a, b], \quad \gamma_1, \quad \gamma_2$$

appearing in (21.11) is either known or easily computed from known values; moreover, since $\gamma_1 - \gamma_2 \neq 0$, we can efficiently solve (21.11) for $[u, b]$, which is the same as $[s, t]$. □

21.11.6 Protocol $\texttt{PAKE}_2^+$

Often, users play very distinct roles. One user may be a **client**, which obtains the password by keyboard entry, while the other is a **server**, which is a machine that keeps a *password file*, containing information for each client who is authorized to access the server. A type of attack that we would like to provide some defense against is a *server compromise*, in which an adversary obtains the server's password file. Given the password file, the adversary can certainly impersonate the server; however, we would like to make it as hard as possible for the adversary to impersonate a client, and gain unauthorized access to the server.

Given the password file, an adversary can always mount an offline dictionary attack to discover a given client's password: the adversary can just run both the client and server side of the protocol, using a guess for the password on the client's side, and using the data stored in the password file on the server's side, and see if the protocol completes successfully. Ideally, this would be all the adversary could do. Recall that such an offline dictionary attack can made more difficult for the adversary by using a slow hash function (Section 18.4.3).

Consider protocol $\texttt{PAKE}_2$, which as we argued, provides security against offline dictionary attacks by both eavesdropping and active adversaries. The roles of the two users in that protocol are quite symmetric, but for concreteness, let us say that P is the client, and Q is the server. In the most obvious implementation, Q would explicitly store the password pw in the password file. Clearly, this implementation is undesirable, as an adversary that compromises the server immediately obtains the password.

We now present protocol $\texttt{PAKE}_2^+$, which has the property that if the server is compromised, the best an adversary can do to impersonate a client is an offline dictionary attack. Like $\texttt{PAKE}_2$, protocol $\texttt{PAKE}_2^+$ makes use of a cyclic group $\mathbb{G}$ of prime order q generated by $g \in G$, group elements $a, b \in \mathbb{G}$, and a hash function H, which we model as a random oracle. In addition, the protocol

public system parameters: $a, b \in \mathbb{G}$
password: pw, $\quad (\pi_0, \pi_1) := H'(pw, id_P, id_Q)$

P | Q

secret: π_0, π_1 | secret: $\pi_0, c := g^{\pi_1}$

$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, u \leftarrow g^{\alpha} a^{\pi_0}$ $\quad \xrightarrow{\quad u \quad}$

$\beta \xleftarrow{\text{R}} \mathbb{Z}_q,\ v \leftarrow g^{\beta} b^{\pi_0}$
$w \leftarrow (u/a^{\pi_0})^{\beta},\ d \leftarrow c^{\beta}$
$k \leftarrow H(\pi_0, u, v, w, d)$

$w \leftarrow (v/b^{\pi_0})^{\alpha},\ d \leftarrow (v/b^{\pi_0})^{\pi_1}$ $\quad \xleftarrow{\quad v \quad}$
$k \leftarrow H(\pi_0, u, v, w, d)$

session key: k

Figure 21.16: Protocol $\texttt{PAKE}_2^+$

employs another hash function H', which has range $\mathbb{Z}_q \times \mathbb{Z}_q$, and which we also model as a random oracle.

Let pw be the password shared between client P and server Q, which is an arbitrary bit string. The protocol is described in Fig. 21.16. Here, the client stores (π_0, π_1), while the server stores (π_0, c), where $c := g^{\pi_1}$, where

$$(\pi_0, \pi_1) := H'(pw, id_P, id_Q) \in \mathbb{Z}_q \times \mathbb{Z}_q.$$

Of course, the client can derive (π_0, π_1) from pw. Both users compute the values $w = g^{\alpha\beta}$ and $d = g^{\pi_1 \beta}$, and then compute the session key as $k = H(\pi_0, u, v, w, d)$.

It is not hard to argue that protocol $\texttt{PAKE}_2^+$ offers the same level of security as protocol $\texttt{PAKE}_2$ under normal conditions, when the server is not compromised. However, consider what happens if the server Q is compromised in protocol $\texttt{PAKE}_2^+$, and the adversary obtains π_0 and c. At this point, the adversary could attempt an offline dictionary attack, as follows: evaluate H' at points (pw', id_P, id_Q) for various passwords pw', trying to find pw' such that $H'(pw', id_P, id_Q) = (\pi_0, \cdot)$. If this succeeds, then with high probability, $pw' = pw$, and the adversary can easily impersonate the client.

The key property we want to prove is the following: if the above dictionary attack fails, then under the CDH assumption, the adversary cannot impersonate the client.

To prove this property, first suppose that an adversary compromises the server, then attempts a dictionary attack, and finally, attempts to log in to the server. Compromising the server means that the adversary obtains π_0 and $c = g^{\pi_1}$. Now suppose the dictionary attack fails, which means that the adversary has not evaluated H' at the point (pw, id_P, id_Q). The value π_1 is completely random, and the adversary has no other information about π_1, other than the fact that $c = g^{\pi_1}$. When he attempts to log in, he sends the server Q some group element u, and the server responds with $v := g^{\beta} b^{\pi_0}$ for random $\beta \in \mathbb{Z}_q$. Now, the adversary knows π_0, and therefore can compute the value $e := g^{\beta}$. However, to successfully impersonate the client, he must evaluate the random oracle H at the point $(\pi_0, u, v, [u/a^{\pi_0}, e], [c, e])$, which means he has to compute $[c, e]$. But since c and e

are random group elements from the adversary's point of view, computing $[c, e]$ is tantamount to solving the CDH problem.

The complication we have not addressed in this argument is that the adversary may also interact with the client P at some point, giving an arbitrary value v to P, who raises v/b^{π_0} to the power π_1, and derives a session key from this value. Because of this, P acts to a certain degree as a DDH oracle, essentially giving the adversary an oracle for recognizing DH-triples of the form $(g^{\pi_1}, \cdot, \cdot)$. The issues are much the same as in the proof of Theorem 12.4. At first glance, it might appear that we need to make use of the interactive CDH assumption (see Definition 12.4) to prove security; however, a closer examination shows that this is not the case. This is because in deriving the session key, P also passes the value $w := (v/b^{\pi_0})^{\alpha}$ to the function H, and so P acts as an oracle for recognizing 2DH-tuples (Exercise 12.32) of the form $(g^{\pi_1}, g^{\alpha}, \cdot, \cdot, \cdot)$, where α is generated at random by P. Using the trapdoor test in Exercise 12.32, we can prove security under the CDH assumption.

21.11.7 Explicit key confirmation

As it is now, if an adversary runs protocol $\mathtt{PAKE}_2$ or $\mathtt{PAKE}_2^+$ with an honest user, using a guess at the password that turns out to be wrong, then the adversary will have the wrong session key, but the honest user will have no immediate indication that something went wrong. While higher level protocols will most likely fail, possibly raising some suspicions, from a system design point of view, it is perhaps better if the key exchange protocol itself raises the alarm. This is easy to do using a simple form of what is called **explicit key confirmation**. Instead of just deriving a session key k from the hash, both users P and Q can derive keys k_0 and k_1, and then:

- P sends k_0 to Q,
- Q sends k_1 to P,
- P checks that the value $\tilde{k}_1$ it receives is equal to its computed value k_1,
- Q checks that the value $\tilde{k}_0$ it receives is equal to its computed value k_0.

If an online dictionary attack is underway, the protocol will be immediately alerted to this, and can take defensive measures (see Section 18.3.1). Thus, in using PAKE protocols such as $\mathtt{PAKE}_2$ or $\mathtt{PAKE}_2^+$, it is highly recommended to augment them with an explicit key confirmation step.

21.11.8 Phishing again

PAKE protocols provide some protection against phishing attacks (see Section 21.11.1). However, a phishing adversary can still attempt to bypass the PAKE protocol entirely. For example, an adversary may lure a client to a fake login page on his web browser, and the client enters his password into the web browser in such a way that the password gets transmitted directly to the adversary, rather than being processed by a PAKE protocol. This problem can be defended against by an appropriate user interface design, so that the web browser presents an easy-to-identify and hard-to-fake "safe area," into which passwords should be entered, to be processed using a PAKE protocol. Admittedly, designing such a user interface is not easy.

PAKE protocols can be combined to useful effect with a one-sided AKE protocol (see Section 21.6), despite the susceptibility of such protocols to phishing attacks.

First, consider the problem of how a client establishes a shared password with the server in the first place. Using a secure channel set up using a one-sided AKE protocol might be the only reasonable way to do this, short of using a more expensive or less convenient type of secure channel.

Second, once the client and server have a shared password, it cannot hurt to run a PAKE protocol through a secure channel set up using a one-sided AKE protocol. This way, an adversary's ability to attack the PAKE protocol will be limited by his ability to mount a successful phishing attack.

21.11.9 Case study: PAKE used in the WiFi WPA3 protocol

To be written.

21.12 Key exchange using an online trusted third party

So far in this chapter we looked at key exchange protocols where the trusted third party (TTP) is active only during the registration phase. If the the TTP is allowed to play an active role during both registration and key exchange then it is possible to construct secure key exchange protocols using only symmetric ciphers. There is no need for public-key cryptography. A TTP that plays an active role in a key exchange protocol is sometimes called a **key distribution center**, or **KDC**, for short.

While a key exchange protocol with an online TTP is very efficient for clients, it can generate a heavy load on the TTP. This is especially true in settings where a large number of clients engage in many key exchanges every second. The online TTP must play an active role in all these exchanges, making it expensive to build a scalabe and reliable system. Moreover, a compromise of the TTP compromises all key exchanges, past and future. These scalability and security concerns make these protocols difficult to use for the global Internet. However, for corporate networks, a trusted online TTP may be acceptable and quite practical.

21.12.1 A key exchange protocol with an online TTP

We now present a secure key exchange protocol with an online TTP called `OnlineTTP`. The protocol uses a CPA-secure symmetric cipher and a secure MAC. Let $\mathcal{K}_{\text{e}}$ be the cipher key space and $\mathcal{K}_{\text{m}}$ be the MAC key space. The **registration protocol** with the TTP is straightforward, but requires a private channel to the TTP. User P with identity id_P who wishes to register with the TTP does the following:

1. Send the message id_P to the TTP to indicate that the user is requesting to be registered.

2. The TTP generates a random long term secret key

$$k_P \xleftarrow{\text{R}} (k_{\text{enc},P},\ k_{\text{mac},P}) \in \mathcal{K}_{\text{e}} \times \mathcal{K}_{\text{m}}$$

   and stores the pair $(id_P,\ k_P)$ in a private table. The TTP sends the message k_P to P.

User P now has a long term key k_P shared with the TTP. Clearly the adversary should not be allowed to eavesdrop or modify these messages. We note that the TTP need not store all user keys k_P. Instead, the TTP need only store one PRF master secret key k_{TTP} and re-generate k_P as needed by evaluating $k_P \leftarrow F(k_{\text{TTP}}, id_P)$. If needed, one can prepend a random client nonce to the

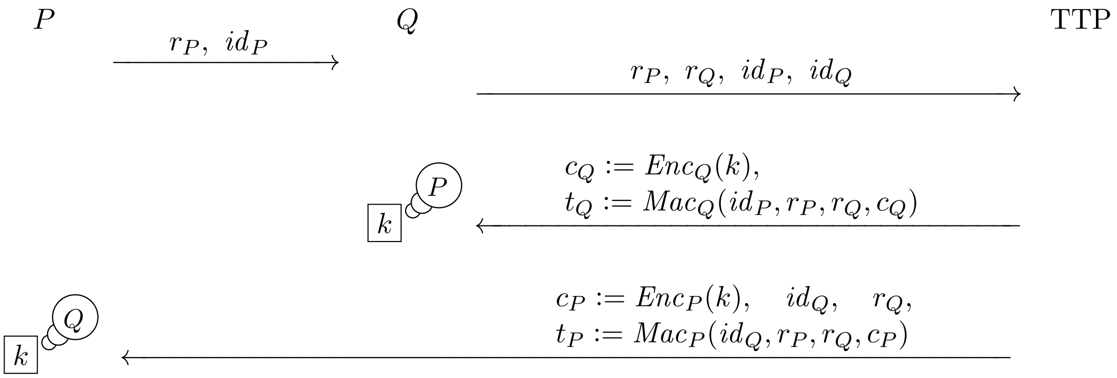

P
Q
TTP
$r_P,\ id_P$
$r_P,\ r_Q,\ id_P,\ id_Q$
$c_Q := Enc_Q(k),$
$t_Q := Mac_Q(id_P, r_P, r_Q, c_Q)$
P
k
$c_P := Enc_P(k), \quad id_Q, \quad r_Q,$
$t_P := Mac_P(id_Q, r_P, r_Q, c_P)$
Q
k

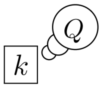
Figure 21.17: Key exchange with an online TTP

id_P input to the PRF so that the user can refresh their shared key with the TTP at any time by generating a new client nonce.

Key exchange. Next, we describe the key exchange protocol between users P and Q. We use the following notation:

- id_P and id_Q denote the identities of the users P and Q;
- For a user X (either P or Q) we let $Enc_X(m)$ denote the encryption of message m using the secret key $k_{\mathrm{enc},X}$ shared between X and the TTP;
- For a user X we let $Mac_X(m)$ denote a MAC of the message m using the secret key $k_{\mathrm{mac},X}$ shared between X and the TTP.
- $\mathcal{K}$ denotes the set of session keys;
- $\mathcal{R}$ denotes a large set, which will be used to generate random nonces.

As before, we use the notation



to indicate that user P terminated the protocol successfully, P obtained a session key k, and P thinks he is talking to Q.

The key exchange protocol `OnlineTTP` between parties P and Q is illustrated in Fig. 21.17 and runs as follows:

1. P computes $r_P \xleftarrow{\text{R}} \mathcal{R}$, and sends $(r_P,\ id_P)$ to Q;
2. Q computes $r_Q \xleftarrow{\text{R}} \mathcal{R}$, and sends $(r_P,\ r_Q,\ id_P,\ id_Q)$ to the TTP;
3. TTP checks that it has a secret key k_P for P and k_Q for Q and aborts if not; otherwise TTP computes
$$\begin{array}{ll} k \xleftarrow{\text{R}} \mathcal{K}, & \\ c_Q \xleftarrow{\text{R}} Enc_Q(k), & t_Q \xleftarrow{\text{R}} Mac_Q\big(id_P, r_P, r_Q, c_Q\big) \\ c_P \xleftarrow{\text{R}} Enc_P(k), & t_P \xleftarrow{\text{R}} Mac_P\big(id_Q, r_P, r_Q, c_P\big) \end{array} \quad .$$

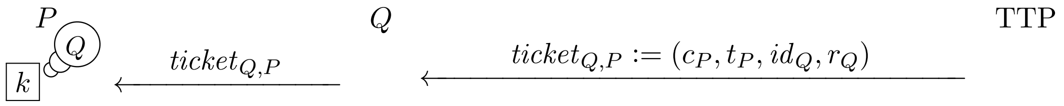
TTP sends $(c_Q,\ t_Q)$ to Q and sends $(c_P,\ t_P,\ id_Q,\ r_Q)$ to P.

4. Q verifies that t_Q is a valid tag on the message (id_P, r_P, r_Q, c_Q) and aborts if not; otherwise, Q decrypts the ciphertext c_Q and verifies that c_Q decrypts to a message $k \in \mathcal{K}$; if not, Q aborts; otherwise, Q terminates successfully, and outputs the session key k, and partner identity id_P.

5. P verifies that t_P is a valid tag on the message (id_Q, r_P, r_Q, c_P) and aborts if not; otherwise, P decrypts the ciphertext c_P and verifies that c_P decrypts to a message $k \in \mathcal{K}$; if not, P aborts; otherwise, P terminates successfully, and outputs the session key k, and partner identity id_Q.

The term id_Q is sent in the last flow from the TTP to P so that P knows the peer's identity. The term r_Q is sent in this last flow to enable P to verify the MAC t_P.

Party P in this protocol sends a message to Q and, in response, receives a message from the TTP. If needed, protocol `OnlineTTP` can be easily modified so that P only communicates with Q by simply routing the final message to P through Q. The information from the TTP intended for P but routed through Q, is called a **ticket**. In more detail, the last flow in `OnlineTTP` can be changed to



Now P only communicates with Q, as required.

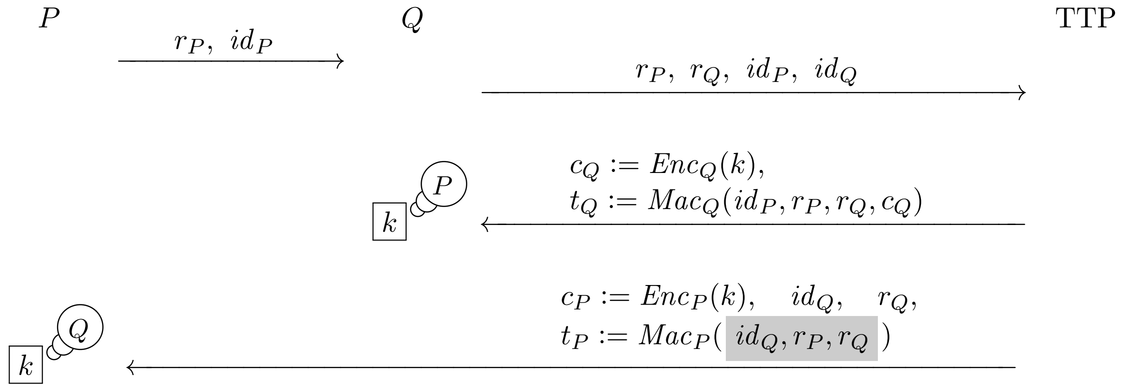
Channel bindings. Recall that in Section 21.8 we discussed the notion of a channel binding, which is a globally unique name by which a higher-level application can refer to a session. For this protocol, we can optionally add such a channel binding. An appropriate channel binding is (id_P, id_Q, r_P, r_Q).

21.12.2 Insecure variations of protocol `OnlineTTP`

To get a feel for why protocol `OnlineTTP` is designed the way it is, we examine several variants of the protocol and show that they are vulnerable to attack. In particular we show that removing elements from the MACs computed by the TTP leads to insecure protocols. These attacks demonstrate why each and every piece of protocol `OnlineTTP` is essential to achieve security.

Insecure variation 1: TTP does not MAC the ciphertext c_P — key exposure attack

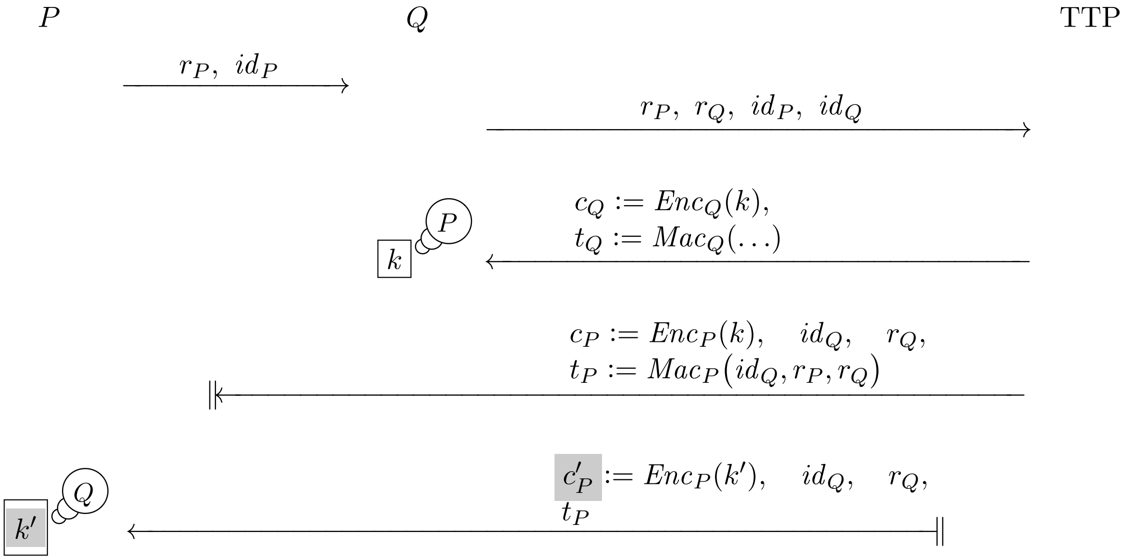
Suppose we modify protocol `OnlineTTP` so that the MAC in the message from the TTP to P does not include the ciphertext c_P. The resulting protocol runs as follows:


P
Q
TTP
$r_P,\ id_P$
$r_P,\ r_Q,\ id_P,\ id_Q$
$c_Q := Enc_Q(k)$,
$t_Q := Mac_Q(id_P, r_P, r_Q, c_Q)$
k
P
$c_P := Enc_P(k),\quad id_Q,\quad r_Q$,
$t_P := Mac_P(\ id_Q, r_P, r_Q\ )$
k
Q

This modified protocol is vulnerable to a severe key exposure attack as follows:

- First, the adversary registers a new user R with the TTP and initiates a conversation with P. The adversary obtains $c_R := Enc_R(k')$ from the TTP and decrypts it to obtain k'. In addition, the adversary eavesdrops on the message from the TTP to P and obtains $c'_P := Enc_P(k')$.
- At some time later, P and Q engage in a key exchange protocol. When the TTP sends the message $(c_P,\ t_P,\ id_Q,\ r_Q)$ to P, the adversary intercepts it and instead delivers the message $(c'_P,\ t_P,\ id_Q,\ r_Q)$ to P.

The following diagram illustrates step 2 of the attack, after the adversary obtained k' and c'_P.

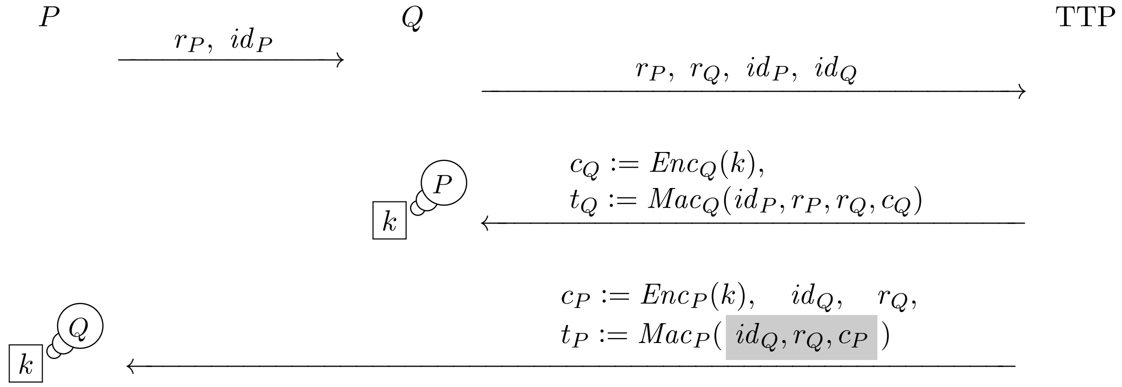

P
Q
TTP
$r_P,\ id_P$
$r_P,\ r_Q,\ id_P,\ id_Q$
$c_Q := Enc_Q(k)$,
$t_Q := Mac_Q(\ldots)$
k
P
$c_P := Enc_P(k),\quad id_Q,\quad r_Q$,
$t_P := Mac_P(id_Q, r_P, r_Q)$
$c'_P := Enc_P(k'),\quad id_Q,\quad r_Q$,
t_P
k'
Q

P will not detect the adversary's interference since the TTP does not MAC the ciphertext c_P in its message to P. At the end of the attack, Q is holding the session key k, which is unknown to the adversary; however, P is holding the session key k', which is known to the adversary. Once the protocol completes, P will likely begin using k' to encrypt private information. The adversary can read everything P sends. We previously referred to this as a key exposure attack.

Key exposure attack on Q. A similar key exposure attack is possible if the MAC in the message from the TTP to Q does not include the ciphertext c_Q. To attack Q the adversary replaces c_Q by c'_Q where $c'_Q := Enc_Q(k')$.

Insecure variation 2: TTP does not MAC the nonce r_P — replay attack

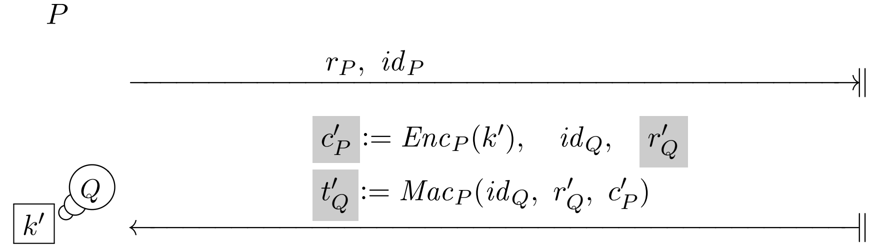
Suppose we modify protocol `OnlineTTP` so that the MAC in the message from the TTP to P does not include the nonce r_P. The resulting protocol runs as follows:


P
Q
TTP
$r_P,\ id_P$
$r_P,\ r_Q,\ id_P,\ id_Q$
$c_Q := Enc_Q(k),$
$t_Q := Mac_Q(id_P, r_P, r_Q, c_Q)$
P
k
$c_P := Enc_P(k),\quad id_Q,\quad r_Q,$
$t_P := Mac_P(\ id_Q, r_Q, c_P\ )$
Q
k

This new protocol is susceptible to the following replay attack, which is similar to the attack discussed in Variation 2 in Section 21.2:

- first, the adversary eavesdrops on a conversation between P and Q; suppose P sent the message (r'_P, id_P) to Q, who then sends the appropriate message to the TTP. The TTP responds with a message to Q and a message (c'_P, t'_P, id_Q, r'_Q) to P where $c'_P := Enc_P(k')$; these messages are recorded by the adversary;
- at some later time, P initiates a new run of the protocol with Q; P sends out the message (r, id_P); the adversary intercepts this message, throws it away, and sends the message (c'_P, t'_P, id_Q, r'_Q) to P recorded from the previous run of the protocol.

The following diagram illustrates the second step in the attack. The shaded elements are replayed by the adversary from the old conversation between P and Q:

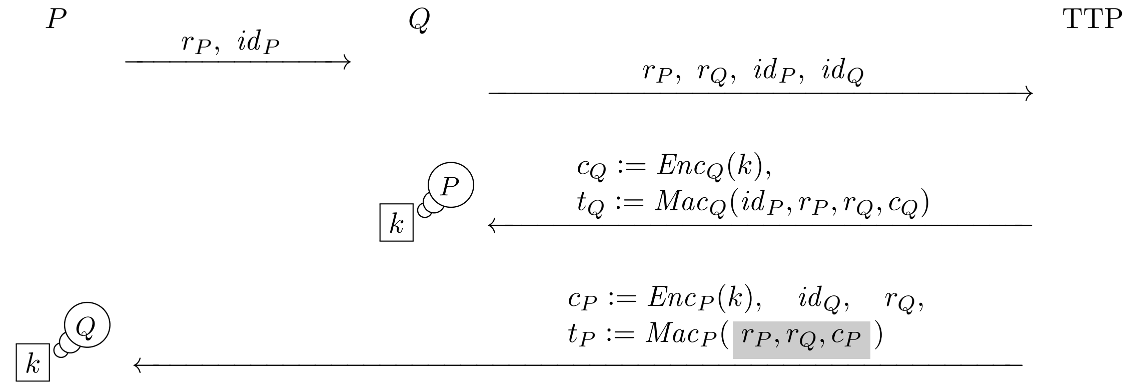

P
$r_P,\ id_P$
$c'_P := Enc_P(k'),\quad id_Q,\quad r'_Q$
$t'_Q := Mac_P(id_Q,\ r'_Q,\ c'_P)$
Q
k'

At the end of the attack, user P thinks he is talking to Q, but the session key k' is the same key used in a previous conversion with Q.

Replay attack on Q. A similar replay attack on Q is possible if the MAC on the message from the TTP to Q does not include the nonce r_Q. In the second step in the attack the adversary initiates a key exchange with Q and sends the message (r'_P, id_P) to Q. It then intercepts the message from the TTP to Q and modifies it to contain (c'_Q, t'_Q). User Q successfully completes the protocol, but is holding a session key k' from a previous conversation.

Insecure variation 3: TTP does not MAC the identity id_P — identity misbinding attack

Suppose we modify protocol `OnlineTTP` so that the MAC in the message from the TTP to P does not include id_Q. The resulting protocol runs as follows:

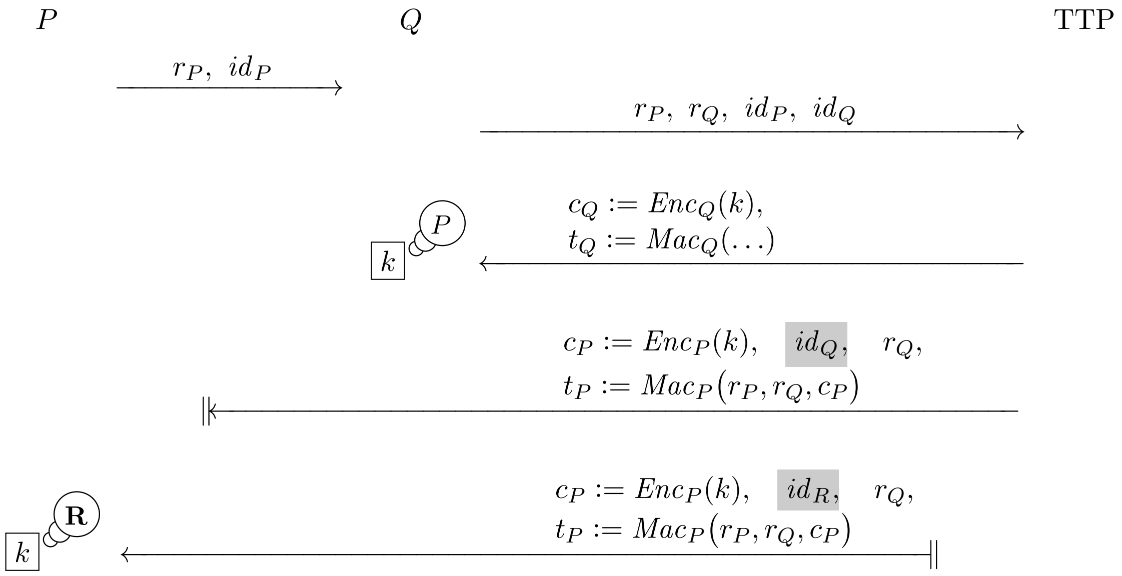


P
Q
TTP
$r_P,\ id_P$
$r_P,\ r_Q,\ id_P,\ id_Q$
$c_Q := Enc_Q(k),$
$t_Q := Mac_Q(id_P, r_P, r_Q, c_Q)$
P
k
$c_P := Enc_P(k),\quad id_Q,\quad r_Q,$
$t_P := Mac_P(\ r_P, r_Q, c_P\ )$
Q
k

The attack on this protocol is an identity misbinding attack, similar to the attack discussed in Variation 3 in Section 21.2. The attack causes Q to think he is speaking with P while P thinks he speaking with R. Here is how the attack works:

- first, the adversary registers a new user R;
- at some time later, P and Q engage in a key exchange protocol. When the TTP sends the final message $(c_P,\ t_P,\ id_Q,\ r_Q)$ to P, the adversary intercepts this message, and instead delivers the message $(c_P,\ t_P,\ id_R,\ r_Q)$ to P.

The following diagram illustrates the attack:



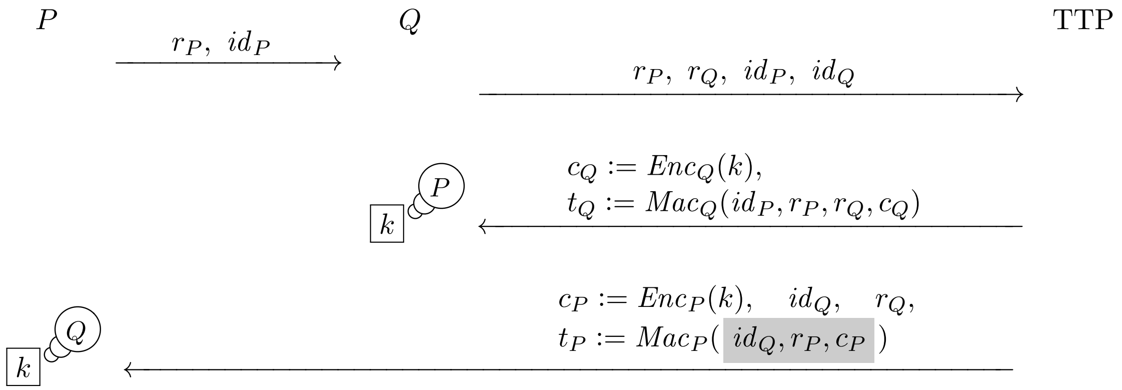
P
Q
TTP
$r_P,\ id_P$
$r_P,\ r_Q,\ id_P,\ id_Q$
$c_Q := Enc_Q(k),$
$t_Q := Mac_Q(\ldots)$
P
k
$c_P := Enc_P(k),\quad id_Q,\quad r_Q,$
$t_P := Mac_P\big(r_P, r_Q, c_P\big)$
$c_P := Enc_P(k),\quad id_R,\quad r_Q,$
$t_P := Mac_P\big(r_P, r_Q, c_P\big)$
R
k

Both sides complete the protocol successfully and share the same key k. The adversary has no direct information on k. Nevertheless, the parties are misbound.

Identity misbinding attack on Q. A similar misbinding attack on Q is possible if the MAC on the message from the TTP to Q does not include id_P. The adversary intercepts the initial message (r_P, id_P) from P to Q and delivers instead the message (r_P, id_R) to Q. Next, the adversary intercepts the message (r_P, r_Q, id_R, id_Q) from Q to the TTP and delivers instead the message (r_P, r_Q, id_P, id_Q) to the TTP. Now both parties complete the protocol successfully, but Q thinks he is talking to R and P thinks he is talking to Q.

Insecure variation 4: TTP does not MAC the nonce r_Q

Suppose we modify protocol `OnlineTTP` so that the MAC in the message from the TTP to P does not include r_Q. The resulting protocol runs as follows:



P
Q
TTP
$r_P,\ id_P$
$r_P,\ r_Q,\ id_P,\ id_Q$
$c_Q := Enc_Q(k),$
$t_Q := Mac_Q(id_P, r_P, r_Q, c_Q)$
P
k
$c_P := Enc_P(k),\quad id_Q,\quad r_Q,$
$t_P := Mac_P(\ id_Q, r_P, c_P\ )$
Q
k

Note that r_Q sent to P in the last flow appears to be redundant and can be removed. However, this is not the case, if the protocol is to provide secure channel bindings.

The attack works as follows: The adversary waits for P and Q to engage in a key exchange protocol. When the TTP sends the final message $(c_P,\ t_P,\ id_Q,\ r_Q)$ to P, the adversary intercepts this message, and instead delivers the message $(c_P,\ t_P,\ id_Q,\ 0)$ to P. In other words, the adversary changes r_Q to 0.

At the end of the attack, P thinks he is talking to Q and Q thinks he is talking to P, as required. Similarly both sides agree on the same session key k and k is unknown to the adversary, as required. However, they would disagree on their channel bindings. P's channel binding would be $(id_P, id_Q, r_P, 0)$ and Q's channel binding would be (id_P, id_Q, r_P, r_Q).

21.12.3 Security for protocol `OnlineTTP`

Protocol `OnlineTTP` clearly satisfies the correctness requirement in Section 21.9. It also can be proved to satisy the definition of *static security* in Section 21.9 (and to provide secure channel bindings, if these are implemented). We state the following theorem without proof.

Theorem 21.9. *Protocol* `OnlineTTP` *is a statically secure key exchange protocol (optionally providing secure channel bindings), assuming the nonce space $\mathcal{R}$ is large, the underlying cipher is CPA secure, and the underlying MAC system is secure.*

Note that protocol `OnlineTTP` is trivially not PFS secure. Once the adversary learns either P's key, Q's key, or the TTP's key, all past sessions between P and Q are exposed.

21.13 A fun application: establishing Tor channels

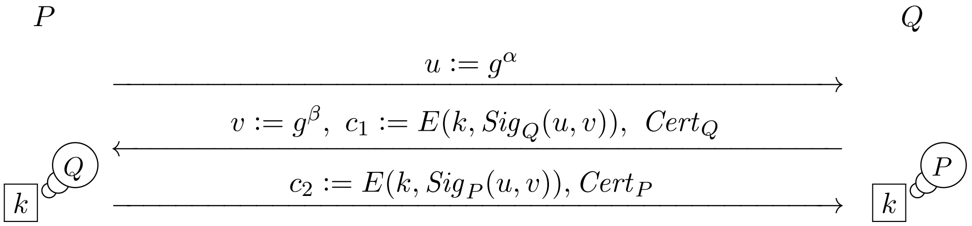
To be written.

21.14 Notes

Citations to the literature to be added.

21.15 Exercises

21.1 (Station to station). The station to station (STS) protocol runs as follows:


P
Q
$u := g^{\alpha}$
$v := g^{\beta},\ c_1 := E(k, Sig_Q(u, v)),\ Cert_Q$
$c_2 := E(k, Sig_P(u, v)), Cert_P$
Q
k
P
k

Here, $\mathbb{G}$ is a cyclic group of prime order q generated by $g \in \mathbb{G}$, and $\alpha, \beta \in \mathbb{Z}_q$ are chosen at random. The session key k is computed as $k \leftarrow H(w)$, where $w = g^{\alpha\beta}$ and H is hash function. Also, $\mathcal{E} = (E, D)$ is a symmetric cipher.

(a) Suppose the signatures $Sig_Q(u, v)$ and $Sig_P(u, v)$ are not encrypted. Show that an adversary can easily carry out an identity misbinding attack, where Q thinks he is talking a corrupt user R. However, Q shares a key with P, and P thinks he is talking to Q.

(b) The STS protocol uses the session key k itself within the protocol to encrypt the signatures $Sig_Q(u, v)$ and $Sig_P(u, v)$. Suppose that a higher-level communication protocol uses the session key to encrypt messages using $\mathcal{E}$ and the key k, and that the adversary can force either party to encrypt a message of its choice. Show how to carry out the same identity misbinding attack from part (a), even when the signatures are encrypted.

(c) Suppose that we fix the protocol so that it derives two keys $(k', k) \leftarrow H(w)$, where k' is used to encrypt the signatures and k is used as the session key. Show that an adversary can still carry out the same identity misbinding attack.

Hint: The adversary does not follow the usual protocol for registration with the CA.

(d) Suppose we fix the protocol just as in part (c). Show another identity misbinding attack in which P and Q share a key, but P thinks he is talking to Q, while Q thinks he is talking to another instance of himself.

21.2 (An attack on an `AKE2` ***variant).*** Show that protocol `AKE2` may not be secure if the second signature does not include pk. To do this, you should start with a CCA-secure public-key encryption scheme and then "sabotage" it in an appropriate way so that it is still CCA secure, but when `AKE2` is instantiated with this sabotaged encryption scheme, there is a KCI vulnerability that leaves it open to a key exposure attack.

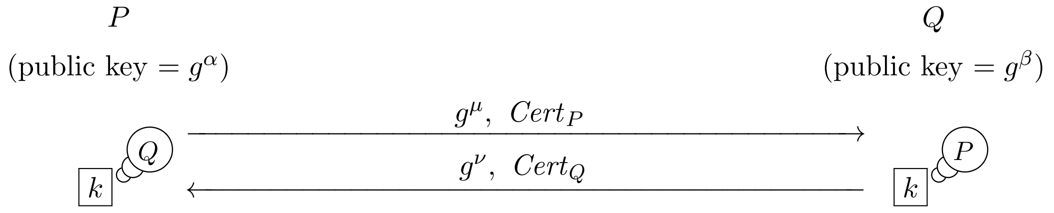
Hint: Assume P's signing key is exposed. The attack should expose P's session key, even though P thinks he is talking to an honest user Q whose signing key has not been exposed.

21.3 (An attack on an `AKE4` ***variant).*** Show that protocol `AKE4` may not be secure if the symmetric cipher does not provide ciphertext integrity. Specifically, assuming the symmetric cipher is a stream cipher, show that protocol `AKE4` is vulnerable to an identity misbinding attack.

21.4 (Strongly unpredictable ciphertexts). This exercise illustrates why the assumption that the encryption scheme in protocol `AKE3` has strongly unpredictable ciphertexts is necessary. To do this, you should start with a semantically secure public-key encryption scheme and then "sabotage" it in an appropriate way so that it is still semantically secure, but when `AKE3` is instantiated with this sabotaged encryption scheme, it is open to an attack of the following type. After making a couple of queries to user P's HSM, the adversary can impersonate P at will to any user Q, and the adversary will know the low-order bit of each resulting session key. Assume session keys are bit strings of some fixed length.

21.5 (An insecure variant of `AKE5`***).*** Suppose that we leave the group element $g^{\mu\nu}$ out of the hash in protocol `AKE5`. Show that this variant is not statically secure. In particular, show that it does not provide authenticity.

21.6 (Implicit authentication). Consider the following variant of protocol `AKE5`:



$$k := H(g^{(\alpha+\mu)(\beta+\nu)},\ g^{\mu\nu},\ g^{\alpha}, g^{\mu}, g^{\beta}, g^{\nu},\ id_P, id_Q)$$

Show that this protocol is not PFS secure.

Discussion: This protocol relies solely on implicit authentication: while each user needs to know their long term secret key to compute the session key, they do not need to know it to run the protocol. That is, neither user has to explicitly prove to the other that they know their long term secret key. In fact, any protocol with these properties cannot be PFS secure.

Note: This protocol does, in fact, satisfy our definition of static security, under appropriate assumptions.

21.7 (Insecure variants of `AKE1`***).*** For each of the insecure variants of `AKE1` described in Section 21.2.1, show how the formal definition of security (Definition 21.1) is violated.

21.8 (Security proof for `AKE2`***).*** Prove claims 1 and 2 in the proof of Theorem 21.2.

21.9 (Security proof for `AKE3`***).*** Prove claims 1 and 2 in the proof of Theorem 21.3.

21.10 (Security proof for `AKE4`***).*** Prove claims 1 and 2 in the proof of Theorem 21.4.

21.11 (TLS 1.3 is not a secure PAKE). In Section 21.10.1 we described the TLS 1.3 protocol for establishing a secure session between P and Q using a pre-shared key psk. Suppose psk is a password chosen at random from some small dictionary $\mathcal{D}$. Show that an active adversary can recover psk after a single attempted key exchange with Q. You may assume that the adversary already has $Cert_Q$. Your attack shows that one should not use TLS 1.3 as a PAKE. Specifically, psk should not be a human generated password.

21.12 (Non-interactive key exchange). Let $\mathbb{G}$ be a group of prime order q generated by $g \in \mathbb{G}$ and let $H : \mathbb{G} \times \mathbb{Z}_q^2 \to \mathcal{K}$ be a hash function. Consider a system with n users where, for $i = 1, \ldots, n$, user i chooses a random $\alpha_i \overset{\textsc{r}}{\leftarrow} \mathbb{Z}_q$, computes $h_i := g^{\alpha_i}$, and publishes (i, h_i) on a public bulletin board. Assume that no one can update i's entry on the board. Now, every pair of users $1 \leq i \leq j \leq n$ can establish an authenticated shared key $k_{i,j} := H(g^{\alpha_i \alpha_j}, i, j)$ without any interaction, other than reading the bulletin board. This is called **non-interactive key exchange** or **NIKE**. Our goal is to prove that this approach is secure.

(a) Adapt the definition of HSM security from Section 21.9.4 to the NIKE settings of this protocol.

(b) Prove that the NIKE protocol above is secure assuming ICDH holds in $\mathbb{G}$, when H is modeled as a random oracle.

Chapter 22

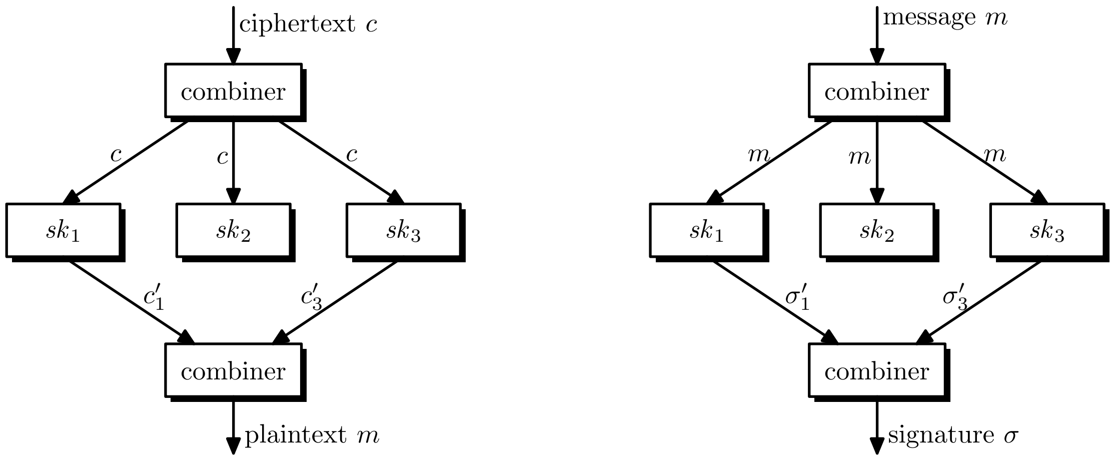
Threshold cryptography

In this chapter we explore **threshold cryptography**, an important technique used to protect secret keys in a public key cryptosystem. We will see how to apply this technique to protect the decryption key in a public key encryption scheme, and how to protect the signing key in a signature scheme. We begin with threshold decryption, which is a way to protect a decryption key.

Threshold decryption. Let (pk, sk) be a key pair in a public key encryption scheme. If an attacker steals sk, then all ciphertexts ever encrypted under pk can be decrypted by the attacker. For this reason, important secret keys are often stored in a special hardware device, called a **hardware security module**, or **HSM**, that responds to decryption requests, but never exports the secret key in the clear from the device. An attacker who compromises the computing environment can temporarily use the key by sending requests to the HSM, but cannot steal the key and use it, or sell it, later.

A different approach to protecting a secret key is to split it into a number of pieces, called **shares**. Each share is stored on a different machine, called a **decryption server**. All decryption servers must participate in a protocol in order to decrypt a ciphertext, and decryption fails if even one decryption server does not participate. In particular, to steal the secret key, an attacker must break the security of all the decryption servers, which can be harder than breaking the security of a single decryption server. Furthermore, if each decryption server stores its share in an HSM, then an attacker must compromise all of these HSMs to recover the key, which is much harder than compromising a single HSM.

In what follows, we use N to denote the total number of decryption servers, each of which stores a share of the secret key. Requiring all N decryption servers to participate in every decryption hurts availability: if even a single decryption server is unavailable for a period of time, decryption becomes impossible during that time; worse, if a single decryption server loses its share (say, because of a hardware failure), then decryption becomes permanently impossible. For this reason, we often require that decryption can proceed if t of the N decryption servers are available, for some parameter t in the range $1, \ldots, N$. For security, even if $t - 1$ decryption servers are compromised, and an attacker steals the shares stored by these compromised servers, the attacker should not learn anything useful about the secret key sk, and in particular, should not be able to decrypt ciphertexts on its own. Such a scheme is called a **t-out-of-N threshold decryption scheme**. For example, in a 3-out-of-5 scheme, stealing just two shares reveals nothing useful to the adversary, yet decryption can proceed even if two servers crash or otherwise lose their shares. Note that some



(a) **Threshold decryption** using two responses from three decryption servers. The combiner sends the ciphertext c to all three decryption servers, two servers respond, and the combiner obtains the plaintext m.

(b) **Threshold signing** using two responses from three signing servers. The combiner sends the message m to all three signing servers, two servers respond, and the combiner obtains the signature σ.

Figure 22.1: Threshold decryption and threshold signing

applications require larger values for t and N.

During the decryption protocol, the secret key *sk* should *never* be reconstituted in a single location, as doing so would create a single point of failure that an attacker can target to steal the key. In this chapter we will focus on a type of decryption protocol that is particularly simple and practical, in the sense that it requires minimal interaction. In this type of protocol, in addition to the N decryption servers, there is an additional entity called a **combiner**, which orchestrates the decryption protocol. One could also think of the combiner as a "decryption client". The combiner takes as input a ciphertext c to decrypt, and forwards c to all the decryption servers. Every available server applies its key share to c, and sends back a **decryption share**. Once the combiner receives t responses from the decryption servers, it can construct the complete decryption of c. The entire process is illustrated in Fig. 22.1a.

In Section 22.3 we will construct threshold decryption systems that are chosen-ciphertext secure. Moreover, the schemes we construct are also designed to be robust against decryption servers that behave incorrectly: if a decryption server returns an incorrect decryption share, the combiner can detect that the decryption server has misbehaved (in which case the combiner can report the misbehaving server and/or seek a correct decryption share from some other decryption server).

Threshold signatures. The same idea can be used to protect the signing key *sk* in a signature scheme. In a **t-out-of-N threshold signature scheme**, the signing key *sk* is split into N shares. Each share is stored on a different machine, called a **signing server**, and t signing servers must participate to sign a message. Even if $t-1$ signing servers are compromised, and an attacker steals the shares stored by these compromised servers, the attacker should not learn anything useful about the signing key *sk*, and in particular, should not be able to sign messages on its own. Moreover, if

$(N-t)$ servers crash, or lose their key shares, the remaining t servers can continue to sign messages. Just as for threshold decryption, it is essential that the signing key sk is never reconstituted in a single location. Fig. 22.1b illustrates the process of threshold signing.

In Section 22.2 we will construct threshold signature schemes that are designed to be secure and robust.

Distributed key generation. At this point the reader is probably wondering how the decryption and signing servers obtained the key shares $sk_1, \ldots, sk_N$ in the first place? A simple solution involves a trusted *key generator* that generates a key pair (pk, sk), and then splits sk into N shares $sk_1, \ldots, sk_N$. For $i = 1, \ldots, N$, the key generator sends share number i to decryption/signing server i, and then deletes sk. Often the machine that runs the key generator meets a violent end — such as melting its motherboard — to ensure that residual data on the machine does not expose sk.

This solution is a bit disappointing. We took special care to never reconstitute the key sk in a single location, and yet it is in a single location when generated. If the single key generator is attacked while it is generating the key, then sk will be compromised.

Fortunately, there is a better solution called **distributed key generation** or **DKG**. For all the threshold systems we describe, there are practical DKG protocols in which the N servers exchange messages so that at the end of the DKG protocol a public key pk is publicly known, and each server knows its own key share. Even if $t-1$ of the servers are already compromised while they are running the DKG protocol, the system still remains secure. We will discuss such DKG protocols in more detail in Section 22.4.

22.1 Shamir's secret sharing scheme

Many threshold cryptography schemes are built from an elegant general technique for splitting a secret into multiple shares. We first describe this technique, and in the next two sections show how to use it for threshold decryption and for threshold signatures.

Suppose Alice has a secret $\alpha \in Z$, where Z is some finite set. She wishes to generate N shares of α, each belonging to some finite set Z', and denoted $\alpha_1, \ldots, \alpha_N \in Z'$, so that the following property is satisfied:

> any t of the N shares are sufficient to reconstruct α, but every set of $t-1$ shares reveals nothing about α.

This sharing lets Alice give one share of her secret α to each of her N friends, so that any t friends can help her recover α, but $t-1$ friends learn nothing. A scheme satisfying this property is called a secret sharing scheme.

Definition 22.1. *A* ***secret sharing scheme over*** *Z is a pair of efficient algorithms (G, C):*

- *G is a probabilistic* ***sharing algorithm*** *that is invoked as $(\alpha_1, \ldots, \alpha_N) \xleftarrow{R} G(N, t, \alpha)$, where $0 < t \leq N$ and $\alpha \in Z$, to generate a t-out-of-N sharing of α.*
- *C is a deterministic* ***combining algorithm*** *that is invoked as $\alpha \leftarrow C(\mathcal{J}, \{\alpha_j\}_{j \in \mathcal{J}})$, where $\mathcal{J}$ is a subset of $\{1, \ldots, N\}$ of size t, to recover α.*

- **Correctness:** *we require that for every $\alpha \in Z$, for every possible output $(\alpha_1, \ldots, \alpha_N)$ of $G(N, t, \alpha)$, and every t-size subset $\mathcal{J}$ of $\{1, \ldots, N\}$, we have*
$$C(\mathcal{J}, \{\alpha_j\}_{j \in \mathcal{J}}) = \alpha.$$

Threshold secret sharing schemes often impose a bound on the parameter N. This bound can be modeled as a system parameter. Some schemes may impose additional constraints on the relationship between the parameters N and t. For any particular scheme, we will refer to **allowable parameters** as those (N, t) pairs that are allowed by the scheme.

Intuitively, a secret sharing scheme is secure if every set of $t - 1$ shares output by $G(N, t, \alpha)$ reveals nothing about α. To define this notion formally, it will be convenient to use the following notation: for a set $\mathcal{J} \subseteq \{1, \ldots, N\}$, we denote by $G(N, t, \alpha)[\mathcal{J}]$ the collection of shares $\{\alpha_j\}_{j \in \mathcal{J}}$, where the output of $G(N, t, \alpha)$ is $(\alpha_1, \ldots, \alpha_N)$.

Definition 22.2. *A secret sharing scheme (G, C) over Z is* **secure** *if for all allowable parameters $0 < t \leq N$, for every $\alpha, \alpha' \in Z$, and every subset $\mathcal{J}$ of $\{1, \ldots, N\}$ of size $t - 1$, the distribution $G(N, t, \alpha)[\mathcal{J}]$ is identical to the distribution $G(N, t, \alpha')[\mathcal{J}]$.*

The definition implies that by looking at $t - 1$ shares, one cannot tell if the secret is α or α', for all α and α' in Z. Hence, looking at only $t - 1$ shares reveals nothing about the secret. The definition is unconditional: even an infinitely powerful adversary cannot tell if the secret is α or α'.

22.1.1 Shamir secret sharing

An elegant secret sharing scheme over $\mathbb{Z}_q$, where q is prime, is due to Shamir. This scheme makes use of the following general fact about **polynomial interpolation**: a polynomial of degree at most $t-1$ is uniquely determined by t points on the polynomial. For example, two points determine a line, and three points determine a parabola. This general fact not only holds for the real and complex numbers, but over any algebraic domain in which all non-zero elements have a multiplicative inverse. Such a domain is called a *field*. When q is prime, $\mathbb{Z}_q$ is a field, and so this general fact holds here as well.

Shamir's scheme $(G_{\text{sh}}, C_{\text{sh}})$ is a t-out-of-N secret sharing scheme over $\mathbb{Z}_q$ that requires that $q > N$, and works as follows:

- $G_{\text{sh}}(N, t, \alpha)$: choose random $\kappa_1, \ldots, \kappa_{t-1} \xleftarrow{\text{R}} \mathbb{Z}_q$ and define the polynomial
$$\boldsymbol{\omega}(x) := \kappa_{t-1}x^{t-1} + \kappa_{t-2}x^{t-2} + \ldots + \kappa_1 x + \alpha \quad \in \mathbb{Z}_q[x].$$
Notice that $\boldsymbol{\omega}$ has degree at most $t - 1$ and that $\boldsymbol{\omega}(0) = \alpha$.

  For $i = 1, \ldots, N$ compute $\alpha_i \leftarrow \boldsymbol{\omega}(i) \in \mathbb{Z}_q$.

  Output the N shares $\alpha_1, \ldots, \alpha_N$.

- $C_{\text{sh}}(\mathcal{J}, \{\alpha_j\}_{j \in \mathcal{J}})$: since every polynomial of degree at most $t - 1$ is determined by its value at t points, the t shares α_j for $j \in \mathcal{J}$ completely determine the polynomial $\boldsymbol{\omega}$. Algorithm C_{sh} interpolates the polynomial $\boldsymbol{\omega}$ and outputs the constant term $\alpha := \boldsymbol{\omega}(0)$.

The description of algorithm C_{sh} needs a bit more explanation, which we provide by the following more general lemma, and which will be useful in other contexts as well. The lemma essentially says that the value of a polynomial of degree at most $t - 1$ at an arbitrary point is a linear combination of the value of the polynomial on any set of t points.

Lemma 22.1 (Linearity of interpolation). *Let q be a prime, let $\mathcal{J} \subseteq \mathbb{Z}_q$ be a set of size t, and let $j^* \in \mathbb{Z}_q$. Then there is a collection of values $\{\lambda_j\}_{j\in\mathcal{J}}$, with each $\lambda_j \in \mathbb{Z}_q$, such that the following holds:*

> *for every polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$, we have*
>
> $$\boldsymbol{\omega}(j^*) = \sum_{j\in\mathcal{J}} \lambda_j \boldsymbol{\omega}(j).$$

Moreover, the values λ_j for $j \in \mathcal{J}$ are efficiently computable given $\mathcal{J}$ and j^*.

Proof. This can be easily derived from the **Lagrange interpolation formula**, which states that for every polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$, we have

$$\boldsymbol{\omega} = \sum_{j\in\mathcal{J}} \boldsymbol{\omega}(j) \prod_{j'\in\mathcal{J}\setminus\{j\}} \frac{x-j'}{j-j'}.$$

Indeed, one can verify that if g is the polynomial of degree at most $t-1$ on the right hand side of this formula, then $\boldsymbol{\omega}(j) = g(j)$ for all $j \in \mathcal{J}$; from this, and the fact that a polynomial of degree at most $t-1$ is uniquely determined by t points on the polynomial, we see that $\boldsymbol{\omega} = g$.

Now just plug in j^* for x in this formula, and we obtain

$$\boldsymbol{\omega}(j^*) = \sum_{j\in\mathcal{J}} \boldsymbol{\omega}(j) \prod_{j'\in\mathcal{J}\setminus\{j\}} \frac{j^*-j'}{j-j'},$$

and the lemma follows by setting

$$\lambda_j := \prod_{j'\in\mathcal{J}\setminus\{j\}} \frac{j^*-j'}{j-j'} \qquad \text{for } j \in \mathcal{J}. \quad \square$$

The values λ_j in the above lemma are called **Lagrange interpolation coefficients**.

To implement $C_{\text{sh}}(\mathcal{J}, \{\alpha_j\}_{j\in\mathcal{J}})$, we can apply the above lemma with the given set $\mathcal{J}$ and $j^* := 0$ to compute the Lagrange interpolation coefficients $\{\lambda_j\}_{j\in\mathcal{J}}$, which satisfy

$$\boldsymbol{\omega}(0) = \sum_{j\in\mathcal{J}} \lambda_j \boldsymbol{\omega}(j)$$

for every polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$. It follows that

$$\alpha = \sum_{j\in\mathcal{J}} \lambda_j \alpha_j.$$

Hence, $C_{\text{sh}}(\mathcal{J}, \{\alpha_j\}_{j\in\mathcal{J}})$ operates in two steps: (i) compute the Lagrange interpolation coefficients $\{\lambda_j\}_{j\in\mathcal{J}}$, and (ii) compute the secret α as $\alpha := \sum_{j\in\mathcal{J}} \lambda_j \alpha_j$.

Besides implementing algorithm C_{sh}, Lemma 22.1 has other applications as well. For example, it allows us to do "interpolation in the exponent", as the following corollary shows:

Corollary 22.2 (Interpolation in the exponent). *Let $\mathbb{G}$ be a group of prime order q. Let $\mathcal{J} \subseteq \mathbb{Z}_q$ be a set of size t, and let $j^* \in \mathbb{Z}_q$. Given $\mathcal{J}$, j^*, as well as a collection of group elements of the form $\{h^{\boldsymbol{\omega}(j)}\}_{j \in \mathcal{J}}$, where $h \in \mathbb{G}$ and $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ is a polynomial of degree at most $t-1$, we can efficiently compute the group element $h^{\boldsymbol{\omega}(j^*)}$.*

Proof. To compute $h^{\boldsymbol{\omega}(j^*)}$, we first compute the Lagrange interpolation coefficients λ_j for $j \in \mathcal{J}$, as in Lemma 22.1. Then we have

$$h^{\boldsymbol{\omega}(j^*)} = h^{\sum_{j \in \mathcal{J}} \lambda_j \boldsymbol{\omega}(j)} = \prod_{j \in \mathcal{J}} (h^{\boldsymbol{\omega}(j)})^{\lambda_j}.$$

Thus, we compute each of the powers $(h^{\boldsymbol{\omega}(j)})^{\lambda_j}$ for $j \in \mathcal{J}$, and multiply these together. □

We stress that the algorithm in this corollary is not given as input the polynomial $\boldsymbol{\omega}$, or even the group element h. This corollary will be essential in several of our threshold schemes in this chapter, both in terms of implementing the schemes and in proving their security.

22.1.2 Security of Shamir secret sharing

It remains to show that Shamir's secret sharing scheme is secure, as in Definition 22.2.

Theorem 22.3. *Shamir's secret sharing scheme $(G_{\text{sh}}, C_{\text{sh}})$ is secure. In particular, if $0 < t \leq N < q$, then for every $\alpha \in \mathbb{Z}_q$, any collection of $t-1$ shares $\{\alpha_\ell\}_{\ell \in \mathcal{L}}$ is a mutually independent family of random variables, with each share uniformly distributed over $\mathbb{Z}_q$,*

Proof. Let α be a fixed value in $\mathbb{Z}_q$ and let $j'_1, \ldots, j'_t$ be fixed, distinct, non-zero values in $\mathbb{Z}_q$. Consider the map that sends

$$(\kappa_1, \ldots, \kappa_{t-1}) \in \mathbb{Z}_q^{t-1} \qquad \text{to} \qquad (\alpha'_1, \ldots, \alpha'_{t-1}) \in \mathbb{Z}_q^{t-1},$$

where $\alpha'_\ell := \boldsymbol{\omega}(j'_\ell)$ for $\ell = 1, \ldots, t-1$ and $\boldsymbol{\omega} := \alpha + \kappa_1 x + \cdots + \kappa_{t-1} x^{t-1}$.

Claim: this map is one-to-one.

The theorem follows from the claim, since if $(\kappa_1, \ldots, \kappa_{t-1})$ is chosen uniformly over $\mathbb{Z}_q^{t-1}$, then $(\alpha'_1, \ldots, \alpha'_{t-1})$ must also be uniformly distributed over $\mathbb{Z}_q^{t-1}$.

To prove the claim, suppose by way of contradiction that this map is not one-to-one. This would imply the existence of two distinct polynomials $g, h \in \mathbb{Z}_q[x]$ of degree at most $t-2$, such that the polynomials $\alpha + xg$ and $\alpha + xh$ agree at the $t-1$ *non-zero* points $j'_1, \ldots, j'_{t-1}$. But then this implies that g and h themselves agree at these same $t-1$ points, which contradicts our basic fact about polynomial interpolation. □

Remark 22.1. In presenting Shamir's secret sharing scheme, we have streamlined the notation a bit by using the values $j = 1, \ldots, N$ as the points at which we evaluate the polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ — we are implicitly identifying the integer j with its image in $\mathbb{Z}_q$ under the natural map from $\mathbb{Z}$ onto $\mathbb{Z}_q$. More generally, we could use an arbitrary, fixed sequence $(\beta_1, \ldots, \beta_N)$ of distinct, non-zero values in $\mathbb{Z}_q$ as evaluation points. Everything we present in this chapter can be trivally adapted to such general sequences of evaluation points. Indeed, Shamir secret sharing works perfectly well over finite fields other than $\mathbb{Z}_q$, and in such settings, one might have to use a more general sequence of evaluation points. □

22.2 Threshold signatures

In this section, we turn to the problem of designing threshold signature schemes. We begin by defining the basic syntax of a threshold signature scheme.

Definition 22.3. *A **threshold signature scheme** (G, S, V, C) is a tuple of four efficient algorithms:*

- *G is a probabilistic **key generation algorithm** that is invoked as*

  $$(pk, pkc, sk_1, \ldots, sk_N) \overset{\text{R}}{\leftarrow} G(N, t)$$

  *to generate a t-out-of-N shared key. It outputs a **public key** pk, a **combiner public key** pkc, and N **signing key shares**, $sk_1, \ldots, sk_N$.*

- *S is a (possibly) probabilistic **signing algorithm** that is invoked as $\sigma'_i \overset{\text{R}}{\leftarrow} S(sk_i, m)$, where sk_i is one of the key shares generated by G, m is a message, and σ'_i is a signature share for m using sk_i.*

- *V is a deterministic **verification algorithm** as in a signature scheme, invoked as $V(pk, m, \sigma)$ and outputs either* accept *or* reject*.*

- *C is a deterministic **combiner algorithm** that is invoked as $\sigma \leftarrow C(pkc, m, \mathcal{J}, \{\sigma'_j\}_{j \in \mathcal{J}})$, where pkc is the combiner public key, m is a message, $\mathcal{J}$ is a subset of $\{1, \ldots, N\}$ of size t, and each σ'_j is a signature share for m. The algorithm either outputs a signature σ, or outputs a special message* blame$(\mathcal{J}^*)$*, where $\mathcal{J}^*$ is a nonempty subset of $\mathcal{J}$.*

  Intuitively, the message blame$(\mathcal{J}^*)$ *indicates that the provided signature shares σ'_j for $j \in \mathcal{J}^*$, are invalid.*

- ***Correctness:** as usual, the verification algorithm should accept a properly constructed signature; specifically, for all possible outputs $(pk, pkc, sk_1, \ldots, sk_N)$ of $G(N, t)$, all messages m, and all t-size subsets $\mathcal{J}$ of $\{1, \ldots, N\}$, we have*

  $$\Pr\Big[V\big(pk,\ m,\ C(pkc,\ m,\ \mathcal{J},\ \{S(sk_j, m)\}_{j \in \mathcal{J}})\big) = \mathsf{accept}\Big] = 1.$$

In the above definition, messages lie in some finite message space $\mathcal{M}$ and signatures in some finite signature space Σ. We say that the signature scheme is defined over $(\mathcal{M}, \Sigma)$. Also, just as for a secret sharing scheme, a threshold signature scheme may impose constraints on the parameters N and t (such as an upper bound on N and on the relationship between N and t). For any particular scheme, we will refer to **allowable parameters** as those (N, t) pairs that are allowed by the scheme.

Note that Definition 22.3 does not require that the sk_i's are actually shares of any particular signing key sk, for some non-threshold signature scheme, but for many threshold signature schemes, this is indeed the case.

Let's map these four algorithms back to Fig. 22.1b. Algorithm $G(N, t)$ is used to provision the signing servers with their signing key shares $sk_1, \ldots, sk_N$. Next, when the combiner requests a signature on some message m, each signing server examines the message and decides if it wants to sign it. If server i is willing to sign, it computes $\sigma'_i \overset{\text{R}}{\leftarrow} S(sk_i, m)$ and sends the resulting signature share σ'_i to the combiner. Once the combiner receives t signature shares from distinct signing servers, it runs algorithm C to (hopefully) obtain either a valid signature on m or a set of indices $\mathcal{J}^*$ that identifies the misbehaving servers that contributed invalid shares. Note that the above definition *does not* actually require that a signature produced by the combiner algorithm is valid or that the set of indices $\mathcal{J}^*$ actually identifies misbehaving servers — such a requirement is a **robustness** property that will be formally defined below. If a scheme is robust in this sense, then if the t shares collected by the combiner do not yield a valid signature, the set $\mathcal{J}^*$ may be used to discard invalid shares from misbehaving servers, and the combiner may then seek out valid shares from other servers. This process may continue until a valid signature is obtained, provided there are at least t signing servers that are available and behaving correctly.

Later, after we look at some constructions, we will formally define what it means for a threshold signature scheme to be secure. However, let us give an informal definition of security here. We will assume that an adversary has compromised $t - 1$ signing servers, obtaining their signing key shares. During the attack, the adversary may ask any of the uncompromised servers for signature shares on any messages of its choice. Security will mean that the adversary cannot forge a valid signature on any *other* message.

Note that this security definition rules out a broad range of attacks. First, it rules out a key-recovery attack: if an attacker steals the signing key shares from at most $t-1$ servers, the attacker cannot reconstruct the signing key itself. Second, it also rules out attacks in which an attacker corrupts the behavior of some (but fewer than t) of the signing servers, even if it does not obtain their signing key shares. Indeed, in any application, the signing servers will be implementing some particular policy that determines which messages should be signed under what circumstances, and will only issue signature shares if this policy is satisfied. Security means that if at most $t-1$ servers are corrupted, in the sense that either (a) their signing key shares have been stolen or (b) they do not implement the signing policy correctly, then an attacker still cannot get a valid signature on a message unless at least one of the uncompromised servers determines that the policy is satisfied and issues a signature share. We should note, conversely, that if an attacker can get $t-1$ compromised servers to issue signature shares on a given message (either by stealing their signing key shares or by subverting their signing policy), then the attacker needs to get just one of the uncompromised servers to issue a signature share on that message in order to obtain a valid signature on that message.

For example, a certificate authority (CA) is supposed to validate certain credentials before issuing a certificate (see Section 13.8). Performing this validation correctly is just as important as securely storing the signing key. To improve the security of the CA, we may implement it as a distributed system using a threshold signature scheme, where each signing server stores one signing key share and independently validates credentials before issuing a signature share. Security means that if at most $t - 1$ servers are corrupted, in the sense that either (a) their signing key shares have been stolen or (b) they do not implement credential validation correctly, then an attacker still cannot get a valid certificate unless it presents valid credentials to at least one of the uncompromised servers.

Remark 22.2. Definition 22.3 requires that t and N be specified at key generation time. However, the schemes in this section can be extended so that both t and N can be changed after the secret key shares are generated, without changing the public key pk. See Exercise 22.4. □

22.2.1 A generic threshold signature scheme

Every secure signature scheme (G, S, V) can be trivially transformed into a threshold signature scheme (G', S', V', C') as follows:

- During setup, the key generator creates N key pairs $(pk_i, sk_i) \xleftarrow{R} G()$, for $i = 1, \ldots, N$, and gives key sk_i to signing server i. The public verification key is $pk := (pk_1, \ldots, pk_N)$, and the combiner public key is the same, $pkc := pk$. More precisely, we define threshold key generation $G'(N, t)$ as an algorithm that outputs $(pk, pk, sk_1, \ldots, sk_N)$.

- To sign a message m, signing server i, for $i \in \{1, \ldots, N\}$, outputs the signature share $\sigma'_i \xleftarrow{R} S(sk_i, m)$.

- The combiner collects t signature shares from the signing servers, and outputs all t of them as the full signature. In more detail, algorithm $C'(pkc, m, \mathcal{J}, \{\sigma'_j\}_{j \in \mathcal{J}})$, where $pkc = (pk_1, \ldots, pk_N)$ and $|\mathcal{J}| = t$, first checks that each σ'_j is a valid signature on m under pk_j. If not, it outputs $\mathsf{reject}(\mathcal{J}^*)$, where $\mathcal{J}^*$ is the set of indices j for which σ'_j is invalid. Otherwise, it outputs the signature $\sigma := (\mathcal{J}, \{\sigma'_j\}_{j \in \mathcal{J}})$.

- Finally, algorithm $V'(pk, m, \sigma)$, where $pk = (pk_1, \ldots, pk_N)$ and $\sigma = (\mathcal{J}, \{\sigma'_j\}_{j \in \mathcal{J}})$, outputs accept if $|\mathcal{J}| = t$ and each σ'_j is a valid signature on m under pk_j.

This scheme satisfies the security and robustness requirements for a threshold signature scheme (see Exercise 22.3). However, one major drawback of this scheme is that its performance degrades as t grows. In particular, the size of the signature grows linearly in t, as does the time to verify a signature. Our goal is to construct a threshold signature scheme where the size of the signature, and the time to verify it, are *independent* of the parameters N and t.

Another drawback of this scheme is that the threshold t is publicly known. Anyone can examine the public verification algorithm V' and learn t. As a result, an adversary knows exactly how many signing servers it needs to corrupt in order to steal the secret signing key sk. Generally, it is desirable that the threshold scheme not reveal the threshold t to the public. The scheme that we construct next has the property that t is only known to the combiner, and to no one else.

Remark 22.3 (Decentralized key provisioning). This scheme has the additional benefit that we do not need a trusted key generator, and thus can eliminate that single point of failure. Indeed, server i can generate the key pair (pk_i, sk_i) on its own; moreover, corrupt servers are allowed to generate their public keys in an arbitrary fashion (not necessarily by running G, and possibly depending on the public keys of the honest servers). We call this type of setup **decentralized key provisioning**. □

22.2.2 BLS threshold signing

The BLS signature scheme (Section 15.5.1) supports a very efficient threshold signing mechanism, even for large t and N. A threshold signature looks like a regular (non-threshold) BLS signature.

First, let us briefly recall how the BLS signature scheme works. Let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$. In addition, we assume there is an efficient algorithm $\mathcal{O}_{\text{DDH}}$ that solves the decision Diffie-Hellman (DDH) problem in $\mathbb{G}$. In particular, for all $\alpha, \beta \in \mathbb{Z}_q$ algorithm $\mathcal{O}_{\text{DDH}}$ satisfies

$$\mathcal{O}_{\text{DDH}}(g^\alpha, g^\beta, g^\gamma) = \begin{cases} \mathsf{accept} & \text{if } \gamma = \alpha\beta; \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

Recall that in Section 15.4 we saw that a symmetric pairing in a group $\mathbb{G}$ gives an efficient algorithm for DDH in $\mathbb{G}$. In particular, (15.8) gives an efficient implementation for $\mathcal{O}_{\text{DDH}}$. Then in Section 15.5.1 we presented the BLS scheme using a pairing. Here we present BLS more abstractly, using a generic algorithm for DDH in $\mathbb{G}$.

The BLS signature scheme (G, S, V_{BLS}) uses a hash function $H : \mathcal{M} \to \mathbb{G}$ and works as follows:

- $G() \to (pk, sk)$: choose a random $\alpha \xleftarrow{R} \mathbb{Z}_q$, compute $u \leftarrow g^\alpha$, and output $pk := u \in \mathbb{G}$ and $sk := \alpha$.
- $S(sk, m) \to \sigma$: output $\sigma := H(m)^\alpha \in \mathbb{G}$, where $sk = \alpha$.
- $V_{\text{BLS}}(pk, m, \sigma)$: output $\mathcal{O}_{\text{DDH}}(pk,\ H(m),\ \sigma)$.

Indeed, for a public key $pk = g^\alpha$, a message $m \in \mathcal{M}$, and a valid signature $\sigma = H(m)^\alpha$, the tuple $\big(pk = g^\alpha,\ H(m) = g^\beta,\ \sigma = g^{\alpha\beta}\big)$ is a DDH tuple and V_{BLS} outputs accept. In Section 15.5.1 we showed that the scheme is secure when CDH holds in $\mathbb{G}$ and H is modeled as a random oracle.

Remark 22.4. In Section 15.5.1 we presented the BLS scheme using an asymmetric pairing. While our discussion of threshold BLS will use the abstract description of BLS presented above, the construction generalizes almost verbatim to threshold BLS using an asymmetric pairing. □

22.2.2.1 The BLS threshold signature scheme

To make BLS into a threshold signature scheme we will apply Shamir's secret sharing scheme $(G_{\text{sh}}, C_{\text{sh}})$ to the secret key $\alpha \in \mathbb{Z}_q$. We will show that a signature can be generated without ever reconstituting the secret key α at a single location.

The scheme. We describe the BLS threshold signature scheme $\mathcal{S}_{\text{thBLS}} = (G', S', V', C')$ using the same group $\mathbb{G}$ and hash function $H : \mathcal{M} \to \mathbb{G}$ as in BLS. The scheme uses the BLS signature verification algorithm V_{BLS} and works as follows.

- $G'(N, t) \to (pk, pkc, sk_1, \ldots, sk_N)$:
  - choose a random $\alpha \xleftarrow{R} \mathbb{Z}_q$ and compute $pk \leftarrow g^\alpha$;
  - run $G_{\text{sh}}(N, t, \alpha)$ to obtain N shares $\alpha_1, \ldots, \alpha_N \in \mathbb{Z}_q$;
    Note: recall that for some polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$, we have $\boldsymbol{\omega}(0) = \alpha$ and $\boldsymbol{\omega}(i) = \alpha_i$ for $i = 1, \ldots, N$;
  - for $i = 1, \ldots, N$, compute $u_i \leftarrow g^{\alpha_i}$, and set $sk_i := \alpha_i$ and $pk_i := u_i$;
  - set $pkc := (pk_1, \ldots, pk_N)$;
  - output $(pk,\ pkc,\ sk_1, \ldots, sk_N)$.
- $S'(sk_i, m) \to \sigma'_i$, where $sk_i = \alpha_i$: output the signature share $\sigma'_i \leftarrow H(m)^{\alpha_i} \in \mathbb{G}$.

  Note: observe that σ'_i is a regular BLS signature on m with respect to the public key $u_i = g^{\alpha_i}$.
- $C'(pkc,\ m,\ \mathcal{J},\ \{\sigma'_j\}_{j \in \mathcal{J}}) \to \sigma$, where $\mathcal{J}$ is a subset of $\{1, \ldots, N\}$ of size t, and

$$pkc = (pk_1, \ldots, pk_N) = (u_1, \ldots, u_N):$$

- *step 1:* verify that all t signature shares are valid: let $\mathcal{J}^*$ be the set of all $j \in \mathcal{J}$ such that
$$V_{\text{BLS}}(u_j, m, \sigma'_j) = \mathsf{reject};$$
if $\mathcal{J}^*$ is nonempty, output $\mathsf{blame}(\mathcal{J}^*)$ and abort;

  Note: assuming all the shares are valid, then for $j \in \mathcal{J}$, we have $\sigma'_j = H(m)^{\boldsymbol{\omega}(j)}$, where $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ is the polynomial of degree at most $t-1$ used to share the secret key α;

  Note: Remark 15.5 shows that one can verify the t signature shares as a batch far more efficiently than verifying them one by one;
- *step 2:* compute the signature: interpolate in the exponent by applying Corollary 22.2 with the given set $\mathcal{J}$ and $j^* := 0$ to compute the signature $\sigma := H(m)^\alpha = H(m)^{\boldsymbol{\omega}(0)}$ from the t group elements $\sigma'_j = H(m)^{\boldsymbol{\omega}(j)}$ for $j \in \mathcal{J}$; output σ as the signature on m;

  more explicitly, the signature σ is computed as
$$\sigma \leftarrow \prod_{j \in \mathcal{J}} (\sigma'_j)^{\lambda_j} \quad \in \mathbb{G}$$
where the exponents $\{\lambda_j\}_{j \in \mathcal{J}}$ are the Lagrange interpolation coefficients, which depend only on the the set $\mathcal{J}$, as computed in Lemma 22.1.

- $V'(pk, m, \sigma)$: output $V_{\text{BLS}}(pk, m, \sigma)$.

The size of a signature σ and the time to verify it are independent of the parameters N and t. Moreover, the verification algorithm V' is independent of t, so that the public learns nothing about the threshold t.

Further enhancements. The BLS threshold signature scheme can be strengthened in several ways. First, the system easily generalizes to more flexible access structures than strict threshold. For example, it is easy to extend the scheme to support the following access structure: signing is possible if signing server number 1 participates, and at least t of the remaining $N-1$ signing servers participate. We explore more general access structures in Section 22.5 and Exercise 22.2.

Another enhancement, called **proactive security**, further strengthens the system by forcing the adversary to break into t signing servers *within a short period of time*, say ten minutes [87]. Otherwise, the adversary gets nothing. This is done by having the key servers proactively refresh the sharing of their secret key every ten minutes, without changing the public key. We discuss this in Exercise 22.5.

The shared secret signing key α can be generated using a **distributed key generation** (DKG) protocol to ensure that α is never found in a single location, not even during key generation — see Section 22.4.

Finally, as discussed in Remark 22.4, the BLS threshold signature scheme can easily be adapted to the asymmetric pairing setting. Using the notation from Section 15.5.1, the public key and the combiner public key for threshold BLS would be in the group $\mathbb{G}_1$, while the signatures and signature shares would be in the group $\mathbb{G}_0$. All of the results we prove below for threshold BLS in the symmetric setting hold as well in the asymmetric setting.

22.2.3 Threshold signature security

We next prove that the BLS threshold scheme is secure. To do so we must first define what it means for a t-out-of-N threshold signature scheme to be secure. The security model gives the adversary two capabilities:

- We allow the adversary to completely control up to $t-1$ of the signing servers. The adversary must declare at the beginning of the game a set $\mathcal{L}$ of up to $t-1$ servers that it wants to control.[1]
- In addition, the combiner is not a trusted party, and may be controlled by the adversary. This means that a scheme that temporarily reconstitutes the signing key at the combiner would be insecure. To capture the idea that the combiner may be controlled by the adversary, we allow the adversary to request signatures on messages of its choice, and we then give the adversary all N signature shares generated by the signing servers in response to these requests.

Even with control of the combiner, and up to $t-1$ servers, the adversary should not be able to break the signature scheme. In particular, it should be unable to create an existential forgery under a chosen message attack. Formally, we define security using the following attack game.

Attack Game 22.1 (threshold signature security). For a given threshold signature scheme $\mathcal{S} = (G, S, V, C)$, defined over $(\mathcal{M}, \Sigma)$, and a given adversary $\mathcal{A}$, we define the following attack game.

- *Setup:* the adversary sends poly-bounded allowable parameters N and t, where $0 < t \leq N$, and a subset $\mathcal{L} \subseteq \{1, \ldots, N\}$ of size $t-1$ to the challenger. The challenger runs

$$(pk, pkc, sk_1, \ldots, sk_N) \xleftarrow{R} G(N, t),$$

  and sends pk, pkc, and $\{sk_\ell\}_{\ell \in \mathcal{L}}$ to the adversary.
- *Signing queries:* for $j = 1, 2, \ldots$, signing query j from $\mathcal{A}$ is a message $m_j \in \mathcal{M}$. Given m_j, the challenger computes all N signature shares $\sigma'_{j,i} \xleftarrow{R} S(sk_i, m_j)$ for $i = 1, \ldots, N$. It sends $(\sigma'_{j,1}, \ldots, \sigma'_{j,N})$ to the adversary.
- *Forgery:* eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, \sigma) \in \mathcal{M} \times \Sigma$.

We say that the adversary wins the game if the following two conditions hold:

- $V(pk, m, \sigma) = \mathsf{accept}$, and
- m is new, namely, $m \notin \{m_1, m_2, \ldots\}$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$ denoted thSIG$\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 22.4 (secure threshold signatures). *We say that a threshold signature scheme $\mathcal{S}$ is* ***secure*** *if for all efficient adversaries $\mathcal{A}$, the quantity* thSIG$\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

[1] This is what is known as a **static corruption model**. A stronger model, called the **adaptive corruption model**, allows the adversary to adaptively choose which signing servers to corrupt as the attack game proceeds.

22.2.3.1 Robust threshold signatures

Beyond security for the signature scheme, a threshold signature scheme should also be robust against a malicious signing server that is trying to disrupt the signature generation process. We define two such robustness properties that any practical threshold signature scheme should provide.

Suppose a combiner receives a collection $\{\sigma'_j\}_{j\in\mathcal{J}}$ of t signature shares on a given message m. By Definition 22.3, the combiner will output either a signature σ, or a special message $\mathsf{blame}(\mathcal{J}^*)$, where $\mathcal{J}^*$ is a nonempty subset of $\mathcal{J}$.

- If the output of the combiner is $\mathsf{blame}(\mathcal{J}^*)$, then we would like it to be the case that all of the signature shares σ'_j for $j \in \mathcal{J}^*$ are "bad", in the sense that they were incorrectly generated by misbehaving signing servers. A threshold signature scheme that guarantees that this always happens is said to provide **accurate blaming**.
- Otherwise, the output of the combiner is a signature σ on the given message m. A threshold signature scheme is said to be **consistent** if no efficient adversary can cause the combiner to output a signature that is invalid on m, with non-negligible probability. When the combiner outputs a valid signature we say that the given collection of shares $\{\sigma'_j\}_{j\in\mathcal{J}}$ is **valid** for m.

A threshold signature scheme is **robust** if it satisfies both properties.

In practice, robustness allows a combiner who is trying to construct a valid signature on m to proceed as follows. If the combiner algorithm outputs $\mathsf{blame}(\mathcal{J}^*)$, then the combiner can discard the "bad" shares in $\mathcal{J}^*$, and seek out $t - |\mathcal{J}^*|$ "good" shares from among the remaining signing servers. As long as there are t correctly behaving servers available, the accurate blaming property guarantees that the combiner can repeat this process until it gets a valid collection of shares, and the consistency property guarantees that when this happens, the resulting signature must be valid (with overwhelming probability).

We now define these two properties formally.

Definition 22.5 (accurate blaming). *We say that a threshold signature scheme $\mathcal{S} = (G, S, V, C)$ provides **accurate blaming** if the following holds:*

> *for all possible outputs $(pk, pkc, sk_1, \ldots, sk_N)$ of $G(N, t)$, all messages m, all t-size subsets $\mathcal{J}$ of $\{1, \ldots, N\}$, and all collections of signature shares $\{\sigma'_j\}_{j\in\mathcal{J}}$:*
>
> $$C\left(pkc, m, \mathcal{J}, \{\sigma'_j\}_{j\in\mathcal{J}}\right) = \mathsf{blame}(\mathcal{J}^*) \quad\Longrightarrow\quad \Pr\bigl[S(sk_j, m) = \sigma'_j\bigr] = 0 \textit{ for all } j \in \mathcal{J}^*.$$

In other words, if the combiner algorithm outputs $\mathsf{blame}(\mathcal{J}^*)$, then σ'_j, for $j \in \mathcal{J}^*$, cannot be an output of $S(sk_j, m)$. This ensures that an honest server that outputs $S(sk_j, m)$ cannot be blamed. The requirement is unconditional: not even an infinitely powerful adversary can cause an honest server to be blamed.

The *consistency* property is defined using an attack game.

Attack Game 22.2 (consistent threshold signatures). For a given threshold signature scheme $\mathcal{S} = (G, S, V, C)$, defined over $(\mathcal{M}, \Sigma)$, and a given adversary $\mathcal{A}$, we define the following attack game.

- The adversary sends to the challenger poly-bounded allowable parameters N and t, where $0 < t \le N$.

- The challenger runs $(pk, pkc, sk_1, \ldots, sk_N) \xleftarrow{R} G(N,t)$ and sends all this data to the adversary.
- The adversary outputs a message m, a subset $\mathcal{J}$ of $\{1, \ldots, N\}$ of size t, and a collection $\{\sigma'_j\}_{j \in \mathcal{J}}$ of signature shares.
- We say the adversary wins the game if $C(pkc, m, \mathcal{J}, \{\sigma'_j\}_{j\in\mathcal{J}})$ outputs a signature σ (rather than a "blame" message), but $V(pk, m, \sigma) = \mathsf{reject}$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$, denoted $\text{conSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 22.6 (consistent threshold signatures). *We say that a threshold signature scheme $\mathcal{S}$ is **consistent** if for all efficient adversaries $\mathcal{A}$, the quantity* $\text{conSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

At first it may appear that every threshold signature scheme (G, S, V, C) can be easily made consistent: construct a new scheme (G, S, V, C') where $C'(pkc, m, \mathcal{J}, \{\sigma'_j\}_{j\in\mathcal{J}})$ works as follows: (i) run C on the same input, (ii) if C outputs a signature σ, check that σ is valid for m, (iii) if it is valid, then output σ; otherwise abort. This ensures that if C' outputs a signature, the signature must be valid for m. However, this does not work because C cannot simply abort: if it does not output a signature, it is required to flag at least one of the provided signature shares as invalid. However, when C' obtains an invalid signature from C, it does not know which signature share to blame. Consequently, we must require consistency as an explicit property.

Definition 22.7 (robustness). *We say that a threshold signature scheme is **robust** if it* provides accurate blaming *and is* consistent*.*

Remark 22.5. While we require accurate blaming to be perfect and unconditional, we only require consistency against an efficient adversary. The reason is that some threshold schemes, such as the threshold RSA signature scheme (Exercise 22.10), do not satisfy perfect unconditional consistency. We note that the BLS threshold signature scheme does provide perfect and unconditional consistency: if the combiner outputs a signature, we are guaranteed that the signature is always valid for the message being signed. □

22.2.4 Security of threshold BLS

Next, we argue that the BLS threshold signature scheme is secure and robust. Security follows directly from the security of the BLS signature scheme. Robustness holds unconditionally.

Theorem 22.4. *If $\mathcal{S}_{\text{BLS}}$ is a secure signature scheme, then $\mathcal{S}_{\text{thBLS}}$ is a secure threshold signature scheme.*

> *In particular, for every adversary $\mathcal{A}$ that attacks $\mathcal{S}_{\text{thBLS}}$ as in Attack Game 22.1, there exists an adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, that attacks $\mathcal{S}_{\text{BLS}}$ as in Attack Game 13.1, such that*
>
> $$\text{thSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{thBLS}}] = \text{SIG}\mathsf{adv}[\mathcal{B}, \mathcal{S}_{\text{BLS}}].$$

Proof. We design $\mathcal{B}$ to play the role of challenger to $\mathcal{A}$. Recall that $\mathcal{A}$ begins by outputting the parameters N and t, and a subset $\mathcal{L}$ of size $t-1$ of $\{1, \ldots, N\}$. Now, when $\mathcal{B}$ receives $pk = g^\alpha$ from its own challenger, $\mathcal{B}$ needs to provide to $\mathcal{A}$ not only pk, but also $t-1$ key shares corresponding to $\mathcal{L}$, as well as pkc.

To do this, $\mathcal{B}$ sets $\alpha_\ell \xleftarrow{\text{R}} \mathbb{Z}_q$ for each $\ell \in \mathcal{L}$, and sends to $\mathcal{A}$ the $t-1$ key shares $sk_\ell = \alpha_\ell$ for $\ell \in \mathcal{L}$. By Theorem 22.3, these key shares have the same distribution as in an actual run of Attack Game 22.1.

We know that there is a unique polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$ such that $\boldsymbol{\omega}(0) = \alpha$ and $\boldsymbol{\omega}(\ell) = \alpha_\ell$ for each $\ell \in \mathcal{L}$. Moreover, for $i \in \{1, \ldots, N\} \setminus \mathcal{L}$, the key share $sk_i = \alpha_i$ satisfies $\boldsymbol{\omega}(i) = \alpha_i$.

Adversary $\mathcal{B}$ also has to give pkc to $\mathcal{A}$. Recall that $pkc = (u_1, \ldots, u_N)$, where $u_i = g^{\alpha_i} = g^{\boldsymbol{\omega}(i)}$ for $i = 1, \ldots, N$. For $\ell \in \mathcal{L}$, adversary $\mathcal{B}$ can compute $u_\ell = g^{\alpha_\ell}$ directly, since $\mathcal{B}$ knows the value α_ℓ. For each $i \in \{1, \ldots, N\} \setminus \mathcal{L}$, adversary $\mathcal{B}$ can compute u_i via "interpolation in the exponent", by applying Corollary 22.2 with $\mathcal{J} := \mathcal{L} \cup \{0\}$ and $j^* := i$ to compute $u_i = g^{\boldsymbol{\omega}(i)}$ from the collection of group elements $\{g^{\boldsymbol{\omega}(j)}\}_{j \in \mathcal{J}}$; more explicitly, $\mathcal{B}$ computes u_i as

$$u_i \leftarrow pk^{\lambda_{i0}} \cdot g^{\sum_{\ell \in \mathcal{L}} \alpha_\ell \cdot \lambda_{i\ell}},$$

where the λ_{ij}'s are the Lagrange interpolation coefficients computed as in Lemma 22.1.

Next, when $\mathcal{A}$ issues a signing query for a message $m \in \mathcal{M}$, adversary $\mathcal{B}$ forwards this query to its own challenger and gets back $\sigma = H(m)^\alpha = H(m)^{\boldsymbol{\omega}(0)}$. Now $\mathcal{B}$ needs to send to $\mathcal{A}$ all the signature shares for m. That is, $\mathcal{B}$ needs to construct $\sigma'_1, \ldots, \sigma'_N$, without knowing the secret key α. Recall that for $i = 1, \ldots, N$, we have $\sigma'_i = H(m)^{\alpha_i} = H(m)^{\boldsymbol{\omega}(i)}$. For $\ell \in \mathcal{L}$, adversary $\mathcal{B}$ can compute $\sigma'_\ell = H(m)^{\alpha_\ell}$ directly, since it knows the value α_ℓ. For $i \in \{1, \ldots, N\} \setminus \mathcal{L}$, adversary $\mathcal{B}$ can compute σ'_i again via "interpolation in the exponent", by again applying Corollary 22.2, with $\mathcal{J}$ and $j^* = 0$ as in the previous paragraph, to compute $\sigma'_i = H(m)^{\alpha_i} = H(m)^{\boldsymbol{\omega}(i)}$ from the collection of group elements $\{H(m)^{\boldsymbol{\omega}(j)}\}_{j \in \mathcal{J}}$; more explicitly, $\mathcal{B}$ computes σ'_i as

$$\sigma'_i \leftarrow \sigma^{\lambda_{i0}} \cdot H(m)^{\sum_{\ell \in \mathcal{L}} \alpha_\ell \cdot \lambda_{i\ell}},$$

where the λ_{ij}'s are the same Lagrange interpolation coefficients used above.

When eventually $\mathcal{A}$ outputs a signature forgery (m, σ), our adversary $\mathcal{B}$ outputs the same (m, σ). Now $\mathcal{B}$ has the same advantage in its attack game that $\mathcal{A}$ has in its attack game, which completes the proof. □

Theorem 22.5. *$\mathcal{S}_{\text{thBLS}}$ is a robust threshold signature scheme. In particular, every adversary has advantage zero in Attack Game 22.2.*

Proof. For accurate blaming, it is easy to see that a correctly generated signature share will never be flagged as invalid. Conversely, for consistency, it is easy to see that any valid collection of signature shares will combine to yield a valid signature. □

This completes our discussion of the BLS threshold signature scheme. We showed that the scheme is secure and robust.

22.2.5 Accountability versus privacy

Secure threshold signatures used in practice come in two flavors. One is called *accountable threshold signatures* and the other is called *private threshold signatures*.

- An **accountable threshold signature** scheme is a threshold signature scheme where a valid signature identifies a set of parties that must have participated in generating the signature.

For example, consider a 3-out-of-5 threshold setup, and let σ be a valid signature on a message m. Then anyone should be able to examine σ and learn the identity of three parties that must have participated in generating σ. For security we require that a set of signers cannot generate a signature σ that falsely makes it look as if some other signer, not in the set, participated in generating σ.

- A **private threshold signature** scheme is the opposite: it is a threshold signature scheme where a valid signature reveals nothing about the set of parties that participated in generating the signature. Moreover, the signature should reveal nothing about the total number of parties or the threshold.

We will define these concepts more precisely momentarily.

An accountable threshold signature scheme is frequently needed in financial applications. Consider again a 3-out-of-5 threshold setup, where a valid signature is needed to move funds from Alice's account to Bob's account. Now, suppose Alice complains that funds were fraudulently moved out of her account. The financial institution could then examine the signature that authorized the transfer, and identify the three signing parties that participated in the fraud. Those three parties will have some explaining to do. In general, accountable threshold signatures can be used to hold the signing parties accountable for their actions.

A private threshold signature scheme is needed when an organization prefers to keep its internal organizational structure private. For example, suppose a certificate authority (CA) splits its signing key across five servers so that any three can sign a certificate request. The CA prefers that the public not know the number of servers nor the threshold, lest that information help an attacker who is trying to compromise the CA.

We have already seen examples of both flavors of threshold signatures:

- The generic threshold signature scheme from Section 22.2.1 is accountable. This scheme is used in the Bitcoin system for precisely this reason. Bitcoin transactions that are signed this way are called `multisig` transactions.

- The BLS threshold signature scheme from Section 22.2.2.1 is private.

The trouble with the generic threshold construction is that signatures can be long — their length is proportional to the threshold. Later in this section we will construct an accountable threshold signature scheme with much shorter signatures: each signature is a single group element along with a compact description of the signing set, no matter the threshold or the number of signing parties.

First, we define privacy and accountability more precisely.

22.2.5.1 Accountable threshold signatures: definition

We begin by defining accountable threshold signatures, or ATS. An ATS scheme is a threshold signature scheme with an additional tracing algorithm T.

Definition 22.8. *An **accountable threshold signature scheme**, or **ATS**, is a tuple of five efficient algorithms (G, S, V, C, T) where (G, S, V, C) is a threshold signature scheme, and*

- *T is a deterministic **tracing algorithm** that is invoked as $\mathcal{J} \leftarrow T(pk, m, \sigma)$, where pk is a public key obtained by running $G(N, t)$. Algorithm T outputs a set $\mathcal{J} \subseteq \{1, \ldots, N\}$ of size t that indicates the set of parties that participated in generating σ.*

- ***Correctness:*** *for all possible outputs* $(pk, pkc, sk_1, \ldots, sk_N)$ *of* $G(N, t)$*, all messages* m*, and all* t*-size subsets* $\mathcal{J}$ *of* $\{1, \ldots, N\}$*, we have*

$$\Pr\Big[T\big(pk,\ m,\ C(pkc,\ m,\ \mathcal{J},\ \{S(sk_j, m)\}_{j\in\mathcal{J}})\big) = \mathcal{J}\Big] = 1.$$

ATS security. An ATS is secure if no set of parties can frame an innocent party. In particular, even if all but one of the parties collude, they cannot frame the one remaining party. We capture this in the following game.

Attack Game 22.3 (ATS security). For a given ATS scheme $\mathcal{S} = (G, S, V, C, T)$, and a given adversary $\mathcal{A}$, we define the following attack game.

- *Setup:* the adversary sends poly-bounded allowable parameters N and t, where $0 < t \leq N$, and a *target victim* $v \in \{1, \ldots, N\}$ to the challenger. The challenger runs

  $$(pk, pkc, sk_1, \ldots, sk_N) \xleftarrow{R} G(N, t),$$

  and sends pk, pkc, and $\{sk_\ell\}_{\ell \neq v}$ to the adversary.

- *Signing queries:* for $j = 1, 2, \ldots,$ signing query j from $\mathcal{A}$ is a message m_j. Given m_j, the challenger computes $\sigma'_j \xleftarrow{R} S(sk_v, m_j)$ and sends σ'_j to the adversary.

- *Forgery:* eventually $\mathcal{A}$ outputs a candidate forgery pair (m, σ).

We say that the adversary wins the game if the following conditions hold:

- $V(pk, m, \sigma) = \mathsf{accept}$,

- m is new, namely, $m \notin \{m_1, m_2, \ldots\}$, and

- $v \in T(pk, m, \sigma)$ meaning that T incorrectly says that v participated in generating σ.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$ denoted $\text{ATS}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 22.9 (secure ATS). *We say that an ATS scheme* $\mathcal{S}$ *is* ***secure*** *if for all efficient adversaries* $\mathcal{A}$*, the quantity* $\text{ATS}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

One might consider a seemingly stronger definition where the adversary is allowed to adaptively choose the target victim $v \in \{1, \ldots, N\}$ after seeing the public key, some secret keys, and some signatures. However, security under Definition 22.9, where the adversary has to commit to v ahead of time, implies security under a definition that allows the adversary to choose v adaptively. We leave this as a good exercise. The key observation is that an adversary $\mathcal{A}$ that chooses v adaptively can be converted into an adversary $\mathcal{B}$ that chooses v in advance by having $\mathcal{B}$ guess $v \in \{1, \ldots, N\}$ at the beginning of the game. This $\mathcal{B}$ has a $1/N$ chance of correctly guessing $\mathcal{A}$'s choice for the victim v to attack.

The generic threshold scheme is accountable. Let us briefly verify that the generic threshold signature scheme (G', S', V', C') from Section 22.2.1 is accountable by giving a tracing algorithm T' so that (G', S', V', C', T') is a secure ATS. Recall that the generic scheme is built from a regular (non-threshold) signature scheme (G, S, V). The public key is $pk := (pk_1, \ldots, pk_N)$, and a threshold signature on a message m is a collection $\sigma := (\mathcal{J}, \{\sigma'_j\}_{j\in\mathcal{J}})$ of t valid signatures by t of the public keys in pk.

The tracing algorithm $T'(pk, m, \sigma)$, where $\sigma = (\mathcal{J}, \{\sigma'_j\}_{j\in\mathcal{J}})$, simply outputs $\mathcal{J}$. It is a simple exercise to show that if (G, S, V) is a secure signature scheme, then (G', S', V', C', T') is a secure ATS.

Remark 22.6 (Trusted key generation). Our description of an ATS assumes that a trusted party, Tracy, runs algorithm $G(N, t)$ to generate the signing keys for the parties. Clearly Tracy can frame an innocent signing party since she has all the signing keys. This is clearly undesirable. Note that the generic ATS (G', S', V', C', T') described above can be securely implemented using decentralized key provisioning (see Remark 22.3), and as such are not susceptible to this type of attack. For other schemes, we may need to use a more expensive DKG protocol (such as in Section 22.4) to avoid this type of attack. Later in this section we will see an ATS based on the BLS signature scheme. This ATS can also be implemented using decentralized key provisioning. □

22.2.5.2 Private threshold signatures: definition

A private threshold signature scheme, or PTS, is the opposite of an ATS: a valid signature should reveal nothing about the set that generated it. Moreover, a signature should reveal nothing about the threshold or the total number of parties. To capture this property we require that signatures output by the combiner are indistinguishable from signatures generated by a regular (non-threshold) signature scheme, even if the adversary gets to see the public key and many signatures on messages of its choice. This is captured by the following attack game.

Attack Game 22.4 (PTS security). For a given threshold signature scheme $\mathcal{S}_0 = (G_0, S_0, V_0, C_0)$, and a regular (non-threshold) signature scheme $\mathcal{S}_1 = (G_1, S_1, V_1)$, we define two experiments.

Experiment b (for $b = 0, 1$):

- the adversary sends poly-bounded allowable parameters N and t, where $0 < t \le N$, to the challenger. The challenger computes

  $$(pk_0, pkc_0, sk'_1, \ldots, sk'_N) \xleftarrow{R} G_0(N, t) \qquad \text{and} \qquad (pk_1, sk_1) \xleftarrow{R} G_1()$$

  and sends pk_b to the adversary.

- The adversary submits a sequence of queries to the challenger. For $i = 1, 2, \ldots$, the ith query is a pair $(m_i, \mathcal{J}_i)$ where m_i is a message and $\mathcal{J}_i$ is a t-size subset of $\{1, \ldots, N\}$. In response, the challenger computes the signature σ_i as follows:

  - if $b = 0$, $\sigma_i \xleftarrow{R} C_0\big(pkc_0, m_i, \mathcal{J}_i, \big\{S_0(sk'_j, m_i)\big\}_{j\in\mathcal{J}_i}\big)$;
  - if $b = 1$, $\sigma_i \xleftarrow{R} S_1(sk_1, m_i)$.

  The challenger sends σ_i to the adversary.

- Finally, the adversary outputs a bit $\hat{b} \in \{0,1\}$.

Let W_b be the event that $\mathcal{A}$ outputs 1 in Experiment b. Define $\mathcal{A}$'s **advantage** with respect to $(\mathcal{S}_0, \mathcal{S}_1)$ as

$$\text{PTS}\mathsf{adv}[\mathcal{A}, \mathcal{S}_0, \mathcal{S}_1] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 22.10. *A threshold signature scheme $\mathcal{S}_0$ is* ***private****, or a* ***PTS****, if there exists a regular (non-threshold) signature scheme $\mathcal{S}_1$ such that for all efficient adversaries $\mathcal{A}$, the quantity* $\text{PTS}\mathsf{adv}[\mathcal{A}, \mathcal{S}_0, \mathcal{S}_1]$ *is negligible.*

The fact that an adversary cannot distinguish between a sequence of signatures generated by the threshold scheme, and a sequence of signatures generated by the regular (non-threshold) scheme, means that the threshold signatures and the public key reveal nothing about N, t, and $\mathcal{J}$.

Definition 22.10 captures privacy for N, t, and $\mathcal{J}$ from the public's point of view, where only pk and a sequence of signatures are visible. This is sufficient to ensure that a sequence of threshold signatures reveals nothing to the public about the inner workings of a signer such as a certificate authority. The definition, however, does not ensure privacy against an insider. For example, one of the N signers may be able to use its secret key to determine the set $\mathcal{J}$ that generated a particular signature. It is not difficult to define a game that prevents this. See Exercise 22.7.

The BLS threshold scheme is private. Let us briefly verify that the BLS threshold signature scheme (G', S', V', C') from Section 22.2.2.1 is private. Referring back to the notation in that section, recall that the key generation algorithm samples a random $\alpha \xleftarrow{R} \mathbb{Z}_q$, and then generates Shamir secret shares of this α. The public key is $pk := g^{\alpha}$. It is a simple exercise to show that for all messages m, all N, t, and all t-size subsets $\mathcal{J} \subseteq \{1, \ldots, N\}$, the resulting threshold signature is $\sigma := H(m)^{\alpha}$. Hence, the threshold signature is always the same as a regular (non-threshold) BLS signature with secret key α. It follows that no adversary can distinguish between a sequence of threshold BLS signatures and a sequence of regular (non-threshold) BLS signatures. Hence, the definition of privacy is satisfied.

22.2.6 BLS accountable threshold signatures

So far, our only example of an accountable threshold signature scheme, or ATS, is the generic scheme from Section 22.2.1. The trouble with that scheme is that it generates long signatures, where signature length is proportional to the threshold.

To construct a better ATS, with shorter signatures, we need not look far. In fact, the BLS aggregate signature scheme from Section 15.5.3.2 directly gives an ATS. Let's see how. As usual, let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$.

The BLS accountable threshold scheme $\mathcal{S}_{\text{bls-ats}} = (G, S, V, C, T)$ works as follows:

- $G(N, t) \to (pk, pkc, sk_1, \ldots, sk_N)$:
  - choose random $\alpha_1, \ldots, \alpha_N \xleftarrow{R} \mathbb{Z}_q$ and compute $sk_i \leftarrow \alpha_i$ and $pk_i \leftarrow g^{\alpha_i}$ for $i = 1, \ldots, N$;
  - set $pk := (pk_1, \ldots, pk_N)$ and $pkc := pk$;
  - output $(pk,\ pkc,\ sk_1, \ldots, sk_N)$.
- $S(sk_i, m) \to \sigma_i$, where $sk_i = \alpha_i$: output the signature share $\sigma_i \leftarrow H(m)^{\alpha_i} \in \mathbb{G}$.

- $C\big(pkc, m, \mathcal{J}, \{\sigma_j\}_{j\in\mathcal{J}}\big) \to \sigma$, where $\mathcal{J} \subseteq \{1,\ldots,N\}$ is of size t, and $pkc = (pk_1,\ldots,pk_N)$:
  - *step 1:* verify that for all $j \in \mathcal{J}$ the signature share σ_j is a valid BLS signature on m with respect to the public key pk_j; let $\mathcal{J}^*$ be the set of all $j \in \mathcal{J}$ for which this does not hold; if $\mathcal{J}^*$ is nonempty, output $\mathsf{blame}(\mathcal{J}^*)$ and abort;
  - *step 2:* compute the aggregate signature $\sigma \leftarrow \prod_{j\in\mathcal{J}} \sigma_j \in \mathbb{G}$ and output $(\mathcal{J},\sigma)$ as the signature on m.
- $V\big(pk, m, (\mathcal{J},\sigma)\big)$:
  - *step 1:* compute the aggregate public key $apk := \prod_{j\in\mathcal{J}} pk_j$;
  - *step 2:* Verify that $|\mathcal{J}| = t$ and that σ is a valid BLS signature on m with respect to the public key apk. If so output accept; otherwise output reject.
- $T\big(pk, m, (\mathcal{J},\sigma)\big)$: output $\mathcal{J}$.

A signature $(\mathcal{J},\sigma)$ is a single group elements in $\sigma \in \mathbb{G}$ along with a compact description of the signing set $\mathcal{J}$. This is much shorter than a signature produced by the generic scheme. We next prove that the scheme is a secure ATS as in Definition 22.9.

Theorem 22.6. *If the BLS signature scheme is secure in the group $\mathbb{G}$, then $\mathcal{S}_{\text{bls-ats}}$ is a secure ATS.*

Proof idea. Let $\mathcal{A}$ be an ATS adversary attacking $\mathcal{S}_{\text{bls-ats}}$. We build an adversary $\mathcal{B}$ that plays the role of challenger to $\mathcal{A}$ and attacks the BLS signature scheme. $\mathcal{B}$ begins by receiving a challenge public key pk' from its BLS challenger and receiving N, t, and v from $\mathcal{A}$. Our $\mathcal{B}$ now does:

- sample random $\alpha_1,\ldots,\alpha_N \xleftarrow{\text{R}} \mathbb{Z}_q$ and compute $pk_i \leftarrow g^{\alpha_i}$ for $i = 1,\ldots,N$;
- set $pk_v \leftarrow pk'$; (this embeds the challenge public key in position v)
- set $pk := (pk_1,\ldots,pk_N)$ and $pkc := pk$;
- send pk, pkc, and $\{sk_\ell\}_{\ell\neq v}$ to $\mathcal{A}$.

Next, $\mathcal{A}$ issues a sequence of signature queries for $pk_v = pk'$, which $\mathcal{B}$ answers by querying its BLS challenger. Eventually $\mathcal{A}$ outputs a forgery $(m, (\mathcal{J},\sigma))$, where $(\mathcal{J},\sigma)$ is a valid signature on m and where $v \in \mathcal{J}$. Because the signature is valid, we know that $\sigma = \prod_{j\in\mathcal{J}} H(m)^{\alpha_j}$. Moreover, since $v \in \mathcal{J}$, we can rewrite this as $H(m)^{\alpha_v} = \sigma / \prod_{j\in\mathcal{J}, j\neq v} H(m)^{\alpha_j}$. Our $\mathcal{B}$ can compute the right hand side by itself and obtain $\sigma' := H(m)^{\alpha_v}$. This σ' is the BLS signature on the message m under pk'. Finally, $\mathcal{B}$ sends (m,σ') to its challenger as a valid BLS forgery. □

Remark 22.7 (Trusted key generation). Since $\mathcal{S}_{\text{bls-ats}}$ assumes a trusted key generator, we do not have to defend against a *rogue key attack* as in Section 15.5.2.1. However, as discussed in Remark 22.6, it is desirable to eliminate this trusted key generator as a single point of failure. Just as for the generic ATS scheme, $\mathcal{S}_{\text{bls-ats}}$ can be implemented using decentralized key provisioning, but doing so exposes the scheme to a rogue public key attack. Therefore, when generating keys for $\mathcal{S}_{\text{bls-ats}}$ in this way, it is important to require that every party publish a proof of possession of the secret key, as discussed in Section 15.5.3.2, or alternatively to use message augmentation, as discussed in Section 15.5.3.1. A third option with some benefits is presented in [33]. □

This completes our discussion of threshold signature schemes.

22.3 Threshold decryption schemes

We now change topics and turn to the problem of designing threshold decryption schemes where the decryption key is shared among N parties so that t of them are needed to decrypt a ciphertext. We begin by defining the basic syntax of a threshold decryption scheme. Such a scheme will make use of the concept of **associated data**, as discussed in Section 12.7. Recall that associated data is public data that is supplied as an additional input to the encryption algorithm. In order to recover the encrypted plaintext, the identical associated data must be supplied during decryption.

Definition 22.11. *A **public-key threshold decryption scheme** $\mathcal{E} = (G, E, D, C)$ is a tuple of four efficient algorithms:*

- *G is a probabilistic **key generation algorithm** that is invoked as*

$$(pk, pkc, sk_1, \ldots, sk_N) \xleftarrow{\text{R}} G(N, t)$$

  *to generate a t-out-of-N shared key. It outputs a **public key** pk, a **combiner public key** pkc, and N **decryption key shares** $sk_1, \ldots, sk_N$.*

- *E is a probabilistic **encryption algorithm** that is invoked as $c \xleftarrow{\text{R}} E(pk, m, d)$, where pk is a public key output by G, m is a message, and d is associated data.*

- *D is a deterministic **decryption algorithm** that is invoked as $c'_i \leftarrow D(sk_i, c, d)$, where sk_i is one of the decryption key shares output by G, c is a ciphertext, d is associated data, and c'_i is a decryption share for c using sk_i.*

- *C is a deterministic **combiner algorithm** that is invoked as $m \leftarrow C(pkc, c, d, \mathcal{J}, \{c'_j\}_{j \in \mathcal{J}})$, where pkc is the combiner public key, c is a ciphertext, d is associated data, $\mathcal{J}$ is a subset of $\{1, \ldots, N\}$ of size t, and each c'_j is a decryption share of c. The algorithm either outputs a plaintext m, the special symbol reject, or a special message $\mathsf{blame}(\mathcal{J}^*)$, where $\mathcal{J}^*$ is a nonempty subset of $\mathcal{J}$.*

  Intuitively, the message $\mathsf{blame}(\mathcal{J}^*)$ indicates that the provided decryption shares c'_j for $j \in \mathcal{J}^*$ are invalid.

- ***Correctness:** as usual, decryption should correctly decrypt a properly constructed ciphertext; specifically, for all possible outputs $(pk, pkc, sk_1, \ldots, sk_N)$ of $G(N, t)$, all messages m, all associated data d, all possible outputs c of $E(pk, c, d)$, and all t-size subsets $\mathcal{J}$ of $\{1, \ldots, N\}$, we have*

$$C\big(pkc,\ c,\ \mathcal{J},\ \{D(sk_j, c, d)\}_{j \in \mathcal{J}}\big) = m.$$

In the above definition, messages lie in some finite message space $\mathcal{M}$, ciphertexts in some finite ciphertext space $\mathcal{C}$, and associated data in some finite space $\mathcal{D}$. We say that $\mathcal{E}$ is defined over $(\mathcal{M}, \mathcal{D}, \mathcal{C})$. Also, just as for a secret sharing scheme, a threshold decryption scheme may impose constraints on the parameters N and t (such as an upper bound on N and on the relationship between N and t). For any particular scheme, we will refer to **allowable parameters** as those (N, t) pairs that are allowed by the scheme.

Remark 22.8. Definition 22.11 requires that t and N be specified at key generation time. However, all the schemes in this section can be extended so that both t and N can be changed after the secret key shares are generated, without changing the public key pk. □

Note that Definition 22.11 does not require that the sk_i's are actually shares of any particular decryption key sk for a non-threshold scheme, but for many threshold decryption schemes, this is indeed the case.

Let's map these four algorithms back to Fig. 22.1a. Algorithm $G(N, t)$ is used to provision the decryption servers with their decryption key shares $sk_1, \ldots, sk_N$. Next, when the combiner requests a decryption of some ciphertext c, along with associated data d, each decryption server examines the ciphertext and associated data, and it decides whether or not it wants to perform the decryption. If server i is willing to perform the decryption, it computes $c'_i \xleftarrow{R} D(sk_i, c, d)$ and sends the resulting decryption share c'_i to the combiner. Once the combiner receives t decryption shares from distinct signing servers, it runs algorithm C to (hopefully) obtain either a plaintext m or a set of indices $\mathcal{J}^*$ that identifies the misbehaving servers that contributed invalid shares.

Just as we did for threshold signature schemes, we will formally define an appropriate **robustness** property for threshold decryption schemes. Intuitively, this property says two things: first, if the combiner algorithm outputs $\mathsf{blame}(\mathcal{J}^*)$, then the set of indices $\mathcal{J}^*$ identifies *only* misbehaving servers; second, if the combiner outputs $m \in \mathcal{M} \cup \{\mathsf{reject}\}$, then the value of m is independent of *which* servers contributed decryption shares. If a scheme is robust in this sense, then if the t shares collected by the combiner do not yield $m \in \mathcal{M} \cup \{\mathsf{reject}\}$, the set $\mathcal{J}^*$ may be used to discard invalid shares from misbehaving servers, and the combiner may then seek out valid shares from other servers. This process may continue until some $m \in \mathcal{M} \cup \{\mathsf{reject}\}$ is obtained, provided there are at least t decryption servers that are available and behaving correctly. Moreover, it is guaranteed that this value of m is independent of which t servers ultimately contributed valid decryption shares.

After we look at some constructions, we will formally define what it means for a threshold decryption scheme to be secure. However, let us give an informal definition of security here. We will assume that an adversary has compromised $t-1$ decryption servers, obtaining their decryption key shares. During the attack, the adversary may ask any of the uncompromised servers for decryption shares on any ciphertext/associated data pairs of its choice. Security will mean that the adversary will not glean any useful information about any message m that was encrypted using associated data d to yield a ciphertext c, so long as it does not ask any of the uncompromised servers for decryption shares on a ciphertext/associated data pair $(\hat{c}, \hat{d})$ where $\hat{d} = d$.

This security definition corresponds to the nontion of AD-only CCA security for non-threshold schemes (as defined in Section 12.7.1), and it rules out a broad range of attacks. First, it rules out a key-recovery attack: if an attacker steals the decryption key shares from at most $t-1$ servers, the attacker cannot reconstruct the decryption key itself. Second, it also rules out attacks in which an attacker corrupts the behavior of some (but not too many) of the decryption servers, even if it does not obtain their decryption key shares. Indeed, in any application, each decryption server will be implementing some particular policy that determines which ciphertexts should be decrypted under what circumstances, based on the associated data and other data presented to the server, and will only issue a decryption shares if this policy is satisfied. Suppose a message m is encrypted using associated data d, resulting in a ciphertext c. Security means that if at most $t-1$ servers are corrupted, in the sense that either (a) their decryption key shares have been stolen or (b) they do not implement the decryption policy correctly, then an attacker still cannot get any information about m unless at least one of the uncompromised servers determines that the policy is satisfied and issues a decryption share for a ciphertext/associated data pair $(\hat{c}, \hat{d})$ where $\hat{d} = d$. We should note, conversely, that if an attacker can get $t-1$ compromised servers to issue decryption shares for the ciphertext/associated data pair (c, d) (either by stealing their decryption key shares or by subverting their decryption policy), then the attacker needs to get just one of the uncompromised servers to issue a decryption share for (c, d) in order to obtain m.

As a more concrete example, consider the key-escrow application discussed in Sections 12.2.3 and 12.7. In that application, a user's file encryption key was encrypted under the public key of an escrow service. Moreover, some metadata md associated with the file was used as associated data in the computation of the ciphertext c_{ES}. At a later time, some requesting entity may present the pair (c_{ES}, md) to the escrow service to obtain the file encryption key. The escrow service may impose some type of access control policy, based on the given metadata, along with the identity or credentials of the requesting entity. Enforcing this access control policy is just as important as securely storing the decryption key. To improve the security of the escrow service, we may implement it as a distributed system using a threshold decryption scheme, where each decryption server stores one decryption key share and independently enforces the access control policy. Security means that if at most $t - 1$ servers are corrupted, in the sense that either (a) their decryption key shares have been stolen or (b) they do not correctly enforce the access control policy, then an attacker still cannot get any information about the escrowed file encryption key unless at least one of the uncompromised servers determines that the access control policy is satisfied and issues a decryption share for $(\hat{c}_{\mathrm{ES}}, md)$ for some ciphertext $\hat{c}_{\mathrm{ES}}$.

Note that one can naturally define a stronger notion of security, corresponding to full CCA security, rather than AD-only CCA security. Here, security means that the adversary will not glean any useful information about any message m that was encrypted using associated data d to yield a ciphertext c, so long as it does not ask any of the uncompromised servers for decryption shares on the ciphertext/associated data pair (c, d). This is a stronger notion of security, as the adversary is less restricted in its attack. We focus on the weaker notion of AD-only CCA security for three reasons:

- for most applications of threshold decryption, AD-only CCA security is sufficient,
- AD-only CCA-secure schemes are somewhat less complicated and more efficient than fully CCA-secure schemes, and
- any AD-only CCA-secure threshold decryption scheme can be easily converted into a fully CCA-secure scheme using the simple technique described in Exercise 14.13 of "wrapping" the ciphertext with a strongly one-time secure signature (see also Remark 22.10 and Exercise 22.11).

22.3.1 A generic threshold decryption scheme

Any AD-only CCA-secure public key encryption scheme (G, E, D) can be transformed into an AD-only CCA-secure threshold decryption scheme (G', E', D', C') by applying secret sharing to the plaintext.

For parameters $0 < t \leq N$, the t-out-of-N threshold decryption scheme works as follows:

- During setup, the key generator creates N key pairs $(pk_i, sk_i) \xleftarrow{\mathrm{R}} G()$, for $i = 1, \ldots, N$, and gives key sk_i to signing server i. The public encryption key is $pk := (pk_1, \ldots, pk_N)$, and the combiner public key is the same, $pkc := pk$. More precisely, we define threshold key generation $G'(N, t)$ as an algorithm that outputs $(pk, pk, sk_1, \ldots, sk_N)$.
- Algorithm $E'(pk, m, d)$: for a public key $pk = (pk_1, \ldots, pk_N)$, we encrypt a message $m \in \mathbb{Z}_q$ for pk using associated data d as follows:

- run a t-out-of-N secret sharing on m to obtain N shares $m_1, \dots, m_N$,
  - for $i = 1, \dots, N$ compute $c_i \xleftarrow{R} E(pk_i, m_i, d)$,
  - output $\boldsymbol{c} := (c_1, \dots, c_N)$.

- Algorithm $D'(sk_i, \boldsymbol{c}, d)$: output $c_i' := D(sk_i, c_i, d)$.

- The combiner collects t decryption shares and combines them using the combining algorithm of the secret sharing scheme to recover the plaintext m.

As we will see, this construction satisfies the security requirements for a threshold decryption scheme. However, one major drawback with this scheme is that its performance degrades as N grows. In particular, the size of the ciphertext size grows linearly with N, as does the time to encrypt a message. Our goal is to construct a threshold decryption scheme where ciphertext size and encryption time are *independent* of the parameters N and t.

Another major drawback of this scheme is that it does not provide robustness.

- As one example of what could go wrong, the scheme does not give the combiner any way of detecting invalid decryption shares — a misbehaving decryption server could give the combiner an incorrect share of the plaintext, causing the combiner to output the wrong plaintext.

- As another example of what could go wrong, even if all of the decryption servers behave correctly, a party could generate a ciphertext $(c_1, \dots, c_N)$ where $c_1, \dots, c_N$ do not encrypt consistent shares of any one plaintext, and the combiner could end up outputting different plaintexts depending on which subset of servers it obtained decryption shares from.

There are various techniques that could be employed to make this scheme robust, but we will not discuss them here.

Remark 22.9 (Decentralized key provisioning). Just as we observed in Remark 22.3 for the generic threshold signing scheme, the generic threshold decryption scheme can be securely implemented using decentralized key provisioning. Thus, for this scheme, do we not need a trusted key generator, and thus can eliminate that single point of failure. □

22.3.2 An insecure threshold decryption scheme

Recall that the ElGamal encryption scheme $\mathcal{E}_{\mathrm{EG}}$, introduced in Section 11.5, uses a group $\mathbb{G}$ of prime order q with generator $g \in \mathbb{G}$, a symmetric cipher $\mathcal{E}_{\mathrm{s}} = (E_{\mathrm{s}}, D_{\mathrm{s}})$, defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, and a hash function $H : \mathbb{G}^2 \to \mathcal{K}$. The secret key is an element $\alpha \in \mathbb{Z}_q$, the public key is $u := g^\alpha \in \mathbb{G}$. To encrypt a plaintext $m \in \mathcal{M}$, we first compute $\beta \xleftarrow{R} \mathbb{Z}_q$, $v \leftarrow g^\beta$, $w \leftarrow u^\beta$, and then $k \leftarrow H(v, w)$ and $e \xleftarrow{R} E_{\mathrm{s}}(k, m)$; the ciphertext is $(v, e) \in \mathbb{G} \times \mathcal{C}$. Such a ciphertext is decrypted by first computing $w \leftarrow v^\alpha$, and then $k \leftarrow H(v, w)$ and $m \leftarrow D_{\mathrm{s}}(k, e)$.

In Section 11.5 we showed that if we model H as a random oracle, then this scheme is semantically secure under the CDH assumption for $\mathbb{G}$. In Section 12.4 we showed that if we model H as a random oracle, then this scheme is CCA secure under the stronger ICDH assumption for $\mathbb{G}$. As discussed in Exercise 12.17, we can add support for associated data by passing associated data $d \in \mathcal{D}$ as an additional input to the hash function H to derive the symmetric key in both the encryption and decryption algorithms as $k \leftarrow H(v, w, d)$.

To convert this into a t-out-of-N decryption scheme, we might be tempted to proceed as follows. The key generation algorithm first generates a t-out-of-N Shamir secret sharing of the ElGamal decryption key $\alpha \in \mathbb{Z}_q$. The resulting shares α_i for $i = 1, \ldots, N$ are the decryption key shares, and each decryption server is given one share. Given a ciphertext (v, e), the ith decryption server uses its decryption key share to generate the decryption share $w_i := v^{\alpha_i}$.

Recall that for some polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t - 1$, we have $\boldsymbol{\omega}(0) = \alpha$ and $\boldsymbol{\omega}(i) = \alpha_i$ for $i = 1, \ldots, N$. The design of the combiner algorithm is based on this fact. Given a ciphertext (v, e), associated data d, and a collection of t decryption shares $\{w_j\}_{j \in \mathcal{J}}$, the combiner works as follows:

- First, the combiner performs "interpolation in the exponent", by applying Corollary 22.2 with the given set $\mathcal{J}$ and $j^* := 0$ to compute the group element $w := v^\alpha = v^{\boldsymbol{\omega}(0)} \in \mathbb{G}$. More explicitly, w is computed as
$$w \leftarrow \prod_{j \in \mathcal{J}} w_j^{\lambda_j} \quad \in \mathbb{G}$$
where the exponents $\{\lambda_j\}_{j \in \mathcal{J}}$ are the Lagrange interpolation coefficients, which depend only on the set $\mathcal{J}$, as computed in Lemma 22.1.
- Once the combiner has computed w, it then computes the symmetric secret key $k \leftarrow H(v, w, d)$ and outputs plaintext $m \leftarrow D_{\mathrm{s}}(k, e)$.

This threshold decryption scheme is neither robust nor CCA secure. While the scheme could be augmented without too much difficulty to make it robust, it is inherently insecure against CCA attacks.

To see why this scheme is inherently insecure against CCA attacks, observe that a decryption server does not use the associated data d in generating its decryption share. Recall that in applications, a decryption server should be able to enforce some kind of decryption policy based on the associated data, but there is no way to do that here.

As a more concrete example, suppose Alice encrypts a message m with associated data d, generating a ciphertext (v, e). Perhaps d says that her friend Bob should have the right to decrypt this ciphertext. However, the ciphertext (v, e) falls into the hands of Carol, who presents to the decryption servers the ciphertext (v, e), along with associated data d' that says Carol has the right to decrypt this ciphertext, as well as any credentials that are required by the decryption servers to validate Carol's identity. Now the decryption servers will send Carol the decryption shares that she needs to decrypt the ciphertext (v, e).

Without making much stronger assumptions, this scheme does not even satisfy a weaker notion of CCA security, analogous to that considered in Exercise 12.28, in which the attacker has unfettered access to the decryption servers *before* getting its hands on the ciphertext it really wants to decrypt. Indeed, such an attacker may submit an arbitrary ciphertext $(\hat{v}, \hat{e})$ to the decryption servers, and from the resulting decryption shares, it can reconstruct the group element $\hat{v}^\alpha$. Thus, access to the decryption servers essentially gives the adversary an oracle for computing $\hat{v}^\alpha$ for arbitrary group elements $\hat{v}$. This implies security in this weaker sense depends on an assumption at least as strong as the *static DH assumption*, discussed in Section 16.1.4. The precise attack game is the same as Attack Game 16.1, except that after making a sequence of SDH queries, the challenger presents the adversary with a random challenge $v^* \in \mathbb{G}$ and the adversary wins the game if it can compute

$(v^*)^\alpha$. As we saw in Section 16.1.4, in some settings, there are attacks on static DH that are much more efficient than CDH or ICDH.

While this threshold decryption scheme is not CCA secure, it does in fact satisfy a notion of CPA security, which is explored in Exercise 22.8.

22.3.3 A secure threshold decryption scheme based on CDH

It is not too difficult modify the insecure threshold scheme discussed in Section 22.3.2 to obtain a secure threshold scheme. The conceptually easiest way to do this makes use of an efficient algorithm $\mathcal{O}_{\text{DDH}}$ that solves the decision Diffie-Hellman (DDH) problem in the group $\mathbb{G}$, we we did for BLS signatures in Section 22.2.2.

We shall present here a threshold decryption scheme that makes use of such an algorithm $\mathcal{O}_{\text{DDH}}$. This can be implemented using a pairing, and one can also devise a variation of the scheme that uses an asymmtric pairing. However, unlike the sitiation with BLS, it turns out that we do not really need to use a pairing at all — we can instead use a non-interactive proof system for proving that a given triple is a DH-triple. We shall describe this in Section 22.3.6.

22.3.3.1 The GS encryption scheme

We begin by presenting an ordinary (non-threshold) encryption scheme, which we call the GS encryption scheme, also denoted $\mathcal{E}_{\text{GS}}$.

This scheme has a message space $\mathcal{M}$ and associated data space $\mathcal{D}$, and makes use of the following components:

- a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$; we shall assume that the CDH assumption holds in $\mathbb{G}$;
- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C}_{\text{s}})$; we shall assume that $\mathcal{E}_{\text{s}}$ is semantically secure;
- a hash function $H_{\mathcal{K}} : \mathbb{G} \times \mathbb{G} \to \mathcal{K}$; we shall model $H_{\mathcal{K}}$ as a random oracle;
- a hash function $H_{\mathbb{G}} : \mathbb{G} \times \mathcal{D} \to \mathbb{G}$; we shall model $H_{\mathbb{G}}$ as a random oracle;
- an efficient algorithm $\mathcal{O}_{\text{DDH}}$ that on input $(g^\alpha, g^\beta, g^\gamma)$ that outputs accept if $\gamma = \alpha\beta$, and otherwise outputs reject.

Key generation. The key generation algorithm computes

$$\alpha \in \mathbb{Z}_q, \;\; u \leftarrow g^\alpha,$$

and outputs the public key u and the secret key α.

Encryption. To encrypt a message $m \in \mathcal{M}$ with associated data $d \in \mathcal{D}$ under the public key $u \in \mathbb{G}$, the encryption algorithm computes

$$\begin{aligned}
&\beta \xleftarrow{\text{R}} \mathbb{Z}_q, \;\; v \leftarrow g^\beta, \;\; w \leftarrow u^\beta, \;\; k \leftarrow H(v, w), \;\; e \xleftarrow{\text{R}} E_{\text{s}}(k, m), \\
&\underline{u} \leftarrow H_{\mathbb{G}}(v, d), \;\; \underline{w} \leftarrow \underline{u}^\beta, \\
&\text{output } c := (v, e, \underline{w}).
\end{aligned}$$

Decryption. To decrypt a ciphertext $c = (v, e, \underline{w})$ with associated data d under the secret key α, the decryption first checks that

$$\mathcal{O}_{\text{DDH}}(\underline{u}, v, \underline{w}) = \mathsf{accept}, \quad \text{where} \quad \underline{u} := H_{\mathbb{G}}(v, d). \tag{22.1}$$

If this check passes, we say c **is valid with respect to** d, and the decryption algorithm computes

$$w \leftarrow v^{\alpha}, \;\; k \leftarrow H(v, w), \;\; m \leftarrow D_{\text{s}}(k, e)$$

and outputs m. If this check does not pass, the decryption algorithm outputs reject.

Discussion. This scheme is basically the same as the normal ElGamal encryption scheme, except that the encryption algorithm essentially proves that it knows β by raising a challenge group element $\underline{u} := H_{\mathbb{G}}(v, d)$ to the power β. The decryption algorithm verifies that this was done by performing the validity check in (22.1).

The scheme $\mathcal{E}_{\text{GS}}$ was in fact presented in Exercise 12.20, where you were asked to show that it AD-only CCA-secure under the assumptions listed above. In fact, in that exercise, you were asked to prove security against a stronger form of CCA attack in which when an adversary submits a decryption query $(\hat{c}, \hat{d})$, where $\hat{c} = (\hat{v}, \hat{e}, \hat{\underline{w}})$ is valid with respect to $\hat{d}$, the challenger returns the group element $\hat{w} := \hat{u}^{\alpha}$. This stronger security property is crucial to converting this scheme to a threshold scheme. We shall refer to this security property it as **AD-only strong-CCA security**.

22.3.3.2 Converting the GS scheme into a threshold decryption scheme

To make the GS scheme support threshold decryption, we will apply Shamir's secret sharing scheme $(G_{\text{sh}}, C_{\text{sh}})$ to the secret key $\alpha \in \mathbb{Z}_q$. We will show how to decrypt a ciphertext without ever reconstituting the value α at a single location.

The scheme. We now present the threshold GS decryption scheme, also denoted $\mathcal{E}_{\text{thGS}} = (G, E, D, C)$.

- $G(N, t) \to (pk, pkc, sk_1, \ldots, sk_N)$:
  - compute
$$\alpha \xleftarrow{\text{R}} \mathbb{Z}_q, \;\; u \leftarrow g^{\alpha},$$
    and set $pk := u$;
  - run $G_{\text{sh}}(N, t, \alpha)$ to obtain N shares $\alpha_1, \ldots, \alpha_N$ in $\mathbb{Z}_q$;
    Note: recall that for some polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$, we have $\boldsymbol{\omega}(0) = \alpha$ and $\boldsymbol{\omega}(i) = \alpha_i$ for $i = 1, \ldots, N$;
  - for $i = 1, \ldots, N$, compute $u_i \leftarrow g^{\alpha_i}$;
  - for $i = 1, \ldots, N$, set $sk_i := \alpha_i$;
  - set $pkc := (u_1, \ldots, u_N)$;
  - output $(pk, \; pkc, \; sk_1, \ldots, sk_N)$.
- $E(pk, m, d) \to c$, where $pk = u$:

- set $c := (v, e, \underline{w})$, where

$$\beta \xleftarrow{R} \mathbb{Z}_q, \;\; v \leftarrow g^\beta, \;\; w \leftarrow u^\beta, \;\; k \leftarrow H(v, w), \;\; e \xleftarrow{R} E_s(k, m),$$
$$\underline{u} \leftarrow H_{\mathbb{G}}(v, d), \;\; \underline{w} \leftarrow \underline{u}^\beta.$$

Note: this encryption algorithm is *identical* to that in $\mathcal{E}_{\text{GS}}$.

- $D(sk_i, c, d) \to c'_i$, where $sk_i = \alpha_i$ and $c = (v, e, \underline{w})$: check if c is valid with respect to d, as in (22.1); if not, output $c'_i :=$ reject; otherwise, compute $w_i \leftarrow v^{\alpha_i}$ and output $c'_i := w_i \in \mathbb{G}$.

- $C(pkc, c, d, \mathcal{J}, \{c'_j\}_{j \in \mathcal{J}}) \to m$, where $pkc = (u_1, \ldots, u_N)$, $c = (v, e, \underline{w})$, $\mathcal{J}$ is a subset of $\{1, \ldots, N\}$ of size t, and where $c'_j = w_j \in \mathbb{G}$ or $c'_j =$ reject for $j \in \mathcal{J}$:
  - *Step 1:* validate the ciphertext and the decryption shares:
    - ∗ check that c is valid with respect to d, as in (22.1): if not, output reject and halt;
    - ∗ check if for some $j \in \mathcal{J}$, we have either $c'_j =$ reject or

$$c'_j = w_j \quad \text{and} \quad \mathcal{O}_{\text{DDH}}(u_j, v, w_j) = \text{reject};$$

    if so, output blame$(\mathcal{J}^*)$ and halt, where $\mathcal{J}^*$ is the set of all such $j \in \mathcal{J}$;

    Note: assuming all the shares are valid, then for $j \in \mathcal{J}$, we have $c'_j = w_j = v^{\alpha_j} = v^{\boldsymbol{\omega}(j)}$, where $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ is the polynomial of degree at most $t-1$ used to share the secret key α;
  - *Step 2:* compute the ephemeral key w: "interpolate in the exponent" by applying Corollary 22.2 with the given set $\mathcal{J}$ and $j^* := 0$ to compute the ephemeral key $w := v^\alpha = v^{\boldsymbol{\omega}(0)}$; more explicitly, w is computed as

$$w \leftarrow \prod_{j \in \mathcal{J}} w_j^{\lambda_j} \quad \in \mathbb{G}$$

    where the exponents $\{\lambda_j\}_{j \in \mathcal{J}}$ are the Lagrange interpolation coefficients, which depend only on the set $\mathcal{J}$, as computed in Lemma 22.1;
  - *Step 3:* compute $k \leftarrow H_{\mathcal{K}}(v, w)$ and output $m \leftarrow D_s(p, e)$.

22.3.4 Threshold decryption security

Before analyzing the GS threshold decryption scheme (and its variants), we define what it means for a t-out-of-N threshold decryption scheme to be CCA secure. The security model gives the adversary two capabilities:

- We allow the adversary to completely control up to $t-1$ of the decryption servers. The adversary must declare at the beginning of the game a set $\mathcal{L}$ of up to $t-1$ servers that it wants to control.[2]

[2] Just as for threshold signatures, this is a *static corruption model*.

- In addition, the combiner is not a trusted party, and may be controlled by the adversary. This means that a scheme that temporarily reconstitutes the decryption key at the combiner would be insecure. To capture the idea that the combiner may be controlled by the adversary, we allow the adversary to request decryptions on ciphertexts of its choice, and we then give the adversary all N decryption shares generated by the decryption servers in response to these requests.

Even with control of the combiner, and up to $t-1$ servers, the adversary should not be able to break the AD-only CCA security of the scheme. Formally, we define security using the following attack game.

Attack Game 22.5 (threshold AD-only CCA security). For a public-key threshold decryption scheme $\mathcal{E} = (G, E, D, C)$ defined over $(\mathcal{M}, \mathcal{D}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b **(**$b = 0, 1$**):**

- *Setup:* the adversary sends poly-bounded allowable parameters N and t, where $0 < t \leq N$, and a subset $\mathcal{L} \subseteq \{1, \ldots, N\}$ of size $t-1$ to the challenger. The challenger runs

$$(pk, pkc, sk_1, \ldots, sk_N) \overset{\text{R}}{\leftarrow} G(N, t),$$

  and sends pk, pkc, and $\{sk_\ell\}_{\ell \in \mathcal{L}}$ to $\mathcal{A}$.

- $\mathcal{A}$ then makes a series of queries to the challenger. Each query can be one of two types:
  - *Encryption query:* for $i = 1, 2, \ldots$, the ith encryption query consists of a triple $(m_{i0}, m_{i1}, d_i) \in \mathcal{M}^2 \times \mathcal{D}$, where the messages m_{i0}, m_{i1} are of the same length. The challenger computes $c_i \overset{\text{R}}{\leftarrow} E(pk, m_{ib}, d_i)$ and sends c_i to $\mathcal{A}$.
  - *Decryption query:* for $j = 1, 2, \ldots$, the jth decryption query consists of a pair $(\hat{c}_j, \hat{d}_j) \in \mathcal{C} \times \mathcal{D}$ such that $\hat{d}_j$ is not among the associated data values $d_1, d_2, \ldots$ submitted in previous encryption queries. The challenger computes the N decryption shares $c'_{j,i} \leftarrow D(sk_i, \hat{c}_j, \hat{d}_j)$, for $i = 1, \ldots, N$, and sends these to $\mathcal{A}$.

- At the end of the game, $\mathcal{A}$ outputs a bit $\hat{b} \in \{0, 1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{thCCA}_{\text{ado}}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Definition 22.12 (threshold AD-only CCA security). *A public-key threshold decryption scheme $\mathcal{E}$ is* ***AD-only CCA secure*** *if for all efficient adversaries $\mathcal{A}$, the value* $\text{thCCA}_{\text{ado}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Remark 22.10 (from AD-only to full CCA security). As already discussed at the beginning of Section 22.3, AD-only CCA security is sufficient for many applications of threshold decryption. However, one can also define a notion of (full) CCA security for threshold decryption schemes. The only difference is that in the attack game, when submitting a decryption query $(\hat{c}_j, \hat{d}_j)$, the restriction is that the pair $(\hat{c}_j, \hat{d}_j)$ must not appear among the ciphertext/associated data pairs

$(c_1, d_1), (c_2, d_2), \ldots$ corresponding to previous encryption queries. Any AD-only CCA-secure threshold decryption scheme can be easily converted into a fully CCA-secure scheme using the simple technique described in Exercise 14.13 of "wrapping" the ciphertext with a strongly one-time secure signature. You are asked to work out the details of this in Exercise 22.11. □

22.3.4.1 Robust threshold decryption

Beyond security for the scheme, a threshold decryption scheme should also be robust against a malicious decryption server that is trying to disrupt the decryption process. We define two such robustness properties that any practical threshold decryption scheme should provide.

Suppose a combiner receives a collection $\{c'_j\}_{j \in \mathcal{J}}$ of t decryption shares on a given ciphertext/associated data pair (c, d). By Definition 22.11, the combiner will output either a plaintext $m \in \mathcal{M} \cup \{\mathsf{reject}\}$, or a special message $\mathsf{blame}(\mathcal{J}^*)$, where $\mathcal{J}^*$ is a nonempty subset of $\mathcal{J}$.

- If the output of the combiner is $\mathsf{blame}(\mathcal{J}^*)$, then we would like it to be the case that all of the decryption shares c'_j for $j \in \mathcal{J}^*$ are "bad", in the sense that they were incorrectly generated by misbehaving decryption servers. A threshold signature scheme that guarantees that this always happens is said to provide **accurate blaming**.

- Otherwise, if the output of the combiner is a plaintext m, we say that the collection of shares $\{c'_j\}_{j \in \mathcal{J}}$ is **valid** for (c, d). We say that the threshold decryption scheme is **consistent** if it is infeasible to come up with any other valid collection of decryption shares that combines to yield a plaintext $m' \neq m$.

  For many threshold decryption schemes, including the robust GS scheme, shares are individually validated, but there is some non-zero, but negligible, probability that an incorrectly generated share passes the validity test, and so the consistency property is violated with negligible probability.

A threshold decryption scheme is **robust** if it satisfies both properties.

In practice, robustness allows a combiner who is trying to decrypt a ciphertext to proceed as follows. If the combiner algorithm outputs $\mathsf{blame}(\mathcal{J}^*)$, then the combiner can discard the "bad" shares in $\mathcal{J}^*$, and seek out $t - |\mathcal{J}^*|$ "good" shares from among the remaining decryption servers. As long as there are t correctly behaving servers available, the accurate blaming property guarantees that the combiner can repeat this process until it gets a valid collection of shares, and the consistency property guarantees that when this happens, the resulting plaintext is the same that it would get from any other valid collection of decryption shares.

Also note that the consistency property, together with the correctness property in Definition 22.11, guarantees that if a ciphertext is a correctly generated encryption of a message, then any valid collection of decryption shares when combined will yield the original message.

We now define these two properties formally. The *accurate blaming* property for threshold decryption is almost identical to the corresponding property for threshold signatures.

Definition 22.13 (accurate blaming). *We say that a threshold decryption scheme $\mathcal{E} = (G, E, D, C)$ provides* ***accurate blaming*** *if the following holds:*

> *for all possible outputs $(pk, pkc, sk_1, \ldots, sk_N)$ of $G(N, t)$, all ciphertext/associated data pairs (c, d), all t-size subsets $\mathcal{J}$ of $\{1, \ldots, N\}$, and all collections of decryption shares*

$\{c'_j\}_{j\in\mathcal{J}}$:

$$C(pkc, c, d, \mathcal{J}, \{c'_j\}_{j\in\mathcal{J}}) = \mathsf{blame}(\mathcal{J}^*) \implies \Pr\big[D(sk_j, c, d) = c'_j\big] = 0 \textit{ for all } j \in \mathcal{J}^*.$$

The *consistency* property is defined using an attack game.

Attack Game 22.6 (consistent threshold decryption). For a given threshold decryption scheme $\mathcal{E} = (G, E, D, C)$, defined over $(\mathcal{M}, \mathcal{D}, \mathcal{C})$ and a given adversary $\mathcal{A}$, we define the following attack game.

- The adversary sends to the challenger poly-bounded allowable parameters N and t, where $0 < t \leq N$.
- The challenger runs $(pk, pkc, sk_1, \ldots, sk_N) \xleftarrow{R} G(N, t)$ and sends all this data to the adversary.
- The adversary outputs
$$(c, d, \mathcal{J}_1, \{c'_{1j}\}_{j\in\mathcal{J}_1}, \mathcal{J}_2, \{c'_{2j}\}_{j\in\mathcal{J}_2}),$$
where $c \in \mathcal{C}$, $d \in \mathcal{D}$, $\mathcal{J}_1$ and $\mathcal{J}_2$ are subsets of $\{1, \ldots, N\}$ of size t, and $\{c'_{1j}\}_{j\in\mathcal{J}_1}$ and $\{c'_{2j}\}_{j\in\mathcal{J}_2}$ are collections of decryption shares.
- We say the adversary wins the game if
  - $C(pkc, c, d, \mathcal{J}_1, \{c'_{1j}\}_{j\in\mathcal{J}_1}) = m_1 \in \mathcal{M} \cup \{\mathsf{reject}\}$,
  - $C(pkc, c, d, \mathcal{J}_2, \{c'_{2j}\}_{j\in\mathcal{J}_2}) = m_2 \in \mathcal{M} \cup \{\mathsf{reject}\}$, and
  - $m_1 \neq m_2$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$, denoted conPKE$\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 22.14 (consistent threshold decryption). *We say that a threshold decryption scheme $\mathcal{E}$ is* ***consistent*** *if for all efficient adversaries $\mathcal{A}$, the quantity* conPKE$\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

To exercise the definition, the reader should show that the generic threshold decryption scheme from Section 22.3.1 is not consistent.

Definition 22.15 (robustness). *We say that a threshold decryption scheme is* ***robust*** *if it* provides accurate blaming *and is* consistent.

22.3.5 Security of threshold GS

Next, we argue that the GS threshold decryption scheme is AD-only CCA secure. Security follows directly from the AD-only strong-CCA security of the non-threshold GS scheme, which you were asked to establish in Exercise 12.20. Let us denote by sCCA$_{\text{ado}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{GS}}]$ the advantage of an adversary $\mathcal{A}$ in attacking $\mathcal{E}_{\text{GS}}$ in the strengthened AD-only CCA attack game Section 22.3.3.1, in which a decryption query yields the group element from which the symmetric decryption key is derived, rather than just a plaintext.

Theorem 22.7. *If $\mathcal{E}_{\text{GS}}$ is AD-only strong-CCA secure, then $\mathcal{E}_{\text{thGS}}$ is an AD-only CCA-secure threshold decryption scheme.*

In particular, for every adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\text{thGS}}$ as in Attack Game 22.5, there exists an adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, that attacks $\mathcal{E}_{\text{GS}}$ as in the strengthened AD-only CCA attack game, such that

$$\text{thCCA}_{\text{ado}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\text{thGS}}] = \text{sCCA}_{\text{ado}}\mathsf{adv}[\mathcal{B}, \mathcal{E}_{\text{GS}}].$$

Proof. We design $\mathcal{B}$ to play the role of challenger to $\mathcal{A}$. Recall that $\mathcal{A}$ begins by outputting the parameters N and t, and a subset $\mathcal{L}$ of size $t-1$ of $\{1, \ldots, N\}$. Now, when $\mathcal{B}$ receives $pk = u$, where $u = g^{\alpha}$, from its own challenger, $\mathcal{B}$ needs to provide to $\mathcal{A}$ not only pk, but also the key shares $\{\alpha_\ell\}_{\ell \in \mathcal{L}}$, as well as $pkc = (u_1, \ldots, u_N)$, where $u_i = g^{\alpha_i}$ for $i = 1, \ldots, N$.

To do this, $\mathcal{B}$ sets $\alpha_\ell \xleftarrow{\text{R}} \mathbb{Z}_q$ for each $\ell \in \mathcal{L}$. By Theorem 22.3, these key shares have the same distribution as in an actual run of Attack Game 22.5. We know that there is a unique polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$ such that $\boldsymbol{\omega}(0) = \alpha$ and $\boldsymbol{\omega}(i) = \alpha_i$ for $i = 1, \ldots, N$.

For $\ell \in \mathcal{L}$, adversary $\mathcal{B}$ can compute $u_\ell = g^{\alpha_\ell}$ directly, since it knows the value α_ℓ. For $i \in \{1, \ldots, N\} \setminus \mathcal{L}$, adversary $\mathcal{B}$ can compute u_i via "interpolation in the exponent", by applying Corollary 22.2 with $\mathcal{J} := \mathcal{L} \cup \{0\}$ and $j^* := i$ to compute $u_i = g^{\boldsymbol{\omega}(i)}$ from the collection of group elements $\{g^{\boldsymbol{\omega}(j)}\}_{j \in \mathcal{J}}$; more explicitly, $\mathcal{B}$ computes u_i as

$$u_i \leftarrow u^{\lambda_{i0}} \, g^{\sum_{\ell \in \mathcal{L}} \alpha_\ell \cdot \lambda_{i\ell}}, \tag{22.2}$$

where the λ_{ij}'s are the Lagrange interpolation coefficients computed as in Lemma 22.1.

Next, whenever $\mathcal{A}$ submits an encryption query, adversary $\mathcal{B}$ forwards this query to its own challenger, and passes back to $\mathcal{A}$ whatever ciphertext it generates.

Whenever $\mathcal{A}$ issues a decryption query $(\hat{c}, \hat{d})$, adversary $\mathcal{B}$ forwards this query to its own challenger, and gets back $\hat{w} \in \mathbb{G} \cup \{\mathsf{reject}\}$. Adversary $\mathcal{B}$ needs to send to $\mathcal{A}$ all the corresponding decryption shares, without knowing the secret key α.

If $\hat{w} = \mathsf{reject}$, all decryption shares are set to reject.

Otherwise, $\hat{w} \in \mathbb{G}$; moreover, if $\hat{c} = (\hat{v}, \hat{e}, \underline{\hat{w}})$, then we have $\hat{w} = \hat{v}^{\alpha} = \hat{v}^{\boldsymbol{\omega}(0)}$.

For $i = 1, \ldots, N$, the ith decryption share is $\hat{w}_i = \hat{v}^{\alpha_i} = \hat{v}^{\boldsymbol{\omega}(i)}$. For $\ell \in \mathcal{L}$, adversary $\mathcal{B}$ can compute $\hat{w}_\ell = \hat{v}^{\alpha_\ell}$ directly, since it knows the value α_ℓ. For $i \in \{1, \ldots, N\} \setminus \mathcal{L}$, adversary $\mathcal{B}$ can compute $\hat{w}_i$ as

$$\hat{w}_i \leftarrow \hat{w}^{\lambda_{i0}} \, \hat{v}^{\sum_{\ell \in \mathcal{L}} \alpha_\ell \cdot \lambda_{i\ell}},$$

where the λ_{ij}'s are precisely the same Lagrange interpolation coefficients as used in (22.2).

When eventually $\mathcal{A}$ outputs a bit $\hat{b}$, our adversary $\mathcal{B}$ outputs the same bit. One sees that $\mathcal{B}$ has the same advantage in its attack game that $\mathcal{A}$ has in its attack game, which completes the proof. □

The next theorem establishes the robustness of $\mathcal{E}_{\text{thGS}}$.

Theorem 22.8. *$\mathcal{E}_{\text{thGS}}$ is a robust threshold decryption scheme. In particular, every adversary has advantage zero in Attack Game 22.6.*

Proof. For accurate blaming, it is easy to see that a correctly generated decryption share will never be flagged as invalid. Conversely, for consistency, it is easy to see that any valid collection of decryption shares will combine to yield the same plaintext as would be output by the regular (non-threshold) GS decryption scheme. □

22.3.6 A pairing-free version of $\mathcal{E}_{\text{thGS}}$

As already mentioned, we do not really need $\mathcal{O}_{\text{DDH}}$ (or pairings) to get an efficient and secure threshold decryption scheme. In fact, we can easily modify $\mathcal{E}_{\text{thGS}}$, replacing the each use of $\mathcal{O}_{\text{DDH}}$ by a non-interactive zero-knowledge proof that a given triple is a DH-triple.

22.3.6.1 $\mathcal{E}_{\text{GS}}^{\dagger}$: a pairing-free version of $\mathcal{E}_{\text{GS}}$

Let us begin by describing a variant of $\mathcal{E}_{\text{GS}}$, denoted $\mathcal{E}_{\text{GS}}^{\dagger}$, which is a regular (non-threshold) encryption scheme.

Just like $\mathcal{E}_{\text{GS}}$, the scheme $\mathcal{E}_{\text{GS}}^{\dagger}$ has a message space $\mathcal{M}$ and associated data space $\mathcal{D}$, and makes use of the following components:

- a cyclic group $\mathbb{G}$ of prime order q generated by $g \in \mathbb{G}$; we shall assume that the CDH assumption holds in $\mathbb{G}$;
- a symmetric cipher $\mathcal{E}_{\text{s}} = (E_{\text{s}}, D_{\text{s}})$ defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C}_{\text{s}})$; we shall assume that $\mathcal{E}_{\text{s}}$ is semantically secure;
- a hash function $H_{\mathcal{K}} : \mathbb{G} \times \mathbb{G} \to \mathcal{K}$; we shall model $H_{\mathcal{K}}$ as a random oracle;
- a hash function $H_{\mathbb{G}} : \mathbb{G} \times \mathcal{D} \to \mathbb{G}$; we shall model $H_{\mathbb{G}}$ as a random oracle.

The scheme $\mathcal{E}_{\text{GS}}^{\dagger}$ also makes use of a non-interactive proof system $\Phi = (\mathit{GenPrf}, \mathit{VrfyPrf})$ for the relation

$$\mathcal{R} = \{ \ (\ \beta, \ (u, v, w)) \ ) \ \in \ \mathbb{Z}_q \times \mathbb{G}^3 \ : \ v = g^{\beta}, \ w = u^{\beta} \ \};$$

We presented the details of such a proof system in Section 20.3.6 based on the application of Fiat-Shamir transform to the Chaum-Pedersen protocol (see Section 19.5.2). As discussed in Section 20.3.6, this proof system (as well as the optimized version) provides both soundness (as in Definition 20.4) and zero knowledge (as in Definition 20.5).

Key generation. The key generation algorithm computes

$$\alpha \in \mathbb{Z}_q, \ \ u \leftarrow g^{\alpha},$$

and outputs the public key u and the secret key α. This is exactly the same as the key generation algorithm for $\mathcal{E}_{\text{GS}}$.

Encryption. To encrypt a message $m \in \mathcal{M}$ with associated data $d \in \mathcal{D}$ under the public key $u \in \mathbb{G}$, the encryption algorithm computes

```
$\beta \xleftarrow{R} \mathbb{Z}_q, \ \ v \leftarrow g^{\beta}, \ \ w \leftarrow u^{\beta}, \ \ k \leftarrow H(v, w), \ \ e \xleftarrow{R} E_{\text{s}}(k, m),$
$\underline{u} \leftarrow H_{\mathbb{G}}(v, d), \ \ \underline{w} \leftarrow \underline{u}^{\beta},$
$\pi \xleftarrow{R} \mathit{GenPrf}( \ \beta, \ (\underline{u}, v, \underline{w}) \ )$
output $c := (v, e, \underline{w}, \pi)$.
```

This is the same as the encryption algorithm for $\mathcal{E}_{\text{GS}}$, except for the additional proof element π in the ciphertext that proves that $(\underline{u}, v, \underline{w})$ is a DH-triple.

Decryption. To decrypt a ciphertext $c = (v, e, \underline{w}, \pi)$ with associated data d under the secret key α, the decryption first checks that

$$VrfyPrf(\ (\underline{u}, v, \underline{w}),\ \pi\ ) = \mathsf{accept}, \quad \text{where} \quad \underline{u} := H_{\mathbb{G}}(v, d). \tag{22.3}$$

If this check passes, we say c **is valid with respect to** d, and the decryption algorithm computes

$$w \leftarrow v^{\alpha}, \ \ k \leftarrow H(v, w), \ \ m \leftarrow D_{\mathrm{s}}(k, e)$$

and outputs m. If this check does not pass, the decryption algorithm outputs reject. This is the same as the decryption algorithm for $\mathcal{E}_{\mathrm{GS}}$, except that rather than invoking $\mathcal{O}_{\mathrm{DDH}}$, the proof π is verified using $VrfyPrf$ to ensure that $(\underline{u}, v, \underline{w})$ is a DH-triple.

Discussion. The scheme $\mathcal{E}_{\mathrm{GS}}^{\dagger}$ was in fact presented in Exercise 20.19, where you were asked to to show that it is AD-only strong-CCA secure under the assumptions listed above.

22.3.6.2 Converting $\mathcal{E}_{\mathrm{GS}}^{\dagger}$ to a pairing-free threshold decryption scheme

We can easily convert $\mathcal{E}_{\mathrm{GS}}^{\dagger}$ to a pairing-free threshold decryption scheme, which we denote $\mathcal{E}_{\mathrm{thGS}}^{\dagger} = (G, E, D, C)$. The scheme is very similar to the scheme $\mathcal{E}_{\mathrm{thGS}}$ presented in Section 22.3.3.2. The only realy difference is that we use the proof system Φ rather than $\mathcal{O}_{\mathrm{DDH}}$ to verify that certain triples are DH-triples.

- $G(N, t) \to (pk, pkc, sk_1, \ldots, sk_N)$:
  - compute
    $$\alpha \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \ \ u \leftarrow g^{\alpha},$$
    and set $pk := u$;
  - run $G_{\mathrm{sh}}(N, t, \alpha)$ to obtain N shares $\alpha_1, \ldots, \alpha_N$ in $\mathbb{Z}_q$;
    Note: recall that for some polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree at most $t-1$, we have $\boldsymbol{\omega}(0) = \alpha$ and $\boldsymbol{\omega}(i) = \alpha_i$ for $i = 1, \ldots, N$;
  - for $i = 1, \ldots, N$, compute $u_i \leftarrow g^{\alpha_i}$;
  - for $i = 1, \ldots, N$, set $sk_i := (\alpha_i, u_i)$;
  - set $pkc := (u_1, \ldots, u_N)$;
  - output $(pk,\ pkc,\ sk_1, \ldots, sk_N)$.

  Note: this key generation algorithm is identical to that in $\mathcal{E}_{\mathrm{thGS}}$, except that sk_i includes u_i; this is not strictly necessary, as u_i could also be computed from α_i.

- $E(pk, m, d) \to c$, where $pk = u$:
  - set $c := (v, e, \underline{w})$, where
    $$\beta \xleftarrow{\mathrm{R}} \mathbb{Z}_q, \ \ v \leftarrow g^{\beta}, \ \ w \leftarrow u^{\beta}, \ \ k \leftarrow H(v, w), \ \ e \xleftarrow{\mathrm{R}} E_{\mathrm{s}}(k, m),$$
    $$\underline{u} \leftarrow H_{\mathbb{G}}(v, d), \ \ \underline{w} \leftarrow \underline{u}^{\beta},$$
    $$\pi \xleftarrow{\mathrm{R}} GenPrf(\ \beta,\ (\underline{u}, v, \underline{w})\ ).$$

Note: this encryption algorithm is *identical* to that in $\mathcal{E}_{\text{GS}}^{\dagger}$.

- $D(sk_i, c, d) \to c_i'$, where $sk_i = \alpha_i$ and $c = (v, e, \underline{w}, \pi)$: check if c is valid with respect to d, as in (22.3); if not, output $c_i' := \textsf{reject}$; otherwise, compute

  $$w_i \leftarrow v^{\alpha_i}, \;\; \pi_i' \leftarrow \mathit{GenPrf}(\; \alpha_i, \; (u_i, v, w_i) \;),$$

  where $u_i := g^{\alpha_i}$, and output $c_i' := (w_i, \pi_i')$.

  Note: this decryption algorithm is identical to that in $\mathcal{E}_{\text{thGS}}$, except for the validity test (22.3), which verifies the proof π, rather than using $\mathcal{O}_{\text{DDH}}$ to verify that $(\underline{u}, v, \underline{w})$ is a DH-triple, and except for the addition of the proof π_i' to prove that (u_i, v, w_i) is a DH-triple.

- $C(pkc, c, d, \mathcal{J}, \{c_j'\}_{j \in \mathcal{J}}) \to m$, where $pkc = (u_1, \ldots, u_N)$, $c = (v, e, \underline{w}, \pi)$, $\mathcal{J}$ is a subset of $\{1, \ldots, N\}$ of size t, and where $c_j' = w_j \in \mathbb{G}$ or $c_j' = \textsf{reject}$ for $j \in \mathcal{J}$:

  – *Step 1:* validate the ciphertext and the decryption shares:

  ∗ check that c is valid with respect to d, as in (22.3): if not, output $\textsf{reject}$ and halt;

  ∗ check if for some $j \in \mathcal{J}$, we have either $c_j' = \textsf{reject}$ or

  $$c_j' = (w_j, \pi_j) \quad \text{and} \quad \mathit{VrfyPrf}(\; (u_j, v, w_j), \; \pi_j' \;) = \textsf{reject};$$

  if so, output $\textsf{blame}(\mathcal{J}^*)$ and halt, where $\mathcal{J}^*$ is the set of all such $j \in \mathcal{J}$;

  Note: assuming all the shares are valid, then for $j \in \mathcal{J}$, we have $c_j' = (w_j, \pi_j)$, and with overwhelming probability (based on the soundness of the proof system π), we have $w_j = v^{\alpha_j} = v^{\boldsymbol{\omega}(j)}$, where $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ is the polynomial of degree at most $t-1$ used to share the secret key α;

  – *Step 2:* compute the ephemeral key w: "interpolate in the exponent" by applying Corollary 22.2 with the given set $\mathcal{J}$ and and $j^* := 0$ to compute the ephemeral key $w := v^{\alpha} = v^{\boldsymbol{\omega}(0)}$;

  more explicitly, w is computed as

  $$w \leftarrow \prod_{j \in \mathcal{J}} w_j^{\lambda_j} \quad \in \mathbb{G}$$

  where the exponents $\{\lambda_j\}_{j \in \mathcal{J}}$ are the Lagrange interpolation coefficients, which depend only on the set $\mathcal{J}$, as computed in Lemma 22.1;

  – *Step 3:* compute $k \leftarrow H_{\mathcal{K}}(v, w)$ and output $m \leftarrow D_{\text{s}}(p, e)$.

  Note: this combiner algorithm is identical to that in $\mathcal{E}_{\text{thGS}}$, except that it uses π to verify that $(\underline{u}, v, \underline{w})$ is a DH-triple and π_j' to verify that (u_j, v, w_j) is a DH-triple, rather than using $\mathcal{O}_{\text{DDH}}$.

We next state a theorem on the security of $\mathcal{E}_{\text{thGS}}^{\dagger}$. The security of $\mathcal{E}_{\text{thGS}}^{\dagger}$ relies on the security of $\mathcal{E}_{\text{GS}}^{\dagger}$ as well as the zero-knowledge property of the proof system Φ, which is used in the decryption algorithm of the threshold scheme. We have formulated the notion of zero knowledge in Section 20.3.5 explicitly in the random oracle model. Moreover, the proof system Φ is also used in the implementation of $\mathcal{E}_{\text{GS}}^{\dagger}$. Therefore, in the following theorem, we state things in terms of the security of $\mathcal{E}_{\text{GS}}^{\dagger}$ in the random oracle model.

Theorem 22.9. *If $\mathcal{E}_{\mathrm{GS}}^{\dagger}$ is AD-only strong-CCA secure in the random oracle model, then $\mathcal{E}_{\mathrm{thGS}}^{\dagger}$ is an AD-only CCA-secure threshold decryption scheme in the random oracle model.*

> *In particular, if Sim be the zero knowledge simulator for Φ (as in Theorem 20.3), then for every adversary $\mathcal{A}$ that attacks $\mathcal{E}_{\mathrm{thGS}}^{\dagger}$ as in the random oracle version of Attack Game 22.5, there exists adversaries $\mathcal{B}_{\mathrm{GS}}$ and $\mathcal{B}_{\mathrm{zk}}$, such that are elementary wrappers around $\mathcal{A}$, where $\mathcal{B}_{\mathrm{GS}}$ attacks $\mathcal{E}_{\mathrm{GS}}^{\dagger}$ as in the random oracle model of the strengthened AD-only CCA attack game, and $\mathcal{B}_{\mathrm{zk}}$ attacks the zero-knowledge property of Φ as in Attack Game 20.3, where*
>
> $$\mathrm{thCCA}^{\mathrm{ro}}_{\mathrm{ado}}\mathsf{adv}[\mathcal{A}, \mathcal{E}_{\mathrm{thGS}}^{\dagger}] \leq \mathrm{sCCA}^{\mathrm{ro}}_{\mathrm{ado}}\mathsf{adv}[\mathcal{B}_{\mathrm{GS}}, \mathcal{E}_{\mathrm{GS}}^{\dagger}] + 2 \cdot \mathrm{niZK}\mathsf{adv}[\mathcal{B}_{\mathrm{zk}}, \Phi, \mathit{Sim}].$$

The next two theorems establish the robustness of $\mathcal{E}_{\mathrm{thGS}}$.

Theorem 22.10. $\mathcal{E}_{\mathrm{thGS}}^{\dagger}$ *provides accurate blaming.*

Proof. This follows directly from the fact that each non-interactive proof generated by a signing server is always valid. □

Theorem 22.11. $\mathcal{E}_{\mathrm{thGS}}^{\dagger}$ *is consistent, assuming Φ is existentially sound.*

Proof. This follows directly from the existential soundness of the non-interactive proofs generated by the signing servers. Indeed, to the extent that the validity of these proofs ensures that the corresponding statements are true, the output of the combiner algorithm will be the same as that of the regular (non-threshold) GS decryption scheme. □

22.4 Distributed key generation

22.4.1 Defining the problem

In our discussion of threshold cryptography so far we assumed that a trusted party runs the key generation process to generate secret keys. The trusted party then hands out secret keys to the signing parties in a threshold signature scheme, or to the decryption parties in a threshold decryption scheme. In this section we show how the parties can engage in a distributed protocol to generate the secret keys themselves, without the help of a trusted party.

Throughout the remainder of Section 22.4, the following notation is fixed:

- q is a prime;
- $\mathbb{G}$ is a group of order q generated by $g \in \mathbb{G}$;
- N, t, L are integers with $0 \leq L < t \leq N < q$.

As usual, N represents the number of parties, and t represents the reconstruction threshold, i.e., the number of shares that need to be combined to reconstruct a secret. The value L is new: it represents a bound on the number of corrupt parties. Up until now, we have been assuming that $L = t-1$, but in this section, we take a somewhat more general view. All of these values are viewed as *system parameters*, which are available to any protocol or algorithm.

Additional notation. We introduce some notation that we will use throughout Section 22.4. Suppose $\boldsymbol{\omega} = \kappa_0 + \kappa_1 x + \cdots + \kappa_{t-1}x^{t-1} \in \mathbb{Z}_q[x]$ is some polynomial of degree less than t with coefficients in $\mathbb{Z}_q$. We define the corresponding vector of group elements

$$g^{\boldsymbol{\omega}} := (g^{\kappa_0}, \ldots, g^{\kappa_{t-1}}) \in \mathbb{G}^t. \tag{22.4}$$

In addition, if $\mathbf{U} = (u_0, \ldots, u_{t-1}) \in \mathbb{G}^t$ is a vector of group elements, and $\beta \in \mathbb{Z}_q$, we define the group element

$$\mathbf{U}^{(\beta)} := \prod_{j=0}^{t-1} u_j^{\beta^j}. \tag{22.5}$$

With this notation, we have the relation

$$(g^{\boldsymbol{\omega}})^{(\beta)} = g^{\boldsymbol{\omega}(\beta)}.$$

The *dealer* algorithm. For both the threshold signature scheme in Section 22.2.2 and the threshold decryption schemes in Sections 22.3.3 and 22.3.6, the core of the trusted key generation process can be implemented in terms of following *dealer* algorithm.

> Algorithm *dealer*() takes no input, and outputs an N-tuple $(out_1, \ldots, out_N)$, computed as follows:
>
> ```
>     compute $\kappa_0, \ldots, \kappa_{t-1} \xleftarrow{R} \mathbb{Z}_q$
>     set $\boldsymbol{\omega} := \kappa_0 + \cdots + \kappa_{t-1}x^{t-1} \in \mathbb{Z}_q[x]$ and $\mathbf{U} := g^{\boldsymbol{\omega}} \in \mathbb{G}^t$
>     for $i = 1, \ldots, N$: set $out_i := (\mathbf{U}, \boldsymbol{\omega}(i))$
> ```

Note that the first component of each out_i value is the same, namely, $\mathbf{U} := g^{\boldsymbol{\omega}}$, while the second component $\boldsymbol{\omega}(i)$ is party P_i's share of the secret $\boldsymbol{\omega}(0) = \kappa_0 \in \mathbb{Z}_q$. Note also that given $\mathbf{U}$ and any $\beta \in \mathbb{Z}_q$, the group element $g^{\boldsymbol{\omega}(\beta)}$ can be efficiently computed as $\mathbf{U}^{(\beta)}$. We shall sometimes denote by $\mathrm{Pub}(out_i)$ the first component of out_i, the "public part" of out_i, which is the value $\mathbf{U}$.

What we want is a protocol Π that allows N parties $P_1, \ldots, P_N$ to implement the logic of the *dealer* algorithm in a secure distributed fashion, without relying on a trusted party. Such a protocol Π should work reliably and securely even if L parties are corrupt. We shall also assume that $N - L \geq t$. Note that this implies that the honest parties themselves have enough shares to reconstruct the secret, which is essential if we want the protocol to succeed without the help of the corrupt parties. Indeed, the protocol we present does guarantee this.

The assumption that $N - L \geq t$ also implies that $N > 2L$. However, for reasons to be discussed below, we may need to additionally assume $N > 3L$ in order to implement a critical subprotocol.

We shall also need to rely on a weak form of trusted system setup. Namely, we will assume **decentralized key provisioning**, by which we mean the following:

- Each honest party P_i generates their own public-key/private key pair (pk_i, sk_i), which generally speaking may be keys for an encryption scheme and/or a signature scheme (or several such schemes).

- Each corrupt party P_ℓ generates its public key pk_ℓ in an arbitrary fashion, possibly depending on the public keys of the honest parties.

- Each honest party is then provisioned with the tuple of public keys $(pk_1, \ldots, pk_N)$.

We have already used this notion previously in the chapter, but without defining it very carefully (see, for example, Remark 22.3). One way to realize semi-trused key provisioning is to use a certificate authority (CA). Alternatively, it can be done by some *ad hoc* manual process.

During the execution of protocol Π, if one honest party sends a message to another honest party, we shall assume that this message is *eventually* delivered; however, we shall not assume any particular bound on the amount of time it takes for a message to be delivered. Such an **asynchronous** network model is the most realistic for a protocol distributed over a wide area network. However, it does make the design of such a protocol more challenging, since it is impossible for a party P_i who is waiting a message from P_j to distinguish between the case that the network is just very slow (and the message will eventually arrive) and the case that P_j is corrupt (and may never send the message).

After the protocol starts running, each honest party P_i should eventually output the value out_i. Because of the asynchronous network model, the honest parties will not produce their outputs all at the same time. The precise timing and ordering of the outputs is dictated by the protocol Π, by the behavior of the corrupt parties, and by the behavior of the communication network. We will think of the corrupt parties and the communication network as being under the control of a single adversary, who is coordinating an attack.

We will state the security requirements more formally in a moment. However, the salient properties are that:

- the outputs of all parties should have essentially the same joint distribution as if they had been generated for them by *dealer*();

- the corrupt parties should jointly learn essentially nothing more than the outputs that would be generated for them by *dealer*().

22.4.1.1 Using a DKG protocol

At a high level, the idea is that we want to use a secure DKG protocol Π as a "drop in" replacement for the key generation algorithm G used in Definitions 22.3 and 22.11 when instantiating these definitions with the BLS threshold signature scheme in Section 22.2.2 and the GS threshold decryption schemes in Sections 22.3.3 and 22.3.6 (respectively). For concreteness, we shall focus here on the BLS threshold signature scheme; however, everything we say translates directly to the GS threshold decryption schemes, and indeed applies to any threshold scheme with a similar structure.

While this high-level idea is generally accurate, there are a few wrinkles.

- Instead of viewing N and t as inputs to G, we now view them (as well as L) as system parameters. The reason is that all of the participants of protocol Π need to agree in advance on these values, and the easiest way to do this is to view these values as system parameters.

- The public key *pk* and combiner public key *pkc* must be available to the relevant entities. The public key *pk* is needed by any entity who needs to verify a signature. The combiner public key *pkc* is needed by any entity who needs to play the role of combiner.

  In our original formulation, we assumed a trusted party that ran algorithm G and then either securely distributed these keys to all relevent entities, or securely published them in some location accessible by all relevent entities. But now, these keys are computed by the parties $P_1, \ldots, P_N$ running the DKG protocol Π. Specifically each party P_i computes and outputs

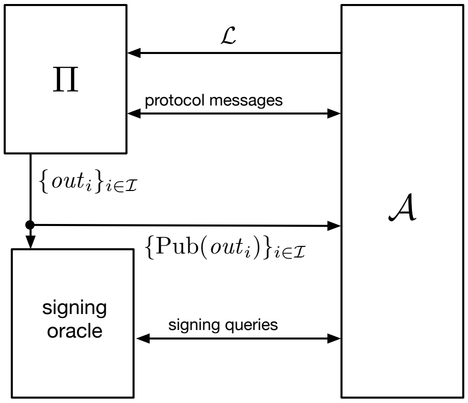


Figure 22.2: Attack Game 22.1 with a DKG protocol Π in place of G

$out_i = (\mathbf{U}, \boldsymbol{\omega}(i))$ where $\mathbf{U} = g^{\boldsymbol{\omega}} = \mathrm{Pub}(out_i)$. As already noted, given $\mathbf{U}$ and any $\beta \in \mathbb{Z}_q$, any entity can efficiently compute the group element $g^{\boldsymbol{\omega}(\beta)}$ as $\mathbf{U}^{(\beta)}$. This means that any entity that knows $\mathbf{U}$ can efficiently compute pk and pkc.

Now, if one of the parties P_i needs pk or pkc, they can compute it themselves as soon as it finishes protocol Π. However, suppose that an entity Q external to $P_1, \ldots, P_N$ needs to obtain pk or pkc. Entity Q can simply ask for this information from $P_1, \ldots, P_N$; however, this must be done with some care, since we are assuming that L of the parties $P_1, \ldots, P_N$ may be corrupt, and Q does not know who these corrupt parties are. Moreover, we are also assuming an asynchronous network, and so Q cannot distinguish between a very slow network and a party P_i that is corrupt and non-responsive. So Q can request this information from all parties $P_1, \ldots, P_N$, and wait for $L+1$ responses that all agree:

- since any set of $L+1$ parties must contain an honest party, Q can be sure that this common value is correct;
- since $N > 2L$, then $N - L \geq L+1$ honest parties will eventualy respond to Q's request, so Q will not be left waiting forever.

Of course, all of this assumes that Q can verify the authenticity of messages coming from each party P_i. To achieve this, we may need to extend the decentralized key provisioning setup assumption to include provisioning all relevent entities Q with the public key of each P_i.

The above discussion focused on the logistics of deploying a threshold signature or decryption scheme that uses a DKG protocol Π. We also need to discuss briefly how using a secure DKG protocol relates to Definitions 22.4 and 22.12 for secure threshold signature and decryption schemes. Again, for concreteness, we focus on secure threshold signature schemes, but as above, our remarks apply to threshold decryption schemes, and more generally, to any threshold schemes with a similar structure.

Fig. 22.2 shows how Attack Game 22.1 is modified to allow for a DKG protocol Π. Just as in Attack Game 22.1, we have an adversary $\mathcal{A}$ and a challenger; however, the attack game and the challenger are split into two components: one component corresponds to the execution of the

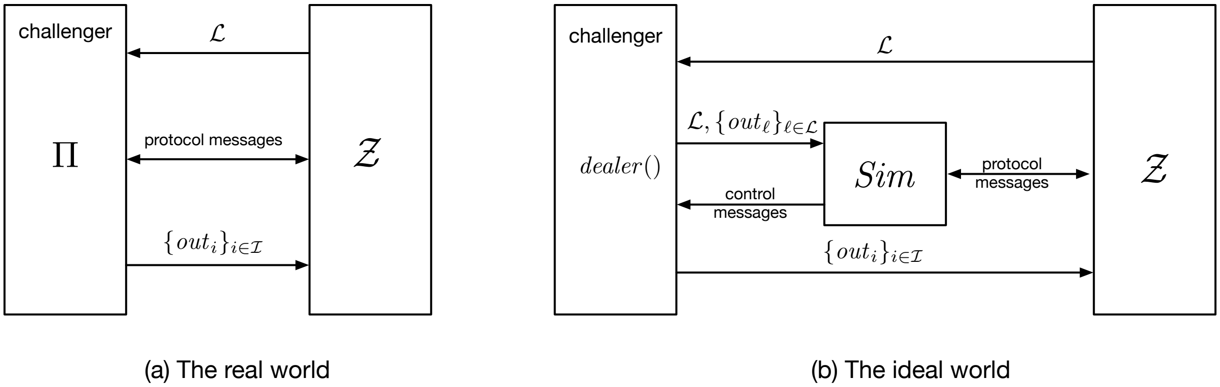


Figure 22.3: The real and ideal worlds for a secure DKG protocol

protocol Π, which replaces the "setup" phase of Attack Game 22.1, and the other component corresponds to the signing queries in Attack Game 22.1.

Just as in Attack Game 22.1, the adversary $\mathcal{A}$ starts out by selecting a subset $\mathcal{L}$ of $L := t - 1$ corrupt parties (as already discussed above, the values N and t are now system parameters, and are not chosen by $\mathcal{A}$). But now, instead of running G, the challenger runs the protocol Π on behalf of the honest parties P_i for $i \in \mathcal{I} := \{1, \ldots, N\} \setminus \mathcal{L}$. Somewhat more precisely, as indicated in Fig. 22.2, the adversary sends various protocol messages to the honest parties, and their responses are sent back to the adversary. Thus, the communication network is essentially under the complete control of the adversary.

As indicated in Fig. 22.2 by the lines labeled $\{out_i\}_{i\in\mathcal{I}}$ and $\{\text{Pub}(out_i)\}_{i\in\mathcal{I}}$, when an honest party P_i finishes the protocol Π, its output out_i is sent to a "signing oracle", while the corresponding public part $\text{Pub}(out_i)$ is given to the adversary. The security of protocol Π will ensure that all of these public parts are actually the same; however, to keep things simple, let us define the public key of the system to be $pk = \mathbf{U}^{(0)}$, where $\mathbf{U}$ is public part output by the first honest party that finishes protocol Π.

Once an honest party P_i that has finished protocol Π, the adversary may make any number of corresponding signing queries, obtaining P_i's signature share on any message of its choosing. This is indicated in Fig. 22.2 by the line labeled "signing queries".

At the end of the attack game, adversary $\mathcal{A}$ attempts to output a forgery pair (m, σ), where m is a message that was never submitted as a signing query to *any* honest party, and σ is a valid signature on m with respect to the public key pk. The whole scheme, consisting of the threshold signature scheme combined with the DKG protocol Π, is considered secure if for every efficient adversary $\mathcal{A}$, its probability of successfully outputting such a forgery pair is negligible.

22.4.1.2 Formal definition of security

The security of a DKG protocol Π is defined via simulation. We will say Π is secure if there is a simulator Sim, such that no efficient adversary $\mathcal{Z}$ can effectively distinguish between the "real world" execution of protocol Π and an "ideal world" in which $\mathcal{Z}$ instead interacts with $dealer()$ via the simulator Sim. We now describe the real world and the ideal world more carefully.

The real world. We model execution of protocol Π in the "real world" as in Fig. 22.3(a). On the right side of this figure is an adversary $\mathcal{Z}$, which interacts with a challenger who among other things is responsible for executing the logic of the honest parties running protocol Π. At the beginning of $\mathcal{Z}$'s attack, it sends a subset $\mathcal{L} \subseteq \{1, \ldots, N\}$ of size at most L, indicating the set of corrupt parties. We let $\mathcal{I} := \{1, \ldots, N\} \setminus \mathcal{L}$ be the set of honest parties. Next the challenger and the adversary interact with message exchanges corresponding to

- the steps of the decentralized key provisioning (discussed above);
- messages sent to/from the honest parties from/to other parties (honest or corrupt);
- queries made to any random oracles or other types of idealized object.

In the figure, these messages are referred to as "protocol messages". As the adversary delivers such protocol messages, the honest parties will eventually (and at different times) generate outputs out_i for $i \in \mathcal{I}$. The challenger delivers these to the adversary, as indicated.

The ideal world. Fig. 22.3(b) shows the "ideal world", which includes the adversary $\mathcal{Z}$, the simulator *Sim*, and a challenger. The key difference is that now, instead of running the protocol Π on behalf of honest parties, the challenger now runs the *dealer* algorithm. In more detail, in the ideal-world execution, the adversary $\mathcal{Z}$ begins just as in the real world by giving the challenger the set $\mathcal{L}$ of corrupt parties. Also, just as in the real world, the adversary $\mathcal{Z}$ exchanges protocol messages, but with the simulator *Sim*, and not the challenger. It is the job of *Sim* to "cook up" realistic looking protocol messages to make the adversary "think" it is still in the real world. The only assistance that *Sim* gets are the outputs out_ℓ for $\ell \in \mathcal{L}$ from *dealer*() for the corrupt parties — these outputs are given to *Sim* right away, in response to providing the challenger with the set $\mathcal{L}$. Also, the challenger will give to $\mathcal{Z}$, but not *Sim*, the outputs out_i for $i \in \mathcal{I}$ from *dealer*() for the honest parties; however, in order to continue making the adversary "think" it is running the real world, the simulator tells the challenger via "control messages" exactly when these outputs should be delivered to the adversary. A well-designed simulator will coordinate these "control messages" with the "protocol messages" so that $\mathcal{Z}$ will receive outputs and protocol messages in the same relative order in the ideal world as it would in the real world.

Attack Game 22.7 (Secure DKG). For a DKG protocol Π, adversary $\mathcal{Z}$, and a simulator *Sim*, we define two experiments. Experiment 0 is the "real world" protocol execution involving Π and $\mathcal{Z}$ described above, and Experiment 1 is the "ideal world" execution involving *dealer*, *Sim*, and $\mathcal{Z}$, also described above. We assume that at the end of each experiment, $\mathcal{Z}$ outputs a bit. If W_b is the event that $\mathcal{Z}$ outputs 1 in Experiment b, define $\mathcal{Z}$'s **advantage** with respect to Π and *Sim* as

$$\mathrm{DKG}\mathsf{adv}[\mathcal{Z}, \Pi, \mathit{Sim}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \Box$$

Definition 22.16 (Secure DKG). *A DKG protocol* Π *is* ***secure*** *if there exists an efficient simulator Sim such that for every efficient adversary* $\mathcal{Z}$*, the value* $\mathrm{DKG}\mathsf{adv}[\mathcal{Z}, \mathcal{E}, \mathit{Sim}]$ *is negligible.*

22.4.1.3 Connection to threshold signing and decryption

We now discuss the connection between the above definition of a secure DKG protocol and the variation of the threshold signature attack game discussed in Section 22.4.1.1, and illustrated in

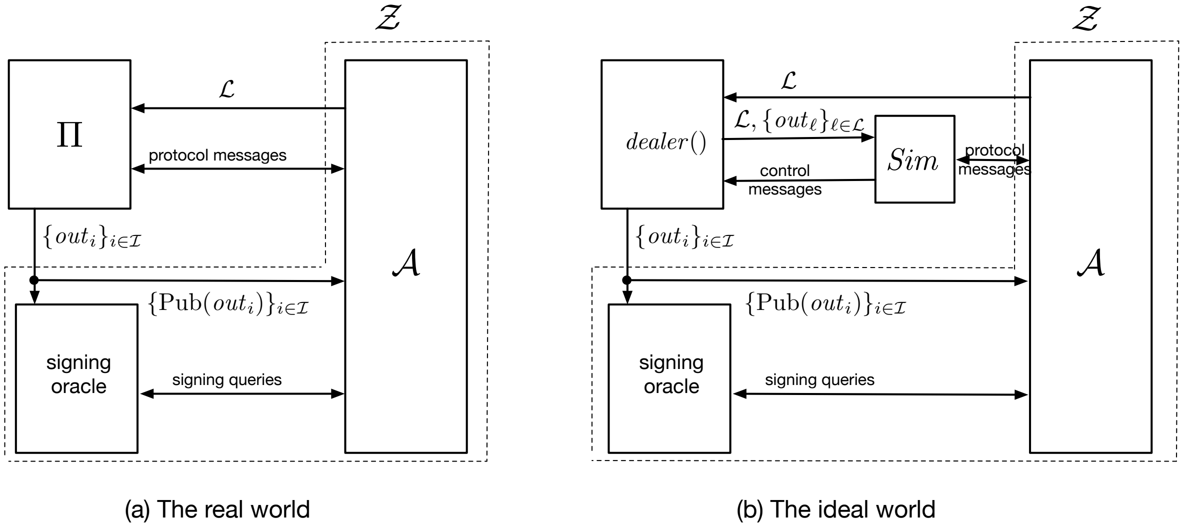


Figure 22.4: The real and ideal worlds in the threshold signing and decryption attack games

Fig. 22.2. As always, everything we say here applies as well to threshold decryption, as well as any threshold schemes with a similar structure.

Roughly speaking, the adversary $\mathcal{Z}$ in Attack Game 22.7 plays the role of *both the adversary and the callenger* in the attack game illustrated in Fig. 22.2. This is illustrated in Fig. 22.4(a). shares corresponding to corrupt parties. The adversary $\mathcal{Z}$ in Fig. 22.4(a) will output 1 if adversary $\mathcal{A}$ forges a signature.

The security of the DKG protocol Π implies that we can effectively replace Π by the *dealer* algorithm plus a simulator *Sim*, as shown in Fig. 22.4(b), without changing the probability that $\mathcal{Z}$ outputs a 1 by more than a negligible amount. Moreover, one sees that in Fig. 22.4(b), the adversary $\mathcal{B}$, consisting of $\mathcal{A}$ together with *Sim*, essentially corresponds to an attack on a version of Attack Game 22.1 in which a trusted party generates all of the keys, and the analysis we did earlier in the chapter implies that the advantage of $\mathcal{B}$ in this game is negligible. All of this together implies that the advantage of adversary $\mathcal{A}$ in the attack game illustrated in Fig. 22.2 is also negligible.

22.4.2 A simple DKG protocol

In this section, we present a DKG protocol. While it is conceptually simple, it is rather expensive from the point of view of computational and communication complexity.

Before presenting the protocol itself in Section 22.4.2.3, we shall first present two cryptographic tools that we shall need for the protocol in Sections 22.4.2.1 and 22.4.2.2.

22.4.2.1 Verifiable encryption of a secret sharing

Roughly speaking, a **verifiably encrypted secret sharing scheme**, or **VESS scheme**, is a semantically secure public-key encryption scheme that allows one party, say P_k, to encrypt a t-out-of-N Shamir sharing of a secret under public keys of parties $P_1, \ldots, P_N$, so that any party can efficiently verify that the ciphertext does indeed encrypt such a sharing.

In somewhat more detail, we assume that each party P_i generates a vector of public encryption keys $\boldsymbol{pk}_i = (pk_i^{(1)}, \ldots, pk_i^{(N)})$, along with a vector of corresponding private decryption keys $\boldsymbol{sk}_i = (sk_i^{(1)}, \ldots, sk_i^{(N)})$, for a semantically secure public-key encryption scheme. We assume a *decentralized key provisioning* mechanism, as discussed above, so that all parties obtain the public key vectors $\boldsymbol{pk}_1, \ldots, \boldsymbol{pk}_N$, but corrupt parties may generate their public keys in an arbitrary fashion. The idea is that a party P_k encrypts a message to P_i under the public encryption key $pk_i^{(k)}$ (but see Remark 22.12 below).

Now, to verifiably encrypt a secret sharing, party P_k generates $\kappa_0, \ldots, \kappa_{t-1} \in \mathbb{Z}_q$ at random and sets $\boldsymbol{\omega} = \kappa_0 + \kappa_1 x + \cdots + \kappa_{t-1} x^{t-1} \in \mathbb{Z}_q[x]$. Party P_k then generates a **transcript** $(\mathbf{C}, \mathbf{U}, \pi)$, where

- $\mathbf{C} = (c_1, \ldots, c_N)$ is a vector of ciphertexts, where each c_i is an encryption of P_i's share $\alpha_i := \boldsymbol{\omega}(i)$ under the public key $pk_i^{(k)}$;
- $\mathbf{U} := g^{\boldsymbol{\omega}} \in \mathbb{G}^t$;
- π is a non-interactive zero-knowledge proof that c_i is indeed an encryption of $\alpha_i \in \mathbb{Z}_q$ under $pk_i^{(k)}$ satisfying $g^{\alpha_i} = \mathbf{U}^{(i)}$ for $i = 1, \ldots, N$.

While it is possible to define a VESS scheme as an abstract cryptographic primitive, to keep things more concrete, we shall work with a VESS scheme as just described. Formally, such a scheme consists of

- a semantically public key encryption scheme $\mathcal{E}$ with message space $\mathbb{Z}_q$;
- a non-intertactive proof system Φ (see Section 20.3) for an appropriate relation that provides existential soundness (see Section 20.3.4) and non-interactive zero knowledge (see Section 20.3.5);

  the relation must be chosen so that soundness implies (with overwhelming probability) that given $(\mathbf{C}, \mathbf{U}, \pi)$ along with public keys $pk_1^{(k)}, \ldots, pk_N^{(k)}$ as above, if $\mathbf{C} = (c_1, \ldots, c_N)$ and the proof π is valid, then for each honest party P_i, the ciphertext c_i decrypts under the secret key $sk_i^{(k)}$ to α_i satisfying $g^{\alpha_i} = \mathbf{U}^{(i)}$.

Remark 22.11 (Implementing a secure VESS scheme). One approach to implementing a secure VESS scheme is to base it on the ideas in Exercise 20.11. There, you were asked to work out the details of using the multiplcative ElGamal encryption scheme to encrypt a discrete logarithm bit by bit, and to design a Sigma protocol to prove that the encryptor does this correctly. One can extend such a Sigma protocol using the AND-proof construction of Section 19.7 to encrypt all the Shamir shares of a secret and prove that this was done correctly. Finally, one can convert this Sigma protocol into a non-interactive zero-knowledge proof using the Fiat-Shamir heuristic, as in Section 20.3. While this protocol works, there is plenty of room for practical improvements. □

Remark 22.12 (Reducing the number of keys). Instead of a vector of public keys $\boldsymbol{pk}_i$ for each party P_i, we could have just a single public key pk_i. However, in this case, we would require that the encryption scheme be a CCA-secure scheme that supports associated data (as in Section 12.7). Specifically, a party P_k would encrypt a message to P_i under the key pk_i with associated data k. In fact, for this (and many similar secure distributed computations), it suffices that the encryption scheme satisfies the weaker notion of AD-only CCA security discussed in Section 12.7.1. To this end, the reader may wish to work out the details of the analog of Exercise 20.11 based on the

AD-only CCA-secure encryption scheme presented in Exercise 20.20, rather than multiplicative ElGamal. □

22.4.2.2 A secret bulletin board

Our DKG protocol will crucially depend on a sub-protocol that securely implements a **secret bulletin board**, which essentially allows each party to *privately* submit a proposed message to the sub-protocol, so that each party eventualy obtains as output the *same* collection of messages.

We will describe the security properties of such a sub-protocol in terms of a trusted party *SBB* that provides a secret bulletin board as a service to all of the parties $P_1, \ldots, P_N$ running the DKG protocol. In reality, there is no trusted party *SBB*, and the service provided by *SBB* will have to be implemented by $P_1, \ldots, P_N$ themselves. However, we can modularly design a secure DKG protocol using the trusted party *SBB* and then replace *SBB* by a sub-protocol that securely implements *SBB* — this will result in a secure DKG protocol that does not rely on any trusted party.

One can securely implement the service provided by *SBB* using fairly efficient protocols, even over an asynchronous network, under our assumption that $N > 3L$. Moreover, these protocols do not require any trusted setup beyond the type of decentralized key provisioning discussed in Section 22.4.1. Below, in Section 22.4.2.4, we will see how to securely implement *SBB* in terms of a *public* bulletin board, which itself a special case of the well-studied problem of *Byzantine Agreement*. The formalities of secure composition of protocols will be discussed in much more detail in Chapter 23 (in particular, see Section 23.6.1).

So let us define how *SBB* is supposed to behave. We assume we have N parties $P_1, \ldots, P_N$. The protocol is parameterized by the value L, which is the bound on the number of corrupt parties, as well as a value s, where $L < s \leq N - L$, which is the size of the collection of messages output by the *SBB*. The idea is that each party P_i proposes a message m_i for the bulletin board by sending its proposal m_i *privately* to *SBB*. Eventually, *SBB* will choose a collection $\mathcal{C} = \{m_k\}_{k \in \mathcal{K}}$ of s such proposals (so $|\mathcal{K}| = s$), and send $(\mathcal{K}, \mathcal{C})$ to all parties. We assume an adversary chooses a set of at most L parties to corrupt at the beginning of its attack, and may choose the proposals for all of these corrupt parties. Note that the size parameter s cannot be larger than $N - L$ simply because L corrupt parties may refuse to cooperate and propose any message to the bulletin board.

In more detail, *SBB* works as follows.

- Each party P_i (both honest and corrupt) sends its proposed message m_i privately to *SBB*.

  We assume that the channel on which P_i sends m_i to *SBB* is perfectly secure; more precisely, when a party sends m_i to *SBB*, the adversary is informed of the index i of the party P_i that submitted m_i, but is not given any information about m_i; moreover, the adversary cannot alter the message m_i.

- At some point in time after *SBB* has received proposals from a set $\mathcal{K}^*$ of parties, where $|\mathcal{K}^*| \geq s$, the adversary chooses a subset $\mathcal{K} \subseteq \mathcal{K}^*$ of size s, and sends $\mathcal{K}$ to *SBB*. At this time, *SBB* sends $(\mathcal{K}, \mathcal{C})$, where $\mathcal{C} := \{m_k\}_{k \in \mathcal{K}}$ to all parties.

  Note that while each honest party will *eventually* receive *SBB*'s output $(\mathcal{K}, \mathcal{C})$, the adversary will determine exacty when each individual party actually receives it by sending *SBB* an appropriate control message.

The above description of *SBB* guarantees that every honest party sees the same collection $\mathcal{C}$ of s proposed messages. Moreover, at least one of the messages in $\mathcal{C}$ was actually proposed by an honest party, since by assumption, there are at most $L < s$ corrupt parties.

The adversary has some very limited control over $\mathcal{C}$. Indeed, the adversary can determine which honest parties' proposals are included in $\mathcal{C}$, and can determine the proposals in $\mathcal{C}$ from the corrupt parties in any way it likes. However, *the corrupt proposals will not depend on any of the honest proposals.*

22.4.2.3 The DKG protocol

Now we have all the ingredients we need to construct a simple yet secure DKG protocol. We need

- a VESS scheme, as described in Section 22.4.2.1 consisting of a semantically secure public-key encryption scheme $\mathcal{E}$ with message space $\mathbb{Z}_q$, and a corresponding non-interactive proof system Φ, and
- a secret bulletin board *SBB*, as described in Section 22.4.2.2, with the size parameter $s := L + 1$.

The protocol, which we call Π_{sdkg}, runs as follows.

1. In the decentralized key provisioning phase, each honest party P_i generates its own public-key/secret-key vector pair $(\boldsymbol{pk}_i, \boldsymbol{sk}_i)$ for the encryption scheme $\mathcal{E}$, and is provisioned with the public-key vectors $(\boldsymbol{pk}_1, \ldots, \boldsymbol{pk}_N)$ of all parties. Recall that each $\boldsymbol{pk}_i$ is a vector of public keys $(pk_i^{(1)}, \ldots, pk_i^{(N)})$ and each $\boldsymbol{sk}_i$ is a vector of secret keys $(sk_i^{(1)}, \ldots, sk_i^{(N)})$.

2. Each honest party P_i locally generates a random polynomial $\boldsymbol{\omega}_i \in \mathbb{Z}_q[x]$ of degree less than t, and generates a corresponding transcript
$$(\mathbf{U}_i, \mathbf{C}_i, \pi_i)$$
as described in Section 22.4.2.1. Recall that $\mathbf{U}_i = g^{\boldsymbol{\omega}_i}$.

3. Each party P_i submits its own transcript $(\mathbf{U}_i, \mathbf{C}_i, \pi_i)$ to *SBB*.

4. Eventually, each honest P_i party obtains from *SBB* the pair $(\mathcal{K}, \mathcal{C})$, where
$$\mathcal{C} = \{(\mathbf{U}_k, \mathbf{C}_k, \pi_k)\}_{k \in \mathcal{K}},$$
and $\mathcal{K}$ is a subset of $\{1, \ldots, N\}$ of size $L + 1$.

   Party P_i then locally performs the following computation:

   (a) compute
   $$\mathcal{K}' := \big\{\ k \in \mathcal{K} : \pi_k \text{ is a valid proof } \big\};$$

   (b) for $k \in \mathcal{K}'$, decrypt c_{ki} under the secret key $sk_i^{(k)}$ to obtain $\alpha_{ki} \in \mathbb{Z}_q$, where $\mathbf{C}_k = (c_{k1}, \ldots, c_{kN})$;

   (c) compute
   $$\mathbf{U} \leftarrow \prod_{k \in \mathcal{K}'} \mathbf{U}_k \quad \text{and} \quad \alpha_i \leftarrow \sum_{k \in \mathcal{K}'} \alpha_{ki}.$$

   Finally, P_i outputs $out_i := (\mathbf{U}, \alpha_i)$.

Before analyzing the security of protocol Π_{sdkg}, let us first at least verify that it computes correct values, at least when all parties are honest. For $k \in \mathcal{K}'$, we have $\mathbf{U}_k = g^{\boldsymbol{\omega}_k}$, and for each $i = 1, \ldots, N$, the ciphertext c_{ki} is an encryption of the plaintext $\alpha_{ki} = \boldsymbol{\omega}_k(i)$ under the public key $pk_i^{(k)}$. Let $\boldsymbol{\omega} = \sum_{k \in \mathcal{K}'} \boldsymbol{\omega}_k$. Then one may easily verify that $\mathbf{U} = g^{\boldsymbol{\omega}}$ and $\alpha_i = \boldsymbol{\omega}(i)$ for $i = 1, \ldots, N$, as required.

Theorem 22.12. *Protocol* Π_{sdkg} *is a secure DKG protocol, assuming* $N - L \geq t$*, and that we have decentralized key provisioning, a secure VESS scheme, and a secret bulletin board SBB.*

Proof sketch. We have to show that protocol Π_{sdkg} satisfies the security definition in Section 22.4.1.2. To that end, we have to show the existence of an efficient simulator *Sim* so that no adversary $\mathcal{Z}$ can distinguish between the real world, as presented in Fig. 22.3(a), in which $\mathcal{Z}$ interacts with Π_{sdkg}, and the ideal world, as presented in Fig. 22.3(b), in which $\mathcal{Z}$ instead interacts with the dealer and *Sim*.

Now, the real world protocol Π_{sdkg} actually contains several idealized components, and the "protocol messages" in the real world are really just the interactions between $\mathcal{Z}$ and these idealized components.

- First, we are assuming decentralized key provisioning, as discussed in Section 22.4.1. This means that the challenger in the real world generates public-key/secret-key pairs on behalf of all of the honest parties for the encryption scheme used in the VESS scheme, and gives the public keys to $\mathcal{Z}$; the adversary $\mathcal{Z}$ then generates public keys for the encryption scheme for the corrupt parties and gives these to the challenger.

- Second, the VESS scheme consists of a non-interactive proof system Φ for an appropriate relation that provides existential soundness and non-interactive zero knowledge. Our notion of non-interactive zero knowledge (see Section 20.3.5) is explicitly defined in terms of a random oracle, and so even in the real world, the adversary $\mathcal{Z}$ makes explicit queries to this random oracle, which are processed by the challenger.

- Third, the secret bulletin board *SBB* used by Π_{sdkg} is modeled here as a trusted party, so that even in the real world, the adversary $\mathcal{Z}$ explicitly interacts with *SBB*: $\mathcal{Z}$ explicitly supplies to *SBB* the proposals of the corrupt parties, and is explicitly informed by *SBB* when an honest party supplies a proposal; $\mathcal{Z}$ also supplies explicit control messages to *SBB* to indicate the subset of output proposals and to schedule when each honest party receives these proposals. All of these interactions between $\mathcal{Z}$ and *SBB* are processed by the challenger.

  We will see later (see Section 22.4.2.4) how to securely implement *SBB* using a distributed protocol, so that the only idealized components needed are a random oracle and decentralized key provisioning.

We now sketch the design of the simulator *Sim*. Recall that $\mathcal{L}$ is the set of indices of corrupt parties. Let $\mathcal{I} := \{1, \ldots, N\} \setminus \mathcal{L}$, which is the set of indices of the honest parties. Also recall that $\mathcal{K}$ is the set of indices supplied by the adversary $\mathcal{Z}$ to *SBB*, and $\mathcal{K}'$ is the set of indices for which the corresponding proposals consist of transcripts with valid proofs.

The first thing that happens in the ideal world is that $\mathcal{Z}$ provides $\mathcal{L}$ to the challenger, who runs the dealer algorithm and gives to *Sim* the collection of values $\{out_\ell\}_{\ell \in \mathcal{L}}$. So now *Sim* has $\mathbf{U} \in \mathbb{G}^t$ and the secret shares α_ℓ for $\ell \in \mathcal{L}$. The simulator *Sim* continues by running the decentralized key provisioning logic just as in the real protocol. After this, the adversary $\mathcal{Z}$ will give to *Sim* transcripts from various corrupt parties — remember that all "protocol messages" that $\mathcal{Z}$ would give to the challenger in the real world are given directly to *Sim* in the ideal world. In addition, whenever an honest party P_i is initialized, *Sim* will simply send to $\mathcal{Z}$ its index, as if from *SBB* in the real world. Eventually, $\mathcal{Z}$ will give to *Sim* the set $\mathcal{K}$, which also determines the set $\mathcal{K}'$. At this point in time (and no earlier), *Sim* will "cook up" consistent-looking transcripts on behalf of the honest parties in the set $\mathcal{K}$. Note that *Sim* does not get to see the secret shares α_i for $i \in \mathcal{I}$, while $\mathcal{Z}$ does get to see these shares. This means that the transcripts that *Sim* cooks up will be "fake" transcripts that must be computed without the knowledge of these shares. However, based on the security properties of the VESS scheme, and exploiting the fact that *Sim* knows $\mathbf{U}$ and $\{\alpha_\ell\}_{\ell \in \mathcal{L}}$, the simulator *Sim* can cook up this transcript in such a way that the adversary $\mathcal{Z}$ will not notice that it is indeed fake.

To provide further details of *Sim* and to prove that it works, we shall present a sequence of games, Game 0, ..., Game 5, where Game 0 is the real world, and Game 5 is equivalent to the ideal world.

Game 1: In this game, we make a couple of small, essentially syntactic changes. First, we have the challenger delay the generation of the transcripts for the honest parties until after the set $\mathcal{K}$ has been specified — in particular, until after the adversary $\mathcal{Z}$ has already submitted its proposals to *SBB*. Moreover, it is only necessary to generate a transcript $(\mathbf{U}_k, \mathbf{C}_k, \pi_k)$ for honest parties P_k with $k \in \mathcal{K}$.

In addition, whenever the challenger decrypts a ciphertext c_{ki} from such an honest party P_k on behalf of any honest party P_i, it does not decrypt the ciphertext at all, but simply uses the plaintext value $\alpha_{ki} = \boldsymbol{\omega}_k(i)$.

Because of the behavior of *SBB*, and under the correctness property for the encryption scheme, there is no detectable difference between Games 0 and 1.

Game 2: In this game, we play our "soundness card", so that when the adversary specifies the set $\mathcal{K}$ of indices to *SBB*, the challenger tests whether there is some $k \in \mathcal{K}' \cap \mathcal{L}$ such that the given transcript from party P_k is of the form $(\mathbf{U}_k, \mathbf{C}_k, \pi_k)$, where

- $\mathbf{C}_k = (c_{k1}, \ldots, c_{kN})$, but
- c_{ki} does not decrypt to the right value under secret key the secret key $sk_i^{(k)}$ for some $i \in \mathcal{I}$.

If so, the challenger immediately "aborts", meaning that it responds to all subsequent queries by the adversary $\mathcal{Z}$ with an "aborted" message.

Under the assumption of the soundness of the underlying proof system Φ, such an "abort" happens with negligible probability, and $\mathcal{Z}$ cannot effectively distinguish between Games 1 and 2.

Game 3: Now we play our "zero knowledge card", so that in the transcript $(\mathbf{U}_k, \mathbf{C}_k, \pi_k)$ generated by each honest party P_k for $k \in \mathcal{K}$, the proof π_k is replaced by a simulated proof.

Under the zero-knowledge assumption for the proof system Φ, $\mathcal{Z}$ cannot effectively distinguish between Games 2 and 3.

Game 4: Now we play our "semantic security card", so that for the transcript $(\mathbf{U}_k, \mathbf{C}_k, \pi_k)$ generated by each honest party P_k for $k \in \mathcal{K}$, if $\mathbf{C}_k = (c_{k1}, \ldots, c_{kN})$, for each honest party P_i, the ciphertext c_{ki} is replaced by an encryption of 0.

Under the semantic security assumption, and the fact that we have already replaced the corresponding proof π_i by a simulated proof, and the fact that we are not using any of the relevant decryption keys $sk_i^{(k)}$, adversary $\mathcal{Z}$ cannot effectively distinguish between Games 3 and 4.

Let us pause and take stock of what is happening in Game 4. Specifically, let us look at the transcript $(\mathbf{U}_k, \mathbf{C}_k, \pi_k)$ for each $k \in \mathcal{K}'$, assuming the challenger does not abort. Let us write $\mathbf{U}_k = (u_{k,0}, \ldots, u_{k,t-1})$ and $\mathbf{C}_k = (c_{k1}, \ldots, c_{kN})$. Let $\mathcal{M}$ consist of 0 and the indices of $t - |\mathcal{L}| - 1$ arbitrary honest parties (in the case where $|\mathcal{L}| = L = t - 1$, we have $\mathcal{M} = \{0\}$).

- Suppose $k \in \mathcal{L}$. Then the ciphertexts c_{ki} for $i \in \mathcal{I}$ all decrypt to correct values, that is, each c_{ki} decrypts to $\alpha_{ki} \in \mathbb{Z}_q$ that satisfies $g^{\alpha_{ki}} = \mathbf{U}_k^{(i)}$. Moreover, the assumption that $N - L \geq t$ implies that the number of honest parties is at least t, which means that from the collection of values $\{\alpha_{ki}\}_{i \in \mathcal{I}}$, we can interpolate to recover the unique polynomial $\boldsymbol{\omega}_k \in Z_q[x]$ of degree less than t that satisfies $g^{\boldsymbol{\omega}_k} = \mathbf{U}_k$.

- Suppose $k \in \mathcal{I}$. The ciphertexts $c_{k\ell}$ for $\ell \in \mathcal{L}$ encrypt elements $\alpha_{k\ell}$ of $\mathbb{Z}_q$. Moreover, the random variables $\alpha_{k\ell}$ for $\ell \in \mathcal{L}$ are uniformly distributed over $\mathbb{Z}_q$, and the random variables $\mathbf{U}_k^{(i)}$ for $i \in \mathcal{M}$ are uniformly distributed over $\mathbb{G}$; in addition, these random variables $\alpha_{k\ell}$ and $\mathbf{U}_k^{(i)}$ are mutually independent and together determine the group elements $u_{k,0}, \ldots, u_{k,t-1}$ via "interpolation in the exponent". Finally, for $i \in \mathcal{I}$, the ciphertext c_{ki} and the proof π_k are "fake".

To visualize this better, consider the following table:

	$\ell \in \mathcal{L}$	$i \in \mathcal{M}$
$k \in \mathcal{K}' \cap \mathcal{L}$	$\boldsymbol{\omega}_k(\ell)$	$\boldsymbol{\omega}_k(i)$
$k \in \mathcal{K}' \cap \mathcal{I}$	$\alpha_{k\ell}$	$\mathbf{U}_k^{(i)}$

The rows of this table correspond to transcripts generated by parties P_k for $k \in \mathcal{K}'$. The columns corresppond to the polynomial evaluation points $\ell \in \mathcal{L}$ and $i \in \mathcal{M}$. As discussed above, in the rows corresponding to $k \in \mathcal{K}' \cap \mathcal{L}$, the values $\boldsymbol{\omega}_k(\ell)$ and $\boldsymbol{\omega}_k(i)$ are all known to the challenger, and in the rows corresponding to $k \in \mathcal{K}' \cap \mathcal{I}$, the random variables $\alpha_{k\ell}$ and $\mathbf{U}_k^{(i)}$ are uniformly distributed and mutually independent.

Let us define

$$\boldsymbol{\omega}^* := \sum_{k \in \mathcal{K}' \cap \mathcal{L}} \boldsymbol{\omega}_k \quad \in \mathbb{Z}_q[x].$$

As we saw above, the challenger in Game 4 can efficiently compute $\boldsymbol{\omega}^*$.

Game 5: In this game, the logic of the challenger is modified to use the dealer algorithm as in the ideal world. The dealer chooses a random polynomial $\boldsymbol{\omega} \in \mathbb{Z}_q[x]$ of degree less than t, and gives to the challenger $\mathbf{U} := g^{\boldsymbol{\omega}} = (u_0, \ldots, u_{t-1})$ and $\alpha_\ell := \boldsymbol{\omega}(\ell)$ for $\ell \in \mathcal{L}$. Now, when the adversary $\mathcal{Z}$ specifies the set $\mathcal{K}$ of indices to *SBB*, the challenger will use the polynomial $\boldsymbol{\omega}^*$ defined above to "cook up" appropriate transcripts for the honest parties P_k with $k \in \mathcal{K}$ in such a way that for each honest party P_i, the logic in Step 4 of protocol Π_{sdkg} will compute the public part of its output as $\mathbf{U}$ and its share of the secret key α_i as $\boldsymbol{\omega}(i)$.

To this end, for each $\ell \in \mathcal{L}$, the challenger generates $\{\alpha_{k\ell}\}_{k\in\mathcal{K}\cap\mathcal{I}}$ at random from $\mathbb{Z}_q$, subject to

$$\boldsymbol{\omega}^*(\ell) + \sum_{k\in\mathcal{K}\cap\mathcal{I}} \alpha_{k\ell} = \alpha_\ell.$$

Also, for each $i \in \mathcal{M}$, the challenger generates $\{\mathbf{U}_k^{(i)}\}_{k\in\mathcal{K}\cap\mathcal{I}}$ at random from $\mathbb{G}$, subject to

$$g^{\boldsymbol{\omega}^*(i)} \prod_{k\in\mathcal{K}\cap\mathcal{I}} \mathbf{U}_k^{(i)} = \mathbf{U}^{(i)}.$$

Then, for each $k \in \mathcal{K}\cap\mathcal{I}$, the challenger can compute $\mathbf{U}_k$ from $\{\alpha_{k\ell}\}_{\ell\in\mathcal{L}}$ and $\{\mathbf{U}_k^{(i)}\}_{i\in\mathcal{M}}$ by "interpolation in the exponent". The challenger can also compute the encryptions c_ℓ of $\alpha_{k\ell}$ for $\ell \in \mathcal{L}$, and combine these with encryptions c_i of 0 for $i \in \mathcal{I}$, thus obtaining $\mathbf{C}_k = (c_{k1}, \ldots, c_{kN})$. Finally, the challenger can generate a "fake" proof π_k, using the zero-knowledge simulator for Φ.

One sees that this game is logically equivalent to Game 4. Moreover, one sees that this game is also logically equivalent to the ideal world depicted in Fig. 22.3(b). We leave it to the reader to extract from our description of this game the logic of the simulator *Sim*. □

Remark 22.13. Although Theorem 22.12 only assumes $N - L \geq t$, in order to implement *SBB* in the asynchronous network communication model, we require $L < N/3$. □

Remark 22.14. As presented, protocol Π_{sdkg} is only intended to be executed just once. However, with a minor modification, it can be securely executed multiple times, using the same keys provided by the decentralized key provisioning across all executions. Moreover, these executions can be run either sequentially or concurrently with one another. To achieve this, we just need separate, independent instances of *SBB*, one for each instance of protocol Π_{sdkg}.

We have not formally defined the security of a DKG protocol in this multi-instance setting, but it is fairly straightforward to do so. Note that in such a model, the set $\mathcal{L}$ of corrupt parties is the same across all execution instances.

We also note that if the DKG protocol will actually be run only once using the provisioned keys, then if we replace the vector $\boldsymbol{pk}_i$ of keys by a single key pk_i as discussed in Remark 22.12, then the reader may verify that the only security property required of the public-key encryption scheme is that it is non-adaptive CCA secure, as in Exercise 12.28. □

22.4.2.4 Implementing a secret bulletin board

We briefly sketch how to implement a secret bulletin board in terms of a *public* bulletin board.

We can define the service provided by a public bulletin board in terms of a trusted party *PBB* that behaves almost identically to *SBB*, but with the following changes:

- When a party P_i sends a message m_i to *PBB*, it is not sent privately; in fact, if P_i is an honest party, *PBB*, explicitly sends m_i to the adversary.
- In addition, *PBB* will only accept a message m_i if it passes an application-dependent *validity test*, which (in addition to the index i and the message m_i) may take as input public data, such as public keys.

Byzantine Agreement. The problem of implementing a public bulletin board is a special case of a classic and well-studied problem called **Byzantine Agreement**. In a Byzantine Agreement protocol, we have N parties, L of which may be corrupt and behave arbitrarily (in the literature in this area, the corrupt parties are called *Byzantine*). The goal of such a protocol is to have all of the honest parties agree on a common value. There are typically some constraints on the allowable common values (as otherwise, the honest parties could just agree in some predetermined default value). For example, for our public bulletin board, the common value would be a set of s triples of the form (i, m_i, σ_i), where the i values are distinct, each m_i satisfies an application-dependent validity test, and each σ_i is a valid signature on m_i under P_i's public key.

Any Byzantine Agreement protocol should satisfy the following two properties:

Safety: no two honest parties produce different values;

Liveness: all honest parties eventually produce some value.

Some protocols for this problem only work if the communication network is assumed to be *synchronous*, which means that there is a bound δ such that all protocol messages sent from one honest party to another are guaranteed to be delivered within time δ. These are the most practical Byzantine Agreement protocols, and with some kind of PKI (such as the decentralized key provisioning we are assuming in this section), they can withstand $L < N/2$ corruptions, which is optimal (without a PKI, $L < N/3$ is optimal).

However, such a synchrony assumption is unrealistic in many settings, such as a globally distributed network of parties, which is why we have not made this assumption in this section. Instead, we assume only an *asynchronous* network, where there is no *a priori* bound on message delivery time, even though we do assume that all messages are *eventually* delivered. Liveness in this setting means that

- the total number of messages generated by all honest parties is poly-bounded with overwhelming probability, and
- when all messages generated by honest parties have been delivered, then all honest parties will produce a value.

There are theoretically efficient protocols for asynchronous Byzantine Agreement that can withstand $L < N/3$ corruptions, which is optimal. However, these protocols are not very practical. A reasonable tradeoff that is often made is to insist that safety is guaranteed without any synchrony assumption, but that liveness is only guaranteed when the network eventually becomes synchronous (at least for some sufficiently long period of time). There are very practical protocols that satisfy these assumptions that can also withstand $L < N/3$ corruptions, which is optimal.

Implementing *SBB* using *PBB*. We next sketch one simple way to implement a secure *SBB* using a *PBB*. The protocol we present makes use of an $(L+1)$-out-of-N VESS scheme, and we require that $N > 2L$. Assume that messages m_i can be encoded as bit strings of a fixed length, and let $\mathcal{M}$ denote the set of all such bit strings. We also need a hash function $H : \mathbb{G} \to \mathcal{M}$, which will be viewed as a random oracle.

When party P_i is ready to submit its message m_i to the secret bulletin board, it generates a transcript $(\mathbf{U}_i, \mathbf{C}_i, \pi_i)$, which encrypts a random polynomial $\boldsymbol{\omega}_i \in \mathbb{Z}_q[x]$ of degree at most L. Party P_i then computes $\chi_i \leftarrow H(\boldsymbol{\omega}_i(0)) \oplus m_i$, and submits the message $M_i := (\mathbf{U}_i, \mathbf{C}_i, \pi_i, \chi_i)$ to the public bulletin board *PBB*. Recall that a public bulletin board applies an application-specific validity test — in this application, the validity test checks that M_i has the correct syntactic form and that π_i is a valid proof.

Eventually, *PBB* sends $(\mathcal{K}, \{M_k\}_{k \in \mathcal{K}})$, to all parties. Upon receiving this, each party P_i decrypts the transcripts in $\{M_k\}_{k \in \mathcal{K}}$ to obtain the values $\boldsymbol{\omega}_k(i)$ for $k \in \mathcal{K}$, and for each such k, it sends the message $(k, \boldsymbol{\omega}_k(i))$ to all parties. Then party P_i does the following for each $k \in \mathcal{K}$:

- wait for $L+1$ messages of the form (k, α_{kj}), where $g^{\alpha_{kj}} = \mathbf{U}_k^{(j)}$ (so that $\alpha_{kj} = \boldsymbol{\omega}_k(j)$);
- interpolate to obtain $\boldsymbol{\omega}_k(0)$;
- compute $m_k \leftarrow H(\boldsymbol{\omega}_k(0)) \oplus \chi_k$.

Finally, party P_i outputs $(\mathcal{K}, \{m_i\}_{k \in \mathcal{K}})$.

One can show that this protocol is secure in a simulation-based sense, analogous to Definition 22.16, assuming the discrete logarithm problem in $\mathbb{G}$ is hard, and modeling H as a random oracle. In the real world, the adversary interacts with with *PBB* and the random oracle H, while in the ideal world, the simulator interacts with *SBB*. We will not go into any further details here. We also note that this protocol will inherit the liveness properties of whatever Byzantine Agreement protocol is used to implement *PBB*, in the following sense: if $(\mathcal{K}, \{M_k\}_{k \in \mathcal{K}})$ is delivered to all honest parties, and all of the messages $(k, \boldsymbol{\omega}_k(i))$ that they subsequently generate are also delivered to all honest parties, then all honest parties will produce an output.

Using *PBB* instead of *SBB*. It is natural to ask, why not just directly use a public bulletin board *PBB* in our DKG protocol, rather than a secret bulletin board *SBB*? This would clearly be more efficient, but is it secure? The short answer is, "it depends on the application". To give a longer answer, let us first examine what additional advantage an adversary would have if we used *PBB* directly in our DKG protocol. In our DKG protocol in Section 22.4.2.3, each honest party P_i submits a transcript $(\mathbf{U}_i, \mathbf{C}_i, \pi_i)$ to the bulletin board. The adversary may also submit a transcript $(\mathbf{U}_\ell, \mathbf{C}_\ell, \pi_\ell)$ to the bulletin board on behalf of each corrupt party P_ℓ, and then choose which of these honest and corrupt transcript will be included in the set of $L+1$ transcripts output by the bulletin board. Since the bulletin board is public, the adversary may choose the corrupt transcripts and the output set based on the honest transcripts. Because of this, the adversary can bias the public key — it will no longer be a random element of $\mathbb{G}$. For example, the adversary could pick a set $\widetilde{\mathcal{K}}$ of L honest transcripts, and then generate a single corrupt transcript $(\mathbf{U}_\ell, \mathbf{C}_\ell, \pi_\ell)$ in some arbitrary way that depends on the transcripts in $\widetilde{\mathcal{K}}$, so that the resulting value

$$\mathbf{U} := \mathbf{U}_\ell \cdot \prod_{k \in \widetilde{\mathcal{K}}} \mathbf{U}_k,$$

and in particular, the resulting public key $\mathbf{U}^{(0)}$, is biased in some particular way that could make it easier to break a threshold cryptosystem.

Despite the ability of an adversary to bias the public key in this way, one can prove that any non-threshold cryptosystem with secret key $\alpha \in \mathbb{Z}_q$ and public key $u := g^\alpha \in \mathbb{Z}_q$ can be securely deployed as a secure threshold scheme using this simplified DKG protocol with a public bulletin board, provided the non-threshold cryptosystem remains secure under **tweaked key generation**, by which we mean the following *interactive* key generation procedure:

- The challenger generates the **initial secret key** $\tilde{\alpha} \in \mathbb{Z}_q$ at random and gives the **initial public key** $\tilde{u} := g^{\tilde{\alpha}} \in \mathbb{G}$ to the adversary.
- The adversary gives to the challenger $\mu \in \mathbb{Z}_q^*$ and $\nu \in \mathbb{Z}_q$.
- The challenger then computes the **tweaked secret key** $\alpha := \mu\tilde{\alpha}+\nu$ and the **tweaked public key** $u := g^{\alpha} \in \mathbb{G}$.

The rest of the attack game defining the security of the non-threshold cryptosystem is then executed as usual, except that we use the tweaked secret and public keys in place of the normal secret and public keys.

One can show that the BLS signature scheme and the GS encryption scheme are secure with respect to tweaked key generation, and hence could be securely deployed as threshold schemes using the simpler DKG protocol with a public bulletin board. In constrast, it is not hard to show that the ECDSA signature scheme (see Section 19.3) is insecure with respect to tweaked key generation.

22.5 Beyond threshold: monotone access structures

So far in this chapter we looked at *threshold* secret sharing systems where a secret is shared among N parties, and can be reconstructed by any t of them. The secret is hidden from any subset that is smaller than t. We used threshold secret sharing to distribute the ability to sign and decrypt without ever reconstructing the secret key in a single location.

Simple threshold access to a secret is not sufficient in many real world settings. For example, a bank may want to share a secret α among two groups of employees $\mathcal{S}_1$ and $\mathcal{S}_2$, so that at least t_1 people from $\mathcal{S}_1$ and at least t_2 people from $\mathcal{S}_2$ are needed to reconstruct the secret. Otherwise, the secret α should remain hidden.

This simple example shows the need for more flexible access structures beyond a simple threshold. Consider a total of N parties, denoted $\mathcal{S} = \{1, \ldots, N\}$. An *access structure* is an explicit collection of subsets of $\mathcal{S}$ that are allowed to reconstruct the secret. We will only consider access structures that have the following natural property: if a set $\mathcal{J} \subseteq \mathcal{S}$ can reconstruct the secret, then so can any superset of $\mathcal{J}$. An access structure that has this property is said to be *monotone*.

Definition 22.17. *Let $\mathcal{S}$ be a finite set, and let $\mathbb{A} \subseteq 2^{\mathcal{S}}$ be a collection of subsets of $\mathcal{S}$. We say that $\mathbb{A}$ is* ***monotone*** *if $\mathcal{J} \in \mathbb{A}$ and $\mathcal{J} \subseteq \mathcal{J}' \subseteq \mathcal{S}$ implies that $\mathcal{J}' \in \mathbb{A}$.*

Definition 22.18. *A pair $(\mathcal{S}, \mathbb{A})$ is called an* ***access structure*** *if $\mathcal{S}$ is finite and $\mathbb{A} \subseteq 2^{\mathcal{S}}$ is monotone. Subsets of $\mathcal{S}$ in $\mathbb{A}$ are called* ***authorized*** *and subsets not in $\mathbb{A}$ are called* ***unauthorized****.*

For example, let $\mathbb{A}_t$ be the collection of all subsets of $\mathcal{S}$ containing t or more elements. Then $(\mathcal{S}, \mathbb{A}_t)$ is an access structure that implements a t threshold access: all subsets of $\mathcal{S}$ containing t or more elements are authorized, and all subsets containing fewer than t elements are unauthorized.

Our definition of threshold secret sharing (Definition 22.1) easily generalizes to arbitrary access structures.

Definition 22.19. *Let $(\mathcal{S}, \mathbb{A})$ be an access structure. A* ***secret sharing scheme over Z that implements*** *$(\mathcal{S}, \mathbb{A})$ is a pair of efficient algorithms (G, C) so that:*

- *G is a probabilistic* ***sharing algorithm*** *that is invoked as $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N) \xleftarrow{\text{R}} G(\alpha)$, for $\alpha \in Z$, to generate a secret sharing of α.*

- *C is a deterministic* ***combining algorithm*** *that is invoked as $\alpha \leftarrow C(\mathcal{J}, \{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}})$, where $\mathcal{J} \in \mathbb{A}$, to recover α. The algorithm outputs an element of Z.*

- **Correctness:** *we require that for every $\alpha \in Z$, for every possible output $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N)$ of $G(\alpha)$, and every set $\mathcal{J} \in \mathbb{A}$ we have*

$$C(\mathcal{J}, \{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}}) = \alpha.$$

The security definition generalizes too. It says that an unauthorized set cannot distinguish one secret from another from the shares that they have.

Definition 22.20. *A secret sharing scheme (G, C) over Z that implements $(\mathcal{S}, \mathbb{A})$ is* ***secure*** *if for every $\alpha, \alpha' \in Z$, and every unauthorized subset $\mathcal{J} \notin \mathbb{A}$, the distribution $G(\alpha)[\mathcal{J}]$ is identical to the distribution $G(\alpha')[\mathcal{J}]$.*

To exercise the definition, let us show that if (G, C) over Z implements an access structure $(\mathcal{S}, \mathbb{A})$, then the set $\mathbb{A}$ is efficiently recognizable: there is an efficient randomized algorithm that takes a set $\mathcal{J} \subseteq \mathcal{S}$ as input, and decides if $\mathcal{J} \in \mathbb{A}$. On input $\mathcal{J}$, the decision algorithm does:

- choose a random $\alpha \xleftarrow{R} Z$,

- compute $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N) \xleftarrow{R} G(\alpha)$ and $\alpha' \leftarrow C(\mathcal{J},\ \{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}})$,

- output accept if $\alpha = \alpha'$ and output reject otherwise.

Correctness implies that when $\mathcal{J} \in \mathbb{A}$, the algorithm always outputs accept. Security implies that when $\mathcal{J} \notin \mathbb{A}$, the algorithm outputs accept with probability $1/|Z|$. Hence, assuming $|Z| \geq 2$, this efficient randomized procedure decides if $\mathcal{J}$ is in $\mathbb{A}$. If needed, it can be repeated several times to reduce the error probability.

This observation shows that if an access structure $(\mathcal{S}, \mathbb{A})$ is so complicated that $\mathbb{A}$ cannot be efficiently recognized, then there no efficient secret sharing scheme that implements it.

22.5.1 The generic secret sharing construction

Every access structure $(\mathcal{S}, \mathbb{A})$ can be securely implemented by a secret sharing scheme (G, C) over $\mathbb{Z}_q$, where the length of each share output by G is linear in the size of $\mathbb{A}$. To explain how the scheme works, let us first look at a simple example.

Example 22.1. Let $\mathcal{S} := \{1, 2, 3, 4\}$ and $\mathbb{A} := \big\{\{1, 2, 3\},\ \{1, 3, 4\},\ \mathcal{S}\big\}$. One can verify that $\mathbb{A}$ is monotone. For $\alpha \in \mathbb{Z}_q$, algorithm $G(\alpha)$ generates an independent secret sharing of α for each of the 3-element sets in $\mathbb{A}$:

- For the set $\{1, 2, 3\}$ it chooses random $\rho_1^{(1)}, \rho_2^{(1)}, \rho_3^{(1)}$ in $\mathbb{Z}_q$ such that their sum is α.

- For the set $\{1, 3, 4\}$ it chooses random $\rho_1^{(2)}, \rho_3^{(2)}, \rho_4^{(2)}$ in $\mathbb{Z}_q$ such that their sum is α.

The algorithm outputs the shares

$$\boldsymbol{\alpha}_1 := (\rho_1^{(1)}, \rho_1^{(2)}), \qquad \boldsymbol{\alpha}_2 := \rho_2^{(1)}, \qquad \boldsymbol{\alpha}_3 := (\rho_3^{(1)}, \rho_3^{(2)}), \qquad \boldsymbol{\alpha}_4 := \rho_4^{(2)}.$$

The reader can verify that all three authorized sets can reconstruct α, while unauthorized sets learn nothing about α. □

Construction 22.1 (generic construction). Let $(\mathcal{S}, \mathbb{A})$ be an access structure. For $i = 1, \ldots, N$ let $\mathbb{A}^{(i)}$ be the set of all authorized sets that contain i. That is, $\mathbb{A}^{(i)} = \{\mathcal{J} \in \mathbb{A} : i \in \mathcal{J}\}$. The generic construction works as follows:

- $G(\alpha)$: for every authorized set $\mathcal{J} = \{j_1, \ldots, j_\ell\}$ in $\mathbb{A}$ choose random $\rho_{j_1}^{(\mathcal{J})}, \ldots, \rho_{j_\ell}^{(\mathcal{J})}$ in $\mathbb{Z}_q$ such that their sum is equal to α. For example, choose the first $\ell - 1$ elements at random in $\mathbb{Z}_q$ and set the last element to be α minus the sum of the first $\ell - 1$ elements. Then, for $i = 1, \ldots, N$ the share $\boldsymbol{\alpha}_i$ is
$$\boldsymbol{\alpha}_i := \left\{\rho_i^{(\mathcal{J})}\right\}_{\mathcal{J} \in \mathbb{A}^{(i)}}$$
This $\boldsymbol{\alpha}_i$ contains one element in $\mathbb{Z}_q$ for every authorized set that contains i. In particular, $\boldsymbol{\alpha}_i$ contains $|\mathbb{A}^{(i)}|$ elements. Output the list of shares $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N)$.

- $C(\mathcal{J}, \{\boldsymbol{\alpha}_j\}_{j \in \mathcal{J}})$: output $\sum_{j \in \mathcal{J}} \rho_j^{(\mathcal{J})} \in \mathbb{Z}_q$.
In words, the algorithm picks out one element in $\mathbb{Z}_q$ from each of the provided shares, and outputs their sum.

We leave it to the reader to verify that the scheme is correct and secure. □

There are many ways to optimize this scheme to obtain somewhat smaller shares. An example is given in Exercise 2.20.

This generic construction works well when the number of authorized sets is small, but is unworkable as soon as the number of authorized sets is even mildly large. Even for a 10-out-of-20 threshold access structure this generic construction requires each share to contain over a hundred thousand elements in $\mathbb{Z}_q$. More generally, for an N-out-of-$2N$ access structure, this scheme outputs shares whose size is exponential in N. In contrast, Shamir's secret sharing scheme from Section 22.1.1 implements an N-out-of-$2N$ threshold access structure using much shorter shares: every share contains a single element in $\mathbb{Z}_q$ no matter how big N is.

22.5.2 Linear secret sharing schemes and monotone span programs

Recall that Shamir's secret sharing scheme $(G_{\text{sh}}, C_{\text{sh}})$ implements a threshold access structure. For $t \leq N < q$, a t-out-of-N sharing of $\alpha \in \mathbb{Z}_q$ is a set of shares $\alpha_1, \ldots, \alpha_N$ in $\mathbb{Z}_q$ generated as follows: choose a random polynomial $\boldsymbol{\omega}$ in $\mathbb{Z}_q[x]$ of degree less than t such that $\boldsymbol{\omega}(0) = \alpha$, and set $\alpha_i := \boldsymbol{\omega}(i)$ for $i = 1, \ldots, N$.

Shamir's secret sharing scheme has three important properties that makes it especially useful for constructing secure threshold signing and threshold decryption schemes.

Linearity of the secret: For every subset $\mathcal{J} \subseteq \{1, \ldots, N\}$ of size t, the secret α can be expressed as a fixed linear combination (depending only on $\mathcal{J}$) of the shares α_j for $j \in \mathcal{J}$.

Independence of the shares: Consider a subset $\mathcal{L} \subseteq \{1, \ldots, N\}$ of size $t - 1$. Then for every secret α, the $t - 1$ random variables α_ℓ for $\ell \in \mathcal{L}$ are uniformly and independently distributed over $\mathbb{Z}_q$.

Linearity of the shares: Consider a subset $\mathcal{L} \subseteq \{1, \ldots, N\}$ of size $t - 1$. For each $i \in \{1, \ldots, N\}$, the share α_i can be expressed as a fixed linear combination (depending only on $\mathcal{L}$ and i) of α and α_ℓ for $\ell \in \mathcal{L}$.

The first property, linearity of the secret, follows directly from Lemma 22.1 (with $j^* = 0$). It is used to combine signature shares into a full signature, and combine decryption shares into a full decryption. For example, in the threshold BLS signature scheme, linearity of the secret enables the combiner to compute a signature $\sigma := H(m)^\alpha$ on a message m from t signature shares $\{\sigma'_j := H(m)^{\alpha_j}\}_{j\in\mathcal{J}}$ via "interpolation in the exponent". Indeed, by linearity of the secret, there are efficiently computable scalars $\{\lambda_j\}_{j\in\mathcal{J}}$ such that $\sigma = \prod_{j\in\mathcal{J}}(\sigma'_j)^{\lambda_j}$.

The second property, independence the shares, was proved in Theorem 22.3. It is used to prove the security of threshold schemes. Specifically, in the reductions in the proofs of Theorems 22.4 and 22.7, we used this property to justify generating the adversary's secret shares α_ℓ for $\ell \in \mathcal{L}$ at random.

The third property, linearity of the shares, also follows directly from Lemma 22.1 (with $j^* = i$). It is also used to prove the security of threshold schemes. Specifically, in the reductions in the proofs of Theorems 22.4 and 22.7, we used this property to compute verification keys and signature/decryption shares via "interpolation in the exponent".

Note that this third property also follows directly from the first and second properties. Indeed, for $i \notin \mathcal{L}$, we can apply the first property with $\mathcal{J} := \mathcal{L} \cup \{i\}$, writing

$$\alpha = \lambda_i\alpha_i + \sum_{\ell\in\mathcal{L}} \lambda_\ell\alpha_\ell.$$

Moreover, by the second property, we must have $\lambda_i \neq 0$, as otherwise, the secret key would be completely determined by the $t-1$ shares α_ℓ for $\ell \in \mathcal{L}$. This allows us to then express α_i as a linear combination of α and α_ℓ for $\ell \in \mathcal{L}$.

22.5.2.1 Monotone span programs

In this section we abstract the three properties of Shamir secret sharing by defining the concept of a *linear* secret sharing scheme. Linear secret sharing schemes are important because they can be used as a "drop in" replacement for Shamir secret sharing in all the constructions presented in this chapter. In particular, if an access structure $(\mathcal{S}, \mathbb{A})$ can be implemented efficiently by a linear secret sharing scheme, then all the constructions in this chapter for threshold signing and threshold decryption can be adapted to support the access structure $(\mathcal{S}, \mathbb{A})$. We will see an example momentarily.

We begin by defining a somewhat odd looking computation model called a *monotone span program.* As we will see, every access structure $(\mathcal{S}, \mathbb{A})$ that is recognized by an efficient monotone span program has an efficient linear secret sharing scheme.

Definition 22.21. *A* ***monotone span program*** *P is a tuple $\big(N, q, \{M_i\}_{i=1}^N\big)$, where N is a positive integer, q is a prime, and each M_i is a $d_i \times d$ matrix over $\mathbb{Z}_q$ for some $d, d_1, \ldots, d_N$.*

- *We say that the program P* ***accepts*** *a set $\mathcal{J} \subseteq \{1, \ldots, N\}$ if and only if the row vector $\boldsymbol{e}_1 := (1, 0, 0, \ldots, 0) \in \mathbb{Z}_q^d$ is in the linear span of the rows of the matrices $\{M_i\}_{i\in\mathcal{J}}$.*
- *Let $\mathbb{L}(P)$ be the set of subsets of $\mathcal{S} := \{1, \ldots, N\}$ that are accepted by P. We say that P* ***recognizes*** *the access structure $(\mathcal{S}, \mathbb{A})$ if $\mathbb{L}(P) = \mathbb{A}$.*

The set $\mathbb{L}(P)$ is clearly monotone: if $\boldsymbol{e}_1$ is in the row span of $\{M_i\}_{i\in\mathcal{J}}$ then it is also in the row span of $\{M_i\}_{i\in\mathcal{J}'}$ for every superset $\mathcal{J}'$ of $\mathcal{J}$.

Example 22.2. Let's show that a t-out-of-N threshold access structure $(\mathcal{S}, \mathbb{A}_t)$ can be recognized by a monotone span program $P = \big(N, q, \{M_i\}_{i=1}^N\big)$, where $q > N$. For $i = 1, \ldots, N$, the matrix M_i is simply

$$M_i := (1, i, i^2, \ldots, i^{t-1}) \in \mathbb{Z}_q^{1\times t}.$$

The $N \times t$ matrix formed by the rows of $M_1, \ldots, M_N$ is called a **Vandermonde matrix**. A property of a Vandermonde matrix is that every t rows are linearly independent. Therefore, for every set $\mathcal{J}$ of size at least t, the rows of $\{M_i\}_{i\in\mathcal{J}}$ span all of $\mathbb{Z}_q^t$, and in particular, their span contains $\boldsymbol{e}_1$. Hence $\mathcal{J} \in \mathbb{L}(P)$. Moreover, if $\mathcal{L}$ is a set with fewer than t elements, then it is a simple exercise to show that $\boldsymbol{e}_1$ is not in the span of the rows of $\{M_i\}_{i\in\mathcal{L}}$. Therefore, $\mathbb{L}(P) = \mathbb{A}_t$, as required. □

Example 22.3. Consider again the access structure $(\mathcal{S}, \mathbb{A})$ where $\mathcal{S} = \{1, 2, 3, 4\}$ and $\mathbb{A} = \big\{\{1, 2, 3\},\ \{1, 3, 4\},\ \mathcal{S}\big\}$ from Example 22.1. Let $P = (4, q, \{M_i\}_{i=1}^4)$ be the monotone span program where $q \geq 2$ and the matrices M_1, M_2, M_3, M_4 are

$$M_1 := \left|\begin{array}{c|cc|cc} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{array}\right| \in \mathbb{Z}_q^{2\times 5}$$

$$M_2 := \left|\begin{array}{c|cc|cc} 0 & 0 & 1 & 0 & 0 \end{array}\right| \in \mathbb{Z}_q^{1\times 5}$$

$$M_3 := \left|\begin{array}{c|cc|cc} 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{array}\right| \in \mathbb{Z}_q^{2\times 5}$$

$$M_4 := \left|\begin{array}{c|cc|cc} 0 & 0 & 0 & -1 & -1 \end{array}\right| \in \mathbb{Z}_q^{1\times 5}$$

The set $\{1, 2, 3\}$ is accepted by the program P because the first rows of M_1, M_2, M_3 sum to $\boldsymbol{e}_1$. The set $\{1, 3, 4\}$ is accepted by P because bottom rows of M_1, M_3, M_4 sum to $\boldsymbol{e}_1$. No other subset of $\mathcal{S}$ is accepted by P other than the set $\mathcal{S}$ itself. Hence $\mathbb{L}(P) = \mathbb{A}$ as required.

This example can be generalized to show that for every access structure $(\mathcal{S}, \mathbb{A})$ there is a monotone span program $(N, q, \{M_i\}_{i=1}^N)$ that recognizes $\mathbb{A}$, where for $i = 1, \ldots, N$ the matrix $M_i \in \mathbb{Z}_q^{d_i \times d}$ for some $d \leq N \cdot |\mathbb{A}|$ and $d_i \leq |\mathbb{A}^{(i)}|$. □

22.5.2.2 The secret sharing scheme derived from a monotone span program

Let $(\mathcal{S}, \mathbb{A})$ be an access structure, where $|\mathcal{S}| = N$. Suppose $\mathbb{A}$ is recognized by a monotone span program P. Then the following secret sharing scheme implements $(\mathcal{S}, \mathbb{A})$.

Construction 22.2. The secret sharing scheme (G, C) derived from $P = \big(N, q, \{M_i\}_{i=1}^N\big)$, where M_i is a $d_i \times d$ matrix over $\mathbb{Z}_q$, works as follows:

- $G(\alpha)$, for $\alpha \in \mathbb{Z}_q$, works as follows:
  - choose random $\rho_2, \ldots, \rho_d$ in $\mathbb{Z}_q$ and let $\boldsymbol{r}$ be the column vector $\boldsymbol{r} := (\alpha, \rho_2, \ldots, \rho_d) \in \mathbb{Z}_q^d$.
  - compute the column vector $\boldsymbol{\alpha}_i \leftarrow M_i \cdot \boldsymbol{r} \in \mathbb{Z}_q^{d_i}$ for $i = 1, \ldots, N$.
  - output the shares $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N)$.
- $C(\mathcal{J},\ \{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}})$, for $\mathcal{J} \in \mathbb{A}$, works as follows:
  - Let $d' := \sum_{i\in\mathcal{J}} d_i$. Concatenate all the shares $\{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}}$ into one column vector $\boldsymbol{\alpha} \in \mathbb{Z}_q^{d'}$. Similarly, concatenate all the matrices $\{M_j\}_{j\in\mathcal{J}}$ into one $d' \times d$ matrix M over $\mathbb{Z}_q$.

– Find a row vector $\boldsymbol{w} \in \mathbb{Z}_q^{d'}$ such that $\boldsymbol{w} \cdot M = \boldsymbol{e}_1$. We know that $\boldsymbol{w}$ exists because $\mathcal{J} \in \mathbb{A} = \mathbb{L}(P)$, and therefore $\boldsymbol{e}_1$ is in the row span of M.

– Output the inner product $\boldsymbol{w} \cdot \boldsymbol{\alpha} \in \mathbb{Z}_q$.

This completes the description of the scheme. Note that $d_1, \dots, d_N$ affect the size of the shares, where as d affects the running time of algorithms G and C. □

Let us show that the scheme is correct. Let $\boldsymbol{r}$ be the vector constructed by algorithm G, and let $M, \boldsymbol{\alpha}, \boldsymbol{w}$ be the quantities constructed during an execution of $C(\mathcal{J},\ \{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}})$. Then:

$$\boldsymbol{w} \cdot \boldsymbol{\alpha} = \boldsymbol{w} \cdot (M \cdot \boldsymbol{r}) = \big(\boldsymbol{w} \cdot M\big) \cdot \boldsymbol{r} = \boldsymbol{e}_1 \cdot \boldsymbol{r} = \alpha.$$

Hence, $C(\mathcal{J},\ \{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}})$ outputs α, as required.

As for security, we have:

Theorem 22.13. *The secret sharing scheme (G, C) securely implements the access structure $(\mathcal{S}, \mathbb{A})$.*

We shall defer the proof of this theorem until a bit later (see the discussion following Theorem 22.14).

Definition 22.22. *Let P be a monotone span program. The secret sharing scheme (G, C) obtained from P via Construction 22.2 is called the* ***linear secret sharing scheme*** *derived from P.*

Let's see some familiar linear secret sharing schemes:

- The linear secret sharing scheme derived from the monotone span program in Example 22.2 is identical to Shamir's secret sharing scheme. To see the correspondence, observe that the vector $(\alpha, \rho_2, \dots, \rho_d)$ in Construction 22.2 plays the same role as the vector $(\alpha, \kappa_1, \dots, \kappa_{t-1})$ that is used to define the polynomial in Shamir's secret sharing scheme. Hence, Theorem 22.13, on the security of the general construction, is a generalization of Theorem 22.3, which proves security of Shamir secret sharing.

- The linear secret sharing scheme derived from the monotone span program in Example 22.3 is the same as the generic secret sharing scheme from Example 22.1.

The main application for linear secret sharing schemes is that they can be used as a "drop in" replacement for Shamir secret sharing in all the constructions presented in this chapter. In particular, if an access structure $(\mathcal{S}, \mathbb{A})$ can be implemented efficiently by a linear secret sharing scheme, then all the constructions in this chapter for threshold signing and threshold decryption can be adapted to support the access structure $(\mathcal{S}, \mathbb{A})$. We will see an example of this in Section 22.5.3 and also in Exercise 22.2.

22.5.2.3 Properties of linear secret sharing schemes

Recall that Shamir secret sharing satisfies three important properties described at the beginning of the section: (i) linearity of the secret, (ii) independence of the shares, and (iii) linearity of the shares. We show that a linear secret sharing scheme derived from a monotone span program has analogous properties. To generalize the second and third properties to a general access structure we need the following useful concept.

Definition 22.23. *Let $P = \big(N, q, \{M_i\}_{i=1}^N\big)$ and $P' = \big(N, q, \{M'_i\}_{i=1}^N\big)$ be two monotone span programs, and let (G, C) and (G', C') be the linear secret sharing schemes derived from P and P' respectively. We say that P and P' are **equivalent** if for all $\alpha \in \mathbb{Z}_q$ the distributions $G(\alpha)$ and $G'(\alpha)$ are identical.*

The definition says that P and P' are equivalent if the derived secret sharing schemes always generate the same distribution of shares. We will see examples of equivalence in Lemma 22.15 below.

Properties of linear secret sharing schemes. The next theorem shows that a linear secret sharing scheme has the same two properties as Shamir secret sharing.

Theorem 22.14. *Let $P = \big(N, q, \{M_i\}_{i=1}^N\big)$ be a monotone span program that recognizes the access structure $(\mathcal{S}, \mathbb{A})$, where $N = |\mathcal{S}|$ and where M_i is a $d_i \times d$ matrix for all $i = 1, \ldots, N$. Let (G, C) be the linear secret sharing scheme derived from P.*

*(a) **Linearity of the secret**: There is an efficient deterministic algorithm Lin that is invoked as Lin($\mathcal{J}$), where $\mathcal{J} \in \mathbb{A}$, and outputs vectors $\{\boldsymbol{\lambda}_j \in \mathbb{Z}_q^{d_j}\}_{j\in\mathcal{J}}$. For all $\alpha \in \mathbb{Z}_q$ and all shares $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N)$ output by $G(\alpha)$, we have*

$$\alpha = \sum_{j\in\mathcal{J}} \boldsymbol{\lambda}_j \cdot \boldsymbol{\alpha}_j.$$

Here, $\boldsymbol{\lambda}_j \cdot \boldsymbol{\alpha}_j$ is the dot product of the vectors $\boldsymbol{\lambda}_j$ and $\boldsymbol{\alpha}_j$.

*(b) **Independence and linearity of the shares**: For every unauthorized set $\mathcal{L} \subseteq \mathcal{S}$ there is an efficiently computable monotone span program $P_{\mathcal{L}} = \big(N, q, \{M'_i\}_{i=1}^N\big)$ that is (i) equivalent to P, and (ii) for each $\ell \in \mathcal{L}$ the left-most column of M'_ℓ is zero.*

To understand part (b) better, let (G', C') be the linear secret sharing scheme derived from $P_{\mathcal{L}} = \big(N, q, \{M'_i\}_{i=1}^N\big)$. Recall that for a secret $\alpha \in \mathbb{Z}_q$, algorithm $G'(\alpha)$ generates shares by computing

$$\boldsymbol{\alpha}_i := M'_i \cdot \boldsymbol{r} \in \mathbb{Z}_q^{d_i} \quad \text{for } i = 1, \ldots, N,$$

where $\boldsymbol{r}$ is the column vector $\boldsymbol{r} := (\alpha, \rho_2, \ldots, \rho_d)$ for $\rho_2, \ldots, \rho_d \xleftarrow{\text{R}} \mathbb{Z}_q$. Because P and $P_{\mathcal{L}}$ are equivalent, we know that the resulting shares are sampled from the same distribution as $G(\alpha)$. Moreover, since the left-most column of M'_ℓ is zero for $\ell \in \mathcal{L}$, the share $\boldsymbol{\alpha}_\ell := M'_\ell \cdot \boldsymbol{r}$ does not depend on the secret α; it only depends on $\rho_2, \ldots, \rho_d$. Hence, for an unauthorized set $\mathcal{L}$, all the shares $\{\boldsymbol{\alpha}_\ell\}_{\ell\in\mathcal{L}}$ can be sampled from a distribution that does not depend at all on the secret $\alpha \in \mathbb{Z}_q$ — these shares only depend on $\rho_2, \ldots, \rho_d$. This corresponds to the "independence of the shares" property for Shamir secret sharing, and clearly implies Theorem 22.13. Moreover, every share $\boldsymbol{\alpha}_i$ can be expressed linearly in terms of α and $\rho_2, \ldots, \rho_d$. This corresponds to the "linearity of the shares" property for Shamir secret sharing.

The proof of Theorem 22.14 part (a) is immediate from the definition of algorithm $C(\mathcal{J},\ \{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}})$. The algorithm reconstructs the secret α by computing a linear combination of the shares $\{\boldsymbol{\alpha}_j\}_{j\in\mathcal{J}}$. To illustrate this concretely, consider again Example 22.1, and observe that for the authorized set $\mathcal{J} := \{1, 2, 3\}$ algorithm *Lin*($\mathcal{J}$) would output the vectors

$$\boldsymbol{\lambda}_1 := (1, 0), \quad \boldsymbol{\lambda}_2 := (1), \quad \boldsymbol{\lambda}_3 := (1, 0).$$

To prove Theorem 22.14 part (b), we first state the following simple lemma, which gives a sufficient condition for when two monotone span programs are equivalent.

Lemma 22.15. *Let $P = \big(N, q, \{M_i\}_{i=1}^N\big)$ be a monotone span program where each M_i is a $d_i \times d$ matrix. Let R be an invertible $d \times d$ matrix over $\mathbb{Z}_q$ where the top most row of R is $\boldsymbol{e}_1$. Let P' be the monotone span program $P' = \big(N, q, \{M_i R\}_{i=1}^N\big)$. Then P and P' are equivalent.*

Proof. Fix some secret $\alpha \in \mathbb{Z}_q$ and shares $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N$. For $(\rho_2, \ldots, \rho_d) \in \mathbb{Z}_q^{d-1}$ let $\boldsymbol{r}$ be the column vector $\boldsymbol{r} := (\alpha, \rho_2, \ldots, \rho_d)$. For $(\rho'_2, \ldots, \rho'_d) \in \mathbb{Z}_q^{d-1}$ let $\boldsymbol{r}'$ be the column vector $\boldsymbol{r}' := (\alpha, \rho'_2, \ldots, \rho'_d)$.

The proof follows from the fact that there is a one-to-one correspondence between tuples $(\rho_2, \ldots, \rho_d) \in \mathbb{Z}_q^{d-1}$ such that $\boldsymbol{\alpha}_i = M_i \cdot \boldsymbol{r}$ for all $i = 1, \ldots, N$, and tuples $(\rho'_2, \ldots, \rho'_d) \in \mathbb{Z}_q^{d-1}$ such that $\boldsymbol{\alpha}_i = (M_i R) \cdot \boldsymbol{r}'$ for all $i = 1, \ldots, N$. The correspondence is defined by $\boldsymbol{r} = R \cdot \boldsymbol{r}'$. □

Proof of Theorem 22.14 part (b). Lemma 22.15 reduces the proof to a linear algebra question. Let $\mathcal{L}$ be an unauthorized set. It suffices to show that there is a $d \times d$ matrix R such that (i) R is invertible, (ii) the top most row of R is $\boldsymbol{e}_1$, and (iii) for all $\ell \in \mathcal{L}$ the left-most column of $M_\ell \cdot R$ is zero. Then $P_{\mathcal{L}} := \big(N, q, \{M_i R\}_{i=1}^N\big)$ is the required monotone span program.

First, consider the subspace V of $\mathbb{Z}_q^d$ spanned by the rows of all of the matrices M_ℓ for $\ell \in \mathcal{L}$. We can choose an ordered basis for V, denoted $\boldsymbol{v}_{k+1}, \ldots, \boldsymbol{v}_d$. We can extend this to an ordered basis for $\mathbb{Z}_q^d$, denoted $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k, \boldsymbol{v}_{k+1}, \ldots, \boldsymbol{v}_d$. By assumption, $\boldsymbol{e}_1 \notin V$, and so without loss of generality, we may assume $\boldsymbol{v}_1 = \boldsymbol{e}_1$. Define R to be a matrix that represents a change of basis from the standard basis $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d$ to the basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d$. More precisely, R is the $d \times d$ matrix that maps the row vector $\boldsymbol{a} \in \mathbb{Z}_q^d$ onto the row vector $(b_1, \ldots, b_d) := \boldsymbol{a}R \in \mathbb{Z}_q^d$ whose entries are the coordinates of $\boldsymbol{a}$ on the basis $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d$, namely $\boldsymbol{a} = b_1\boldsymbol{v}_1 + \cdots + b_d\boldsymbol{v}_d$. We claim that R satisfies properties (i)–(iii) above. Property (i) is clear, since R represents a change of basis. Property (ii) is clear, since $\boldsymbol{v}_1 = \boldsymbol{e}_1$, and hence $\boldsymbol{e}_1 R = \boldsymbol{e}_1$. Property (iii) is clear, since for each $\ell \in \mathcal{L}$, each row $\boldsymbol{a}$ of M_ℓ lies in V, and hence the coordinate vector $\boldsymbol{a}R$ is only nonzero in positions $k+1, \ldots, d$. In particular, since $k \geq 1$, the left-most entry of $\boldsymbol{a}R$ is zero. □

Remark 22.15. A linear secret sharing scheme (G, C) obtained from Construction 22.2 has another important linearity property. If $(\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_N) \overset{\text{R}}{\leftarrow} G(\alpha)$ is a sharing of α, and $(\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_N) \overset{\text{R}}{\leftarrow} G(\beta)$ is a sharing of β, then $(\boldsymbol{\alpha}_1 + \boldsymbol{\beta}_1, \ldots, \boldsymbol{\alpha}_N + \boldsymbol{\beta}_N)$ is a sharing of $\alpha + \beta$. Indeed, if $\mathcal{J}$ is authorized, then $C(\mathcal{J},\ \{\boldsymbol{\alpha}_j + \boldsymbol{\beta}_j\}_{j \in \mathcal{J}})$ must output $\alpha + \beta$. This property is often used to "compute" on data that is secret shared among multiple parties. □

22.5.2.4 Linear secret sharing from monotone formulas

Finally, we show that every access structure defined by a monotone formula can be implemented by a linear secret sharing scheme. We will explain these terms momentarily. The key point is that if one can write down a monotone formula that captures the access structure that one has in mind, then it is straightforward to derive a linear secret sharing scheme that implements this access structure. This, in turn, lets us build signing and decryption schemes that support the access structure defined by the monotone formula, as long as the formula size is poly-bounded.

Monotone formulas. We first briefly review what is a monotone formula. A **t-out-of-N threshold gate** is a function $g_t : \{0,1\}^N \to \{0,1\}$ that takes N boolean inputs, and outputs '1' if t or more the inputs is equal to '1'. Note that a 1-out-of-N threshold gate computes the logical OR of its N inputs. Similarly, an N-out-of-N threshold gate computes the logical AND of its N inputs.

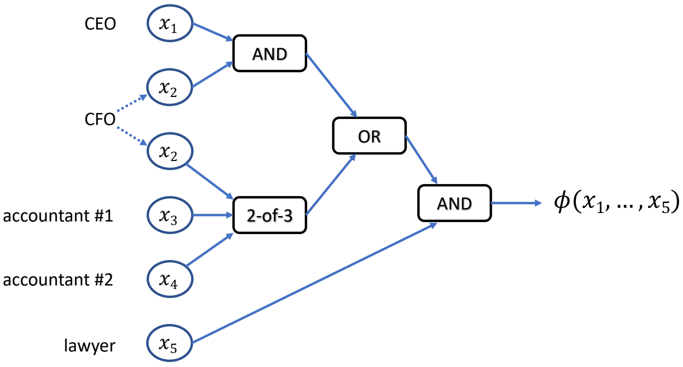


Figure 22.5: An example monotone formula ϕ that defines an access structure. To reconstruct the secret one needs a lawyer, and (both the CEO and CFO) or (two of CFO and two accountants).

A **monotone formula** ϕ with N inputs is a directed acyclic graph, a DAG, where every node has at most one outgoing edge. Every input node (i.e., a node with in-degree zero) is labeled with a variable x_i for some $i \in \{1,\ldots,N\}$. Every internal node that has in-degree k is labeled as t-out-of-k, for some $0 < t \leq k$, to indicate that it is a t-out-of-k threshold gate. Finally, there is a single output node (i.e., a node with out-degree zero). An example monotone formula is shown in Fig. 22.5.

A monotone formula ϕ with N inputs computes a function $F_\phi : \{0,1\}^N \to \{0,1\}$. For an input $x \in \{0,1\}^N$ one evaluates all the threshold gates one by one from the inputs to the output in a topological order. The output of the final gate is defined to be the value of $F_\phi(x)$.

A monotone formula ϕ with N inputs defines an access structure $(\mathcal{S}, \mathbb{A}_\phi)$ where $|\mathcal{S}| = N$. To see how, we say that a set $\mathcal{J} \subseteq \mathcal{S}$ is authorized (i.e., namely $\mathcal{J} \in \mathbb{A}_\phi$), if and only if the characteristic vector of $\mathcal{J}$, denoted $\chi(\mathcal{J}) \in \{0,1\}^N$, satisfies $F_\phi(\chi(\mathcal{J})) = 1$. Recall that for $\mathcal{J} \subseteq \{1,\ldots,N\}$, the *characteristic vector* of $\mathcal{J}$ is a vector in $\{0,1\}^N$ that is '1' at exactly the positions that correspond to the elements of $\mathcal{J}$. For example, the characteristic vector of $\{2,4\} \subseteq \{1,2,3,4\}$ is 0101. The reader can verify that $\mathbb{A}_\phi$ is monotone.

The linear secret sharing scheme derived from a monotone formula. The following theorem is the main point of this section. For a monotone formula ϕ we write $|\phi|$ for the total number of input nodes to the formula. For example, the formula ϕ in Fig. 22.5 has five inputs $x_1,\ldots,x_5$, but $|\phi| = 6$ because x_2 is provided as an input twice. We write $|\phi|_i$ for the number of times that x_i appears as an input to ϕ. For the formula ϕ in Fig. 22.5 we have $|\phi|_2 = 2$.

Theorem 22.16. *Let $(\mathcal{S}, \mathbb{A}_\phi)$ be an access structure defined by a monotone formula ϕ with N inputs. Then for every prime $q > |\phi|$, there is a monotone span program $P_\phi = (N, q, \{M_i\}_{i=1}^N)$ that recognizes $(\mathcal{S}, \mathbb{A}_\phi)$ where the dimension of each matrix M_i is $d_i \times d$ where $d_i = |\phi|_i$ and $d \leq |\phi|$.*

The linear secret sharing scheme (G_ϕ, C_ϕ) derived from P_ϕ generates shares $\boldsymbol{\alpha}_1,\ldots,\boldsymbol{\alpha}_N$, where each $\boldsymbol{\alpha}_i$ is a vector over $\mathbb{Z}_q$ of length $|\phi|_i$.

The proof of Theorem 22.16 is by induction on the number gates in ϕ. We will not prove the theorem here, but instead give an example for how algorithm $G_\phi(\alpha)$ works for the monotone formula in Fig. 22.5. Algorithm $G_\phi(\alpha)$ processes one gate at a time, starting from the output gate, and working its way towards the inputs. The output gate in Fig. 22.5 is an AND gate, so the algorithm chooses random $\tau_1, \tau_2 \xleftarrow{R} \mathbb{Z}_q$ such that $\tau_1 + \tau_2 = \alpha$. Both τ_1 and τ_2 are needed to reconstruct α, so this properly implements an AND gate. Next, it recursively processes each gate. The next gate is an OR gate which the algorithm processes by setting $\tau_3 := \tau_1$ and $\tau_4 := \tau_1$. Either one of τ_3 or τ_4 can reconstruct τ_1, so this properly implements an OR gate. The next gate is an AND gate so the algorithm chooses $\tau_5, \tau_6 \xleftarrow{R} \mathbb{Z}_q$ such that $\tau_5 + \tau_6 = \tau_3$. Finally, the 2-out-of-3 gate is processed by calling $(\tau_7, \tau_8, \tau_9) \xleftarrow{R} G_{\text{sh}}(3, 2, \tau_4)$. The final shares given to the five parties are:

$$\boldsymbol{\alpha}_1 := \tau_5, \quad \boldsymbol{\alpha}_2 := (\tau_6, \tau_7), \quad \boldsymbol{\alpha}_3 := \tau_8, \quad \boldsymbol{\alpha}_4 := \tau_9, \quad \boldsymbol{\alpha}_5 := \tau_2. \tag{22.6}$$

The reconstruction algorithm works from the leaves to the output, and can reconstruct α whenever the subset of shares satisfies the formula ϕ. It is not too difficult to see how this procedure generalizes to any monotone formula.

We can express this procedure as a monotone span program $P = (5, q, \{M_i\}_{i=1}^5)$ where the matrices are

$$M_1 := \begin{vmatrix} 0 & 0 & 1 & 0 \end{vmatrix}$$

$$M_2 := \begin{vmatrix} 0 & 1 & -1 & 0 \\ 0 & 1 & 0 & 1 \end{vmatrix}$$

$$M_3 := \begin{vmatrix} 0 & 1 & 0 & 2 \end{vmatrix}$$

$$M_4 := \begin{vmatrix} 0 & 1 & 0 & 3 \end{vmatrix}$$

$$M_5 := \begin{vmatrix} 1 & -1 & 0 & 0 \end{vmatrix}$$

The derived share generation algorithm $G_\phi(\alpha)$ operates by computing $\boldsymbol{\alpha}_i \leftarrow M_i \cdot \boldsymbol{r}$ for $i = 1, \ldots, 5$, where $\boldsymbol{r} := (\alpha, \rho_2, \rho_3, \rho_4)$ is a column vector and $\rho_2, \rho_3, \rho_4 \xleftarrow{R} \mathbb{Z}_q$. This gives the shares

$$\boldsymbol{\alpha}_1 := \rho_3, \quad \boldsymbol{\alpha}_2 := (\rho_2 - \rho_3,\ \rho_2 + \rho_4), \quad \boldsymbol{\alpha}_3 := \rho_2 + 2\rho_4, \quad \boldsymbol{\alpha}_4 := \rho_2 + 3\rho_4, \quad \boldsymbol{\alpha}_5 := \alpha - \rho_2.$$

which is equivalent to (22.6).

22.5.3 A signature scheme from linear secret sharing

Now that we understand linear secret sharing schemes and their properties, let us generalize the BLS threshold signature scheme to work with any secure linear secret sharing scheme. The GS threshold decryption scheme can be generalized similarly.

Let $\mathbb{G}$ be a group of prime order q with generator $g \in \mathbb{G}$. Let $P = \big(N, q, \{M_i\}_{i=1}^N\big)$ be a monotone span program that recognizes an access structure $(\mathcal{S}, \mathbb{A})$. Let (G, C) be the linear secret sharing scheme derived from P. Then (G, C) is a secret sharing scheme over $\mathbb{Z}_q$ that implements $(\mathcal{S}, \mathbb{A})$, where $|\mathcal{S}| = N$. We will also need a hash function $H : \mathcal{M} \to \mathbb{G}$, and the BLS signature verification algorithm V_{BLS}.

Construction 22.3. The $\mathcal{S}_{\text{linBLS}} = (G', S', V', C')$ signature scheme for the access structure $(\mathcal{S}, \mathbb{A})$ works as follows.

- $G'() \to (pk, pkc, sk_1, \ldots, sk_N)$:

- choose a random $\alpha \xleftarrow{R} \mathbb{Z}_q$ and compute $pk \leftarrow g^{\alpha}$;
- run $G(\alpha)$ to obtain N shares $(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_N)$ where each $\boldsymbol{\alpha}_i \in \mathbb{Z}_q^{d_i}$ for $i = 1, \dots, N$;
- for $i = 1, \dots, N$ set $sk_i := \boldsymbol{\alpha}_i \in \mathbb{Z}_q^{d_i}$ and $pk_i \leftarrow g^{\boldsymbol{\alpha}_i} := \left(g^{\boldsymbol{\alpha}_{i,1}}, \dots, g^{\boldsymbol{\alpha}_{i,d_i}}\right) \in \mathbb{G}^{d_i}$;
- set $pkc := (pk_1, \dots, pk_N)$;
- output $(pk,\ pkc,\ sk_1, \dots, sk_N)$.

- $S'(sk_i, m) \to \sigma'_i$, where $sk_i = \boldsymbol{\alpha}_i \in \mathbb{Z}_q^{d_i}$, output the signature share

$$\boldsymbol{\sigma}'_i \leftarrow H(m)^{\boldsymbol{\alpha}_i} := \left(H(m)^{\boldsymbol{\alpha}_{i,1}}, \dots, H(m)^{\boldsymbol{\alpha}_{i,d_i}}\right) \in \mathbb{G}^{d_i}.$$

- $C'(pkc,\ m,\ \mathcal{J},\ \{\boldsymbol{\sigma}'_j\}_{j\in\mathcal{J}}) \to \sigma$, where $\mathcal{J} \in \mathbb{A}$:
  - *step 1:* verify that all the provided signature shares are valid: with $pkc := (pk_1, \dots, pk_N)$, let $\mathcal{J}^*$ be the set of all $j \in \mathcal{J}$ such that

$$V_{\text{BLS}}(pk_{j,u}, m, \boldsymbol{\sigma}'_{j,u}) = \mathsf{reject} \quad \text{for some } u = 1, \dots, d_j;$$

    if $\mathcal{J}^*$ is nonempty, output $\mathsf{blame}(\mathcal{J}^*)$ and abort;
  - *step 2:* compute the vectors $\{\boldsymbol{\lambda}_j \in \mathbb{Z}_q^{d_j}\}_{j\in\mathcal{J}} \leftarrow Lin(\mathcal{J})$, where Lin is the algorithm from Theorem 22.14(a);
  - *step 3:* output the signature

$$\sigma \leftarrow \prod_{j\in\mathcal{J}} \prod_{u=1}^{d_j} (\boldsymbol{\sigma}'_{j,u})^{\boldsymbol{\lambda}_{j,u}} \quad \in \mathbb{G}.$$

- $V'(pk, m, \sigma)$: output $V_{\text{BLS}}(pk, m, \sigma)$.

This completes the description of the signature scheme. □

To see that the scheme is correct, recall that for $\mathcal{J} \in \mathbb{A}$ the vectors $\{\boldsymbol{\lambda}_j \in \mathbb{Z}_q^{d_j}\}_{j\in\mathcal{J}} \leftarrow Lin(\mathcal{J})$ satisfy $\sum_{j\in\mathcal{J}} \boldsymbol{\alpha}_j \cdot \boldsymbol{\lambda}_j = \alpha$. Therefore, the signature σ output in step 3 of algorithm C' satisfies

$$\sigma = \prod_{j\in\mathcal{J}} \prod_{u=1}^{d_j} (\boldsymbol{\sigma}'_{j,u})^{\boldsymbol{\lambda}_{j,u}} = \prod_{j\in\mathcal{J}} \prod_{u=1}^{d_j} \left(H(m)^{\boldsymbol{\alpha}_{j,u}}\right)^{\boldsymbol{\lambda}_{j,u}} = \prod_{j\in\mathcal{J}} H(m)^{\boldsymbol{\alpha}_j \cdot \boldsymbol{\lambda}_j} = H(m)^{\alpha}$$

as required.

Security. To argue that the scheme is secure we first need to define signature security for a general access structure $(\mathcal{S}, \mathbb{A})$. We do so by generalizing Definition 22.4. Recall that Definition 22.4 allows the adversary to corrupt up to $t-1$ signing parties and learn their secret keys. When defining security for a general access structure, we allow the adversary to corrupt any unauthorized set of parties. Security requires that even with so many secret keys at its disposal, the adversary cannot forge a signature.

Attack Game 22.8 (signature security for an access structure). Let $\mathcal{S} = (G', S', V', C')$ be a signature scheme defined over $(\mathcal{M}, \Sigma)$ for an access structure $(\mathcal{S}, \mathbb{A})$, where $|\mathcal{S}| = N$. For a given adversary $\mathcal{A}$, we define the following attack game.

- *Setup:* the adversary sends an unauthorized subset $\mathcal{L} \subseteq \mathcal{S}$ to the challenger. The challenger runs
$$(pk, pkc, sk_1, \ldots, sk_N) \xleftarrow{\text{R}} G'(),$$
and sends pk, pkc, and $\{sk_\ell\}_{\ell \in \mathcal{L}}$ to $\mathcal{A}$.
- *Signing queries:* for $j = 1, 2, \ldots$, signing query j from $\mathcal{A}$ is a message $m_j \in \mathcal{M}$. Given m_j, the challenger computes all N signature shares $\sigma'_{j,i} \xleftarrow{\text{R}} S'(sk_i, m_j)$ for $i = 1, \ldots, N$. It sends $\big(\sigma'_{j,1}, \ldots, \sigma'_{j,N}\big)$ to the adversary.
- *Forgery:* eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, \sigma) \in \mathcal{M} \times \Sigma$.

We say that the adversary wins the game if the following two conditions hold:

- $V'(pk, m, \sigma) = \mathsf{accept}$, and
- m is new, namely, $m \notin \{m_1, m_2, \ldots\}$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$ denoted asSIG$\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 22.24 (secure signatures for an access structure). *We say that a signature scheme $\mathcal{S}$ for an access structure $(\mathcal{S}, \mathbb{A})$ is **secure** if for all efficient adversaries $\mathcal{A}$, the quantity* asSIG$\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ *is negligible.*

Next, we argue that the signature scheme $\mathcal{S}_{\text{linBLS}}$ in Construction 22.3 is secure. Security follows from the security of the BLS signature scheme, and is a direct generalization of the proof of Theorem 22.4.

Theorem 22.17. *Let $P = \big(N, q, \{M_i\}_{i=1}^N\big)$ be a monotone span program where each M_i is a $d_i \times d$ matrix and where all $d, d_1, \ldots, d_N$ are poly-bounded. Let $(\mathcal{S}, \mathbb{A})$ be the access structure recognized by P. If $\mathcal{S}_{\text{BLS}}$ is a secure signature scheme, then $\mathcal{S}_{\text{linBLS}}$ from Construction 22.3 is a secure signature scheme for $(\mathcal{S}, \mathbb{A})$.*

> *In particular, for every adversary $\mathcal{A}$ that attacks $\mathcal{S}_{\text{linBLS}}$ as in Attack Game 22.8, there exists an adversary $\mathcal{B}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, that attacks $\mathcal{S}_{\text{BLS}}$ as in Attack Game 13.1, such that*
> $$\text{asSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}_{\text{linBLS}}] = \text{SIG}\mathsf{adv}[\mathcal{B}, \mathcal{S}_{\text{BLS}}].$$

Proof sketch. Let us sketch the main idea for how $\mathcal{B}$ works. First, $\mathcal{B}$ is given a BLS public key $pk := g^\alpha$ for some unknown $\alpha \in \mathbb{Z}_q$.

Next, $\mathcal{A}$ outputs an unauthorized set $\mathcal{L} \subseteq \mathcal{S}$. Our $\mathcal{B}$ needs to respond with pk, pkc, and $\{sk_\ell\}_{\ell \in \mathcal{L}}$. Let $P_{\mathcal{L}} = \big(N, q, \{M'_i\}_{i=1}^N\big)$ be the monotone span program equivalent to P from Theorem 22.14(b). Recall that for all $\ell \in \mathcal{L}$ the left-most column of M'_ℓ is zero.

$\mathcal{B}$ chooses random $\rho_2, \ldots, \rho_d \xleftarrow{\text{R}} \mathbb{Z}_q$. Now let $\boldsymbol{r} := (\alpha, \rho_2, \ldots \rho_d) \in \mathbb{Z}_q^d$. Although $\mathcal{B}$ does not know α, it can compute the secret keys $\{sk_\ell\}_{\ell \in \mathcal{L}}$, where $sk_\ell := \boldsymbol{\alpha}_\ell = M'_\ell \cdot \boldsymbol{r}$, because the left-most column of all the matrices M'_ℓ for $\ell \in \mathcal{L}$ is zero. Moreover, $\mathcal{B}$ can compute $pkc = (g^{\boldsymbol{\alpha}_1}, \ldots, g^{\boldsymbol{\alpha}_N})$

because $g^{\boldsymbol{\alpha}_i} = g^{M'_i \cdot \boldsymbol{r}}$ and $\mathcal{B}$ has g^{α} as well as $\rho_2, \ldots, \rho_d \in \mathbb{Z}_q$. Now $\mathcal{B}$ sends pk, pkc, and $\{sk_\ell\}_{\ell \in \mathcal{L}}$ to $\mathcal{A}$. We know that these quantities are distributed as in the real scheme because the programs P and $P_{\mathcal{L}}$ are equivalent.

Next, $\mathcal{A}$ issues signing queries. To respond to a query for $m \in \mathcal{M}$ our $\mathcal{B}$ queries its own BLS signing oracle to obtain $\sigma \leftarrow H(m)^{\alpha}$. It can now compute all the signature shares $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_N$ for m because

$$\boldsymbol{\sigma}_i = H(m)^{M'_i \cdot \boldsymbol{r}} \quad \in \mathbb{G}^{d_i} \quad \text{for all } i = 1, \ldots, N.$$

$\mathcal{B}$ can compute these values because $\boldsymbol{r} = (\alpha, \rho_2, \ldots \rho_d)$ and $\mathcal{B}$ has $H(m)^{\alpha}$ as well as $\rho_2, \ldots, \rho_d \in \mathbb{Z}_q$.

Now that $\mathcal{B}$ supplied $\mathcal{A}$ with all the information it requires, $\mathcal{A}$ will eventually produce a BLS forgery (m, σ) with a certain probability. This is a valid BLS forgery because $\mathcal{A}$ never queried for a signature for m. Therefore, $\mathcal{B}$ succeeds in forging a BLS signature with the same probability as $\mathcal{A}$'s advantage against $\mathcal{S}_{\text{linBLS}}$. □

Remark 22.16 (Stronger notions of security for an access structure). We note that the above definition of signature security for an access structure is not as strong as one might intuitively expect. In a stronger definition, we could model an attack in which for a given message m, the adversary might obtain signature shares from just a subset of servers $\mathcal{T}_m \subseteq \{1, \ldots, N\}$, rather than all of the servers, and so long as $\mathcal{L} \cup \mathcal{T}$, the adversary should not be able to compute a signature on m. Our security definition above is too crude to model this type of attack, because in a signing query, the adversary always gets signing shares from all of the servers. In the next section, we will develop a more refined security model that does model this type of attack. See, in particular, Remark 22.17 for a stronger definition of secure signatures for an access structure (as well Remark 22.19 for a stronger definition of secure decryption for an access structure). The good news is that the BLS threshold signature scheme based on linear secret sharing also satisfies this stronger definition, albeit under a stronger (but reasonable) assumption (and the same holds true for the GS threshold decryption scheme based on linear secret sharing). □

22.6 Gap security for threshold cryptosystems

In discussing the security of t-out-of-N threshold schemes, we have assumed that there may be up to $t-1$ corrupt parties that obtain shares of the secret key. Moreover, our formal definitions of security only imply the following:

> if an adversary is able to obtain a signature/decryption, then *at least one* honest party must have generated a corresponding signature/decryption share.

In this section, we briefly discuss a stronger, more nuanced notion of **gap security** that works with two bounds: a bound t on the number of shares required to construct a signature/decryption, and a bound $L \leq t-1$ on the number of corrupt parties. (Note that in Section 22.4, we already modeled these two separate parameters for DKG protocols.)

Before getting into more details, let us first develop some intuition. Consider the generic t-out-of-N threshold signature scheme discussed in Section 22.2.1, where a threshold signature consists of a collection of t ordinary signatures. Assume that at most $L \leq t-1$ corrupt parties obtain secret key shares (i.e., ordinary signing keys). Now suppose an adversary, which collaborates with or controls the corrupt parties, is able to construct a threshold signature on a particular message m (i.e., constructs t ordinary signatures on m). The corrupt parties may generate at most L

signature shares (i.e., ordinary signatures) on m which means that *at least* $n - L$ honest parties must have generated signature shares (i.e., ordinary signatures) on m as well. This is a stronger security property than is guaranteed by Definition 22.4. There are applications where this stronger security property is necessary (for example, some protocols for Byzantine Agreement, which we touched on Section 22.4.2.4). Does this mean that for such applications we must use the generic threshold signature scheme, rather than more compact threshold signature schemes, such as BLS from Section 22.2.2? The answer is *no*: we can prove that the BLS signature scheme provides this stronger security property; however, to do so, we must make a somewhat stronger (but still reasonable) computational assumption.

In this section, we sketch the changes that must be made to our definitions of security for threshold signing and decryption to capture this notion of gap security, and the stronger assumptions needed to prove that the BLS signature scheme and the GS decryption scheme (see Section 22.3.3) satisfy these stronger definitions.

22.6.1 Gap security for threshold signature schemes

Consider Definition 22.4, which defines the security of a threshold signature scheme, and more specifically, Attack Game 22.1. Here is how this attack game must be modified to capture the notion of gap security.

Attack Game 22.9 (threshold signature gap security). For a given threshold signature scheme $\mathcal{S} = (G, S, V, C)$, defined over $(\mathcal{M}, \Sigma)$, and a given adversary $\mathcal{A}$, we define the following attack game.

- *Setup:* the adversary sends poly-bounded allowable parameters N and t, where $0 < t \leq N$, and a subset $\mathcal{L} \subseteq \{1, \ldots, N\}$, where $|\mathcal{L}| < t$, to the challenger.

  The challenger does the following:

  – initialize an associative array
  $$Map : \mathcal{M} \to \{1, \ldots, N\} \setminus \mathcal{L},$$
  initialized with $Map[m] = \emptyset$ for all $m \in \mathcal{M}$,

  – run
  $$(pk, pkc, sk_1, \ldots, sk_N) \xleftarrow{\text{R}} G(N, t),$$

  – send pk, pkc, and $\{sk_\ell\}_{\ell \in \mathcal{L}}$ to $\mathcal{A}$.

- *Signing queries:* for $j = 1, 2, \ldots$, signing query j from $\mathcal{A}$ consists of a message $m_j \in \mathcal{M}$ and an index $i_j \in \{1, \ldots, N\} \setminus \mathcal{L}$.

  The challenger does the following:

  – update $Map[m_j] \leftarrow Map[m_j] \cup \{i_j\}$,

  – compute $S(sk_{i_j}, m_j)$ and send this to the adversary.

- *Forgery:* eventually $\mathcal{A}$ outputs a candidate forgery pair $(m, \sigma) \in \mathcal{M} \times \Sigma$.

We say that the adversary wins the game if the following two conditions hold:

- $V(pk, m, \sigma) = \mathsf{accept}$, and
- $|\mathcal{L} \cup Map[m]| < t$.

We define $\mathcal{A}$'s advantage with respect to $\mathcal{S}$ denoted $\mathrm{dthSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 22.25 (gap secure threshold signatures). *We say that a threshold signature scheme $\mathcal{S}$ is **gap secure** if for all efficient adversaries $\mathcal{A}$, the quantity $\mathrm{dthSIG}\mathsf{adv}[\mathcal{A}, \mathcal{S}]$ is negligible.*

The essential differences between Attack Game 22.1 and Attack Game 22.9 are that

- in the latter, the adversary obtains one signature share at a time, rather than all at once;
- in the latter, the adversary wins if it obtained signature shares on m from fewer than $t - L$ honest parties, where $L := |\mathcal{L}|$.

One can easily show that the generic threshold signature scheme presented in Section 22.2.1 satisfies Definition 22.25, assuming the underlying non-threshold scheme is secure.

The threshold BLS signature scheme presented in Section 22.2.2 satisfies Definition 22.25, assuming a certain interactive variant of the CDH assumption holds in the group $\mathbb{G}$. As usual, we define this assumption via an attack game.

Attack Game 22.10 (Linear one-more DH). Let $\mathbb{G}$ be a group of prime order q generated by $g \in \mathbb{G}$. For a given adversary $\mathcal{A}$, we define the following attack game.

- The challenger chooses $\mu_1, \ldots, \mu_k \in \mathbb{Z}_q$ and $\nu_1, \ldots, \nu_\ell \in \mathbb{Z}_q$ at random, and gives $\{g^{\mu_i}\}_{i=1}^{k}$ and $\{g^{\nu_j}\}_{j=1}^{\ell}$ to the adversary.
- The adversary makes a sequence of queries to the challenger, each of which is a vector over $\mathbb{Z}_q$ of the form $\{\kappa_{i,j}\}_{i,j}$, to which the challenger responds with
$$\prod_{i,j} \left(g^{\mu_i \nu_j}\right)^{\kappa_{i,j}}.$$
- To end the game, the adversary outputs a vector $\{\lambda_{i,j}\}_{i,j}$ over $\mathbb{Z}_q$ and a group element $h \in \mathbb{G}$, and wins the game if
$$h = \prod_{i,j} \left(g^{\mu_i \nu_j}\right)^{\lambda_{i,j}}$$
and the output vector $\{\lambda_{i,j}\}_{i,j}$ is not a $\mathbb{Z}_q$-linear combination of the query vectors.

We define $\mathcal{A}$'s **advantage in solving the linear one-more DH problem for** $\mathbb{G}$, denoted $\mathrm{lin1mDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$, as the probability that $\mathcal{A}$ wins this game. □

Definition 22.26 (Linear one-more DH assumption). *We say that the **one-more DH assumption** holds for $\mathbb{G}$ if for all efficient adversaries $\mathcal{A}$ the quantity $\mathrm{lin1mDH}\mathsf{adv}[\mathcal{A}, \mathbb{G}]$ is negligible.*

It is not difficult to prove the gap security of threshold BLS under the linear one-more DH assumption. One can also show that the same holds in the *asymmetric pairing* setting under a variation of the linear one-more DH assumption in which Attack Game 22.10 is modified so that (with notation as in Section 15.5.1):

- g^{μ_i} is replaced by $g_1^{\mu_i}$,
- g^{ν_j} is replaced by $g_0^{\nu_j}$,
- $g^{\mu_i \nu_j}$ is replaced by $g_0^{\mu_i \nu_j}$,
- h is in $\mathbb{G}_0$.

Note that these security results also hold if we use a secure DKG protocol as in Section 22.4.

Remark 22.17 (Monotone access structures). All of the above definitions generalize to monotone access structures, as in Section 22.5. To model gap security with respect to monotone access structures, Attack Game 22.9 may be modified as follows:

- In the *Setup* stage, instead of requiring that $|\mathcal{L}| < t$, the requirement is that $\mathcal{L}$ is an unauthorized set.
- In defining the winning]ondition, instead of requiring that $|\mathcal{L} \cup \mathit{Map}[m]| < t$, the requirement is that $\mathcal{L} \cup \mathit{Map}[m]$ is an unauthorized set.

One can also show that the BLS threshold signature scheme based on linear secret sharing presented in Section 22.5.3 satisfies this definition under the linear one-more DH assumption. □

22.6.2 Gap security for threshold decryption schemes

Consider Definition 22.12, which defines the security of a threshold decryption scheme, and more specifically, Attack Game 22.5. Here is how this attack game must be modified to capture the notion of gap security.

Attack Game 22.11 (threshold AD-only CCA gap security). For a public-key threshold decryption scheme $\mathcal{E} = (G, E, D, C)$ defined over $(\mathcal{M}, \mathcal{D}, \mathcal{C})$, and for a given adversary $\mathcal{A}$, we define two experiments.

Experiment b $(b = 0, 1)$:

- *Setup:* the adversary sends poly-bounded allowable parameters N and t, where $0 < t \leq N$, and a subset $\mathcal{L} \subseteq \{1, \dots, N\}$, where $|\mathcal{L}| < t$, to the challenger.

  The challenger does the following:

  – initialize an empty associative array
  $$\mathit{Map} : \mathcal{D} \to \{1, \dots, N\} \setminus \mathcal{L},$$
  – run
  $$(pk, pkc, sk_1, \dots, sk_N) \xleftarrow{R} G(N, t),$$
  – send pk, pkc, and $\{sk_\ell\}_{\ell \in \mathcal{L}}$ to $\mathcal{A}$.

- $\mathcal{A}$ then makes a series of queries to the challenger. Each query can be one of two types:

  – *Encryption query:* for $i = 1, 2, \dots$, the ith encryption query consists of a triple $(m_{i0}, m_{i1}, d_i) \in \mathcal{M}^2 \times \mathcal{D}$, where the messages m_{i0}, m_{i1} are of the same length.
  The challenger does the following:

- $*$ compute $c_i \xleftarrow{\text{R}} E(pk, m_{ib}, d_i)$,
- $*$ if $d_i \notin \text{Domain}(\mathit{Map})$, then initialize $\mathit{Map}[d_i] \leftarrow \emptyset$,
- $*$ send c_i to $\mathcal{A}$.

– *Decryption query:* for $j = 1, 2, \ldots$, the jth decryption query consists of a triple

$$(\hat{c}_j, \hat{d}_j, i_j) \in \mathcal{C} \times \mathcal{D} \times \{1, \ldots, N\} \setminus \mathcal{L}$$

such that either

- $*$ $\hat{d}_j \notin \text{Domain}(\mathit{Map})$, or
- $*$ $\hat{d}_j \in \text{Domain}(\mathit{Map})$ and $|\mathcal{L} \cup \mathit{Map}[\hat{d}_j] \cup \{i_j\}| < t$.

The challenger does the following:

- $*$ if $\hat{d}_j \in \text{Domain}(\mathit{Map})$, then update $\mathit{Map}[\hat{d}_j] \leftarrow \mathit{Map}[\hat{d}_j] \cup \{i_j\}$,
- $*$ compute $D(sk_{i_j}, \hat{c}_j, \hat{d}_j)$ and send this to $\mathcal{A}$.

- At the end of the game, $\mathcal{A}$ outputs a bit $\hat{b} \in \{0, 1\}$.

If W_b is the event that $\mathcal{A}$ outputs 1 in Experiment b, define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{dthCCA}_{\text{ado}}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Definition 22.27 (threshold AD-only CCA gap security). *A public-key threshold decryption scheme $\mathcal{E}$ is **AD-only CCA gap secure** if for all efficient adversaries $\mathcal{A}$, the value* $\text{dthCCA}_{\text{ado}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

Remark 22.18 (from AD-only to full CCA gap security). Analogous to Remark 22.10, one can also define a notion of (full) CCA *gap* security for threshold decryption schemes. To do model this, in the above attack game, the domain of the associated array map needs to be extended from $\mathcal{D}$ to $\mathcal{C} \times \mathcal{D}$; for an encryption query, the input to *Map* is (c_i, d_i), and for a decryption query, $(\hat{c}_j, \hat{d}_j)$. Just as before, we can convert any AD-only CCA gap secure scheme to a CCA gap secure scheme via "wrapping" with a strongly one-time secure signature, to the reader. □

One can show that the generic threshold decryption scheme presented in Section 22.3.1 satisfies Definition 22.27, assuming the underlying non-threshold scheme is secure.

One can show that the threshold GS decryption scheme presented in Section 22.3.3 (as well as the pairing-free variant in Section 22.3.6) satisfies Definition 22.27, under the linear one-more DH assumption.

Remark 22.19 (Monotone access structures). All of the above definitions generalize to monotone access structures, as in Section 22.5. To model gap security with respect to monotone access structures, Attack Game 22.11 may be modified as follows:

- In the *Setup* stage, instead of requiring that $|\mathcal{L}| < t$, the requirement is that $\mathcal{L}$ is an unauthorized set.

- In defining the pre-condition on decryption queries, instead of requiring that $|\mathcal{L} \cup \mathit{Map}[\hat{d}_j] \cup \{i_j\}| < t$, the requirement is that $\mathcal{L} \cup \mathit{Map}[\hat{d}_j] \cup \{i_j\}$ is an unauthorized set.

One can also show that the GS threshold decryption scheme based on linear secret sharing satisfies this definition under the linear one-more DH assumption. □

22.7 A fun application: a randomness beacon

To be written.

22.8 Notes

Citations to the literature to be added.

22.9 Exercises

22.1 (Special thresholds). In Section 22.1.1 we saw Shamir's t-out-of-N secret sharing scheme over $\mathbb{Z}_q$ which works for any $0 < t \leq N < q$. At the two extremes, $t = 1$ and $t = N$, there are much simpler secret sharing constructions.

(a) Consider the following N-out-of-N secret sharing scheme (G, C) for a secret $\alpha \in \mathbb{Z}_q$:

- algorithm $G(N, N, \alpha)$ chooses random $\alpha_1, \ldots, \alpha_{N-1} \xleftarrow{\text{R}} \mathbb{Z}_q$, sets $\alpha_N := \alpha - \sum_{i=1}^{N-1} \alpha_i \in \mathbb{Z}_q$, and outputs $\alpha_1, \ldots, \alpha_N$ as the shares of $\alpha \in \mathbb{Z}_q$.
- algorithm C reconstructs the secret $\alpha \in \mathbb{Z}_q$ from $\alpha_1, \ldots, \alpha_N$ by computing $\alpha \leftarrow \sum_{i=1}^{N} \alpha_i$.

Show that this is a secure N-out-of-N secret sharing scheme as in Definition 22.2.

(b) Construct a simple 1-out-of-N secret sharing scheme (G_1, C_1) and prove it to be a secure 1-out-of-N scheme. Show that your scheme is the same as Shamir's t-out-of-N secret sharing scheme with $t = 1$.

22.2 (Access structures). Generalize the BLS threshold signature scheme of Section 22.2.2 to the following settings: The N decryption servers are split into two disjoint groups $\mathcal{S}_1$ and $\mathcal{S}_2$, and signing a message m should be possible only if the combiner receives at least t_1 responses from the set $\mathcal{S}_1$, and at least t_2 responses from the set $\mathcal{S}_2$, where $t_1 \leq |\mathcal{S}_1|$ and $t_2 \leq |\mathcal{S}_2|$. A signature in your scheme should be a single group element. Adapt the security definition to these settings, and prove that your scheme is secure.

Discussion: Your construction is an example of adapting BLS signatures to more general access structure than a simple threshold. It is a special case of the technique discussed in Section 22.5

22.3 (Generic threshold signing). Prove that the generic threshold signature scheme (G', S', V', C') from Section 22.2.1 is secure and robust, as defined in Section 22.2.3, assuming the underlying signature scheme (G, S, V) is secure.

22.4 (Updating** N **and** t**). In our description of the BLS signature scheme, the number of key shares N and the threshold t were determined during key generation time. What if we want to change N or t after the key shares have been generated, but without changing the public key pk?

(a) *Increasing N:* to increment N by one, the signing servers need to generate a new key share for the new key server, and update pkc. Let's design a protocol that lets t signing servers provision a new server with a key share. Suppose that the t signing servers have somehow generated a t-out-of-t sharing of zero: for $i = 1, \ldots, t$ server i has $\rho_i \in \mathbb{Z}_q$, and $\rho_1 + \ldots + \rho_t = 0$. Explain how each of the t key servers can send a single $\mathbb{Z}_q$ element to the new server that lets the new server construct its key share. The new server should learn nothing other than its new key share.

(b) *Decreasing t:* a single signing server can decrease the threshold t by one by simply sending its secret key share to all other servers. Show that the resulting scheme is $(t-1)$-out-of-N secure.

Discussion: Incrementing t is more problematic. All the signing servers have to be issued new key shares, and they must be trusted to delete the old key shares, otherwise they can continue to use the old key shares with a t threshold. We will come back to this question in Exercise 22.4.

22.5 (Proactive security). Let $\alpha \in \mathbb{Z}_q$ be a secret and let $(\alpha_1, \ldots, \alpha_N) \xleftarrow{\text{R}} G_{\text{sh}}(N, t, \alpha)$ be a Shamir secret sharing of α. The parties holding the shares are concerned that an adversary may slowly steal the shares, one share at a time. Concretely, suppose that every week the attacker can obtain one additional share. Then after t weeks the attacker will learn the secret α. To defend against this slow leakage, the parties decide to use a technique called **proactive security**, where every week the N parties engage in a *proactive refresh protocol.* The protocol generates new shares $(\alpha'_1, \ldots, \alpha'_N)$ for the parties, but without changing the shared secret $\alpha \in \mathbb{Z}_q$.

(a) At the very least, a proactive refresh should ensure that an adversary who obtains $t-1$ shares $\{\alpha_j\}_{j \in \mathcal{J}}$ before the refresh, and obtains $t-1$ shares $\{\alpha'_j\}_{j \in \mathcal{J}'}$ after the refresh, learns nothing about α. Describe a security game that captures this requirement.

(b) For Shamir secret sharing, design a simple protocol among the N share holders to implement a secure proactive refresh for a sharing of $\alpha \in \mathbb{Z}_q$. Prove that (i) your protocol results in a new sharing of the same secret α, and (ii) the protocol satisfies the security requirement from part (a).
Hint: Use the linearity of Shamir secret sharing from Remark 22.15. Your protocol can designate one party to generate a fresh sharing of zero by computing $(\alpha''_1, \ldots, \alpha''_N) \xleftarrow{\text{R}} G_{\text{sh}}(N, t, 0)$. For $i = 1, \ldots, N$, the party sends α''_i to party i, and party i adds this α''_i to its own share.

22.6 (Blind BLS threshold signing). In the BLS threshold signature scheme from Section 22.2.2.1, the combiner C first sends the message m to the signers who then return the signature shares, as in Fig. 22.1b. The combiner then assembles the signature shares into a signature on m. Show how the behavior of the combiner and the signers be modified so that the signers never see the message m to be signed in the clear, and yet the combiner obtains a valid signature on m.
Hint: use BLS blind signatures from Exercise 15.7.

22.7 (Privacy against insiders). Definition 22.10 defines the concept of a private threshold signature scheme, a PTS, where privacy holds against the public.

(a) Give an example of a PTS that satisfies Definition 22.10, but any one of the N signers can use its secret key to examine a signature and determine the set of parties that generated it.

(b) Consider the following two experiments that ensure privacy against insiders. Experiment b, for $b = 0, 1$, proceeds as follows: (i) the adversary outputs N and t; (ii) the challenger runs $G(N, t)$ and sends all the resulting information to the adversary including the N secret keys; (iii) the adversary issues a sequence of queries where query number i is a triple $(m_i, \mathcal{J}_i^{(0)}, \mathcal{J}_i^{(1)})$ containing a message and two t-size sets, and the challenger responds with a signature on m_i by obtaining signature shares from $\mathcal{J}_i^{(b)}$ and combining them; (iv) the adversary outputs a guess $\hat{b}$ for b. We say that a PTS is **insider private** if no adversary can distinguish the two experiments.

Show that the BLS threshold signature scheme is insider private.

22.8 (CPA secure threshold decryption). This exercise introduces a notion of security against chosen plaintext attack (CPA security) for threshold decryption schemes. The attack game is the same as Attack Game 22.5, except that decryption queries work as follows:

> For $j = 1, 2, \ldots$, the jth decryption query consists of $(\hat{m}_j, \hat{d}_j) \in \mathcal{M} \times \mathcal{D}$. The challenger computes $\hat{c}_j \overset{\text{R}}{\leftarrow} E(pk, \hat{m}_j, \hat{d}_j)$ along with the N decryption shares $c'_{j,i} \leftarrow D(sk_i, \hat{c}_j, \hat{d}_j)$, for $i = 1, \ldots, N$, and sends $\hat{c}_j$ and all of the decryption shares $c'_{j,i}$ to $\mathcal{A}$.

$\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ is denoted thCPA$\mathsf{adv}[\mathcal{A}, \mathcal{E}]$. Note that in the CPA security setting, associated data really plays no role, and we may assume there is none.

Now consider the threshold ElGamal decryption scheme introduced in Section 22.3.2, which is derived from the ordinary ElGamal encryption scheme $\mathcal{E}_{\text{EG}}$ introduced in Section 11.5. Show that if $\mathcal{E}_{\text{EG}}$ is semantically secure, then the threshold ElGamal decryption scheme is CPA secure.

22.9 (Threshold RSA decryption). Let us show how to enable simple threshold decryption for the RSA public key encryption scheme of Section 11.4.1.

(a) Recall that the key generation algorithm generates numbers n, e, d, where n is the RSA modulus, e is the encryption exponent, and d is the decryption exponent. We extend the key generation algorithm with two more steps: choose a random integer d_1 in $[1, n^2]$ and set $d_2 = d_1 - d \in \mathbb{Z}$. Then output the two key shares $sk_1 := (n, d_1)$ and $sk_2 := (n, d_2)$, and the public key $pk := (n, e)$. Explain how to use this setup to convert $\mathcal{E}_{\text{RSA}}$ (see Section 11.4.1) to a 2-out-of-2 threshold decryption scheme.

Hint: Show that the distribution of the key share d_2 is statistically close to the uniform distribution on $\{1, \ldots, n^2\}$.

(b) Prove that your scheme from part (a) satisfies the security definition for CPA secure threshold decryption from the previous exercise.

(c) Generalize the scheme to provide 2-out-of-3 threshold decryption, using the mechanism of Exercise 2.20. Prove that the scheme is CPA secure.

22.10 (Threshold RSA signatures). In Exercise 22.9 we showed how a secret RSA decryption key can be split into three shares, so that two shares are needed to decrypt a given ciphertext, but a single share reveals nothing. In this exercise we show that the same can be done for RSA signatures, namely two shares are needed to generate a signature, but one share reveals nothing.

(a) Use Exercise 22.9 to construct a 2-out-of-3 threshold RSA signature scheme.

(b) Prove that your scheme from part (a) satisfies Definition 22.4.

22.11 (From AD-only CCA security to CCA security). Attack Game 22.5 captures the notion of AD-only CCA security for threshold decryption. In Remark 22.10, we outlined how to modify this game game to capture full CCA security. The conversion from AD-only to full CCA security in Exercise 14.13 can be easily adapted to the threshold setting, where the signature check is performed by the decryption algorithm (as opposed to the combiner algorithm). Prove that this conversion also coverts an AD-only CCA secure threshold decryption scheme to a fully CCA secure threshold decryption scheme, assuming the signature scheme is strongly one-time secure.

22.12 (Threshold Schnorr signatures). Show that the Schnorr signature scheme supports t-out-of-N threshold signing. To generate a signature we allow for three rounds of communication between the combiner and the key servers. The servers communicate with the combiner, but not with each other. This is quite different from the threshold signature schemes presented in this chapter, where signing required only a single message from each participating server to the combiner.

22.13 (Threshold master key in identity based encryption). In Section 15.6.1 we constructed a number of IBE schemes. One way to protect the master key msk in an IBE scheme is to secret share it among N parties so that $t > 1$ of the N parties must work together to generate a secret key sk_{id} for an identity id. This way, no single party can misuse msk to generate sk_{id}.

(a) Recall that in the IBE scheme $\mathcal{E}_{\text{BF}}$ from Section 15.6.3.1 the master key msk is a secret value $\alpha \in \mathbb{Z}_q$ and $sk_{id} := H_0(id)^\alpha$. Use t-out-of-N Shamir secret sharing, as defined in Section 22.1, to show how to secret share α among N parties so that $sk_{id} = H_0(id)^\alpha$ can be generated without ever reconstructing α at a single location.

(b) Show that $t - 1$ parties cannot generate sk_{id}, assuming CDH holds in $\mathbb{G}_0$, and H_0 is modeled as a random oracle.

22.14 (A CCA-secure threshold decryption scheme from pairings). In Section 15.6.4.1 we saw a direct construction for a CCA secure public key encryption scheme from identity based encryption (IBE). The construction supports a public validity test for ciphertexts: anyone can validate that a ciphertext is well formed by verifying a signature embedded in the ciphertext. Instantiating this general transformation using a pairing-based IBE gives a simple CCA-secure public key threshold decryption system. In particular, consider the IBE system $\mathcal{E}_{\text{BF}}$ from Section 15.6.3.1. In Exercise 22.13 we showed that the master key in this IBE system can be secret shared among N parties so that any t parties can generate a decryption key.

(a) Show how to apply the construction from Section 15.6.4.1 to the threshold IBE system from Exercise 22.13 to obtain a public key threshold decryption system.

(b) Prove that your construction is secure as in Definition 22.12 and is robust.

22.15 (An alternative to Construction 22.1). Let $(\mathcal{S}, \mathbb{A})$ be an access structure as discussed in Section 22.5. Construction 22.1 is an additive secret sharing scheme over $\mathbb{Z}_q$ that implements $(\mathcal{S}, \mathbb{A})$, albeit with secret shares that can be quite large. In this exercise we develop a different generic additive secret sharing scheme. Let $\mathbb{U} \subseteq 2^{\mathcal{S}}$ be the set of all unauhorized sets, namely $\mathbb{U}$ is the complement of $\mathbb{A}$. For $i = 1, \ldots, N$, let $\mathbb{U}^{(i)} \subseteq \mathbb{U}$ be all the sets in $\mathbb{U}$ that do not contain i. Let us construct a secret sharing scheme (G, C) that implements $(\mathcal{S}, \mathbb{A})$.

- $G(\alpha)$: for each $\mathcal{L} \in \mathbb{U}$ sample a random $\rho^{(\mathcal{L})} \xleftarrow{\text{R}} \mathbb{Z}_q$ such that the entire collection $\{\rho^{(\mathcal{L})}\}_{\mathcal{L}\in\mathbb{U}}$ satisfies $\alpha = \sum_{\mathcal{L}\in\mathbb{U}} \rho^{(\mathcal{L})}$. Then, for $i = 1, \ldots, N$ the share $\boldsymbol{\alpha}_i$ is

$$\boldsymbol{\alpha}_i := \left\{\rho^{(\mathcal{L})}\right\}_{\mathcal{L}\in\mathbb{U}^{(i)}}$$

This $\boldsymbol{\alpha}_i$ contains one element in $\mathbb{Z}_q$ for every unauthorized set that does not contain i. Output the list of shares $(\boldsymbol{\alpha}_1, \dots, \boldsymbol{\alpha}_N)$.

(a) Explain how algorithm $C(\mathcal{J}, \{\boldsymbol{\alpha}_j\}_{j \in \mathcal{J}})$ works to output $\alpha \in \mathbb{Z}_q$ when $\mathcal{J} \in \mathbb{A}$. First, show that for every authorized set $\mathcal{J} \in \mathbb{A}$, the set $\{\boldsymbol{\alpha}_j\}_{j \in \mathcal{J}}$ must include all the elements in $\{\rho^{(\mathcal{L})}\}_{\mathcal{L} \in \mathbb{U}}$.

(b) Prove that the scheme (G, C) is secure as in Definition 22.20.

Discussion: This scheme leads to smaller shares compared to Construction 22.1 when $(\mathcal{S}, \mathbb{A})$ has more authorized sets than unauthorized sets. The size of the secret shares can be further reduced by generating a random $\rho^{(\mathcal{L})}$ in $\mathbb{Z}_q$ only when $\mathcal{L} \in \mathbb{U}$ is a maximally unauthorized set (namely $\mathcal{L}$ is unauthorized, but every superset of $\mathcal{L}$ is authorized).

22.16 (Revocation using threshold decryption). In this exercise we look at an unexpected application for threshold decryption. Recall that in Section 5.6 we saw a revocation scheme that enables a sender to broadcast an encrypted message to N recipients so that all but r of them can decrypt. The ciphertext size was proportional to $r \log_2(N/r)$. In this exercise we develop a revocation scheme using a threshold decryption system (G, E, D, C), where ciphertext size is only proportional to r, which is better. For now, suppose there is an upper bound t on the maximum number of recipients to revoke.

- Initialize a $(t+1)$-out-of-$(N+t)$ threshold decryption scheme to get pk and $sk_1, \dots, sk_{N+t}$. For $i = 1, \dots, N$ give recipient i the key sk_i. The sender keeps the encryption key $ek := (pk, sk_{N+1}, \dots, sk_{N+t})$ to itself.

- When no one is revoked, the sender encrypts a message m by first computing $c := E(pk, m)$ and then computing the decryption shares $c_i := D(sk_{N+i}, c)$ for $i = 1, \dots, t$. The sender broadcasts $(c, c_1, \dots, c_t)$. Every recipient computes the decryption share of c using its secret key. It then uses its own decryption share plus the t decryption shares in the ciphertext to recover m. An eavesdropper only has t decryption shares and cannot decrypt.

- Now, suppose recipients 1 and 2 are revoked. The sender operates as before, but uses keys $sk_1, sk_2, sk_{N+1}, \dots, sk_{N+t-2}$ to compute the t decryption shares embedded in c. Now, recipients 1 and 2 only have t decryption shares and cannot decrypt. Everyone else has $t+1$ decryption shares and can decrypt. Hence, we revoked users 1 and 2 and no one else.

(a) Exercise 5.21 defines the syntax for a broadcast encryption scheme (G', E', D'). Using this syntax, instantiate the idea above using ElGamal threshold decryption. You may assume that t is a fixed system parameter. Explain exactly how encryption $E'(ek, m, S)$ to recipients in a set S works, where $|S| \geq N - t$, and how decryption $D(sk_i, c)$ works. Show that ciphertext size is linear in t.

(b) Exercise 5.21 defines a security model for a broadcast encryption scheme. Show that the scheme from part (a) is secure assuming the threshold decryption scheme is secure, and the adversary requests at most t secret keys.

(c) So far the scheme can only revoke up to an a-priori bound of t recipients. However, suppose the sender sets up $\lceil \log_2 N \rceil$ such schemes, where in scheme number i the value of t is set to $t := 2^i$. Show how this lets the sender revoke as many recipients as it wants, but ciphertext size is only linear in the number of revoked recipients.

Chapter 23

Secure multi-party computation

Suppose several parties wish to participate in an auction. For concreteness, let us assume this is a *sealed-bid second-price auction*, also called a *Vickrey auction*, where parties submit sealed bids, so that no party knows how much the other parties have bid. The highest bidder wins the auction, but the price paid is the second highest bid. Moreover, losing bids are not revealed. The reason to use a second-price auction with sealed bids is that every bidder is then incentivized to bid their true valuation of the item for sale, assuming no collusion among the bidders.

Now suppose the auction is to be conducted over the Internet. The auction house runs a trusted server T that will collect bids from the various parties who wish to make bids. After collecting all the bids, the auction house selects the highest bidder, and announces the winner and the amount that the winner will pay. All other information from the bidders should remain secret.

It goes without saying that the communication links between the bidders and the server T will be secured, for example, using the TLS protocol which is used to secure traffic between two parties. Provided T is secure and follows the protocol, all should be well. But what if T itself is compromised and/or does not follow the protocol? Consider the following failure scenarios:

Failure Scenario 1: T follows the protocol, collecting all of the bids and announcing the correct winner. However, some time after running the protocol, a hacker breaks into T and retrieves all of the bids — not just the winning bid, but all of the losing bids as well.

Even though the auction is over, the release of this information can have detrimental consequences. For example, if there are subsequent auctions, this can reveal the bidding strategies of some parties.

Failure Scenario 2: T does not follow the protocol. For example, T may collude with one of the bidding parties P: after collecting bids from all other parties, it informs P of the highest bid received so far, allowing P to submit a bid that just beats the current highest bid.

Failure Scenario 3: T may follow the protocol, but crash. Worse, T could crash after it has reported the winner to some parties but not to others. Now if the parties re-run the auction, some parties already know the value of the winning bid, and can change their bidding strategy.

The problem with using a trusted server T to implement the auction is that it is a *single point of failure.* If T is hacked, the integrity of the auction may be compromised, as illustrated in the failure scenarios outlined above.

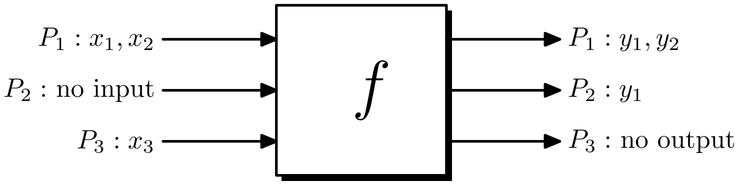


Figure 23.1: Example inputs and outputs of three parties P_1, P_2, P_3 to a function f

A technique that may be used to mitigate the risk of a single point of failure is to use several servers in place of a single server. For example, the auction service could be implemented using several servers instead of just one. The hope, then, is that if only one (or a small number) of these servers is compromised, the integrity of the auction itself will not be compromised.

Note that simply *replicating* the trusted server T does not solve the problem. For example, in Scenario 1 above, if the servers are just replicas of T, then by hacking in to any one of these replicas, an attacker can retrieve the losing bids.

So the question of whether we can securely distribute the service provided by T to several servers is an interesting and non-trivial question. It turns out that the answer is *yes*. The techniques used to solve this type of problem go by the name **multi-party computation** or **MPC**. This is a very rich and broad topic, and one can easily devote an entire book to it. In this chapter, we will introduce the reader to some of the core MPC concepts and techniques.

Recall that in Section 8.12 we presented a simple auction protocol using a "commit and reveal" approach. However, in that protocol, the server T had to reveal all the bids at the end of the auction, so that that the parties could verify that the auction was run correctly. As explained in Failure Scenario 1, all the non-winning bids should remain hidden. MPC techniques give us the best of both worlds: the bidders can verify that the auction was run correctly, but the non-winning bids remain hidden.

23.1 The basic idea of MPC

We can generalize the auction example as follows. Suppose we have a function f that takes n inputs and produces m outputs:

$$(y_1, \ldots, y_m) = f(x_1, \ldots, x_n).$$

We have N parties, $P_1, \ldots, P_N$. Their goal is to run a protocol where each input value x_i is submitted by one of the parties, and each output y_j is obtained by one or more of the parties. Which party submits a particular input x_i and which parties obtain a particular output y_j is a part of the protocol specification. Note that an individual party need not submit any inputs or obtain any outputs, as shown in Fig. 23.1.

23.1.1 Informal notions of security

It is challenging enough to even *define* secure multi-party computation, let alone to design and analyze protocols that satisfy such definitions. We begin by discussing some informal security requirements.

In an execution of a protocol, some parties may be honest and others may be corrupt. Some essential security properties that any secure protocol should provide are *privacy*, *soundness*, and *input independence*:

privacy: no party learns anything about any other party's inputs (except for information that is inherently revealed by the outputs);

soundness: honest parties compute correct outputs (if they compute any output at all);

input independence: all parties must choose their inputs independently of the other parties' inputs.

In the auction example, all of these properties are clearly desirable. *Privacy* guarantees that only the winning bid is revealed. *Soundness* guarantees that the highest bidder wins. *Input independence* guarantees that no party can place a bid that depends on any other bid. For example, if Alice bids x dollars, it is not enough for the protocol to hide the value x from other parties; rather, it should be infeasible for another party to use Alice's bid to bid, say, $x + 1$ dollars without knowledge of x.

Note that our informal definition of soundness says that for every honest party, if it computes an output at all, then its output must be correct. This somewhat wishy-washy definition allows for the possibility that corrupt parties can disrupt the protocol to such a degree that some honest parties end up with no output at all. In some cases, this can mean that some (or all) corrupt parties obtain their outputs, while some (or all) honest parties fail to do so. As we saw, this is not acceptable in the auction example.

Therefore, we can define two other desirable properties.

Guaranteed output delivery: all honest parties are guaranteed to obtain their output.

Fairness: if any party (corrupt or honest) obtains their output, then all honest parties do so.

The *guaranteed output delivery* property is stronger than the *fairness* property. In the auction example, guaranteed output delivery is important, and fairness even more so; however, in other applications, fairness and guaranteed output delivery may not be essential. Moreover, it turns out that achieving fairness or guaranteed output delivery typically comes at a cost in terms of the complexity of the protocol. Therefore, it makes sense to consider different types of protocols: those that do not achieve fairness, those that achieve fairness but not guaranteed output delivery, and those that achieve guaranteed output delivery. In this chapter, to keep things simple, we will focus mostly on protocols that do not attempt to achieve fairness or guaranteed output delivery. However, in Section 23.6.2, we will present a protocol that does achieve fairness.

What are the inputs contributed by corrupt parties? We need to address a subtle issue regarding the inputs of corrupt parties. Even if a corrupt party is initialized with some particular input, such a party may behave arbitrarily, and in particular, it can run the protocol using a different input. The notion of soundness allows for this, but it still requires that the behavior of a corrupt party is such that, no matter what it does, it behaves as if it runs the protocol with *some* specific (yet arbitrary) input. Moreover, the input independence property requires that this input is chosen independently of the honest parties' inputs.

23.1.2 Assumptions

To guarantee any of these security properties, protocols need to make certain assumptions.

Cryptographic assumptions. Not surprisingly, the security of a protocol will depend on various cryptographic assumptions. Most protocols require secure point-to-point communication channels, which can be implemented using the techniques in Chapter 9. Some protocols require nothing more than that.

Number of corrupt parties. Some protocols require that a majority, or even a super-majority, of parties are honest (a super-majority is when more than two thirds of the parties are honest). Other protocols work no matter how many parties are corrupt. In this chapter, we will see various protocols with different assumptions on the number of corrupt parties.

Communication assumptions. Some protocols require that communication among parties is **synchronous**, which essentially means that all messages sent from one honest party to another honest party will be delivered within a fixed amount of time. This implies that if an honest party "times out" waiting for a message from another party, the honest party may safely conclude that the other party is corrupt. Other protocols make no such assumptions, and work using a purely **asynchronous** network, where messages may be arbitrarily delayed.

Protocols that work in an asynchronous communication model are clearly more robust than those that rely on a synchronous communication model. In this chapter, *we will focus exclusively on protocols in the asynchronous communication model.*

Types of corruption. Some protocols are secure even if the corrupt parties may behave in an arbitrary malicious way, which includes the possibility of colluding with one another. We call security in this sense **security against malicious adversaries**. This type of security protects against Failure Scenarios 1 and 2 in the above auction example (as well as Failure Scenario 3 if the protocol provides guaranteed output delivery, or at least fairness).

A weaker model of security that is sometimes considered is **security against honest-but-curious adversaries**. This model of security only protects against the following type of attack: while the protocol is running, all parties — even corrupt ones — faithfully follow the protocol; however, corrupt parties may leak their internal state to an adversary. In this type of attack, the main security property at issue is *privacy*: after seeing this internal state information (in addition to all network traffic), the adversary should still not have any additional information about honest parties' inputs. (This type of security only protects against Failure Scenario 1 in the above auction example.)

Protocols that are secure against malicious adversaries are clearly more robust than those that are only secure against honest-but-curious adversaries. In this chapter, our main goal is to describe protocols that are secure against malicious adversaries; however, when introducing a new technique, we will typically start by describing a protocol that is only secure against honest-but-curious adversaries, and then show how to enhance it to achieve security against malicious adversaries. Presenting techniques in this way makes them a bit easier to understand.

We stress that security against honest-but-curious adversaries is typically not an acceptable security model in the real world, and that our main reason for presenting such protocols is chiefly as an aid to understanding, and as a stepping stone to protocols that are secure against malicious adversaries.

Even with malicious adversaries, one can consider two types: those that decide which parties to corrupt at the very beginning of the protocol execution, or those who use a more adaptive strategy, perhaps watching network traffic for a while, and based on this, corrupting one party, and then based on information learned after that, corrupting another party, and so on. If the adversary is of the first type, we say the corruptions are **static**, and otherwise, we say the corruptions are **adaptive**. Security against adaptive corruptions is stronger, and therefore, more desirable; however, for simplicity, we will focus exclusively on static corruptions.

23.1.3 How to define security formally

The way to define security formally is to say that an attack on the protocol in the "real world" is equivalent to some attack on the protocol in an "ideal world" in which no damage can be done. In the ideal world, the protocol is implemented using a trusted party to which all parties (both honest and corrupt) submit inputs, and from which all parties (both honest and corrupt) obtain their designated outputs. The trusted party itself cannot be corrupted.

In somewhat more detail, in such a definition, we first carefully describe the details of the real-world protocol execution, including the powers of the adversary, as well as the characteristics of the communication network. In terms of the assumptions discussed in Section 23.1.2, in the definition we present, we will mainly be assuming an *asynchronous* communication network, and a *malicious* adversary who *statically* corrupts various parties.

We then describe the details of the ideal-world execution, including the behavior of the trusted party who evaluates the function.

The definition of security then basically says: for every efficient adversary $\mathcal{A}$ in the real world, there exists an "equivalent" efficient adversary $\mathcal{S}$ in the ideal world (usually called a simulator).

Since there is no possible attack in the ideal world, there is no possible attack in the real world. In particular, the informal notions of *privacy*, *soundness*, and *input independence* are implied by this type of definition. Indeed, these informal notions are clearly attained in the ideal world, and therefore, because the real world and ideal worlds are "equivalent", they must also be attained in the real world. Clearly, to make such an argument more rigorous, we need to have a more rigorous definition of what "equivalent" means, and we will give one later in the chapter in Section 23.5. The formal definition of security we present in Section 23.5 will *not* provide for any assurance of *guaranteed output delivery* or *fairness*.

23.1.4 Other applications of MPC

Before going any further, we present a few more examples of where MPC can be helpful.

Elections. Instead of an auction service, we may consider another type of service, such as an *election service*. Without MPC, parties submit votes to a trusted server T who tallies the votes and announces the winner (without revealing any individual votes). Using MPC, the goal is to implement T as a distributed system of servers that remains secure even if a small number of these servers is hacked. Note that in Section 20.3.1 we considered some aspects of such an election protocol. While the techniques presented there are useful, they do not completely solve the problem. Note that like the auction example, guaranteed output delivery is important, otherwise a single misbehaving server could prevent the results of the election from being produced.

Privacy-preserving mining of medical data. Several hospitals store medical data on a large number of patients. A scientist wishes to carry out a medical study. The goal of the study is to find out if there is a significant correlation among several variables represented in the data stored by the hospitals. Because of privacy concerns and regulation, the hospitals do not wish to share any data with each other. One solution to this problem is to have the hospitals and the scientist run an MPC protocol that allows the scientist to test her hypothesis, without any hospital revealing any data to the scientist or to any other hospital. Note that unlike the auction and voting examples, neither guaranteed output delivery nor fairness are essential in this example. In fact, because the hospitals are believed to be honest, but are not allowed to see all the data, it may even be sufficient to use a protocol that only provides security against honest-but-curious adversaries.

Privately detecting misbehavior in the financial system. Banks are frequently tasked with identifying misbehavior, such as money laundering. Abstractly, one can think of financial transactions as a graph, where the vertices are accounts, and every funds transfer is a directed edge in the graph. Every bank only has a partial view of the graph: it only sees transfers that originate or terminate in an account held at that bank. Detecting a money laundering pattern, possibly spanning accounts at multiple banks in multiple countries, requires a global view of the graph. Yet, due to privacy concerns and regulation, banks cannot share this data with each other, or with any single government. Instead, the banks can run an MPC protocol with each other to look for a global pattern in the transaction graph, without learning any other information about the graph. Here guaranteed output delivery is important, since otherwise, a single corrupt bank may prevent all honest banks from learning the results.

Protecting a secret key by splitting it. Consider a challenge-response identification protocol based on a message authentication code, such as the CRYPTOcard protocol described in Section 18.6.1. For each user, the server has to store a secret key used for a message authentication code. For concreteness, let us assume that the message authentication code is implemented as a PRF F defined over $(\mathcal{K}, \mathcal{C}, \mathcal{T})$, where $\mathcal{K} = \{0,1\}^{\ell}$. It is essential that the key $k \in \mathcal{K}$ is never revealed to an attacker. To make this less likely, the server does not store the key — in fact, the key is never stored in any one machine. Rather, the key is split into pieces, or "shares", k_1 and k_2, so that $k = k_1 \oplus k_2$. The server S only stores the share k_1, and an auxiliary server, S', stores the other share k_2. Now, to run the identification protocol, the server S generates a random challenge $c \in \mathcal{X}$, sends this to the user and receives a response $t \in \mathcal{Y}$. To validate the user, S must now compute $F(k, c)$ and test whether this is equal to t. To do this, the two servers S and S' will run an MPC protocol that will allow S to obtain $F(k_1 \oplus k_2, c)$, based on the input k_1 provided by S, and the input k_2 provided to S' — in this example, the value c can be viewed as a publicly known value, rather than a private input. Note that unlike the auction and voting examples, neither guaranteed output delivery nor fairness are essential in this example.

This application may seem familiar: it is very similar to threshold cryptography discussed in Chapter 22. In fact, threshold cryptography is a special case of MPC — parties who are holding shares of a secret key wish to use the key without revealing anything else about their shares. In Chapter 22 we saw that for certain algebraic signature and decryption schemes, one can design custom threshold protocols that are very simple and efficient, without using the full machinery of MPC. For a PRF such as AES, one needs MPC to securely distribute the key (although, see Exercise 4.28).

Federated machine learning. Consider an organization that wants to train a machine learning (ML) model to recommend movies. For training data, it wants to use the combined preferences from a large group of people. Many people, however, are unwilling to reveal their preferences, lest they be judged for their taste in movies. Instead, the organization could run an MPC protocol with the group of people so that the final ML model is available to everyone, and nothing is revealed about the participant's individual preferences other than the model. This approach to training a model is called **federated machine learning**. It enables a group of contributors to train a model without revealing their personal data. Of course, one has to ensure that the resulting ML model does not leak the underlying training data, and this is often addressed by an appropriate use of statistical techniques to introduce noise into the model. It is a good reminder that MPC does not solve all privacy problems in the world, and that often, additional techniques are needed.

Zero knowledge. In Chapter 20 we saw zero knowledge protocols, where the prover has a statement x and a witness w, and the verifier only has the statement x. The prover wants to convince the verifier that there exists a valid witness w for x, namely that $\mathcal{R}(w, x) = 1$, for an agreed upon relation $\mathcal{R}$. This is a special case of two party computation, where the verifier's output is $\mathcal{R}(w, x) \in \{0, 1\}$ and the prover's output is nil. The protocol needs to be private to ensure that the verifier learns nothing about the witness, and maliciously secure to ensure that a cheating prover cannot fool the verifier.

23.2 Securely evaluating arithmetic circuits

Recall the general goal. We have a function f that takes n inputs and produces m outputs:

$$(y_1, \ldots, y_m) = f(x_1, \ldots, x_n).$$

We also have N parties, $P_1, \ldots, P_N$. Their goal is to run a protocol where each input value x_i is contributed by one of the parties, and each output y_j is obtained by one or more of the parties.

Ultimately, we want to design an efficient protocol for this problem that provides *privacy*, *soundness*, and *input independence*, as discussed in Section 23.1.1. Our first examples of such protocols will assume that the function f is represented in terms of an *arithmetic circuit*.

23.2.1 Arithmetic circuit evaluation

Let q be a prime. An **arithmetic circuit** for $\mathbb{Z}_q$ is a circuit with four types of gates: addition, multiplication, constant addition, and scalar multiplication.

- An *addition gate* takes two inputs, $x, y \in \mathbb{Z}_q$ and produces a single output $z = x + y \in \mathbb{Z}_q$.

- A *multiplication gate* takes two inputs, $x, y \in \mathbb{Z}_q$ and produces a single output $z = x \cdot y \in \mathbb{Z}_q$.

- A *constant addition gate* takes one input $x \in \mathbb{Z}_q$ and produces a single output $z = x + c \in \mathbb{Z}_q$, where $c \in \mathbb{Z}_q$ is a constant associated with the gate.

- A *scalar multiplication gate* takes one input $x \in \mathbb{Z}_q$ and produces a single output $z = cx \in \mathbb{Z}_q$, where $c \in \mathbb{Z}_q$ is a constant associated with the gate.

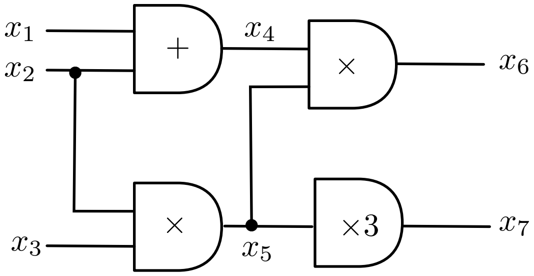


Figure 23.2: An arithmetic circuit

The input to a gate may be an input to the circuit or the output of some other gate. There is no restriction on the "fan out" of a gate: the output of a gate may be used as an input in any number of gates. The output of some gates are designated as circuit outputs.

Fig. 23.2 shows a simple arithmetic circuit. This particular circuit has three inputs, x_1, x_2, and x_3, and two outputs x_6 and x_7. For each input to the circuit, and each gate output, we have drawn a "wire" on which the value input to the circuit or computed by the gate is carried. Besides the three input wires and two output wires, there are two internal wires, which carry the internal values x_4 and x_5. In general, such a circuit should contain no cycles. This guarantees that once the values carried on the circuit's input wires are determined, we can calculate the values carried on all of the remaining wires in the circuit by evaluating the gates one at a time, in some fixed order (known as a "topological ordering"), so that by the time we evaluate a particular gate, the values carried on its input wires have already been calculated.

In the example in Fig. 23.2, if we feed input values x_1, x_2, and x_3 to the circuit, then the values on the remaining wires may be computed as follows:

$$\begin{aligned}
&x_4 \leftarrow x_1 + x_2\\
&x_5 \leftarrow x_2 \cdot x_3\\
&x_6 \leftarrow x_4 \cdot x_5\\
&x_7 \leftarrow 3 \cdot x_5
\end{aligned}$$

where x_6 and x_7 are the outputs of the circuit.

Every output of an arithmetic circuit can be written as a multivariate polynomial over $\mathbb{Z}_q$, where the variables of the polynomial are the inputs to the circuit. For example, in Fig. 23.2 we see that

$$x_6 = f(x_1, x_2, x_3) := (x_1 + x_2) \cdot x_2 \cdot x_3.$$

However, an arithmetic circuit is more than just a multivariate polynomial: the circuit gives a specific recipe for evaluating the polynomial.

As we shall see, the efficiency of the protocols we design for secure arithmetic circuit evaluation will depend on the size of the circuit — primarily on the number of multiplication gates in the circuit.

Restricting ourselves to arithmetic circuits is not a substantive restriction. First, from a theoretical point of view, any function that can be efficiently computed by an algorithm, can be translated into an arithmetic circuit whose size is polynomial in the running time of the algorithm and the size of the input. Second, from a more practical point of view, once we construct an MPC protocol for arithmetic circuits, there are a number of techniques that can leverage the protocol to efficiently construct an MPC protocol for any polynomial time computation.

23.2.2 Beaver's protocol: an honest-but-curious 2.5-party protocol

We shall present a protocol that allows two parties to securely evaluate an arithmetic circuit. As presented, the protocol is not exactly a two-party protocol: it makes use of a pre-processing phase that involves a third party, who we call the *dealer*. This pre-processing phase can be done before either of the two parties receive their inputs to the protocol. The dealer provides some data to the two parties that they will use when they eventually run the protocol; however, the dealer does not otherwise participate in the protocol. Hence, we call this a "2.5-party" protocol.

If needed, one can modify the protocol to eliminate the dealer, so that the protocol becomes a true 2-party protocol (see Exercise 23.3). However, it is easiest to describe the protocol using a dealer.

We want to design a protocol that lets two parties, call them P_1 and P_2, securely evaluate an arithmetic circuit (with the help of the dealer). We assume the circuit itself is fixed and publicly known. We assume the circuit takes a number of inputs $x_1, \ldots, x_n$ and produces some number of outputs $y_1, \ldots, y_m$, where each x_i and y_j is an element of $\mathbb{Z}_q$, for some prime q. Some inputs will be supplied by P_1 and the others will be supplied by P_2. Some outputs will be obtained by P_1 and some will be obtained by P_2 — some outputs may be obtained by both P_1 and P_2. The goal is to design a protocol that allows P_1 and P_2 to securely evaluate the circuit. As discussed in Section 23.1.1, this means that the protocol provides *privacy*, *soundness*, and *input independence*:

privacy: neither party learns anything about the other party's inputs (except for information that is inherently revealed by the outputs);

soundness: honest parties compute correct outputs (if they compute any output at all);

input independence: both parties must choose their inputs independently of the other's.

Our first attempt at such a protocol will only provide security against honest-but-curious adversaries. That is, we assume that both parties correctly follow the protocol. In this case, the main security concern is *privacy.*

High level description of the protocol. Here is a very high-level idea behind the protocol. Imagine that we evaluated the circuit directly, using inputs as supplied by P_1 and P_2. As discussed above, once all of the inputs to the circuit have been supplied, each wire in the circuit carries some particular value $x \in \mathbb{Z}_q$. Suppose that instead of having either P_1 or P_2 store the value x, they each store a "share" of x. Namely, P_1 stores a value $x_1 \in \mathbb{Z}_q$, and P_2 stores a value $x_2 \in \mathbb{Z}_q$, such that $x = x_1 + x_2$, but neither x_1 nor x_2 alone leaks any information about x.

The protocol begins with the parties P_1 and P_2 sharing the values of their input wires. For example, if x is one of P_1's inputs, then P_1 sends a random $x_2 \in \mathbb{Z}_q$ to P_2 and keeps $x_1 = x - x_2$ to itself. This way, $x_1 + x_2 = x$ where P_1 holds the share x_1 and P_2 holds the share x_2. Observe that party P_2's share x_2 reveals nothing to P_2 about P_1's input x.

Next, they process the gates in the circuit one by one. If the left input wire to a certain gate carries the value x, and the right input wire carries the values y, then by construction, $x = x_1 + x_2$ and $y = y_1 + y_2$, where P_1 holds the shares x_1, y_1 and P_2 holds the shares x_2, y_2. Their goal is to compute a sharing $z = z_1 + z_2$ of the output wire from the gate, so that P_1 holds the share z_1 and P_2 holds the share z_2. We will show below that for addition, scalar multiplication, and constant addition gates, this is straightforward. For a multiplication gate, creating a sharing $z = z_1 + z_2$ takes more work, and this is the only situation where the dealer is needed.

Eventually P_1 and P_2 obtain a sharing of all the output wires. If $x = x_1 + x_2$ is a sharing of an output wire that P_1 is supposed to learn, then P_2 sends its share x_2 to P_1, and P_1 computes $x_1 + x_2$ to obtain the desired output x. They do the same for P_2's output wires (but with their roles reversed).

23.2.2.1 The details

Now, let's see the details. We start by describing how parties P_1 and P_2 process the input wires, and then explain how they process each of the four types of gates (addition, scalar multiplication, constant addition, and multiplication) and output wires. The dealer is only needed when processing a multiplication gate.

Processing input wires. Suppose that the value $x \in \mathbb{Z}_q$ is carried on an input wire for an input provided by P_1. In this case, P_1 can choose x_2 at random in $\mathbb{Z}_q$ and set $x_1 \leftarrow x - x_2$. Now, P_1 sends x_2 to P_2, and keeps x_1 for itself. Clearly, x_1 and x_2 have the desired properties: $x = x_1 + x_2$, but neither x_1 nor x_2 alone leak any information about x.

Processing addition gates. Now suppose we have an addition gate with inputs x and y. Assume that P_1 and P_2 have already computed shares of x and y as described above: P_1 holds x_1, y_1 and P_2 holds x_2, y_2, where $x = x_1 + x_2$ and $y = y_1 + y_2$. They want to compute a sharing of $z = x + y$. But this is easy to do: P_1 simply computes $z_1 \leftarrow x_1 + y_1$ and P_2 computes $z_2 \leftarrow x_2 + y_2$. No interaction is required. Clearly, z_1 and z_2 is a sharing of z, in the sense that $z = z_1 + z_2$. Moreover, if x_1 and y_1 leaks no information to P_1 about x or y, then $z_1 = x_1 + y_1$ cannot leak any information to P_1 about z, simply because P_1 computed z_1 locally. Similarly, z_2 cannot leak any information to P_2 about z.

Processing scalar multiplication gates. Now suppose we have a scalar multiplication gate with input x. Assume that P_1 and P_2 have already computed shares of x as described above: P_1 holds x_1 and P_2 holds x_2, where $x = x_1 + x_2$. They want to compute a sharing of $z = cx$ for a constant c associated with the gate. To do this, party P_1 locally computes $z_1 \leftarrow cx_1$, and P_2 locally computes $z_2 \leftarrow cx_2$. Clearly, z_1 and z_2 is a sharing of z, in the sense that $z = z_1 + z_2$.

Processing constant addition gates. Now suppose we have a constant addition gate with input x. Assume that P_1 and P_2 have already computed shares of x as described above: P_1 holds x_1 and P_2 holds x_2, where $x = x_1 + x_2$. They want to compute a sharing of $z = x + c$ for a constant c associated with the gate. To do this, party P_1 just locally computes $z_1 \leftarrow x_1 + c$, and P_2 locally computes $z_2 \leftarrow x_2$. Clearly, z_1 and z_2 is a sharing of z, in the sense that $z = z_1 + z_2$.

Processing multiplication gates. Handling multiplication gates is the tricky and interesting part. To make the protocol work for multiplication gates, we will make use of a "dealer" D, which (as discussed above) provides some information to P_1 and P_2 as part of a pre-processing phase. Specifically, D will supply P_1 and P_2 with random sharings of triples $a, b, c \in \mathbb{Z}_q$, such that a and b are randomly chosen, and $c = a \cdot b$. More precisely,

- D gives P_1 a_1, b_1, c_1 in $\mathbb{Z}_q$,
- D gives P_2 a_2, b_2, c_2 in $\mathbb{Z}_q$,

where $a_1, b_1, c_1, a_2, b_2, c_2$ are random, subject to

$$(a_1 + a_2)(b_1 + b_2) = (c_1 + c_2).$$

We call this a **Beaver triple sharing**.

The dealer D may generate a Beaver triple sharing, for example, by first choosing a_1, b_1, a_2, b_2, c_2 in $\mathbb{Z}_q$ at random, and then computing $c_1 \leftarrow (a_1 + a_1)(b_1 + b_2) - c_2$. The dealer gives P_1 the values a_1, b_1, c_1, and P_2 the values a_2, b_2, c_2.

The dealer genereates one such Beaver triple sharing for each multiplication gate in the circuit. This can be done in a pre-processing phase, before any inputs are known to either party. The dealer D plays no other role in the protocol. As we are still focused on the honest-but-curious security model, we shall assume (for now) that the dealer is honest as well, and follows the above protocol exactly.

So now consider a multiplication gate with inputs x and y. Assume that P_1 and P_2 have already computed shares of x and y as described above: P_1 holds x_1, y_1 and P_2 holds x_2, y_2, where $x = x_1 + x_2$ and $y = y_1 + y_2$. They want to compute a sharing of $z = x \cdot y$. Moreover, assume that P_1 and P_2 have a Beaver triple sharing, that is, they have shares of a, b, c, where $c = ab$, as supplied by the dealer D. That is, P_1 holds a_1, b_1, c_1, and P_2 holds a_2, b_2, c_2, where $a = a_1 + a_2$, $b = b_1 + b_2$, and $c = a \cdot b = c_1 + c_2$.

To understand how the protocol works, observe that

$$\begin{aligned} z = xy = (x - a + a)(y - b + b) &= (\underbrace{(x-a)}_{=:u} + a)(\underbrace{(y-b)}_{=:v} + b) \\ &= uv + ub + va + c. \end{aligned}$$

So the idea is that using the logic for addition and scalar multiplication gates, each party locally computes its share of $u = x - a$ and $v = y - b$. After this, the two parties exchange their shares of u and v with each other so that they both can compute the values u and v. Now, since each party knows that values u and v, and has a share of the values a, b, and c, they can each locally compute a share of z from the equation

$$z = uv + ub + va + c$$

and the logic for processing addition, scalar multiplication, and constant addition gates. The details of the protocol are given in Fig. 23.3. Intuitively, the only thing each party learns by doing this are the values u and v; moreover, u and v are essentially "one-time pad" encryptions of x and y, so each party actually learns nothing at all about the values x and y.

P_1 : input $x_1, y_1, a_1, b_1, c_1 \in \mathbb{Z}_q$		P_2 : input $x_2, y_2, a_2, b_2, c_2 \in \mathbb{Z}_q$
$u_1 \leftarrow x_1 - a_1$ $v_1 \leftarrow y_1 - b_1$		$u_2 \leftarrow x_2 - a_2$ $v_2 \leftarrow y_2 - b_2$
	$\xrightarrow{\quad u_1, v_1 \quad}$	
	$\xleftarrow{\quad u_2, v_2 \quad}$	
$u \leftarrow u_1 + u_2$, $v \leftarrow v_1 + v_2$ $z_1 \leftarrow uv + ub_1 + va_1 + c_1$		$u \leftarrow u_1 + u_2$, $v \leftarrow v_1 + v_2$ $z_2 \leftarrow ub_2 + va_2 + c_2$

Figure 23.3: Protocol for processing a multiplication gate

Processing output wires. Finally, suppose some output wire carries a value x. Suppose P_1 and P_2 have already computed a sharing of x, so P_1 holds x_1 and P_2 holds x_2, where $x = x_1 + x_2$. Moreover, suppose this value x is designated as an output to be obtained by P_1. Then in this case, P_2 sends its share x_2 of x to P_1, and P_1 computes $x \leftarrow x_1 + x_2$. Similarly, if x is designated as an output to be obtained by P_2, then P_1 sends its share x_1 of x to P_2, and P_2 computes $x \leftarrow x_1 + x_2$.

Putting everything together. So the whole protocol runs as follows.

- *Pre-processing phase:* The dealer D first distributes shares of Beaver triples to P_1 and P_2, as described above, one Beaver triple per multiplication gate. The dealer is no longer needed after this phase.
- *Input phase:* The parties P_1 and P_2 process each of the input wires to create a sharing of all the inputs to the circuit.
- *Evaluation phase:* The parties P_1 and P_2 proceed gate by gate. For each gate, as soon as they compute the shares of the values carried on the input wires for that gate, they can compute the shares of the value carried on the output wire of that gate. Viewing the circuit as a directed acyclic graph, the parties can use any topological ordering of the vertices in order to schedule the order in which gates are processed.
- *Output phase:* Finally, the parties P_1 and P_2 process each of the output wires.

In the evaluation phase, every multiplication gate requires P_1 to send two $\mathbb{Z}_q$ elements to P_2, and vice versa. To reduce the “round complexity” of the protocol, the circuit can be processed level by level, processing all the multiplication gates in a single level in parallel.

Performance. The amount of data exchanged between the three parties is proportional to the number of multiplication gates in the circuit. The number of rounds of communication between the parties P_1 and P_2 is proportional to the longest path of multiplication gates in the circuit, also known as the *multiplication depth* of the circuit. The dealer is only needed in a pre-processing phase before the party’s inputs are known, and need not participate in the actual joint evaluation of the circuit.

An interesting aspect of the protocol is that it does not rely on cryptographic assumptions. Security when the parties honestly follow the protocol is unconditional.

23.2.3 Abstracting Beaver's 2.5-party protocol

Before going further, we "repackage" Beaver's 2.5-party protocol by introducing a simple layer of abstraction. This will make it easier to generalize the protocol to handle more parties, and to enhance the protocol so that it is secure against malicious adversaries, rather than just honest-but-curious adversaries.

23.2.3.1 Sharings

For a value $x \in \mathbb{Z}_q$, we denote by $[x]$ a sharing of x between P_1 and P_2, so that P_1 holds x_1 and P_2 holds x_2, where $x = x_1 + x_2$. We may write $[x] = (x_1, x_2)$, but it is also to be understood that P_1 holds x_1 and P_2 holds x_2.

We describe some low-level sub-protocols for working with such sharings.

Open a sharing $[x]$ to P_i: if P_1 and P_2 hold a sharing $[x] = (x_1, x_2)$, then P_{3-i} sends x_{3-i} to P_i, and P_i computes the value $x \leftarrow x_1 + x_2$.

Add two sharings: $[z] \leftarrow [x] + [y]$**:** if P_1 and P_2 hold a sharing $[x] = (x_1, x_2)$ and $[y] = (y_1, y_2)$, then they both locally compute a sharing $[z] = (z_1, z_2)$ of $z = x + y$, by simply adding their shares: P_1 computes $z_1 \leftarrow x_1 + y_1$, and P_2 computes $z_2 \leftarrow x_2 + y_2$.

Multiply a sharing by a constant: $[z] \leftarrow c[x]$**:** if P_1 and P_2 hold a sharing $[x] = (x_1, x_2)$, and $c \in \mathbb{Z}_q$ is a publicly known value, then they both locally compute a sharing $[z] = (z_1, z_2)$ of $z = cx$, by simply multiplying their shares by c: P_1 computes $z_1 \leftarrow cx_1$, and P_2 computes $z_2 \leftarrow cx_2$.

Add a constant to a sharing: $[z] \leftarrow [x] + c$**:** if P_1 and P_2 hold a sharing $[x] = (x_1, x_2)$, and $c \in \mathbb{Z}_q$ is a publicly known value, then they both locally compute a sharing $[z] = (z_1, z_2)$ of $z = x + c$ by the following local computation: P_1 computes $z_1 \leftarrow x_1 + c$, and P_2 computes $z_2 \leftarrow x_2$.

Note that among these sub-protocols, the only one that requires any interaction is the *open* sub-protocol. The others involve only local computations.

23.2.3.2 Pre-processing phase

Recall that Beaver's 2.5-party protocol starts with a pre-processing phase in which a dealer D distributes sharings to P_1 and P_2 of various values.

Protocol 23.1 (The dealer's protocol). To create a **random sharing** $[x] = (x_1, x_2)$ of a value $x \in \mathbb{Z}_q$, the dealer D simply chooses $x_1 \in \mathbb{Z}_q$ at random, and sets $x_2 \leftarrow x - x_1 \in \mathbb{Z}_q$, and gives x_1 to P_1 and x_2 to P_2. The dealer distributes shares of a number of random elements and Beaver triples:

- A **singleton sharing** is a random sharing $[a]$ of a random element $a \in \mathbb{Z}_q$.

- A **Beaver triple sharing** is a triple of random sharings $([a], [b], [c])$, where $a, b \in \mathbb{Z}_q$ are chosen at random, and $c = ab$.

The dealer generates one singleton sharing $[a]$ for each input wire in the circuit, and one Beaver triple sharing $([a], [b], [c])$ for each multiplication gate in the circuit. □

23.2.3.3 Beaver's 2.5 party protocol

We can now state the main protocol:

Protocol 23.2 (Beaver's 2.5-party protocol). After obtaining the required singleton and Beaver triple sharings from the dealer D, parties P_1 and P_2 first process input wires, then process gates in a topological order, and then process output wires, as follows.

Input wire for P_i: to produce a sharing $[x]$ of one of P_i's inputs $x \in \mathbb{Z}_q$, a singleton sharing $[a]$ from the dealer is used as follows:

1. execute open $[a]$ to P_i;
2. P_i sends $\delta \leftarrow x - a$ to P_{3-i};
3. execute $[x] \leftarrow [a] + \delta$.

Addition gate: to add $[x]$ and $[y]$, obtaining $[z] = [x + y]$, the parties execute $[z] \leftarrow [x] + [y]$.

Scalar multiplication gate: to multiply $[x]$ by a constant c, obtaining $[z] = [cx]$, the parties execute $[z] \leftarrow c[x]$.

Constant addition gate: to add a constant c to $[x]$, obtaining $[z] = [x + c]$, the parties execute $[z] \leftarrow [x] + c$.

Multiplication gate: to multiply $[x]$ and $[y]$, obtaining $[z] = [xy]$, a Beaver triple sharing $([a], [b], [c])$ from the dealer is used as follows:

1. execute $[u] \leftarrow [x] - [a]$;
2. execute $[v] \leftarrow [y] - [b]$;
3. execute open $[u]$ and open $[v]$ to both P_1 and P_2;
4. execute $[z] \leftarrow uv + u[b] + v[a] + [c]$.

Output wire for P_i: to give to P_i the value x of a sharing $[x]$, execute open $[x]$ to P_i. □

This protocol is essentially the same as that described in Section 23.2.2. The only substantive difference is the logic of processing the inputs. In this protocol, we make use of one singleton sharing $[a]$ per input, generated in the first bullet of Protocol 23.1. This $[a]$ is used to share an input x between the two parties, so that the two parties end up with a random sharing of x. The formulation here is more suitable to further generalizations and enhancements, as we shall see. Observe that the only interactive steps in the protocol (not including the pre-processing step), are Steps 1 and 2 of the sub-protocol for processing input wires, and Step 3 of the sub-protocol for processing multiplication gates, and the sub-protocol for processing output wires.

23.2.4 A maliciously secure version of Beaver's 2.5-party protocol

We now present a variant of Beaver's 2.5-party protocol with pre-processing that is secure against malicious adversaries. As before, this protocol involves two parties, P_1 and P_2, as well as a dealer D who distributes information to P_1 and P_2 in a pre-processing phase. The protocol is secure even if one of P_1, P_2, or D is corrupt. Again, as discussed in Section 23.1.1, this means that the protocol provides *privacy*, *soundness*, and *input independence.*

One can ask for more, and require that the protocol be secure even if P_1 and D are corrupt, or P_2 and D are corrupt. While the protocol can be made secure against two corrupt players, here we will only consider the single corruption case. This is called the *honest majority* setting, because we are assuming that at least two of the three parties are honest.

23.2.4.1 Authenticated sharings

The idea is to use *authenticated sharings.* Intuitively, an authenticated sharing $[\![x]\!]$ of a value $x \in \mathbb{Z}_q$ has all the properties of an ordinary sharing of x, but the operation of opening $[\![x]\!]$ to either P_1 or P_2 is *authenticated*, so that, say, if P_1 opens $[\![x]\!]$ to P_2, the value obtained by P_2 is guaranteed to be x (with overwhelming probability). We will build such authenticated sharings from ordinary sharings using a very simple type of message authentication code. Specifically, as part of the pre-processing phase, the dealer will distribute sharings $[K^{(1)}]$ and $[K^{(2)}]$ to P_1 and P_2, where $K^{(1)}$ and $K^{(2)}$ are random elements of $\mathbb{Z}_q$. Moreover, as part of the pre-processing phase, P_1 and P_2 will execute a very simple interactive sub-protocol that allows each party P_i to reliably obtain the value $K^{(i)}$, without learning anything about $K^{(3-i)}$.

An *authenticated sharing* $[\![x]\!]$ of a value $x \in \mathbb{Z}_q$ consists of three ordinary sharings $([x], [x^{(1)}], [x^{(2)}])$. Concretely, P_1 has $(x_1, x_1^{(1)}, x_1^{(2)})$ and P_2 has $(x_2, x_2^{(1)}, x_2^{(2)})$, such that $x_1+x_2 = x$, $x_1^{(1)} + x_2^{(1)} = x^{(1)}$, and $x_1^{(2)} + x_2^{(2)} = x^{(2)}$. Such an authenticated sharing $[\![x]\!] = ([x], [x^{(1)}], [x^{(2)}])$ of $x \in \mathbb{Z}_q$ is called *valid* if

$$x^{(1)} = K^{(1)}x \qquad \text{and} \qquad x^{(2)} = K^{(2)}x.$$

We now show how the low-level subprotocols required for Beaver's 2.5-party protocol can be adapted from ordinary sharings to authenticated sharings. We describe the protocols in terms of the corresponding sub-protocols for ordinary sharings.

Open a sharing $[\![x]\!]$ to P_i: if P_1 and P_2 hold an authenticated sharing $[\![x]\!] = ([x], [x^{(1)}], [x^{(2)}])$, then do the following:

1. execute open $[x]$ and $[x^{(i)}]$ to P_i;
2. P_i checks that $K^{(i)}x = x^{(i)}$; if not, P_i aborts the protocol.

Add two sharings: $[\![z]\!] \leftarrow [\![x]\!] + [\![y]\!]$: if P_1 and P_2 hold an authenticated sharing $[\![x]\!] = ([x], [x^{(1)}], [x^{(2)}])$ and $[\![y]\!] = ([y], [y^{(1)}], [y^{(2)}])$, then they locally compute an authenticated sharing $[\![z]\!] = ([z], [z^{(1)}], [z^{(2)}])$ of $z = x + y$ by executing:

$$[z] \leftarrow [x] + [y], \quad [z^{(1)}] \leftarrow [x^{(1)}] + [y^{(1)}], \quad [z^{(2)}] \leftarrow [x^{(2)}] + [y^{(2)}].$$

Multiply a sharing by a constant: $[\![z]\!] \leftarrow c[\![x]\!]$: if P_1 and P_2 hold an authenticated sharing $[\![x]\!] = ([x], [x^{(1)}], [x^{(2)}])$, and $c \in \mathbb{Z}_q$ is a publicly known value, then they locally compute an authenticated sharing $[\![z]\!] = ([z], [z^{(1)}], [z^{(2)}])$ of $z = cx$ by executing:

$$[z] \leftarrow c[x], \quad [z^{(1)}] \leftarrow c[x^{(1)}], \quad [z^{(2)}] \leftarrow c[x^{(2)}].$$

Add a constant to a sharing: $[\![z]\!] \leftarrow [\![x]\!] + c$**:** if P_1 and P_2 hold an authenticated sharing $[\![x]\!] = ([x], [x^{(1)}], [x^{(2)}])$, and $c \in \mathbb{Z}_q$ is a publicly known value, then they locally compute an authenticated sharing $[\![z]\!] = ([z], [z^{(1)}], [z^{(2)}])$ of $z = x + c$ by executing:

$$[z] \leftarrow [x] + c, \quad [z^{(1)}] \leftarrow [x^{(1)}] + c[K^{(1)}], \quad [z^{(2)}] \leftarrow [x^{(2)}] + c[K^{(2)}].$$

One can easily verify that for the *addition*, *scalar multiplication*, and *constant addition* sub-protocols, if the two parties start with valid authenticated sharings of the inputs, and they both follow the protocol, they end up with a valid authenticated sharing of the output.

Now consider the *open* sub-protocol. Suppose the two parties start with a valid authenticated sharing $[\![x]\!]$ of a value x. If P_{3-i} follows the protocol, then P_i will not abort and will compute the value x. Now consider what happens if P_{3-i} does not follow the protocol. In this case, after Step 1 of the sub-protocol, P_i could end up with incorrect values $x + \delta$ and $x^{(i)} + \delta^{(i)}$, instead of x and $x^{(i)}$, where $\delta \neq 0$ or $\delta^{(i)} \neq 0$. We want to argue that P_i ends up with incorrect values with very small probability, namely, $1/q$. Indeed, if P_i does not abort, then the following equation holds:

$$K^{(i)}(x + \delta) = x^{(i)} + \delta^{(i)}, \tag{23.1}$$

and since $x^{(i)} = K^{(i)}x$, equation (23.1) is equivalent to

$$K^{(i)}\delta = \delta^{(i)}. \tag{23.2}$$

Moreover, we may assume that $K^{(i)}$ is independent of δ and $\delta^{(i)}$. It is clear that (23.2) holds with probability at most $1/q$: for any fixed $\delta, \delta^{(i)}$, if $\delta \neq 0$, then it holds with probability $1/q$, while if $\delta = 0$ and $\delta^{(i)} \neq 0$, then it holds with probability 0.

23.2.4.2 Pre-processing phase

We shall first present a protocol that is secure assuming that the dealer D is not corrupt, but remains secure if one of P_1 or P_2 is corrupt. Later, we will show how to protect against a dishonest dealer (but assuming that neither P_1 or P_2 is corrupt).

Protocol 23.3 (The dealer's protocol (authenticated sharings)). The dealer distributes a number of random sharings and authenticated sharings:

- **singleton sharings** of $[K^{(1)}]$ and $[K^{(2)}]$, for random $K^{(1)}, K^{(2)} \in \mathbb{Z}_q$;
- several ***authenticated*** **singleton sharings** $[\![a]\!]$, for random $a \in \mathbb{Z}_q$ — one for each input wire, plus two additional authenticated singleton sharings (which will be used in the *reliable key opening sub-protocol* described below);
- several ***authenticated*** **Beaver triple sharings** $([\![a]\!], [\![b]\!], [\![c]\!])$, where $a, b \in \mathbb{Z}_q$ are random, and $c = ab$ — one for each multipliplication gate. □

After running Protocol 23.3, parties P_1 and P_2 need to run a special "reliable key opening" sub-protocol that allows P_1 to obtain $K^{(1)}$ and P_2 to obtain $K^{(2)}$.

Protocol 23.4 (Reliable key opening sub-protocol). In this protocol, we reliably open $[K^{(i)}]$ to P_i using an authenticated singleton sharing $[\![a]\!] = ([a], [a^{(1)}], [a^{(2)}])$ from the dealer, as follows:

1. execute open $[a]$, $[a^{(i)}]$, and $[K^{(i)}]$ to P_i;

2. P_i checks that $K^{(i)}a = a^{(i)}$; if not, P_i aborts the protocol. □

Consider what happens if P_{3-i} does not follow the protocol. We will show that in this case, P_i aborts with probability $\geq 1 - 1/q$. Suppose that after opening the sharings for a, $a^{(i)}$, and $K^{(i)}$, party P_i ends up with values $a + \delta$, $a^{(i)} + \delta^{(i)}$, and $K^{(i)} + \epsilon$, where at least one of δ, $\delta^{(i)}$, or ϵ is not zero. If P_i does not abort, then the following equation holds:

$$(K^{(i)} + \epsilon)(a + \delta) = (a^{(i)} + \delta^{(i)}), \tag{23.3}$$

and since $a^{(i)} = K^{(i)}a$, equation (23.3) is equivalent to

$$K^{(i)}\delta + a\epsilon + \epsilon\delta = \delta^{(i)}. \tag{23.4}$$

Moreover, we may assume that $K^{(i)}$ and a are independent of each other and of δ, $\delta^{(i)}$, and ϵ. It is clear that (23.4) holds with probability at most $1/q$: for any fixed $\delta, \delta^{(i)}, \epsilon$, if $(\delta, \epsilon) \neq (0, 0)$, then it holds with probability $1/q$, while if $(\delta, \epsilon) = (0, 0)$ and $\delta^{(i)} \neq 0$, then it holds with probability 0.

23.2.4.3 Beaver's maliciously secure 2.5-party protocol (with a trusted dealer)

Using the setup we built, it is now quite easy to describe the maliciously secure version of Beaver's protocol. The protocol is as follows:

- *step 1:* P_1 and P_2 run the reliable key opening sub-protocol for both $K^{(1)}$ and $K^{(2)}$,
- step 2: P_1 and P_2 run Protocol 23.2 exactly as is, except we replace all ordinary sharings $[\cdot]$ with authenticated sharings $[\![\cdot]\!]$.

The net result is that by moving from ordinary sharings to authenticated sharings, the basic protocol becomes maliciously secure. A fine point:

> A party P_i must not run any *output wire* subprotocol until
>
> - it has received the value δ in Step 2 of the *input wire* subprotocol for every input to P_{3-i}, and
> - it has actually received all of its own inputs.

23.2.4.4 Security

We will formally analyze the security of this protocol later, in Section 23.5.9, after we have a more rigorous definition of security. For now, we give heuristic arguments as to why the protocol provides privacy, soundness, and input independence.

Privacy. Privacy holds, at least heuristically, assuming that the dealer is not corrupt. Each party P_i only sees random shares of values, rather than the values themselves, except at the end of the protocol when the output values are revealed.

Soundness. Suppose that P_1 is corrupt and D and P_2 are honest. By the above analysis (circa (23.4)), after the dealing phase and the reliable key opening phase, with overwhelming probability (assuming q is large enough), party P_2 holds the correct authentication key $K^{(2)}$, which remains unknown to P_1.

During the execution of the circuit evaluation protocol, by the above analysis (circa (23.2)), whenever P_1 opens a share to P_2, with overwhelming probability, party P_1 must open the share correctly. Thus, P_1 is essentially constrained to follow the protocol.

Now consider P_1's inputs. It can choose its inputs however it likes. However, it must effectively commit to each of its individual inputs x at the point in time that the corresponding input wire is processed. Recall the steps involved:

1. execute open $[\![a]\!]$ to P_1;
2. P_1 sends $\delta \leftarrow x - a$ to P_2;
3. execute $[\![x]\!] \leftarrow [\![a]\!] + \delta$.

Here, a is a value generated by the dealer. A corrupt P_1 can send any value δ it likes to P_2, but then its effective input is $x := a + \delta$. It is at this time that P_1 is fully committed to x. More precisely, the function computed by the circuit will be a function of x (as well as all of the other inputs provided by P_1 and P_2).

Input independence. One can also argue that the effective inputs provided by P_1 are independent of P_2's inputs. Intuitively, this follows from the fact that at the time P_1 commits to an effective input x as above, the value x is statistically independent of P_2's inputs.

23.2.4.5 Keeping the dealer honest

Now suppose that the dealer D is corrupt, and does not follow the protocol. Since we are assuming that at most one of P_1, P_2, or D is corrupt, then it must be the case that P_1 and P_2 are honest. So in this case, the only thing we need to do is to ensure that the sharings distributed by D satisfy the requisite relations: if they do not, the two honest parties P_1 and P_2 could compute incorrect outputs. We do not, however, have to worry about whether or not D chooses values according to any particular distribution.

The general version of the problem that we need to solve is as follows. The dealer sends to party P_1 values

$$a_{11}, \ldots, a_{m1}, \quad b_{11}, \ldots, b_{m1}, \quad c_{11}, \ldots, c_{m1} \quad \in \mathbb{Z}_q. \tag{23.5}$$

and to P_2 values

$$a_{12}, \ldots, a_{m2}, \quad b_{12}, \ldots, b_{m2}, \quad c_{12}, \ldots, c_{m2} \quad \in \mathbb{Z}_q. \tag{23.6}$$

The goal is to have D send P_1 and P_2 additional data that will allow P_1 and P_2 to verify that the following relation holds:

$$(a_{k1} + a_{k2})(b_{k1} + b_{k2}) = (c_{k1} + c_{k2}) \quad \text{for } k = 1, \ldots, m. \tag{23.7}$$

The security requirements are:

(D1) If D is corrupt and both P_1 and P_2 are honest, and both successfully complete the protocol without aborting, then with overwhelming probability, the relation (23.7) must hold.

(D2) If D is honest and one of P_1 or P_2 is corrupt, then the corrupt party should not learn anything about the honest party's values.

Here is the protocol:

Protocol 23.5 (Proving product relations).

1. Given values a_{ki}, b_{ki}, and c_{ki}, for $i = 1, 2$ and $k = 1, \ldots, m$, which satisfy (23.7), the dealer D performs the following local computations:

   (a) The dealer chooses values $a_{01}, a_{02}, b_{01}, b_{02} \in \mathbb{Z}_q$ at random, and chooses $c_{01}, c_{02} \in \mathbb{Z}_q$ at random, subject to $(a_{01} + a_{02})(b_{01} + b_{02}) = (c_{01} + c_{02})$.

   (b) The dealer runs a polynomial interpolation algorithm to obtain the unique polynomials $A_1(X)$, $A_2(X)$, $B_1(X)$, $B_2(X)$, each of degree at most m, such that

   $$A_i(k) = a_{ki} \quad \text{and} \quad B_i(k) = b_{ki} \qquad \text{for } i = 1, 2 \ \text{ and } \ k = 0, \ldots, m.$$

   (c) The dealer computes the polynomial $C(X) \leftarrow (A_1(X)+A_2(X))(B_1(X)+B_2(X))$, which has degree at most $2m$.

   (d) For $i = 1, 2$ and $k = m + 1, \ldots, 2m$, the dealer chooses $c_{k1}, c_{k2} \in \mathbb{Z}_q$ at random, subject to $c_{k1} + c_{k2} = C(k)$.

   (e) Finally, for $i = 1, 2$, the dealer sends to P_i the values

   $$a_{ki} \quad \text{and} \quad b_{ki} \quad \text{for } k = 0, \ldots, m$$

   and

   $$c_{ki} \quad \text{for } k = 0, \ldots, 2m.$$

2. Each party P_i (for $i = 1, 2$) runs a polynomial interpolation algorithm on the values it receives to obtain polynomials $A_i(X)$, $B_i(X)$, and $C_i(X)$, where $A_i(X)$ and $B_i(X)$ are of degree at most m, and $C_i(X)$ is of degree at most $2m$, such that

   $$A_i(k) = a_{ki} \quad \text{and} \quad B_i(k) = b_{ki} \quad \text{for } k = 0, \ldots, m$$

   and

   $$C_i(k) = c_{ki} \qquad k = 0, \ldots, 2m.$$

   Note that by construction, we have

   $$(A_1(X) + A_2(X))(B_1(X) + B_2(X)) = (C_1(X) + C_2(X)) \tag{23.8}$$

   if the dealer is honest.

3. Party P_1 chooses $r \in \mathbb{Z}_q \setminus \{0, \ldots, m\}$ at random, and sends it to P_2.

4. Party P_2 verifies that $r \in \mathbb{Z}_q \setminus \{0, \ldots, m\}$; if not, P_2 aborts.

5. Each party P_i (for $i = 1, 2$) sends the other party

   $$\alpha_i \leftarrow A_i(r), \quad \beta_i \leftarrow B_i(r), \quad \gamma_i \leftarrow C_i(r).$$

6. Each party P_i (for $i = 1, 2$) locally checks that

   $$(\alpha_1 + \alpha_2)(\beta_1 + \beta_2) = (\gamma_1 + \gamma_2).$$

   If not, the party aborts the protocol. □

23.2.4.6 Analysis of property (D1)

The following lemma captures the security property (D1).

Lemma 23.1. *Suppose D is corrupt and both P_1 and P_2 are honest. If the relation (23.7) does not hold, then the probability that either P_1 or P_2 will finish the protocol without aborting is at most*

$$\frac{2m}{q-m-1}.$$

Proof. On the one hand, if the polynomial identity (23.8) does not hold, then since the degrees on both sides of the equation are at most $2m$, the probability that either P_1 or P_2 finish the protocol without aborting is at most $(2m)/(q-m-1)$. This is because the polynomial $(A_1(X)+A_2(X))(B_1(X)+B_2(X))-(C_1(X)+C_2(X))$ has degree at most $2m$, and hence at most $2m$ roots, and party P_1 is choosing r from a set of size $q-m-1$. On the other hand, if the polynomial identity does hold, then P_1 and P_2 have values that satisfy the relation (23.7). □

23.2.4.7 Analysis of property (D2)

To analyze security property (D2), suppose P_1 is corrupt and D and P_2 are honest. The analysis of the case where P_2 is corrupt and D and P_1 are honest is almost identical. Of course, during the protocol, P_1 obtains the values (23.5), and the goal is to show that P_1 does not obtain any information about P_2's values (23.6). Consider the other information P_1 obtains during the protocol:

(I1) *The values a_{01}, b_{01}, and c_{01}, as well as the values c_{k1} for $k = m+1, \ldots, 2m$ obtained from D.* These values were the result of the dealer creating random shares, and as such are just random values in $\mathbb{Z}_q$.

(I2) *The values $\alpha_2, \beta_2, \gamma_2$ obtained from P_2.* We have $\alpha_2 = A_2(r)$. By Lagrange interpolation, we can write

$$A_2(X) = \sum_{k=0}^{m} \lambda_k(X) A_2(k),$$

where the $\lambda_k(X)$'s are the Lagrange basis polynomials for the evaluation points $\{0, \ldots, m\}$. Plugging in r for X, since $r \notin \{0, \ldots, m\}$, the values $\lambda_k(r)$ are non-zero; moreover, since $A_2(0)$ was chosen at random by the dealer, from P_1's point of view, α_2 is just a random value in $\mathbb{Z}_q$. Similarly, β_2 is just a random value in $\mathbb{Z}_q$. Finally, γ_2 is uniquely determined by the equation

$$(\alpha_1+\alpha_2)(\beta_1+\beta_2) = (\gamma_1+\gamma_2), \tag{23.9}$$

where $\alpha_1 = A_1(r)$, $\beta_1 = B_1(r)$, and $\gamma_1 = C_1(r)$.

(I3) *P_2's abort decision.* Party P_1 sends arbitrary values $\hat{\alpha}_1, \hat{\beta}_1, \hat{\gamma}_1 \in \mathbb{Z}_q$ to P_2, who aborts if

$$(\hat{\alpha}_1+\alpha_2)(\hat{\beta}_1+\beta_2) \neq (\hat{\gamma}_1+\gamma_2), \tag{23.10}$$

where $\alpha_2, \beta_2, \gamma_2$ were already determined as above. Thus, P_2's decision to abort or not abort (that is the question) does not tell P_1 anything that it does not already know.

So we see that the information P_1 obtains could have been generated by P_1 itself, just from the data (23.5). We can formalize this result as follows. This will be convenient later when we perform a more rigorous security analysis. We begin by defining a machine, $\mathcal{S}_D^{(1)}$, which is called a **simulator**. It is designed to interact with P_1, playing the roles of *both* D and P_2 in Protocol 23.5. It is initialized just with values (23.5), and it generates all of the data that P_1 receives (including P_2's decision to abort) as above in (I1)–(I3):

- The values a_{01}, b_{01}, and c_{01}, as well as the values c_{k1} for $k = m+1, \ldots, 2m$, are generated at random.

- The values α_2 and β_2 are generated at random, and γ_2 is determined by (23.9).

- P_2's abort decision is determined by (23.10).

We consider two experiments involving an adversary $\mathcal{A}$ and a challenger.

Experiment 0: $\mathcal{A}$ generates values (23.5) and (23.6) satisfying (23.7) and gives these to the challenger.

$\mathcal{A}$ then interacts with the challenger: $\mathcal{A}$ plays the role of P_1, while the challenger plays the roles of D and P_2, where D is initialzed with (23.5) and (23.6). At the end of the experiment, $\mathcal{A}$ outputs $\hat{b} \in \{0, 1\}$.

Experiment 1: This is the same as Experiment 0, except that the challenger ignores the values (23.6), and plays the roles of D and P_2 using $\mathcal{S}_D^{(1)}$, which is initialized with the values (23.5).

From the above discussion, the following should be clear:

Fact 23.2. *Every adversary $\mathcal{A}$ (even a computationally unbounded one) outputs 1 with the same probability in both of the above experiments.*

In the case where P_2 is corrupt, and D and P_1 are honest, there is a simulator $\mathcal{S}_D^{(2)}$ that plays a role analogous to that of $\mathcal{S}_D^{(1)}$. The only difference is that $\mathcal{S}_D^{(2)}$, playing the role now of P_1, chooses r itself.

23.2.4.8 Application to Protocol 23.3

We make a few remarks about how this general protocol is applied specifically to our authenticated dealer's protocol. The dealer distributes sharings $[K^{(1)}]$ and $[K^{(2)}]$. Consider one authenticated singleton sharing $[\![a]\!]$. This is really three sharings $([a], [a^{(1)}], [a^{(2)}])$, and the dealer has to prove two product relations, namely, $a^{(1)} = K^{(1)}a$ and $a^{(2)} = K^{(2)}a$. Now consider one authenticated Beaver triple sharing $([\![a]\!], [\![b]\!], [\![c]\!])$. For each of $[\![a]\!]$, $[\![b]\!]$, and $[\![c]\!]$, the dealer has to prove two product relations, as above, as well as the product relation $c = ab$. So each authenticated Beaver triple sharing leads to seven product relations that need to be proved.

When these get translated into relations as presented in (23.7), one sees that there are in fact additional relations among the a_{ki}'s, b_{ki}'s, and c_{ki}'s in (23.7). Specifically, many of these values will be identical, and do not need to be explicitly transmitted from the dealer to the players. For example, the shares of $K^{(1)}$ and $K^{(2)}$ appear many times.

An optimization. Except for values that are supposed to be identical, all of the a_{ki}'s, b_{ki}'s, and c_{ki}'s seen by either party are in fact just random values in $\mathbb{Z}_q$. As an optimization, the dealer could just transmit to one party, say P_1, the seed to a PRG, who can then generate these values by itself. This will reduce the communication cost of the dealer's protocol essentially in half.

23.2.4.9 Security of the entire circuit evaluation protocol

We have already analyzed the case where D is honest and either P_1 or P_2 is corrupt, but without Protocol 23.5. Security property (D2) ensures that the protocol remains secure in this case.

Now consider the case where D is corrupt and P_1 and P_2 are honest. Privacy is trivially attained, simply because P_1 and P_2 are honest, and the fact that D learns nothing about either of P_1's or P_2's inputs or outputs. Security property (D1) ensures that P_1 and P_2 compute correct outputs, so the circuit evaluation protocol provides soundness. Input independence is also trivially attained, simply because D contributes no inputs to the protocol.

The above is really only an intuitive argument of security. A formal proof will be presented later, in Section 23.5.9.

23.3 Garbled circuits: another approach to MPC

In this section, we present another approach to MPC, which is based on a technique called **garbled circuits**. This technique yields a simple 2-party protocol that is secure against honest-but-curious adversaries. This technique can also be used to construct 2-party protocols that are secure against malicious adversaries. However, we present a much simpler and more efficient *3-party* protocol that is secure against malicious adversaries, assuming at most one of the three parties is corrupt. Just like for Beaver's 2.5-party maliciously secure protocol, for the protocol we present, only two parties have inputs and outputs — the third party is only there to assist the other two parties, but does not itself contribute any inputs or obtain any outputs. However, there are some key differences between Beaver's protocol and our 3-party garbled circuit protocol. Specifically, our 3-party garbled circuit protocol

- only needs a constant number of rounds of communication between the parties (an important advantage over Beaver's protocol, where the number of rounds was essentially the multiplicative depth of the arithmetic circuit);

- works on boolean circuits, rather than arithmetic circuits (a limitation compared to Beaver's protocol because the smallest boolean circuit for a function tends to be larger than the smallest arithmetic circuit for the same function);

- uses the third party not just in a pre-processing phase, but in the execution of the actual protocol.

23.3.1 Boolean circuit evaluation

A boolean circuit is a circuit with several types of logical gates, where each gate has several inputs and one output. Typically, such a circuit may be comprised of AND, OR, and NOT gates, but other logic gates may also be used. We will restrict ourselves to gates with exactly two inputs — this is not a significant restriction, as any boolean circuit can be efficiently converted to a circuit

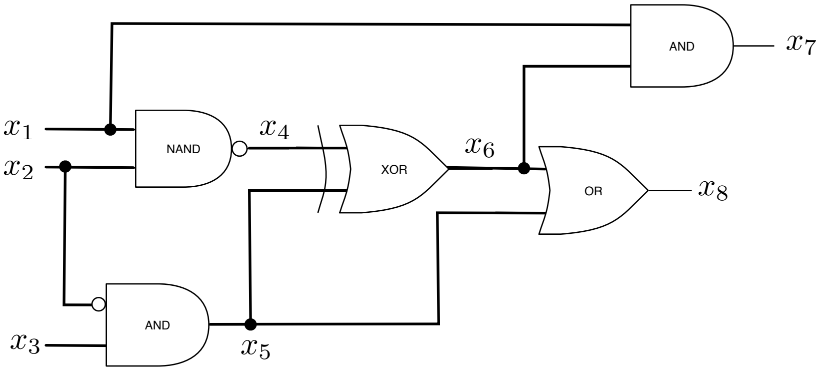


Figure 23.4: A boolean circuit

with only such gates. Each input of a gate is either an input of the circuit or the output of some other gate. There is no restriction on the "fan out" of a gate: the output of a gate may be fed as an input to any number of gates. The output of some gates are designated as circuit outputs.

Fig. 23.4 shows a simple boolean circuit. This particular circuit has three inputs, x_1, x_2, and x_3, and two outputs x_7 and x_8. For each input to the circuit, and each gate output, we have drawn a "wire" on which the value input to the circuit or computed by the gate is carried. Besides the three input wires and two output wires, there are three internal wires, which carry the internal values x_4, x_5, and x_6. In general, such a circuit should contain no cycles. This guarantees that once the values carried on the input wires are determined, we can calculate the values carried on all of the remaining wires by evaluating the gates one at a time, in some fixed order (known as a "topological ordering"), so that by the time we evaluate a particular gate, the values carried on its input wires have already been calculated. In this example, if we feed input values x_1, x_2, and x_3 to the circuit, then the values on the remaining wires may be computed as follows:

$$\begin{aligned}
&x_4 \leftarrow \overline{x_1 \wedge x_2}\\
&x_5 \leftarrow \overline{x_2} \wedge x_3\\
&x_6 \leftarrow x_4 \oplus x_5\\
&x_7 \leftarrow x_1 \wedge x_6\\
&x_8 \leftarrow x_6 \vee x_5
\end{aligned}$$

We want to design an algorithm that lets two parties, call them P_1 and P_2, securely evaluate such a circuit. We assume the circuit itself is fixed and publicly known. We assume the circuit takes a number of inputs $x_1, \ldots, x_n$ and produces some number of outputs $y_1, \ldots, y_m$, where each x_i and y_j is a bit. Some inputs will be supplied by P_1 and the others will be supplied by P_2. For simplicity, we assume that P_1 and P_2 obtain the same outputs (although it is not hard to generalize things so that some outputs will be given to P_1 and others will be given to P_2). The goal is to design a protocol that allows P_1 and P_2 to securely evaluate the circuit. As discussed in Section 23.1.1, this means that the protocol provides *privacy*, *soundness*, and *input independence.*

privacy: neither party learns anything about the other party's inputs (except for information that is inherently revealed by the outputs);

soundness: honest parties compute correct outputs (if they compute any output at all);

input independence: both parties must choose their inputs independently of the other's.

23.3.2 Yao's 2-party garbled circuit technique: basic ideas

At a very high level, the idea is this.

- P_1 will generate a "garbled encoding" of the circuit and send this to P_2.
- P_1 and P_2 then execute a special interactive subprotocol that lets P_2 obtain "garbled encodings" of both P_1's and P_2's inputs. These "garbled encodings" reveal to P_2 nothing about P_1's inputs, and the subprotocol itself reveals to P_1 nothing about P_2's inputs.
- Once P_2 has the "garbled encodings" of the circuit and all of the inputs, it locally runs a special *evaluation algorithm*, which allows P_2 to compute "garbled encodings" of the outputs.
- P_2 then sends these "garbled encodings" of the outputs back to P_1. These "garbled encodings" of the outputs allows P_1 to compute the actual outputs, but nothing more.
- Finally, P_1 sends the actual outputs back to P_2.

As a first step towards fleshing out the above idea, we give a formal definition of the syntax for a garbling scheme.

Definition 23.1 (Garbling scheme). *A* ***garbling scheme*** *consists of four efficient algorithms:*

1. *A probabilistic* ***circuit garbling algorithm*** *Garble that is invoked as*

$$(\mathcal{F}, \mathscr{e}, \mathscr{d}) \xleftarrow{\text{R}} Garble(f),$$

where the input f is a boolean circuit. The result $\mathcal{F}$ is called a ***garbled encoding of*** *f, the result $\mathscr{e}$ is called the* ***input encoding data****, and the result $\mathscr{d}$ is called the* ***output decoding data****.*

2. *A deterministic* ***input encoding algorithm*** *Encode that is invoked as*

$$\mathcal{X} \leftarrow Encode(\mathscr{e}, \boldsymbol{x}),$$

where $\mathscr{e}$ is the input encoding data, and $\boldsymbol{x}$ is a vector of bits. The result $\mathcal{X}$ is called a ***garbled encoding of $\boldsymbol{x}$****.*

3. *A deterministic* ***garbled circuit evaluation algorithm*** *Eval that is invoked as*

$$\mathcal{Y} \leftarrow Eval(\mathcal{F}, \mathcal{X}),$$

where $\mathcal{F}$ is a garbled encoding of a circuit and $\mathcal{X}$ is a garbled encoding of an input vector. The result $\mathcal{Y}$ is called a ***garbled output****.*

4. *A deterministic* ***output decoding algorithm*** *Decode that is invoked as*

$$\boldsymbol{y} \leftarrow Decode(\mathscr{d}, \mathcal{Y}),$$

where $\mathscr{d}$ is the output decoding data and $\mathcal{Y}$ is a garbled output. The result $\boldsymbol{y}$ is either the special symbol reject *or a vector of bits.*

The basic **correctness requirement** for any general garbling scheme is as follows:

> *for every boolean circuit f, every possible output $(\mathcal{F}, \mathscr{e}, \mathscr{d})$ of $\mathit{Garble}(f)$, and every boolean input vector $\boldsymbol{x}$ for f, we have*
>
> $$\mathit{Decode}\Big(\, \mathscr{d},\ \mathit{Eval}\big(\, \mathcal{F},\ \mathit{Encode}(\mathscr{e}, \boldsymbol{x})\,\big)\,\Big) = f(\boldsymbol{x}).$$

We next state an intuitive summary of the security properties we require from a garbling scheme.

Obliviousness: $(\mathcal{F}, \mathcal{X})$ reveals nothing about $\boldsymbol{x}$;

Authenticity: given $(\mathcal{F}, \mathcal{X})$, it is hard to find $\widehat{\mathcal{Y}} \neq \mathit{Eval}(\mathcal{F}, \mathcal{X})$ that decodes to something besides reject;

Output simulatability: $\mathcal{Y}$ can be efficiently computed from $f(\boldsymbol{x})$ and $\mathscr{d}$.

Note that in our presentation here, our notion of obliviousness does not require that the garbled encoding of a circuit hides the circuit itself. There are, in fact, stronger notions of obliviousness that one can consider, in which the garbled encoding of the circuit should hide partial information of the circuit. For simplicity, we do not discuss here these notions or a number of other security notions that are useful in some applications.

We can recast our high-level idea presented above in the language of garbling schemes as follows:

- P_1 executes $(\mathcal{F}, \mathscr{e}, \mathscr{d}) \xleftarrow{\mathrm{R}} \mathit{Garble}(f)$ and sends $\mathcal{F}$ to P_2.
- P_1 and P_2 then execute a special interactive subprotocol that lets P_2 obtain $\mathcal{X} := \mathit{Encode}(\mathscr{e}, \boldsymbol{x})$, where $\boldsymbol{x}$ is the vector comprising both P_1's and P_2's inputs. The subprotocol itself reveals nothing to P_1 and nothing to P_2 besides $\mathcal{X}$. The obliviousness property of the garbling scheme ensures that $\mathcal{F}$ and $\mathcal{X}$ reveal to P_2 nothing about P_1's inputs.
- P_2 executes $\mathcal{Y} \leftarrow \mathit{Eval}(\mathcal{F}, \mathcal{X})$ and sends $\mathcal{Y}$ to P_1.
- P_1 executes $\boldsymbol{y} \leftarrow \mathit{Decode}(\mathscr{d}, \mathcal{Y})$, and sends $\boldsymbol{y}$ to P_2. The output simulatability property ensures that P_1 learns nothing other than $\boldsymbol{y}$.

In the honest-but-curious setting, while we need the obliviousness and output simulatability properties of the garbling scheme to ensure that each party's inputs remain hidden from the other party, we do not need the authenticity property. We will only need the authenticity property when we discuss maliciously secure variants of this protocol.

23.3.3 Garbling schemes: an application to outsourcing computation

Garbling schemes have a number of applications. To get more of an idea of how one can use a garbling scheme, we start with an application to **outsourcing computation**.

Alice is a spy on a mission, traveling with her mobile device, which has only limited computational power and communication bandwidth. She needs to perform a computation on sensitive data she has just collected. The computation is too expensive to perform on her mobile device, so she outsources it to a cloud-based service. Alice does not trust the cloud-based service at all: neither to carry out the computation correctly nor to protect her sensitive data.

Assume that the computation Alice wants to carry out can be represented by a boolean circuit f. Suppose that before heading out on her mission, Alice computes a garbled encoding $\mathcal{F}$ of f, along with corresponding encoding data $\mathscr{e}$ and decoding data $\mathscr{d}$, as follows:

$$(\mathcal{F}, \mathscr{e}, \mathscr{d}) \xleftarrow{\mathrm{R}} \mathit{Garble}(f).$$

She loads $\mathscr{e}$ and $\mathscr{d}$ on her mobile device and sends $\mathcal{F}$ to one or more cloud-based services. This computation is done on a secure computing device, and the memory of that device is wiped clean before she heads out.

So now, while Alice is out on her mission, $\mathscr{e}$ and $\mathscr{d}$ are stored on her mobile device, and nowhere else. When Alice wants to evaluate f on a specific input vector $\boldsymbol{x}$ she uses the input encoding data $\mathscr{e}$ to compute the garbled encoding $\mathcal{X}$ of $\boldsymbol{x}$, as follows:

$$\mathcal{X} \leftarrow \mathit{Encode}(\mathscr{e}, \boldsymbol{x}).$$

She then sends $\mathcal{X}$ to the cloud-based service.

The cloud-based service now has the garbled encoding $\mathcal{F}$ of f, as well as the garbled encoding $\mathcal{X}$ of $\boldsymbol{x}$. It can apply the garbled circuit evaluation algorithm to compute a garbled encoding $\mathcal{Y}$ of the output, as follows:

$$\mathcal{Y} \leftarrow \mathit{Eval}(\mathcal{F}, \mathcal{X}).$$

The service then sends $\mathcal{Y}$ back to Alice.

Finally, Alice decodes the garbled encoding $\mathcal{Y}$ using the output decoding data $\mathscr{d}$, as follows:

$$\boldsymbol{y} \leftarrow \mathit{Decode}(\mathscr{d}, \mathcal{Y}).$$

The *obliviousness* property of the garbling scheme ensures that the cloud-based service learns nothing about Alice's input $\boldsymbol{x}$, while *authenticity* property ensures that Alice either computes the correct output $\boldsymbol{y} = f(\boldsymbol{x})$, or outputs reject. The *output simulatability* property is not required for this application. Of course, the basic correctness requirement ensures that Alice *does* get a correct result as long as nothing goes wrong with the communication network, and as long as the cloud-based service does its job correctly. If Alice does output reject, then Alice can retry the computation, possibly with a different cloud-based service (if she sent $\mathcal{F}$ to several services).

This technique is effective when the size of the circuit is much bigger than the number of inputs and outputs of the circuit. In this case, the computation and communication costs for Alice to encode and transmit her input and to receive and decode the output will be small compared to the cost of evaluating the circuit herself.

A severe limitation of this technique is that if Alice needs to compute f on several input vectors while she is out on her mission, she will have to repeat the above procedure once for every input. That is, she cannot use the same garbled encoding $\mathcal{F}$ on two different input vectors without compromising security.

23.3.4 `Garble0`: a simple but efficient garbling scheme

We now present a simple but fairly efficient garbling scheme, which we call `Garble0`. Actually, `Garble0` is a special case of a somewhat more generic scheme, which works as follows.

Consider a given boolean circuit f that we wish to garble. The circuit garbling algorithm will generate a pair $(X^{(0)}, X^{(1)})$ of random "tokens" for each wire in the circuit: the token $X^{(0)}$ will represent the value 0 for that wire and the token $X^{(1)}$ will represent the value 1 for that wire. We stress that each wire gets its own pair of random tokens, which are independent of the tokens for all other wires. We also stress that there is no correlation between a token and the value it represents, so learning the token representing a value yields no information about the value itself, nor does it yield any information about the value of any other token.

Suppose the circuit f has n input wires. Then the input encoding data is defined to be

$$\mathscr{e} := \big((X_1^{(0)}, X_1^{(1)}), \ldots, (X_n^{(0)}, X_n^{(1)})\big), \tag{23.11}$$

where, for the ith input wire, $X_i^{(0)}$ is the token representing the value 0 and $X_i^{(1)}$ is the token representing the value 1 for that wire.

The garbled encoding of the input vector $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ to the circuit consists of the tuple

$$\mathscr{X} = (X_1^{(x_1)}, \ldots, X_n^{(x_n)}).$$

That is, each input x_i is encoded independently as the token that represents the value x_i carried on the ith input wire. We stress that the garbled encoding of the inputs includes *only* these tokens — it does not include the tokens representing the values *not* carried on the input wires.

Suppose the circuit f has m output wires. Then the output decoding data is defined to be

$$\mathscr{d} = \big((Y_1^{(0)}, Y_1^{(1)}), \ldots, (Y_m^{(0)}, Y_m^{(1)})\big), \tag{23.12}$$

where, for the jth output wire, $Y_j^{(0)}$ is the token representing the value 0 and $Y_j^{(1)}$ is the token representing the value 1 for that wire.

A garbled output $\mathscr{Y}$ is of the form $(Y_1, \ldots, Y_m)$. The output decoding algorithm determines the output vector $\boldsymbol{y} = (y_1, \ldots, y_m) \in \{0,1\}^m$ as follows. For $j = 1, \ldots, m$, tests if either $Y_j = Y_j^{(0)}$ or $Y_j = Y_j^{(1)}$: if the first equality holds, then $y_j := 0$; if the second holds, then $y_j := 1$; if neither holds, then a decoding error occurs, and the output of the decoding algorithm is reject.

The garbled encoding $\mathscr{F}$ of the circuit f consists of a gate-by-gate encoding of the circuit. For each gate in the circuit, the circuit garbling algorithm will produce a corresponding **garbled encoding** $\mathscr{G}$ of the gate. Intuitively, $\mathscr{G}$ will be designed to work in conjunction with an efficient **garbled *gate* evaluation algorithm**, *GateEval*, as follows. Suppose $g : \{0,1\} \times \{0,1\} \to \{0,1\}$ is the function computed by the gate. Further suppose that

- $(X^{(0)}, X^{(1)})$ is the pair of tokens associated with the first input wire of the gate,
- $(Y^{(0)}, Y^{(1)})$ is the pair of tokens associated with the second input wire of the gate, and
- $(Z^{(0)}, Z^{(1)})$ is the pair of tokens associated with the output wire of the gate.

Then for all values $u, v \in \{0,1\}$, we must have

$$\mathit{GateEval}(\mathscr{G}, X^{(u)}, Y^{(v)}) = Z^{(g(u,v))};$$

moreover, given $\mathscr{G}, X^{(u)}, Y^{(v)}$, it should be hard to gain any information about the token $Z^{(g(u',v'))}$ for any $(u', v') \neq (u, v)$.

For example, for an AND gate with two input wires we have

$$
\begin{aligned}
&\mathit{GateEval}(\mathcal{G}, X^{(0)}, Y^{(0)}) = Z^{(0)} \quad \text{(0 AND 0 = 0)}\,, \quad \mathit{GateEval}(\mathcal{G}, X^{(0)}, Y^{(1)}) = Z^{(0)} \quad \text{(0 AND 1 = 0)}\,,\\
&\mathit{GateEval}(\mathcal{G}, X^{(1)}, Y^{(0)}) = Z^{(0)} \quad \text{(1 AND 0 = 0)}\,, \quad \mathit{GateEval}(\mathcal{G}, X^{(1)}, Y^{(1)}) = Z^{(1)} \quad \text{(1 AND 1 = 1)}\,.
\end{aligned}
$$

Here is how the garbled circuit evaluation algorithm works. Recall that the garbled encoding of the input gives us, for each input wire, the token representing the value carried on that wire. Now we iterate the following procedure, gate by gate:

- pick a gate for which we have already computed the tokens X and Y representing the values carried on its two input wires;
- if $\mathcal{G}$ is the garbled encoding of the gate, compute $Z \leftarrow \mathit{GateEval}(\mathcal{G}, X, Y)$, which represents the value carried on the output wire of the gate.

When this finishes, for each output wire, we will have the token representing the value on that wire, and so can form the garbled output by simply concatenating those tokens.

23.3.4.1 An example

Consider again the application in Section 23.3.3. Suppose the circuit f Alice wants to evaluate is the one in Fig. 23.4. There are 8 wires in this circuit, which we shall call wires 1–8. Wires 1–3 will carry the input values x_1, x_2, x_3, wires 4–6 will carry the intermediate values x_4, x_5, x_6, and wires 7–8 will carry the output values x_7, x_8. When Alice generates the garbled encoding of f, she generates tokens $X_i^{(0)}, X_i^{(1)}$ for $i = 1, \ldots, 8$, along with garbled encodings of the gates, which we call $\mathcal{G}_4, \ldots, \mathcal{G}_8$, so that $\mathcal{G}_i$ is the garbled encoding of the gate whose output is carried on wire i. The encoding data is

$$\mathscr{e} = ((X_1^{(0)}, X_1^{(1)}), (X_2^{(0)}, X_2^{(1)}), (X_3^{(0)}, X_3^{(1)})),$$

the decoding data is

$$\mathscr{d} = ((X_7^{(0)}, X_7^{(1)}), (X_8^{(0)}, X_8^{(1)})),$$

and the garbled encoding of the circuit is

$$\mathcal{F} = (\mathcal{G}_5, \mathcal{G}_6, \mathcal{G}_7, \mathcal{G}_8).$$

All of this data is shown in Fig. 23.5.

Initially, Alice sends $\mathcal{F}$ to the cloud-based service and stores $\mathscr{e}$ and $\mathscr{d}$ on her mobile device. While she is out on her mission, suppose that the wants to evaluate f on the input vector $\boldsymbol{x} = (x_1, x_2, x_3) = (1, 0, 1)$. Using the encoding data $\mathscr{e}$, Alice computes the garbled encoding $\mathcal{X}$ of $\boldsymbol{x}$ as

$$\mathcal{X} = (X_1^{(1)}, X_2^{(0)}, X_3^{(1)}),$$

and sends $\mathcal{X}$ to the cloud-based service. The cloud-based service now knows the tokens for wires 1–3, but is oblivious to the values they represent. Using $\mathcal{F}$ and $\mathcal{X}$, the cloud-based service now proceeds gate by gate, using the garbled gate evaluation algorithm, to compute the tokens representing the values on wires 4–8 of the circuit, all the while oblivious to the values these tokens represent. Using the tokens $X_1^{(0)}$ and $X_2^{(1)}$, and the garbled gate encoding $\mathcal{G}_4$, it obtains the token $X_4^{(1)}$; using the tokens $X_2^{(1)}$ and $X_3^{(0)}$, and the garbled gate encoding $\mathcal{G}_5^{(0)}$, it obtains the token $X_5^{(0)}$; using the

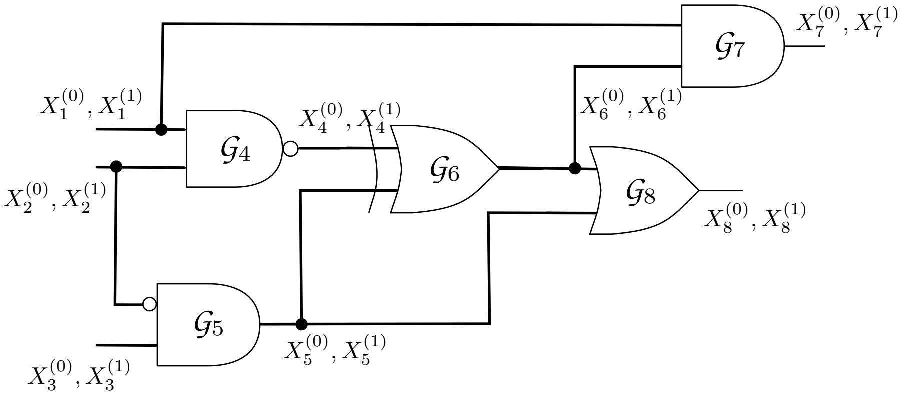


Figure 23.5: A garbled circuit

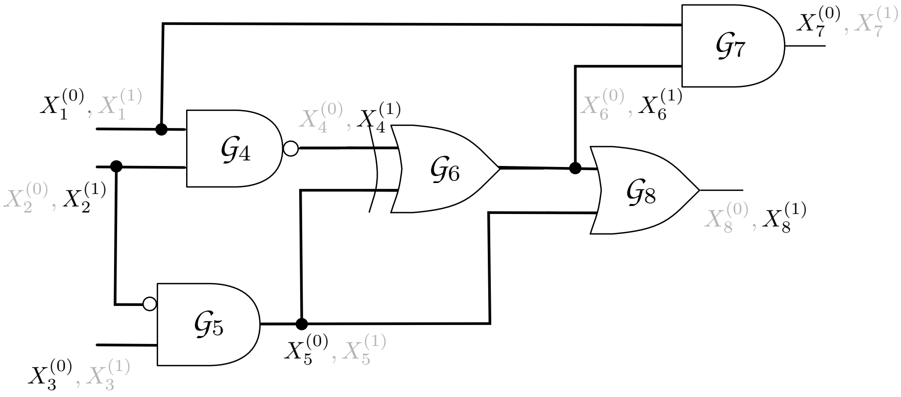


Figure 23.6: Oblivious evaluation of a garbled circuit

tokens $X_4^{(1)}$ and $X_5^{(0)}$, and the garbled gate encoding $\mathcal{G}_6$, it obtains the token $X_6^{(1)}$; and so on. This is illustrated in Fig. 23.6.

Finally, the cloud-based service sends the garbled output $(X_7^{(0)}, X_8^{(1)})$ back to Alice, who uses the decoding data $\mathcal{d}$ to determine the output vector $\boldsymbol{y} = (0, 1)$.

During the process, for each wire in the circuit, the cloud-based service is either given or computes the token that represents the value carried on that wire, but it cannot compute the other token for that wire. This property ensures that the cloud-based service cannot make Alice accept an incorrect result.

23.3.4.2 A simple implementation of garbled gate encodings

We will now discuss one possible implementation of garbled gate encoding that satisfies the requirements informally introduced in Section 23.3.2. We stress, however, that there are a number of other techniques that have been developed.

Our garbled gate encoding scheme is built from the following ingredients:

- Let $\mathcal{T} := \{0,1\}^\ell$ be our set of tokens. We call tokens whose first bit is 0 **type-0 tokens** and tokens whose first bit is 1 **type-1 tokens**. There is nothing really significant about the way we partition tokens into two types. The salient property is that it is easy to determine the type of any token.
- Let $\mathcal{I}$ be a finite set of identifiers. Each gate in the circuit to be garbled will have a unique identifier $gID \in \mathcal{I}$.
- Let $H : \mathcal{T} \times \mathcal{T} \times \mathcal{I} \to \mathcal{T}$ be a hash function. To obtain a secure garbling scheme, we will model H as a random oracle, and require that $|\mathcal{T}|$ is super-poly.

For each wire in the circuit, the garbling algorithm will choose a random type-0 token $A^{(0)}$ and a random type-1 token $A^{(1)}$, along with a random bit $r \in \{0,1\}$. We call $(A^{(0)}, A^{(1)}, r)$ the *private encoding data* for this wire — it is only known to the garbling algorithm itself. For $u \in \{0,1\}$, define $X^{(u)} := A^{(u\oplus r)}$, which is the token that represents the value u for this wire.

Consider a gate in the circuit with associated identifier $gID \in \mathcal{I}$. Assume that the gate computes the function $g : \{0,1\} \times \{0,1\} \to \{0,1\}$. Further assume that the private encoding data for the gate's first input wire is $(A^{(0)}, A^{(1)}, r)$, for the gate's second input wire is $(B^{(0)}, B^{(1)}, s)$, and for the gate's output wire is $(C^{(0)}, C^{(1)}, t)$. For $a, b \in \{0,1\}$ set

$$E^{(a,b)} := H\big(A^{(a)}, B^{(b)}, gID\big) \oplus C^{(g(a\oplus r, b\oplus s)\oplus t)} \in \mathcal{T}. \tag{23.13}$$

We define the garbled encoding of this gate as the 4-tuple

$$\mathcal{G} := \big(gID, E^{(0,0)}, E^{(0,1)}, E^{(1,0)}, E^{(1,1)}\big) \in \mathcal{I} \times \mathcal{T}^4. \tag{23.14}$$

The garbled gate evaluation algorithm is defined as follows:

$$GateEval(\mathcal{G}, X, Y) := H(X, Y, gID) \oplus E^{(a,b)}, \tag{23.15}$$

where $\mathcal{G}$ is as in (23.14), a is the type of X, and b is the type of Y.

To see why this works, let $u, v \in \{0,1\}$, and suppose we are given two tokens:

- $X^{(u)} = A^{(u\oplus r)}$, which is the token representing u on the first input wire, and which has type $u \oplus r$, and
- $Y^{(v)} = B^{(v\oplus s)}$, which is the token representing v on the second input wire, and which has type $v \oplus s$.

We want to show that

$$GateEval(\mathcal{G}, X^{(u)}, Y^{(v)}) = Z^{(g(u,v))},$$

where

- $Z^{(g(u,v))} = C^{(g(u,v)\oplus t)}$, which is the token representing $g(u,v)$ on the output wire.

We have

$$\begin{aligned}
&GateEval(\mathcal{G}, X^{(u)}, Y^{(v)})\\
&\qquad = H(A^{(x\oplus r)}, B^{(y\oplus s)}, gID) \oplus E^{(x\oplus r, y\oplus s)}\\
&\qquad = C^{(g(u,v)\oplus t)}\\
&\qquad = Z^{(g(u,v))}.
\end{aligned}$$

Intuitively, each $E^{(a,b)}$ is an encryption of the token $C^{(g(a \oplus r, b \oplus s) \oplus t)}$, and one can decrypt $E^{(a,b)}$ using *GateEval* as long as one knows both $A^{(a)}$ and $B^{(b)}$. Moreover, in running the circuit evaluation algorithm, one will only know one of $A^{(0)}$ and $A^{(1)}$ and one of $B^{(0)}$ and $B^{(1)}$, and so can decrypt only one of the ciphertexts $E^{(0,0)}, E^{(0,1)}, E^{(1,0)}, E^{(1,1)}$.

23.3.5 Garbling schemes: formally defining security properties

We can formulate the *obliviousness* property more precisely using an attack game, which is analogous to the game used to define semantic security for encryption:

Attack Game 23.1 (Oblivious garbling). For a given garbling scheme

$$(\mathit{Garble}, \mathit{Encode}, \mathit{Eval}, \mathit{Decode}),$$

and for a given adversary, we define two experiments, Experiment 0 and Experiment 1. For $b = 0, 1$, we define

Experiment b:

- The adversary submits a circuit f and two circuit input vectors $\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}$ to the challenger.
- The challenger computes

$$(\mathcal{F}, e, d) \xleftarrow{\text{R}} \mathit{Garble}(f), \quad \mathcal{X} \leftarrow \mathit{Encode}(e, \boldsymbol{x}^{(b)}),$$

  and sends $(\mathcal{F}, \mathcal{X})$ to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

For $b = 0, 1$, let W_b be the event that the adversary outputs 1 in Experiment b. We define the adversary's advantage in the game as $|\Pr[W_0] - \Pr[W_1]|$. □

Definition 23.2 (Oblivious garbling). *A garbling scheme is called* ***oblivious*** *if every efficient adversary has a negligible advantage in Attack Game 23.1.*

We can formulate the *authenticity* property more precisely with an attack game, which is analogous to the game used to define ciphertext integrity for encryption:

Attack Game 23.2 (Authentic garbling). For a given garbling scheme

$$(\mathit{Garble}, \mathit{Encode}, \mathit{Eval}, \mathit{Decode}),$$

the attack game runs as follows:

- The adversary submits a circuit f and an input vector $\boldsymbol{x}$ to the challenger.
- The challenger computes

$$(\mathcal{F}, e, d) \xleftarrow{\text{R}} \mathit{Garble}(f), \quad \mathcal{X} \leftarrow \mathit{Encode}(e, \boldsymbol{x}),$$

  and sends $(\mathcal{F}, \mathcal{X})$ to the adversary.

- The adversary outputs a garbled output $\widehat{\mathcal{Y}}$.

We say the adversary wins the game if $\widehat{\mathcal{Y}} \neq \mathit{Eval}(\mathcal{F}, \mathcal{X})$ and $\mathit{Decode}(d, \widehat{\mathcal{Y}}) \neq \mathsf{reject}$. We define the adversary's advantage in the game as the probability that it wins the game. □

Definition 23.3 (Authentic garbling). *A garbling scheme is called* ***authentic*** *if every efficient adversary has a negligible advantage in Attack Game 23.2.*

Finally, we can define the *output simulatability* property as follows:

Definition 23.4 (Output simulatable garbling). *A garblng scheme is called* ***output simulatable*** *if there is an efficient deterministic algorithm Reverse with the following property: for every circuit f, every possible output $(\mathcal{F}, e, d)$ of Garble(f), and every input vector $\boldsymbol{x}$ for f, we have*

$$\mathit{Eval}\big(\,\mathcal{F},\ \mathit{Encode}(e, \boldsymbol{x})\,\big) = \mathit{Reverse}\big(\,d,\ f(\boldsymbol{x})\,\big).$$

On the security of `Garble0`. Recall our basic garbling scheme, `Garble0`. One can prove that if the token space is of super-poly size and the hash function used in the garbled gate encoding mechanism is modeled as a random oracle, then `Garble0` is both oblivious and authentic. We leave this proof to the reader. It is also easy to see that `Garble0` is output simulatable unconditionally. Indeed, the *Reverse* algorithm works in the same way as the *Encode* algorithm for `Garble0`.

Adaptive security. The careful reader may have noticed a potential weakness in our formal definitions of obliviousness and authenticity, if we try to apply them directly to our computation outsourcing example in Section 23.3.3. Suppose that the sensitive data $\boldsymbol{x}$ that our spy Alice has collected has been tainted. Specifically, suppose that her adversaries, after bribing the cloud-based service, learn the garbled encoding $\mathcal{F}$ of the circuit, and are able to plant some data that at least partially influences the value of $\boldsymbol{x}$ in some clever way that depends on $\mathcal{F}$. In general, in any such application of this computation outsourcing technique, it may not be possible to ensure that the circuit input vector $\boldsymbol{x}$ does not somehow depend on the garbled encoding $\mathcal{F}$.

Thus, to use garbled circuits for general computation outsourcing, we need stronger notions of obliviousness and authenticity. Specifically, we need to give the adversary more power in the corresponding attack games. In Attack Game 23.1, the adversary should be allowed to choose the input vectors $\boldsymbol{x}^{(0)}$ and $\boldsymbol{x}^{(1)}$ *after* seeing the garbled encoding $\mathcal{F}$ of the circuit f. Similarly, in Attack Game 23.2, the adversary should be allowed to choose the input vector $\boldsymbol{x}$ *after* seeing the garbled encoding $\mathcal{F}$ of the circuit f. These modified attack games yield corresponding notions of ***adaptive*** **obliviousness** and ***adaptive*** **authenticity**.

Again, we leave it to the reader to prove that if the token space is of super-poly size and the hash function used in the garbled gate encoding mechanism is modeled as a random oracle, then `Garble0` is both adaptively oblivious and adaptively authentic. Note that for our eventual applications to multi-party computation, we will not need these adaptive security notions. For the non-adaptive security notions, it is possible to prove the security of `Garble0` without modeling the hash function as a random oracle, but rather, under a specific (and reasonable) indistinguishability assumption. However, for the adaptive security notions, the only known proof of security of `Garble0` is with a random oracle.

23.3.6 A 2-party garbling-based protocol secure against honest-but-curious adversaries

We shall now present our first protocol based on garbled circuits. It is a 2-party protocol, and is only secure against honest-but-curious adversaries.

However, before presenting the protocol, we will need some more ingredients. One ingredient consists of a syntactic property enjoyed by many practical garbling schemes, including our basic garbling scheme, `Garble0`, called "projective input encoding". The second ingredient is a subprotocol for 1-out-of-2 oblivious transfer (OT). We presented several OT protocols in Section 11.6.

23.3.6.1 Projective input encoding

We shall require that our garbling scheme has the **projective input encoding** property, which means that it encodes inputs in essentially the same way as `Garble0`. That is, the encoding data $\mathscr{e}$ is of the form (23.11), and the encoding algorithm works as in (23.11). We do not, however, make any particular requirements on the form of the elements $X_i^{(b)}$ appearing in (23.11), other than that they come from some finite set $\mathcal{T}$ of bit strings of some specified length. We call $\mathcal{T}$ the **set of input encoding tokens**.

23.3.6.2 1-out-of-2 oblivious transfer (OT)

Roughly speaking, in a 1-out-of-2 OT protocol, one party, called the *sender*, has two inputs, $X^{(0)}, X^{(1)}$, which may be bit strings of some given length, and another party, called the *receiver*, has a one-bit input $u \in \{0,1\}$, called a *choice bit*; at the end of the protocol, the receiver obtains the value $X^{(u)}$. The essential security properties of such a protocol are:

Sender privacy: the receiver learns nothing about the sender's other input $X^{(1-u)}$;

Receiver privacy: the sender learns nothing about the receiver's choice bit u.

As we will only need security against honest-but-curious adversaries, the protocols presented in Section 11.6 are adequate for our purposes here.

The 2-party protocol we describe will run multiple instances of the 1-out-of-2 OT protocol. We can do so by running many parallel instances of the protocols presented in Section 11.6 without sacrificing security. However, an efficient technique called **OT extension**, discussed in Section 23.7, lets one run n instances of the 1-out-of-2 OT protocol far more efficiently than running n parallel instances of the protocols in Section 11.6.

23.3.6.3 The protocol

Without further ado, here is the protocol for a given circuit f, which really just fills in the details of the high-level idea presented in Section 23.3.2.

1. Party P_1 computes $(\mathcal{F}, \mathscr{e}, \mathscr{d}) \overset{\text{R}}{\leftarrow} \mathit{Garble}(f)$ and sends $\mathcal{F}$ to P_2.

   Note: because we are assuming projective input encoding, the encoding data $\mathscr{e}$ *is of the form*
   $$\mathscr{e} = ((X_1^{(0)}, X_1^{(1)}), \ldots, (X_n^{(0)}, X_n^{(1)})).$$

2. For each input x_i provided by P_1, party P_1 sends $X_i^{(x_i)}$ to P_2.

3. For each input x_i provided by P_2, party P_1 obliviously transfers $X_i^{(x_i)}$ to P_2 using a 1-out-of-2 OT protocol, where $(X_i^{(0)}, X_i^{(1)})$ are P_1's inputs (in the role of sender) to the OT protocol, and x_i is P_2's input (in the role of receiver) to the OT protocol.

4. Now party P_2 has $\mathcal{F}$ as well as
$$\mathcal{X} = (X_1^{(x_1)}, \ldots, X_n^{(x_n)}).$$
Party P_2 computes $\mathcal{Y} \leftarrow \mathit{Eval}(\mathcal{F}, \mathcal{X})$ and sends $\mathcal{Y}$ to P_1.

5. After receiving $\mathcal{Y}$ from P_2, party P_1 computes $\boldsymbol{y} \leftarrow \mathit{Decode}(\mathcal{d}, \mathcal{Y})$.
Party P_1 outputs $\boldsymbol{y}$ and also sends $\boldsymbol{y}$ to P_2.

6. Party P_2 receives $\boldsymbol{y}$ and then outputs $\boldsymbol{y}$.

Note that Steps 1–3 require only a constant number of rounds of communication, assuming all of the 1-out-of-2 OTs are constant round and run in parallel. Thus, the entire protocol requires only a constant number of rounds of communication. Observe that we used the special *projective input encoding* property of the garbling scheme directly in the design of the protocol.

23.3.6.4 Security of the protocol

We shall now give some intuition as to why the protocol is secure against honest-but-curious adversaries. Here, we are assuming that both P_1 and P_2 follow the protocol, and the goal is to argue that neither party learns any information about the other party's input, other than information revealed by the output $\boldsymbol{y} = f(\boldsymbol{x})$.

The assumptions we rely on are the following:

- sender and receiver privacy properties for the OT protocol, and
- the obliviousness and output simulatability properties of the garbling scheme.

Privacy for P_1. We argue that P_2 learns no information about P_1's input, other than information revealed by the output $\boldsymbol{y} = f(\boldsymbol{x})$. This argument relies on the following assumptions:

- sender privacy for the OT protocol, and
- the obliviousness property of the garbling scheme.

Because of sender privacy for the OT protocol, the only information P_2 obtains during the protocol, besides $\boldsymbol{y} = f(\boldsymbol{x})$, is the garbled circuit encoding $\mathcal{F}$ and the garbled input encoding $\mathcal{X}$. By the obliviousness property of the garbling scheme, P_2 learns nothing more about the input vector $\boldsymbol{x}$.

Privacy for P_2. We argue that P_1 learns no information about P_2's input, other than information revealed by the output $\boldsymbol{y} = f(\boldsymbol{x})$. This argument relies on the following assumptions:

- receiver privacy for the OT protocol, and
- the output simulatability property of the garbling scheme.

Because of receiver privacy for the OT protocol, the only information P_1 obtains during the protocol is the garbled output $\mathcal{Y}$. Because of the output simulatability property of the garbling scheme, P_1 could have computed $\mathcal{Y}$ directly from $\boldsymbol{y} = f(\boldsymbol{x})$, and hence $\mathcal{Y}$ gives P_1 no more information about P_2's input than that implied by $\boldsymbol{y}$.

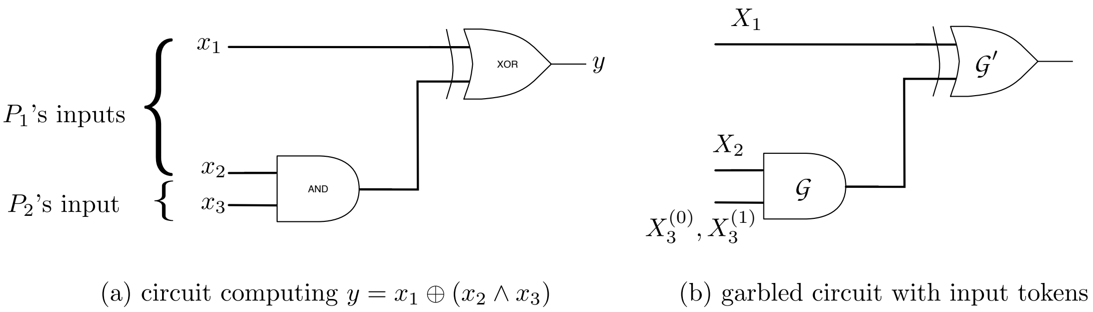


Figure 23.7: What goes wrong without OT

23.3.6.5 Why OT is needed

It is instructive to see why OT is needed in this protocol. More precisely, we address the question: why not just have P_1 send to P_2 both tokens $X_i^{(0)}, X_i^{(1)}$ for those inputs x_i provided by P_2, and let P_2 use the appropriate token $X_i^{(x_i)}$? To see why this is a bad idea, let us assume the garbling scheme is our basic garbling scheme, `Garble0`. Further, let us assume that the circuit computes the function $y = x_1 \oplus (x_2 \wedge x_3)$, where inputs x_1 and x_2 are contributed by P_1 and the input x_3 is contributed by P_2. The reader may verify the following: for every $x_2, x_3, y \in \{0, 1\}$, there exists $x_1 \in \{0, 1\}$ such that $y = x_1 \oplus (x_2 \wedge x_3)$. For example, if $x_3 = 0$, just set $x_1 \leftarrow y$, and if $x_3 = 1$, just set $x_1 \leftarrow y \oplus x_2$. This means that if P_2 knows his own input x_3, and learns the circuit output y, he really has no information about x_2: this information he has is consistent with both $x_2 = 0$ and $x_2 = 1$. The privacy property for the protocol should ensure that after the protocol runs, P_2 still has no information about x_2. But we will see that if P_2 obtains both tokens corresponding to the input x_3, then P_2 can determine x_2.

Fig. 23.7(a) shows the circuit computing $y = x_1 \oplus (x_2 \wedge x_3)$, which is composed of one AND gate and one XOR gate. Fig. 23.7(b) shows the situation where P_2 has received a token X_1 representing P_1's input x_1, a token X_2 representing P_1's input x_2, and two tokens $X_3^{(0)}, X_3^{(1)}$, so that P_2 can use $X_3^{(x_3)}$ to represent his input x_3. Party P_2 has also received the garbled encodings $\mathcal{G}$ of the AND gate and $\mathcal{G}'$ of the XOR gate. Even if P_2 honestly runs the protocol, a curious P_2 can determine P_1's input x_2 as follows. Party P_2 simply runs the garbled gate evaluation algorithm twice: once on inputs $\mathcal{G}$, X_2, and $X_3^{(0)}$, and again on inputs $\mathcal{G}$, X_2, and $X_3^{(1)}$. By the logic of an AND gate, if $x_2 = 0$, both executions of the algorithm will output the same token, and if $x_2 = 1$, the two executions will produce different tokens. Thus, P_2 can determine x_2. We conclude that this protocol does not provide privacy.

23.3.7 A 3-party garbling-based protocol secure against malicious adversaries

We shall now present a protocol based on garbled circuits that is secure against malicious adversaries. It is a 3-party protocol, and is secure against a malicious adversary that may corrupt at most one of the three parties. While the protocol involves three parties, P_1, P_2, and P_3, only P_1 and P_2 provide inputs or obtain outputs — party P_3 is only there to assist P_1 and P_2.

The protocol requires a garbling scheme that provides *projective input encoding* (see Section 23.3.6.1).

The protocol will make use of a collision resistant hash function which is used as a non-interactive commitment scheme, as discussed in Section 8.12 (We could, in fact, using any non-interactive commitment scheme.) Specifically, the protocol makes use of a collision resistant hash function

$$H_1 : \mathcal{T} \times \mathcal{R} \to \mathcal{C},$$

where $\mathcal{T}$ is the set of input encoding tokens, $\mathcal{R}$ is some suitably large space of randomizing elements, and $\mathcal{C}$ is some finite output space (the "commitment" space).

The randomizing elements will be used to hide the tokens. This **hiding** property means that for adversarially chosen $X^{(0)}, X^{(1)} \in \mathcal{T}$, and for randomly chosen $r \in \mathcal{R}$,

$$H_1(X^{(0)}, r) \quad \text{and} \quad H_1(X^{(1)}, r)$$

are computationally indistinguishable. We could, in fact, use any secure (non-interactive) commitment scheme.

Finally we need a secure pseudo-random generator G. Let $\mathcal{S}$ be the seed space of G. We shall assume that the output space of G is large enough so that we can use its output in lieu of the random bits needed to run the circuit garbling algorithm, as well as to generate several other random objects.

23.3.7.1 The protocol

Here is the protocol for a given circuit f:

1. Upon receiving all of their input values, parties P_1 and P_2 each send each other a "ready" message:
   (a) party P_1's ready message consists of a randomly chosen seed $s \in \mathcal{S}$;
   (b) party P_2's ready message is empty.
2. Upon receiving their respective ready messages, each party P_1 and P_2 does the following, using the output of $G(s)$ in lieu of any random bits:
   (a) Compute $(\mathcal{F}, e, d) \xleftarrow{R} Garble(f)$.
   Note: because we are assuming projective input encoding, the encoding data e *is of the form*
   $$e = ((X_1^{(0)}, X_1^{(1)}), \ldots, (X_n^{(0)}, X_n^{(1)})).$$
   (b) Compute $b_i \xleftarrow{R} \{0, 1\}$ for $i = 1, \ldots, n$.
   (c) Compute $r_i^{(b)} \xleftarrow{R} \mathcal{R}$ for $i = 1, \ldots, n$ and $b = 0, 1$.
   (d) Compute $C_i^{(b)} \leftarrow H_1(X_i^{(b \oplus b_i)}, r_i^{(b)})$ for $i = 1, \ldots, n$ and $b = 0, 1$.
   (e) Send $\mathcal{F}$ along with the collection of all hashes $\{C_i^{(b)}\}_{i,b}$ to P_3.
3. Each party P_1 and P_2 also sends encodings of their input bits to P_3 as follows: for each party, for $i = 1, \ldots, n$, if that party contributes the ith bit x_i of the input vector, then it sends to P_3 the tuple (i, a_i, X_i, r_i), where $a_i := x_i \oplus b_i$, $X_i := X_i^{(x_i)}$, and $r_i := r_i^{(a_i)}$.
4. Upon receiving all of the above data from both P_1 and P_2, party P_3 does the following:

(a) Check that the values $\mathcal{F}$ and $\{C_i^{(b)}\}_{i,b}$ received from P_1 and P_2 are identical — if not, abort.

(b) For $i = 1, \ldots, n$, check that $C_i^{(a_i)} = H_1(X_i, r_i)$ — if not, abort.

(c) Compute $\mathcal{Y} \leftarrow \mathit{Eval}(\mathcal{F}, \mathcal{X})$, where $\mathcal{X} = (X_1, \ldots, X_n)$, and send $\mathcal{Y}$ to both P_1 and P_2.

5. Upon receiving $\mathcal{Y}$ from P_3, each party P_1 and P_2 computes $\boldsymbol{y} \leftarrow \mathit{Decode}(d, \mathcal{Y})$, and then either outputs $\boldsymbol{y}$ (if $\boldsymbol{y} \neq \mathsf{reject}$) or aborts (if $\boldsymbol{y} = \mathsf{reject}$).

At a high level, in Step 2, each of the parties P_1 and P_2 (using the same random seed) generate the garbled encoding $\mathcal{F}$ of the circuit, along with commitments to each token associated with each input wire, and send these to party P_3. For pair of tokens $(X_i^{(0)}, X_i^{(1)})$ associated with the ith input wire, a random bit b_i is used to randomize the order of the committed pair. In Step 3, each of the parties P_1 and P_2 open the commitments to the tokens corresponding to their own inputs. Because of the random ordering of the commitment pairs, party P_3 does not learn the input values x_i of parties P_1 and P_2. In Step 4, party P_3 checks that the data it receives from P_1 and P_2 is consistent, and then applies the garbled circuit evaluation algorithm to the garbled inputs it received, and sends the garbled output back to both P_1 and P_2, which each party then decodes.

23.3.7.2 Security of the protocol

We shall now give some intuition as to why the protocol is secure against malicious adversaries, assuming at most one party is corrupt.

The assumptions we rely on are the following:

- the collision resistance and hiding properties of the hash function H_1,
- the security of the pseudo-random generator G, and
- the obliviousness, authenticity, and output simulatability properties of the garbling scheme.

We consider in turn the cases where P_1, P_2, and P_3 are corrupt.

Party P_1 corrupt. In this case, the assumptions we need are:

- the collision resistance of the hash function H_1,
- the output simulatability of the garbling scheme.

First, observe that P_1 must send to P_3 the same values $\mathcal{F}$ and $\{C_i^{(b)}\}_{i,b}$ that P_2 sends to P_3, since otherwise, P_3 aborts in Step 4(a), in which case P_1 cannot affect the security of the protocol. Because of the collision resistance property of H_1, if P_3 does not abort in Step 4(b), we may assume that for $i = 1, \ldots, n$, if the ith input is contributed by P_1, then the corresponding tuple (i, a_i, X_i, r_i) received by P_3 in Step 4(b) satisfies $X_i = X_i^{(x_i)}$, where $x_i = a_i \oplus b_i$, the value $X_i^{(x_i)}$ was derived from the seed s by P_2 in Step 2(a), and the value b_i was derived from the seed s by P_2 in Step 2(b). This means that at this step, each of P_1's inputs x_i is effectively determined by the information P_1 sent to both P_2 and P_3, and, moreover, P_3 will use the corresponding input encoding tokens $X_i^{(x_i)}$ when it evaluates the garbled circuit. The soundness property for the protocol now follows immediately. The input independence property is also clear, since at the time P_1's inputs are determined, it literally has no information related to P_2's inputs. The privacy property for the protocol follows from the output simulatability property of the garbling scheme. Indeed, the only information P_1 obtains during the protocol is the garbled output $\mathcal{Y}$. This is information P_1 could already compute by itself from the circuit output vector $\boldsymbol{y} = f(\boldsymbol{x})$, along with the decoding data $\mathcal{d}$ as derived in Step 2(a) from the value s that P_1 shared with P_2 in Step 1. Thus, party P_1 learns no information about P_2's input, other than that implied by the circuit output vector $\boldsymbol{y}$.

Party P_2 corrupt. This is essentially the same argument as the case where P_1 is corrupt, with the roles of P_1 and P_2 reversed.

Party P_3 corrupt. In this case, the assumptions we need are:

- the hiding properties of the hash function H_1,
- the security of the pseudo-random generator G, and
- the obliviousness and authenticity properties of the garbling scheme.

First, by the security of the pseudo-random generator G, we can assume that the data generated by P_1 and P_2 was generated using truly random bits, rather than bits output by G. By the hiding property of H_1, the only information obtained by P_3 is the garbled encoding $\mathcal{F}$ of the circuit f, and the garbled encoding $\mathcal{X}$ of the input vector $\boldsymbol{x}$. The privacy property for the protocol follows from the obliviousness property of the garbling scheme: from $\mathcal{F}$ and $\mathcal{X}$, party P_3 learns nothing at all about $\boldsymbol{x}$. The soundness property for the protocol follows from the authenticity property of the garbling scheme: party P_3 must compute $\mathcal{Y} = \mathit{Eval}(\mathcal{F}, \mathcal{X})$, as otherwise, parties P_1 and P_2 will reject with overwhelming probability. Note that the exchange of "ready" messages in Step 1 of the protocol ensures that all of P_1's and P_2's inputs are locked in before P_3 sees $\mathcal{F}$; otherwise, we would actually need to assume *adaptive* security properties for the garbling scheme. The input independence property for the protocol holds trivially, since P_3 contributes no inputs.

23.3.7.3 An optimization

A simple optimization can reduce the amount of data transmitted over the network by a factor of two. Recall that in Step 2(e), both parties P_1 and P_2 send $\mathit{gbData} := \big(\mathcal{F}, \big\{C_i^{(b)}\big\}_{i,b}\big)$ to P_3, and then in Step 4(a), party P_3 verifies that it received the same value of gbData from both P_1 and P_2.

The optimization is to have one of the parties, say P_2, send to P_3 a collision-resistant hash h of gbData and to have P_3 verify that the hash of the value gbData it receives from P_1 is equal of the value h it receives from P_2. In the security analysis, the collision-resistance property of this hash function is only needed in the analysis where P_1 is corrupt.

23.4 Multi-party computation based on a secure distributed core

In practice, a typical way to build a multi-party protocol is to start with a secure 3-party protocol, and to use that 3-party protocol as a service provided to a large set of parties. We show how this can be done using the maliciously secure version of Beaver's 2.5-party protocol (Section 23.2.4).

Recall that Beaver's 2.5-party protocol makes use of three parties, P_1, P_2, and D. Only P_1 and P_2 supply inputs and obtain outputs, while the dealer D is only used to distribute information to P_1 and P_2 as part of a pre-processing phase.

Now suppose we have N parties $Q_1, \ldots, Q_N$ who want to securely evaluate an arithmetic circuit over $\mathbb{Z}_q$. Each party Q_j supplies zero or more inputs and obtains zero or more outputs. Each input or output is an element of $\mathbb{Z}_q$. The Q_j's will use the "secure core" consisting of P_1, P_2, and D to evaluate the circuit. We shall present a protocol that is secure provided at most one of P_1, P_2, or D are corrupt, while *any number* of the Q_j's may be corrupt. Moreover, the Q_j's participate in a very simple way, and the overall structure of the protocol is as follows:

- The Q_j's begin by appropriately sharing each of their inputs with P_1 and P_2.
- After running the pre-processing phase with D, parties P_1 and P_2 run the secure circuit evaluation protocol on an appropriately modified arithmetic circuit.
- Finally, P_1 and P_2 forward their outputs to the Q_j's.

23.4.1 Processing inputs

We first describe how each Q_j processes its inputs.

Processing inputs: a first idea. Here is a first idea that does not quite work. For each input $x \in \mathbb{Z}_q$ that it supplies, party Q_j breaks $x \in \mathbb{Z}_q$ into random shares $x_1, x_2 \in \mathbb{Z}_q$, subject to $x = x_1 + x_2$, and gives x_1 to P_1 and x_2 to P_2. The arithmetic circuit evaluated by Beaver's protocol is then modified so that instead of x being provided as an input to the circuit, x_1 is an input supplied by P_1, and x_2 is an input supplied by P_2, and an addition gate is added to the circuit that computes x as $x_1 + x_2$.

With this approach, a corrupt party P_i learns nothing about Q_j's input x. Unfortunately, except in the honest-but-curious setting, this approach does not work, as there is nothing preventing a corrupt P_i from changing its input x_i into something else, thereby causing the circuit to compute an incorrect result. That incorrect result may leak unintended information about Q_j's secret input, for example, if the circuit output is meant to become publicly available.

To give a concrete example, suppose that the circuit computes the function $f(x) = x(x-1)$ and makes the result public, where x is Q_j's input. If the output is zero, then everyone learns that $x \in \mathbb{Z}_q$ is binary (i.e., $x \in \{0,1\}$), but without learning the exact value of x. Now suppose Q_j writes $x = x_1 + x_2$, and sends x_1 to P_1 and x_2 to P_2. A corrupt P_1 can add one to x_1 and use that as its input to the protocol. This effectively changes Q_j's input to the protocol from x to $x+1$. Now, when the incorrect result is published, if $x = 0$ then the output is zero, but if $x = 1$ the output is non-zero. Hence, if x is binary, then the incorrect result leaks that exact value of x.

Processing inputs: a second idea. Roughly speaking, for $i = 1, 2$, the idea is to use a simple authentication code to force each party P_i to use the share x_i it was given by Q_j. This problem is only relevant when Q_j is honest.

1. The first step is for Q_j to choose $A, B \in \mathbb{Z}_q$ at random, set $C \leftarrow AB$, and generate random sharings of A, B, and C, that is, random $A_1, A_2, B_1, B_2, C_1, C_2 \in \mathbb{Z}_q$, subject to $A = A_1 + A_2$, $B = B_1 + B_2$, and $C = C_1 + C_2$. Party Q_j sends A_i, B_i, C_i, to party P_i for $i = 1, 2$.

2. For each input $x \in \mathbb{Z}_q$ that it supplies, party Q_j will do the following:
   (a) compute $E \leftarrow Ax \in \mathbb{Z}_q$,
   (b) generate random sharings of x and E, that is, random $x_1, x_2, E_1, E_2 \in Z_q$, subject to $x = x_1 + x_2$ and $E = E_1 + E_2$;
   (c) send x_i, E_i, to party P_i, for $i = 1, 2$.

3. Upon receiving A_i, B_i, C_i, party P_i chooses random $R_i \in \mathbb{Z}_q$, and supplies inputs A_i, B_i, C_i, R_i to a modified circuit, described below.

4. For each input x supplied by Q_j, upon receiving x_i, E_i from Q_j, party P_i chooses random $S_i \in \mathbb{Z}_q$, and supplies inputs x_i, E_i, S_i to the modified circuit.

We next describe the modified circuit:

1. For each Q_j, the circuit computes a *validity code*
$$D = (R_1 + R_2)[(A_1 + A_2)(B_1 + B_2) - (C_1 + C_2)] \in \mathbb{Z}_q,$$
based on the inputs A_i, B_i, C_i, R_i supplied by P_i for $i = 1, 2$.

2. For Q_j and each input x supplied by Q_j, the circuit computes a *validity code*
$$F = (S_1 + S_2)[(A_1 + A_2)(x_1 + x_2) - (E_1 + E_2)],$$
based on the inputs x_i, E_i, S_i from P_i for $i = 1, 2$.

3. The circuit computes the sum V of all validity codes (for all Q_j's and all of their inputs from Steps 1 and 2 above).

4. The rest of the circuit is the same as the original circuit, using $x_1 + x_2$ in place of x.

5. Before obtaining any other outputs, parties P_1 and P_2 both obtain the value V.

6. Each party P_i checks that $V = 0$, and aborts if it is not. We discuss below how other circuit outputs are processed (which may require additional modifications to the circuit).

Note that to implement the above strategy, we exploit the fact that in Beaver's 2.5-party protocol, party P_1 needs to actively participate in order for P_2 to obtain an output; likewise, party P_2 needs to actively participate in order for P_1 to obtain an output. Thus, we can implement the logic in Steps 5 and 6 that requires that both parties first obtain V and verify that $V = 0$ before allowing any other outputs to be obtained.

Analysis assuming P_1 corrupt and P_2 and Q_j honest. Assume that P_1 is corrupt and Q_j is honest. The case where P_2 is corrupt and Q_j is honest is symmetric. There are other cases to consider, such as when both P_1 and Q_j are corrupt, or when D is corrupt and Q_j is honest, but these are less interesting.

The values A_2, B_2, C_2 by themselves reveal no information to P_1 about A or B. Now suppose P_1 supplies inputs to the circuit $A_2 + \delta_A$, $B_2 + \delta_B$, and $C_2 + \delta_C$, in place of A_2, B_2, C_2. Then in this case, if $R = R_1 + R_2$, the validity code D is computed as
$$D = R[(A + \delta_A)(B + \delta_B) - (C + \delta_C)] = R[A\delta_B + B\delta_A + \delta_A\delta_B - \delta_C].$$

On the one hand, if any of δ_A, δ_B, or δ_C are non-zero, then because A, B, and R are independent of each other and of everything else in P_1's view, the computed validity code D will be a random element of $\mathbb{Z}_q$. On the other hand, if $\delta_A = \delta_B = \delta_C = 0$, the computed validity D code is zero.

Now consider what happens in processing one input x. The values x_2, E_2 by themselves reveal no information to P_1 about A or x. Now suppose P_1 supplies inputs $x_2 + \delta_x$ and $E_2 + \delta_E$ to the circuit, in place of x_2 and E_2. Also assume that δ_A, defined above, is zero. Then in this case, if $S = S_1 + S_2$, the validity code F is computed as

$$F = S[A(x + \delta_x) - (E + \delta_E)] = S[A\delta_x - \delta_E].$$

On the one hand, if either of δ_x or δ_E are non-zero, then because A and S are independent of each other and of everything else in P_1's view, the computed validity code F will be a random element of $\mathbb{Z}_q$. On the other hand, if $\delta_x = \delta_E = 0$, the computed validity code F is zero.

It follows that if P_1 ever supplies the wrong inputs, the value V will be a random value in $\mathbb{Z}_q$, and hence with probability $1 - 1/q$, party P_2 aborts the protocol before allowing any outputs are provided to any of the Q_j's.

23.4.2 Processing outputs

Processing outputs: a first idea. Suppose the value x on some wire is to be obtained as an output by Q_j. If x is a public output that may also be obtained by anyone, then P_1 and P_2 can both be allowed to obtain x and then just forward x to Q_j. Then Q_j checks that it received the same value x from both P_1 and P_2. If not, Q_j aborts; otherwise, the common value x is the output obtained by Q_j.

It should be clear that even if one of P_1 and P_2 is corrupt, party Q_j is guaranteed to either obtain the correct output or abort.

Processing outputs: a second idea. If the output x is a "private output" that should be obtained only by Q_j, then the protocol is easily modified to deal with that as follows. For each such output x, party Q_j generates a random "output mask" $G \in \mathbb{Z}_q$, which is supplied as an additional input, subject to the same processing as an ordinary input, as in Section 23.4.1. The circuit itself is modified to compute the "masked output" $x' \leftarrow x + G$, rather than the "unmasked output" x. Party Q_j checks that it receives the same value x' from both P_1 and P_2. If not, Q_j aborts; otherwise, Q_j computes the unmasked output $x \leftarrow x' - G$.

It should be clear that even if one of P_1 and P_2 is corrupt, party Q_j is guaranteed to either obtain the correct output or abort. Moreover, neither party P_1 or P_2 learns anything about the unmasked output x.

23.5 Formal models for multi-party computation: the universal composability framework

We now turn to presenting a precise security model for multi-party computation (MPC). Such a model is needed in order to prove that a proposed MPC protocol has the claimed security properties. Without a formal security model we cannot rigorously prove security; we can only give an intuitive argument of security, and such arguments can miss clever attacks that the protocol designers were not aware of.

In Section 23.1.3, we briefly introduced the basic ideas of our formal security definition. The reader should review that section, if necessary, before continuing.

23.5.1 The real protocol and its execution

Let us describe in more detail an N-party protocol and its execution. We have N parties $P_1, \ldots, P_N$, each executing some protocol Π. We shall assume that each party P_i knows its own index i. There is, of course, an adversary $\mathcal{A}$. For now, we will focus on completely malicious adversaries. Later, in Section 23.5.6, we will consider honest-but-curious adversaries. If any of the parties $P_1, \ldots, P_N$ are corrupt, we simply absorb the behavior of the corrupt parties into the logic of $\mathcal{A}$ itself. This models the fact that corrupt parties may collude with each other and with "outside" attackers, so it is simpler to just roll this all together into a single entity modeled by $\mathcal{A}$.

We assume that the parties communicate over an asynchronous communication network $\mathcal{C}$. We will generally assume that $\mathcal{C}$ provides secure point-to-point channels. Each point-to-point channel provides both message privacy and integrity. Assuming keys have been pre-distributed, such secure point-to-point channels can be implemented using the techniques discussed in Chapter 9. However, in this chapter, we will abstract all of this away, and simply model $\mathcal{C}$ as providing *physically secure* point-to-point channels, which is essentially justified by the analysis in Section 9.3. Nevertheless, the adversary learns some information about and exerts some control over messages sent over these channels:

- when a message is sent, the adversary learns of this fact, as well as the length of the message, the sender of the message, and its intended recipient — this models the fact that encryption schemes leak some information about the length of the encrypted message, and the fact that in any real implementation of the network, the adversary may see who sent the message and who receives the message;

- the adversary determines if and when a message is delivered — this models the fact that we are viewing $\mathcal{C}$ as an asynchronous communication network, with no guarantees on if and when a message is delivered, so we simply let the adversary make these scheduling decisions;

- the adversary learns no other information about a message; nor can he modify a message or its apparent sender; nor can he deliver it to anyone other than its intended recipient.

There is one more element in the picture, which we call the **environment**. The environment is responsible for supplying inputs to the honest parties, and obtaining outputs from honest parties. It also determines which parties are corrupt and which are honest. It also may coordinate and interact with $\mathcal{A}$ during the execution of the protocol. We denote the environment by $\mathcal{Z}$.

Fig. 23.8 illustrates the basic topology of a real protocol execution.

- In this figure, there are 4 parties, $P_1, \ldots, P_4$; however, we are assuming here that parties P_3 and P_4 are corrupt, and absorbed into the adversary $\mathcal{A}$.

- The communication network $\mathcal{C}$ has 4 wires, which connect to each of the 4 parties; however, since parties P_3 and P_4 are absorbed into $\mathcal{A}$, these wires connect directly to $\mathcal{A}$. In this way, $\mathcal{A}$ can send a message, say, to P_1 as if it came from P_3, simply by sending the message along the wire labeled 3. Similarly, $\mathcal{A}$ may receive a message from P_1 intended for P_3 along the same wire. We call these *$\mathcal{C}$-I/O wires.*

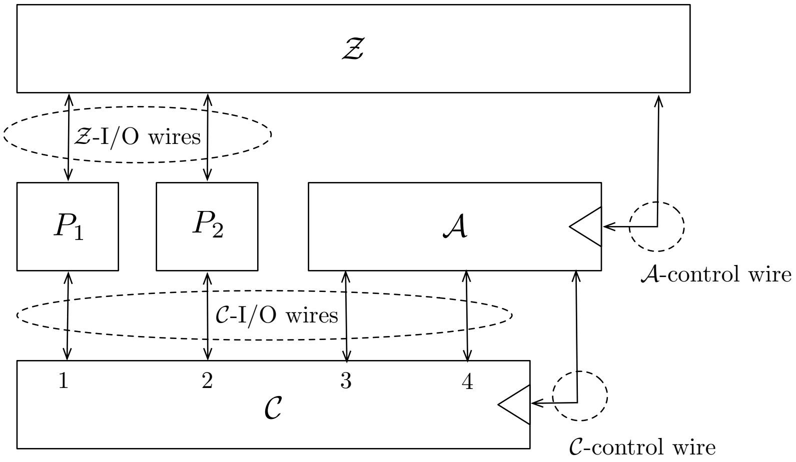


Figure 23.8: Real protocol execution, including communication channel $\mathcal{C}$, adversary $\mathcal{A}$, and environment $\mathcal{Z}$, along with honest parties P_1 and P_2, and corrupt parties P_3 and P_4

- There is also a "control wire", which we call the *$\mathcal{C}$-control wire*, between $\mathcal{A}$ and $\mathcal{C}$. Along this wire, $\mathcal{A}$ receives "leakage messages" from $\mathcal{C}$, which give $\mathcal{A}$ partial information about protocol messages sent from an honest party. Also along this control wire, $\mathcal{A}$ may send to $\mathcal{C}$ "control messages", which instruct $\mathcal{C}$ about when to actually deliver protocol messages to an honest party.

- There is also an environment $\mathcal{Z}$, which has wires connecting it to the honest parties P_1 and P_2, through which the environment can send protocol inputs and receive protocol outputs. We call these the *$\mathcal{Z}$-I/O wires*. There is also a "control wire", which we call the *$\mathcal{A}$-control wire*, that allows communication directly between $\mathcal{Z}$ and $\mathcal{A}$. The idea is that along this wire, $\mathcal{Z}$ may send "control messages" to $\mathcal{A}$, which may give $\mathcal{A}$ guidance on what it should do next, and $\mathcal{Z}$ may receive "leakage messages" from $\mathcal{A}$, which represent information that leaks to the environment.

Remark 23.1 (The role of the environment). As its name suggests, the environment models the environment in which a protocol executes. Indeed, a protocol may be used as a subprotocol in some larger protocol, and all of the elements of the larger protocol are abstracted away and modeled by a single entity modeled by $\mathcal{Z}$. Because the environment is allowed allowed interact with the adversary during protocol execution, we can model possible interactions between the subprotocol and the larger protocol. This will allow us to prove important *composition theorems*, which (intuitively) say that if a protocol is secure, then it remains secure when the protocol is used as a subprotocol in *any* larger protocol, and also when many instances of the protocol are run concurrently. Because of these composition theorems, the security model we present here is called the **universal composability framework**. □

23.5.1.1 The real protocol execution

We now describe the detailed mechanics of a real protocol execution, which involves a number of *machines*. There is a machine representing $\mathcal{A}$, a machine representing $\mathcal{Z}$, a machine representing

$\mathcal{C}$, and a separate machine representing each individual honest party P_i. So, for example, every box in Fig. 23.8 is a machine. Communication among machines is done by message passing. Moreover, our formal model of protocol execution only allows one machine to execute at a time; nevertheless, this model still captures concurrent execution of multiple machines, with the adversary, in effect, controlling how the computations of various machines are interleaved.

After computing for a while, the currently executing machine passes a message to a second machine; at that point, the first machine suspends execution, and the second machine resumes execution. This continues for the entirety of the protocol. Initially, $\mathcal{Z}$ starts executing and determines which subset of the parties $P_1, \ldots, P_N$ is to be considered corrupt. This information is written to a special tape that is visible to $\mathcal{A}$ and $\mathcal{C}$. Note, however, that this information is not visible to the honest parties, reflecting the fact that honest parties do not know who the corrupt parties are. Protocol execution halts when $\mathcal{Z}$ halts. Before halting, $\mathcal{Z}$ writes a single bit to a special output tape.

Next, we describe the types of messages that may be passed between machines.

1. $\mathcal{Z}$ may pass a protocol input to any honest party P_i along a $\mathcal{Z}$-I/O wire.

2. $\mathcal{Z}$ may pass a control message to $\mathcal{A}$ along the $\mathcal{A}$-control wire.

3. An honest party P_i, or $\mathcal{A}$ on behalf of a corrupt party P_i, may pass a *send messages request* to $\mathcal{C}$ along the $\mathcal{C}$-I/O wire corresponding to P_i. Such a *send messages request* is of the form $(\texttt{send}, (j_1, m_1), (j_2, m_2), \ldots)$, where each j_k is the intended recipient of the message, and m_k is the actual payload of the message.

4. $\mathcal{A}$ may pass a *deliver message request* along the $\mathcal{C}$-control wire. Such a *deliver message request* is of the form $(\texttt{deliver}, k)$, where k is an index that determines the message that is to be delivered — details of this are described below.

5. $\mathcal{A}$ may pass a leakage message to $\mathcal{Z}$ along the $\mathcal{A}$-control wire.

6. An honest party P_i may pass a protocol output to $\mathcal{Z}$ along its $\mathcal{Z}$-I/O wire.

7. Upon receiving a *send messages request* $(\texttt{send}, (j_1, m_1), (j_2, m_2), \ldots)$ along the $\mathcal{C}$-I/O wire of party P_i (either honest or corrupt), the communication network $\mathcal{C}$ appends the tuples $(i, j_1, m_1), (i, j_2, m_2), \ldots$ to an internal list *buf* (initially empty), and passes the leakage message $(\texttt{sent}, i, (j_1, \text{len}(m_1)), (j_2, \text{len}(m_2)), \ldots)$ to $\mathcal{A}$ along the $\mathcal{C}$-control wire.

8. Upon receiving a *deliver message request* $(\texttt{deliver}, k)$ from $\mathcal{A}$ along the $\mathcal{C}$-control wire, the communication network $\mathcal{C}$ does the following: if the kth tuple in *buf* is (i, j, m), it passes the message $(\texttt{receive}, i, m)$ along the $\mathcal{C}$-I/O wire corresponding to P_j, and this goes to either P_j (if P_j is honest) or $\mathcal{A}$ (if P_j is corrupt); if there is no kth tuple, it passes an error message to $\mathcal{A}$ along the $\mathcal{C}$-control wire.

23.5.1.2 Direct communication between the adversary and honest parties

As it stands, the adversary $\mathcal{A}$ cannot interact directly with the honest parties P_i: all such interactions are performed indirectly through the secure network $\mathcal{C}$. While not strictly necessary, we may also allow direct interaction between the adversary and the honest parties. This is especially useful when we consider variations on the model in which instead endowing the model with a built-in secure communication network $\mathcal{C}$, we use the adversary to model an insecure communication network, and effectively construct a secure communication channel using some other type of built-in functionality (for example, a public-key infrastructure). We will see an example of this in Section 23.6.1.2.

23.5.1.3 The trivial adversary

For a variety of reasons, it is convenient to introduce a special adversary, called the "trivial adversary", denoted as $\mathcal{A}_{\text{triv}}$. The trivial adversary acts as a simple "router" between the environment and other machines: control messages from $\mathcal{Z}$ are instructions that tell $\mathcal{A}_{\text{triv}}$ to send a specific message to a specific machine along a specific wire; moreover, messages received by $\mathcal{A}_{\text{triv}}$ from any machine other than $\mathcal{Z}$ are forwarded to $\mathcal{Z}$ (along with information indicating from which machine/wire the message came from).

23.5.1.4 Running time considerations

When considering such a system, we need to restrict ourselves to protocols, adversaries, and environments that are efficient in some appropriate sense. It is a bit tricky to get this type of definition correct. One workable approach is as follows. We define two notions.

- We say an environment $\mathcal{Z}$ is **well behaved** if its running time is strictly bounded by a polynomial (which may depend on $\mathcal{Z}$) in the security parameter.
- We say that $\mathcal{A}$ is **compatible with** Π if for every well-behaved $\mathcal{Z}$, the total running time of the honest P_i's and $\mathcal{A}$ is bounded (with overwhelming probability) by some polynomial (which depends on Π and $\mathcal{A}$ but not on $\mathcal{Z}$) in the security parameter and the sum of the lengths of all messages sent by $\mathcal{Z}$ along the $\mathcal{Z}$-I/O wires and the $\mathcal{A}$-control wire.

With these definitions, if $\mathcal{Z}$ is well behaved and $\mathcal{A}$ is compatible with Π, then the overall running time of all machines will be bounded (with overwhelming probability) by some polynomial (which depends on Π, $\mathcal{A}$, and $\mathcal{Z}$) in the security parameter. We make one more definition:

- Protocol Π is **efficient** if the trivial adversary is compatible with Π.

This notion of an efficient protocol essentially says that the running time of all the machines in the protocol will be bounded, with overwhelming probability, by a fixed polynomial in the amount of data that flows into the protocol from the environment. This definition is flexible enough to accommodate protocols that process an *a priori* unbounded number of requests from their environment.

23.5.2 The ideal protocol and its execution

We now describe the ideal protocol execution. The main changes from the real protocol execution are as follows:

- There is a different adversary $\mathcal{S}$, which we shall call the *simulator*.
- Instead of a communication network, there is an *ideal functionality* $\mathcal{F}$. The logic of the ideal functionality depends on the goals of the protocol. Below, we will describe an ideal

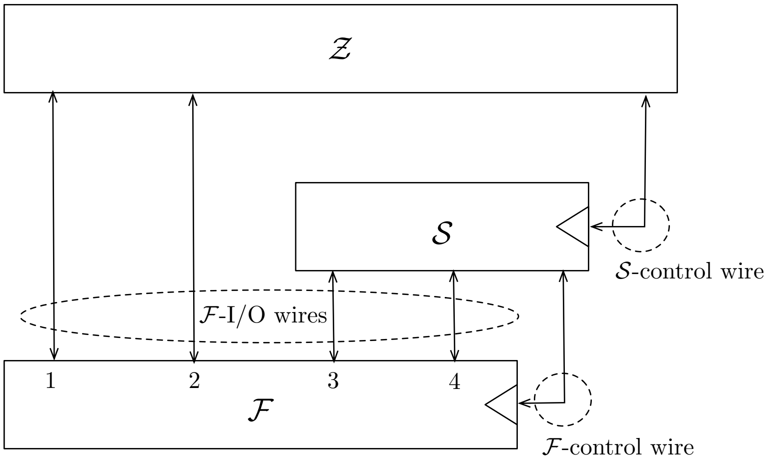


Figure 23.9: Ideal protocol execution, including the ideal functionality $\mathcal{C}$, simulator $\mathcal{S}$, and environment $\mathcal{Z}$, along with honest parties P_1 and P_2, and corrupt parties P_3 and P_4

functionality suitable for the secure function evaluation problem, which basically receives inputs from all parties (both honest and corrupt), evaluates the function, and then delivers outputs to all parties (again, both honest and corrupt). However, other protocols for other tasks may be best modeled by different ideal functionalities.

- The honest parties themselves play no active role in the protocol: any input messages they receive from $\mathcal{Z}$ are passed directly to $\mathcal{F}$, and any output messages they receive from $\mathcal{F}$ are output directly to $\mathcal{Z}$.

Fig. 23.9 illustrates the basic topology of an ideal protocol execution. Instead of $\mathcal{C}$-I/O wires, we have $\mathcal{F}$-I/O wires: for honest parties, these connect directly to the environment, and for corrupt parties, these connect to $\mathcal{S}$. In place of the $\mathcal{C}$-control wire, we have an $\mathcal{F}$-control wire, connecting $\mathcal{S}$ and $\mathcal{F}$. In place of the $\mathcal{A}$-control wire, we have an $\mathcal{S}$-control wire, connecting $\mathcal{Z}$ and $\mathcal{S}$.

As will become apparent soon, an important feature of this system is that both the real and ideal protocols present exactly the same interface to the environment $\mathcal{Z}$.

23.5.2.1 The ideal protocol execution

We now describe the mechanics of an ideal protocol execution. As in the real protocol execution, initially, $\mathcal{Z}$ starts executing and determines which subset of $P_1, \ldots, P_N$ is to be considered corrupt. This information is written to a special tape that is also visible to $\mathcal{S}$ and $\mathcal{F}$. Also, protocol execution halts when $\mathcal{Z}$ halts. Before halting, $\mathcal{Z}$ writes a single bit to a special output tape.

Next, we describe the types of messages that may be passed between machines.

1. $\mathcal{Z}$ may pass a protocol input to any honest party P_i along an $\mathcal{F}$-I/O wire, which is forwarded directly to $\mathcal{F}$.

2. $\mathcal{Z}$ may pass a control message to $\mathcal{S}$ along the $\mathcal{S}$-control wire.

3. $\mathcal{S}$ may pass a protocol input to $\mathcal{F}$ along one of the $\mathcal{F}$-I/O wires corresponding to a corrupt party, or a control message to $\mathcal{F}$ along the $\mathcal{F}$-control wire.

4. $\mathcal{S}$ may pass a leakage message to $\mathcal{Z}$ along the $\mathcal{S}$-control wire.

5. $\mathcal{F}$ may do one of the following:

   (a) pass a protocol output to $\mathcal{Z}$ along one of the $\mathcal{F}$-I/O wires corresponding to an honest party;

   (b) pass a protocol output to $\mathcal{S}$ along one of the $\mathcal{F}$-I/O wires corresponding to a corrupt party;

   (c) pass a leakage message to $\mathcal{S}$ along the $\mathcal{F}$-control wire.

23.5.2.2 Running time considerations

Similar to what we did above, we need to restrict ourselves to ideal functionalities, simulators, and environments that are efficient in some appropriate sense. We make the following definition:

- We say that $\mathcal{S}$ is **compatible with** $\mathcal{F}$ if for every well-behaved $\mathcal{Z}$, the total running time of $\mathcal{F}$ and $\mathcal{S}$ is bounded (with overwhelming probability) by some polynomial (which depends on $\mathcal{F}$ and $\mathcal{S}$ but not on $\mathcal{Z}$) in the security parameter and the sum of the lengths of all messages sent by $\mathcal{Z}$ along the $\mathcal{F}$-I/O wires and the $\mathcal{S}$-control wire.

With this definition, if $\mathcal{Z}$ is well behaved and $\mathcal{S}$ is compatible with $\mathcal{F}$, then the overall running time of all machines will be bounded (with overwhelming probability) by some polynomial (which depends on $\mathcal{F}$, $\mathcal{S}$, and $\mathcal{Z}$) in the security parameter.

23.5.3 Example: the ideal functionality for secure function evaluation

We now describe the ideal functionality, $\mathcal{F}_{\text{sfe}}$, for secure function evaluation. At a high level, $\mathcal{F}_{\text{sfe}}$ receives protocol inputs from all parties (both honest and corrupt) and delivers protocol outputs to all parties (both honest and corrupt). $\mathcal{F}_{\text{sfe}}$ acts as a "trusted third party", in the sense that, by design, it computes all outputs correctly, and leaks no additional information to the adversary (other than the outputs delivered to corrupt parties). However, $\mathcal{F}_{\text{sfe}}$ does allow the adversary to control when and if honest parties actually receive their outputs. In particular, the design of $\mathcal{F}_{\text{sfe}}$ provides no guarantee of output delivery: an honest party may fail to receive some or all of its outputs (even if the corrupt parties receive all of their outputs).

In somewhat more detail, $\mathcal{F}_{\text{sfe}}$ works as follows. First of all, we assume that the function itself is publicly known and fixed in advance. Typically, such a function may be described by a boolean or arithmetic circuit, but our presentation here is more general. Nevertheless, we shall still speak of *input wires* and *output wires* of the function. We assume that each input wire is labeled with

- an *inputID* that identifies the wire, and

- the index of the party that is designated to supply the value on that wire.

We also assume that each output wire is labeled with

- an *outputID* that identifies the wire, and

- a list of the indices of the parties that are designated to obtain that output.

Here is how the functionality $\mathcal{F}_{\text{sfe}}$ operates:

1. The functionality $\mathcal{F}_{\text{sfe}}$ may receive a protocol input along a $\mathcal{F}_{\text{sfe}}$-I/O wire, either from an honest party P_i or from the adversary $\mathcal{S}$ on behalf of a corrupt party P_i. A protocol input is a tuple of the form $(\texttt{input}, \textit{inputID}, x)$. As a pre-condition, party P_i must be the party that is designated to supply this input value, and must not already have supplied a value for this input wire. The functionality $\mathcal{F}_{\text{sfe}}$ records the tuple $(\texttt{input}, \textit{inputID}, x)$ and passes to $\mathcal{S}$, along the $\mathcal{F}_{\text{sfe}}$-control wire, the leakage message $(\texttt{input}, \textit{inputID})$.

2. The functionality $\mathcal{F}_{\text{sfe}}$ may receive a control message from $\mathcal{S}$ along the $\mathcal{F}_{\text{sfe}}$-control wire. Such a control message is a tuple of the form $(\texttt{output}, \textit{outputID}, i)$. As a pre-condition, all inputs for all parties (both honest and corrupt) must have already been supplied, and i must be in the list of parties designated to obtain this output value. Functionality $\mathcal{F}_{\text{sfe}}$ computes the appropriate output value y and passes the tuple $(\texttt{output}, \textit{outputID}, y)$ along the $\mathcal{F}_{\text{sfe}}$-I/O wire corresponding to party P_i, which goes to P_i (if P_i is honest) or to $\mathcal{S}$ (if P_i is corrupt).

23.5.3.1 Probabilistic functionalities

So far, we have only discussed $\mathcal{F}_{\text{sfe}}$ for functions. However, one can generalize $\mathcal{F}_{\text{sfe}}$ to cover probabilistic functions as well. Essentially, this means that the function computed takes an auxiliary random input, which is supplied by the functionality $\mathcal{F}_{\text{sfe}}$ itself.

23.5.4 Secure implementation: a strong security notion

We can now formally define protocol security.

- We denote by $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}]$ the random variable representing the output of an environment $\mathcal{Z}$ interacting with a protocol Π and an adversary $\mathcal{A}$ in the real world execution.

- We denote by $\text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}]$ the random variable representing the output of an environment $\mathcal{Z}$ interacting with an ideal functionality $\mathcal{F}$ and a simulator $\mathcal{S}$ in the ideal world execution.

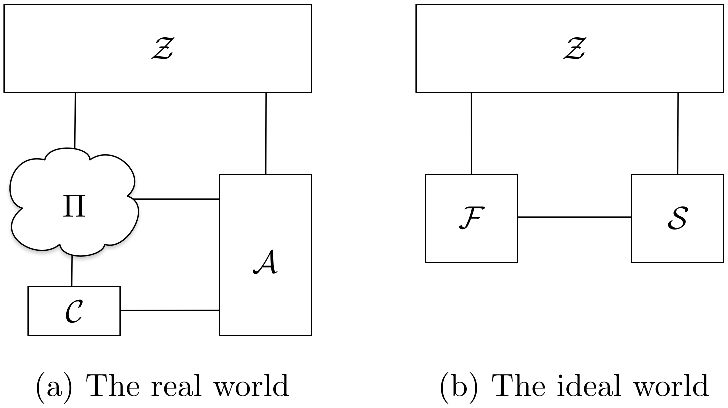
To streamline notation in this section, for random variables X, Y, we will write $X \approx Y$ to mean X and Y are statistically indistinguiable (as in Section 3.11). When X and Y are 0/1-valued random variables, this is equivalent to saying that $|\Pr[X = 1] - \Pr[Y = 1]|$ is negligible.

We begin with a preliminary definition that defines security with respect to a specific adversary.

Definition 23.5. *Let $\mathcal{A}$ be an adversary that is compatible with protocol Π. We say that Π* ***securely implements $\mathcal{F}$ against*** *$\mathcal{A}$ if there exists a simulator $\mathcal{S}$ (which may depend on $\mathcal{A}$) that is compatible with $\mathcal{F}$, such that for every well-behaved environment $\mathcal{Z}$, we have*

$$\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}].$$

Fig. 23.10 gives a schematic diagram for Definition 23.5. Fig. 23.10(a) shows the real world. The cloud labeled Π represents the honest protocol machines $P_1, P_2, \ldots$ that are running protocol Π. The protocol machines interact directly with the environment $\mathcal{Z}$, with the communication network $\mathcal{C}$, and with the adversary $\mathcal{A}$. The adversary $\mathcal{A}$ may also interact directly with the environment and



(a) The real world (b) The ideal world

Figure 23.10: Schematic diagram for Definition 23.5

communication network. Fig. 23.10(b) shows the ideal world. Here, instead of interacting with the honest protocol machines, the environment instead interacts with the ideal functionality $\mathcal{F}$, and instead of interacting with the adversary $\mathcal{A}$, the environment instead interacts with the simulator $\mathcal{S}$. The simulator and ideal functionality may also interact directly with one another. Definition 23.5 essentially says that there is no environment that can effectively distinguish between the real world and ideal world.

The next definition defines security with respect to all adversaries. This definition captures what we mean by a secure protocol:

Definition 23.6. *We say that* Π ***securely implements*** $\mathcal{F}$ *if* Π *securely implements* $\mathcal{F}$ *against* $\mathcal{A}$ *for every adversary* $\mathcal{A}$ *that is compatible with* Π*.*

Unpacking this definition, it says that Π securely implements $\mathcal{F}$ if and only if for every adversary $\mathcal{A}$ (compatible with Π) there exists a simulator $\mathcal{S}$ (compatible with $\mathcal{F}$, and which may depend on $\mathcal{A}$), such that for every well-behaved environment $\mathcal{Z}$, we have $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}]$.

Hopefully, it is clear that this formal definition captures the intuition described at the beginning of this chapter in Section 23.1.3. Security means that the damage caused by any adversary $\mathcal{A}$ attacking the real protocol Π is limited to the damage caused by another adversary $\mathcal{S}$ (which may depend on $\mathcal{A}$) attacking the ideal protocol. In the case of secure function evaluation, where the ideal functionality is $\mathcal{F}_{\text{sfe}}$, it should be clear that this implies privacy, soundness, and input independence.

A fact that is convenient in proving the security of any protocol is the following:

Theorem 23.3 (Completeness of the trivial adversary). *Let* Π *be an efficient protocol. If* Π *securely implements* $\mathcal{F}$ *against the trivial adversary, then* Π *securely implements* $\mathcal{F}$*.*

Proof sketch. Suppose Π securely implements $\mathcal{F}$ against the trivial adversary. This means there is a simulator $\mathcal{S}_{\text{triv}}$ (that is compatible with $\mathcal{F}$) such that for every well-behaved environment $\mathcal{Z}$, which interacts with the trivial adversary via the $\mathcal{A}_{\text{triv}}$-control wire, we have $\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}]$.

Now consider an adversary $\mathcal{A}$ (that is compatible with Π). Our goal is to show that Π securely implements $\mathcal{F}$ against $\mathcal{A}$. Consider a well-behaved environment $\mathcal{Z}$ that interacts with $\mathcal{A}$ via the $\mathcal{A}$-control wire. We can construct a new environment $\mathcal{A}|\mathcal{Z}$ that embeds $\mathcal{A}$ into $\mathcal{Z}$. The environment $\mathcal{A}|\mathcal{Z}$ interacts with the trivial adversary via the $\mathcal{A}_{\text{triv}}$-control wire, running the code for both $\mathcal{A}$ and $\mathcal{Z}$. (Some care must be taken to ensure that $\mathcal{A}|\mathcal{Z}$ is well behaved.) One can show that $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{A}|\mathcal{Z}]$. See Fig. 23.11 for diagrams that illustrate this and other steps in the proof.

Using the fact that Π securely implements $\mathcal{F}$ against the trivial adversary, we have $\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{A}|\mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{A}|\mathcal{Z}]$.

Finally, we can construct a new simulator $\mathcal{S}_{\text{triv}}|\mathcal{A}$ that embeds $\mathcal{A}$ into $\mathcal{S}_{\text{triv}}$. The simulator $\mathcal{S}_{\text{triv}}|\mathcal{A}$ interacts with the environment via the $\mathcal{A}$-control wire, running the code for both $\mathcal{S}_{\text{triv}}$ and $\mathcal{A}$. One can show that $\text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{A}|\mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}|\mathcal{A}, \mathcal{Z}]$ (and that $\mathcal{S}_{\text{triv}}|\mathcal{A}$ is compatible with $\mathcal{F}$).

Putting this all together, we have

$$\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{A}|\mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{A}|\mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}|\mathcal{A}, \mathcal{Z}],$$

which shows that $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}|\mathcal{A}, \mathcal{Z}]$. From this, we conclude that Π securely implements $\mathcal{F}$ against $\mathcal{A}$. □

This theorem says that to prove that Π securely implements $\mathcal{F}$, instead of showing how to build a simulator corresponding to every adversary, it suffices to just build a simulator for the trivial adversary.

Remark 23.2 (Black box reductions). The proof of Theorem 23.3 actually shows that in Definition 23.6, the simulator $\mathcal{S}$ that we need to show exists for every adversary $\mathcal{A}$ can actually be constructed using $\mathcal{A}$ as a black box. Specifically, we can set $\mathcal{S} := \mathcal{S}_{\text{triv}}|\mathcal{A}$ (see proof of Theorem 23.3). This simulator $\mathcal{S}$ is essentially an "elementary wrapper around $\mathcal{A}$" (although there are some technical differences between this and Definition 2.12.) □

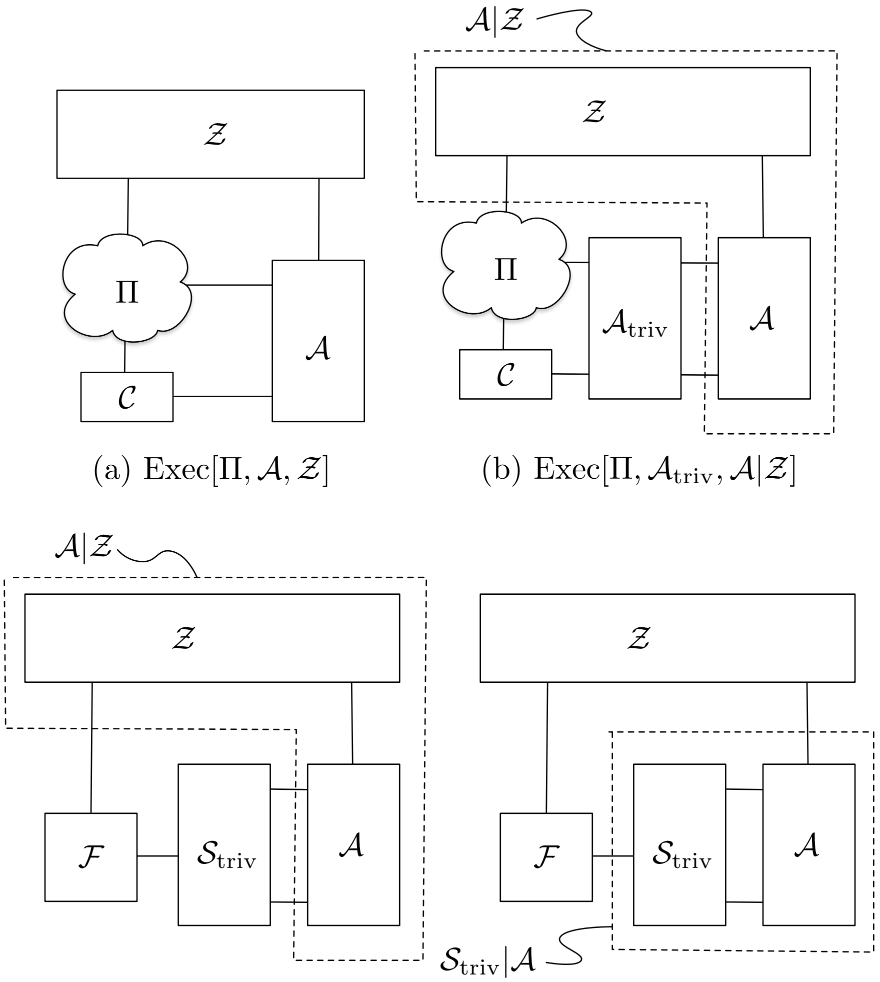
Remark 23.3 (On a simpler definition of security). In light of Theorem 23.3, in principle, we could have defined security more simply, by defining it strictly in terms of the trivial adversary. However, there are situations where we need to work with more general adversaries. Examples of this may be seen below in the proofs of Theorems 23.5 and 23.6. □

23.5.5 Consequences of secure implementation

Secure implementation is a very strong security notion. In this section we explore some useful consequences. We state some of these consequences as theorems, but only provide the proof ideas, as formal proofs would be quite tedious.

23.5.5.1 Concurrent composition

One consequence of secure implementation is that it guarantees that a secure protocol remains secure even when many instances of the protocol are allowed to execute concurrently with one another. (In Exercise 11.20, we saw an example of how a seemingly secure protocol became insecure when two instances ran concurrently.)


$\mathcal{Z}$
Π
$\mathcal{A}$
$\mathcal{C}$
$\mathcal{A}|\mathcal{Z}$
$\mathcal{Z}$
Π
$\mathcal{A}_{\text{triv}}$
$\mathcal{A}$
$\mathcal{C}$
$\mathcal{A}|\mathcal{Z}$
$\mathcal{Z}$
$\mathcal{F}$
$\mathcal{S}_{\text{triv}}$
$\mathcal{A}$
$\mathcal{Z}$
$\mathcal{F}$
$\mathcal{S}_{\text{triv}}$
$\mathcal{A}$
$\mathcal{S}_{\text{triv}}|\mathcal{A}$

(a) $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}]$

(b) $\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{A}|\mathcal{Z}]$

(c) $\text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{A}|\mathcal{Z}]$

(d) $\text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}|\mathcal{A}, \mathcal{Z}]$

Figure 23.11: Diagram for proof of Theorem 23.3

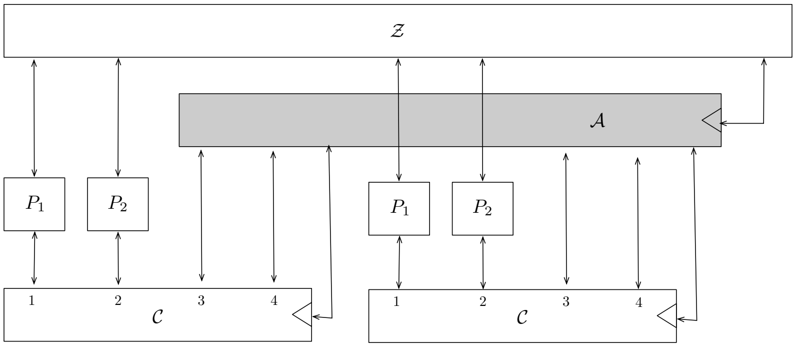


Figure 23.12: Real protocol execution with two concurrent instances

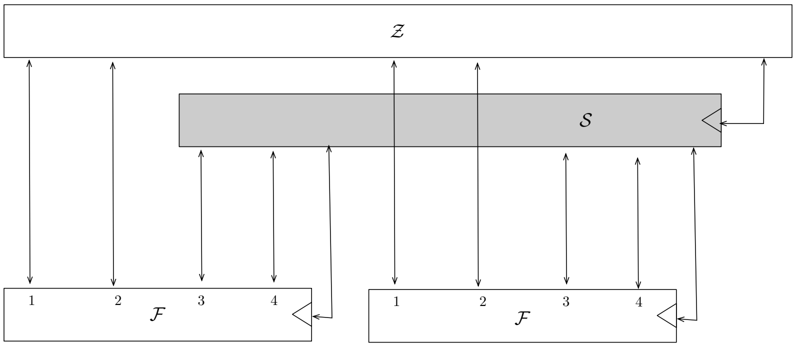


Figure 23.13: Ideal protocol execution with two concurrent instances

It takes some work to properly formulate this result. To start with, we have to generalize the notions of real and ideal protocol execution to multiple protocol instances. The essential features are as follows:

- for each instance of a real protocol, there is a separate communication network;
- for each instance of an ideal protocol, there is a separate ideal functionality.

See Figures 23.12 and 23.13, which generalize Figures 23.8 and 23.9 from one protocol instance to two instances. In these figures, we have four parties $P_1, \ldots, P_4$ that are running two independent instances of the same protocol Π. Parties P_1 and P_2 are honest and parties P_3 and P_4 are corrupt (in both instances of the protocol, in both the real and ideal executions).

Note that because the two communication networks appearing in Fig. 23.12 are separate, if P_1 in the first instance sends a message to P_2 in the first instance, it is impossible for the adversary to forward that message to P_2 in the second instance. In practice, it is easy to effectively implement such separate communication networks. For example, if each protocol instance has a unique "instance ID", then this instance ID can be included as associated data in an authenticated encryption scheme that supports associated data (see the notion of an AEAD cipher in Section 9.5). Similarly, because the two ideal functionalities appearing in Fig. 23.13 are separate, it is impossible for the adversary $\mathcal{S}$ to create any surprising interactions between them.

In fact, in any situation where we want to model the execution of several protocols, or several instances of the same protocol, we need to add to our formal model the notion of a protocol "instance ID", which can be used by all parties to identify which protocol instance a message belongs to. Every protocol machine P_i in the real world execution will be initialized not only with its index i but also with its instance ID. Similarly, an ideal functionality in the ideal world execution will be initialized with its instance ID.

We state the following theorem:

Theorem 23.4 (concurrent composition). *If* Π *is an efficient protocol in the single-instance setting, it remains an efficient protocol in the multi-instance setting. Moreover, if* Π *securely implements* $\mathcal{F}$ *in the single-instance setting, then it does so as well in the multi-instance setting.*

Proof sketch. We do not address the running time claim here, as it is proved by a fairly simple (though somewhat tedious) argument. We focus on the security claim.

We are assuming single-instance security, which means that there is a simulator $\mathcal{S}_{\text{triv}}$ (that is compatible with $\mathcal{F}$) such that for every well-behaved single-instance environment $\mathcal{Z}$, which interacts with the trivial adversary via the $\mathcal{A}_{\text{triv}}$-control wire, we have $\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}]$.

The proof is essentially a hybrid argument. For simplicity, we illustrate the proof idea with just two protocol instances. By a generalization of Theorem 23.3, we may assume the adversary in the multi-instance real world is the trivial adversary that simply routes messages between the environment and other protocol machines. This is illustrated in Fig. 23.14(a). Here, the unlabeled boxes represent simple routing machines — the top unlabeled box with which the environment interacts directly routes messages to the lower unlabeled boxes based on instance IDs that are assumed to be embedded in messages.

By virtue of single-instance security, we can replace the first real-world protocol instance by a corresponding ideal world protocol instance. This is illustrated in Fig. 23.14(b). To formally justify this step, we define a single-instance environment $\mathcal{Z}_1$ that absorbs the second protocol instance (as illustrated by the dashed box in the figure). Single-instance security implies that $\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}_1] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}_1]$, which implies that the multi-instance environment $\mathcal{Z}$ cannot distinguish between the real world in Fig. 23.14(a) and the hybrid world in Fig. 23.14(b).

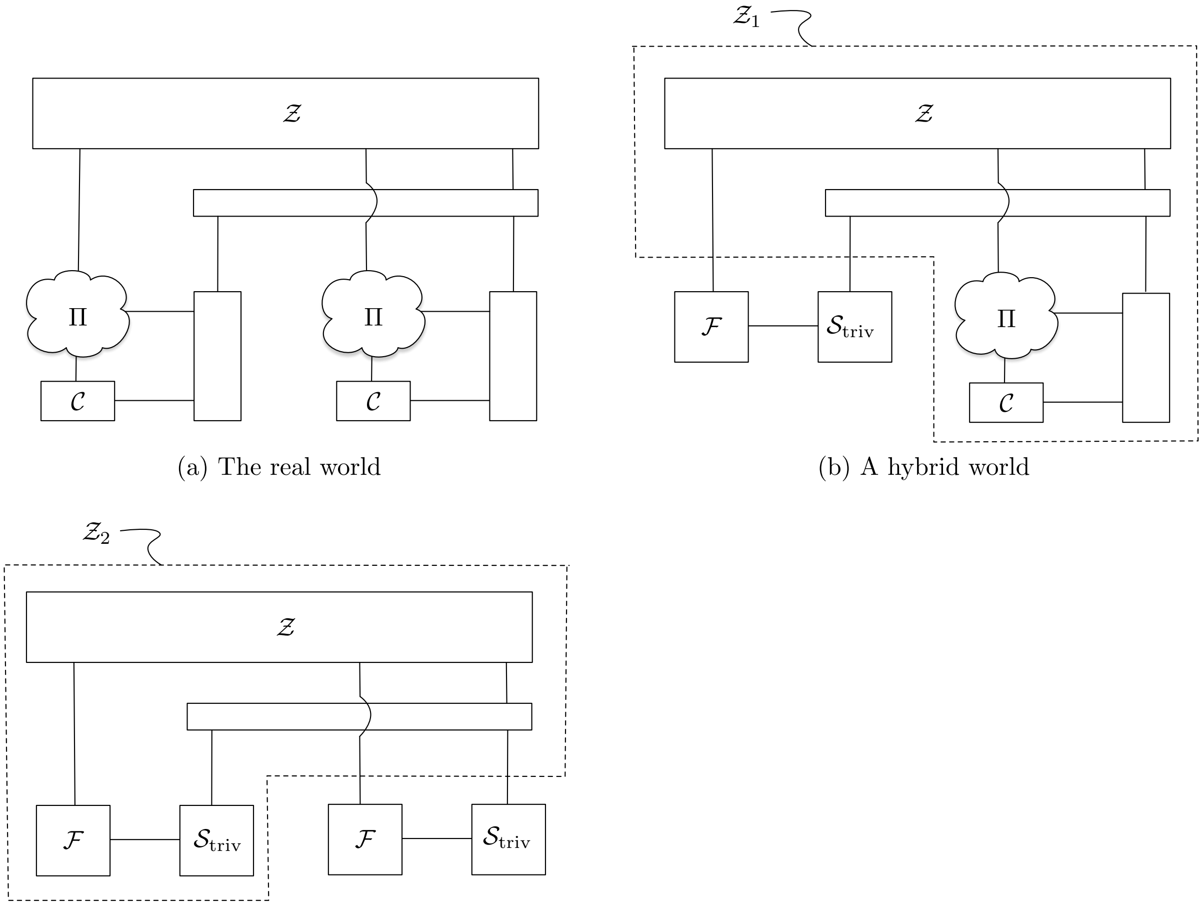
Again by virtue of single-instance security, we can replace the second real-world protocol instance by a corresponding ideal world protocol instance. This is illustrated in Fig. 23.14(c). To formally justify this step, we define a single-instance environment $\mathcal{Z}_2$ that absorbs the first protocol instance (as illustrated by the dashed box in the figure). Single-instance security implies that $\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}_2] \approx \text{Exec}[\mathcal{F}, \mathcal{S}_{\text{triv}}, \mathcal{Z}_2]$, which implies that the multi-instance environment $\mathcal{Z}$ cannot distinguish between the hybrid world in Fig. 23.14(b) and the ideal world in Fig. 23.14(c). The multi-instance simulator in the ideal world consists of a simple routing machine plus two copies of the single-instance simulator $\mathcal{S}_{\text{triv}}$. □

23.5.5.2 Subprotocol composition

Another consequence of secure implementation is that we can use a secure protocol as a subprotocol in a larger protocol, and the larger protocol will remain secure.

To describe what this means takes some work, and we will just sketch the idea.

Suppose we design a protocol Π and we want to prove that Π securely implements some ideal functionality $\mathcal{F}$. Moreover, suppose that Π uses another protocol Π^* as a subprotocol, and suppose that Π^* securely implements some ideal functionality $\mathcal{F}^*$.

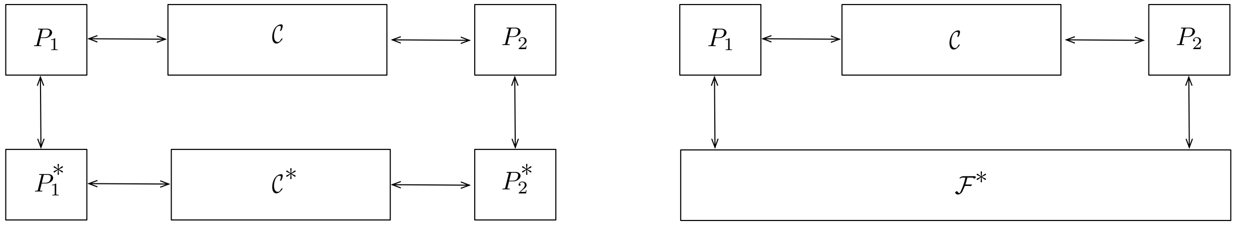


(a) The real world

(b) A hybrid world

(c) The ideal world

Figure 23.14: Diagram for proof of Theorem 23.4



(a) Protocol Π

(b) Protocol $\Pi_{\mathcal{F}^*}$

Figure 23.15: A protocol Π and a hybrid protocol $\Pi_{\mathcal{F}^*}$

To carry out the proof, we can proceed in a modular fashion. We first consider the **hybrid protocol** $\Pi_{\mathcal{F}^*}$ that uses the ideal functionality $\mathcal{F}^*$ as a subprotocol in place of Π^*. Typically, $\Pi_{\mathcal{F}^*}$ is much simpler and easier to analyze than Π itself, as we have stripped out all of the implementation details of Π^*. Now suppose we prove that the hybrid protocol $\Pi_{\mathcal{F}^*}$ securely implements $\mathcal{F}$. Then it turns out that this implies that Π securely implements $\mathcal{F}$.

See Fig. 23.15 for an illustration. Fig. 23.15(a) shows a protocol Π that uses a protocol Π^* as a subprotocol. Here, we are assuming there are just two parties, and this figure does not show the adversary or the environment. The machines labeled P_1 and P_2, which implement the "top part" of Π (i.e., the part that does not include the subprotocol Π^*), are peers communicating through a secure communication network $\mathcal{C}$. The machines labeled P_1^* and P_2^*, which implement Π^*, are peers communicating through a separate communication network $\mathcal{C}^*$. Both machines P_1 and P_1^* belong to the first party. It is best to think of them as two different programs executing on the same computer, so P_1 may supply inputs directly to P_1^* and obtain outputs directly from P_1^*. Similarly, both machines P_2 and P_2^* belong to the second party.

Fig. 23.15(b) shows the hybrid protocol $\Pi_{\mathcal{F}^*}$. It is called a hybrid protocol as it has elements of both the real world (P_1, P_2, and $\mathcal{C}$) and an ideal world (the ideal functionality $\mathcal{F}^*$). In this hybrid protocol, instead of P_1 supplying an input to P_1^*, that input is supplied directly to $\mathcal{F}^*$. Similarly, instead of P_1 obtaining an output from P_1^*, that output comes directly from $\mathcal{F}^*$.

The formal model for protocol execution of such a hybrid protocol is essentially the same as that of a real-world protocol, except that now there is both a communication network $\mathcal{C}$ and an ideal functionality $\mathcal{F}^*$. It should be straightforward for the reader to fill in these details.

We state the following theorem:

Theorem 23.5 (subprotocol composition). *Suppose the following:*

- Π^* *is an efficient protocol that securely implements an ideal functionality* $\mathcal{F}^*$*,*
- Π *is an efficient protocol that uses* Π^* *as a subprotocol, and*
- *the hybrid protocol* $\Pi_{\mathcal{F}^*}$ *(in which the ideal functionality* $\mathcal{F}^*$ *replaces the subprotocol* Π^**) securely implements* $\mathcal{F}$*.*

Then Π *securely implements* $\mathcal{F}$*.*

Proof sketch. We are assuming that Π^* securely implements $\mathcal{F}^*$, which means that there is a simulator $\mathcal{S}^*_{\text{triv}}$ (that is compatible with $\mathcal{F}^*$) such that for every well-behaved environment $\mathcal{Z}$, we have $\text{Exec}[\Pi^*, \mathcal{A}^*_{\text{triv}}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}^*, \mathcal{S}^*_{\text{triv}}, \mathcal{Z}]$. Here, $\mathcal{A}^*_{\text{triv}}$ is the trivial adversary (which interacts with Π^*).

We are also assuming that $\Pi_{\mathcal{F}^*}$ securely implements $\mathcal{F}$, which means that for every adversary $\mathcal{A}$ (that is compatible with $\Pi_{\mathcal{F}^*}$), there is a simulator $\mathcal{S}$ (that is compatible with $\mathcal{F}$), such that for every well-behaved environment $\mathcal{Z}$, we have $\text{Exec}[\Pi_{\mathcal{F}^*}, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}]$.

The proof is essentially a hybrid argument. By a generalization of Theorem 23.3, we may assume the adversary in the real world execution of protocol Π is the trivial adversary that simply routes messages between the environment and other protocol machines. This is illustrated in Fig. 23.16(a). Here, the cloud labeled Π^* represents the honest protocol machines running the subprotocol Π^*, while the cloud labeled Π_{top} represents the honest protocol machines running the "top part" of Π (i.e., the part that does not include the subprotocol Π^*). The unlabeled box represents a simple routing machine, and the box labeled $\mathcal{A}^*_{\text{triv}}$ is a the trivial adversary that interacts with the subprotocol Π^* — the unlabeled box routes messages from $\mathcal{Z}$ to either Π_{top} or $\mathcal{A}^*_{\text{triv}}$, based on instance IDs that are assumed to be embedded in messages.

By virtue of the fact that Π^* securely implements $\mathcal{F}^*$, we can replace Π^* by $\mathcal{F}^*$, as illustrated in Fig. 23.16(b). To formally justify this step, we define an environment $\mathcal{Z}_{\text{top}}$ that absorbs Π_{top} (as illustrated by the dashed box labeled $\mathcal{Z}_{\text{top}}$ in the figure). The fact that Π^* securely implements $\mathcal{F}^*$ implies that $\text{Exec}[\Pi^*, \mathcal{A}^*_{\text{triv}}, \mathcal{Z}_{\text{top}}] \approx \text{Exec}[\mathcal{F}^*, \mathcal{S}^*_{\text{triv}}, \mathcal{Z}_{\text{top}}]$, which implies that the environment $\mathcal{Z}$ cannot distinguish between the real world in Fig. 23.16(a) and the hybrid world in Fig. 23.16(b).

By virtue of the fact that $\Pi_{\mathcal{F}^*}$ securely implements $\mathcal{F}$, we can replace $\Pi_{\mathcal{F}^*}$ by $\mathcal{F}$, as illustrated in Fig. 23.16(c). To formally justify this step, we define an adversary $\mathcal{A}$ that comprises both the router and $\mathcal{S}^*_{\text{triv}}$ (as illustrated by the dashed box labeled $\mathcal{A}$ in the figure). The fact that $\Pi_{\mathcal{F}^*}$ securely implements $\mathcal{F}$ implies that $\text{Exec}[\Pi_{\mathcal{F}^*}, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}, \mathcal{S}, \mathcal{Z}]$, which implies that environment $\mathcal{Z}$ cannot distinguish between the hybrid world in Fig. 23.16(b) and the ideal world in Fig. 23.16(c). □

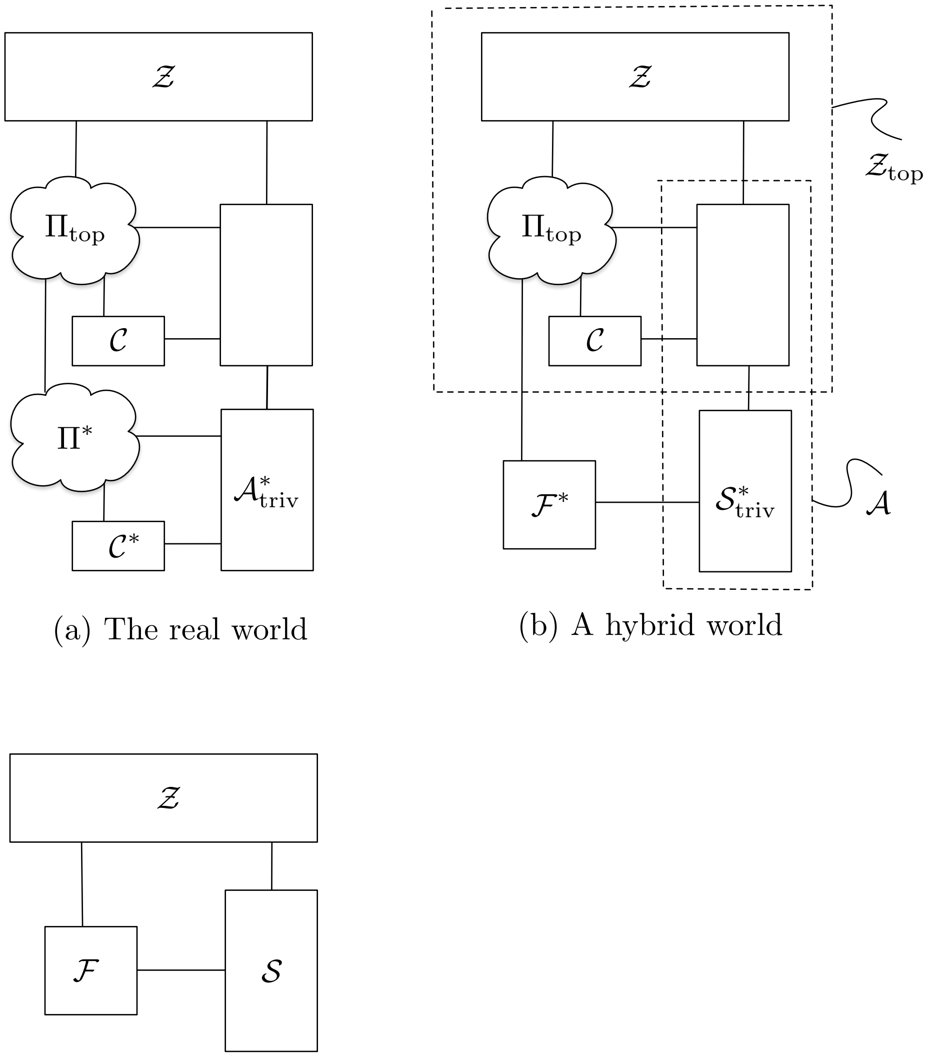
We shall illustrate the use of Theorem 23.5 below in Section 23.5.10.

The subprotocol composition theorem holds in a more general sense. Specifically, a single instance of protocol Π may invoke many instances of subprotocol Π^*, which may themselves run concurrently with another. Similarly, we can replace each of those instances of subprotocol Π^* by an instance of the ideal functionality $\mathcal{F}^*$ to obtain a corresponding hybrid protocol $\Pi_{\mathcal{F}^*}$. Theorem 23.5 holds in this more general sense as well.

Remark 23.4 (On the utility of a simulator that works in all environments). The techniques used in the proofs of Theorems 23.4 and 23.5 highlight the usefulness of separating the adversary and the environment in defining protocol security to say that there must be a simulator that works for all environments. In the proofs of these composition theorems, this allows us to construct environments that represent some elements of the system, while replacing other elements of the system by idealized elements. □

23.5.5.3 Hybrid protocols and transitivity of secure implementation

We introduced in Section 23.5.5.2 the notion of a *hybrid protocol*, which consists of a combination of ordinary protocol machines (which execute the code of the honest parties) and ideal functionalities. A real protocol is a special kind of hybrid protocol, which only makes use of the ideal functionality $\mathcal{C}$ representing a secure communication network. Similarly, an ideal protocol is also a special kind



(a) The real world

(b) A hybrid world

(c) The ideal world

Figure 23.16: Diagram for proof of Theorem 23.5

of hybrid protocol, in which the protocol machines simply pass their inputs and outputs directly between the environment and the ideal functionality. Thus, all protocols, including those in the real and ideal worlds, can be viewed as hybrid protocols.

The notion of *securely implements* can be extended to arbitrary hybrid protocols. For hybrid protocols Π and Π', we say Π **securely implements** Π' if for every adversary $\mathcal{A}$ (compatible with Π) there exists an adversary $\mathcal{A}'$ (compatible with Π', and which may depend on $\mathcal{A}$), such that for every well-behaved environment $\mathcal{Z}$, we have $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\Pi', \mathcal{A}', \mathcal{Z}]$. Note that in this context, there is no advantage in making a distinction in terminology and notation between adversaries and simulators.

We note that Theorem 23.3 on the completeness of the trivial adversary, Theorem 23.4 on concurrent composition, as well as Theorem 23.5 on subprotocol composition easily generalize to arbitrary hybrid protocols. We can also prove a convenient *transitivity property*:

Theorem 23.6 (Transitivity of secure implementation). *Let Π, Π', Π'' be efficient hybrid protocols. If Π securely implements Π' and Π' securely implements Π'', then Π securely implements Π''.*

Proof sketch. The assumption that Π securely implements Π' means that for every adversary $\mathcal{A}$ (compatible with Π) there exists an adversary $\mathcal{A}'$ (compatible with Π'), such that $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\Pi', \mathcal{A}', \mathcal{Z}]$. The assumption that Π' securely implements Π'' means that for every adversary $\mathcal{A}'$ (compatible with Π') there exists an adversary $\mathcal{A}''$ (compatible with Π''), such that $\text{Exec}[\Pi, \mathcal{A}', \mathcal{Z}] \approx \text{Exec}[\Pi', \mathcal{A}'', \mathcal{Z}]$. It immediately follows that for every adversary $\mathcal{A}$ (compatible with Π) there exists an adversary $\mathcal{A}''$ (compatible with Π''), such that $\text{Exec}[\Pi, \mathcal{A}, \mathcal{Z}] \approx \text{Exec}[\Pi'', \mathcal{A}'', \mathcal{Z}]$. □

23.5.6 Defining honest-but-curious security

The most natural way to define honest-but-curious security requires us to make some modifications to our framework for protocol execution. We call this modified framework the **restricted framework**. Unlike our original unrestricted framework, in this restricted framework, adversaries in the real world (and simulators in the ideal world) are inherently limited in what they are allowed to do. Instead of using this restricted framework, it is possible to equivalently define a restricted notion of adversary (or simulator) in the original, *unrestricted* framework, but we shall not do this here.

23.5.6.1 Defining the restricted framework

Restricted framework: the real world. We begin by describing the restricted framework for the real world. Here, the corrupt parties are *not* absorbed into the adversary as in the unrestricted framework. Rather, they are represented by machines separate and apart from the adversary. Moreover, just like an honest party, a corrupt party receives its input directly from the environment and delivers its outputs directly to the environment. The corrupt parties run an "honest but leaky" version of the protocol, which means that they follow the protocol exactly, *except* that they *report* to the adversary

- all messages that they receive from the communication network, and
- all random bit strings that they generate (we assume that all random objects are ultimately derived from these random bit strings).

These **reports** are special messages that have some special rules associated with them. Specifically, upon receiving a *report*, a *valid adversary* must either

- return control immediately to the reporting machine, or
- send a sequence of one or more *reports* to the environment, and thereafter return control to the reporting machine.

Moreover, a *valid environment* must obey the following rule: upon receiving a *report* from the adversary, it must either

- return control immediately to the adversary, or
- halt.

We will always restrict ourselves to valid adversaries and environments that adhere to the above rules.

The above rules essentially mean that while the corrupt parties follow the protocol, the adversary (and, indirectly, the environment) is allowed to continuously observe the evolution of their internal states. However, the rules pertaining to *reports* ensure that the act of observing does not interrupt the normal flow of control of the protocol.

Note that the behavior of the communication network $\mathcal{C}$ is exactly the same as in our unrestricted framework, except that now, the adversary can only communicate with $\mathcal{C}$ over the $\mathcal{C}$-control wire, and *not* over the $\mathcal{C}$-I/O wires: only the honest and corrupt parties may communicate with $\mathcal{C}$ over the $\mathcal{C}$-I/O wires. In particular, the adversary can still decide if and when messages are delivered to any party (honest or corrupt); however, it cannot learn or alter the content of a message sent between two honest parties, nor can it alter the content of any message sent from a corrupt party to any other party. Of course, because of the "leakiness" of the corrupt parties, the adversary will learn the content of messages sent to corrupt parties, and will (in principle) also know the content of messages sent by corrupt parties.

Just as in the unrestricted framework, one can define the trivial adversary $\mathcal{A}_{\text{triv}}$, which essentially just does the bidding of the environment. A difference to keep in mind, however, is that in the restricted framework, in addition to the trivial adversary $\mathcal{A}_{\text{triv}}$, we also have protocol machines that execute the protocol on behalf of the corrupt parties. These protocol machines will report information to $\mathcal{A}_{\text{triv}}$, which will relay these reports to the environment.

Restricted framework: the ideal world. In the restricted framework for the ideal world, all of the $\mathcal{F}$-I/O wires connect the environment $\mathcal{Z}$ directly to the ideal functionality $\mathcal{F}$. This means that the simulator $\mathcal{S}$ cannot alter the inputs to or outputs from the corrupt parties. We also allow $\mathcal{S}$ to deliver reports to $\mathcal{Z}$.

With these restrictions, not only is it impossible for $\mathcal{S}$ to alter inputs to or outputs from corrupt parties, $\mathcal{S}$ cannot even *see* these values. This will unfortnately make it quite difficult, if not impossible, to prove the security of just about any protocol. To rectify this situation, we have to provide $\mathcal{S}$ with some mechanism for seeing the inputs to or outputs from the corrupt parties. One could build such a mechanism into the framework, but it is easier to just assume that $\mathcal{F}$ itself provides such a mechanism. For example, when working with the $\mathcal{F}_{\text{sfe}}$ functionality for secure function evaluation (as in Section 23.5.3), we can simply augment $\mathcal{F}_{\text{sfe}}$ with an interface that allows $\mathcal{S}$ to query the inputs to or outputs from corrupt parties (provided these inputs or outputs are available).

23.5.6.2 Security definitions and consequences

Definitions 23.5 and 23.5 carry over directly to the restricted framework. For the analog of of Definition 23.5, we say that Π **semi-securely implements $\mathcal{F}$ against $\mathcal{A}$**, while for the analog of Definition 23.6, we say that or Π **semi-securely implements $\mathcal{F}$**.

We note that Theorem 23.3 on the completeness of the trivial adversary, Theorem 23.4 on concurrent composition, Theorem 23.5 on subprotocol composition, as well as Theorem 23.6 on the transitivity property, also carry over directly to the restricted framework, essentially just replacing "securely implements" by "semi-securely implements" throughout.

23.5.7 A warmup honest-but-curious security proof: a simple OT protocol

Recall that in a 1-out-of-2 oblivious transfer (OT) protocol, one party, called the sender and denoted P_{s}, has two inputs m_0, m_1, while the other party, called the receiver and denoted P_{r}, has an input $\sigma \in \{0, 1\}$. Roughly speaking, at the end of the protocol, P_{r} should obtain m_σ, while P_{s} obtains nothing.

The ideal functionality for OT is just a special case of $\mathcal{F}_{\mathrm{sfe}}$, where

- the function computed is $f(m_0, m_1, \sigma) := m_\sigma$,
- the inputs m_0, m_1 are supplied by P_{s},
- the input σ is supplied by P_{r}, and
- the output m_σ is only obtained by P_{r}.

We shall denote by $\mathcal{F}_{\mathrm{ot}}$ this specific ideal functionality. As discussed above in Section 23.5.6, in the honest-but-curious setting, we assume $\mathcal{F}_{\mathrm{ot}}$ is augmented with an interface that allows the simulator in the ideal world to retrieve inputs to and outputs from corrupt parties.

We shall present a simple protocol for 1-out-of-2 OT, and give a fairly complete proof that it is secure in the honest-but-curious security model. The protocol is a variant of the protocol in Section 11.6. Instead of being based on the hashed ElGamal scheme, it is based on the multiplicative ElGamal scheme discussed in Exercise 11.5, and its security is based on the DDH assumption.

Let $\mathbb{G}$ be a group of prime order q generated by $g \in \mathbb{G}$. We will require that P_{s}'s inputs m_0, m_1 are (or can be encoded as) elements of $\mathbb{G}$.

The OT protocol Π is presented in Fig. 23.17, and in more detail, it runs as follows:

- *Inputs:*
  - P_{s} takes as input $m_0, m_1 \in \mathbb{G}$;
  - P_{r} takes as input $\sigma \in \{0, 1\}$.
- *P_{s}'s first move:*
  - P_{s} computes $d \xleftarrow{\mathrm{R}} \mathbb{G}$ and sends d to P_{r}.
- *P_{r}'s move:*

P_{s} : input $(m_0, m_1) \in \mathbb{G} \times \mathbb{G}$		P_{r} : input $\sigma \in \{0, 1\}$
$d \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathbb{G}$	$\xrightarrow{\quad d \quad}$	$\alpha \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathbb{Z}_q$ $u_\sigma \leftarrow g^\alpha$, $u_{1-\sigma} \leftarrow d/g^\alpha$
$\beta \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathbb{Z}_q$, $v \leftarrow g^\beta$ $c_0 \leftarrow u_0^\beta m_0$, $c_1 \leftarrow u_1^\beta m_1$	$\xleftarrow{\quad (u_0, u_1) \quad}$	
	$\xrightarrow{\quad (v, c_0, c_1) \quad}$	output c_σ / v^α

Figure 23.17: A simple DDH-based OT protocol

- After receiving d from P_{s}, party P_{r} computes

$$\alpha \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathbb{Z}_q, \quad u_\sigma \leftarrow g^\alpha, \quad u_{1-\sigma} \leftarrow d/g^\alpha,$$

and sends (u_0, u_1) to P_{s}.

- *P_{s}'s second move:*
  - After receiving (u_0, u_1) from P_{r}, party P_{s} computes

$$\beta \overset{\scriptscriptstyle\mathrm{R}}{\leftarrow} \mathbb{Z}_q, \quad v \leftarrow g^\beta, \quad c_0 \leftarrow u_0^\beta m_0, \quad c_1 \leftarrow u_1^\beta m_1$$

and sends (v, c_0, c_1) to P_{r}.

- *P_{r}'s output stage:*
  - After receiving (v, c_0, c_1) from P_{s}, party P_{r} outputs c_σ / v^α.

Assuming both parties follow the protocol, we see that

$$c_\sigma = v^\alpha \cdot m_\sigma \quad \text{and} \quad c_{1-\sigma} = d^\beta / v^\alpha \cdot m_{1-\sigma}.$$

So it is clear that P_{r} outputs m_σ.

23.5.7.1 Security: intuition

We first give some intuition as to why the protocol is secure in the honest-but-curious model, assuming that the encryption scheme is semantically secure.

We first want to argue intuitively that P_{s} learns nothing about P_{r}'s input σ. Observe that (u_0, u_1) is a pair of random group elements that multiply out to d. The distribution of (u_0, u_1) is the same regardless of whether $\sigma = 0$ or $\sigma = 1$. This implies that P_{s} learns nothing about P_{r}'s input σ.

We next want to argue intuitively that while P_{r} learns P_{s}'s input m_σ, it does not learn anything about P_{s}'s other input $m_{1-\sigma}$. For this, we will need the DDH assumption for $\mathbb{G}$. Now, P_{r} knows its own input σ, the random value α (which it generated itself), as well as its output m_σ (which it is supposed to learn). In addition, P_r receives the values d, v, c_0, c_1 from P_s. The values d and v are just random group elements, so by themselves, they reveal nothing about $m_{1-\sigma}$. We also know that $c_\sigma = v^\alpha m_\sigma$, so the value of c_σ is completely determined by values P_r already knows, and so it also does not reveal anything about $m_{1-\sigma}$. Finally, we know that

$$c_{1-\sigma} = d^\beta / v^\alpha \cdot m_{1-\sigma}.$$

Under the DDH assumption, the triple (d, g^β, d^β) is computationally indistinguishable from the triple (d, g^β, e), where $e \in \mathbb{G}$ is a random group element. Thus, $c_{1-\sigma}$ is itself computationally indistinguishable from a random group element, and hence it also does not reveal anything about $m_{1-\sigma}$.

23.5.7.2 Security: a formal proof

Now we want to give a more formal proof of security in the honest-but-curious setting. Specifically, we want to prove the following:

Theorem 23.7. *The above OT protocol* Π semi-securely *implements the ideal functionality* $\mathcal{F}_\mathrm{ot}$*, under the DDH assumption for* $\mathbb{G}$.

By the completeness of the trivial adversary, it suffices to build a simulator $\mathcal{S}$ such that in the restricted framework of Section 23.5.6, for every well-behaved environment $\mathcal{Z}$, we have

$$\mathrm{Exec}[\Pi, \mathcal{A}_\mathrm{triv}, \mathcal{Z}] \approx \mathrm{Exec}[\mathcal{F}_\mathrm{ot}, \mathcal{S}, \mathcal{Z}]. \tag{23.16}$$

Here, $\mathcal{A}_\mathrm{triv}$ is the trivial adversary.

Review of the real world. Let us review the most important details of the real-world execution. The environment $\mathcal{Z}$ supplies inputs to and receives outputs from all parties, after initially deciding which parties are corrupt. (In this particular protocol, only P_r generates an output.) The communication network is completely controlled by $\mathcal{Z}$ (indirectly, via instructions it sends over the $\mathcal{C}$-control wire via $\mathcal{A}_\mathrm{triv}$), and so it decides if and when protocol messages get delivered. As the protocol executes, any corrupt party immediately reports to $\mathcal{Z}$ (again, indirectly via $\mathcal{A}_\mathrm{triv}$) the details of any message it receives over the communication network or any random string that it generates. Other than generating these reports, a corrupt party faithfully executes the protocol. Recall that after receiving such a report, $\mathcal{Z}$ immediately returns control to the reporting machine (again, indirectly via $\mathcal{A}_\mathrm{triv}$).

Review of the ideal world. Now let us review the details of the ideal-world execution. There is a simulator $\mathcal{S}$. As in the real world, $\mathcal{Z}$ supplies inputs to and receives outputs from all parties, after initially deciding which parties are corrupt. Each party's input is forwarded directly to the ideal functionality. Once all inputs are provided to the ideal functionality, $\mathcal{S}$ can choose if and when to tell the ideal functionality to give any party's output to $\mathcal{Z}$. Moreover, at any point in time, $\mathcal{S}$ may ask the ideal functionality for the input to or output from a corrupt party, even though $\mathcal{S}$ cannot modify these values.

High-level proof strategy. So to prove security of protocol Π, we need to design $\mathcal{S}$ in such a way that (23.16) holds for all well-behaved $\mathcal{Z}$. Intuitively, this means that $\mathcal{S}$, running in the ideal world, must interact with $\mathcal{Z}$ in such a way that essentially makes $\mathcal{Z}$ "think" it is still in the real world. In particular, whenever a real-world corrupt party would normally receive a protocol message from a real-world honest party, it would normally be reported to $\mathcal{Z}$, and so $\mathcal{S}$ must somehow *cook up* a simulated protocol message, and report this simulated protocol message to the environment instead. Similarly, whenever a real-world corrupt party would normally generate a random string, $\mathcal{S}$ must also cook up a corresponding simulated random string. $\mathcal{S}$ must do all of this knowing only the corrupt parties' inputs and outputs, but without knowing the honest parties' inputs or outputs. Moreover, all of this must be done so that $\mathcal{Z}$, who knows *all* parties' inputs and outputs, cannot effectively distinguish the ideal world execution from the real world execution.

Further assumptions about randomness. To formally prove security in the honest-but-curious setting, the protocol itself must be very explicitly specified in terms of how random objects are generated by all parties. This is because a corrupt party must follow the protocol exactly, including when and how it generates random objects, and so this must actually be specified precisely. In fact, we require that the only random objects generated are random bit strings.

For this protocol, we assume that $coins_{\mathrm{s}}$ is the random string that real-world P_{s} uses in the derivation of $d \in \mathbb{G}$ and $\beta \in \mathbb{Z}_q$. We assume that the distribution of (d, β) is statistically indistinguishable from the uniform distribution on $\mathbb{G} \times \mathbb{Z}_q$ (see Definition 3.6). We also assume that real-world P_{s} generates $coins_{\mathrm{s}}$ as soon as it receives its input.

Suppose that $coins_{\mathrm{r}}$ is the random string that real-world P_{r} uses in the derivation of $\alpha \in \mathbb{Z}_q$. We assume that the distribution of α is statistically indistinguishable from the uniform distribution on $\mathbb{Z}_q$. We also assume that real-world P_{r} generates $coins_{\mathrm{r}}$ as soon as it receives its input.

While we should specify the exact algorithms that are used for deriving (d, β) from $coins_{\mathrm{s}}$ and for deriving α from $coins_{\mathrm{r}}$, this is not necessary for the analysis of this particular protocol.

Note that for some protocols (such as the one we are analyzing here), $\mathcal{S}$ may be able to just generate the simulated random strings just as random strings; for other protocols, it may need to be more clever (for example, see Exercise 23.3).

Remark 23.5 (Generating random numbers mod q). For deriving a nearly random element of $\mathbb{Z}_q$ from a random bit string, a protocol may use the following simple technique. Let k be a parameter chosen so that the value $q/2^k$ is negligible. Then given a random bit string $\rho \in \{0,1\}^k$, we can naturally view ρ as an integer uniformly distributed over $\{0, \ldots, 2^k - 1\}$, and then reduce it mod q to get an element of $\mathbb{Z}_q$ whose statistical distance from the uniform distribution on $\mathbb{Z}_q$ is less than $q/2^k$ (see Example 3.2). □

The simulator $\mathcal{S}$. Our simulator $\mathcal{S}$ will work differently depending on which of the two parties are corrupt. If neither party is corrupt or both parties are corrupt, there is not much for $\mathcal{S}$ to do. So we focus on the case where one of the two parties is corrupt.

Simulation when P_{s} is corrupt. Here is how $\mathcal{S}$ will work assuming that P_{s} is corrupt.

- When $\mathcal{Z}$ sends an input (m_0, m_1) to P_{s}, this is forwarded directly to $\mathcal{F}_{\mathrm{ot}}$, which notifies $\mathcal{S}$ that this input has been received. At this point, $\mathcal{S}$ queries $\mathcal{F}_{\mathrm{ot}}$ to obtain this input. $\mathcal{S}$ then generates the string $coins_{\mathrm{s}}$ at random, and uses $coins_{\mathrm{s}}$ to derive $d \in \mathbb{G}$ and $\beta \in \mathbb{Z}_q$, just like real-world P_{s} would do. $\mathcal{S}$ then reports $coins_{\mathrm{s}}$ to $\mathcal{Z}$ (just like real-world P_{s} would do).

- At some later time, $\mathcal{Z}$ will generate a control message that would cause real-world P_{s} to receive the protocol message (u_0, u_1) from real-world P_{r}. Of course, in the ideal world, $\mathcal{S}$ receives no such protocol message, but has to cook up a simulated protocol message, as discussed above. So it simply computes
$$u_0 \xleftarrow{\mathrm{R}} \mathbb{G}, \quad u_1 \leftarrow d/u_0,$$
and reports the message (u_0, u_1) to $\mathcal{Z}$ (just like real-world P_{s} would do).

That completes our description of the simulator $\mathcal{S}$. There are few insignificant details we have left out, mostly related to how $\mathcal{S}$ processes control messages from $\mathcal{Z}$ related to delivery of protocol messages to real-world P_{r}. The simulation is essentially perfect; in particular, (23.16) holds. This holds without any computational assumption. The main thing to argue (as we already did above in Section 23.5.7.1), is that regardless of the value of σ, the distribution of (u_0, u_1) is just a random pair of group elements that multiply out to d.

Simulation when P_{r} is corrupt. Here is how $\mathcal{S}$ will work assuming that P_{r} is corrupt.

- When $\mathcal{Z}$ sends an input σ to P_{r}, this input is forwarded directly to $\mathcal{F}_{\mathrm{ot}}$, which notifies $\mathcal{S}$ that this input has been received. At this point, $\mathcal{S}$ queries $\mathcal{F}_{\mathrm{ot}}$ to obtain this input. $\mathcal{S}$ then generates the string $coins_{\mathrm{r}}$ at random, and uses $coins_{\mathrm{r}}$ to derive $\alpha \in \mathbb{Z}_q$, just like real-world P_{r} would do. $\mathcal{S}$ then reports $coins_{\mathrm{r}}$ to $\mathcal{Z}$ (just like real-world P_{r} would do).

- At some later time, $\mathcal{Z}$ will generate a control message that would cause real-world P_{r} to receive the protocol message d from real-world P_{s}. Of course, in the ideal world, $\mathcal{S}$ receives no such protocol message, but has to cook up a simulated message, as discussed above. So it simply computes $d \xleftarrow{\mathrm{R}} \mathbb{G}$ and reports the message d to $\mathcal{Z}$ (just like real-world P_{r} would do).

- At some later time, $\mathcal{Z}$ will generate a control message that would cause real-world P_{r} to receive the protocol message (v, c_0, c_1) from real-world P_{s}. Of course, in the ideal world, $\mathcal{S}$ receives no such protocol message, and again, it has to cook up a simulated version of this protocol message. Here is how it does so. At this point in time, the environment must have already sent an input P_{s}, which was forwarded to $\mathcal{F}_{\mathrm{ot}}$. Note that the values (m_0, m_1) are not directly available to $\mathcal{S}$. However, $\mathcal{S}$ may ask for P_{r}'s output m_σ at this point in time, and then compute
$$v \xleftarrow{\mathrm{R}} \mathbb{G}, \quad c_\sigma \leftarrow v^\alpha m_\sigma, \quad c_{1-\sigma} \xleftarrow{\mathrm{R}} \mathbb{G}.$$
$\mathcal{S}$ then reports the simulated protocol message (v, c_0, c_1) to $\mathcal{Z}$ (as real-world P_{r} would do). When control returns to $\mathcal{S}$, who then instructs $\mathcal{F}_{\mathrm{ot}}$ to deliver P_{r}'s output to $\mathcal{Z}$.

That completes the description of the simulator $\mathcal{S}$. There are a few insignificant details we have left out, mostly related to how $\mathcal{S}$ processes control messages from $\mathcal{Z}$ related to delivery of protocol messages to real-world P_{s}. The simulation is not perfect, but one can show (using the argument sketched above in Section 23.5.7.1) that (23.16) holds under the DDH assumption.

23.5.7.3 Further remarks on this protocol

Remark 23.6. While this protocol is secure in the honest-but-curious setting, it is completely insecure in the malicious setting. Indeed, suppose P_{r} is maliciously corrupt. Then P_{r} could simply generate u_0 and u_1 so that it knows the discrete logs of both, thereby allowing it to compute both m_0 and m_1. One could mitigate this attack by modifying the protocol so that P_{s} checks that $u_0 \cdot u_1 = d$, and aborts if this is not the case. While this modification does mitigate against this attack, it does not yield a protocol that we can prove secure in the malicious setting. □

Remark 23.7. In the honest-but-curious setting, we require that corrupt parties follow the protocol exactly, and in particular, that they generate random bit strings as specified by the protocol. It is possible to study a slightly stronger notion of security whereby the corrupt parties may use adversarially chosen bit strings in place of random bit strings, but otherwise follow the protocol. This protocol is secure in this stronger sense. □

23.5.8 A warmup malicious security proof: a simple OT protocol

In Section 23.5.7, we presented a two-party protocol for 1-out-of-2 oblivious transfer (OT), and proved that is was secure in the honest-but-curious setting. In this section, we present a 3-party OT protocol, and prove that it is secure in the *malicious* setting, assuming that at most one out of the three parties is corrupt.

In this protocol, we have a sender, P_{s}, which has two inputs m_0, m_1, a receiver, P_{r}, which has an input $\sigma \in \{0, 1\}$, and a "helper" P_{h}, which has no input. At the end of the protocol, P_{r} should obtain m_σ, while P_{s} and P_{h} obtain nothing.

We shall also assume that the message space $\mathcal{M}$, from which m_0, m_1 are drawn, is the set of all bit strings of some fixed length.

The ideal functionality for such a protocol is essentially the same as $\mathcal{F}_{\mathrm{ot}}$ presented in Section 23.5.7, except for the additional "helper" party P_{h}, which contributes no input and obtains no output.

As in Section 23.3.7, we make use of a collision-resistant hash function

$$H : \mathcal{M} \times \mathcal{R} \to \mathcal{C},$$

which is used as a non-interactive commitment scheme, as discussed in Section 8.12. Here, $\mathcal{R}$ is some suitably large space of randomizing elements, and $\mathcal{C}$ is some finite output space (the "commitment" space). Again, the randomizing elements will be used to hide input messages, meaning that for adversarially chosen $m_0, m_1 \in \mathcal{M}$, and for randomly chosen $r \in \mathcal{R}$,

$$H(m_0, r) \quad \text{and} \quad H(m_1, r)$$

are computationally indistinguishable.

The OT protocol Π is presented in Fig. 23.18, and in more detail, it runs as follows:

- *Inputs:*
  - P_{s} takes as input $m_0, m_1 \in \mathcal{M}$;
  - P_{r} takes as input $\sigma \in \{0, 1\}$.
- P_{h}*'s move:*

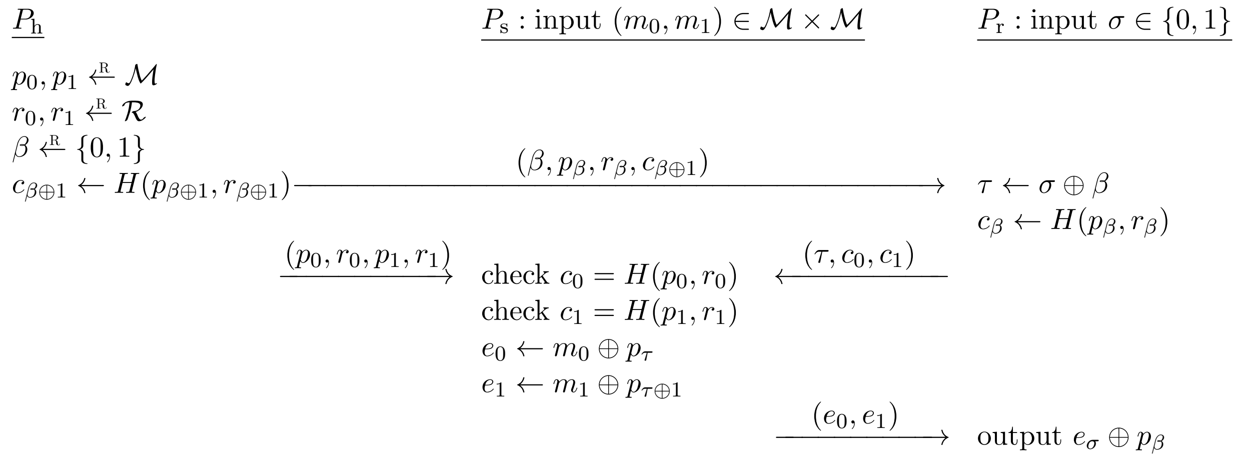

P_{h}
P_{s} : input $(m_0, m_1) \in \mathcal{M} \times \mathcal{M}$
P_{r} : input $\sigma \in \{0,1\}$
$p_0, p_1 \xleftarrow{\mathrm{R}} \mathcal{M}$
$r_0, r_1 \xleftarrow{\mathrm{R}} \mathcal{R}$
$\beta \xleftarrow{\mathrm{R}} \{0,1\}$
$c_{\beta\oplus 1} \leftarrow H(p_{\beta\oplus 1}, r_{\beta\oplus 1})$
$(\beta, p_\beta, r_\beta, c_{\beta\oplus 1})$
$\tau \leftarrow \sigma \oplus \beta$
$c_\beta \leftarrow H(p_\beta, r_\beta)$
(p_0, r_0, p_1, r_1)
check $c_0 = H(p_0, r_0)$
(τ, c_0, c_1)
check $c_1 = H(p_1, r_1)$
$e_0 \leftarrow m_0 \oplus p_\tau$
$e_1 \leftarrow m_1 \oplus p_{\tau\oplus 1}$
(e_0, e_1)
output $e_\sigma \oplus p_\beta$

Figure 23.18: A 3-party OT protocol

- P_{h} computes

$$p_0, p_1 \xleftarrow{\mathrm{R}} \mathcal{M} \quad r_0, r_1 \xleftarrow{\mathrm{R}} \mathcal{R}, \quad \beta \xleftarrow{\mathrm{R}} \{0,1\}, \quad c_{\beta\oplus 1} \leftarrow H(p_{\beta\oplus 1}, r_{\beta\oplus 1}),$$

  and sends

  * $(\beta, p_\beta, r_\beta, c_{\beta\oplus 1})$ to P_{r}, and
  * (p_0, r_0, p_1, r_1) to P_{s}.

- P_{r}'s move:

  - After receiving $(\beta, p_\beta, r_\beta, c_{\beta\oplus 1})$ from P_{h}, party P_{r} computes

$$\tau \leftarrow \sigma \oplus \beta, \quad c_\beta \leftarrow H(p_\beta, r_\beta)$$

    and sends (τ, c_0, c_1) to P_{s}.

- P_{s}'s move:

  - After receiving (p_0, r_0, p_1, r_1) from P_{h} and (τ, c_0, c_1) from P_{r}, party P_{s} first checks that

$$c_0 = H(p_0, r_0) \quad \text{and} \quad c_1 = H(p_1, r_1).$$

  - If this check fails, P_{s} aborts (and may also notify P_{r} that it should abort as well).
  - Otherwise, P_{s} computes

$$e_0 \leftarrow m_0 \oplus p_\tau, \quad e_1 \leftarrow m_1 \oplus p_{\tau\oplus 1},$$

    and sends (e_0, e_1) to P_{r}.

- P_{r}'s output stage:
  - After receiving (e_0, e_1) from P_{s}, party P_{r} outputs $e_\sigma \oplus p_\beta$.

Assuming that all parties follow the protocol, one can easily verify that

$$e_\sigma = m_\sigma \oplus p_\beta \quad \text{and} \quad e_{\sigma\oplus 1} = m_{\sigma\oplus 1} \oplus p_{\beta\oplus 1}. \tag{23.17}$$

An intuitive description. Roughly speaking, the protocol is doing the following:

- The helper generates commitments to two random pads, p_0, p_1, and gives the sender openings to both, while giving the receiver an opening to p_β, where β is a random bit, along with β and just the commitment to the other pad.
- Next, the receiver uses β as a one-time pad to encrypt its selection bit σ, which yields τ, and gives τ along with commitments to both p_0 and p_1 to the sender.
- The sender first checks that the openings it obtained from the helper correspond to the commitments it obtained from the sender. This is designed to catch a cheating helper who attempts to give inconsistent information to the sender and receiver. If this check does not pass, the sender aborts the protocol. Otherwise, it encrypts m_0 using the pad p_τ, which yields e_0, and m_1 using the other pad, $p_{\tau\oplus 1}$, which yields e_1, and gives both encryptions, e_0 and e_1, to the receiver.
- Finally, the receiver decrypts e_σ using the pad p_β, which, by (23.17), yields m_σ.

23.5.8.1 Security: intuition

Before diving into a more formal security proof, we first present some intuition.

The security analysis is done assuming at most one of the three parties is corrupt. We want to argue, intuitively, that the protocol provides *privacy*, *soundness*, and *input independence* (see Section 23.1.1).

Helper corrupt: By the collision resistance property of H, if the corrupt helper sends inconsistent information to the sender and receiver, the protocol will abort. Otherwise, if the corrupt helper sends consistent information, the receiver will output the correct value, m_σ, and the corrupt helper will not learn anything at all about m_0, m_1, or σ. This follows from the assumption that the communication channels between the sender and the receiver are secure, and therefore the corrupt helper cannot modify or learn anything about the protocol messages exchanged between the sender and receiver. Thus, the *privacy* and *soundness* properties are satisfied. Since the corrupt helper has no input, the *input independence* property is trivially satisfied.

Receiver corrupt: When the corrupt receiver gives the value τ to the honest sender, its selection bit is effectively determined as $\sigma := \tau \oplus \beta$, where β is the bit chosen by the honest dealer. Since at this point in time, the corrupt receiver has received nothing that depends in any way on the honest sender's inputs m_0, m_1, the *input independence* property is clearly satisfied. Now consider what happens when the corrupt receiver obtains (e_0, e_1) from the honest sender. By (23.17), we see that e_σ reveals m_σ to the corrupt receiver. Also, by the hiding property of the hash function, the commitment $c_{\beta\oplus 1}$, which the corrupt receiver obtained from the honest helper, reveals no information about $p_{\beta\oplus 1}$ to the corrupt receiver. Thus, by (23.17), we see that $e_{\sigma\oplus 1}$ reveals nothing about $m_{\sigma\oplus 1}$ to the corrupt receiver. Thus, the *privacy* property is satisfied. Finally, since the honest helper and sender have no outputs, the *soundness* property is trivially satisfied.

Sender corrupt: When the corrupt sender gives (e_0, e_1) to the honest receiver, its inputs m_0, m_1 are effectively determined as $m_0 := e_0 \oplus p_\tau$ and $m_1 := e_1 \oplus p_{\tau\oplus 1}$. At this point in time, the corrupt sender has no information about the honest sender's selection bit σ, which is perfectly masked by β, and so the *input independence* property is satisfied. Indeed, the corrupt sender never learns anything more about σ, and so the *privacy* property is also satisfied. By (23.17), we see that the honest receiver decrypts e_σ as m_σ, and hence the *soundness* property is satisfied.

23.5.8.2 Security: a formal proof

Now we want to give a more formal proof of security in the malicious setting. Specifically, we want to prove the following:

Theorem 23.8. *The above OT protocol* Π securely *implements the ideal functionality* $\mathcal{F}_{\text{ot}}$*, assuming* H *is collision resistant and satisfies the hiding property described above, and that at most one party is corrupt.*

By the completeness of the trivial adversary, it suffies to build a simulator $\mathcal{S}$ such that for every well-behaved environment $\mathcal{Z}$, we have

$$\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}_{\text{ot}}, \mathcal{S}, \mathcal{Z}]. \tag{23.18}$$

Here, $\mathcal{A}_{\text{triv}}$ is the trivial adversary.

Review of the real world. Let us review the most important details of the real-world execution. The environment $\mathcal{Z}$ supplies inputs to and receives outputs from the honest parties, after initially deciding which parties are corrupt. (In this particular protocol, only P_{s} and P_{r} receive an input and only P_{r} generates an output.) The communication network is completely controlled by $\mathcal{Z}$ (indirectly, via instructions it sends over the $\mathcal{C}$-control wire via $\mathcal{A}_{\text{triv}}$), and so it decides if and when protocol messages get delivered to any honest parties. Recall that corrupt parties really do not exist; rather, any message that a corrupt party would normally send to an honest party is actually sent by $\mathcal{Z}$ (again, indirectly via $\mathcal{A}_{\text{triv}}$), and any message that an honest party sends to a corrupt party is actually sent to $\mathcal{Z}$ (again, indirectly via $\mathcal{A}_{\text{triv}}$). Nevertheless, for simplicity, we shall talk about "messages sent to an honest party from the corrupt party", and "messages sent from an honest party to the corrupt party".

Review of the ideal world. Now let us review the details of the ideal-world execution. There is a simulator $\mathcal{S}$. As in the real world, $\mathcal{Z}$ supplies inputs to and receives outputs from the honest parties, after initially deciding which parties are corrupt. Each honest party's input is forwarded directly to the ideal functionality. The inputs for any corrupt parties are provided to the ideal functionality by $\mathcal{S}$. Once all inputs are provided to the ideal functionality, $\mathcal{S}$ can choose if and when to tell the ideal functionality generate outputs: an honest party's output is forwarded directly to $\mathcal{Z}$, while a corrupt party's output is given to $\mathcal{S}$.

High-level proof strategy. So to prove security of protocol Π, we need to design $\mathcal{S}$ in such a way that (23.18) holds for all well-behaved $\mathcal{Z}$. Intuitively, this means that $\mathcal{S}$, running in the ideal world, should interact with $\mathcal{Z}$ in such a way that essentially makes $\mathcal{Z}$ "think" it is still in the real world. In particular, $\mathcal{S}$ should exchange messages with $\mathcal{Z}$ (over the $\mathcal{S}$-control wire) that look like messages that would normally be used to control the communication network in the real world (over the $\mathcal{C}$-control wire). Thus, even in the ideal world, we can talk about "messages sent to an honest party from a corrupt party", whose contents are given to $\mathcal{S}$ from $\mathcal{Z}$, and "messages sent from an honest party to a corrupt party", whose contents must be generated by $\mathcal{S}$ and given to $\mathcal{Z}$. There may also be "messages sent between honest parties", whose contents are never given to or generated by $\mathcal{S}$; rather, $\mathcal{S}$ just exchanges control messages with $\mathcal{Z}$ that correspond to the points in time when these messages would have been sent or received in the real world.

Now, whenever a real-world honest party would normally send a protocol message to a real-world corrupt party, $\mathcal{S}$ must somehow *cook up* a simulated protocol message to be sent to that corrupt party (which is given to $\mathcal{Z}$). In addition, based on the messages sent from the real-world corrupt parties to the real-world honest parties, $\mathcal{S}$ must somehow *extract* from those messages an effective input for the corrupt parties, and submit this to the ideal functionality. $\mathcal{S}$ must do all this without knowing any of the honest parties' inputs or outputs, although it will (eventually) learn the corrupt parties' output. All of this must be done so that $\mathcal{Z}$, who also knows all *honest* parties' inputs and outputs, cannot effectively distinguish the ideal world execution from the real world.

Note that one technical distinction (among several) between the malicious setting and the honest-but-curious setting is that in the malicious setting, the simulator must extract effective inputs for corrupt parties, whereas in the honest-but-curious setting, those inputs are simply given to the simulator.

The simulator $\mathcal{S}$. Our simulator $\mathcal{S}$ will work differently depending on which of the three parties is corrupt. If no party is corrupt, there is not much for $\mathcal{S}$ to do. So we focus on the case where exactly one of the three parties is corrupt.

Simulation when P_{h} is corrupt. Here is how $\mathcal{S}$ works when the helper, P_{h}, is corrupt.

- At some point in time, $\mathcal{Z}$ will generate a control message that corresponds to the instant at which real-world P_{s} would have received both a message from real-world P_{h} and a message from real-world P_{r}.

  At this point in time, $\mathcal{S}$ has obtained the message $(\beta, p_\beta, r_\beta, c_{\beta\oplus 1})$ that real-world P_{h} sent to P_{r}, as well as the message (p_0, r_0, p_1, r_1) that real-world P_{h} sent to P_{s}: recall that $\mathcal{S}$ obtains the contents of all messages sent from the corrupt party to the honest parties. $\mathcal{S}$ checks that these two messages are consistent, i.e., that that the values p_β, r_β sent to P_{r} match the corresponding values sent to P_{s}, and that the value $c_{\beta\oplus 1}$ sent to P_{r} is indeed the hash of the values $p_{\beta\oplus 1}, r_{\beta\oplus 1}$ sent to P_{s}.

  If this check fails, $\mathcal{S}$ will continue as if P_{s} aborted. In particular, it will never instruct $\mathcal{F}_{\mathrm{ot}}$ to generate an output for P_{r}.

- If the above check passes, $\mathcal{S}$ will continue as if P_{s} did not abort. At a later point in time, $\mathcal{Z}$ will generate a control message that corresponds to the instant at which real-world P_{r} receives its message from real-world P_{s}. At this point in time, $\mathcal{S}$ instructs $\mathcal{F}_{\mathrm{ot}}$ to generate the output for P_{r}.

That completes the description of the simulator $\mathcal{S}$.

We now show that (23.18) holds under the collision-resistance assumption for the hash function.

To this end, let Game 0 be the real-world execution. Now define Game 1 to be the same as Game 0, except that we make $P_{\rm s}$ abort if $P_{\rm h}$ sends inconsistent information to $P_{\rm s}$ and $P_{\rm h}$. (Of course, $P_{\rm s}$ could not do this in the real world, as it does not have access to all of this information; however, that is not a problem, as this is just a thought experiment.) Observe that Games 0 and 1 proceed identically unless the values (p_β, r_β) received by $P_{\rm s}$ and $P_{\rm h}$ differ, yet hash to the same value. This means that the probability that the environment can distinguish Game 0 from Game 1 is bounded by the probability of finding a collision for the hash function, which is negligible, by the collision-resistance assumption.

Finally, one can verify that Game 1 and the ideal-world execution are completely equivalent from the point of view of the environment.

Simulation when $P_{\rm r}$ is corrupt. Here is how $\mathcal{S}$ works when the receiver, $P_{\rm r}$, is corrupt.

- At some point in time, $\mathcal{Z}$ will generate a control message that corresponds to the instant at which real-world $P_{\rm r}$ receives the message $(\beta, p_\beta, r_\beta, c_{\beta\oplus 1})$ from real-world $P_{\rm h}$. $\mathcal{S}$ must "cook up" a simulated message of this form. It does this as follows. It computes

  $$\beta \xleftarrow{\text{R}} \{0,1\}, \quad p_0, p_1 \xleftarrow{\text{R}} \mathcal{M}, \quad r_0, r_1 \xleftarrow{\text{R}} \mathcal{R},$$

  just as in the real world, as well as

  $$c_0 \leftarrow H(p_0, r_0), \quad c_1 \leftarrow H(p_1, r_1). \tag{23.19}$$

- At some later point in time, $\mathcal{Z}$ will generate a control message that corresponds to the instant at which real-world $P_{\rm s}$ would have received both a message from real-world $P_{\rm h}$ and a message from real-world $P_{\rm r}$.

  At this point in time, $\mathcal{S}$ has obtained the message $(\tau, \hat{c}_0, \hat{c}_1)$ that real-world $P_{\rm r}$ sent to real-world $P_{\rm s}$. First, $\mathcal{S}$ checks that $\hat{c}_0 = c_0$ and $\hat{c}_1 = c_1$, where c_0, c_1 are computed as above in (23.19), just like $P_{\rm s}$ would have done.

  If this check does not pass, $\mathcal{S}$ will continue as if $P_{\rm s}$ aborted.

  Otherwise, $\mathcal{S}$ "extracts" $P_{\rm r}$'s effective input $\sigma := \tau \oplus \beta$ and submits this on $P_{\rm r}$'s to $\mathcal{F}_{\rm ot}$.

- Assuming the protocol did not abort, at some later point in time, $\mathcal{Z}$ will generate a control message that corresponds to the instant at which real-world $P_{\rm r}$ receives the message (e_0, e_1) from real-word $P_{\rm s}$.

  $\mathcal{S}$ must "cook up" a simulated message of this form. It does so as follows. Note that at this point in time, $\mathcal{F}_{\rm ot}$ must have received $P_{\rm s}$'s inputs m_0, m_1, but $\mathcal{S}$ is not directly privy to these inputs. However, $\mathcal{S}$ can instruct $\mathcal{F}_{\rm ot}$ to generate $P_{\rm r}$'s output m_σ, which $\mathcal{F}_{\rm ot}$ gives directly to $\mathcal{S}$. Having obtained m_σ, $\mathcal{S}$ computes

  $$e_\sigma \leftarrow m_\sigma \oplus p_\beta, \quad e_{\sigma\oplus 1} \xleftarrow{\text{R}} \mathcal{M}.$$

  So now $\mathcal{S}$ uses these two values e_0, e_1 for the simulated message.

That completes the description of the simulator $\mathcal{S}$.

We now show that (23.18) holds under the hiding assumption for the hash function.

To this end, let Game 0 be the real-world execution. Let us assume that the values c_0, c_1 are computed by $P_{\rm s}$ and $P_{\rm h}$ as in (23.19), so that so that

- $P_{\rm h}$ sends the value $c_{\beta\oplus 1}$ to $P_{\rm r}$, and
- when $P_{\rm s}$ receives a message $(\tau, \hat{c}_0, \hat{c}_1)$ from $P_{\rm r}$, it checks that $\hat{c}_0 = c_0$ and $\hat{c}_1 = c_1$.

Now define Game 1 to be the same as Game 0, except that instead of computing the values c_0, c_1 as in (23.19), $P_{\rm s}$ and $P_{\rm h}$ instead compute these values as

$$c_\beta \leftarrow H(p_\beta, r_\beta), \quad c_{\beta\oplus 1} \leftarrow H(p, r_{\beta\oplus 1}), \tag{23.20}$$

where $p \in \mathcal{M}$ is chosen at random. Note that in this game, $P_{\rm s}$ still uses the values p_0, p_1 as usual in the computation of e_0, e_1. (Of course, $P_{\rm s}$ could not do this in the real world, as it does not have access to the value β; however, that is not a problem, as this is just a thought experiment.) It is easily to see that the probability that the environment can distinguish Game 0 from Game 1 is bounded by the probability of distinguishing $H(p_{\beta\oplus 1}, r_{\beta\oplus 1})$ from $H(p, r_{\beta\oplus 1})$, which is negligible, by the hiding assumption for the hash function.

Now observe that in Game 1, the only place where $p_{\beta\oplus 1}$ is used at all is in the one-time pad encryption of $m_{\sigma\oplus 1}$, where $\sigma := \tau \oplus \beta$. So in Game 2, we further modify $P_{\rm s}$ so that it computes

$$e_{\sigma\oplus 1} \xleftarrow{\rm R} \mathcal{M}.$$

One can verify that Game 1 and Game 2 are completely equivalent from the point of view of the environment. Finally, one can also see that Game 2 is equivalent to the ideal world with our simulator.

Simulation when $P_{\rm s}$ is corrupt. Here is how $\mathcal{S}$ works when the sender, $P_{\rm s}$, is corrupt.

- At different points in time, $\mathcal{Z}$ will generate control messages that correspond to the instants at which real-world $P_{\rm s}$ receives the message (p_0, r_0, p_1, r_1) from real-world $P_{\rm h}$, and the message (τ, c_0, c_1) from real-world $P_{\rm r}$.

  $\mathcal{S}$ must "cook up" simulated messages of this form. It does so by computing

  $$p_0, p_1 \xleftarrow{\rm R} \mathcal{M}, \quad r_0, r_1 \xleftarrow{\rm R} \mathcal{R}, \quad c_0 \leftarrow H(p_0, r_0), \quad c_1 \leftarrow H(p_1, r_1), \quad \tau \xleftarrow{\rm R} \{0, 1\}.$$

  That is, $\mathcal{S}$ follows the protocol exactly to compute the values p_0, r_0, c_0 and p_1, r_1, c_1, but it just computes τ as a random bit.

- At some later point in time, $\mathcal{Z}$ will generate a control message that corresponds to the instant at which real-world real-world $P_{\rm r}$ receives the message (e_0, e_1) from real-word $P_{\rm s}$, and $\mathcal{S}$ obtains this message.

  $\mathcal{S}$ "extracts" $P_{\rm s}$'s effective inputs m_0, m_1 by computing

  $$m_0 \leftarrow e_0 \oplus p_\tau, \quad m_1 \leftarrow e_1 \oplus p_{\tau\oplus 1}.$$

  $\mathcal{S}$ then submits these inputs on behalf of $P_{\rm s}$ to $\mathcal{F}_{\rm ot}$. At this point in time, $\mathcal{F}_{\rm ot}$ has inputs from both $P_{\rm s}$ and $P_{\rm r}$, and so $\mathcal{S}$ instructs $\mathcal{F}_{\rm ot}$ to generate $P_{\rm s}$'s output at this time, which is forwarded directly to $\mathcal{Z}$.

That completes the description of the simulator $\mathcal{S}$. One can easily verify that (23.18) holds (perfectly).

23.5.9 An example malicious security proof: Beaver's 2.5-party protocol

We now prove that Beaver's 2.5-party protocol in Section 23.2.4 is secure in the malicious setting. Specifically, we prove:

Theorem 23.9. *Beaver's 2.5-party protocol in Section 23.2.4 securely implements the ideal functionality $\mathcal{F}_{\rm sfe}$ for arithmetic circuits over $\mathbb{Z}_q$, assuming q is super-poly, and that at most one party is corrupt.*

Proof. Let Π denote Beaver's protocol. By the completeness of the trivial adversary, it suffices to build a simulator $\mathcal{S}$ such that for every well-behaved environment $\mathcal{Z}$, we have

$$\text{Exec}[\Pi, \mathcal{A}_{\rm triv}, \mathcal{Z}] \approx [\text{Exec}[\mathcal{F}_{\rm sfe}, \mathcal{S}, \mathcal{Z}].$$

The reader should review the general proof strategy in the proof of Theorem 23.8, which applies equally well here (in particular, see the text labeled "review of the real world", "review of the ideal world", and "high-level proof strategy").

We now describe $\mathcal{S}$. So there are four cases to consider:

1. no party corrupt;

2. P_1 corrupt;

3. P_2 corrupt;

4. D corrupt.

We will focus on case 2, where P_1 is corrupt. Case 3, where P_2 is corrupt, is essentially the same. We leave Cases 1 and 4 to the reader.

The basic ideas of the operation of our simulator $\mathcal{S}$ are as follows:

- $\mathcal{S}$ will not see P_2's inputs. These inputs are chosen by the environment $\mathcal{Z}$ and forwarded directly to the ideal functionality $\mathcal{F}_{\rm sfe}$. However, $\mathcal{S}$ does receive a notification from $\mathcal{F}_{\rm sfe}$ that an input was given.

- $\mathcal{S}$ must somehow *extract* P_1's effective inputs, and feed these to $\mathcal{F}_{\rm sfe}$.

- $\mathcal{S}$ does not get to see P_2's outputs. Rather, $\mathcal{S}$ tells $\mathcal{F}_{\rm sfe}$ when to forward these outputs directly to $\mathcal{Z}$.

- $\mathcal{S}$ may ask $\mathcal{F}_{\rm sfe}$ for P_1's outputs.

- $\mathcal{S}$ must somehow play the roles of both P_2 and D, *cooking up* simulated protocol messages to send to P_1 as if they were coming from P_2 and D, in a way that fools $\mathcal{Z}$ into thinking it is still operating in the real world, even though $\mathcal{S}$ has no idea what P_2's inputs or outputs actually are.

Now the details.

The dealer. Recall that the dealer distributes singleton sharings $[K^{(1)}]$ and $[K^{(2)}]$, several authenticated singleton sharings $\llbracket a \rrbracket$, and several authenticated Beaver triple sharings $(\llbracket a \rrbracket, \llbracket b \rrbracket, \llbracket c \rrbracket)$, where $c = ab$. Also recall that an authenticated sharing $\llbracket x \rrbracket$ is a tuple $([x], [x^{(1)}], [x^{(2)}])$ of ordinary sharings such that $x^{(1)} = K^{(1)}x$ and $x^{(2)} = K^{(2)}x$.

For each sharing that the dealer would normally distribute, the simulator $\mathcal{S}$ will generate a random share for P_1, and then use the simulator $\mathcal{S}_D^{(1)}$, as described in Section 23.2.4.7, to simulate the rest of the dealing protocol. Looking forward, an invariant that we will maintain throughout the ideal world execution is this: ***$\mathcal{S}$ will know the values of all shares that P_1 is supposed to be holding (if it were actually following the protocol).*** However, $\mathcal{S}$ will *not* know the shares that P_2 should be holding.

Simulated opening logic. At various points in the protocol, a sharing $[x]$ is opened to P_1. As noted above, the simulator $\mathcal{S}$ will know the value of the share x_1 of x that P_1 is supposed to be holding. If $\mathcal{S}$ also knows the value x to be opened, then $\mathcal{S}$ can compute P_2's share x_2 as $x_2 \leftarrow x - x_1$. Therefore, if $\mathcal{S}$ knows the value being opened, it can cook up the message that P_2 would be expected to send to P_1 in the real execution of the protocol. We call this ***simulated opening logic***. As described below, $\mathcal{S}$ will make use of this logic at various points in its execution.

Reliable key opening. $\mathcal{S}$ will simulate the *reliable key opening* subprotocol (Protocol 23.4) as follows. First, ***$\mathcal{S}$ generates the value $K^{(1)}$ at random.*** (Note that $\mathcal{S}$ knows $K^{(1)}$, but does not know $K^{(2)}$.)

- Consider reliably opening $K^{(1)}$ to P_1. Recall that this subprotocol makes use of an authenticated singleton sharing $\llbracket a \rrbracket = ([a], [a^{(1)}], [a^{(2)}])$, and opens $[a]$, $[a^{(1)}]$, and $[K^{(1)}]$ to P_1. The simulator $\mathcal{S}$ will generate a at random, from which it can compute $a^{(1)} \leftarrow K^{(1)}a$. Since $\mathcal{S}$ knows a, $a^{(1)}$ and $K^{(1)}$, it may use the simulated opening logic to open $[a]$, $[a^{(1)}]$, and $[K^{(1)}]$ to P_1.

- Consider reliably opening $K^{(2)}$ to P_2. Recall that this subprotocol makes use of an authenticated singleton sharing $\llbracket a \rrbracket = ([a], [a^{(1)}], [a^{(2)}])$, and opens $[a]$, $[a^{(2)}]$, and $[K^{(2)}]$ to P_2. Also recall that $\mathcal{S}$ knows the values of the shares of a, $a^{(2)}$, and $K^{(2)}$ that P_1 is supposed to be holding. The logic of $\mathcal{S}$ is simply this: if the shares supplied by P_1 do not match these known shares, $\mathcal{S}$ makes P_2 abort (i.e., $\mathcal{S}$ proceeds as if P_2 aborted).

Working with authenticated sharings. Let us next discuss the low-level subprotocols for working with authenticated sharings: *open*, *add*, *multiply constant*, and *add constant*.

Again, recall that a valid authenticated sharing $\llbracket x \rrbracket$ is a tuple $([x], [x^{(1)}], [x^{(2)}])$ of ordinary sharings such that $x^{(1)} = K^{(1)}x$ and $x^{(2)} = K^{(2)}x$. We are assuming that $\mathcal{S}$ knows $K^{(1)}$, as well as

- P_1's share x_1 of x,

- P_1's share $x_1^{(1)}$ of $x^{(1)}$, and

- P_1's share $x_1^{(2)}$ of $x^{(2)}$.

Consider the operation opening such an authenticated sharing $\llbracket x \rrbracket$ to P_1. Here, P_2 needs to open $[x]$ and $[x^{(1)}]$ to P_1. If $\mathcal{S}$ also knows the value x, then $\mathcal{S}$ can also compute the value $x^{(1)} = K^{(1)}x$, and therefore the ***simulated opening logic*** can be extended to allow $\mathcal{S}$ to simulate the opening of the authenticated sharing $[\![x]\!]$ to P_1.

Consider the operation opening such an authenticated sharing $[\![x]\!]$ to P_2. Here, P_1 opens $[x]$ and $[x^{(2)}]$ to P_2. Our simulator $\mathcal{S}$ will always use ***guarded opening logic***, in which $\mathcal{S}$ simply makes P_2 abort if P_1 sends any wrong shares to P_2 (i.e., shares of x and $x^{(2)}$ that do not match those already known to $\mathcal{S}$). By using this logic, $\mathcal{S}$ will essentially force P_1 to correctly follow the protocol.

As for the other low-level subprotocols, *add*, *multiply constant*, and *add constant*, these are all just local computations. Since $\mathcal{S}$ knows all of P_1's shares going into these subprotocols, and these are just local computations, $\mathcal{S}$ also knows all of P_1's shares coming out of these subprotocols.

The main subprotocols. Now let's walk through the main subprotocols of Beaver's protocol. For each of these, we start be stating again the logic of the actual subprotocol, and then we describe the logic of the simulator $\mathcal{S}$.

Input wire for P_1: *to produce an authenticated sharing $[\![x]\!]$ of one of P_1's inputs $x \in \mathbb{Z}_q$, an authenticated singleton sharing $[\![a]\!]$ from the dealer is used as follows:*

1. *execute open $[\![a]\!]$ to P_1;*
2. *P_1 sends $\delta \leftarrow x - a$ to P_2;*
3. *execute $[\![x]\!] \leftarrow [\![a]\!] + \delta$.*

In Step 1, $\mathcal{S}$ *chooses a at random*, and uses the *simulated opening logic* with this value to open $[\![a]\!]$ to P_1. Now, in Step 2, P_1 can send any value δ that it likes. Our simulator $\mathcal{S}$ will *extract P_1's effective input $x := a + \delta$*, and feed this input to the ideal functionality $\mathcal{F}_{\text{sfe}}$. Step 3 is a local computation (so $\mathcal{S}$ computes the share of x that P_1 should be holding).

Input wire for P_2: *to produce an authenticated sharing $[\![x]\!]$ of one of P_2's inputs $x \in \mathbb{Z}_q$, an authenticated singleton sharing $[\![a]\!]$ from the dealer is used as follows:*

1. *execute open $[\![a]\!]$ to P_2;*
2. *P_2 sends $\delta \leftarrow x - a$ to P_1;*
3. *execute $[\![x]\!] \leftarrow [\![a]\!] + \delta$.*

The open in Step 1 will be done using the *guarded opening logic*. Now, in Step 2, $\mathcal{S}$ does not have access to the value of P_2's input x (this was passed directly to $\mathcal{F}_{\text{sfe}}$). Instead, $\mathcal{S}$ *just chooses the δ at random* and sends that to P_1. Again, Step 3 is a local computation (so $\mathcal{S}$ computes the share of x that P_1 should be holding).

Multiplication gate: *to multiply $[\![x]\!]$ and $[\![y]\!]$, obtaining $[\![z]\!] = [\![xy]\!]$, an authenticated Beaver triple sharing $([\![a]\!], [\![b]\!], [\![c]\!])$ from the dealer is used as follows:*

1. *execute $[\![u]\!] \leftarrow [\![x]\!] - [\![a]\!]$;*
2. *execute $[\![v]\!] \leftarrow [\![y]\!] - [\![b]\!]$;*
3. *execute open $[\![u]\!]$ and open $[\![v]\!]$ to both P_1 and P_2;*
4. *execute $[\![z]\!] \leftarrow uv + u[\![b]\!] + v[\![a]\!] + [\![c]\!]$.*

Steps 1 and 2 are local computations (so $\mathcal{S}$ computes the share of both u and v that P_1 should be holding). In Step 3, $\mathcal{S}$ *chooses the values* u *and* v *at random* and uses the *simulated opening logic* with these values to open $[\![u]\!]$ and $[\![v]\!]$ to P_1. In addition, $\mathcal{S}$ uses the *guarded opening logic* when P_1 opens $[\![u]\!]$ and $[\![v]\!]$ to P_2. Step 4 is a local computation (so $\mathcal{S}$ computes the share of z that P_1 should be holding).

Output wire for P_1: *to give to P_1 the value x of an authenticated sharing $[\![x]\!]$, execute open $[\![x]\!]$ to P_1.*

This subprotocol will only be run after all inputs have been extracted from P_1 and fed to $\mathcal{F}_{\text{sfe}}$, and all inputs from P_2 have been forwarded to $\mathcal{F}_{\text{sfe}}$. So at this point, $\mathcal{S}$ *requests the corresponding output value x from* $\mathcal{F}_{\text{sfe}}$ (by sending it an appropriate control message of the form $(\texttt{output}, \mathit{outputID}, 1)$). Using this output value x, $\mathcal{S}$ runs the *simulated opening logic.*

Output wire for P_2: *to give to P_2 the value x of an authenticated sharing $[\![x]\!]$, execute open $[\![x]\!]$ to P_2.*

Again, this subprotocol will only be run after all inputs have been extracted from P_1 and fed to $\mathcal{F}_{\text{sfe}}$, and all inputs from P_2 have been forwarded to $\mathcal{F}_{\text{sfe}}$. The simulator $\mathcal{S}$ runs the *guarded opening logic*, and then sends an appropriate control message to $\mathcal{F}_{\text{sfe}}$ that results in the corresponding output being forwarded directly to the environment as an output from P_2.

Proving that $\mathcal{S}$ works. That completes the description of $\mathcal{S}$. However, we still have to prove that $\mathcal{S}$ does the job. One can prove that for every environment $\mathcal{Z}$, we have

$$\big|\Pr[\mathrm{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}] = 1] - \Pr[\mathrm{Exec}[\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}] = 1]\big| \leq \frac{Q+1}{q},$$

where Q is the number of authenticated sharings that are opened to P_2 during the protocol. Note that Q is equal to the number of inputs for P_2, plus twice the number of multiplication gates, plus the number of outputs for P_2.

We sketch the proof outline, which can be organized as a sequence of games. In *Game$_j$*, we denote by W_j the event that the environment outputs 1.

Game$_0$: This is the real world.

Game$_1$: In this game, instead of running Protocol 23.5, we run the simulator $\mathcal{S}_D^{(1)}$, as described in Section 23.2.4.7. By Fact 23.2, we have $\Pr[W_0] = \Pr[W_1]$.

Game$_2$: In this game, we keep track of the shares that P_1 should be holding, and we introduce aborts if P_1 sends any bad shares in the *reliable key opening* subprotocol, or whenever P_1 opens an authenticated sharing to P_2. Based on the fact that we are using $\mathcal{S}_D^{(1)}$ in place of Protocol 23.5, and on the soundness analysis in Section 23.2.4.4, we have $|\Pr[W_2] - \Pr[W_1]| \leq (Q+1)/q$.

Game$_3$: This is the ideal world, with the simulator $\mathcal{S}$ interacting with the ideal functionality $\mathcal{F}_{\text{sfe}}$. It is not hard to see that the distribution of the environment's view in Game$_2$ is identical to that in Game$_3$. Indeed, we are essentially replacing one collection of random, independent values by another collection of random, independent values (the value $K^{(1)}$, the value a in the subprotocol for reliably opening $K^{(1)}$ to P_1, the value a in the P_1-input-wire subprotocol, the value δ in the in P_2-input-wire subprotocol, and the values u, v in the multiplication-gate subprotocol), and using the fact that the simulated opening logic is identical to the real opening logic, when the simulator uses the correct value to be opened. Therefore, $\Pr[W_2] = \Pr[W_3]$.

That completes the proof. □

23.5.10 An example malicious security proof: multi-party computation based on a secure distributed core

We sketch the idea of a proof based on the composition theorem (Theorem 23.5) for the protocol in Section 23.4.

Let Π denote this protocol. It involves parties $Q_1, \ldots, Q_N$, as well as P_1, P_2, and D. The assumption is that at most one of P_1, P_2, or D is corrupt, but any number of the Q_j's may be corrupt. Ultimately, we want to show that Π securely emulates $\mathcal{F}_{\text{sfe}}$. Protocol Π makes use of a protocol Π^* for secure arithmetic circuit evaluation. Also recall that in implementing this protocol, we relied on a feature of Beaver's protocol in which the parties could enforce a rule that some outputs must be delivered first and other outputs may not be delivered at all. Let us call this ideal functionality $\mathcal{F}^*_{\text{sfe}}$. Although we do not do so here, it is not too difficult to show that Π^* securely implements $\mathcal{F}^*_{\text{sfe}}$ by adapting the proof of Theorem 23.9. Note also that the circuits evaluated by $\mathcal{F}^*_{\text{sfe}}$ and $\mathcal{F}_{\text{sfe}}$ are not the same — as described in Section 23.4, the circuit of $\mathcal{F}^*_{\text{sfe}}$ is derived from that of $\mathcal{F}_{\text{sfe}}$, with some extra operations used to process inputs and outputs for the parties $Q_1, \ldots, Q_N$.

We sketch a partial proof of the following:

Theorem 23.10. *Protocol Π securely implements $\mathcal{F}_{\text{sfe}}$, assuming that the subprotocol Π^* securely implements $\mathcal{F}^*_{\text{sfe}}$, that q is super-poly, and that at most one of P_1, P_2, or D is corrupt.*

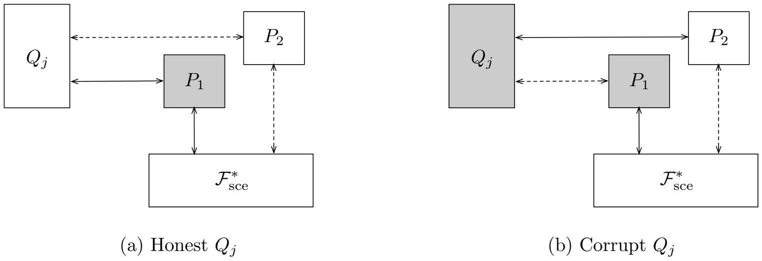
To make things more transparent, let us denote by P_1 and P_2 the machines that implement the high-level logic described in Section 23.4, and let us denote by P_1^* and P_2^* the machines that (along with D) securely implement $\mathcal{F}^*_{\text{sfe}}$. That is, for $i = 1, 2$, each party P_i:

- receives messages from their peers $Q_1, \ldots, Q_N$,
- passes inputs to P_i^*,
- receives outputs from P_i^*, and
- passes messages to their peers $Q_1, \ldots, Q_N$.

Now consider the hybrid protocol $\Pi_{\mathcal{F}^*_{\text{sfe}}}$, in which the subprotocol Π^* is replaced by the ideal functionality $\mathcal{F}^*_{\text{sfe}}$. Since Π^* securely implements $\mathcal{F}^*_{\text{sfe}}$, the composition theorem tells us that is suffices to show that the hybrid protocol $\Pi_{\mathcal{F}^*_{\text{sfe}}}$ securely implements $\mathcal{F}_{\text{sfe}}$. The beauty of this is that now, we do not have to worry about any of the details of the protocol Π^*.

We shall just sketch some of the ideas for the simulator. Let us assume that P_1 is corrupt, so that P_2 and D are honest. This means that P_1^* is also viewed as corrupt, and that P_2^* is honest. (Recall that for $i = 1, 2$, we view P_i and P_i^* as two programs running on the same machine, and that this machine is either corrupt or honest.)

The basic ideas of the operation our simulator $\mathcal{S}$ are as follows:



(a) Honest Q_j

(b) Corrupt Q_j

Figure 23.19: Proof of Theorem 23.10

- For an honest Q_j, the simulator $\mathcal{S}$ will not see Q_j's inputs. These inputs are chosen by the environment $\mathcal{Z}$ and forwarded directly to the ideal functionality $\mathcal{F}_{\text{sfe}}$. However, $\mathcal{S}$ does receive a notification from $\mathcal{F}_{\text{sfe}}$ that an input was given.

- For a corrupt Q_j, the simulator $\mathcal{S}$ must somehow extract Q_j's effective inputs, and feed these to $\mathcal{F}_{\text{sfe}}$.

- For an honest Q_j, the simulator $\mathcal{S}$ will not see Q_j's outputs. Rather, $\mathcal{S}$ tells $\mathcal{F}_{\text{sfe}}$ when to forward these outputs directly to $\mathcal{Z}$.

- For a corrupt Q_j, the simulator may ask $\mathcal{F}_{\text{sfe}}$ for Q_j's outputs.

- The simulator will play the roles of $\mathcal{F}^*_{\text{sfe}}$, P_2, and the honest Q_j's, interacting with P_1 and the corrupt Q_j's in a way that fools $\mathcal{Z}$ into thinking it is still operating in the hybrid world, even though $\mathcal{S}$ has no idea what the honest Q_j's inputs or outputs actually are.

As stated above, $\mathcal{S}$ will be playing the role of $\mathcal{F}^*_{\text{sfe}}$. This means that $\mathcal{S}$ will receive inputs normally intended for $\mathcal{F}^*_{\text{sfe}}$ from P_1, and must provide outputs to P_1 that would normally be coming from $\mathcal{F}^*_{\text{sfe}}$. However, in the ideal world in which $\mathcal{S}$ is running, there is no $\mathcal{F}^*_{\text{sfe}}$ — $\mathcal{S}$ must do all of the work that $\mathcal{F}^*_{\text{sfe}}$ would normally do (with the help of $\mathcal{F}_{\text{sfe}}$, of course).

Fig. 23.19 may be helpful in keeping track of things. The shaded boxes represent machines under adversarial control. In the hybrid world, the unshaded machines are running the hybrid protocol $\Pi_{\mathcal{F}^*_{\text{sfe}}}$. In the ideal world, the unshaded machines (including $\mathcal{F}^*_{\text{sfe}}$) are simulated by $\mathcal{S}$ — more precisely, communication between shaded and unshaded machines (shown in solid lines) is simulated.

We begin with discussing the input processing logic, which is the heart of the analysis. The reader should refer back to Section 23.4.1.

Honest Q_j**.**

- In Step 1 of that logic, in the hybrid world, P_1 would receive random shares A_1, B_1, C_1 from Q_j. So instead, $\mathcal{S}$ just gives P_1 random values A_1, B_1, C_1.

- In Step 2, in the hybrid world, for each input x from Q_j, P_1 would receive random shares x_1, E_1 from Q_j. So instead, $\mathcal{S}$ just gives P_1 random values x_1, E_1.

- In Step 3, in the hybrid world, P_1 inputs some values A_1', B_1', C_1', R_1' to $\mathcal{F}_{\text{sfe}}^*$. In the ideal world, since $\mathcal{S}$ is playing the role of $\mathcal{F}_{\text{sfe}}^*$, these values are actually submitted to $\mathcal{S}$, and $\mathcal{S}$ checks that $(A_1, B_1, C_1) = (A_1', B_1', C_1')$; if not, $\mathcal{S}$ notes that *an input inconsistency has occurred.*

- In Step 4, in the hybrid world, P_1 inputs some values x_1', E_1', S_1' to $\mathcal{F}_{\text{sfe}}^*$. In the ideal world, since $\mathcal{S}$ is playing the role of $\mathcal{F}_{\text{sfe}}^*$, these values are actually submitted to $\mathcal{S}$, and $\mathcal{S}$ checks that $(x_1, E_1) = (x_1', E_1')$; if not, $\mathcal{S}$ notes that *an input inconsistency has occurred.*

Corrupt Q_j**.**

- In the ideal world, Q_j will send some values A_2, B_2, C_2 to party P_2, and $\mathcal{S}$ sees these values. Also, $\mathcal{S}$ will see the corresponding values A_1, B_1, C_1, R_1 that P_1 inputs to $\mathcal{F}_{\text{sfe}}^*$.

  $\mathcal{S}$ sets $A := A_1 + A_2$, $B := B_1 + B_2$, and $C := C_1 + C_2$, and checks that $AB = C$; if not, it notes that *an input inconsistency has occurred.*

- In the ideal world, for each input wire belonging to Q_j, party Q_j will send some values x_2, E_2 to party P_2, and $\mathcal{S}$ sees these values. Also, $\mathcal{S}$ will see corresponding values x_1, E_1, S_1 that P_1 inputs to $\mathcal{F}_{\text{sfe}}^*$.

  $\mathcal{S}$ sets $x := x_1 + x_2$ and $E := E_1 + E_2$, and checks that $Ax = E$; if not, it notes that *an input inconsistency has occurred.* The value x is the "extracted input" that $\mathcal{S}$ feeds to $\mathcal{F}_{\text{sfe}}$, unless this input wire corresponds to an "output mask" G discussed in Section 23.4.2, in which case $\mathcal{S}$ just records this value for later use.

After all inputs are processed as above, if $\mathcal{S}$ noted at any step that an input inconsistency occurred, it sends P_1 a random, non-zero value V (as if it were an output from $\mathcal{F}_{\text{sfe}}^*$), and behaves as if P_2 has aborted, which brings the entire protocol to a halt without producing any outputs for any of the Q_j's. Otherwise, if no error has occurred, $\mathcal{S}$ deals with the output stage as follows.

- For each masked output (see Section 23.4.2) belonging to an honest Q_j, the simulator $\mathcal{S}$ delivers to P_1 a random value x' (as if it were an output from $\mathcal{F}_{\text{sfe}}^*$); if P_1 delivers this same value x' unchanged to Q_j, then $\mathcal{S}$ instructs $\mathcal{F}_{\text{sfe}}$ to deliver the corresponding unmasked output to the environment.

- For each masked output belonging to a corrupt Q_j, the simulator $\mathcal{S}$ obtains from $\mathcal{F}_{\text{sfe}}$ the unmasked output x, and sends to P_1 the value $x' := x + G$ (as if it were an output from $\mathcal{F}_{\text{sfe}}^*$), where G was an "output mask" that was previously input by Q_j and extracted as described above.

We leave it to the reader to verify that the above simulator works. The simulator for the case where P_2 is corrupt is essentially the same. We also leave it to the reader to provide details for the simulator in the case where D is corrupt. After filling in all of these details, one can prove Theorem 23.10.

23.5.11 An example malicious security proof: the 3-party garbled circuit protocol

We now prove that the 3-party garbled circuit protocol in Section 23.3.7 is secure in the malicious setting. Specifically, we prove:

Theorem 23.11. *The 3-party garbled circuit protocol in Section 23.3.7 securely implements the ideal functionality $\mathcal{F}_{\text{sfe}}$, under the assumptions listed in Section 23.3.7.2, and assuming that at most one party is corrupt.*

Proof. Let Π denote the 3-party garbled circuit protocol. By the completeness of the trivial adversary (Theorem 23.3), it suffices to build a simulator $\mathcal{S}$ such that for every well-behaved environment $\mathcal{Z}$, we have

$$\text{Exec}[\Pi, \mathcal{A}_{\text{triv}}, \mathcal{Z}] \approx \text{Exec}[\mathcal{F}_{\text{sfe}}, \mathcal{S}, \mathcal{Z}].$$

We describe simulators for the cases where P_1 is corrupt and where P_3 is corrupt. The case where P_2 is nearly identical to the case where P_1 is corrupt.

P_1 ***corrupt.*** We start by giving a description of a simulator $\mathcal{S}$. The basic ideas of the operations of $\mathcal{S}$ are as follows:

- $\mathcal{S}$ will not see P_2's inputs. These inputs are chosen by the environment $\mathcal{Z}$ and forwarded directly to the ideal functionality $\mathcal{F}_{\text{sfe}}$. However, $\mathcal{S}$ does receive a notification from $\mathcal{F}_{\text{sfe}}$ that an input was given.
- $\mathcal{S}$ must somehow extract P_1's effective inputs, and feed these to $\mathcal{F}_{\text{sfe}}$.
- $\mathcal{S}$ may ask $\mathcal{F}_{\text{sfe}}$ for the common output vector (which is the same for both P_1 and P_2), and tells $\mathcal{F}_{\text{sfe}}$ when to forward this output vector directly to $\mathcal{Z}$ as an output for P_2.
- $\mathcal{S}$ must somehow play the roles of P_2 and P_3, interacting with P_1 in a way that fools $\mathcal{Z}$ into thinking it is still operating in the real world, even though $\mathcal{S}$ has no idea what P_2's actual inputs are.

Here is how our simulator works. As we said above, it will play the role of both P_2 and P_3. When P_2 receives the seed s from P_1, the simulator generates all of the data computed in Step 2 of the protocol. When P_3 receives the data from P_1, the simulator performs the following computations, in lieu of those that P_3 would normally compute in Step (4) of the protocol:

(a) Check that the values $\mathcal{F}$ and $\{C_i^{(b)}\}_{i,b}$ received from P_1 match those generated from the seed — if not, abort.

(b) For $i = 1, \ldots, n$, if the ith input is contributed by P_1, check that $C_i^{(a_i)} = H_1(X_i, r_i)$; if not, abort; otherwise, *extract* the input $x_i := a_i \oplus b_i$ from P_1 (where $C_i^{(a_i)}$ and b_i are generated from the seed), and feed this input x_i to $\mathcal{F}_{\text{sfe}}$.

(c) Obtain the circuit output vector $\boldsymbol{y}$ from $\mathcal{F}_{\text{sfe}}$, and use the output simulatability property to compute a corresponding garbled output $\mathcal{Y}$, and send $\mathcal{Y}$ to P_1.

We leave it to the reader to verify that this simulator works, under the following assumptions:

- the collision resistance of the hash function H_1,
- the output simulatability of the garbling scheme.

P_3 ***corrupt.*** We start by giving a description of a simulator $\mathcal{S}$. The basic ideas of the operations of $\mathcal{S}$ are as follows:

- $\mathcal{S}$ will not see any of P_1 or P_2's inputs. These inputs are chosen by the environment $\mathcal{Z}$ and forwarded directly to the ideal functionality $\mathcal{F}_{\text{sfe}}$. However, $\mathcal{S}$ does receive a notification from $\mathcal{F}_{\text{sfe}}$ that an input was given.
- $\mathcal{S}$ does does not see the common output vector (which is the same for both P_1 and P_2). Rather, it tells $\mathcal{F}_{\text{sfe}}$ when to forward this output vector directly to $\mathcal{Z}$ as an output for either P_1 or P_2.
- $\mathcal{S}$ must somehow play the roles of P_1 and P_2, interacting with P_3 in a way that fools $\mathcal{Z}$ into thinking it is still operating in the real world, even though $\mathcal{S}$ has no idea what either P_1's or P_2's inputs or outputs actually are.

Here is how our simulator works. As we said above, it will play the role of both P_1 and P_2. In fact, the simulator will simply run the actual protocol for P_1 and P_2, with only the following changes:

- In Step 3 of the protocol, use 0 in place of x_i for $i = 1, \ldots, n$.
- In Step 5 of the protocol, instead of using the decoding data $\mathcal{d}$ to decode $\mathcal{Y}$, simply abort if $\mathcal{Y} \neq \mathit{Eval}(\mathcal{F}, \mathcal{X})$, where $\mathcal{X} = (X_1^{(0)}, \ldots, X_n^{(0)})$.

Of course, the actual inputs for P_1 and P_2 are forwarded directly from the environment to $\mathcal{F}_{\text{sfe}}$, and the outputs for P_1 and P_2 are forwarded directly from $\mathcal{F}_{\text{sfe}}$ to the environment (assuming no abort occurs).

One can show that this simulator works under the following assumptions:

- the hiding properties of the hash function H_1,
- the security of the pseudo-random generator G, and
- the obliviousness and authenticity properties of the garbling scheme.

The proof involves a sequence of games, which goes roughly as follows, starting from Game 0, the real world execution:

Game$_1$***:*** Using PRG security, replace the output of the PRG by truly random bits.

Game$_2$***:*** Modify the computation in Steps 2(b)–(d) of the protocol as follows:

(b) Compute $a_i \overset{\text{R}}{\leftarrow} \{0,1\}$ for $i = 1, \ldots, n$.

(c) Compute $r_i^{(b)} \overset{\text{R}}{\leftarrow} \mathcal{R}$ for $i = 1, \ldots, n$ and $b = 0, 1$.

(d) Compute $C_i^{(a_i \oplus b)} \leftarrow H_1(X_i^{(x_i \oplus b)}, r_i^{(a_i \oplus b)})$ for $i = 1, \ldots, n$ and $b = 0, 1$.

Also modify Step 3 of the protocol so that instead of computing a_i as $x_i \oplus b_i$, we just use the random value a_i as chosen above. This modification does not really change anything at all (instead of choosing b_i at random and setting $a_i := x_i \oplus b_i$, we are choosing a_i at random, and effectively setting $b_i := x_i \oplus a_i$).

***Game*$_3$:** Using hiding, for $i = 1, \ldots, n$, replace $C_i^{(a_i \oplus 1)}$ with $H_1(X, r_i^{(a_i \oplus 1)})$, where X is some fixed input token.

***Game*$_4$:** Using authenticity, do not use $\mathcal{d}$ to decode $\mathcal{Y}$ in Step 5 of the protocol. Rather, reject any garbled output $\mathcal{Y} \neq \mathit{Eval}(\mathcal{F}, X)$ received from P_3, and compute the output for P_1 and P_2 by simply evaluating the circuit.

***Game*$_5$:** Using obliviousness, and the fact that $\mathcal{d}$ is not used at all, use 0 in place of x_i in Steps 2 and 3 of the protocol. Specifically, for $i = 1, \ldots, n$, replace $C_i^{(a_i)}$ with $H_1(X_i^{(0)}, r_i^{(a_i)})$ and X_i with $X_i^{(0)}$.

***Game*$_6$:** Undo changes made in Game 3.

***Game*$_7$:** Undo changes made in Game 2.

***Game*$_8$:** Undo changes made in Game 1. This game is equivalent to the ideal world with our simulator.

We leave the remaining details to the reader. Note that we could have stopped at Game 5, but then the description of the simulator would have been a bit more complicated. □

23.6 Distributed key generation: ideal functionalities and extension to threshold MPC

In this section, we revisit the distributed key generation (DKG) problem that we studied back in Section 22.4. Our goals are twofold:

- First, we discuss how to adapt the definitions presented in Section 22.4.1.2 to the framework of this chapter.
- Second, to show how the protocol in Section 22.4.2 can be extended to obtain a threshold MPC protocol for evaluating an arbitrary arithmetic circuit. This protocol works with an arbitrary number N of parties, and is secure and provides guaranteed output delivery, provided at most $L < N/3$ of these parties are corrupt.

23.6.1 Formal models for secure DKG

We already gave a definition of security for a DKG protocol in Section 22.4.1.2. That definition was already in the same spirit of the ideal-world/real-world framework we have defined in this chapter in Section 23.5, but there are some differences.

23.6.1.1 Ideal worlds

The differences between the ideal world in Section 22.4.1.2 and the ideal world in Section 23.5 are essentially syntactic. The ideal world described in Section 22.4.1.2 describes an "adversary" $\mathcal{Z}$ interacting with a challenger and a simulator $\mathcal{S}$, where the challenger runs the *dealer* algorithm. This corresponds to an ideal world as in Section 23.5, where the role of the challenger is played by an *ideal functionality* $\mathcal{F}_{\text{dkg}}$, and the role of $\mathcal{Z}$ is played by the *environment*. Note that the ideal functionality $\mathcal{F}_{\text{dkg}}$ in this application does not take any inputs — it only produces outputs.

23.6.1.2 Real worlds

The real world execution in Section 22.4.1.2 corresponds to the execution of a *hybrid protocol* as in Section 23.5.5.2.

The real world described in Section 22.4.1.2 describes an "adversary" $\mathcal{Z}$ interacting with a challenger. The interactions here correspond to:

- the steps of decentralized key provisioning,
- messages sent to/from the honest parties from/to other parties, and
- queries made to any random oracles or other types of idealized object.

In the formal model of Section 23.5, the decentralized key provisioning steps and the queries to idealized objects would be modeled by an ideal functionality $\mathcal{F}^*$, and the DKG protocol would be modeled as a hybrid protocol $\Pi_{\mathcal{F}^*}$, as in Section 23.5.5.2. Unlike in Section 23.5.5.2, we do not really need a communication network $\mathcal{C}$, as the decentralized key provisioning functionality is sufficient to implement such a network.

The "adversary" $\mathcal{Z}$ in the ideal world of Section 22.4.1.2 corresponds to the environment of Section 23.5. In Section 22.4.1.2, $\mathcal{Z}$ interacts with directly with the challenger, which, in the framework of Section 23.5, corresponds to the environment $\mathcal{Z}$ interacting with the trivial adversary $\mathcal{A}_{\text{triv}}$.

For example, in Section 22.4.2.3, we presented a hybrid DKG protocol that relies on the following idealized components:

- decentralized key provisioning,
- a secret bulletin board, and
- a random oracle used in the VESS scheme (the zero-knowledge property for the VESS scheme relies explicitly on modeling a certain hash function as a random oracle).

The ideal functionality $\mathcal{F}^*$ would model these idealized components.

We also saw in Section 22.4.2.4 how we could implement a secret bulletin board in terms of a public bulletin board and a random oracle, and we noted that a public bulletin board can be implemented using a Byzantine Agreement protocol (as well as decentralized key provisioning, and sometimes also a random oracle). This gives rise to a hybrid DKG protocol that only relies on the following idealized components:

- decentralized key provisioning, and
- some random oracles.

As we noted in Section 22.4.2.4, Byzantine Agreement can be realized assuming $L < N/3$ (in the asynchronous network setting we are considering here). This is as close to a "real world" DKG protocol as we can get in our formal model: in an actual deployment, the decentralized key provisioning would have to be realized by some other mechanism (for example, using a certificate authority, or using some *ad hoc* manual process), and the random oracles would be (heuristically) instantiated with concrete hash functions.

23.6.1.3 Security proofs

Theorem 22.12 analyzes a specific DKG protocol that relies on decentralized key provisioning, a private bulletin board, and a secure VESS scheme. As discussed above, we can view this as a hybrid protocol $\Pi_{\mathcal{F}^*}$, where $\mathcal{F}^*$ is an ideal functionality that models the decentralized key provisioning, the secret bulletin board, and the random oracle used in the VESS scheme. Theorem 22.12 implies that protocol $\Pi_{\mathcal{F}^*}$ securely implements $\mathcal{F}_{\text{dkg}}$.

As noted above, a secret bulletin board can be implemented as a hybrid protocol that relies on decentralized key provisioning and a random oracle. Applying the Composition Theorem 23.5, this ultimately implies a hybrid protocol that securely implements $\mathcal{F}_{\text{dkg}}$, and which relies only on decentralized key provisioning and some random oracles.

23.6.1.4 Guaranteed output delivery

Our DKG protocol actual achieves guaranteed output delivery. Just as for the *liveness* property of Byzantine Agreement, this means that even though an adversary may arbitrarily delay the delivery of protocol messages, if and when all protocol messages sent from honest parties to honest parties have been delivered, all honest parties will obtain an output.

23.6.2 A threshold MPC protocol

In this section, we sketch how the techniques of Section 22.4.2 can be fairly easily extended to obtain a threshold MPC protocol for evaluating an arbitrary arithmetic circuit. This protocol works with an arbitrary number N of parties, and is secure and provides guaranteed output delivery, provided at most $L < N/3$ of these parties are corrupt. This protocol works in an asynchronous network setting, is secure against malicious adversaries, and relies only on decentralized key provisioning and a public bulletin board. We shall also assume secure channels, but this can easily be built on top of decentralized key provisioning, so this is not an extra assumption.

Just as in Section 22.4, we use the following notation:

- q is a prime;
- $\mathbb{G}$ is a group of order q generated by $g \in \mathbb{G}$;
- N is the number of parties,
- L is a bound on the number of corrupt parties.

We assume $L < N/3$: not only is this assumption needed to implement a public bulletin board, but the MPC protocol itself relies explicitly on this assumption.

We will make use of Shamir's t-out-of-N secret sharing scheme (see Section 22.1), where $t := L + 1$. We use the following notation, which slightly generalizes that in Section 22.4:

- If $u \in \mathbb{G}$ is a group elements and $\boldsymbol{\omega} = \kappa_0 + \kappa_1 x + \cdots + \kappa_{t-1} x^{t-1} \in \mathbb{Z}_q[x]$ is a polynomial of degree less than t with coefficients in $\mathbb{Z}_q$, then
$$u^{\boldsymbol{\omega}} := (u^{\kappa_0}, \ldots, u^{\kappa_{t-1}}) \in \mathbb{G}^t.$$

- If $\mathbf{U} = (u_0, \ldots, u_{t-1}) \in \mathbb{G}^t$ is a vector of group elements, and $\beta \in \mathbb{Z}_q$, we define the group element
$$\mathbf{U}^{(\beta)} := \prod_{j=0}^{t-1} u_j^{\beta^j}.$$

With this notation, we have the relation

$$(u^{\boldsymbol{\omega}})^{(\beta)} = u^{\boldsymbol{\omega}(\beta)}.$$

We introduce some additional notation. For $\mathbf{U}, \overline{\mathbf{U}} \in \mathbb{G}^t$, we define $\mathbf{U} \circ \overline{\mathbf{U}} \in \mathbb{G}^t$ to be the vector of group elements obtained by component-wise multiplication. With this notation, for $u, \bar{u} \in \mathbb{G}$ and $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}} \in \mathbb{Z}_q[x]$ of degree less than t, and $\beta \in \mathbb{Z}_q$, we have

$$(u^{\boldsymbol{\omega}} \circ \bar{u}^{\bar{\boldsymbol{\omega}}})^{(\beta)} = u^{\boldsymbol{\omega}(\beta)} \circ \bar{u}^{\bar{\boldsymbol{\omega}}(\beta)}.$$

For $\mathbf{U} \in \mathbb{G}^t$ and $\beta \in \mathbb{Z}_q$, we will write for $\mathbf{U}^\beta$ to be the vector of group elements obtained by component-wise exponentiation.

We assume that in addition to the generator $g \in \mathbb{G}$, we have a second, random generator $\bar{g} \in \mathbb{G}$ We will assume that the discrete logarithm problem in $\mathbb{G}$ is hard, and, in particular, that it is infeasible to compute $\log_g \bar{g}$. (The generator $\bar{g}$ could, for example, or be the output of a hash function modeled as a random oracle, or be generated using a secure DKG protocol; in fact, we could actually just use the DKG protocol built using a public bulletin board as discussed at the end of Section 22.4.2.4, since the only requirement on $\bar{g}$ is that it is hard to compute $\log_g \bar{g}$.)

23.6.2.1 Masked VESS

An essential tool in our threshold MPC protocol will be a generalization of the notion of a VESS scheme introduced in Section 22.4.2.1. We call this generalization a **masked VESS scheme**. Such a scheme works as follows. Given two polynomials $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}} \in \mathbb{Z}_q[x]$ of degree less than t, we can generate a **transcript** $(\mathbf{C}, \mathbf{U}, \pi)$, where

- $\mathbf{C} = (c_1, \ldots, c_N)$ is a vector of ciphertexts, where each c_i is an encryption of
$$(\alpha_i, \bar{\alpha}_i) := (\boldsymbol{\omega}(i), \bar{\boldsymbol{\omega}}(i)) \ \in \mathbb{Z}_q \times \mathbb{Z}_q$$
under P_i's public key;
- $\mathbf{U} := g^{\boldsymbol{\omega}} \circ \bar{g}^{\bar{\boldsymbol{\omega}}} \in \mathbb{G}^t$;
- π is a non-interactive zero-knowledge proof that c_i is indeed an encryption of $(\alpha_i, \bar{\alpha}_i) \in \mathbb{Z}_q$ under pk_i's public key satisfying
$$g^{\alpha_i} \bar{g}^{\bar{\alpha}_i} = \mathbf{U}^{(i)}$$
for $i = 1, \ldots, N$.

We call $\mathbf{U}$ the **polynomial commitment** of the transcript.

As in Section 22.4.2.1, there may be one public key per sender/receiver pair, in which case semantically secure encryption is sufficient. Alternatively, as in Remark 22.12, we can use just one public key per receiver, in which case we need an AD-only CCA-secure encryption scheme, where the associated data identifies the sender. Not only does this reduce the number of public keys dramatically, it also makes it easier to allow parties external to $P_1, \ldots, P_N$ to generate transcripts. This may be useful in some modes of deployment.

We describe here the intuition behind the security properties of such a transcript. Suppose we have a secret $\alpha \in \mathbb{Z}_q$, and as in Shamir's secret sharing scheme, we set $\kappa_0 := \alpha$ and choose $\kappa_1, \ldots, \kappa_{t-1} \in \mathbb{Z}_q$ at random, forming the polynomial

$$\boldsymbol{\omega} := \kappa_0 + \kappa_1 x \cdots + \kappa_{t-1} x^{t-1}.$$

Thus, $\boldsymbol{\omega}$ is a random polynomial of degree less than t, subject to $\boldsymbol{\omega}(0) = \alpha$. Suppose we also choose a random polynomial

$$\bar{\boldsymbol{\omega}} := \bar{\kappa}_0 + \bar{\kappa}_1 x \cdots + \bar{\kappa}_{t-1} x^{t-1} \in \mathbb{Z}_q[x]$$

of degree less than t, and then construct a transcript $(\mathbf{C}, \mathbf{U}, \pi)$ as above from $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$. We call this a **random transcript for the secret** α.

We argue that the adversary does not learn any information about α. We are assuming the adversary controls a set $\mathcal{L}$ of corrupt parties, where $|\mathcal{L}| \leq L$. For simplicity, let us assume that $|\mathcal{L}| = L$. So the adversary can decrypt the ciphertexts in $\mathbf{C}$ directed towards the corrupt parties to obtain $(\alpha_\ell, \bar{\alpha}_\ell)$ for $\ell \in \mathcal{L}$. As we already know (see Theorem 22.3), these shares are all random and independent of $(\kappa_0, \bar{\kappa}_0) = (\alpha, \bar{\kappa}_0)$. In addition, the first component $g^{\alpha} \bar{g}^{\bar{\kappa}_0}$ of the polynomial commitment $\mathbf{U}$ is a random group element that is independent of α, and all of the other components of $\mathbf{U}$ can be computed from this first component and the values $(\alpha_\ell, \bar{\alpha}_\ell)$, via interpolation in the exponent; therefore, the polynomial commitment $\mathbf{U}$ leaks no information about α. Moreover, by semantic security, the ciphertexts in $\mathbf{C}$ directed towards honest parties do not leak any information, and because of the zero knowledge property, neither does the proof π. This is why we called such a scheme a *masked* VESS scheme — it reveals *no* information about the secret α, not even g^{α}, as in an ordinary, "unmasked" VESS scheme.

That describes (at an intuitive level) the security property for a random transcript for a secret generated by an honest party. We now consider the security properties (again, at an intuitive level) for a transcript created by a corrupt party. We assume that such a transcript is valid, in the sense that the proof π is valid. Let $\mathcal{I}$ be the set of honest parties. By the soundness of the proof system, this means that the decryptions $(\alpha_i, \bar{\alpha}_i)$ for $i \in \mathcal{I}$ are all correct, in the sense that

$$g^{\alpha_i} \bar{g}^{\bar{\alpha}_i} = \mathbf{U}^{(i)} \qquad (\text{for } i \in \mathcal{I}).$$

By our assumption that $N > 3L$, we have $|\mathcal{I}| = N - L \geq t = L + 1$ (in fact, $N > 2L$ would have sufficed). This means that we (or a simulator) can choose a subset $\mathcal{I}'$ of t honest parties and interpolate their shares to obtain polynomials $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}} \in \mathbb{Z}_q[x]$ of degree less than t such that $g^{\boldsymbol{\omega}} \circ \bar{g}^{\bar{\boldsymbol{\omega}}} = \mathbf{U}$. It must be the case that $(\alpha_i, \bar{\alpha}_i) = (\boldsymbol{\omega}(i), \bar{\boldsymbol{\omega}}(i))$ for all $i \in \mathcal{I}$, assuming (as we are) that it is hard to compute $\log_g \bar{g}$. Indeed, if we had $(\alpha_i, \bar{\alpha}_i) \neq (\boldsymbol{\omega}(i), \bar{\boldsymbol{\omega}}(i))$, then these would be two distinct representations of $\mathbf{U}^{(i)}$ with respect to g and $\bar{g}$ (as in Section 10.6.1), which would allow us to compute $\log_g \bar{g}$. Similarly, if a corrupt party P_ℓ later reveals $(\alpha_\ell, \bar{\alpha}_\ell)$ such that

$$g^{\alpha_\ell} \bar{g}^{\bar{\alpha}_\ell} = \mathbf{U}^{(\ell)}$$

it must be the case that

$$(\alpha_\ell, \bar{\alpha}_\ell) = (\boldsymbol{\omega}(\ell), \bar{\boldsymbol{\omega}}(\ell)),$$

assuming it is hard to compute $\log_g \bar{g}$. Let us call $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$ the **underlying polynomials** of the transcript.

23.6.2.2 Sharings

Like Beaver's MPC protocol, our threshold MPC protocol evaluates an arithmetic circuit defined over $\mathbb{Z}_q$, and is based on the notion of a **sharing**. However, when a value is shared, the shares are derived as in Shamir's secret sharing scheme, but with a twist. If a value $\alpha \in \mathbb{Z}_q$ is shared among $P_1, \ldots, P_N$, there are polynomials $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}} \in \mathbb{Z}_q[x]$ of degree less than t such that $\boldsymbol{\omega}(0) = \alpha$, and the share held by each honest party P_i is the pair

$$(\alpha_i, \bar{\alpha}_i) := (\boldsymbol{\omega}(i), \bar{\boldsymbol{\omega}}(i)) \ \in \mathbb{Z}_q \times \mathbb{Z}_q.$$

Associated with such a share is a **polynomial commitment $\mathbf{U} \in \mathbb{G}^t$**, which is public and is agreed upon by all honest parties, and which satisfies

$$\mathbf{U} = g^{\boldsymbol{\omega}} \circ \bar{g}^{\bar{\boldsymbol{\omega}}}.$$

By design, the polynomials $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$ are guaranteed to exist and are efficiently computable from the information available (in aggregate) to the honest parties, assuming the soundness of the VESS proof system and that it is hard to compute $\log_g \bar{g}$. Let us call $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$ the **underlying polynomials** of the sharing.

Observe that for each honest party P_i, we have

$$g^{\alpha_i} \bar{g}^{\bar{\alpha}_i} = \mathbf{U}^{(i)}. \tag{23.21}$$

For a any party P_i, honest or corrupt, we say that a (purported) share $(\alpha_i, \bar{\alpha}_i)$ from P_i is **valid** if (23.21) holds. Assuming it is hard to compute $\log_g \bar{g}$, a valid share from P_i must be equal to $(\boldsymbol{\omega}(i), \bar{\boldsymbol{\omega}}(i))$.

Just as we did in Beaver's protocol, we will denote by $[\alpha]$ such a sharing. For each sharing $[\alpha]$, an honest party P_i stores the following local data:

- a polynomial commitment $\mathbf{U} \in \mathbb{G}^t$, and
- a share $(\alpha_i, \bar{\alpha}_i) \in \mathbb{Z}_q \times \mathbb{Z}_q$.

23.6.2.3 Reliable broadcast

In addition to a public bulletin board another tool from distributed computing we will need is a **Reliable Broadcast** protocol. Basically, such a protocol ensures that when a party broadcasts a message, if any honest party receives a message, then every honest party will eventually receive the same message. This prevents the situation where a corrupt party sends one message to one honest party, but a different message, or no message at all, to another honest party.

We will define more precisely the properties such a protocol must satisfy. Designing a Reliable Broadcast protocol is actually fairly easy — much easier than designing a public bulletin board. As such, we present a very simple protocol that satisfies these properties.

In a Reliable Broadcast protocol, we have parties $P_1, \ldots, P_N$, of which $L < N/3$ may be corrupt, and we assume an asynchronous communication network. In one instance of the protcol, each party P_i is allowed to broadcast a single message to all other parties, and each party P_j may *reliably receive* a single message from each party P_i.

The key properties of such a protocol are the following, which should hold for each instance of of the protocol's execution:

Totality: if one honest party reliably receives a message from P_i, then eventually each honest party does so.

Consistency: if two honest parties reliably receive messages m and m', respectively, from a party P_i, then $m = m'$.

Integrity: if an honest party reliably receives a message m from an honest party P_i, then P_i previously reliably broadcast m.

Validity: if an honest party P_i reliably broadcasts a message m, then every honest party eventually reliably receives m from P_i.

For the *totality* and *validity* properties, "eventually" means when all relevant messages generated by honest parties have been delivered.

Here is a very simple protocol for Reliable Broadcast. It assumes that each party has a signing key that is set up via decentralized key provisioning. A single instance of the protocol, with associated "instance ID" *iid*, runs as follows:

(1) Propose. When party P_i wishes to broadcast a message m, it sends the signed message $(\mathit{iid}, \texttt{propose}, i, m)$ to all parties.

(2) Vote. When a party receives the first signed message of the form $(\mathit{iid}, \texttt{propose}, i, m)$ from P_i, it sends the signed message $(\mathit{iid}, \texttt{vote}, i, m)$ to all parties.

(3) Commit. If for a given i, m, a party receives signed messages $(\mathit{iid}, \texttt{vote}, i, m)$ from $N - L$ distinct parties, the party will *reliably receive m from P_i*, and forward all of these vote messages to all other parties.

It is easy to show that this protocol satisfies all of the properties required for Reliable Broadcast, assuming secure signatures. *Totality* follows from the fact that if an honest party reliably receives a message from P_i, then it will forward the required vote messages to all other honest parties to make them do the same. *Consistency* follows from a counting argument. Suppose there are exactly L corrupt parties and so $N - L$ honest parties. Suppose one honest party *reliably receives* a message m from P_i. This means it must have received $N - L$ signed votes for m, and so at least $N - 2L$ honest parties must have voted for m. Similarly, if another honest party *reliably receives* a message $m' \neq m$ from P_i, at least $N - 2L$ honest parties must have voted for m'. Since an honest party only votes for one message from any given party, the set of honest parties who voted for m must be disjoint from the set of honest parties that voted for m'. Therefore,

$$(N - 2L) + (N - 2L) \leq N - L,$$

which implies $N \leq 3L$, which contradicts the assumption that $N > 3L$. We leave the argument for *integrity* and *validity* to the reader.

Note also that to reduce the communication complexity, one can use an $(N - L)$-out-of-N threshold signature on the vote messages — in the commit stage, only this threshold signature needs to be forwarded. However, for this to work, we need a scheme that satisfies the notion of *threshold signature gap security* introduced in Section 22.6.1. This will guarantee that any threshold signature on a particular vote message implies that at least $N - 2L$ honest parties threshold signed the message.

23.6.2.4 A threshold MPC protocol

We now describe our threshold MPC protocol for securely evaluating a given arithmetic circuit over $\mathbb{Z}_q$. As in Section 23.2, we have to present subprotocols for processing *input wires*, *addition gates*, *scalar multiplication gates*, *constant addition gates*, *multiplication gates*, and *output wires*, Just as Beaver's protocol in Section 23.2.2, this protocol first processes input wires. For each input wire, the protocol constructs a sharing $[\alpha]$ of the value α carried in that wire. Then the protocol processes each gate, computing a sharing of the value output by that gate, using the sharings of the input values to that gate. Finally, for output wires, the protocol reveals the value carried on the wire to the appropriate parties.

Input wires. Each honest party P_i does the following:

1. For each of its inputs α, construct a random transcript for α.
2. Construct a message containing all of these transcripts, and broadcast that message using a Reliable Broadcast protocol as in Section 23.6.2.3.
3. Using the Reliable Broadcast protocol, *reliably receive* a message from each party.
4. For each input α to the circuit, locate the transcript $(\mathbf{C}, \mathbf{U}, \pi)$ associated with that input, and record the local data for the sharing $[\alpha]$, which consists of (1) the polynomial commitment $\mathbf{U}$, and (2) the share $(\alpha_i, \bar{\alpha}_i)$ obtained by decrypting the ciphertext in $\mathbf{C}$ directed towards P_i.

   (Note that the underlying polynomials of the sharing $[\alpha]$ are the same as the underlying polynomials of the transcript $(\mathbf{C}, \mathbf{U}, \pi)$.)

Addition gates. These are very easily processed locally, without any communication. Suppose we have sharings $[\alpha]$ and $[\alpha^\dagger]$, and we want to compute the sharing $[\alpha + \alpha^\dagger]$. We assume that each party P_i has corresponding shares $(\alpha_i, \bar{\alpha}_i)$ and $(\alpha_i^\dagger, \bar{\alpha}_i^\dagger)$, along with corresponding polynomial commitments $\mathbf{U}$ and $\mathbf{U}^\dagger$. Its share of $\alpha + \alpha^\dagger$ can be locally computed as $(\alpha_i + \alpha_i^\dagger, \bar{\alpha}_i + \bar{\alpha}_i^\dagger)$, and the corresponding polynomial commitment can be locally computed as $\mathbf{U} \circ \mathbf{U}^\dagger$.

One can see that if the underlying polynomials of $[\alpha]$ are $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$, and the underlying polynomials of $[\alpha^\dagger]$ are $\boldsymbol{\omega}^\dagger, \bar{\boldsymbol{\omega}}^\dagger$, then the underlying polynomials of the sharing $[\alpha + \alpha^\dagger]$ are $\boldsymbol{\omega} + \boldsymbol{\omega}^\dagger$ and $\bar{\boldsymbol{\omega}} + \bar{\boldsymbol{\omega}}^\dagger$.

Scalar multiplication gates. These are also very easily processed locally, without any communication. Suppose we have a sharing $[\alpha]$ and a constant $\gamma \in \mathbb{Z}_q$, and we want to compute the sharing $[\gamma\alpha]$. We assume that each party P_i has a corresponding share $(\alpha_i, \bar{\alpha}_i)$ and polynomial commitment $\mathbf{U}$. Its share of $\gamma\alpha$ can be locally computed as $(\gamma\alpha_i, \gamma\bar{\alpha}_i)$, and the corresponding polynomial commitment can be locally computed as $\mathbf{U}^\gamma$.

One can see that if the underlying polynomials of $[\alpha]$ are $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$, then the underlying polynomials of the sharing $[\gamma\alpha]$ are $\gamma\boldsymbol{\omega}$ and $\gamma\bar{\boldsymbol{\omega}}$.

Constant addition gates. These are also very easily processed locally, without any communication. Suppose we have a sharing $[\alpha]$ and a constant $\gamma \in \mathbb{Z}_q$, and we want to compute the sharing $[\gamma + \gamma]$. We assume that each party P_i has a corresponding share $(\alpha_i, \bar{\alpha}_i)$ and polynomial commitment $\mathbf{U}$. Its share of $\gamma\alpha$ can be locally computed as $(\alpha_i + \gamma, \bar{\alpha}_i)$, and the corresponding polynomial commitment can be locally computed by multiplying each component of $\mathbf{U}$ by g^{γ}.

One can see that if the underlying polynomials of $[\alpha]$ are $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$, then the underlying polynomials of the sharing $[\alpha + \gamma]$ are $\boldsymbol{\omega} + \gamma$ and $\bar{\boldsymbol{\omega}}$.

Multiplication gates. This is the interesting case. Suppose we have sharings $[\alpha]$ and $[\alpha^{\dagger}]$ and we want to compute the sharing $[\alpha \cdot \alpha^{\dagger}]$. We assume that each party P_i has corresponding shares $(\alpha_i, \bar{\alpha}_i)$ and $(\alpha_i^{\dagger}, \bar{\alpha}_i^{\dagger})$, along with corresponding polynomial commitments $\mathbf{U}$ and $\mathbf{U}^{\dagger}$.

For this, we will need to use an instance of our public bulletin board subprotocol (we will need a separate instance for each multiplication gate). To understand the design of the multiplication subprotocol, suppose that the underlying polynomials of $[\alpha]$ are $\boldsymbol{\omega}, \bar{\boldsymbol{\omega}}$, and the underlying polynomials of $[\alpha^{\dagger}]$ are $\boldsymbol{\omega}^{\dagger}, \bar{\boldsymbol{\omega}}^{\dagger}$. Each party P_i will

- generate a random transcript $(\mathbf{C}_i, \mathbf{U}_i, \pi_i)$ for the secret $\boldsymbol{\omega}(i) \cdot \boldsymbol{\omega}^{\dagger}(i) \in \mathbb{Z}_q$, along with a non-interactive zero-knowledge proof π_i^* that this secret is what it is supposed to be (details of this below), and
- submit this augmented transcript $(\mathbf{C}_i, \mathbf{U}_i, \pi_i, \pi_i^*)$ to the public bulletin board.

The public bulletin board will then produce a collection

$$\{(\mathbf{C}_k, \mathbf{U}_k, \pi_k, \pi_k^*)\}_{k \in \mathcal{K}}$$

of $|\mathcal{K}| = 2L + 1$ valid augmented transcripts. By "valid", we mean that each proof π_k and π_k^* is valid.

The reason we need $2L + 1$ transcripts is that $\boldsymbol{\omega}$ and $\boldsymbol{\omega}^{\dagger}$ are each of degree at most $t - 1 = L$, and so the product polynomial $\boldsymbol{\omega} \cdot \boldsymbol{\omega}^{\dagger}$ has degree at most $2L$, and we want $2L + 1$ transcripts to be able to compute $\boldsymbol{\omega}(0) \cdot \boldsymbol{\omega}^{\dagger}(0)$ from the values $\{\boldsymbol{\omega}(k) \cdot \boldsymbol{\omega}^{\dagger}(k)\}_{k \in \mathcal{K}}$ via polynomial interpolation. Indeed, suppose $\{\lambda_k\}_{k \in \mathcal{K}}$ are the Lagrange interpolation coefficients (as in Lemma 22.1) such that

$$\boldsymbol{\omega}(0) \cdot \boldsymbol{\omega}^{\dagger}(0) = \sum_{k \in \mathcal{K}} \lambda_k \boldsymbol{\omega}(k) \boldsymbol{\omega}^{\dagger}(k),$$

which depend only on, and can be efficiently computed from, the set $\mathcal{K}$.

The polynomial commitment $\mathbf{U}^{\ddagger}$ for the sharing $[\alpha \cdot \alpha^{\dagger}]$ will be computed as

$$\mathbf{U}^{\ddagger} \leftarrow \prod_{k \in \mathcal{K}} \mathbf{U}_k^{\lambda_k}.$$

If $\mathbf{C}_k = (c_{k1}, \ldots, c_{kN})$ for $k \in \mathcal{K}$, party P_i will decrypt c_{ki} to obtain $(\alpha_{ki}, \bar{\alpha}_{ki})$ for each $k \in \mathcal{K}$, and then compute is share $(\alpha_i^{\ddagger}, \bar{\alpha}_i^{\ddagger})$ of $[\alpha \cdot \alpha^{\dagger}]$ as

$$\alpha_i^{\ddagger} \leftarrow \sum_{k \in \mathcal{K}} \lambda_k \alpha_{ki}, \qquad \bar{\alpha}_i^{\ddagger} \leftarrow \sum_{k \in \mathcal{K}} \lambda_k \bar{\alpha}_{ki}.$$

To see why this works, each augmented transcript $(\mathbf{C}_k, \mathbf{U}_k, \pi_k, \pi_k^*)$ will have underlying polynomials $\boldsymbol{\omega}_k, \bar{\boldsymbol{\omega}}_k$ such that

$$\boldsymbol{\omega}_k(0) = \boldsymbol{\omega}(k) \cdot \boldsymbol{\omega}^\dagger(k),$$

and

$$\boldsymbol{\omega}_k(i) = \alpha_{ki},\ \bar{\boldsymbol{\omega}}_k(i) = \bar{\alpha}_{ki} \text{ for each honest party } P_i.$$

It follows that the underlying polynomials of $[\alpha \cdot \alpha^\dagger]$ are

$$\boldsymbol{\omega}^\ddagger = \sum_{k \in \mathcal{K}} \lambda_k \boldsymbol{\omega}_k \quad \text{and} \quad \bar{\boldsymbol{\omega}}^\ddagger = \sum_{k \in \mathcal{K}} \lambda_k \bar{\boldsymbol{\omega}}_k.$$

Moreover, we have

$$\boldsymbol{\omega}^\ddagger(0) = \sum_{k \in \mathcal{K}} \lambda_k \boldsymbol{\omega}_k(0) = \sum_{k \in \mathcal{K}} \lambda_k \boldsymbol{\omega}(k) \boldsymbol{\omega}^\dagger(k) = \boldsymbol{\omega}(0) \boldsymbol{\omega}^\dagger(0),$$

as required.

Since we need to collect $2L+1$ transcripts in the bulletin board, we must have $2L+1 \leq N-L$, which is equivalent top saying $L < N/3$. This is the same bound needed to implement Byzantine Agreement (and Reliable Broadcast).

The last remaining detail is the design of the non-interactive zero-knowledge proof π_i^* that each P_i must construct to show that its transcript $(\mathbf{C}_i, \mathbf{U}_i, \pi_i)$ is a transcript for the secret $\boldsymbol{\omega}(i) \cdot \boldsymbol{\omega}^\dagger(i)$. This is fairly straightforward. First, we start with Sigma protocol for an appropriate relation (see Section 19.4). A statement for this relation is a triple $(u, v, w) \in \mathbb{G}^3$. A corresponding witness is a triple $(\beta, \bar{\beta}, \gamma) \in \mathbb{Z}_q^3$ such that

$$g^\beta \bar{g}^{\bar{\beta}} = v \quad \text{and} \quad u^\beta \bar{g}^\gamma = w.$$

We can build such a Sigma protocol as in Section 19.5.3 and convert it to a zero-knowledge non-interactive proof system via the Fiat-Shamir transform as in Section 20.3. Now, P_i will construct a proof for the statement

$$(u, v, w) = (\mathbf{U}^{(i)}, (\mathbf{U}^\dagger)^{(i)}, \mathbf{U}_i^{(0)})$$

using the witness

$$(\beta, \bar{\beta}, \gamma) = (\ \boldsymbol{\omega}^\dagger(i),\ \bar{\boldsymbol{\omega}}^\dagger(i),\ \bar{\boldsymbol{\omega}}_i(0) - \bar{\boldsymbol{\omega}}(i)\boldsymbol{\omega}^\dagger(i)\ ),$$

where $\boldsymbol{\omega}_i, \bar{\boldsymbol{\omega}}_i$ are the polynomials underlying the transcript $(\mathbf{C}_i, \mathbf{U}_i, \pi_i)$. Of course, P_i generated this transcript, and so it knows the polynomials $\boldsymbol{\omega}_i, \bar{\boldsymbol{\omega}}_i$. Moreover, we have $\boldsymbol{\omega}_i(0) = \boldsymbol{\omega}(i) \cdot \boldsymbol{\omega}^\dagger(i)$, by construction. Let us verify that this is indeed a witness. We have

$$g^\beta \bar{g}^{\bar{\beta}} = g^{\boldsymbol{\omega}^\dagger(i)} \bar{g}^{\bar{\boldsymbol{\omega}}^\dagger(i)} = (\mathbf{U}^\dagger)^{(i)} = v$$

as required. We also have

$$\begin{aligned} u^\beta \bar{g}^\gamma &= (\mathbf{U}^{(i)})^{\boldsymbol{\omega}^\dagger(i)} \cdot \bar{g}^{\bar{\boldsymbol{\omega}}_i(0) - \bar{\boldsymbol{\omega}}(i)\boldsymbol{\omega}^\dagger(i)} \\ &= (g^{\boldsymbol{\omega}(i)} \bar{g}^{\bar{\boldsymbol{\omega}}(i)})^{\boldsymbol{\omega}^\dagger(i)} \cdot \bar{g}^{\bar{\boldsymbol{\omega}}_i(0) - \bar{\boldsymbol{\omega}}(i)\boldsymbol{\omega}^\dagger(i)} \\ &= g^{\boldsymbol{\omega}(i)\boldsymbol{\omega}^\dagger(i)} \bar{g}^{\bar{\boldsymbol{\omega}}_i(0)} \\ &= g^{\boldsymbol{\omega}_i(0)} \bar{g}^{\bar{\boldsymbol{\omega}}_i(0)} \\ &= \mathbf{U}_i^{(0)} = w \end{aligned}$$

as required.

Output wires. Consider a output wire which carries the value α, and assume the protocol has already computed a corresponding sharing $[\alpha]$ with polynomial commitment $\mathbf{U}$.

1. To reveal α to a designated receiver P_j, each honest party P_i sends its share $(\alpha_i, \bar{\alpha}_i)$ to P_j. This is sent over a secure channel (but note that if the output wire is designated to reveal its value to all parties, party P_i can simply send this share to all parties in the clear).

2. To obtain the value α, a designated receiver P_j will wait until it obtains valid shares $(\alpha_i, \bar{\alpha}_i)$ from $t = L + 1$ distinct parties P_i, and then interpolate to obtain α as the constant term of the polynomial $\boldsymbol{\omega}$ that satisfies $\boldsymbol{\omega}(i) = \alpha_i$ for each such P_i.

Security. That completes the description of our threshold MPC protocol. While we do not do so here, one can show that this protocol securely emulates the ideal functionality $\mathcal{F}_{\text{sfe}}$. The protocol is a hybrid protocol that relies on

- decentralized key provisioning,
- a public bulletin board, and
- random oracles (used in the VESS and multiplication proof systems).

The proof of security relies on the hardness of the discrete logarithm problem, as well as the security properties of VESS scheme. The analysis of the proof system used in the multiplication subprotocol requires a kind of rewinding argument (such as we did for Schnorr signatures in Section 19.2) to actually extract a witness. Note that the simulator itself does not need to do any rewinding — the rewinding is only needed to show that the simulator works correctly.

Fairness. If we implement the public bulletin board using a Byzantine Agreement protocol, then the fairness property follows from (i) the fairness of the threshold MPC protocol, (ii) the *liveness* property of the Byzantine Agreement protocol, (iii) the *totality* property of the Reliable Broadcast subprotocol, and the (iv) fact that the protocol only delivers outputs after all inputs have been obtained via the Reliable Broadcast subprotocol. Once that happens, the protocol is guaranteed to deliver all outputs to all honest parties. Indeed, one of two things will happen: either all inputs are obtained and all outputs are delivered, or some inputs are not obtained and no outputs are delivered.

23.7 OT extension

Oblivious transfer (OT) is an important tool in cryptography: it is used in the garbled circuit MPC protocol (Section 23.3.6), Exercise 23.5 shows how it can be used to construct Beaver triple sharings for the Beaver MPC protocol (Section 23.2.2), and more applications are discussed in Section 11.6. All these applications require that the parties run many instances of the OT protocol.

Naively, to run N instances of an OT protocol, we can simply choose our favorite OT protocol (say, from Section 11.6) and run it N times. However, a beautiful technique called **OT extension** shows that after running only a small number of OT instances, it is possible to obtain many OT instances using only fast symmetric primitives such as a hash function and a PRF. This gives a tremendous speed-up to OT applications that require many instances of OT.

We will describe OT extension in the honest-but-curious settings. An enhancement to the basic protocol makes it secure against malicious adversaries.

Let us first recall a simple variation of OT previously discussed in Exercise 11.17. A 1-out-of-t **Random OT**, or **ROT**, is a 2-party protocol where

- *input:* the sender has no input, and the receiver has a choice value $s \in \{1, \dots, t\}$,
- *output:* the sender has a random tuple $(y[1], \dots, y[t]) \xleftarrow{\text{R}} \mathcal{Y}^t$, and the receiver has $y[s] \in \mathcal{Y}$.

The difference from standard 1-out-of-t OT is that the sender does not specify an input tuple, but is instead given a random one at the end. Exercise 11.17 shows how to efficiently construct a standard 1-out-of-t OT from 1-out-of-t ROT. Therefore, it suffices to focus on building many instances of 1-out-of-t ROT.

Recall also from Exercise 11.17 that a 1-out-of-2 **OT correlation** is a state where the sender has a random pair $(k_1, k_2) \in \mathcal{K}^2$ and the recipient has $(r, k_r) \in \{0, 1\} \times \mathcal{K}$, where r is random in $\{0, 1\}$. Neither party knows anything else about the other party's data.

The OT extension protocol has two phases. In a setup phase the two parties establish n pairs of 1-out-of-2 OT correlations. Then in the online phase, the two parties use the setup data to quickly run N instances of 1-out-of-t ROT, for some N that is much bigger than n. The setup phase may require a complex protocol, but once that's done, the parties can quickly generate as many ROTs as needed. For the applications of OT in this chapter it suffices to set $t = 2$, but the construction works for larger t. We will need the following tools:

- *Parameters:* given n 1-out-of-2 OT correlations we build N instances of 1-out-of-t ROT, where N is much larger than n. The goal is to minimize n so as to minimize the cost of setup.
- We will do arithmetic in $\mathbb{Z}_\ell$, for some prime ℓ. One often simply sets $\ell = 2$.
- Let F be a secure PRF defined over $(\mathcal{K},\ \{1, \dots, N\},\ \mathbb{Z}_\ell)$.
- Let $H_1, \dots, H_N : \mathbb{Z}_\ell^n \to \mathcal{Y}$ be N hash functions that will be modeled as independent random oracles. One can derive all the H_i from a single hash function H as $H_i(x) := H(i, x)$.

In addition, we will need a list of t codewords, $\mathcal{C} = (\mathcal{C}[1], \dots, \mathcal{C}[t])$, where each codeword $\mathcal{C}[i]$ is a vector in $\mathbb{Z}_\ell^n$. These codewords must be far apart: for all $i \neq j$ the Hamming distance between $\mathcal{C}[i]$ and $\mathcal{C}[j]$ must be at least n' for some $n' \leq n$ (i.e., $\mathcal{C}[i] - \mathcal{C}[j]$ has at least n' non-zero entries). As we will see, to maintain security, we will need n' to be sufficiently large so that $1/2^{n'}$ is negligible (e.g., set $n' := 128$). The parameters ℓ, t, n, n' need to be chosen so that there is an explicit list of codewords $\mathcal{C}$ satisfying this property. For example, if $t = 2$, then we can take $\mathcal{C} := (0^n, 1^n)$ and $n' := n$.

The setup phase. Two parties, a sender and a receiver, prepare n OT correlations:

- The receiver has n random pairs $(k_{j,0},\ k_{j,1}) \in \mathcal{K}^2$ for $j = 1, \dots, n$.
- The sender has n pairs $(r_j, \hat{k}_j)$, where $r_j \in \{0, 1\}$ functions as a choice bit so that $\hat{k}_j = k_{j,r_j}$ for $j = 1, \dots, n$. All the choice bits in $\boldsymbol{r} := (r_1, \dots, r_n)$ are sampled uniformly from $\{0, 1\}$. We will treat $\boldsymbol{r}$ as a binary vector in $\mathbb{Z}_\ell^n$.

Neither party knows anything else about the other party's data. Notice that we reversed the roles of the sender and the receiver: the receiver has $(k_{j,0}, k_{j,1})$ and the sender has the choice bit $r_j \in \{0,1\}$ and k_{j,r_j}. The two parties can generate these n OT correlations using any of the OT protocols from Section 11.6.

The end goal. OT extension uses the setup data to implement N instances of 1-out-of-t ROT. Specifically, in addition to the setup data, the receiver also takes as input N choice values $s_1, \dots, s_N \in \{1, \dots, t\}$. At the end of the OT extension protocol:

- the sender has N tuples $\boldsymbol{y}_1, \dots, \boldsymbol{y}_N \in \mathcal{Y}^t$, that are indistinguishable from N random tuples;
- the receiver has $\hat{y}_1, \dots \hat{y}_N \in \mathcal{Y}$, where $\hat{y}_i = \boldsymbol{y}_i[s_i]$ for $i = 1, \dots, N$.

Here the receiver obtained one entry from each sender tuple, namely entry number s_i from tuple number i. Neither party should learn anything else about the other's party's data.

The OT extension protocol. The protocol, between the sender and receiver, is remarkably simple, using only a single message.

- *Receiver:* for $i = 1, \dots, N$ the receiver computes:

$$\begin{aligned} \boldsymbol{u}_i &\leftarrow \Big(F(k_{1,0}, i), \dots, F(k_{n,0}, i)\Big) \ \in \mathbb{Z}_\ell^n, \\ \boldsymbol{v}_i &\leftarrow \Big(F(k_{1,1}, i), \dots, F(k_{n,1}, i)\Big) \ \in \mathbb{Z}_\ell^n, \\ \boldsymbol{p}_i &\leftarrow \boldsymbol{v}_i - \boldsymbol{u}_i + \mathcal{C}[s_i] \ \in \mathbb{Z}_\ell^n. \end{aligned}$$

  The receiver's output for ROT number i is $\hat{y}_i \leftarrow H_i(\boldsymbol{u}_i) \in \mathcal{Y}$.

- *Receiver → Sender:* send $(\boldsymbol{p}_1, \dots, \boldsymbol{p}_N) \in \mathbb{Z}_\ell^{n \times N}$.

- *Sender:* for $i = 1, \dots, N$ the sender computes:

$$\begin{aligned} \boldsymbol{q}_i &\leftarrow \Big(F(k_{1,r_1}, i), \ \dots, \ F(k_{n,r_n}, i)\Big) - \boldsymbol{r} \circ \boldsymbol{p}_i \ \in \mathbb{Z}_\ell^n, \\ \boldsymbol{y}_i &\leftarrow \Big(H_i\big(\boldsymbol{q}_i + (\boldsymbol{r} \circ \mathcal{C}[1])\big), \ \dots, \ H_i\big(\boldsymbol{q}_i + (\boldsymbol{r} \circ \mathcal{C}[t])\big)\Big) \ \in \mathcal{Y}^t, \end{aligned}$$

  where $\circ$ denotes the entry-wise product of two vectors, so that $\boldsymbol{r} \circ \boldsymbol{c} := (r_1 c_1, \dots, r_n c_n) \in \mathbb{Z}_\ell^n$. The sender's output for ROT number i is $\boldsymbol{y}_i \in \mathcal{Y}^t$.

That's it. As promised, the parties only use simple symmetric primitives, namely a PRF and a hash function.

Let's first verify that the protocol is correct, namely that $\boldsymbol{y}_i[s_i] = \hat{y}_i = H_i(\boldsymbol{u}_i)$ for all $i = 1, \dots, N$. We can write the vector $\boldsymbol{q}_i \in \mathbb{Z}_\ell^n$ computed by the sender as

$$\boldsymbol{q}_i = \boldsymbol{u}_i + \boldsymbol{r} \circ (\boldsymbol{v}_i - \boldsymbol{u}_i) - \boldsymbol{r} \circ \boldsymbol{p}_i = \boldsymbol{u}_i - \boldsymbol{r} \circ \mathcal{C}[s_i].$$

where the last equality follows by plugging in $\boldsymbol{p}_i = \boldsymbol{v}_i - \boldsymbol{u}_i + \mathcal{C}[s_i]$ and canceling terms. Then, by definition of $\boldsymbol{y}_i$, for $a = 1, \dots, t$ we have

$$\boldsymbol{y}_i[a] = H_i\Big(\boldsymbol{q}_i + \boldsymbol{r} \circ \mathcal{C}[a]\Big) = H_i\Big(\boldsymbol{u}_i + \boldsymbol{r} \circ (\mathcal{C}[a] - \mathcal{C}[s_i])\Big). \tag{23.22}$$

Setting $a = s_i$ shows that $\boldsymbol{y}_i[s_i] = H_i(\boldsymbol{u}_i)$, as required.

Remark 23.8. Notice that in the OT extension protocol, in computing the quantities $\boldsymbol{u}_i$, $\boldsymbol{v}_i$, and $\boldsymbol{q}_i$, for $i = 1, \ldots, N$, the same PRF keys are used for all i. If $\ell = 2$, so the output of the PRF is just a single bit, then we can can implement the protocol using, say, AES, which outputs 128 bits, so that the total number of applications of AES for the receiver (resp., sender) is just $2\lceil Nn/128\rceil$ (resp., $\lceil Nn/128\rceil$). If we take $n = 128$ (as discussed below), the number of AES applications is $2N$ (resp., N). □

Security for the recipient. The sender clearly learns nothing about the recipient's choice value s_i. This s_i is only used in $\boldsymbol{p}_i$ where it is blinded by $\boldsymbol{v}_i - \boldsymbol{u}_i$. The vector $\boldsymbol{v}_i - \boldsymbol{u}_i$ is indistinguishable from random to the sender because the sender only knows one of the two keys used in each component of the vector.

Security for the sender. We need to argue that the receiver learns nothing about the sender's tuples $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N \in \mathcal{Y}^t$, other than the entries selected by the receiver's choice values $s_1, \ldots, s_N$.

Let's first consider the simple case when $t = 2$, $\mathcal{C} = (0^n, 1^n)$, and $n' = n$. In addition, let's assume $\ell = 2$, so that addition in $\mathbb{Z}_\ell$ is simply an XOR. For these parameters, $\boldsymbol{y}_i = (\boldsymbol{y}_i[1], \boldsymbol{y}_i[2])$ is a vector in $\mathcal{Y}^2$.

- One entry in $\boldsymbol{y}_i$ is $\boldsymbol{y}_i[s_i] = H_i(\boldsymbol{u}_i)$, where $\boldsymbol{u}_i \in \mathbb{Z}_2^n$ is chosen by the honest recipient.
- By (23.22), the other entry in $\boldsymbol{y}_i$ is $\boldsymbol{y}_i[3 - s_i] = H_i(\boldsymbol{u}_i + \boldsymbol{r} \circ 1^n) = H_i(\boldsymbol{u}_i + \boldsymbol{r})$. This quantity should be hidden from the recipient, for all $i = 1, \ldots, N$.

The requirement in the second bullet is equivalent to saying that no efficient adversary that is given $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N \in \mathbb{Z}_2^n$, can distinguish the distribution $\big(H_1(\boldsymbol{u}_1 + \boldsymbol{r}), \ldots, H_N(\boldsymbol{u}_N + \boldsymbol{r})\big)$, where $\boldsymbol{r} \xleftarrow{\text{R}} \mathbb{Z}_2^n$, from the uniform distribution on $\mathcal{Y}^N$. This is really the only security property we require from the hash functions $H_1, \ldots, H_N$, and it is easy to verify that this property holds when $H_1, \ldots, H_N$ are modeled as independent random oracles, assuming $1/2^n$ is negligible. In practice, taking $n := 128$ is sufficient. Recall that n determines the number of OT correlations that the parties need to generate in the setup phase. Also recall that, as we mentioned earlier, one can instantiate $H_1, \ldots, H_N$ by setting $H_i(\boldsymbol{x}) := H(i, \boldsymbol{x})$ for some fixed hash function H such as SHA256. This completes the analysis of sender security when $\mathcal{C} = (0^n, 1^n)$.

The analysis for general parameters t, ℓ, n, n' is similar. For $\boldsymbol{r} \in \{0,1\}^n$, define the set of $t(t-1)$ related keys

$$\mathcal{K}_+(\boldsymbol{r}) := \Big\{\boldsymbol{k}_{i,j} := \boldsymbol{r} \circ (\mathcal{C}[i] - \mathcal{C}[j]) \in \mathbb{Z}_\ell^n \quad \text{for} \quad 1 \le i \ne j \le t\Big\}$$

By (23.22) we know that $\boldsymbol{y}_i[a] = H_i(\boldsymbol{k}_{a,s_i} + \boldsymbol{u}_i)$ for $a = 1, \ldots, t$ and $i = 1, \ldots, N$. For security, we need that no efficient adversary that is given $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_N \in \mathbb{Z}_2^n$, can distinguish the distribution

$$\big(H_i(\boldsymbol{k}_{a,s_i} + \boldsymbol{u}_i) \quad \text{for} \quad i = 1, \ldots, N, \quad a = 1, \ldots, t, \quad a \ne s_i\big) \quad \text{where } \boldsymbol{r} \xleftarrow{\text{R}} \mathbb{Z}_2^n,$$

from the uniform distribution on $\mathcal{Y}^{N(t-1)}$. When the functions $H_1, \ldots, H_N$ are modeled as independent random oracles, this holds assuming $(t-1)/2^{n'}$ is negligible. Recall that n' is the minimum Hamming distance between the codewords in $\mathcal{C}$, so that $\mathcal{C}[i] - \mathcal{C}[j]$ contains at least n' non-zero entries for all $i \ne j$. In practice, taking $n' := 128 + \lceil \log_2 t \rceil$ is sufficient. One then has to find the smallest n for which there is a set of codewords $\mathcal{C} \subseteq \mathbb{Z}_\ell^n$ of size t with minimal Hamming distance n'.

We can eliminate the additive term $\lceil \log_2 t \rceil$ from n', and simply set $n' := 128$, by adding the coordinate number as an input to the hash functions. Specifically, let $H_{i,a}(\boldsymbol{x}) := H(i, a, \boldsymbol{x})$, for some fixed hash function H. Now, define the sender's output as

$$\boldsymbol{y}_i \leftarrow \Big(H_{i,1}\big(\boldsymbol{q}_i + (\boldsymbol{r} \circ \mathcal{C}[1])\big),\ \ldots,\ H_{i,t}\big(\boldsymbol{q}_i + (\boldsymbol{r} \circ \mathcal{C}[t])\big)\Big) \ \in \mathcal{Y}^t.$$

Notice that every entry now uses a different hash function. Similarly, the receiver's output is $\hat{y}_i \leftarrow H_{i,s_i}(\boldsymbol{u}_i) \in \mathcal{Y}$. If we model all these hash functions as independent random oracles, then security holds assuming $1/2^{n'}$ is negligible. With this modification, taking $n' := 128$ is sufficient.

An attack by a corrupt receiver. The OT extension scheme is insecure if the receiver is corrupt and does not honestly follow the protocol. Let's see an attack. Let $\boldsymbol{e}_j \in \mathbb{Z}_\ell^n$ be a vector that is zero everywhere, except for coordinate j where it is one. Suppose a corrupt receiver computes

$$\boldsymbol{p}_1^* \leftarrow \boldsymbol{v}_1 - \boldsymbol{u}_1 + (\mathcal{C}[1] - \boldsymbol{e}_1) \in \mathbb{Z}_\ell^n \tag{23.23}$$

and sends this $\boldsymbol{p}_1^*$ to the sender in place of a valid $\boldsymbol{p}_1$. Then by (23.22) the sender obtains

$$\boldsymbol{y}_1[1] = H_1\Big(\boldsymbol{u}_1 + \boldsymbol{r} \circ \big(\mathcal{C}[1] - (\mathcal{C}[1] - \boldsymbol{e}_1)\big)\Big) = H_1\Big(\boldsymbol{u}_1 + \boldsymbol{r} \circ \boldsymbol{e}_1\Big).$$

Suppose that the receiver learns this $\boldsymbol{y}_1[1]$ due to a later step in a larger protocol. Now the receiver has $\boldsymbol{y}_1[1] = H_1(\boldsymbol{u}_1 + \boldsymbol{r} \circ \boldsymbol{e}_1)$ and it knows $\boldsymbol{u}_1$. This reveals the first bit of $\boldsymbol{r} \in \{0,1\}^n$. Repeating this by constructing $\boldsymbol{p}_2^*$ using $\boldsymbol{e}_2$ as in (23.23) reveals the second bit of $\boldsymbol{r}$. Continuing this way using the first n ROTs reveals all of $\boldsymbol{r}$ to the recipient. Once $\boldsymbol{r}$ is known, the remaining $(N - n)$ ROTs are no longer secure because the recipient can compute all the sender's data. There are efficient enhancements to OT extension that make it secure against a malicious recipient [137].

23.8 A fun application: another stab at private set intersection

To be written.

23.9 Notes

Citations to the literature to be added.

23.10 Exercises

23.1 (Majority as an arithmetic circuit). For odd n, the majority function takes as input n binary values $x_1, \ldots, x_n \in \{0,1\}$. It outputs 1 if the majority of the inputs is 1, and outputs zero otherwise. Show how to implement the majority function as an arithmetic circuit over $\mathbb{Z}_q$, for some prime $q > n$. Your arithmetic circuit should have just $n - 1$ multiplication gates and $O(n)$ addition gates.

Hint: use polynomial interpolation.

23.2 (Table lookup as an arithmetic circuit). Consider the function $f : \mathbb{Z}_q^{n+1} \to \{0,1\}$, defined as follows:

$$f(x_1, \ldots, x_n,\ x) := \begin{cases} 1 & \text{if } x \in \{x_1, \ldots, x_n\}, \\ 0 & \text{otherwise.} \end{cases}$$

Give an arithmetic circuit for f that uses $O(n + \log q)$ multiplication gates and $O(n)$ addition/subtraction gates.

Hint: use Fermat's little theorem.

23.3 (Generating Beaver triple sharings I). In Section 23.2.2 we described Beaver's 2-party protocol for evaluating an arithmetic circuit. Our description relied on a dealer D to provide the two parties P_1 and P_2 with Beaver triple sharings. If P_1 and P_2 could generate these on their own, without the dealer, the protocol would become a true 2-party protocol. Let's see how to do that in the honest-but-curious setting.

Let (G, E, D) be a public key encryption scheme with message space $\mathbb{Z}_q$. Moreover, let's assume that the scheme is additively homomorphic, so that for all $x, y \in \mathbb{Z}_q$ and $(pk, sk) \xleftarrow{\text{R}} G()$:

- there is an efficient probabilistic algorithm *Sum* that takes pk and two ciphertexts $E(pk, x)$ and $E(pk, y)$ as input, and outputs a fresh ciphertext distributed as $E(pk, z)$, where $z := x + y \in \mathbb{Z}_q$;
- there is an efficient probabilistic algorithm *ScalarMul* that takes pk, a ciphertext $E(pk, x)$, and scalar $s \in \mathbb{Z}_q$ as input, and outputs a fresh ciphertext distributed as $E(pk, z)$, where $z := s \cdot x \in \mathbb{Z}_q$.

We saw an additively homomorphic scheme in Exercise 11.13.

Now the protocol:

$\underline{P_1:}$		$\underline{P_2:}$
$(pk, sk) \xleftarrow{\text{R}} G()$		
$a_1, b_1 \xleftarrow{\text{R}} \mathbb{Z}_q$		
$\varepsilon_a \xleftarrow{\text{R}} E(pk, a_1)$, $\varepsilon_b \xleftarrow{\text{R}} E(pk, b_1)$	$\xrightarrow{(pk, \varepsilon_a, \varepsilon_b)}$	$a_2, b_2, c_2 \xleftarrow{\text{R}} \mathbb{Z}_q$
		compute a fresh encryption ε of
		$a_2 \cdot b_1 + b_2 \cdot a_1 + (a_2 b_2 - c_2) \in \mathbb{Z}_q$
		output (a_2, b_2, c_2)
$c_1 \leftarrow D(sk, \varepsilon) + a_1 b_1 \in \mathbb{Z}_q$	$\xleftarrow{\varepsilon}$	
output (a_1, b_1, c_1)		

Here, P_2 only knows encryptions of a_1 and b_1, but the homomorphic property allows it to efficiently to construct a fresh encryption ε of $a_2 \cdot b_1 + b_2 \cdot a_1 + (a_2 b_2 - c_2)$, as required.

In the above protocol, we assume that random elements of $\mathbb{Z}_q$ are generated as in Remark 23.5.

(a) Show that at the end of the protocol we have $c_1 + c_2 = (a_1 + a_2)(b_1 + b_2)$.

(b) Show exactly how P_2 should compute ε, using the given algorithms *Sum* and *ScalarMul*.

(c) You are now to show that if (G, E, D) is semantically secure, then the above protocol is secure in the honest-but-curious setting. Specifically, you are to show that under this assumption, the protocol semi-securely implements (see Section 23.5.6.2) the corresponding probabilistic instance of $\mathcal{F}_{\text{sfe}}$ (see Section 23.5.3.1), where

- There are no inputs.
- $\mathcal{F}_{\text{sfe}}$ generates a_1, b_1, a_2, b_2, c_2 in $\mathbb{Z}_q$ at random, and then computes $c_1 \leftarrow (a_1 + a_1)(b_1 + b_2) - c_2$.
- P_1 obtains the output (a_1, b_1, c_1) from $\mathcal{F}_{\text{sfe}}$,
- P_2 obtains the output (a_2, b_2, c_2) from $\mathcal{F}_{\text{sfe}}$.

To do this, you need to design a simulator $\mathcal{S}$.

(i) First suppose that P_1 is corrupt. Your simulator $\mathcal{S}$ obtains P_1's outputs a_1, b_1, c_1 from $\mathcal{F}_{\text{sfe}}$. Let $coins_1$ denote the random bits that are used by P_1 to generate $a_1, b_1 \in \mathbb{Z}_q$ (as in Remark 23.5). In the real world, P_1 generates $coins_1$ at random. Using the values a_1, b_2, your simulator $\mathcal{S}$ should cook up a simulated value of $coins_1$ (hint: see Exercise 3.11). Let $coins'_1$ denote the random bits that are used by P_1 to generate the objects $pk, sk, \varepsilon_a, \varepsilon_b$. In the real-world, P_1 generates $coins'_1$ at random. Your simulator $\mathcal{S}$ should do the same. In the real-word, P_1 would receive the protocol message ε from P_2. Using the values a_1, b_1, c_1, your simulator $\mathcal{S}$ should cook up a simulated protocol message ε. Show how $\mathcal{S}$ can do all this. Also argue that the environment (which knows all parties' inputs and outputs) cannot distinguish the simulation from the real world with non-negligible probability. For this, you do not need to make any security assumptions.

(ii) Second suppose that P_2 is corrupt. Your simulator $\mathcal{S}$ obtains P_2's outputs a_2, b_2, c_2 from $\mathcal{F}_{\text{sfe}}$. Let $coins_2$ denote the random bits that are used by P_2 to generate $a_2, b_2, c_2 \in \mathbb{Z}_q$ (as in Remark 23.5). In the real world, P_2 generates $coins_2$ at random. Using the values a_1, b_2, c_2, you simulator $\mathcal{S}$ should cook up a simulated value of $coins_2$ (hint: again see Exercise 3.11). Let $coins'_2$ be the random bits that are used by P_2 to then compute ε. In the real-world, P_2 generates $coins'_2$ at random. Your simulator $\mathcal{S}$ should do the same. When the real-world P_2 would receive the protocol message $(pk, \varepsilon_a, \varepsilon_b)$ from P_1, your simulator must cook up a simulation of the protocol message $(pk, \varepsilon_a, \varepsilon_b)$. Show how $\mathcal{S}$ can do this, and fill in the rest of the details of $\mathcal{S}$. Also argue that the environment (which knows all parties' inputs and outputs) cannot distinguish the simulation from the real world with non-negligible probability. For this, you will need to assume that the encryption scheme is semantically secure.

23.4 (A multiplication protocol from OT). Let's design a 2-party protocol in the honest-but-curious setting for the following problem: P_1 has an input $a \in \mathbb{Z}_q$ and P_2 has an input $b \in \mathbb{Z}_q$. At the end of the protocol, the parties should have a sharing (c_1, c_2) of $a \cdot b$. That is, P_1 should have c_1 and P_2 should have c_2 so that $a \cdot b = c_1 + c_2$. Moreover, P_1 should learn nothing about b, and P_2 should learn nothing about a. Here is a simple protocol using 1-out-of-2 oblivious transfer (OT):

- *Init:* Let $\alpha_0, \ldots, \alpha_n \in \{0, 1\}$ be the binary representation of a, so that $a = \sum_{i=0}^{n} \alpha_i 2^i$. P_2 chooses random $r_0, \ldots, r_n \overset{\text{R}}{\leftarrow} \mathbb{Z}_q$.
- *OT step:* P_1 and P_2 engage in $n + 1$ parallel 1-out-of-2 OTs, where in OT instance number i,
  - P_2 plays the role of sender with input $T_i = (r_i,\ r_i + b) \in \mathbb{Z}_q^2$, and

- P_1 plays the role of receiver with input $\alpha_i \in \{0,1\}$.

  When the OT protocols finish, P_1 has $t_0, \ldots, t_n \in \mathbb{Z}_q$ where $t_i = T_i[\alpha_i]$ for $i = 0, \ldots, n$.

- *Finalize:* P_1 computes $c_1 = \sum_{i=0}^{n} 2^i t_i$ and P_2 computes $c_2 = -\sum_{i=0}^{n} 2^i r_i$.

(a) Show that if P_1 and P_2 honestly follow the protocol, then $c_1 + c_2 = a \cdot b$.

Hint: Use the fact that $a \cdot b = \sum_{i=0}^{n} \alpha_i \cdot 2^i b$.

(b) Design a simulator to show that if the OT protocol is secure, then P_1 learns nothing about b, and P_2 learns nothing about a.

(c) In Section 23.7 we presented a protocol, called OT extension, to cheaply execute many 1-out-of-2 OTs. Show how to optimize this OT extension protocol to the setting above where the shift $b \in \mathbb{Z}_q$ is the same in all sender pairs.

23.5 (Generating Beaver triple sharings II). Continuing with Exercise 23.3, let's see a more efficient protocol for P_1 and P_2 to jointly generate a Beaver triple sharing, this time using 1-out-of-2 oblivious transfer (OT). Again, we restrict to the honest-but-curious setting.

- *Init:* P_1 chooses random $a_1, b_1 \xleftarrow{R} \mathbb{Z}_q$, and P_2 chooses random $a_2, b_2 \xleftarrow{R} \mathbb{Z}_q$.

  Their goal is to generate a random sharing $(c_1, c_2) \in \mathbb{Z}_q^2$, where P_1 has c_1 and P_2 has c_2, such that $c_1 + c_2 = (a_1 + a_2)(b_1 + b_2)$.

- *Multiply:* P_1 and P_2 run the protocol from Exercise 23.4 twice:
  - once to compute a sharing (d_1, d_2) where $a_1 \cdot b_2 = d_1 + d_2$, and
  - once to compute a sharing (e_1, e_2) where $a_2 \cdot b_1 = e_1 + e_2$.

- *Finalize:* P_1 computes $c_1 = a_1 b_1 + d_1 + e_1 \in \mathbb{Z}_q$ and P_2 computes $c_2 = a_2 b_2 + d_2 + e_2 \in \mathbb{Z}_q$.

(a) Show that if P_1 and P_2 honestly follow the protocol, then $c_1 + c_2 = (a_1 + a_2)(b_1 + b_2)$.

(b) Design a simulator to show that if the multiplication protocol from Exercise 23.4 is secure, then P_1 learns nothing about a_2, b_2, and P_2 learns nothing about a_1, b_1.

Discussion: The protocol in this exercise is far more efficient than the protocol in Exercise 23.3 thanks to the OT extension technique discussed in Section 23.7.

Part IV

Appendices

Appendix A

Basic number theory

A.1 Cyclic groups

Notation: for a finite cyclic group $\mathbb{G}$ we let $\mathbb{G}^*$ denote the set of generators of $\mathbb{G}$.

A.2 Arithmetic modulo primes

A.2.1 Basic concepts

We use the letters p and q to denote prime numbers. We will be using large primes, e.g. on the order of 300 digits (1024 bits).

1. For a prime $p > 2$ let $\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}$.
   Elements of $\mathbb{Z}_p$ can be added modulo p and multiplied modulo p. For $x, y \in \mathbb{Z}_p$ we write $x + y$ and $x \cdot y$ to denote the sum and product of x and y modulo p.

2. Fermat's theorem: $g^{p-1} = 1$ for all $0 \neq g \in \mathbb{Z}_p$
   Example: $3^4 = 81 \equiv 1 \pmod{5}$.

3. The *inverse* of $x \in \mathbb{Z}_p$ is an element $a \in \mathbb{Z}_p$ satisfying $a \cdot x = 1$ in $\mathbb{Z}_p$.
   The inverse of x in $\mathbb{Z}_p$ is denoted by x^{-1}.
   Example: 1. 3^{-1} in $\mathbb{Z}_5$ is 2 since $2 \cdot 3 \equiv 1 \pmod{5}$.
   2. 2^{-1} in $\mathbb{Z}_p$ is $\frac{p+1}{2}$.

4. All elements $x \in \mathbb{Z}_p$ except for $x = 0$ are invertible.
   Simple (but inefficient) inversion algorithm: $x^{-1} = x^{p-2}$ in $\mathbb{Z}_p$.
   Indeed, $x^{p-2} \cdot x = x^{p-1} = 1$ in $\mathbb{Z}_p$.

5. We denote by $\mathbb{Z}_p^*$ the set of invertible elements in $\mathbb{Z}_p$. Then $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$.

6. We now have an algorithm for solving linear equations in $\mathbb{Z}_p$: $a \cdot x = b$.
   Solution: $x = b \cdot a^{-1} = b \cdot a^{p-2}$.
   What about an algorithm for solving quadratic equations?

A.2.2 Structure of $\mathbb{Z}_p^*$

1. $\mathbb{Z}_p^*$ is a *cyclic group.*
   In other words, there exists $g \in \mathbb{Z}_p^*$ such that $\mathbb{Z}_p^* = \{1, g, g^2, g^3, \ldots, g^{p-2}\}$.
   Such a g is called a *generator* of $\mathbb{Z}_p^*$.
   Example: in $\mathbb{Z}_7^*$: $\langle 3 \rangle = \{1, 3, 3^2, 3^3, 3^4, 3^5\} \equiv \{1, 3, 2, 6, 4, 5\} \pmod 7 = \mathbb{Z}_7^*$.

2. Not every element of $\mathbb{Z}_p^*$ is a generator.
   Example: in $\mathbb{Z}_7^*$ we have $\langle 2 \rangle = \{1, 2, 4\} \neq \mathbb{Z}_7^*$.

3. The *order* of $g \in \mathbb{Z}_p^*$ is the smallest positive integer a such that $g^a = 1$.
   The order of $g \in \mathbb{Z}_p^*$ is denoted $\text{order}_p(g)$.
   Example: $\text{order}_7(3) = 6$ and $\text{order}_7(2) = 3$.

4. Lagrange's theorem: for all $g \in \mathbb{Z}_p^*$ we have that $\text{order}_p(g)$ divides $p-1$. Observe that Fermat's theorem is a simple corollary:
   $$\text{for } g \in \mathbb{Z}_p^* \text{ we have } g^{p-1} = (g^{\text{order}(g)})^{(p-1)/\text{order}(g)} = (1)^{(p-1)/\text{order}(g)} = 1.$$

5. If the factorization of $p-1$ is known then there is a simple and efficient algorithm to determine $\text{order}_p(g)$ for any $g \in \mathbb{Z}_p^*$.

A.2.3 Quadratic residues

1. The *square root* of $x \in \mathbb{Z}_p$ is a number $y \in \mathbb{Z}_p$ such that $y^2 = x \bmod p$.
   Example: 1. $\sqrt{2}$ in $\mathbb{Z}_7$ is 3 since $3^2 = 2 \bmod 7$.
   2. $\sqrt{3}$ in $\mathbb{Z}_7$ does not exist.

2. An element $x \in \mathbb{Z}_p^*$ is called a *Quadratic Residue* (QR for short) if it has a square root in $\mathbb{Z}_p$.

3. How many square roots does $x \in \mathbb{Z}_p$ have?
   If $x^2 = y^2$ in $\mathbb{Z}_p$ then $0 = x^2 - y^2 = (x-y)(x+y)$.
   $\mathbb{Z}_p$ is an "integral domain" which implies that $x - y = 0$ or $x + y = 0$, namely $x = \pm y$.
   Hence, elements in $\mathbb{Z}_p$ have either zero square roots or two square roots.
   If a is the square root of x then $-a$ is also a square root of x in $\mathbb{Z}_p$.

4. Euler's theorem: $x \in \mathbb{Z}_p$ is a QR if and only if $x^{(p-1)/2} = 1$.
   Example: $2^{(7-1)/2} = 1$ in $\mathbb{Z}_7$ but $3^{(7-1)/2} = -1$ in $\mathbb{Z}_7$.

5. Let $g \in \mathbb{Z}_p^*$. Then $a = g^{(p-1)/2}$ is a square root of 1. Indeed, $a^2 = g^{p-1} = 1$ in $\mathbb{Z}_p$.
   Square roots of 1 in $\mathbb{Z}_p$ are 1 and -1.
   Hence, for $g \in \mathbb{Z}_p^*$ we know that $g^{(p-1)/2}$ is 1 or -1.

6. Legendre symbol: for $x \in Z_p$ define $\left(\frac{x}{p}\right) := \begin{cases} 1 & \text{if } x \text{ is a QR in } \mathbb{Z}_p \\ -1 & \text{if } x \text{ is not a QR in } \mathbb{Z}_p \\ 0 & \text{if } x = 0 \bmod p \end{cases}$.

7. By Euler's theorem we know that $\left(\frac{x}{p}\right) = x^{(p-1)/2}$ in $\mathbb{Z}_p$.
   $\Longrightarrow$ the Legendre symbol can be efficiently computed.

8. Easy fact: let g be a generator of $\mathbb{Z}_p^*$. Let $x = g^r$ for some integer r.
Then x is a QR in $\mathbb{Z}_p$ if and only if r is even.
$\Longrightarrow$ **the Legendre symbol reveals the parity of** r.

9. Since $x = g^r$ is a QR if and only if r is even it follows that exactly half the elements of $\mathbb{Z}_p$ are QR's.

10. When $p = 3 \bmod 4$ computing square roots of $x \in \mathbb{Z}_p$ is easy.
Simply compute $a = x^{(p+1)/4}$ in $\mathbb{Z}_p$.
$a = \sqrt{x}$ since $a^2 = x^{(p+1)/2} = x \cdot x^{(p-1)/2} = x \cdot 1 = x$ in $\mathbb{Z}_p$.

11. When $p = 1 \bmod 4$ computing square roots in $\mathbb{Z}_p$ is possible but somewhat more complicated; it requires a randomized algorithm.

12. We now have an algorithm for solving quadratic equations in $\mathbb{Z}_p$.
We know that if a solution to $ax^2 + bx + c = 0 \bmod p$ exists then it is given by:
$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
in $\mathbb{Z}_p$. Hence, the equation has a solution in $\mathbb{Z}_p$ if and only if $\Delta = b^2 - 4ac$ is a QR in $\mathbb{Z}_p$. Using our algorithm for taking square roots in $\mathbb{Z}_p$ we can find $\sqrt{\Delta} \bmod p$ and recover x_1 and x_2.

13. What about cubic equations in $\mathbb{Z}_p$? There exists an efficient randomized algorithm that solves any equation of degree d in time polynomial in d.

A.2.4 Computing in $\mathbb{Z}_p$

1. Since p is a huge prime (e.g. 1024 bits long) it cannot be stored in a single register.

2. Elements of $\mathbb{Z}_p$ are stored in buckets where each bucket is 32 or 64 bits long depending on the processor's chip size.

3. Adding two elements $x, y \in \mathbb{Z}_p$ can be done in linear time in the *length* of p.

4. Multiplying two elements $x, y \in \mathbb{Z}_p$ can be done in quadratic time in the *length* of p. If p is n bits long, better algorithms work in time $O(n^{1.7})$ (rather than $O(n^2)$).

5. Inverting an element $x \in \mathbb{Z}_p$ can be done in quadratic time in the length of p.

6. Using the repeated squaring algorithm, $x^r \in \mathbb{Z}_p$ can be computed using at most $(2\log_2 r)$ multiplications in $\mathbb{Z}_p$.

7. When the base of the exponentiation x is fixed, one can pre-compute a table of powers of x containing $(2^w/w) \cdot \log_2 r$ group elements, and reduce the time to compute $x^r \in \mathbb{Z}_p$ to only $(\log_2 r)/w$ multiplications in $\mathbb{Z}_p$. Here w is a small constant, such as $w = 5$.

A.2.5 Summary: arithmetic modulo primes

Let p be a 1024 bit prime. Easy problems in $\mathbb{Z}_p$:

1. Generating a random element. Adding and multiplying elements.

2. Computing $g^r \bmod p$ is easy even if r is very large.

3. Inverting an element. Solving linear systems.

4. Testing if an element is a QR and computing its square root if it is a QR.

5. Solving polynomial equations of degree d can be done in polynomial time in d.

Problems that are believed to be hard in $\mathbb{Z}_p$:

1. Let g be a generator of $\mathbb{Z}_p^*$. Given $x \in \mathbb{Z}_p^*$ find an r such that $x = g^r \bmod p$. This is known as the *discrete log problem.*

2. Let g be a generator of $\mathbb{Z}_p^*$. Given $x, y \in \mathbb{Z}_p^*$ where $x = g^{r_1}$ and $y = g^{r_2}$. Find $z = g^{r_1 r_2}$. This is known as the *Diffie-Hellman problem.*

A.3 Arithmetic modulo composites

We are dealing with integers n on the order of 300 digits long, (1024 bits). Unless otherwise stated, we assume that n is the product of two equal size primes, e.g. on the order of 150 digits each (512 bits).

1. For a composite n let $\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$.
   Elements of $\mathbb{Z}_n$ can be added and multiplied modulo n.

2. The inverse of $x \in \mathbb{Z}_n$ is an element $y \in \mathbb{Z}_n$ such that $x \cdot y = 1 \bmod n$.
   An element $x \in \mathbb{Z}_n$ has an inverse if and only if x and n are relatively prime. In other words, $\gcd(x, n) = 1$.

3. Elements of $\mathbb{Z}_n$ can be efficiently inverted using Euclid's algorithm. If $\gcd(x, n) = 1$ then using Euclid's algorithm it is possible to efficiently construct two integers $a, b \in \mathbb{Z}$ such that $ax + bn = 1$. Reducing this relation modulo n leads to $ax = 1 \bmod n$. Hence $a = x^{-1} \bmod n$.
   note: this inversion algorithm also works in $\mathbb{Z}_p$ for a prime p and is more efficient than inverting x by computing $x^{p-2} \bmod p$.

4. We let $\mathbb{Z}_n^*$ denote the set of invertible elements in $\mathbb{Z}_n$.

5. We now have an algorithm for solving linear equations: $\quad a \cdot x = b \bmod n$.
   Solution: $\quad x = b \cdot a^{-1}$ where a^{-1} is computed using Euclid's algorithm.

6. How many elements are in $\mathbb{Z}_n^*$? We denote by $\varphi(n)$ the number of elements in $\mathbb{Z}_n^*$. We already know that $\varphi(p) = p - 1$ for a prime p.

7. One can show that if $n = p_1^{e_1} \cdots p_m^{e_m}$ then $\varphi(n) = n \cdot \prod_{i=1}^m \left(1 - \frac{1}{p_i}\right)$.
   In particular, when $n = pq$ we have that $\varphi(n) = (p-1)(q-1) = n - p - q + 1$.
   Example: $\varphi(15) = \big|\{1, 2, 4, 7, 8, 11, 13, 14\}\big| = 8 = 2 * 4$.

8. Euler's theorem: all $a \in \mathbb{Z}_n^*$ satisfy $a^{\varphi(n)} = 1$ in $\mathbb{Z}_n$.
   note: For primes p Euler's theorem implies that $a^{\varphi(p)} = a^{p-1} = 1$ for all $a \in \mathbb{Z}_p^*$. Hence, Euler's theorem is a generalization of Fermat's theorem.

Structure of $\mathbb{Z}_n$

Theorem A.1 (Chinese Remainder Theorem (CRT)). *state theorem*

Summary

Let n be a 1024 bit integer which is a product of two 512 bit primes. Easy problems in $\mathbb{Z}_n$:

1. Generating a random element. Adding and multiplying elements.

2. Computing $g^r \bmod n$ is easy even if r is very large.

3. Inverting an element. Solving linear systems.

Problems that are believed to be hard if the factorization of n is unknown, but become easy if the factorization of n is known:

1. Finding the prime factors of n.

2. Testing if an element is a QR in $\mathbb{Z}_n$.

3. Computing the square root of a QR in $\mathbb{Z}_n$. This is provably as hard as factoring n. When the factorization of $n = pq$ is known one computes the square root of $x \in \mathbb{Z}_n^*$ by first computing the square root in $\mathbb{Z}_p$ of $x \bmod p$ and the square root in $\mathbb{Z}_q$ of $x \bmod q$ and then using the CRT to obtain the square root of x in $\mathbb{Z}_n$.

4. Computing e'th roots modulo n when $\gcd(e, \varphi(n)) = 1$.

5. More generally, solving polynomial equations of degree $d > 1$. This problem is easy if the factorization of n is known: one first finds the roots of the polynomial equation modulo the prime factors of n and then uses the CRT to obtain the roots in $\mathbb{Z}_n$.

Problems that are believed to be hard in $\mathbb{Z}_n$:

1. Let g be a generator of $\mathbb{Z}_n^*$. Given $x \in \mathbb{Z}_n^*$ find an r such that $x = g^r \bmod n$. This is known as the *discrete log problem.*

2. Let g be a generator of $\mathbb{Z}_n^*$. Given $x, y \in \mathbb{Z}_n^*$ where $x = g^{r_1}$ and $y = g^{r_2}$. Find $z = g^{r_1 r_2}$. This is known as the *Diffie-Hellman problem.*

Appendix B

Basic probability theory

Includes a description of statistical distance.

B.1 The birthday Paradox

Theorem B.1. *Let $\mathcal{M}$ be a set of size n and let $X_1, \ldots, X_k$ be k independent random variables uniform in $\mathcal{M}$. Let C be the event that for some distinct $i, j \in \{1, \ldots, k\}$ we have that $X_i = X_j$. Then*

(i) $\quad \Pr[C] \geq 1 - e^{-k(k-1)/2n} \geq \min\left\{\frac{k(k-1)}{4n}, 0.63\right\}$, *and*

(ii) $\quad \Pr[C] \leq 1 - e^{-k(k-1)/n}$ *when* $k < n/2$.

Proof. These all follow easily from the inequality

$$1 - x \leq e^{-x} \leq 1 - x/2,$$

which holds for all $x \in [0, 1]$. □

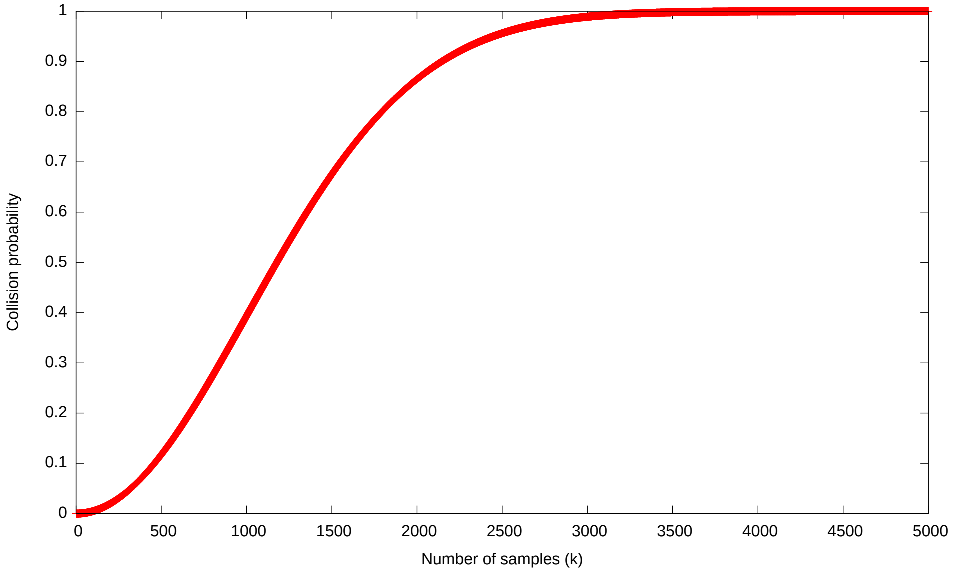
Most frequently we will use the lower bound to say that a collision happens with *at least* a certain probability. But occasionally we will use the upper bound to argue that collisions do not happen.

It is well documented that birthdays are not really uniform throughout the year. For example, in the U.S. the percentage of births in September is higher than in any other month. We show next that this non-uniformity only increases the probability of collision.

We present a stronger version of the birthday paradox that applies to independent random variables that are not necessarily uniform in $\mathcal{M}$. We do, however, require that all random variables are identically distributed. Such random variables are called i.i.d (independent and identically distributed). This version of the birthday paradox is due to Blom [Blom, D. (1973), "A birthday problem", American Mathematical Monthly, vol. 80, pp. 1141-1142].

Corollary B.2. *Let $\mathcal{M}$ be a set of size n and let $X_1, \ldots, X_k$ be k i.i.d random variables over $\mathcal{M}$ where $k \geq 2$. Let C be the event that for some distinct $i, j \in \{1, \ldots, k\}$ we have that $X_i = X_j$. Then*

$$\Pr[C] \geq 1 - e^{-k(k-1)/2n} \geq \min\left\{\frac{k(k-1)}{4n}, 0.63\right\}.$$



The graph shows that collision probability for $n = 10^6$ elements and k ranging from one sample to 5000 samples. It illustrates the threshold phenomenon around the square root. At the square root, $\sqrt{n} = 1000$, the collision probability is about 0.4. Already at $4\sqrt{n} = 4000$ the collision probability is almost 1. At $0.5\sqrt{n} = 500$ the collision probability is small.

Figure B.1: Birthday Paradox

Proof. Let X be a random variable distributed as X_1. Let $\mathcal{M} = \{a_1, \ldots, a_n\}$ and let $p_i = \Pr[X = a_i]$. Let I be the set of all k-tuples over $\mathcal{M}$ containing distinct elements. Then I contains $\binom{n}{k}k!$ tuples. Since the variables are independent we have that:

$$\Pr[\neg C] = \sum_{(b_1,\ldots,b_k)\in I} \Pr[X_1 = b_1 \wedge \ldots \wedge X_k = b_k] = \sum_{(b_1,\ldots,b_k)\in I} \prod_{j=1}^{k} p_{b_j} \tag{B.1}$$

We show that this sum is maximized when $p_1 = p_2 = \ldots = p_n = 1/n$. This will mean that the probability of collision is minimized when all the variables are uniform. The Corollary will then follow from Theorem B.1.

Suppose some p_i is not $1/n$, say $p_i < 1/n$. Since $\sum_{j=1}^{n} p_i = 1$ there must be another p_j such that $p_j > 1/n$. Let $\epsilon = \min((1/n) - p_i,\ p_j - 1/n)$ and note that $p_j - p_i > \epsilon$. We show that replacing p_i by $p_i + \epsilon$ and p_j by $p_j - \epsilon$ increases the value of the sum in (B.1). Clearly, the resulting $p_1, \ldots, p_n$ still sum to 1. Hence, the resulting $p_1, \ldots, p_n$ form a distribution over $\mathcal{M}$ in which there is one less value that is not $1/n$. Furthermore, the probability of no collision in this distribution is greater than in the unmodified distribution. Repeating this replacement process at most n times will show that the sum is maximized when all the p_i's are equal to $1/n$. Again, this means that the probability of not getting a collision is maximized when the variables are uniform.

Now, consider the sum in (B.1). There are four types of terms. First, there are terms that do not contain either p_i or p_j. These terms are unaffected by the change to p_i and p_j. Second, there are terms that contain exactly one of p_i or p_j. These terms pair up. For every k-tuple that contains i but not j there is a corresponding tuple that contains j but not i. Then the sum of the corresponding two terms in (B.1) looks like $A(p_i + \epsilon) + A(p_j - \epsilon)$ for some $A \in [0, 1]$. Since this equals $Ap_i + Ap_j$ the sum of these two terms is not affected by the change to p_i and p_j. Finally, there are terms in (B.1) that contain both p_i and p_j. These terms change by

$$B(p_i + \epsilon)(p_j - \epsilon) - Bp_ip_j = B[\epsilon(p_j - p_i) - \epsilon^2]$$

for some $B \in [0, 1]$. By definition of ϵ we know that $p_j - p_i > \epsilon$ and therefore $\epsilon(p_j - p_i) - \epsilon^2 > 0$. Hence, the sum with modified p_i and p_j is larger than the sum with the unmodified values.

Overall, we proved that the modification to p_i and p_j increases the value of the sum in (B.1), as required. This completes the proof of the Corollary. □

B.1.1 More collision bounds

Consider the sequence $x_i \leftarrow f(x_{i-1})$ for a random function $f : \mathcal{X} \to \mathcal{X}$. Analyze the cycle time of this walk (needed for Pollard). Now, consider the same sequence for a permutation $\pi : \mathcal{X} \to \mathcal{X}$. Analyze the cycle time (needed for analysis of SecurID identification).

B.1.2 A simple distinguisher

We describe a simple algorithm that distinguishes two distributions on strings in $\{0, 1\}^n$. Let $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$ be independent random variables taking values in $\{0, 1\}$. Then

$$X := (X_1, \ldots, X_n) \qquad \text{and} \qquad Y := (Y_1, \ldots, Y_n)$$

are elements of $\{0, 1\}^n$. Suppose, that for $i = 1, \ldots, n$ we have

$$\Pr[X_i = 1] = p \qquad \text{and} \qquad \Pr[Y_i = 1] = (1 + 2\epsilon) \cdot p$$

for some $p \in [0,1]$ and some small $\epsilon > 0$ so that $(1+2\epsilon) \cdot p \leq 1$. Then X and Y induce two distinct distributions on $\{0,1\}^n$.

We are given an n-bit string T and are told that it is either sampled according to the distribution X or the distribution Y, so that both p and ϵ are known to us. Our goal is to decide which distribution T was sampled from. Consider the following simple algorithm $\mathcal{A}$:

```
input: $T = (t_1, \ldots, t_n) \in \{0,1\}^n$
output: 1 if $T$ is sampled from $X$ and 0 otherwise
$s \leftarrow (1/n) \cdot \sum_{i=1}^{n} t_i$
if $s > p \cdot (1+\epsilon)$ output 0 else output 1
```

We are primarily interested in the quantity

$$\Delta := \big| \Pr[\mathcal{A}(T_x) = 1] - \Pr[\mathcal{A}(T_y) = 1] \big| \quad \in [0,1]$$

where $T_x \xleftarrow{\mathrm{R}} X$ and $T_y \xleftarrow{\mathrm{R}} Y$. This quantity captures how well $\mathcal{A}$ distinguishes the distributions X and Y. For a good distinguisher Δ will be close to 1. For a weak distinguisher Δ will be close to 0. The following theorem shows that when n is about $1/(p\epsilon^2)$ then Δ is about $1/2$.

Theorem B.3. *For all $p \in [0,1]$ and $\epsilon < 0.3$, if $n = 4\lceil 1/(p\epsilon^2) \rceil$ then $\Delta > 0.5$*

Proof. The proof follows directly from the Chernoff bound. When T is sampled from X the Chernoff bound implies that

$$\Pr[\mathcal{A}(T_x) = 1] = \Pr[s > p(1+\epsilon)] \leq e^{-n \cdot (p\epsilon^2/2)} \leq e^{-2} \leq 0.135$$

When T is sampled from Y then the Chernoff bound implies that

$$\Pr[\mathcal{A}(T_y) = 0] = \Pr[s < p(1+\epsilon)] \leq e^{-n \cdot (p\epsilon^2/4)} \leq e^{-1} \leq 0.368$$

Hence, $\Delta > |(1 - 0.368) - 0.135| = 0.503$ and the bound follows. □

Appendix C

Basic complexity theory

To be written.

Appendix D

Probabilistic algorithms

To be written.

Bibliography

[1] M. Abboud and T. Prest. Cryptographic divergences: New techniques and new applications. Cryptology ePrint Archive, Report 2020/815, 2020. `https://eprint.iacr.org/2020/815`.

[2] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Z. Béguelin, and P. Zimmermann. Imperfect forward secrecy: how diffie-hellman fails in practice. *Commun. ACM*, 62(1):106–114, 2019.

[3] M. R. Albrecht, K. G. Paterson, and G. J. Watson. Plaintext recovery attacks against SSH. In *30th IEEE Symposium on Security and Privacy*, pages 16–26, 2009.

[4] N. AlFardan, D. Bernstein, K. Paterson, B. Poettering, and J. Schuldt. On the security of RC4 in TLS. In *Proceedings of the 22th USENIX Security Symposium*, pages 305–320, 2013.

[5] N. J. AlFardan and K. G. Paterson. Lucky thirteen: Breaking the TLS and DTLS record protocols. In *2013 IEEE Symposium on Security and Privacy*, pages 526–540, 2013.

[6] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro. Scrypt is maximally memory-hard. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–62. Springer, 2017.

[7] P. Ananth and V. Vaikuntanathan. Optimal bounded-collusion secure functional encryption. *IACR Cryptology ePrint Archive*, 2019:314, 2019.

[8] A. Ash, R. Gross, and R. Gross. *Elliptic tales: curves, counting, and number theory.* Princeton University Press, 2012.

[9] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS with SSLv2. In *25th USENIX Security Symposium*, Aug. 2016.

[10] P. S. L. M. Barreto, B. David, R. Dowsley, K. Morozov, and A. C. A. Nascimento. A framework for efficient adaptively secure composable oblivious transfer in the rom. Cryptology ePrint Archive, Report 2017/993, 2017. `https://eprint.iacr.org/2017/993`.

[11] I. Bashmakova. *Diophantus and Diophantine equations.* Number 20 in Dolciani Mathematical Expositions. The Mathematical Association of America, 1997.

[12] M. Bellare and T. Kohno. A theoretical treatment of related-key attacks: RKA-PRPs, RKA-PRFs, and applications. In *Proceedings of Eurocrypt '03*, volume 2656 of *LNCS*. Springer-Verlag, 2003.

[13] M. Bellare, T. Ristenpart, and S. Tessaro. Multi-instance security and its application to password-based cryptography. In *CRYPTO 2012*, pages 312–329. Springer, 2012.

[14] M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In *Proceedings of Crypto '97*, volume 1294 of *LNCS*. Springer-Verlag, 1997.

[15] S. M. Bellovin. Frank miller: Inventor of the one-time pad. *Cryptologia*, 35(3):203–222, 2011.

[16] K. Bentahar, P. Farshim, J. Malone-Lee, and N. P. Smart. Generic constructions of identity-based and certificateless kems. *J. Cryptology*, 21(2):178–199, 2008.

[17] D. Bernstein and T. Lange. Montgomery curves and the montgomery ladder. In J. Bos and A. Lenstra, editors, *Topics in Computational Number Theory Inspired by Peter L. Montgomery*. Cambridge University Press, 2017. Available at `eprint.iacr.org/2017/293`.

[18] D. J. Bernstein. Curve25519: new Diffie-Hellman speed records. In *International Workshop on Public Key Cryptography*, pages 207–228. Springer, 2006.

[19] K. Bhargavan and G. Leurent. On the practical (in-)security of 64-bit block ciphers: Collision attacks on HTTP over TLS and openvpn. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 456–467, 2016.

[20] E. Biham and R. Anderson. Tiger: a fast new hash function. In *Proceedings of Fast Software Encryption (FSE) '96*, volume 1039 of *LNCS*. Springer-Verlag, 1996.

[21] A. Biryukov and D. Khovratovich. Related-key cryptanalysis of the full AES-192 and AES-256. In *ASIACRYPT 2009*, pages 1–18. 2009.

[22] A. Bishop, A. Jain, and L. Kowalczyk. Function-hiding inner product encryption. In *ASIACRYPT 2015*, pages 470–491, 2015.

[23] J. Black and P. Rogaway. A block-cipher mode of operation for parallelizable message authentication. In *Proceedings of Eurocrypt '02*, volume 2332 of *LNCS*. Springer-Verlag, 2002.

[24] J. Black, P. Rogaway, and T. Shrimpton. Black-box analysis of the block-cipher-based hash-function constructions from pgv. In *Proceedings of Crypto '02*, volume 2442 of *LNCS*. Springer-Verlag, 2002.

[25] K. K. Bodo Möller, Thai Duong. This poodle bites: Exploiting the ssl 3.0 fallback. Google security advisory, 2014.

[26] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full AES. In *ASIACRYPT 2011*, pages 344–371. 2011.

[27] A. Boldyreva, J. P. Degabriele, K. G. Paterson, and M. Stam. Security of symmetric encryption in the presence of ciphertext fragmentation. In *Proceedings of Eurocrypt'12*, pages 682–699, 2012.

[28] D. Boneh. Simplified OAEP for the RSA and rabin functions. In *CRYPTO 2001*, pages 275–291, 2001.

[29] D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008. Conference version in Eurocrypt 2004.

[30] D. Boneh, X. Boyen, and S. Halevi. Chosen ciphertext secure public key threshold encryption without random oracles. In *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, pages 226–243, 2006.

[31] D. Boneh, B. Bünz, and B. Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In *CRYPTO 2019*, pages 561–586, 2019.

[32] D. Boneh, R. Canetti, S. Halevi, and J. Katz. Chosen-ciphertext security from identity-based encryption. *SIAM J. Comput.*, 36(5):1301–1328, 2007.

[33] D. Boneh, M. Drijvers, and G. Neven. Compact multi-signatures for smaller blockchains. In *ASIACRYPT 2018*, volume 11273 of *Lecture Notes in Computer Science*, pages 435–464. Springer, 2018.

[34] D. Boneh, C. Gentry, S. Gorbunov, S. Halevi, V. Nikolaenko, G. Segev, V. Vaikuntanathan, and D. Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT 2014*, pages 533–556, 2014.

[35] D. Boneh and M. Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. *Algorithmica*, 79(4):1233–1285, 2017. Conference version in Crypto 2014.

[36] J. Bonneau and I. Mironov. Cache-collision timing attacks against AES. In *in Proc. Cryptographic Hardware and Embedded Systems (CHES) 2006. Lecture Notes in Computer Science*, pages 201–215. Springer, 2006.

[37] W. Bosma and P. Stevenhagen. On the computation of quadratic 2-class groups. *Journal de Theorie des Nombres*, 1996.

[38] Z. Brakerski and G. Segev. Function-private functional encryption in the private-key setting. *J. Cryptology*, 31(1):202–225, 2018.

[39] J. A. Buchmann, E. Dahmen, and M. Schneider. Merkle tree traversal revisited. *PQCrypto*, 8:63–78, 2008.

[40] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, July 2004.

[41] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *31st IEEE Symposium on Security and Privacy, S&P 2010, 16-19 May 2010, Berleley/Oakland, California, USA*, pages 191–206, 2010.

[42] J. H. Cheon. Security analysis of the strong diffie-hellman problem. In *EUROCRYPT 2006*, pages 1–11, 2006.

[43] H. Cohen. *A course in computational algebraic number theory.* Graduate texts in mathematics. Springer, 2010.

[44] D. Coppersmith. The data encryption standard and its strength against attack. *IBM Journal of Research and Development*, 38(3), 1994.

[45] D. Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In *EUROCRYPT 1996*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 1996.

[46] J. Coron. Security proof for partial-domain hash signature schemes. In *CRYPTO 2002*, pages 613–626, 2002.

[47] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016:086, 2016.

[48] N. T. Courtois, P. Emirdag, and F. Valsorda. Private key recovery combination attacks: On extreme fragility of popular bitcoin key management, wallet and cold storage solutions in presence of poor rng events. Cryptology ePrint Archive, Report 2014/848, 2014. `http://eprint.iacr.org/2014/848`.

[49] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard.* Springer, 2002.

[50] S. Data. Announcing our worst passwords of 2016, 2017.

[51] Y. Desmedt and A. Oldyzko. A chosen text attack on the rsa cryptosystem and some discrete logarithm schemes. In *Proceedings of Crypto '85*, volume 218 of *LNCS*, pages 516–521. Springer-Verlag, 1985.

[52] T. Dierks and C. Allen. The TLS protocol version 1.0. Internet RFC 2246, 1999.

[53] I. Dinur, O. Dunkelman, N. Keller, and A. Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In *CRYPTO 2012*, pages 719–740. 2012.

[54] Y. Dodis, S. Guo, and J. Katz. Fixing cracks in the concrete: Random oracles with auxiliary input, revisited. In *EUROCRYPT 2017*, pages 473–495, 2017.

[55] Y. Dodis, E. Kiltz, K. Pietrzak, and D. Wichs. Message authentication, revisited. In *EUROCRYPT 2012*, volume 7237, pages 355–374. Springer, 2012.

[56] D. Dolev, C. Dwork, and M. Naor. Nonmalleable cryptography. *SIAM J. Comput.*, 30(2):391–437, 2000.

[57] O. Dunkelman and N. Keller. The effects of the omission of last round's mixcolumns on AES. *Inf. Process. Lett.*, 110(8-9):304–308, 2010.

[58] T. Duong and J. Rizzo. Here come the $\oplus$ ninjas, 2011. See also `en.wikipedia.org/wiki/Transport_Layer_Security#BEAST_attack`.

[59] F. B. Durak, T. M. DuBuisson, and D. Cash. What else is revealed by order-revealing encryption? In *Proceedings of ACM CCS*, pages 1155–1166, 2016.

[60] Ethereum 2.0 specifications, 2019. `https://github.com/ethereum/eth2.0-specs`.

[61] A. Fiat and M. Naor. Rigorous time/space tradeoffs for inverting functions. In *STOC'91*, pages 534–541. ACM, 1991.

[62] J. Fischer and J. Stern. An efficient pseudo-random generator provably as secure as syndrome decoding. In *EUROCRYPT '96*, pages 245–255, 1996.

[63] S. Fluhrer, I. Mantin, and A. Shamir. Weaknesses in the key scheduling algorithm of RC4. In *proceedings of selected areas of cryptography (SAC)*, pages 1–24, 2001.

[64] S. Fluhrer and D. McGrew. Statistical analysis of the alleged RC4 keystream generator. In *Proceedings of FSE 2000*, volume 1978 of *LNCS*. Springer-Verlag, 2000.

[65] A. M. Frieze, R. Kannan, and J. C. Lagarias. Linear congruential generators do not produce random sequences. In *FOCS*, pages 480–484, 1984.

[66] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. *J. Cryptology*, 17(2):81–104, 2004.

[67] S. Galbraith. *Mathematics of Public Key Cryptography.* Cambridge University Press, 2012.

[68] S. Garfinkel and A. Shelat. Remembrance of data passed: A study of disk sanitization practices. *IEEE Security and Privacy*, January 2003.

[69] C. Garman, M. Green, G. Kaptchuk, I. Miers, and M. Rushanan. Dancing on the lip of the volcano: Chosen ciphertext attacks against apple imessage. In *USENIX Security Symposium*, 2017.

[70] P. Gaudry. Index calculus for abelian varieties of small dimension and the elliptic curve discrete logarithm problem. *J. Symb. Comput.*, 44(12):1690–1702, 2009.

[71] D. Genkin, A. Shamir, and E. Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *CRYPTO 2014*, pages 444–461, 2014.

[72] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. The most dangerous code in the world: validating SSL certificates in non-browser software. In *the ACM Conference on Computer and Communications Security, CCS'12*, pages 38–49, 2012.

[73] V. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In *Proceedings of fast software encryption (FSE) '01*, volume 2355 of *LNCS*, pages 92–108. Springer-Verlag, 2001.

[74] S. Goldwasser, S. D. Gordon, V. Goyal, A. Jain, J. Katz, F. Liu, A. Sahai, E. Shi, and H. Zhou. Multi-input functional encryption. In *EUROCRYPT 2014*, pages 578–602, 2014.

[75] A. Golynski. Cell probe lower bounds for succinct data structures. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 625–634. Society for Industrial and Applied Mathematics, 2009.

[76] M. T. Goodrich, C. Papamanthou, and R. Tamassia. On the cost of persistence and authentication in skip lists. In *Experimental Algorithms, 6th International Workshop, WEA 2007, Rome, Italy, June 6-8, 2007, Proceedings*, pages 94–107, 2007.

[77] M. T. Goodrich, J. Z. Sun, and R. Tamassia. Efficient tree-based revocation in groups of low-state devices. In *CRYPTO 2004*, pages 511–527, 2004.

[78] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security, CCS 2006, Alexandria, VA, USA, Ioctober 30 - November 3, 2006*, pages 89–98, 2006.

[79] M. Green and S. Hohenberger. Universally composable adaptive oblivious transfer. In *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 179–197. Springer, 2008.

[80] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[81] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp. Cryptanalysis with copacobana. *Computers, IEEE Transactions on*, 57(11):1498–1513, 2008.

[82] S. Halevi and H. Krawczyk. Strengthening digital signatures via randomized hashing. In *CRYPTO 2006*, pages 41–59, 2006.

[83] D. Halevy and A. Shamir. The LSD broadcast encryption scheme. In *CRYPTO 2002*, pages 47–60, 2002.

[84] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*. Clarendon Press, 1979.

[85] J. Haynes and H. Klehr. *Venona: Decoding Soviet Espionage in America*. Yale University Press, 1999.

[86] N. Heninger, Z. Durumeric, E. Wustrow, and J. A. Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In T. Kohno, editor, *USENIX Security*, pages 205–220. USENIX Association, 2012.

[87] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *CRYPTO 1995*, pages 339–352, 1995.

[88] J. Hoffstein, J. Pipher, and J. Silverman. NTRU: A ring-based public key cryptosystem. In *Algorithmic Number Theory (ANTS)*, pages 267–288, 1998.

[89] S. Hohenberger, A. B. Lewko, and B. Waters. Detecting dangerous queries: A new approach for chosen ciphertext security. In *Eurocrypt 2012*, pages 663–681, 2012.

[90] N. Howgrave-Graham. Finding small roots of univariate modular equations revisited. In *Cryptography and Coding*, volume 1355 of *Lecture Notes in Computer Science*, pages 131–142. Springer, 1997.

[91] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *30th Annual Symposium on Foundations of Computer Science*, pages 236–241, 1989.

[92] R. Impagliazzo and S. Rudich. Limits on the provable consequences of one-way permutations. In *Proceedings of the Symposium on Theory of Computing (STOC)*, pages 44–61, 1989.

[93] T. Iwata and K. Kurosawa. OMAC: One-key CBC MAC. In *Proceedings of fast software encryption (FSE) '03*, volume 2887 of *LNCS*, pages 129–153. Springer-Verlag, 2003.

[94] J. Jaeger and S. Tessaro. Expected-time cryptography: Generic techniques and applications to concrete soundness. In *TCC 2020*, volume 12552 of *Lecture Notes in Computer Science*, pages 414–443. Springer, 2020.

[95] J. Jonsson and B. Kaliski. On the security of RSA encryption in TLS. In *CRYPTO 2002*, pages 127–142, 2002.

[96] M. Joye and M. Tunstall, editors. *Fault Analysis in Cryptography.* Information Security and Cryptography. Springer, 2012.

[97] P. Junod. On the complexity of matsui's attack. In *Selected Areas in Cryptography (SAC)*, pages 199–211, 2001.

[98] J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *J. Cryptology*, 26(2):191–224, 2013.

[99] P. C. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.

[100] D. Kogan, N. Manohar, and D. Boneh. T/key: Second-factor authentication from secure hash chains. In *the ACM Conference on Computer and Communications Security, CCS'17*, 2017.

[101] A. Langley. Maintaining digital certificate security, 2014. `security.googleblog.com/2014/07/maintaining-digital-certificate-security.html`.

[102] J. Len, P. Grubbs, and T. Ristenpart. Partitioning oracle attacks. *IACR Cryptol. ePrint Arch.*, 2020:1491, 2020.

[103] A. Lenstra. *Key lengths.* Wiley, 2005.

[104] A. K. Lenstra, J. P. Hughes, M. Augier, J. W. Bos, T. Kleinjung, and C. Wachter. Public keys. In *CRYPTO 2012*, volume 7417, pages 626–642. Springer, 2012.

[105] S. Lucks. Attacking triple encryption. In *Proceedings of Fast Software Encryption 1998*, LNCS, pages 239–253, 1998.

[106] J. Manger. A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS #1 v2.0. In *CRYPTO 2001*, pages 230–238, 2001.

[107] I. Mantin and A. Shamir. A practical attack on broadcast RC4. In *Proceedings of FSE 2001.* Springer-Verlag, 2001.

[108] M. Matsui. The first experimental cryptanalysis of the data encryption standard. In *Proceedings of Crypto'94*, pages 1–11, 1994.

[109] M. Matsui. Linear cryptanalysis method for des cipher. In *Proceedings of Eurocrypt'93*, pages 386–397, 1994.

[110] U. Maurer and S. Wolf. The relationship between breaking the diffie-hellman protocol and computing discrete logarithms. *SIAM J. Comput.*, 28(5):1689–1721, 1999.

[111] R. McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN Progress Report*, 44:114–116, 1978.

[112] R. Merkle and M. Hellman. On the security of multiple encryption. *Communications of the ACM*, 24:465–467, 1981.

[113] Erroneous verisign-issued digital certificates pose spoofing hazard. Microsoft Security Bulletin MS01-017, 2001.

[114] I. Mironov. Hash functions: From merkle-damgard to shoup. In *Proceedings of Eurocrypt '01*, volume 2045 of *LNCS*, pages 166–181. Springer-Verlag, 2001.

[115] F. Monrose, M. K. Reiter, and S. Wetzel. Password hardening based on keystroke dynamics. *International Journal of Information Security*, 1(2):69–83, 2002.

[116] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. In *TCC*, pages 1–18, 2009.

[117] S. Murdoch. Hot or not: Revealing hidden services by their clock skew. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 27–36, 2006.

[118] S. Myers and A. Shelat. Bit encryption is complete. In *FOCS 2009*, pages 607–616, 2009.

[119] C. Namprempre, P. Rogaway, and T. Shrimpton. Reconsidering generic composition. In *EUROCRYPT 2014*, pages 257–274, 2014.

[120] M. Nandi and T. Pandit. Generic conversions from CPA to CCA secure functional encryption. *IACR Cryptology ePrint Archive*, 2015:457, 2015.

[121] D. Naor, M. Naor, and J. Lotspiech. Revocation and tracing schemes for stateless receivers. In *CRYPTO 2001*, pages 41–62, 2001.

[122] M. Naor. Bit commitment using pseudo-randomness. In *CRYPTO 1989*, pages 128–136, 1989.

[123] R. Napier. RNCryptor HMAC Vulnerability, 2013. `http://robnapier.net/rncryptor-hmac-vulnerability`.

[124] M. Nemec, M. Sýs, P. Svenda, D. Klinec, and V. Matyas. The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS'17*, pages 1631–1648. ACM, 2017.

[125] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2011.

[126] K. Nissim and M. Naor. Certificate revocation and certificate update. In *Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, USA, January 26-29, 1998*, 1998.

[127] Nist recommendation for key management part 1: General, 2005.

[128] Recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality, 2004.

[129] P. Patton. The bucklands boys and other tales of the atm. Wired magazine, 1993. `www.wired.com/1993/05/atm-2`.

[130] H. Poincaré. Sur les propriétés arithmétiques des courbes algébriques. *Journal de mathématiques pures et appliquées*, 7:161–234, 1901.

[131] A. Prado, N. Harris, and Y. Gluck. SSL, gone in 30 seconds: a BREACH beyond CRIME, 2013.

[132] B. Preneel, R. Govaerts, and J. Vandewalle. Hash functions based on block ciphers: a synthetic approach. In *Proceedings of Crypto '93*, volume 773 of *LNCS*. Springer-Verlag, 1993.

[133] T. Prest. Sharper bounds in lattice-based cryptography using the rényi divergence. In *ASIACRYPT 2017*, volume 10624 of *LNCS*, pages 347–374, 2017.

[134] R. Rivest. The MD4 message digest algorithm. In *Proceedings of Crypto '90*, volume 537 of *LNCS*. Springer-Verlag, 1990.

[135] R. Rivest. The MD5 message digest algorithm. Internet RFC 1321, 1992.

[136] J. Rizzo and T. Duong. The CRIME attack. Presentation at Ekoparty 2012, 2012.

[137] L. Roy. SoftSpokenOT: Quieter OT extension from small-field silent VOLE in the minicrypt model. In *CRYPTO '22*, volume 13507 of *Lecture Notes in Computer Science*, pages 657–687. Springer, 2022.

[138] M. Sabt and J. Traoré. Breaking into the keystore: A practical forgery attack against android keystore. Cryptology ePrint Archive, Report 2016/677, 2016. `http://eprint.iacr.org/2016/677`.

[139] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 475–484, 2014.

[140] B. Schneier and Mudge. Cryptanalysis of microsoft's point-to-point tunneling protocol (PPTP). In *Proceedings of the 5th ACM Conference on Communications and Computer Security*, 1998.

[141] R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of computation*, 44(170):483–494, 1985.

[142] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-ole: Improved constructions and implementation. In *ACM CCS*, pages 1055–1072. ACM, 2019.

[143] M. Scott. On the efficient implementation of pairing-based protocols. In *IMA International Conference on Cryptography and Coding*, pages 296–308. Springer, 2011.

[144] M. Scott. Pairing implementation revisited. Cryptology ePrint Archive, Report 2019/077, 2019. `https://eprint.iacr.org/2019/077`.

[145] V. Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT 1997*, pages 256–266, 1997.

[146] V. Shoup. A composition theorem for universal one-way hash functions. In *Proceedings of Eurocrypt '00*, volume 1807 of *LNCS*, pages 445–452. Springer-Verlag, 2000.

[147] V. Shoup. OAEP reconsidered. *J. Cryptology*, 15(4):223–249, 2002.

[148] V. Shoup. *A computational introduction to number theory and algebra.* Cambridge University Press, 2008. 2nd edition.

[149] D. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *10th USENIX Security Symposium*, 2001.

[150] F.-X. Standaert, G. Piret, and J.-J. Quisquater. Cryptanalysis of block ciphers: A survey. *UCL Crypto Group*, 2003.

[151] A. Stubblefield, J. Ioannidis, and A. Rubin. A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP). *ACM Transactions on Information. Systems Security*, 7(2):319–332, 2004.

[152] M. Szydlo. Merkle tree traversal in log space and time. In *Eurocrypt*, volume 3027, pages 541–554. Springer, 2004.

[153] B. Vallée. Gauss' algorithm revisited. *J. Algorithms*, 12(4):556–572, 1991.

[154] P. C. van Oorschot and M. J. Wiener. Parallel collision search with application to hash functions and discrete logarithms. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 210–218, 1994.

[155] X. Wang, A. Yao, and F. Yao. New collision search for SHA-1. Rump Session Crypto'05, 2005.

[156] A.-C. Yao. Coherent functions and program checkers. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 84–94. ACM, 1990.