A Graduate Course in Applied Cryptography

Dan Boneh Victor Shoup

August 17, 2015

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. The first part 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. The second part develops the concepts of *public-key encryption* and *digital signatures*, which allows Alice and Bob to do the same, but without having a shared, secret key. The third part is about *cryptographic protocols*, such as protocols for user identification, key exchange, 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 discuss topics not covered in the chapter.

Status of the book

The current draft only contains part I. Parts II and III are forthcoming. We hope you enjoy this write-up. Please send us comments and let us know if you find typos or mistakes.

Citations: While the current draft is mostly complete, we still do not include 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
August, 2015

Contents

Part I

Secret key cryptography

Chapter 2

Encryption

Roughly speaking, encryption is the problem of how two parties can communicate in secret in the presence of an eavesdropper. The main goals of this chapter are to develop a meaningful and useful definition of what we are trying to achieve, and to take some first steps in actually achieving it.

2.1 Introduction

Suppose Alice and Bob share a secret key k, and 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 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 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 Chapters 20 and 21.

2.2 Shannon ciphers and perfect security

2.2.1 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 of 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:

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

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:

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

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:

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

Here, $k[0 \,..\, \ell - 1]$ 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, $\sqcup$). 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

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

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:

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

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

2.2.2 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. This, 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 is key is drawn uniformly from the key space. We write $\mathbf{k}$ to denote the random variable representing this random key. For a message m, $E(\mathbf{k}, m)$ 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 $E(\mathbf{k}, m)$.

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 $\mathbf{k}$ is uniformly distributed over $\mathcal{K}$. If for all $m_0, m_1 \in \mathcal{M}$, and all $c \in \mathcal{C}$, we have*

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

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 N (possibly depending on c) such that for all $m \in \mathcal{M}$, we have

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

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

The proof of this is a simple calculation that we leave to the reader.

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 $N = 1$ 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 and c is an arbitrary length 1 string, and that $\mathbf{k}$ is a random variable that is uniformly distributed over the key space. Then we have

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

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 terrible 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 $\mathbf{k}$ is uniformly distributed over the key space, then the distributions of $E(\mathbf{k}, m_0)$ and $E(\mathbf{k}, m_1)$ 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 verity that this is perfectly secure. Indeed, it satisfies condition (ii) in Theorem 2.1 (with $N = 1$ 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 $\mathbf{k}$ 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*

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

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

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

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

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

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

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

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

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

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

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

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

The restriction that $\mathbf{m}$ and $\mathbf{k}$ 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.

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 $\mathbf{k}$ and $\mathbf{m}$ are random variables, such that*

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

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

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

Proof. We define $\mathcal{M}^*$ to be the set of messages that occur with nonzero probability.

We begin with a simple observation. Consider any fixed $m \in \mathcal{M}^*$ and $c \in \mathcal{C}$. Then we have

$$\Pr[\mathbf{c} = c \mid \mathbf{m} = m] = \Pr[E(\mathbf{k}, m) = c \mid \mathbf{m} = m],$$

and since $\mathbf{k}$ and $\mathbf{m}$ are independent, so are $E(\mathbf{k}, m)$ and $\mathbf{m}$, and hence

$$\Pr[E(\mathbf{k}, m) = c \mid \mathbf{m} = m] = \Pr[E(\mathbf{k}, m) = c].$$

Putting this all together, we have:

$$\Pr[\mathbf{c} = c \mid \mathbf{m} = m] = \Pr[E(\mathbf{k}, m) = c]. \tag{2.1}$$

We now prove the first implication. So assume that $\mathcal{E}$ is perfectly secure. We want to show that $\mathbf{c}$ and $\mathbf{m}$ are independent. To to this, let $m \in \mathcal{M}^*$ and $c \in \mathcal{C}$ be given. It will suffice to show that

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

We have

$$\begin{aligned}
\Pr[\mathbf{c} = c] &= \sum_{m' \in \mathcal{M}^*} \Pr[\mathbf{c} = c \mid \mathbf{m} = m'] \Pr[\mathbf{m} = m'] \qquad \text{(by total probability)} \\
&= \sum_{m' \in \mathcal{M}^*} \Pr[E(\mathbf{k}, m') = c] \Pr[\mathbf{m} = m'] \qquad \text{(by (2.1))} \\
&= \sum_{m' \in \mathcal{M}} \Pr[E(\mathbf{k}, m) = c] \Pr[\mathbf{m} = m'] \qquad \text{(by the definition of perfect security)} \\
&= \Pr[E(\mathbf{k}, m) = c] \sum_{m' \in \mathcal{M}^*} \Pr[\mathbf{m} = m'] \\
&= \Pr[E(\mathbf{k}, m) = c] \qquad \text{(probabilities sum to 1)} \\
&= \Pr[\mathbf{c} = c \mid \mathbf{m} = m] \qquad \text{(again by (2.1))}
\end{aligned}$$

This shows that $\mathbf{c}$ and $\mathbf{m}$ are independent.

That proves the first implication. For the second, we assume that $\mathbf{c}$ and $\mathbf{m}$ are independent, and moreover, that every message occurs with nonzero probability (so $\mathcal{M}^* = \mathcal{M}$). We want to show that $\mathcal{E}$ is perfectly secure, which means that for each $m_0, m_1 \in \mathcal{M}$, and each $c \in \mathcal{C}$, we have

$$\Pr[E(\mathbf{k}, m_0) = c] = \Pr[E(\mathbf{k}, m_1) = c]. \tag{2.2}$$

But we have

$$\begin{aligned}
\Pr[E(\mathbf{k}, m_0) = c] &= \Pr[\mathbf{c} = c \mid \mathbf{m} = m_0] \qquad \text{(by (2.1))} \\
&= \Pr[\mathbf{c} = c] \qquad \text{(by independence of } \mathbf{c} \text{ and } \mathbf{m}) \\
&= \Pr[\mathbf{c} = c \mid \mathbf{m} = m_1] \qquad \text{(again by independence of } \mathbf{c} \text{ and } \mathbf{m}) \\
&= \Pr[E(\mathbf{k}, m_1) = c] \qquad \text{(again by (2.1))}.
\end{aligned}$$

That shows that $\mathcal{E}$ is perfectly secure. □

2.2.3 The bad news

We have saved the bad new 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

$$\Pr[E(\mathbf{k}, m_0) = c] > 0, \quad \text{and} \tag{2.3}$$

$$\Pr[E(\mathbf{k}, m_1) = c] = 0. \tag{2.4}$$

Here, $\mathbf{k}$ 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 (2.3) 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 (2.4), 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 (2.4), and the theorem follows. □

2.3 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 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.

2.3.1 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 (see Chapter D). This means that for fixed inputs k and m, the output of $E(k, m)$ may be one of many values. To emphasize the probabilistic nature of this computation, we write

$$c \xleftarrow{R} E(k, m)$$

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 text whenever we use probabilistic algorithms. Similarly, we write

$$k \xleftarrow{R} \mathcal{K}$$

to denote the process of assigning to the program variable k a random, uniformly distributed element of 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

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

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). □

2.3.2 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

$$\Pr[\phi(E(\mathbf{k}, m_0))] = \Pr[\phi(E(\mathbf{k}, m_1))], \tag{2.5}$$

where $\mathbf{k}$ 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,

$$\Big|\Pr[\phi(E(\mathbf{k}, m_0))] - \Pr[\phi(E(\mathbf{k}, m_1))]\Big| \leq \epsilon, \tag{2.6}$$

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 (2.6) holds for every possible ϕ, m_0, and m_1, we only require that (2.6) 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, (2.6) 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: the **challenger** and an **adversary**. As we will see, the challenger follows a very simple, fixed protocol. However, an adversary $\mathcal{A}$ may follow an arbitrary (but still efficient) protocol. The challenger and the adversary $\mathcal{A}$ send messages back and forth to each other, as specified by their protocols, and at the end of the game, $\mathcal{A}$ outputs some value. Actually, our attack game for defining semantic security comprises two alternative "sub-games," or "experiments" — in both experiments, the adversary follows the same protocol; however, the challenger's behavior is slightly different in the two experiments. The attack game also defines a probability space, and this in turn defines the adversary's *advantage*, which measures the difference between the probabilities of two events in this probability space.

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 $k \xleftarrow{R} \mathcal{K}$, $c \xleftarrow{R} E(k, m_b)$, and sends c to the adversary.
- The adversary outputs a bit $\hat{b} \in \{0, 1\}$.

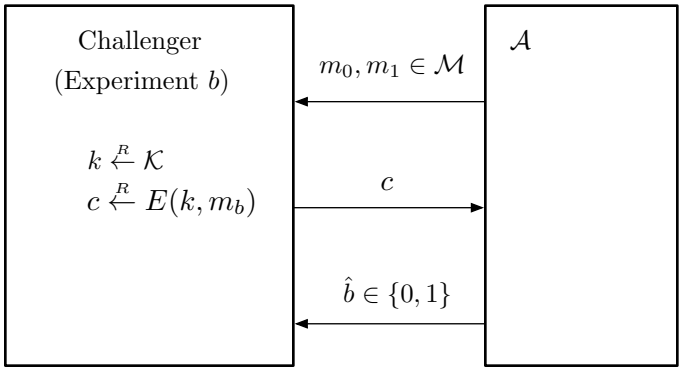


Figure 2.1: Experiment b of Attack Game 2.1

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

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

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 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 (2.6) 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 (2.6), 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 just 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. [20] 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 2.4 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. Also, an *efficient adversary* is one that runs ins a"reasonable" amount time.

We introduce two other terms:

- A value N is called *super-poly* is $1/N$ is negligible.
- A *poly-bounded* value which intuitively a reasonably sized number — in particular, we can say that the running time of any efficient adversary is a poly-bounded value.

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.

2.3.3 Connections to weaker notions of security

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 $m \xleftarrow{R} \mathcal{M}$, $k \xleftarrow{R} \mathcal{K}$, $c \xleftarrow{R} E(k, m)$, 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

$$\mathrm{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \big|\Pr[W] - 1/|\mathcal{M}|\big|. \quad \square$$

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* $\mathrm{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *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, $\mathrm{MR}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, is negligible. Let p denote the probability that $\mathcal{A}$ wins the message recovery game, so that

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

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:

$$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] \leq \mathrm{SSadv}[\mathcal{B}, \mathcal{E}]. \tag{2.7}$$

Since $\mathcal{B}$ is efficient, and since we are assume $\mathcal{E}$ is semantically secure, the right-hand side of (2.7) is negligible, and so we conclude that $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}]$ is negligible.

So all that remains to complete the proof is to show how to construct an efficient $\mathcal{B}$ that satisfies (2.7). The idea is to use $\mathcal{A}$ as a "black box" — we do not have to understand the inner workings of $\mathcal{A}$ as at all.

Here is how $\mathcal{B}$ works. Adversary $\mathcal{B}$ generates two random messages, m_0 and m_1, 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 $\mathcal{B}$ compares m_0 to $\hat{m}$, and outputs $\hat{b} = 1$ if $m_0 = \hat{m}$, and $\hat{b} = 1$ otherwise.

That completes the description of $\mathcal{B}$, which is illustrated in Fig. **??**.

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

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

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

$$\mathrm{SSadv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0| = \left|1/|\mathcal{M}| - p\right| = \mathrm{MRadv}[\mathcal{A}, \mathcal{E}].$$

This proves (2.7). In fact, equality holds in (2.7), 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

$$\mathrm{MRadv}[\mathcal{A}, \mathcal{E}] \leq \mathrm{SSadv}[\mathcal{B}, \mathcal{E}]. \tag{2.8}$$

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 (2.8) 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 (2.8). By assumption, $\mathrm{MRadv}[\mathcal{A}, \mathcal{E}]$ is non-negligible, and (2.8) implies that $\mathrm{SSadv}[\mathcal{B}, \mathcal{E}]$ 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 2.4 (but again, they are extremely tedious).

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 parity(m) to be 1 if the number of 1's in m is odd, and 0 otherwise. Equivalently, 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 parity(m). Now, since 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 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 parity$(m_0) = 0$ and parity$(m_1) = 1$. Then it hands these two messages to its own SS challenger, obtaining a ciphertext c, which it then forwards to it $\mathcal{A}$. 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 $m \xleftarrow{R} \mathcal{M}$, $k \xleftarrow{R} \mathcal{K}$, $c \xleftarrow{R} E(k, m)$, 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 **message recovery advantage** with respect to $\mathcal{E}$ as

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

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

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

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 $m_1 \leftarrow m_0 \oplus (0^{L-1} \,\|\, 1)$; that is, m_1 is that same as m_0, except that the last bit is flipped. In particularly, 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

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

We claim that $p_0 = 1/2 + \epsilon$ and $p_1 = 1/2 - \epsilon$. This 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 $\mathcal{A}$ with output $\text{parity}(m_1)$ with probability $1/2 + \epsilon$, and so outputs $\text{parity}(m_0)$ 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. $\square$

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 3.17).

2.3.4 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 looses, 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 in, 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 $W(r, bet)$, which is 1 if *bet* is a winning bet with respect to r, and 0 otherwise. Let us define

$$\mathrm{IRadv}[\mathcal{A}] := \big|\Pr[W(r, bet) = 1] - 18/37\big|.$$

Our goal is to prove the following theorem.

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

As we did in Section 2.3.3, 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

$$\mathrm{IRadv}[\mathcal{A}] = \mathrm{SSadv}[\mathcal{B}, \mathcal{E}]. \tag{2.9}$$

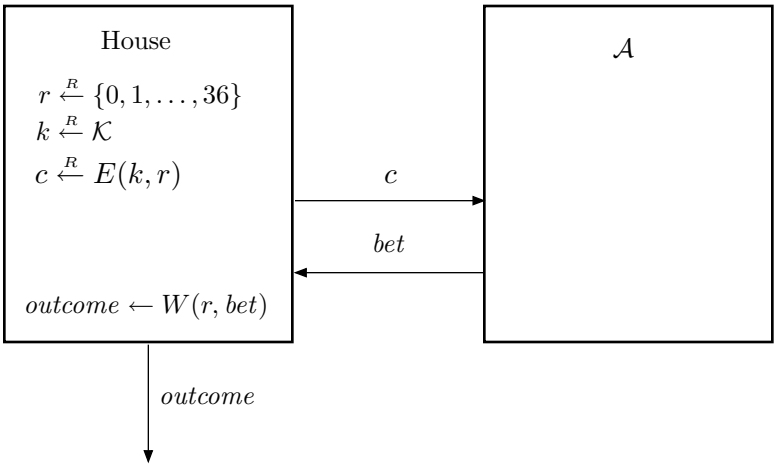


Figure 2.2: Internet roulette

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,

$$\mathsf{SSadv}[\mathcal{B}, \mathcal{E}] = |p_1 - p_0|. \tag{2.10}$$

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 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,

$$\mathsf{IRadv}[\mathcal{A}] = |p_1 - p_0|. \tag{2.11}$$

Combining (2.10) and (2.11), we obtain (2.9).

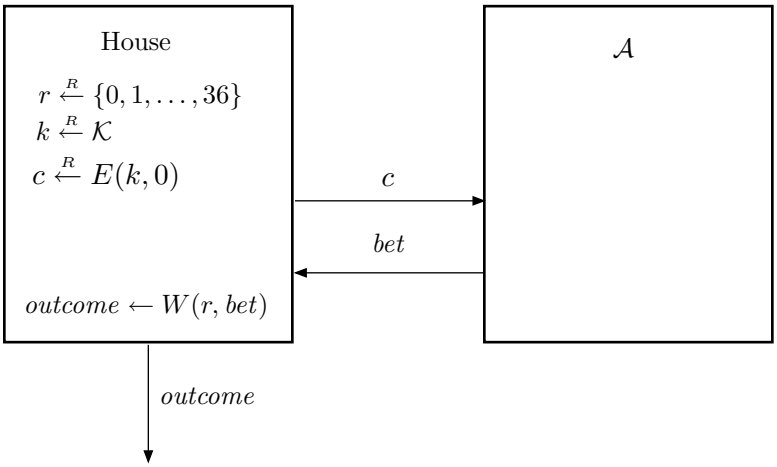


Figure 2.3: ideal Internet roulette

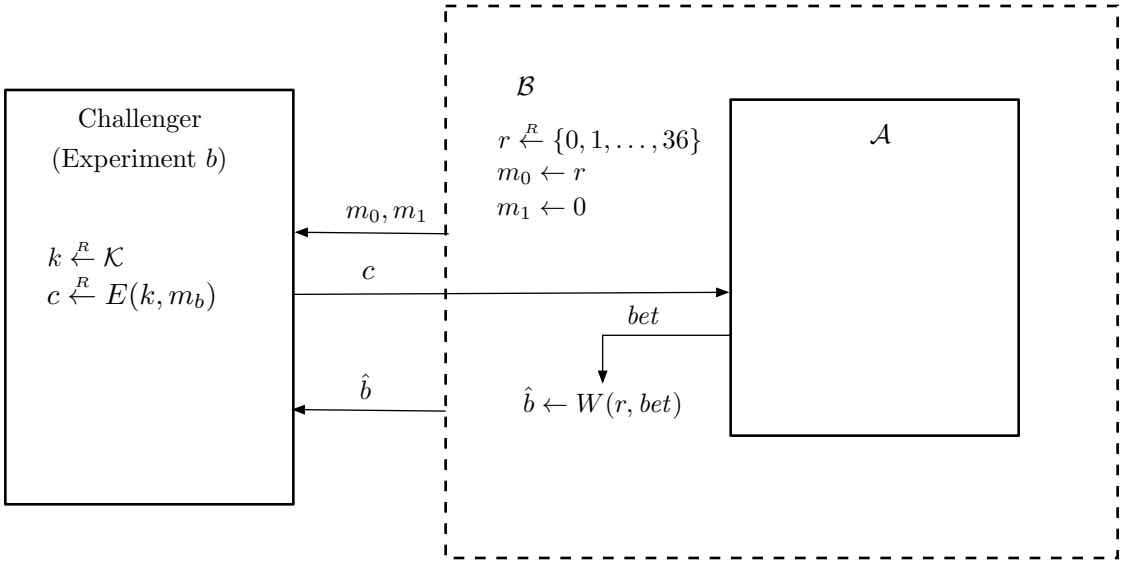


Figure 2.4: The SS adversary $\mathcal{B}$ in Attack Game 2.1

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.

2.3.5 Bit guessing: an alternative characterization of semantic security

The example in Section 2.3.4 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 $b \xleftarrow{R} \{0,1\}$, $k \xleftarrow{R} \mathcal{K}$, $c \xleftarrow{R} E(k, m_b)$, 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 $\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]$. Then we have:

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

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

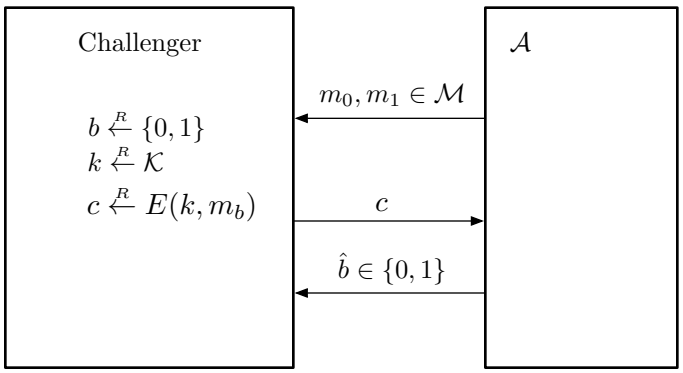


Figure 2.5: Attack Game 2.4

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, the values of k, as well as the random choices made by the encryption algorithm 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 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$ as $\mathcal{A}$'s "SS advantage," we shall refer to $\mathsf{SSadv}^*[\mathcal{A}, \mathcal{E}]$ as $\mathcal{A}$'s "bit-guessing SS advantage."

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 of 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

$$\mathsf{Xadv}[\mathcal{A}, \mathcal{S}] = 2 \cdot \mathsf{Xadv}^*[\mathcal{A}, \mathcal{S}]. \tag{2.12}$$

2.4 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 2.3.4, 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.

2.4.1 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:

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

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. □

2.4.2 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 a 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 polybounded 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:

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.*

- *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:

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:*

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

[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.

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* $\mathsf{reject} \notin \mathcal{M}_{\lambda,\Lambda}$.

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 2.3.1, 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. $\square$

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

[2]Note that the definition of a Shannon cipher in Section 2.2.1 remains unchanged. The claim made at the end of Section 2.3.1 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 $(\mathcal{K}_{\lambda,\Lambda}, \mathcal{M}_{\lambda,\Lambda}, \mathcal{C}_{\lambda,\Lambda})$.

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 2.3.1, 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.

2.4.3 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]

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 bounded ones), the probability that the total running time*

[3] Analogous to the discussion in footnote 1 on page 44, 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 requirement of any machines we consider.

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 message sent by M gets sent to M'. We assume that if M halts, then M' does not send it any more messages. See Fig. **??**.

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.*

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 is 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.

2.4.4 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 made 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 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}]$. This value is actually a function of the security parameter λ. The proper interpretation of Definition 2.3 is that $\mathcal{E}$ is secure if for all efficient adversaries $\mathcal{A}$ (modeled as an interactive machine, as described above), the function $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}](\lambda)$ 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 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}](\lambda)$ 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.

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 be negligible (resp., poly-bounded), which really means that v is a negligible (resp., poly-bounded) function of the security parameter.

2.5 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

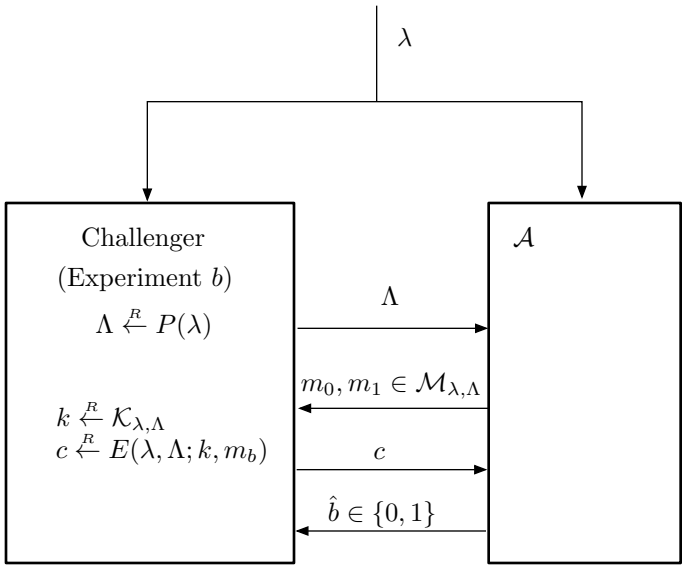


Figure 2.6: The fully detailed version of Attack Game 2.1

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 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 $c_i := E\big(k_i,\ \langle \mathit{destination}_i, m_i \rangle\big)$. 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.

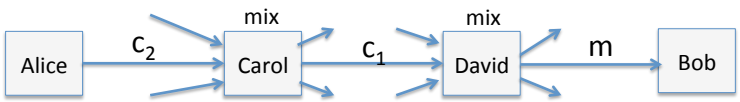


Figure 2.7: An example onion routing using two mixes

To send her message to Bob she constructs the following nested ciphertext c_2:

$$c_2 := E\big(k_c,\ E(k_d, m)\big)\ . \tag{2.13}$$

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

$$c_2 := E\big(k_c,\ \langle \mathsf{David}, c_1\rangle\big) \qquad \text{where} \qquad c_1 := E\big(k_d,\ \langle \mathsf{Bob}, m\rangle\big)\ .$$

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.

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 (2.13). Otherwise, Carol could potentially use the information she learns about m from c_2 to link Alice to her post on Bob's 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 holds for David: it had better be the case that David, who knows k_d, can learn no information about m from the nested ciphertext c_2 in (2.13).

Let us argue that if $\mathcal{E}$ is semantically secure then no efficient adversary can learn any information about m given c_2 and one of k_c or k_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

$$E_n\big((k_0, \ldots, k_{n-1}),\ m\big) = E\big(k_{n-1},\ E(k_{n-2},\ \cdots E(k_0, m)\cdots)\big)\ .$$

Decryption applies the keys in the reverse order:

$$D_n\big((k_0, \ldots, k_{n-1}),\ c\big) = D\big(k_0,\ D(k_1,\ \cdots D(k_{n-1}, c)\cdots)\big)\ .$$

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 $k_0, \ldots, k_{n-1}$. 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 $0 \le d < n$.
- The challenger chooses n keys $k_0, \ldots, k_{n-1} \xleftarrow{R} \mathcal{K}$ and computes $c \xleftarrow{R} E_n\big((k_0, \ldots, k_{n-1}),\ m_b\big)$. It sends c to the adversary along with all keys $k_0, \ldots, k_{n-1}$, 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 $k_0, \ldots, k_{n-1}$ 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.

2.6 Notes

The one time pad is due to Gilbert Vernam in 1917, although there is evidence that it was discovered earlier [9].

Citations to the literature to be added.

2.7 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. 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. Let $\mathcal{E} = (E, D)$ be a perfectly secure 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 $\mathcal{E}'$ is perfectly secure.

2.4. 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.

2.5. 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 $\mathsf{SSadv}[\mathcal{A}, \mathcal{E}] \leq \epsilon$ for *all* adversaries $\mathcal{A}$, even including *computationally unbounded* ones. Show that $|\mathcal{K}| \geq (1-\epsilon)|\mathcal{M}|$.

2.6. This exercise develops a converse of sorts for the previous exercise. For $j = 0, \ldots, L-1$, 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 $L-j$ 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. 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. In Section 2.3.4, 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 2.4.

2.10. 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) $E'(k, m) = 0 \,\|\, E(k, m)$

(b) $E'(k, m) = E(k, m) \,\|\, \text{parity}(m)$

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

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

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

$$\mathsf{KRadv}[\mathcal{A}, \mathcal{E}] \leq \mathsf{SSadv}[\mathcal{B}, \mathcal{E}] + \epsilon$$

Hint: Your semantic security adversary $\mathcal{B}$ will output 1 with probability $\mathsf{KRadv}[\mathcal{A}, \mathcal{E}]$ in the semantic security Experiment 0 and output 1 with probability at most ϵ in Experiment 1. Deduce from this a lower bound on $\mathsf{SSadv}[\mathcal{B}, \mathcal{E}]$ in terms of ϵ and $\mathsf{KRadv}[\mathcal{A}, \mathcal{E}]$ 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. In Section 2.3.3 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. 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 $\{0, \ldots, \ell - 1\}$. 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 $\mathcal{E} = (E, D)$ 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 semantic security experiments, Experiments 0 and 1: in Experiment b, for $b = 0, 1$, the adversary generates two messages m_0 and m_1 and gets back k_1 and $E'\big((k_0, k_1),\ m_b\big)$. The adversary outputs $\hat{b}$ in $\{0,1\}$ and we define its advantage, $\mathrm{NE}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ as in the usual the definition of semantic security. Show that for every nested encryption adversary $\mathcal{A}$ attacking $\mathcal{E}'$, there exists a semantic security adversary $\mathcal{B}$ attacking $\mathcal{E}$, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that

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

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 k_0 (instead of k_1) and $E'\big((k_0, k_1), m_b\big)$ in Experiments 0 and 1. 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 2.5.

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 $k \xleftarrow{R} \mathcal{K}$. 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 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) $\mathsf{SSadv}[\mathcal{A}, \hat{\mathcal{E}}] \leq \mathsf{SSadv}[\mathcal{B}, \mathcal{E}]$.

2.17 (Comparison protocols). Suppose Alice has a secret number a and Bob has a secret number b where both numbers are in $\{0, \ldots, n\}$ for some small n, say $n = 1000$. They wish to run a protocol that will reveal to Alice if $a = b$. However, if $a \neq b$ Alice will learn nothing else about b. Either way, Bob will learn nothing about a. Perhaps these values represent their votes in an election and they wish to test if they voted for the same candidates. To do so, they fix a prime $p > n$ and use the additive one-time pad modulo p (see Example 2.4). They also enlist the help of a third party, Carol, that should learn nothing about a and b from the protocol. Bob begins by choosing three random values k_0, k_1 in $\mathbb{Z}_p$ and $r \in \mathbb{Z}_p^*$. Bob sends k_0, k_1 to Alice and r and $s := k_0 + r(b + k_1)$ in $\mathbb{Z}_p$ to Carol. Next Alice sends $u := k_1 + a$ in $\mathbb{Z}_p$ to Carol and Carol responds to Alice with $v := s - ru$ in $\mathbb{Z}_p$.

(a) Show that $v - k_0 = r(b - a)$ in $\mathbb{Z}_p$ and deduce that $v - k_0 = 0$ if and only if $a = b$. Therefore Alice learns if $a = b$ by testing if $v - k_0 = 0$.

(b) Security: Bob clearly receives no information from this protocol. Carol learns the quantities r, s, u. Use the security of the modular one-time pad to argue that these quantities are independent of a and b (the previous exercise may be helpful). Therefore, Carol learns nothing about a and b. Finally, Alice learns the quantities k_0, k_1, v. Use the security of the multiplicative one-time pad from Exercise 2.1 to argue that when $a \neq b$ the quantity v is independent of $b - a$. Therefore Alice learns nothing else about b. Note that this security analysis assumes that Carol does not collude with either Alice or Bob.

2.18. Two standards committee proposes 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 [60] and BREACH [56] attacks are good representative examples.

2.19 (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: $c_0 \xleftarrow{R} \{0, \ldots, n-1\}$, 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.

2.20 (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 $k_a := r$ and $k_b := k \oplus r$. Now if Alice and Bob get together they can decrypt a ciphertext c by first reconstructing the key k as $k = k_a \oplus k_b$ 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 $\mathrm{2KSadv}[\mathcal{A}, \mathcal{E}]$.

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

2.21 (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 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.

(c) More generally, we can design a t-out-of-n system this way for any $t < n$. How does the size of each share scale with t?

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 are 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 2.4).

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

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 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:

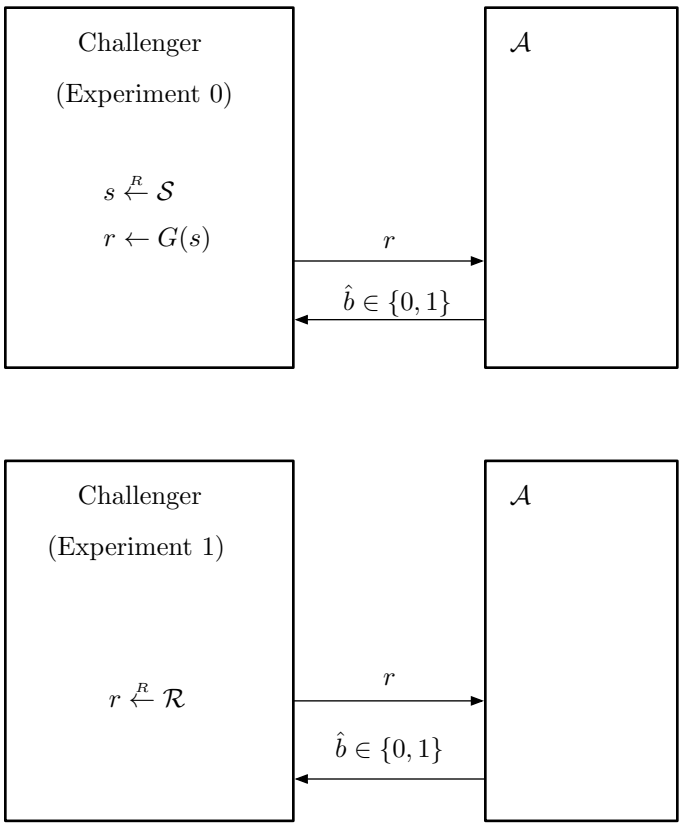


Figure 3.1: Experiments 0 and 1 of Attack Game 3.1

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

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

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

The attack game is illustrated in Fig. 3.1.

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

As discussed in Section 2.3.5, 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*

$\mathsf{PRGadv}^*[\mathcal{A}, G]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.3.5 applies here as well: $\mathsf{PRGadv}[\mathcal{A}, G] = 2 \cdot \mathsf{PRGadv}^*[\mathcal{A}, G]$.

We also note that a PRG can only be secure if the cardinality of the seed space is super-poly (see Exercise 3.6).

3.1.2 Mathematical details

Just as in Section 2.4, we give here more of the mathematical details pertaining to PRGs. Just like Section 2.4, 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 $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, 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 $\mathsf{PRGadv}[\mathcal{A}, G]$ 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

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

and if $|c| = v$, then

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

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*
>
> $$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}] = \mathrm{PRGadv}[\mathcal{B}, G].$$

Proof idea. In proving this theorem, it is more convenient to work with the bit-guessing version of the SS attack game, discussed in Section 2.3.5. Recall that, by Theorem 2.10, the adversary's bit guessing SS advantage, $\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}]$, is twice its SS advantage, $\mathrm{SSadv}[\mathcal{A}, \mathcal{E}]$, so if one is negligible, so is the other.

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 bit-guessing advantage by more than a negligible amount. However, after making this replacement, the adversary's bit-guessing advantage is zero. □

Proof. Let $\mathcal{A}$ be an efficient adversary attacking the semantic security of the cipher 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:
> $b \xleftarrow{R} \{0,1\}$
> $s \xleftarrow{R} \{0,1\}^\ell$, $r \leftarrow G(s)$
> $c \leftarrow r[0\,..\,v-1] \oplus m_b$
> 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

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}] = \cdot|\Pr[W_0] - 1/2|. \tag{3.1}$$

Next, we modify the challenger of Game 0, obtaining 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:
> $b \xleftarrow{R} \{0,1\}$
> $r \xleftarrow{R} \{0,1\}^L$
> $c \leftarrow r[0\,..\,v-1] \oplus m_b$
> send c to $\mathcal{A}$.

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

$$\Pr[W_1] = 1/2. \tag{3.2}$$

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.

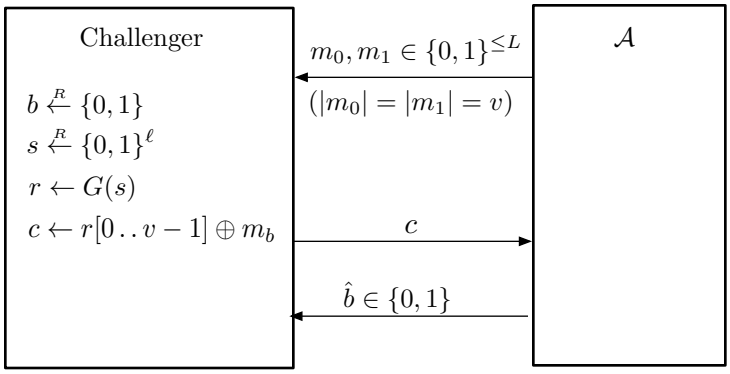


Figure 3.2: Game 0 in the proof of Theorem 3.1

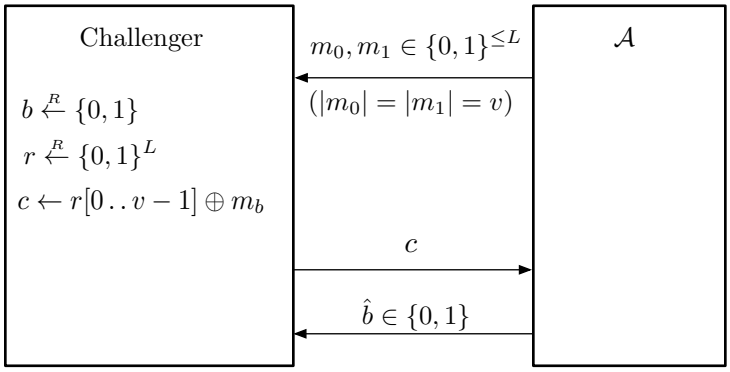


Figure 3.3: Game 1 in the proof of Theorem 3.1

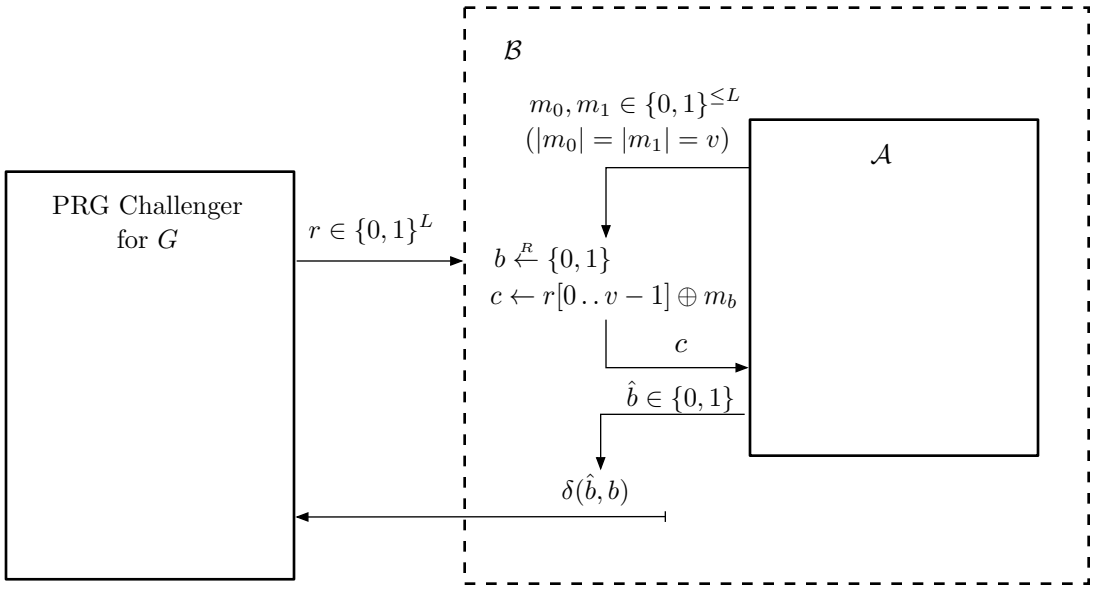


Figure 3.4: The PRG adversary $\mathcal{B}$ in the proof of Theorem 3.1

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

$$|\Pr[W_0] - \Pr[W_1]| = \mathsf{PRGadv}[\mathcal{B}, G]. \tag{3.3}$$

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:

$$\delta(x, y) := \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{if } x \neq y. \end{cases} \tag{3.4}$$

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:
> $b \xleftarrow{R} \{0,1\}$
> $c \leftarrow r[0\,..\,v-1] \oplus m_b$
> 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, $\mathsf{PRGadv}[\mathcal{B}, G] = |p_1 - p_0|$. 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 (3.3) now follows immediately.

Combining (3.1), (3.2), and (3.3), we have

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}] = \mathrm{PRGadv}[\mathcal{B}, G].$$

Moreover, since $\mathcal{B}$ is just as efficient as $\mathcal{A}$, under our assumption that G is a secure PRG, the value $\mathrm{PRGadv}[\mathcal{B}, G]$ must be negligible, and therefore, $\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}]$ must be negligible as well. □

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

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}] \leq \mathrm{PRGadv}[\mathcal{B}, G].$$

(Actually, we showed that equality holds, but that is not so important.) In the proof, we argued that if G is secure, then $\mathrm{PRGadv}[\mathcal{B}, G]$ is negligible, hence by the above inequality, we conclude that $\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}]$ 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 $\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}]$ is non-negligible, and the reduction (and the above inequality) gives us an efficient adversary $\mathcal{B}$ such that $\mathrm{PRGadv}[\mathcal{B}, G]$ 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 2.3.4. 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 highly brittle and become totally 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:

$$c_1 \leftarrow m_1 \oplus G(s) \qquad \text{and} \qquad c_2 \leftarrow m_2 \oplus G(s) \tag{3.5}$$

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 (3.5) leaks the plaintexts after seeing only two sufficiently long ciphertexts.

The construction in (3.5) 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 [62]. 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 [35] 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 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 help give indisputable 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. Suppose Alice stores incoming emails on a disk encrypted with her secret stream cipher key. An email from Bob always starts with the characters `From:␣Bob`. An attacker who knows that a certain email is from Bob, can use the stream cipher malleability to change the encrypted email to say `From:␣Eve`. He simply xors the appropriate three character string with the email characters in positions 7 to 9. The attacker makes this change by only operating on ciphertexts and without knowledge of Alice's secret key. Alice will never know that the sender's address was changed.

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 $s_1, \ldots, s_n \in \mathcal{R}$,

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

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*
>
> $$\mathrm{PRGadv}[\mathcal{A}, G'] = n \cdot \mathrm{PRGadv}[\mathcal{B}, G].$$

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 replaced 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:

$r_1 \leftarrow r$
$s_1 \xleftarrow{R} \mathcal{S}$, $r_2 \leftarrow G(s_2)$
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:

$r_1 \xleftarrow{R} \mathcal{R}$
$r_2 \xleftarrow{R} \mathcal{R}$
send (r_1, r_2) to $\mathcal{A}$.

All we have done is replaced 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:
>
> $r_1 \xleftarrow{R} \mathcal{R}$
> $r_2 \leftarrow r$
> 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 $\epsilon = \mathrm{PRGadv}[\mathcal{A}, G']$, 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

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

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

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 $\mathcal{S}$.

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

$$\mathrm{PRGadv}[\mathcal{A}, G'] = |p_n - p_0|. \tag{3.6}$$

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:

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

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

and combining this with (3.6), we have

$$\mathrm{PRGadv}[\mathcal{A}, G'] = n \cdot \mathrm{PRGadv}[\mathcal{B}, G].$$

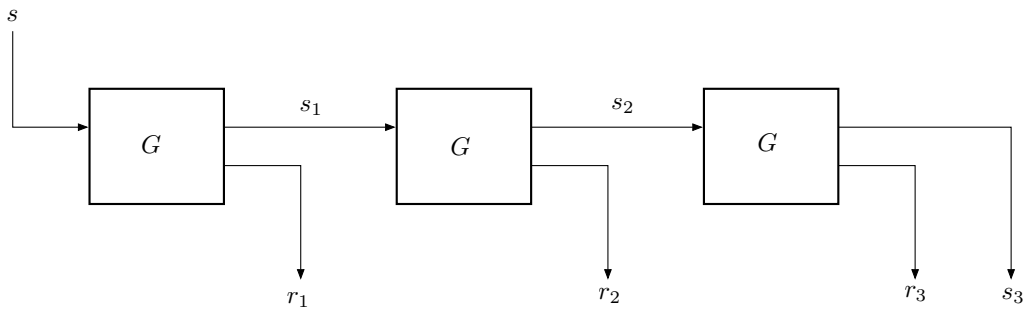


Figure 3.6: The sequential construction for $n = 3$

Since we are assuming G is a secure PRG, it follows that $\mathrm{PRGadv}[\mathcal{B}, G]$ is negligible, and since n is poly-bounded, it follows that $\mathrm{PRGadv}[\mathcal{A}, G']$ 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

$$\mathrm{PRGadv}[\mathcal{A}, G'] = n \cdot \mathrm{PRGadv}[\mathcal{B}, G].$$

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 $(r_1, \ldots, r_n, s_n)$ 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

$$\mathrm{PRG}\mathsf{adv}[\mathcal{A}, G'] = |p_n - p_0|. \tag{3.7}$$

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:

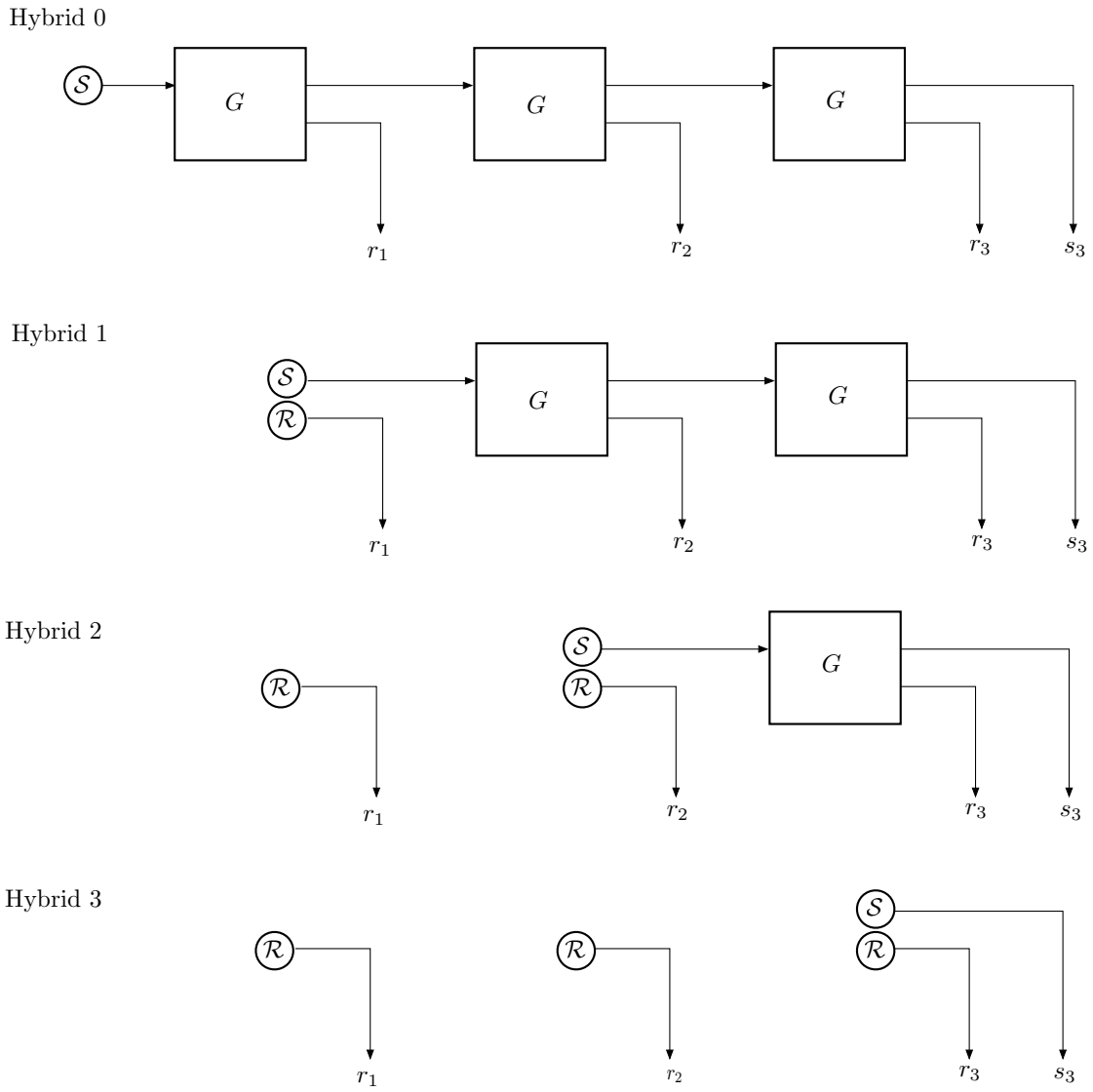


Figure 3.7: The challenger's computation in the hybrid games for $n = 3$. The circles indicate randomly generated elements of $\mathcal{S}$ or $\mathcal{R}$, as indicated by the label.

$\omega \xleftarrow{R} \{1, \ldots, n\}$
$r_1 \xleftarrow{R} \mathcal{R}$
$\vdots$
$r_{\omega-1} \xleftarrow{R} \mathcal{R}$
$(r_\omega, s_\omega) \leftarrow (r, s)$
$(r_{\omega+1}, s_{\omega+1}) \leftarrow G(s_\omega)$
$\vdots$
$(r_n, s_n) \leftarrow G(s_{n-1})$
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 Exercise 4.18).

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.

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 our 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 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 $0 \leq i \leq L-1$, to the challenger.
- The challenger computes
$$s \xleftarrow{R} \mathcal{S}, \;\; r \leftarrow G(s)$$
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** $\mathsf{Predadv}[\mathcal{A}, G]$ to be $|\Pr[\mathcal{A} \text{ wins}] - 1/2|$. □

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

We begin by showing the 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 G as in Attack Game 3.1, where $\mathcal{B}$ is an elementary wrapper around $\mathcal{A}$, such that*
>
> $$\mathsf{Predadv}[\mathcal{A}, G] = \mathsf{PRGadv}[\mathcal{B}, G].$$

Proof. Let $\mathcal{A}$ be an adversary breaking the predictability 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 $\mathsf{Predadv}[\mathcal{A}, G] = |\epsilon|$.

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 $r[0\,..\,i-1]$ 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

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

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_{j+1}, 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, or 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 $\epsilon = \Pr[d(\mathbf{X}, \mathbf{B}) = 1] - \Pr[d(\mathbf{X}, \mathbf{R}) = 1]$.

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

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

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

Define the random variable $\mathbf{B}'$ as follows:

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

Then

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

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

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

where

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

By independence, we have

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

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

We thus calculate that

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

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

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

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

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

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

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

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 the 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 Burnstein 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, SSH, and QUIC.

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. $\quad h_j \leftarrow \text{pad}(s, j, 0) \in \{0,1\}^{512}$
3. $\quad r_j \leftarrow \pi(h_j) \oplus h_j$
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 4.21 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:

$$\begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix} \longleftarrow \begin{pmatrix} c_0 & c_1 & c_2 & c_3 \\ s_0 & s_1 & s_2 & s_3 \\ s_4 & s_5 & s_6 & s_7 \\ j_0 & j_1 & n_0 & n_1 \end{pmatrix} \tag{3.8}$$

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

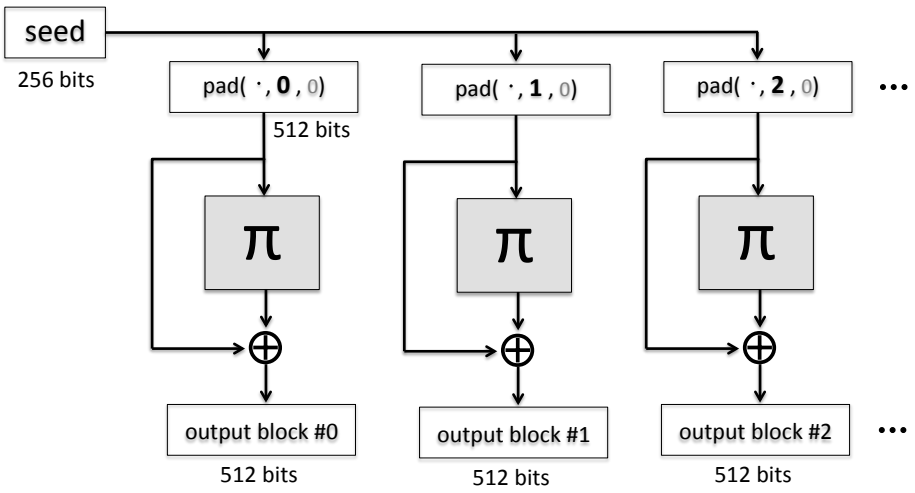


Figure 3.8: A schematic of the Salsa and ChaCha PRGs

The permutation $\pi : \{0,1\}^{512} \to \{0,1\}^{512}$ is constructed by iterating a simpler 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 $x_0, \ldots, x_{15}$. In ChaCha20 the function π is implemented by repeating the following sequence of steps ten times:

$$\begin{array}{lll} \text{QuarterRound}(x_0, x_4, x_8, x_{12}), & \text{QuarterRound}(x_1, x_5, x_9, x_{13}), & \text{QuarterRound}(x_2, x_6, x_{10}, x_{14}), \\ \text{QuarterRound}(x_3, x_7, x_{11}, x_{15}), & \text{QuarterRound}(x_0, x_5, x_{10}, x_{15}), & \text{QuarterRound}(x_1, x_6, x_{11}, x_{12}), \\ \text{QuarterRound}(x_2, x_7, x_8, x_{13}), & \text{QuarterRound}(x_3, x_4, x_9, x_{14}) & \end{array}$$

where QuarterRound(a, b, c, d) is defined as the following sequence of steps written as C code:

```
a += b;  d ^= a;  d <<<= 16;
c += d;  b ^= c;  b <<<= 12;
a += b;  d ^= a;  d <<<= 8;
c += d;  b ^= c;  b <<<= 7;
```

The first four invocations of QuarterRound are applied to each of the first four columns. The next four invocations are applied to each of the four diagonals. 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 and we present it to give an example of some 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: linear congruential generators

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:

$$\begin{aligned} G_{\text{lcg}}^{(n)}(s) := \quad & s_0 \leftarrow s \\ & \text{for } i \leftarrow 1 \text{ to } n \text{ do} \\ & \qquad r_i \leftarrow \lfloor s_{i-1}/w \rfloor, \quad s_i \leftarrow a s_{i-1} + b \bmod q \\ & \text{output } (r_1, \ldots, r_n). \end{aligned}$$

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 $a = \text{0x5DEECE66D}, b = \text{0x0B}$. 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 of the first iteration of the generator reveals all but 22 bits of the seed s. An attacker can easily recover these unknown 22 bits by exhaustive search: for every possible value

of the 22 bits the attacker forms a candidate seed $\hat{s}$. It tests if $\hat{s}$ is the correct seed by comparing subsequent outputs computed from seed $\hat{s}$ to a few subsequent outputs 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 [29] 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.

More precisely, suppose that $w < \sqrt{q}/c$ 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.

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 $s_i \in \mathcal{S}_q$. We have

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

where e_0 and e_1 are the remainders after dividing s_i and s_{i+1} by w; in particular, $0 \le e_0, e_1 < w < \sqrt{q}/c$. The fact that e_0, e_1 are smaller than $\sqrt{q}$ 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 values x, s, e_0, e_1 are unknown to the attacker, but it knows r_i, r_{i+1}, w, a, b. Finally, re-arranging terms to put the terms involving x and s on the left gives

$$s = r_i \cdot w + e_0 \qquad \text{and} \qquad as + qx = r_{i+1}w - b + e_1 \ . \tag{3.9}$$

We can re-write (3.9) in vector form as

$$s \cdot \begin{pmatrix} 1 \\ a \end{pmatrix} + x \cdot \begin{pmatrix} 0 \\ q \end{pmatrix} = \mathbf{g} + \mathbf{e} \qquad \text{where} \qquad \mathbf{g} := \begin{pmatrix} r_i w \\ r_{i+1}w - b \end{pmatrix} \quad \text{and} \quad \mathbf{e} := \begin{pmatrix} e_0 \\ e_1 \end{pmatrix} \ . \tag{3.10}$$

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

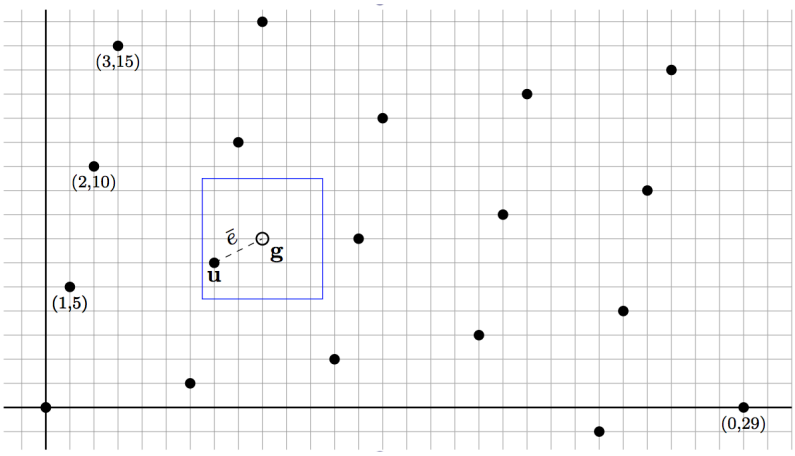


Figure 3.9: The two-dimensional lattice associated with attacking the LCG. Here the lattice is generated by the vectors $(1,5)^\intercal$ and $(0,29)^\intercal$. The attacker has a vector $\mathbf{g} = (9,7)^\intercal$ and wishes to find the closest lattice vector $\mathbf{u}$. In this picture there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

We show how to find $\mathbf{u}$ from $\mathbf{g}$. 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 $\mathbf{g} \in \mathbb{Z}^2$ and knows that his target vector $\mathbf{u} \in \mathcal{L}_a$ is close to $\mathbf{g}$. If he could find the *closest* vector in $\mathcal{L}_a$ to $\mathbf{g}$ then there is a good chance that this vector is the desired vector $\mathbf{u}$. 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 $\mathbf{g} \in \mathbb{Z}^2$ there is at most one vector $\mathbf{u} \in \mathcal{L}_a$ such that $\|\mathbf{g} - \mathbf{u}\|_\infty < \sqrt{q}/c$.*

Taking $c = 32$ in Lemma 3.7 (so that $w = \sqrt{q}/30$) shows that for 98% of the $a \in \mathcal{S}_q$ the closest vector to $\mathbf{g}$ in $\mathcal{L}_a$ is precisely the desired vector $\mathbf{u}$. Before proving the lemma, let us first complete the description of the attack.

It remains to efficiently find the closest vector to $\mathbf{g}$ 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 $\mathbf{g}$, find the vector in $\mathcal{L}$ that is closest to $\mathbf{g}$. When the lattice $\mathcal{L}$ is two dimensional there is an efficient polynomial time algorithm for this problem [69]. 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 some example $a \in \mathcal{S}_q$ in the 2% where the attack fails are $a = 1$ and $a = 2$. For these a there may be many lattice vector in $\mathcal{L}_a$ close to a given $\mathbf{g}$. We leave it as a fun exercise to devise an attack that works for the a in $\mathcal{S}_q$ to which Lemma 3.7 does not apply. We conclude this section with a proof of Lemma 3.7.

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

by the triangle inequality, we have

$$\|\mathbf{u}_0 - \mathbf{u}_1\|_\infty \leq \|\mathbf{u}_0 - \mathbf{g}\|_\infty + \|\mathbf{g} - \mathbf{u}_1\|_\infty \leq 2\sqrt{q}/c \ .$$

Since any lattice is closed under addition, we see that $\mathbf{u} := \mathbf{u}_0 - \mathbf{u}_1$ 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 at most $B := 2\sqrt{q}/c$. So let us bound the number of "bad" a's for which $\mathcal{L}_a$ contains such a short vector.

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

The same bound on the number of bad a's holds when q is not prime. To see why consider a specific non-zero $s \in \mathcal{S}_q$ and let $d = \gcd(s, q)$. As above, a vector $\mathbf{t} = (s, y)^\intercal$ 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 $\mathbf{t} = (s, y)^\intercal$ is in at most d lattices $\mathcal{L}_a$. Since there are $2B$ possible values for s, this shows that 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 $(1 - 16/c^2) \cdot q$ 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 $\mathcal{S}_q := \{0, \ldots, q-1\}$. Choose n integers $\mathbf{a} := (a_0, \ldots, a_{n-1})$ in $\mathcal{S}_q$ and define the subset sum function $f_\mathbf{a} : \{0,1\}^n \to \mathcal{S}_q$ as

$$f_\mathbf{a}(\mathbf{s}) := \sum_{i : s_i = 1} a_i \bmod q \ .$$

Now, for a target integer $t \in \mathcal{S}_q$ the modular subset problem is defined as follows:

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

In other words, the problem is to invert the function $f_\mathbf{a}(\cdot)$ by finding a pre-image of t, 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 random integers $\vec{a} := (a_0, \ldots, a_{n-1})$ in $\mathcal{S}_q$. The PRG $G_{q,\vec{a}}$ takes a seed $\mathbf{s} \in \{0,1\}^n$ and outputs a pseudorandom value in $\mathcal{S}_q$. It is defined as

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

The PRG expands an n bit seed to a $\log_2 q$ bits of output. Choosing an n and q so that $2n = \log_2 q$ gives a PRG whose output is twice the size of the input. We can plug this into the Blum-Micali construction to expand the output further.

While the PRG is far slower than custom constructions like ChaCha20 from Section 3.6, the work 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 [**?**] show that attacking $G_{q,\vec{a}}$ as a PRG is as hard as solving a certain randomized variant of the modular subset sum problem. We present the analysis in Section **??** after we develop a few more tools. While there is considerable work on solving the modular subset problem, the problem appears to be hard when $2n = \log_2 q$ for large n, say $n > 1000$, which implies the security of $G_{q,\vec{a}}$ as a PRG.

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

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

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

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.

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.

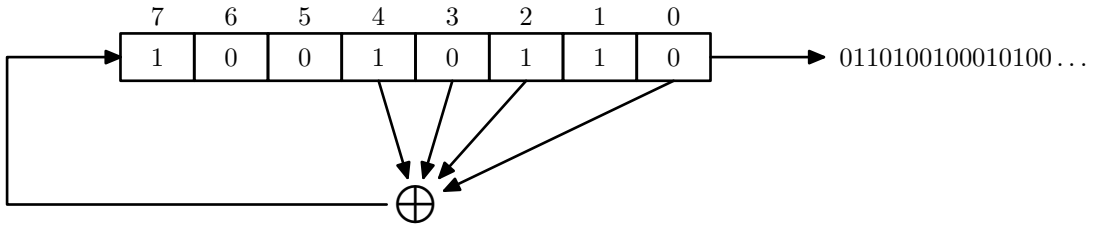


Figure 3.10: The 8 bit linear feedback shift register $\{4, 3, 2, 0\}$

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:

Input: seed $s \in \{0,1\}^{40}$

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

repeat
    run both LFSRs for eight cycles to obtain $x, y \in \{0,1\}^8$
    treat x, y as integers in $0 \ldots 255$
    output $x + y + c \bmod 256$
    if $x + y > 255$ then $c \leftarrow 1$ else $c \leftarrow 0$ // carry bit
forever

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,

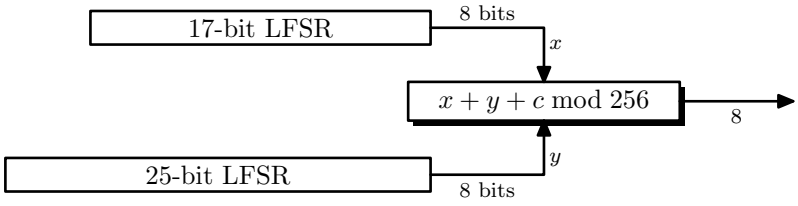


Figure 3.11: The CSS stream cipher

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 simple observations:

- Let (x_1, x_2, x_3) be the first three bytes output by the 17-bit LFSR. The initial state s_2 of the second LFSR is easily obtained once both (z_1, z_2, z_3) and (x_1, x_2, x_3) are known by subtracting one from the other. More precisely, subtract the integer $2^{16}x_3 + 2^8x_2 + x_1$ from the integer $2^{17} + 2^{16}z_3 + 2^8z_2 + z_1$.
- The output (x_1, x_2, x_3) is determined by the 16-bit seed s_1.

With these two observations 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. Subtract (x_1, x_2, x_3) from (z_1, z_2, z_3) to obtain a candidate seed s_2 for the second LFSR. Now, confirm that (s_1, s_2) are the correct secret seed s by running the PRG and comparing 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 found the 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.

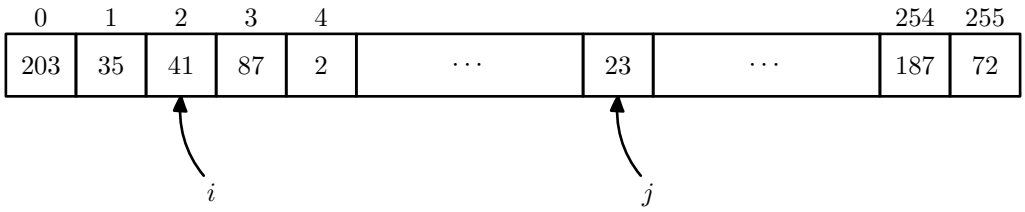


Figure 3.12: An example RC4 internal state

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 an 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**:

$i \leftarrow 0, \quad j \leftarrow 0$

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.

[1]Performance numbers were obtained using the Crypto++ 5.6.0 benchmarks running on a 1.83 GhZ Intel Core 2 processor.

cipher	speed[1](MB/sec)
RC4	126
SEAL	375
Salsa20	408
Sosemanuk	727

Table 3.1: Software stream cipher speeds (higher speed is better)

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 $0 \ldots 255$ 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 [44] 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 $0 \ldots n-1$ 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 $2/n$.*

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

$$\begin{aligned}\Pr[z_2 = 0] &= \Pr\left[(z_2 = 0) \mid P\right] \cdot \Pr[P] + \Pr\left[(z_2 = 0) \mid \neg P\right] \cdot \Pr[\neg P] \\ &\approx 1 \cdot (1/n) + (1/n) \cdot (1 - 1/n) \approx 2/n \quad \square\end{aligned}$$

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. [3] 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

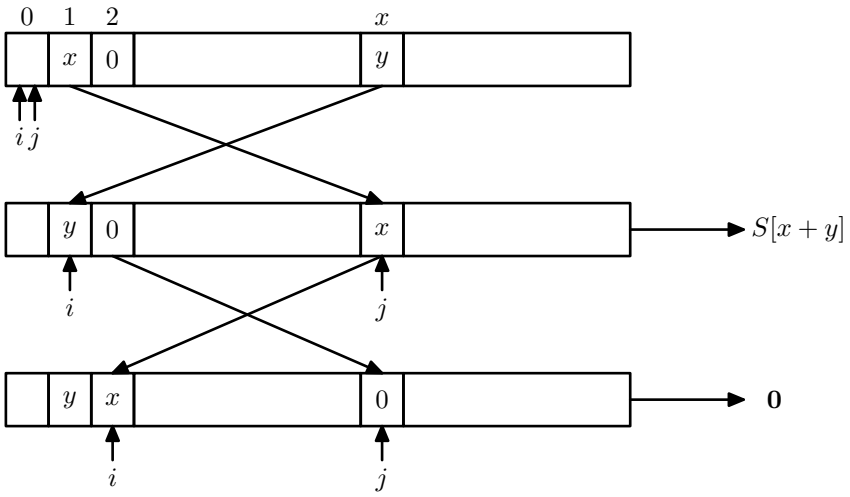


Figure 3.13: Proof of Lemma 3.8

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 [28] 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*

$$i \neq n-1 \implies \Pr[(z_1, z_2) = (0,0)] \geq (1/n^2) \cdot \big(1 + (1/n)\big)$$
$$i \neq 0, 1 \implies \Pr[(z_1, z_2) = (0,1)] \geq (1/n^2) \cdot \big(1 + (1/n)\big)$$

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:

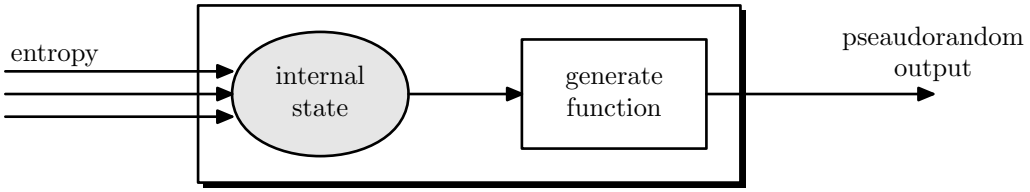


Figure 3.14: A Random Number Generator

input: string $x \in \{0 \ldots n\}^\ell$
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:

$$\ell = 2^{14} \text{ bytes:} \quad \text{PRG}\mathsf{adv}[D, RC4] \geq 2^{-8}$$
$$\ell = 2^{34} \text{ bytes:} \quad \text{PRG}\mathsf{adv}[D, RC4] \geq 0.5$$

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 [27] 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 9.9, 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. 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 to starting with the Ivy processor 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 8.9 where we discuss the key deriviation 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 elements being sampled might stop producing randomness due to hardware glitch. For example, the sampled bits might always be '0' resulting in highly non-random output. To prevent this from happening the RNG output is constantly tested using a fixed set of statistical tests. If any of the tests reports "non-random" the generator is declared to be defective.

3.11 A broader perspective: computational 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 than 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 $x \xleftarrow{R} P_b$ 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:

  $x \xleftarrow{R} P_b$

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

$$\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Definition 3.4 (Computational indistinguishability). *Distributions P_0 and P_1 are called **computationally indistinguishable** if the value $\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1]$ is negligible for all efficient adversaries $\mathcal{A}$.*

Just to exercise the definition a bit: 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 distribution that assigns to each $r \in \mathcal{R}$ the value

$$P_0(r) := \frac{|\{s \in \mathcal{S} : G(s) = r\}|}{|\mathcal{S}|}.$$

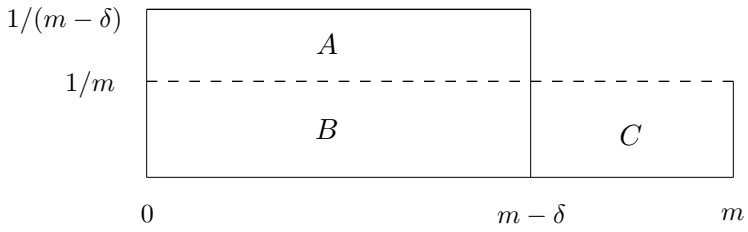
Again, as discussed in Section 2.3.5, Attack Game 3.3 can be recast as a "bit guessing" game, where instead of having two separate experiments, the challenger chooses $b \in \{0,1\}$ at random, and then runs Experiment b against the adversary $\mathcal{A}$. In this game, we measure $\mathcal{A}$'s *bit-guessing advantage* $\mathsf{Dist}\mathsf{adv}^*[\mathcal{A}, P_0, P_1]$ as $|\Pr[\hat{b} = b] - 1/2|$. The general result of Section 2.3.5 applies here as well: $\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1] = 2 \cdot \mathsf{Dist}\mathsf{adv}^*[\mathcal{A}, P_0, P_1]$.

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. *Suppose P_0 and P_1 are probability distributions on a finite set $\mathcal{R}$. Then their **statistical distance** is defined as*

$$\Delta[P_0, P_1] := \frac{1}{2} \sum_{r \in \mathcal{R}} |P_0(r) - P_1(r)|.$$

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,

$$\Delta[P_0, P_1] = \text{area of } A = \text{area of } C = \delta/m. \quad \square$$

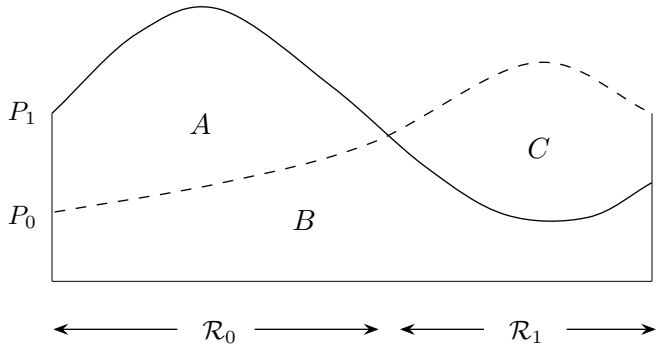
The following theorem allows us to make a connection between the notions of computational indistinguishability and statistical distance:

Theorem 3.10. *Let P_0 and P_1 be probability distributions on a finite set $\mathcal{R}$. Then we have*

$$\max_{\mathcal{R}' \subset \mathcal{R}} |P_0[\mathcal{R}'] - P_1[\mathcal{R}']| = \Delta[P_0, P_1],$$

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 $\Delta[P_0, P_1]$. $\square$

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*

$$\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1] \leq \Delta[P_0, P_1].$$

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

$$\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1] = |P_0[\mathcal{R}'] - P_1[\mathcal{R}']| \leq \Delta[P_0, P_1].$$

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

$$\begin{aligned}
\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1] &= |\Pr[W_0] - \Pr[W_1]| \\
&= \frac{1}{|\mathcal{T}|} \Big| \sum_{t \in \mathcal{T}} (P_0[\mathcal{R}'_t] - P_1[\mathcal{R}'_t]) \Big| \\
&\leq \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} |P_0[\mathcal{R}'_t] - P_1[\mathcal{R}'_t]| \\
&\leq \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \Delta[P_0, P_1] \\
&= \Delta[P_0, P_1]. \quad \square
\end{aligned}$$

As a consequence of this theorem, we see that if $\Delta[P_0, P_1]$ is negligible, then P_0 and P_1 are computationally indistinguishable.

One also defines the statistical distance between two random variables as the statistical distance between their corresponding distributions. That is, if $\mathbf{X}$ and $\mathbf{Y}$ are random variables taking values in a finite set $\mathcal{R}$, then their **statistical distance** is

$$\Delta[\mathbf{X}, \mathbf{Y}] := \frac{1}{2} \sum_{r \in \mathcal{R}} |\Pr[\mathbf{X} = r] - \Pr[\mathbf{Y} = r]|.$$

In this case, Theorem 3.10 says that

$$\max_{\mathcal{R}' \subset \mathcal{R}} \Big| \Pr[\mathbf{X} \in \mathcal{R}'] - \Pr[\mathbf{Y} \in \mathcal{R}'] \Big| = \Delta[\mathbf{X}, \mathbf{Y}],$$

where the maximum is taken over all subsets $\mathcal{R}'$ of $\mathcal{R}$.

Analogously, one can define distinguishing advantage 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 3.12. *If $\mathcal{S}$ and $\mathcal{T}$ are finite sets, $\mathbf{X}$ and $\mathbf{Y}$ are random variables taking values in $\mathcal{S}$, and $f : \mathcal{S} \to \mathcal{T}$ is a function, then $\Delta[f(\mathbf{X}), f(\mathbf{Y})] \leq \Delta[\mathbf{X}, \mathbf{Y}]$.*

Proof. We have

$$\begin{aligned}\Delta[f(\mathbf{X}), f(\mathbf{Y})] &= |\Pr[f(\mathbf{X}) \in \mathcal{T}'] - \Pr[f(\mathbf{Y}) \in \mathcal{T}']| \text{ for some } \mathcal{T}' \subset \mathcal{T} \\ &\qquad \text{(by Theorem 3.10)} \\ &= |\Pr[\mathbf{X} \in f^{-1}(\mathcal{T}')] - \Pr[\mathbf{Y} \in f^{-1}(\mathcal{T}')]| \\ &\leq \Delta[\mathbf{X}, \mathbf{Y}] \text{ (again by Theorem 3.10).} \quad \square\end{aligned}$$

Example 3.2. Let $\mathbf{X}$ be uniformly distributed over the set $\{0, \ldots, m-1\}$, and let $\mathbf{Y}$ 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 $\mathbf{X}$ and $f(\mathbf{Y})$. We can do this as follows. Let $N = qm - r$, where $0 \leq r < m$, so that $q = \lceil N/m \rceil$. Also, let $\mathbf{Z}$ be uniformly distributed over $\{0, \ldots, qm-1\}$. Then $f(\mathbf{Z})$ 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

$$\Delta[\mathbf{X}, f(\mathbf{Y})] = \Delta[f(\mathbf{Z}), f(\mathbf{Y})] \leq \Delta[\mathbf{Z}, \mathbf{Y}],$$

and as we saw in Example 3.1,

$$\Delta[\mathbf{Z}, \mathbf{Y}] = \frac{r}{qm} < \frac{1}{q} \leq \frac{m}{N}.$$

Therefore,

$$\Delta[\mathbf{X}, f(\mathbf{Y})] < \frac{m}{N}. \quad \square$$

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 procedure produces a number that is computationally indistinguishable from a truly random 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, let P_1 be the uniform distribution on $\{0, \ldots, m-1\}$, and 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 $\{0, \ldots, N-1\}$. 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

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

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 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 3.3, we have $\Delta[P, P_1] \leq m/N$.

Putting everything together, we see that

$$\begin{aligned} \mathrm{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1] &= |\Pr[W_0] - \Pr[W_2]| \leq |\Pr[W_0] - \Pr[W_1]| + |\Pr[W_1] + \Pr[W_2]| \\ &\leq \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G] + \frac{m}{N}, \end{aligned}$$

which, by assumption, is negligible. □

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 indistinguishability (Definition 3.4), 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 $\mathrm{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1](\lambda)$, which is a function of λ. Computationally indistinguishability means that this is a negligible function.

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

$$\mathrm{Dist}\mathsf{adv}[\mathcal{A}, P_0, P_1](\lambda) \leq \Delta[P_{0,\lambda}, P_{1,\lambda}].$$

3.12 A fun application: coin flipping and commitments

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 binding property is based on the security of a given 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., [49]).

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 [52].

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 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.

3.13 Notes

Citations to the literature to be added.

3.14 Exercises

Exercises 3.1–3.5 are based on definitions from the previous chapter; however, these exercises should be much more accessible using the techniques introduced in this chapter.

3.1. 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.

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 $\mathcal{M}' = \mathcal{M}$.

3.2. Give an example of a cipher that is semantically secure for random messages (see previous exercise) but that is not semantically secure in the ordinary sense.

3.3. Let $\mathcal{E} = (E, D)$ be a perfectly secure 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 $\mathcal{E}'$ is perfectly secure.

3.4. 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:

$$b \xleftarrow{R} \{0,1\},\ k \xleftarrow{R} \mathcal{K},\ \ m_0 \leftarrow m,\ m_1 \xleftarrow{R} \mathcal{M},\ c \xleftarrow{R} E(k, m_b),$$

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.

3.5. 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:

$$b \xleftarrow{R} \{0,1\},\ k \xleftarrow{R} \mathcal{K},\ \ c_0 \xleftarrow{R} E(k, m),\ \ c_1 \xleftarrow{R} \mathcal{C},\ \ \ c \xleftarrow{R} c_b$$

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 psuedo-random ciphertext secure, then it is semantically secure.

(b) Show that the one-time pad is psuedo-random ciphertext secure.

(c) Give an example of a cipher that is semantically secure, but not psuedo-random ciphertext secure.

3.6. 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 is linear in $|\mathcal{S}|$.

3.7. 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?

3.8. Suppose $G(s)$ is a secure PRG that outputs bit-strings in $\{0,1\}^n$. Which of are 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) $G_4(s) := G(s)[0 \,..\, n-1]$.

(e) $G_5(s) := (G(s), G(s))$.

(f) $G_6(s_1 \parallel s_2) := (s_1, G(s_2))$.

3.9. 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.

3.10. 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).

3.11. Simple statistical distance calculations:

(a) Let $\mathbf{X}$ and $\mathbf{Y}$ be independent random variables, each uniformly distributed over $\mathbb{Z}_p$, where p is prime. Calculate $\Delta[\ (\mathbf{X}, \mathbf{Y}),\ (\mathbf{X}, \mathbf{XY})\ ]$.

(b) Let $\mathbf{X}$ and $\mathbf{Y}$ be random variables, each taking values in the interval $[0, t]$. Show that $|E[\mathbf{X}] - E[\mathbf{Y}]| \leq t\Delta[\mathbf{X}, \mathbf{Y}]$.

The following three exercises should be done together; they will be used in exercises in the following chapters.

3.12. This exercise develops another way of comparing two probability distributions, which considers ratios of probabilities, rather than differences. Let $\mathbf{X}$ and $\mathbf{Y}$ be two random variables taking values on a finite set $\mathcal{R}$, and assume that $\Pr[\mathbf{X} = r] > 0$ for all $r \in \mathcal{R}$. Define

$$\rho[\mathbf{X}, \mathbf{Y}] := \max\big\{\Pr[\mathbf{Y} = r] / \Pr[\mathbf{X} = r] : r \in \mathcal{R}\big\}$$

Show that for every subset $\mathcal{R}'$ of $\mathcal{R}$, we have $\Pr[\mathbf{Y} \in \mathcal{R}'] \leq \rho[\mathbf{X}, \mathbf{Y}] \cdot \Pr[\mathbf{X} \in \mathcal{R}']$.

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. Let $\mathcal{X}$ be a finite set of size N, and let $Q \leq N$. Define random variables $\mathbf{X}$ and $\mathbf{Y}$, where $\mathbf{X}$ is uniformly distributed over all sequences of Q elements in $\mathcal{X}$, and $\mathbf{Y}$ is uniformly distributed over all sequences of Q *distinct* elements in $\mathcal{X}$. Let $\Delta[\mathbf{X}, \mathbf{Y}]$ be the statistical distance between $\mathbf{X}$ and $\mathbf{Y}$, and let $\rho[\mathbf{X}, \mathbf{Y}]$ be defined as in Exercise 3.12. Using the previous exercise, prove the following:

(a) $\Delta[\mathbf{X}, \mathbf{Y}] = 1 - \prod_{i=0}^{Q-1}(1 - i/N) \leq \dfrac{Q^2}{2N}$,

(b) $\rho[\mathbf{X}, \mathbf{Y}] = \dfrac{1}{\prod_{i=0}^{Q-1}(1 - i/N)} \leq \dfrac{1}{1 - \frac{Q^2}{2N}}$ (assuming $Q^2 < 2N$).

3.15. 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 $s[0 \,..\, \ell/2 - 1] = 0^{\ell/2}$
    then output 0^ℓ
    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 $\mathrm{PRG}\mathsf{adv}[\mathcal{B}_0, G_0] = 2^{-\ell/2} - 2^{-\ell}$.

(b) Show that $\mathrm{PRG}\mathsf{adv}[\mathcal{A}_0, G_0'] \geq n2^{-\ell/2} - n(n+1)2^{-\ell}$.

(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*
>
> $$\mathrm{PRG}\mathsf{adv}[\mathcal{A}, G'] \leq cn \cdot \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G],$$
>
> *where G' is the n-wise parallel composition of G.*

3.16. 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'.
This construction is a special case of a more general construction discussed in Section 4.6.

3.17. Let $\mathcal{E} = (E, D)$ be a semantically secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$, where $\mathcal{M} = \{0,1\}$. Show that for every efficient adversary $\mathcal{A}$ that receives an encryption of a random bit b, the probability that $\mathcal{A}$ correctly predicts b is at most $1/2 + \epsilon$, where ϵ is negligible. This proves a converse of sorts to Theorem 2.8. Hint: use Lemma 3.5.

3.18. 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 nonnegligible. 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.

3.19. 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.

3.20. Let $G : \mathcal{S} \to \mathcal{R}$ 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.

3.21. 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.

3.22. Show that a secure PRG $G : \{0,1\}^n \to \mathcal{R}$ can become insecure if the seed is not uniformly random in $\mathcal{S}$.

- 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.
- Show that G' becomes insecure if its random seed $s_0 \parallel s_1$ is chosen so that its last bit is always 0.
- Construct a secure PRG $G'' : \{0,1\}^{n+1} \to \mathcal{R} \times \{0,1\}$ that becomes insecure if its seed s is chosen to that the *parity* of the bits in s is always 0.

3.23. 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. 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. 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. This exercise develops a simple fact from probability that is helpful in understanding the proof of Lemma 3.5. Let $\mathbf{X}$ and $\mathbf{Y}$ be independent random variables, taking values in S and T, respectively, where $\mathbf{Y}$ is uniformly distributed over T. Let $f : S \to \{0, 1\}$ and $g : S \to T$ be functions. Show that the events $f(\mathbf{X}) = 1$ and $g(\mathbf{X}) = \mathbf{Y}$ 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

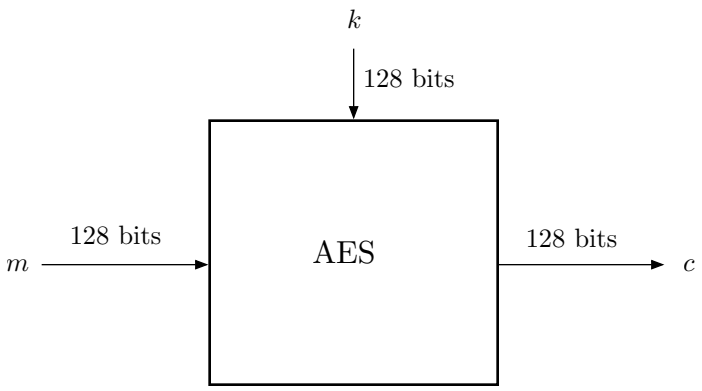


Figure 4.1: The block cipher AES

- 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 randomly chosen $k \in \mathcal{K}$, and 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 $b = 0$: $k \xleftarrow{R} \mathcal{K}$, $f \leftarrow E(k, \cdot)$;
  if $b = 1$: $f \xleftarrow{R} \text{Perms}[\mathcal{X}]$.

- 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 challenger 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 4.7).

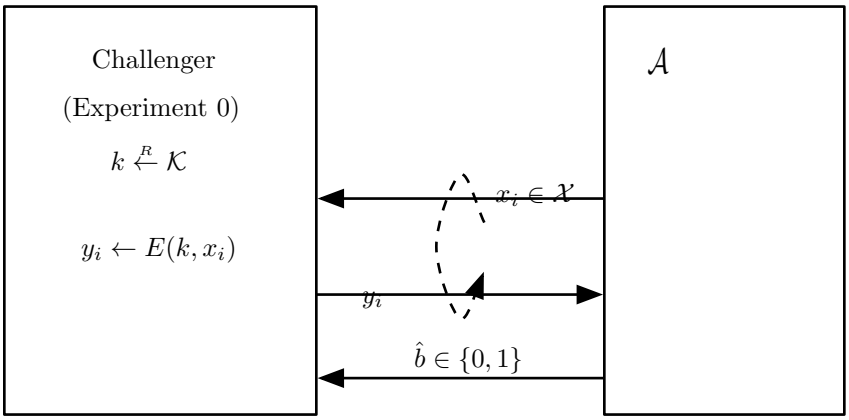
4.1.1 Some implications of security

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).

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 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.



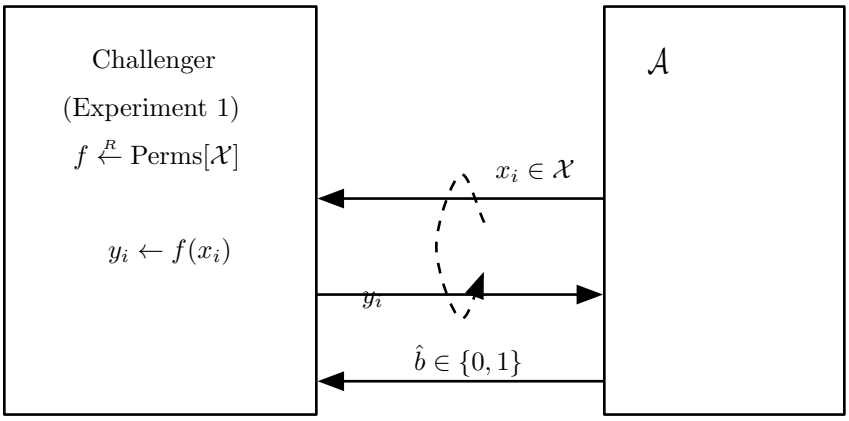


Figure 4.2: Attack Game 4.1

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 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.

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 $k \in \mathcal{K}$, and wins the game if $k = k$.

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 k, adversary $\mathcal{B}$ chooses an arbitrary $x_{Q+1} \notin \{x_1, \ldots, x_Q\}$, computes $y \leftarrow E(k, x_{Q+1})$, 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.

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 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

$$y_i = E(k, x_i) \quad \text{for} \quad i = 1, \ldots, Q. \tag{4.1}$$

So our adversary simply tries all possible keys to find one that satisfies (4.1). 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 (4.1). 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 (4.1).

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 ith 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;

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 4.10). We will not make use this notion in this text, other than an example application in a later chapter (Exercise **??**).

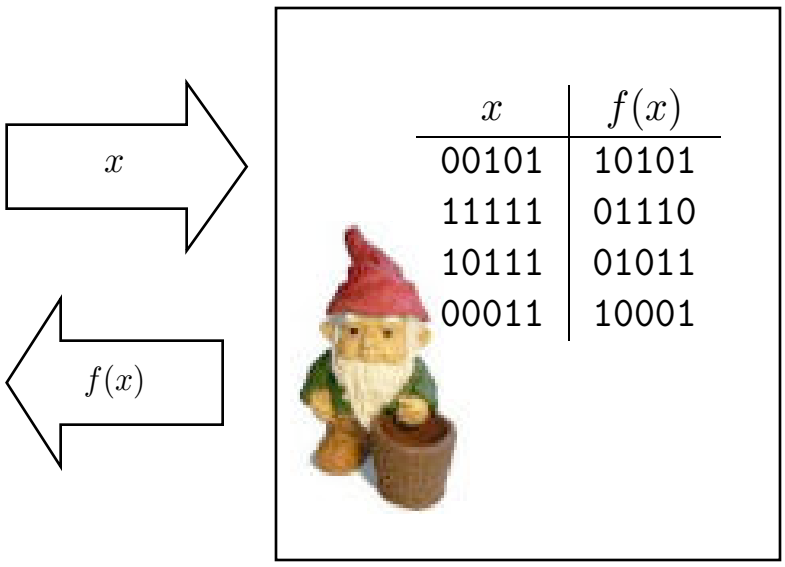


Figure 4.3: A faithful gnome implementing random permutation f

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

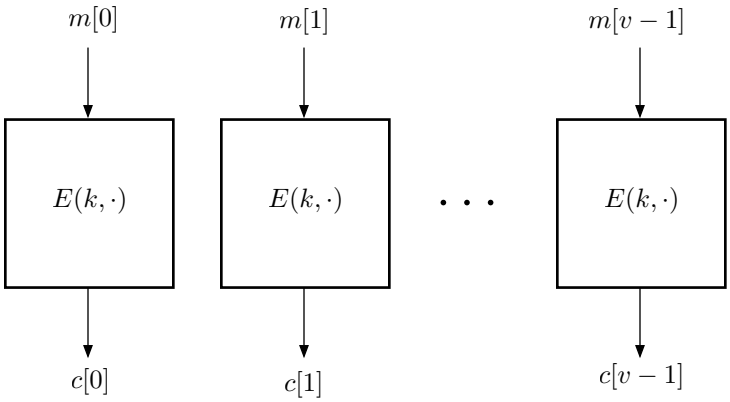
$$E'(k, m) := \big(E(k, m[0]), \ldots, E(k, m[v-1])\big).$$

- For $k \in \mathcal{K}$ and $c \in \mathcal{X}^{\leq \ell}$, with $v := |c|$, we define

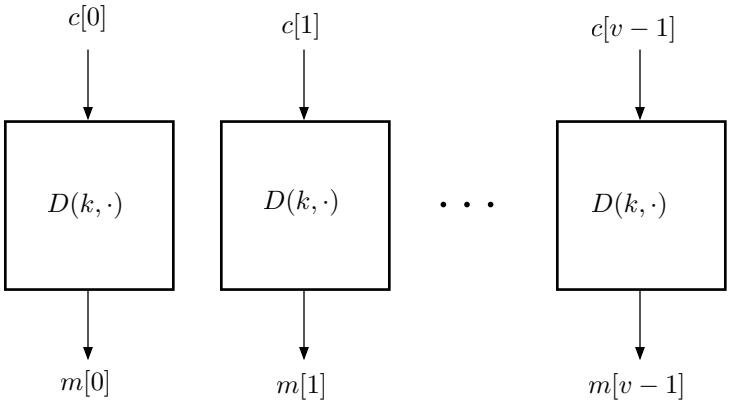
$$D'(k, m) := \big(D(k, c[0]), \ldots, E(k, c[v-1])\big).$$

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



(a) encryption



(b) decryption

Figure 4.4: Encryption and decryption for ECB mode

(a) plaintext

(b) plaintext encrypted in ECB mode using AES

Figure 4.5: Encrypting in ECB mode

— 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., $m_1[0] \neq m_1[1]$). 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 $\mathcal{A}$ SS adversary 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*
>
> $$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}'] = \mathrm{BCadv}[\mathcal{B}, \mathcal{E}].$$

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. Assume that $\mathcal{E}$ is defined over $(\mathcal{K}, \mathcal{X})$ and is a secure block cipher. 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 adversary attacking $\mathcal{E}'$. We shall consider the bit-guessing version of Attack Game 2.1, that is, Attack Game 2.4, discussed in Section 2.3.5. 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

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}] = |\Pr[W_0] - 1/2|. \tag{4.2}$$

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

$$|\Pr[W_0] - \Pr[W_1]| = \mathrm{BCadv}[\mathcal{B}, \mathcal{E}]. \tag{4.3}$$

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:
> $b \xleftarrow{R} \{0,1\}$
> $c \leftarrow (f(m_b[0]), \ldots, f(m_b[v-1]))$
> 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 $\delta(\hat{b}, b)$ (see (3.4)).

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 (4.3) 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

$$\Pr[W_1] = \Pr[W_2]. \tag{4.4}$$

Moreover, we claim that

$$\Pr[W_2] = 1/2. \tag{4.5}$$

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 (4.5) now follows.

Combining (4.2), (4.3), (4.4), and (4.5), we have

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}'] = \mathrm{BCadv}[\mathcal{B}, \mathcal{E}].$$

That completes the proof. □

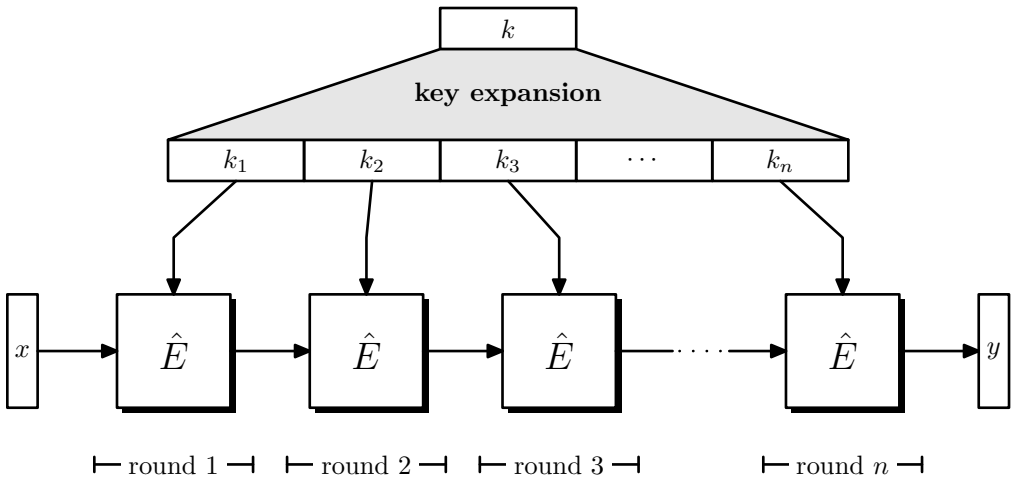


Figure 4.6: Encryption in a real-world block cipher

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 2.4. As usual, Definition 4.1 needs 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 $\mathsf{BCadv}[\mathcal{A}, \mathcal{E}]$ 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	27
3DES	168	64	48	9
AES-128	128	128	10	61
AES-256	256	128	14	?

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] Crypto++ library on Pentium 4, 2.1 GHz.

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 lead 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. 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.

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

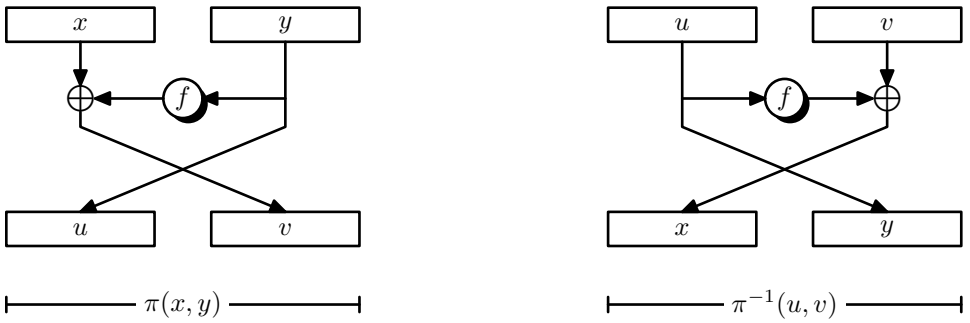


Figure 4.7: The Feistel permutation

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:

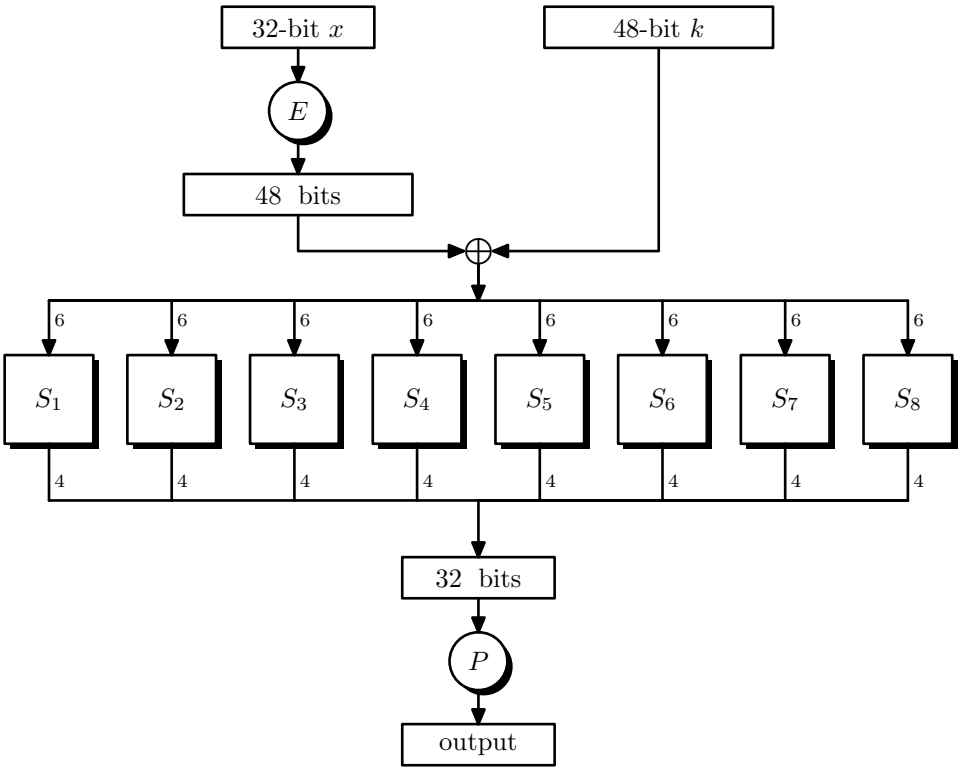


Figure 4.8: The DES round function $F(k, x)$

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: $t := t_1 \,\|\, \cdots \,\|\, t_8$
    for $i = 1$ to 8 : $\quad s_i \leftarrow S_i(t_i)$
    $s \leftarrow s_1 \,\|\, \cdots \,\|\, s_8 \quad \in \{0,1\}^{32}$
    $y \leftarrow P(s) \quad \in \{0,1\}^{32}$
    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 [21], 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 [21]:

> 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 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 of 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 [21].

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.

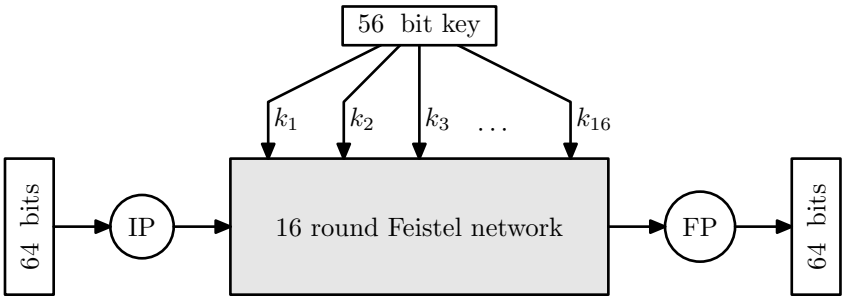


Figure 4.9: The complete DES circuit

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, [21] 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) 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 $k \in \mathcal{K}$ 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 is woefully inadequate.

To prove that exhaustive search on DES is feasible, RSA data security setup 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 $y_1, y_2, \ldots, y_n$ where the first three outputs, y_1, y_2, y_3, were the result of applying DES to the 24-byte plaintext message:

$$\underbrace{\texttt{The unknown message is:}}_{x_1 \qquad x_2 \qquad x_3}$$

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 $y_4 \ldots y_n$.

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 $y_4 \ldots y_n$ 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 off the shelf 120 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 [34].

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.

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 withing reach, as discussed in [11].

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 4.6.

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 then 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.

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 $i = 1, \ldots, Q$ and outputs a key pair $(k_1, k_2) \in \mathcal{K}^2$ such that*

$$y_i = E_2\big(\ (k_1, k_2),\ x_i\big) \quad \textit{for all } i = 1, \ldots, Q. \tag{4.6}$$

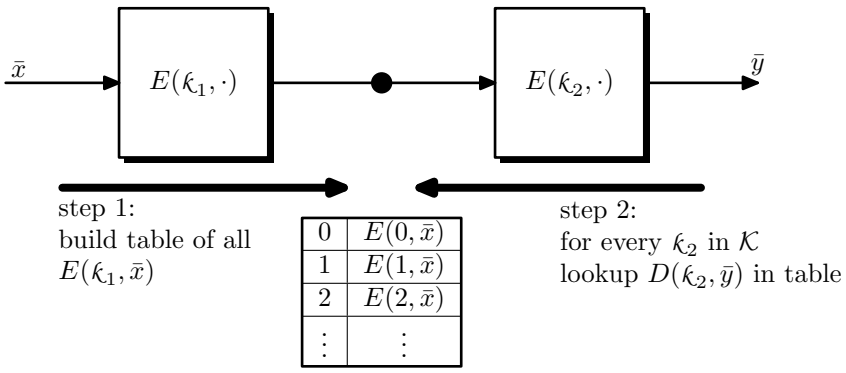


Figure 4.10: Meet in the middle attack on $2\mathcal{E}$

Its running time is dominated by a total of $2Q \cdot |\mathcal{K}|$ *evaluations of algorithms* E *and* D*.*

Proof. Let $\bar{x} := (x_1, \ldots, x_Q)$ and $\bar{y} := (y_1, \ldots, y_Q)$. To simplify the notation let us write

$$\bar{y} = E_2\big((k_1, k_2), \bar{x}\big) = E(k_2,\ E(k_1, \bar{x}))$$

to capture the Q relations in (4.6). We can write this as

$$D(k_2, \bar{y}) = E(k_1, \bar{x}) \tag{4.7}$$

To find a pair (k_1, k_2) satisfying (4.7) the algorithm $\mathcal{A}_{\text{EX}}$ does the following:

step 1: construct a table T containing all pairs $\big(k_1,\ E(k_1, \bar{x})\ \big)$ for all $k_1 \in \mathcal{K}$
step 2: for all $k_2 \in \mathcal{K}$ do:
$\bar{x} \leftarrow D(k_2, \bar{y})$
table lookup: if T contains a pair $(\cdot,\ \bar{x})$ then
let $(k_1, \bar{x})$ be that pair and output (k_1, k_2) and halt

This meet in the middle attack is depicted in Fig. 4.10. By construction, the pair (k_1, k_2) output by the algorithm must satisfy (4.7), a 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 (4.6), 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 short ciphertext. Overall this amounts to about 2^{60} bytes or 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 (i.e., applications in Chapter 6). By the mid-1990s it became apparent that a new federal block cipher standard is 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 proposals, many of which were developed outside of the United Stated. 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 standard should be chosen as the AES. In October of 2000, NIST announced that **Rijndael**, a Belgian block cipher, had been selected as the AES cipher. The 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 [22]. 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.

The AES algorithm

Like most 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:

cipher name	key-size (bits)	number of rounds
AES-128	128	10
AES-192	192	12
AES-256	256	14

For example, the structure of the cipher AES-128 with its ten rounds is shown in Fig. 4.11. Here f_{AES} is a fixed invertible 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 f_{AES}. This is repeated 9 times until in the last round a slightly modified function $\hat{f}_{AES}$ 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.

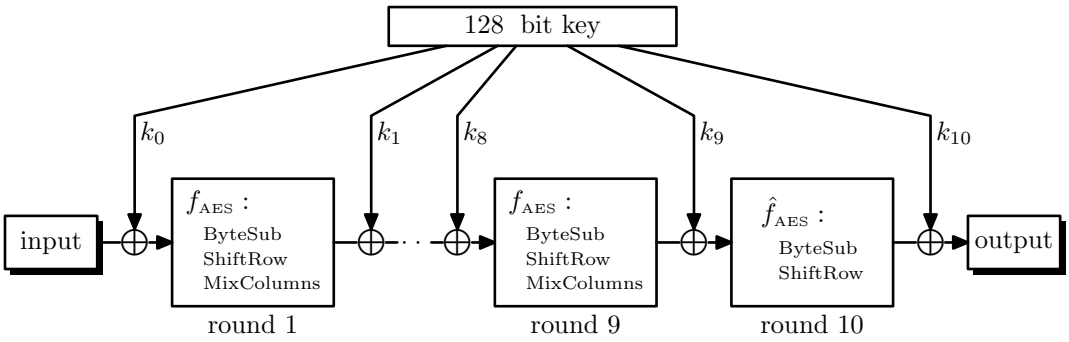


Figure 4.11: Schematic of the AES-128 block cipher

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 function f_{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 AES round function f_{AES} and the AES key expansion PRG. We describe each in turn.

The AES round function. An AES round function f_{AES} is made up of a sequence of three invertible operations on the set $\{0,1\}^{128}$. The input 128-bits is organized as a 4×4 array of cells, where each cell is 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 invertible function. This function is applied to each of the 16 cells, one cell at a time. The function S is hard-coded in the AES standard. 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:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \Longrightarrow \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{pmatrix} \tag{4.8}$$

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 GF(2^8). Elements in the field GF(2^8) are represented as polynomials over 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:

$$\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \times \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{pmatrix} \Longrightarrow \begin{pmatrix} a'_{0,0} & a'_{0,1} & a'_{0,2} & a'_{0,3} \\ a'_{1,1} & a'_{1,2} & a'_{1,3} & a'_{1,0} \\ a'_{2,2} & a'_{2,3} & a'_{2,0} & a'_{2,1} \\ a'_{3,3} & a'_{3,0} & a'_{3,1} & a'_{3,2} \end{pmatrix} \tag{4.9}$$

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 function f_{AES} used in the AES circuit of Fig. 4.11 is the sequential composition of the three functions `SubBytes`, `ShiftRows`, and `MixColumns` in that order. In the very last round AES uses a slightly different function we call $\hat{f}_{AES}$. This function is the same as f_{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 [25].

Because each step in f_{AES} is easily invertible, the entire function f_{AES} is easily invertible, as required for decryption.

Implementing AES using pre-computed tables. The AES round function is built from a function we called f_{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 f_{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 f_{AES} takes as input a 4×4 matrix $A = (a_{i,j})$ and outputs a matrix $A' := f_{AES}(A)$ of the same dimensions. Let us use $S[a]$ to denote the result of applying `SubBytes` to 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 (4.9), we can write the four columns of the output of $f_{AES}(A)$ as:

$$\begin{aligned} A'[0] &= M[0] \cdot S[a_{0,0}] + M[1] \cdot S[a_{1,1}] + M[2] \cdot S[a_{2,2}] + M[3] \cdot S[a_{3,3}] \\ A'[1] &= M[0] \cdot S[a_{0,1}] + M[1] \cdot S[a_{1,2}] + M[2] \cdot S[a_{2,3}] + M[3] \cdot S[a_{3,0}] \\ A'[2] &= M[0] \cdot S[a_{0,2}] + M[1] \cdot S[a_{1,3}] + M[2] \cdot S[a_{2,0}] + M[3] \cdot S[a_{3,1}] \\ A'[3] &= M[0] \cdot S[a_{0,3}] + M[1] \cdot S[a_{1,0}] + M[2] \cdot S[a_{2,1}] + M[3] \cdot S[a_{3,2}] \end{aligned} \tag{4.10}$$

where addition and multiplication is done in $\mathrm{GF}(2^8)$. Each column $M[i]$, $i = 0, 1, 2, 3$, is a vector of four bytes over $\mathrm{GF}(2^8)$ while the quantities $S[a_{i,i}]$ are 1-byte scalars in $\mathrm{GF}(2^8)$.

Every term in (4.10) 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 (4.10) gives a fast way to evaluate the function $f_{AES}(A)$:

$$\begin{aligned}
A'[0] &= T_0[a_{0,0}] + T_1[a_{1,1}] + T_2[a_{2,2}] + T_3[a_{3,3}] \\
A'[1] &= T_0[a_{0,1}] + T_1[a_{1,2}] + T_2[a_{2,3}] + T_3[a_{3,0}] \\
A'[2] &= T_0[a_{0,2}] + T_1[a_{1,3}] + T_2[a_{2,0}] + T_3[a_{3,1}] \\
A'[3] &= T_0[a_{0,3}] + T_1[a_{1,0}] + T_2[a_{2,1}] + T_3[a_{3,2}]
\end{aligned}$$

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. For completeness we note that the circular structure of the matrix M lets us compress the four tables to only 2KB with little impact on performance.

The one exception to (4.10) 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 f_{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 $g_i : \{0,1\}^{32} \rightarrow \{0,1\}^{32}$ 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 $\mathrm{GF}(2^8)$ 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 [17]. 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. None of these attacks impact the security of either AES variant.

- **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 [12]. 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 implementations of AES. Experiments by Emilia Käsper in 2009 show that on Intel Core 2 processors 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 Haswell processors introduced in 2013 the `AESENC`, `AESDEC` and `AESENCLAST` instructions each take seven 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 seven stages. Seven AES blocks can be processed concurrently by different stages of the pipeline. While processing a single AES-128 block takes (7 cycles) $\times$ (10 rounds) = 70 cycles (or 4.38 cycles/byte), processing seven blocks in a pipeline takes only 77 cycles (or 0.69 cycles/byte). Hence, pipelining can speed up AES by a factor of seven. 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 [66] 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 [46, 45], 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 $S \subseteq \{0, \ldots, n-1\}$ we use $m[S]$ to denote the XOR of the bits in positions in S. That is, if $S = \{i_1, \ldots, i_\ell\}$ 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 $S_0, S_1 \subseteq \{0, \ldots, n-1\}$ and $S_2 \subseteq \{0, \ldots, h-1\}$, such that *for all* keys $k \in \mathcal{K}$ and *for randomly chosen* $m \in \mathcal{M}$, we have

$$\Pr\Big[\, m[S_0] \oplus E(k,m)[S_1] = k[S_2] \,\Big] \geq \frac{1}{2} + \epsilon \tag{4.11}$$

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 (4.11) 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 (4.11) 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 (4.11) 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*

$$\Pr\Big[\ k[S_2] = \text{Majority}_{i=1}^{t}(m_i[S_0] \oplus c_i[S_1])\ \Big] \geq 1 - e^{-t\epsilon^2/2}\ . \tag{4.12}$$

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 Chernoff bound (Theorem **??**).

The bound in (4.12) 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:

$$\begin{aligned} m_{\text{R}}[17, 18, 24] \oplus c_{\text{L}}[7, 18, 24, 29] \oplus c_{\text{R}}[15] &= k[S_{\text{e}}] \\ c_{\text{R}}[17, 18, 24] \oplus m_{\text{L}}[7, 18, 24, 29] \oplus m_{\text{R}}[15] &= k[S_{\text{d}}] \end{aligned} \tag{4.13}$$

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 (4.13) the following relations on the entire 16-round DES circuit:

$$\Big(m_{\text{L}} \oplus F(k_1, m_{\text{R}})\Big)[17, 18, 24] \oplus c_{\text{R}}[7, 18, 24, 29] \oplus \Big(c_{\text{L}} \oplus F(k_{16}, c_{\text{R}})\Big)[15] = k[S'_{\text{e}}] \tag{4.14}$$

$$\Big(c_{\text{L}} \oplus F(k_{16}, c_{\text{R}})\Big)[17, 18, 24] \oplus m_{\text{R}}[7, 18, 24, 29] \oplus \Big(m_{\text{L}} \oplus F(k_1, m_{\text{R}})\Big)[15] = k[S'_{\text{d}}] \tag{4.15}$$

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 (4.14). 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 (4.14) 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 (4.14), evaluated at a random plaintext/ciphertext pair, exhibits a bias of about 2^{-21} towards the bit $k[S'_{\text{e}}]$. When the 12 key bits of the key are set incorrectly one assumes that the bias in (4.14) 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 (4.14). That is, evaluate the left hand side of (4.14) 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 (4.15) 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 (4.14) 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 (4.14) 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 (4.14) for any candidate key. Two plaintext/ciphertext pairs that agree on these 13 bits will always result in the same value for (4.14). 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 [38] 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 [38].

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 [51], and even sound [31].

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 130. 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 $A = (a_{i,j})$ denote the 4×4 input to the last round, and letting $(w_{i,j})$ denote the 4×4 last round key, the final AES output is computed as the 4×4 matrix:

$$C = (c_{i,j}) = \begin{pmatrix} S[a_{0,0}] + w_{0,0} & S[a_{0,1}] + w_{0,1} & S[a_{0,2}] + w_{0,2} & S[a_{0,3}] + w_{0,3} \\ S[a_{1,1}] + w_{1,0} & S[a_{1,2}] + w_{1,1} & S[a_{1,3}] + w_{1,2} & S[a_{1,0}] + w_{1,3} \\ S[a_{2,2}] + w_{2,0} & S[a_{2,3}] + w_{2,1} & S[a_{2,0}] + w_{2,2} & S[a_{2,1}] + w_{2,3} \\ S[a_{3,3}] + w_{3,0} & S[a_{3,0}] + w_{3,1} & S[a_{3,1}] + w_{3,2} & S[a_{3,2}] + w_{3,3} \end{pmatrix} \tag{4.16}$$

The attacker is given this final output C.

To mount the attack, consider two consecutive entries in the output matrix C, say $c_{0,0} = S[a_{0,0}] + w_{0,0}$ and $c_{0,1} = S[a_{0,1}] + w_{0,1}$. Subtracting one equation from the other we see that when $a_{0,0} = a_{0,1}$ the following relation holds:

$$c_{0,0} - c_{0,1} = w_{0,0} - w_{0,1} \ .$$

Therefore, with $\Delta := w_{0,0} - w_{0,1}$ we have that $c_{0,0} - c_{0,1} = \Delta$ whenever $a_{0,0} = a_{0,1}$. Moreover, when $a_{0,0} \neq a_{0,1}$ the structure of the S table ensures that $c_{0,0} - c_{0,1} \neq \Delta$.

The key insight is that whenever $a_{0,0} = a_{0,1}$, reading $S[a_{0,0}]$ loads the $a_{0,0}$ entry of S into the L1 cache so that the second access to this entry via $S[a_{0,1}]$ is much faster. However, when $a_{0,0} \neq a_{0,1}$ it is possible that both reads miss the L1 cache so that both are slow. Therefore, when $a_{0,0} = a_{0,1}$ the expected running time of the entire AES cipher is slightly less than when $a_{0,0} \neq a_{0,1}$.

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 $c_{0,0} - c_{0,1} = \Delta$. 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 $\Delta = w_{0,0} - w_{0,1}$. Hence, timing information reveals one linear relation about the last round key: $w_{0,0} - w_{0,1} = \Delta$.

Suppose the implementation evaluates the terms of (4.16) in some sequential order. Repeating the timing procedure above for different consecutive pairs $c_{i,j}$ and $c_{u,v}$ 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 [19], 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.

Power attacks on AES implementations

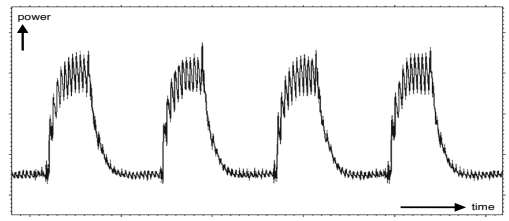
The amount of power consumed by a device as it operates can leak information about the inner-workings of the device, including secret keys stored on the device. Let us see how an attacker can use power measurements to quickly extract secret keys from a physical device.

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.

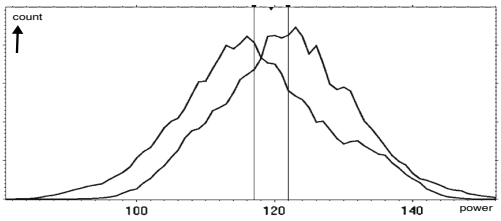
Since the embedded chip must draw power from the terminal (it has no internal power source) it is quite easy for the terminal to measure the amount of power consumed by the chip at any given time. In particular, an attacker can measure the amount of power consumed as the AES algorithm is evaluated. Fig. 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 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).

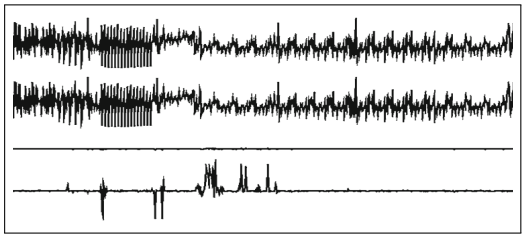
The attack of the previous paragraph, called **simple power analysis** (SPA), will not work on AES: during encryption the secret AES round keys are simply XORed into the cipher state. The power used by the XOR instruction only marginally depends on its operands and therefore



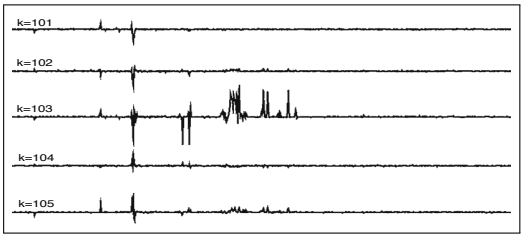
(a) Power used by four iterations of AES



(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. [40])

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 [37].

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 [37]. 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 **??** 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 using classical computers and therefore one would expect that once quantum computers are built they will also be capable of carrying out this scale of computations. As a result, once quantum computers are built, AES-128 will be considered insecure.

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 quantum computers is one reason why AES supports 256-bits keys. 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.

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 [33]. The result says the following: suppose we are given a function $f : \mathcal{K} \to \{0, 1\}$ defined as follows

$$f(k) = \begin{cases} 1 & \text{if } k = k_0 \\ 0 & \text{otherwise} \end{cases} \tag{4.17}$$

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 worse 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 (4.17). 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 block 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 $b = 0$: $k \xleftarrow{R} \mathcal{K}$, $f \leftarrow F(k, \cdot)$;
  if $b = 1$: $f \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{Y}]$.

- 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

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \tag{4.18}$$

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 challenger 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 4.7).

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 (4.18).

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 of 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 ith 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 $\mathrm{Funs}[\mathcal{X}, \mathcal{X}]$. 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, $\mathrm{PRFadv}[\mathcal{A}, E] \geq 1/2$, 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 $\min\{Q(Q-1)/4N, 0.63\}$. 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*

$$\Big|\mathrm{BCadv}[\mathcal{A}, \mathcal{E}] - \mathrm{PRFadv}[\mathcal{A}, E]\Big| \leq Q^2/2N.$$

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

$$\Big|\mathrm{BCadv}[\mathcal{A}, \mathcal{E}] - \mathrm{PRFadv}[\mathcal{A}, E]\Big| \leq Q^2/2N$$

Since N is super-poly and Q is poly-bounded, the value $Q^2/2N$ is negligible (see Fact 2.6). It follows that $\mathrm{BCadv}[\mathcal{A}, \mathcal{E}]$ is negligible if and only if $\mathrm{PRFadv}[\mathcal{A}, E]$ 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 $f \in \text{Funs}[\mathcal{X}, \mathcal{X}]$ as follows:

  if $b = 0$: $f \xleftarrow{R} \text{Perms}[\mathcal{X}]$;
  if $b = 1$: $f \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{X}]$.

- 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

$$\text{PFadv}[\mathcal{A}, \mathcal{X}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

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*

$$\text{PFadv}[\mathcal{A}, \mathcal{X}] \leq Q^2/2N.$$

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,

$$|p_1 - p_0| = \text{BCadv}[\mathcal{A}, \mathcal{E}],$$
$$|p_2 - p_0| = \text{PRFadv}[\mathcal{A}, E],$$

and that by Theorem 4.6,

$$|p_2 - p_1| = \text{PFadv}[\mathcal{A}, \mathcal{X}] \leq Q^2/2N.$$

Putting these together, we get

$$\Big|\text{BCadv}[\mathcal{A}, \mathcal{E}] - \text{PRFadv}[\mathcal{A}, E]\Big| = \Big||p_1 - p_0| - |p_2 - p_0|\Big| \leq |p_2 - p_1| \leq Q^2/2N,$$

which proves the theorem. □

So it remains to prove Theorem 4.6. Before doing so, we state and prove a very simple, but extremely useful fact:

Theorem 4.7 (Difference Lemma). *Let Z, W_0, W_1 be events defined over some probability space. Suppose that $W_0 \wedge \bar{Z}$ occurs if and only if $W_1 \wedge \bar{Z}$ occurs. Then we have*

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z].$$

Proof. This is a simple calculation. We have

$$\begin{aligned} |\Pr[W_0] - \Pr[W_1]| &= |\Pr[W_0 \wedge Z] + \Pr[W_0 \wedge \bar{Z}] - \Pr[W_1 \wedge Z] - \Pr[W_1 \wedge \bar{Z}]| \\ &= |\Pr[W_0 \wedge Z] - \Pr[W_1 \wedge Z]| \\ &\leq \Pr[Z]. \end{aligned}$$

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 ith 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 ith 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

$$\mathrm{PFadv}[\mathcal{A}, \mathcal{X}] = |\Pr[W_0] - \Pr[W_1]|.$$

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 4.22), 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 very 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*
>
> $$\mathrm{PRGadv}[\mathcal{A}, G] = \mathrm{PRFadv}[\mathcal{B}, F].$$

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. □

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.

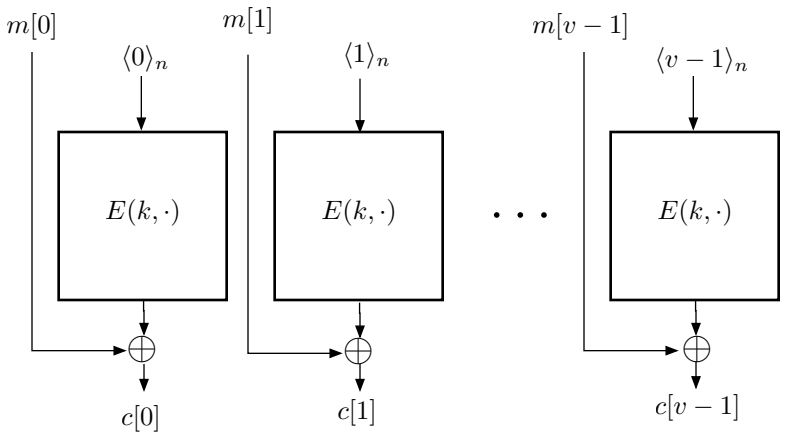
Let us consider this stream cipher in detail. This cipher $\mathcal{E}' = (E', D')$ has key space $\mathcal{K}$, and message and ciphertext space $\mathcal{X}^{\leq \ell}$, where ℓ is a poly-bounded value, and in particular, $\ell \leq N$. We can define $x_1, \ldots, x_\ell$ to be any convenient elements 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

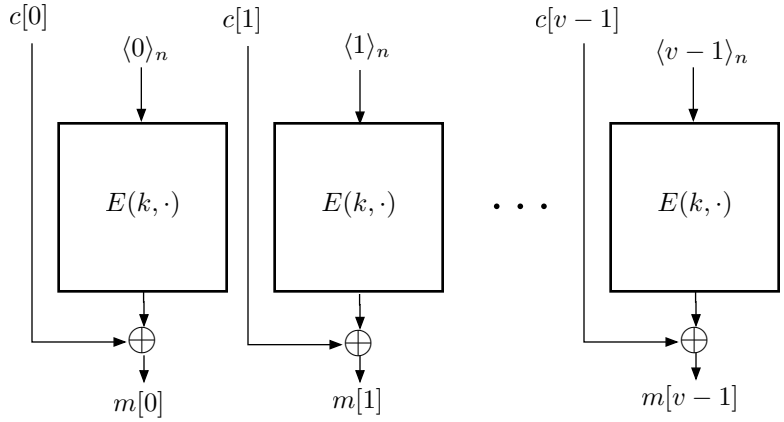
$$E'(k, m) := \big(E(k, \langle 0 \rangle_n) \oplus m[0], \ldots, E(k, \langle v-1 \rangle_n) \oplus m[v-1]\big).$$

- For $k \in \mathcal{K}$ and $c \in \mathcal{X}^{\leq \ell}$, with $v := |c|$, we define

$$D'(k, c) := \big(E(k, \langle 0 \rangle_n) \oplus c[0], \ldots, E(k, \langle v-1 \rangle_n) \oplus c[v-1]\big).$$



(a) encryption



(b) decryption

Figure 4.13: Encryption and decryption for deterministic counter mode

This mode of operation 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

$$\mathrm{SSadv}^*[\mathcal{A}, \mathcal{E}'] \leq \mathrm{BCadv}[\mathcal{B}, \mathcal{E}] + \ell^2/2N. \tag{4.19}$$

Clearly, deterministic counter mode has the advantage over ECB mode 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 $\ell^2/2N$ term in (4.19). 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 $\min\{\ell(\ell-1)/4N, 0.63\}$. 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 2.4.

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:*

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{X} = \{\mathcal{X}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \textit{and} \quad \mathbf{Y} = \{\mathcal{Y}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

such that

1. $\mathbf{K}$, $\mathbf{X}$, *and* $\mathbf{Y}$ *are efficiently recognizable.*
2. $\mathbf{K}$ *and* $\mathbf{Y}$ *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 (after its inventors). 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 some practical block ciphers as Feistel networks (see Section 4.2.1).

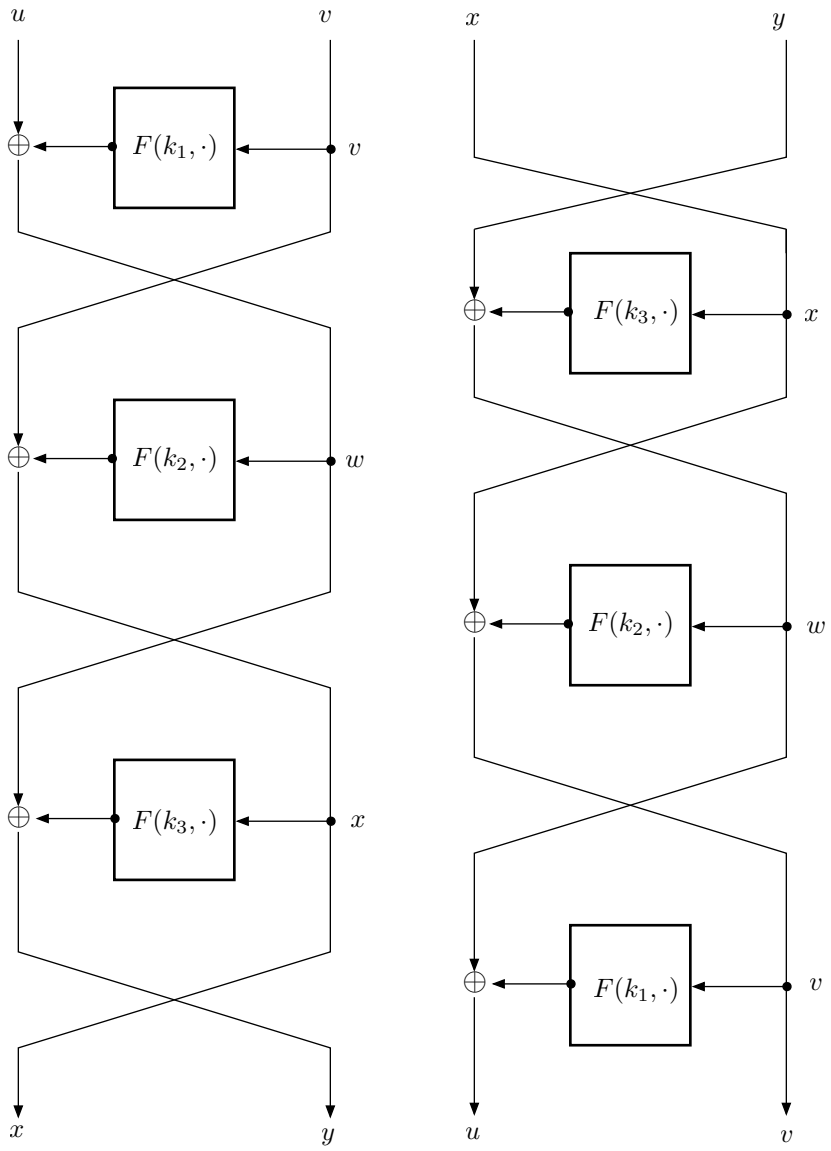
Let F be a PRF, defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, where $\mathcal{X} = \{0,1\}^n$. We describe a block cipher $\mathcal{E} = (E, D)$ whose key space is $\mathcal{K}^3$, and whose data block space is $\mathcal{X}^2$.

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 an 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).



(a) Encryption (b) Decryption

Figure 4.14: Encryption and decryption with Luby-Rackoff

See Fig. 4.14 for an illustration of $\mathcal{E}$.

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

$$D((k_1, k_2, k_3), \cdot) = \phi_{k_1}^{-1} \circ \phi_{k_2}^{-1} \circ \phi_{k_3}^{-1} = \sigma \circ \phi_{k_1} \circ \phi_{k_2} \circ \phi_{k_3} \circ \sigma.$$

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*
>
> $$\mathrm{BC}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq 3 \cdot \mathrm{PR}\mathsf{Fadv}[\mathcal{B}, F] + \frac{Q^2}{N} + \frac{Q^2}{2N^2}.$$

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 $\text{PRFadv}[\mathcal{A}, E]$ 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 $\mathcal{A}$);
- $\text{PRFadv}[\mathcal{A}_0, E] = \text{PRFadv}[\mathcal{A}, E]$.

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

$$|\Pr[W_3] - \Pr[W_0]| = \text{PRFadv}[\mathcal{A}_0, E]$$

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 ith 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 ith 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 4.24, 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

$$|\Pr[W_1] - \Pr[W_0]| = \mathrm{PRFadv}[\mathcal{B}', F']. \tag{4.20}$$

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

$$\begin{aligned} w_i &\leftarrow u_i \oplus f'(1, v_i) \\ x_i &\leftarrow v_i \oplus f'(2, w_i) \\ y_i &\leftarrow w_i \oplus f'(3, x_i). \end{aligned}$$

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 (4.20) follows.

By Exercise 4.24, there exists an adversary $\mathcal{B}$, just as efficient as $\mathcal{B}'$, such that

$$\mathrm{PRFadv}[\mathcal{B}', F'] = 3 \cdot \mathrm{PRFadv}[\mathcal{B}, F]. \tag{4.21}$$

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:

$f_1 \xleftarrow{R} \mathrm{Funs}[\mathcal{X}, \mathcal{X}]$
$X_1, \ldots, X_Q \xleftarrow{R} \mathcal{X}$
$Y_1, \ldots, Y_Q \xleftarrow{R} \mathcal{X}$
upon receiving the ith 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 completely equivalent to that of Game 1, we have

$$\Pr[W_2] = \Pr[W_1]. \tag{4.22}$$

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:

$f_1 \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{X}]$
$X_1, \ldots, X_Q \xleftarrow{R} \mathcal{X}$
$Y_1, \ldots, Y_Q \xleftarrow{R} \mathcal{X}$
upon receiving the ith 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

$$Coins, \quad f_1, \quad X_1, \ldots, X_Q, \quad Y_1, \ldots, Y_Q$$

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

$$Coins, \quad f_1, \quad X_1, \ldots, X_Q, \quad Y_1, \ldots, Y_Q$$

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

$$|\Pr[W_3] - \Pr[W_2]| \leq \Pr[Z] \leq \Pr[Z_1] + \Pr[Z_2]. \tag{4.23}$$

By the fact that $x_1, \ldots, x_Q$ are mutually independent (see Claim 1), it is obvious that

$$\Pr[Z_2] \leq \frac{Q^2}{2} \cdot \frac{1}{N}, \tag{4.24}$$

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

$$\Pr[Z_1] \leq \frac{Q^2}{2} \cdot \frac{1}{N}. \tag{4.25}$$

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 $\mathrm{Funs}[\mathcal{X}, \mathcal{X}]$ 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 $f_2(v_j)$ 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 (4.25) 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 (4.20), (4.21), (4.22), (4.23), (4.24), and (4.25), we conclude that

$$\mathrm{PRFadv}[\mathcal{A}_0, E] \leq 3 \cdot \mathrm{PRFadv}[\mathcal{B}, F] + \frac{Q^2}{N}.$$

Finally, applying Theorem 4.4 to the cipher $\mathcal{E}$, whose data block space has size N^2, we have

$$\mathrm{BCadv}[\mathcal{A}, \mathcal{E}] \leq 3 \cdot \mathrm{PRFadv}[\mathcal{B}, F] + \frac{Q^2}{N} + \frac{Q^2}{2N^2}.$$

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 $x = (a_1, \ldots, a_n) \in \{0,1\}^*$, where $a_i \in \{0,1\}$ for $i = 1, \ldots, n$, and outputs an element $t \in \mathcal{S}$, computed as follows:

$t \leftarrow s$
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 the $F(s, x)$ 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*
>
> $$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F] = \ell Q \cdot \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G].$$

[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.

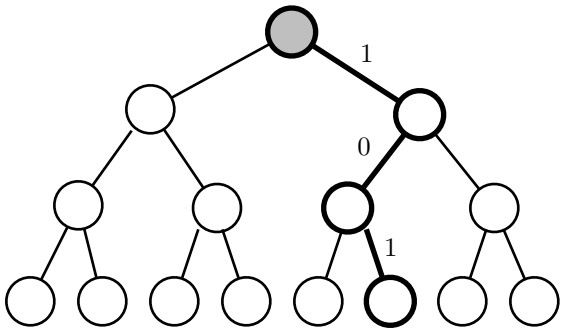


Figure 4.15: Evaluation tree for $\ell = 3$. The highlighted path corresponds to the input $x = 101$. The root is shaded to indicate it is assigned a random label. All other nodes are assigned derived labels.

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:
    $u \leftarrow (a_1, \ldots, a_j)$, $v \leftarrow (a_{j+1}, \ldots, \ell)$
    $y \leftarrow G^*(f(u), v)$
    send y to $\mathcal{A}$.

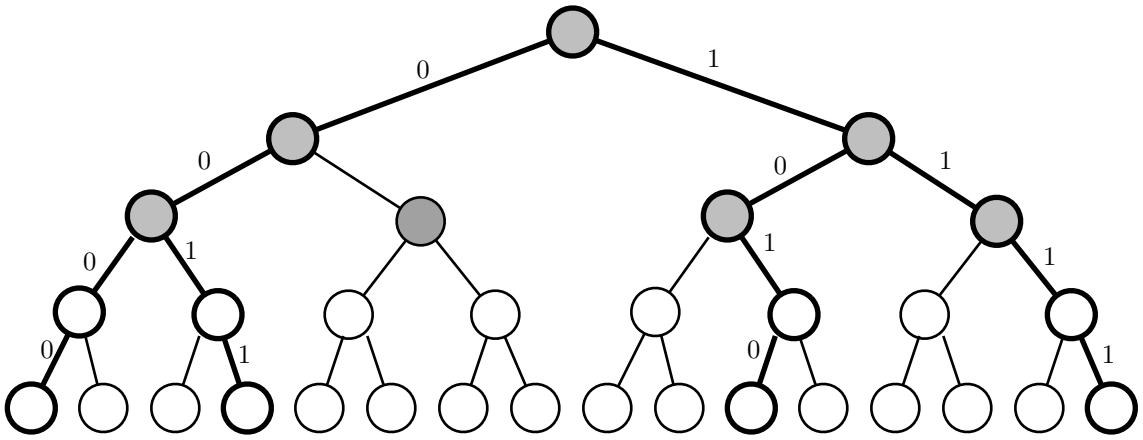


Figure 4.16: Evaluation tree for Hybrid 2 with $\ell = 4$. The shaded nodes are assigned random labels, while the unshaded nodes are assigned derived labels. The highlighted paths correspond to inputs 0000, 0011, 1010, and 1111.

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:

$$\mathrm{PRFadv}[\mathcal{A}, F] = |p_\ell - p_0|. \tag{4.26}$$

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 PRF, then so is G'.

We now build an efficient PRG adversary $\mathcal{B}'$ that attacks G', such that

$$\mathrm{PRGadv}[\mathcal{B}', G'] = \frac{1}{\ell} \cdot |p_\ell - p_0|. \tag{4.27}$$

We first give an overview of how $\mathcal{B}'$ works. In playing Attack Game 4.2 with respect to G', the challenger presents to $\mathcal{B}'$ a vector

$$\vec{r} = ((r_{10}, r_{11}), \ldots, (r_{Q0}, r_{Q1})) \in (\mathcal{S}^2)^Q. \tag{4.28}$$

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 (4.27).

Now the details. We implement our lookup table as an associative array *Map*, indexed by elements of $\{0,1\}^*$, and whose entries are positive integers. Here is the logic for $\mathcal{B}'$:

> upon receiving $\vec{r}$ as in (4.28) from its challenger, $\mathcal{B}'$ plays the role of challenger to $\mathcal{A}$, as follows:
>
> $\omega \xleftarrow{R} \{1,\ldots,\ell\}$
> initialize an empty associative array *Map*
> $ctr \leftarrow 0$
> 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 4.2 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 (4.27) now follows by a familiar, telescoping sum calculation:

$$\begin{aligned}
\text{PRGadv}[\mathcal{B}', G'] &= |\Pr[W_1] - \Pr[W_0]| \\
&\frac{1}{\ell} \cdot \Big| \sum_{j=1}^{\ell} \Pr[W_1 \mid \omega = j] - \sum_{j=1}^{\ell} \Pr[W_0 \mid \omega = j] \Big| \\
&= \frac{1}{\ell} \cdot \Big| \sum_{j=1}^{\ell} p_j - \sum_{j=1}^{\ell} p_{j-1} \Big| \\
&= \frac{1}{\ell} \cdot |p_\ell - p_0|.
\end{aligned}$$

Finally, by Theorem 3.2, there exists an efficient PRG adversary $\mathcal{B}$ such that

$$\text{PRGadv}[\mathcal{B}', G'] = Q \cdot \text{PRGadv}[\mathcal{B}, G]. \tag{4.29}$$

The theorem now follows by combining equations (4.26), (4.27), and (4.29). □

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, $v = u \,\|\, w$ 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 $\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, F]$. Further, let us say that F is a **prefix-free secure PRF** if $\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, F]$ 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*
>
> $$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, \tilde{F}] = \ell Q \cdot \mathrm{PRG}\mathsf{adv}[\mathcal{B}, G].$$

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.

[3] For sequences $x = (a_1 \ldots a_s)$ and $y = (b_1 \ldots b_t)$, if $s \leq t$ and $a_i = b_i$ for $i = 1, \ldots, s$, then we say that x is a **prefix** of y; moreover, if $s < t$, then we say x is a **proper prefix** of y.

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:

$f \xleftarrow{R} \text{Funs}[\{0,1\}^{\leq j}, \mathcal{S}]$

upon receiving a query $x = (a_1, \ldots, a_n) \in \{0,1\}^{\leq \ell}$ from $\mathcal{A}$ do:
    if $n < j$ then
        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 (4.28) from its challenger, $\mathcal{B}'$ plays the role of challenger to $\mathcal{A}$, as follows:

$\omega \xleftarrow{R} \{1, \ldots, \ell\}$
initialize an empty associative array *Map*
$ctr \leftarrow 0$
upon receiving a query $x = (a_1, \ldots, a_n) \in \{0,1\}^{\leq \ell}$ from $\mathcal{A}$ do:
    if $n < \omega$ then
(∗) $y \xleftarrow{R} \mathcal{S}$
    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 $\Pi_k \in \mathrm{Perms}[\mathcal{K}]$ 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 $(k, b) \in \mathcal{K} \times \mathcal{X}$, to which the challenger responds with $\Pi_k^{-1}(b)$.

  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 $\Pi_k(a)$ in place of $E(k, a)$ and $\Pi_k^{-1}(b)$ in place of $D(k, b)$.

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 $\Pi = \Pi_{k_0}$ for some fixed, publicly available key $k_0 \in \mathcal{K}$.

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 $k \in \mathcal{K}$ until a key k satisfying $y_i = E(k, x_i)$ for all $i = 1, \ldots, Q$ is found. For block ciphers used in practice it is likely that this 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 $k \xleftarrow{R} \mathcal{K}$.

- $\mathcal{A}$ 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 $y_i \xleftarrow{R} E(k, x_i)$, and gives y_i to $\mathcal{A}$.
- Eventually $\mathcal{A}$ outputs an candidate key $\mathsf{k} \in \mathcal{K}$.

We say that $\mathcal{A}$ wins the game if $\mathsf{k} = k$. 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 $E(\mathsf{k}, \mathsf{a}) = \Pi_{\mathsf{k}}(\mathsf{a})$ and $D(\mathsf{k}, \mathsf{b}) = \Pi_{\mathsf{k}}^{-1}(\mathsf{b})$, and $\{\Pi_{\mathsf{k}}\}_{\mathsf{k} \in \mathcal{K}}$ 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*

$$\mathrm{KR}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}_{\mathrm{EX}}, \mathcal{E}] \geq (1 - \epsilon) \quad \textit{where} \quad \epsilon := \frac{|\mathcal{K}|}{(|\mathcal{X}| - Q)^Q} \tag{4.30}$$

Proof. In the ideal cipher model, we are modeling the block cipher $\mathcal{E} = (E, D)$ as a family $\{\Pi_{\mathsf{k}}\}_{\mathsf{k} \in \mathcal{K}}$ 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 $\Pi_{\mathsf{k}}(\mathsf{a})$ and $\Pi_{\mathsf{k}}^{-1}(\mathsf{b})$ for points $\mathsf{k} \in \mathcal{K}$ and $\mathsf{a}, \mathsf{b} \in \mathcal{X}$ 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, ..., x_Q} be an arbitrary set of distinct messages in X
for i = 1, ..., Q do:
    make a standard query to obtain y_i := E(k, x_i) = Π_k(x_i)
for each k ∈ K do:
    for i = 1, ..., Q do:
        make an ideal cipher query to obtain b_i := Π_k(x_i)
    if y_i = b_i for all i = 1, ..., Q then
        output 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 (4.30). 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 Π_κ for $\kappa \in \mathcal{K}$.

For each $\kappa \in \mathcal{K}$, let W_κ be the event that $y_i = \Pi_\kappa(x_i)$ for all $i = 1, \ldots, Q$. Note that by definition, W_k occurs with probability 1. Let W be the event that W_κ occurs for some $\kappa \neq k$. We want to show that $\Pr[W] \leq \epsilon$.

Fix $\kappa \neq k$. Since the permutation Π_k is chosen independently of the permutation Π_κ, we know that

$$\Pr[W_\kappa] = \frac{1}{|\mathcal{X}|} \cdot \frac{1}{|\mathcal{X}|-1} \cdots \frac{1}{|\mathcal{X}|-Q+1} \leq \left(\frac{1}{|\mathcal{X}|-Q}\right)^Q$$

As this holds for all $\kappa \neq k$, the result follows from the union bound. □

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 $2^{2.56}$ 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 $\{E(\kappa, \cdot) : \kappa \in \mathcal{K}\}$ 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 $(\mathcal{X}^2, \mathcal{X})$:

$$E\big((P_1, P_2),\ x\big) := \pi(x \oplus P_1) \oplus P_2 \qquad \text{and} \qquad D\big((P_1, P_2),\ y\big) := \pi^{-1}(y \oplus P_2) \oplus P_1 \tag{4.31}$$

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 $\mathcal{A}$:

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{E}_{EM}, \mathcal{A}] \le \frac{2Q_{\mathrm{s}}Q_{\mathrm{ic}}}{|\mathcal{X}|} \tag{4.32}$$

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.

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 (4.32) 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 4.5).

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 f_{AES} plays the role of π. Of course one round of AES is not a secure block cipher: the bound in (4.32) does not imply security because f_{AES} is not a random permutation.

Suppose one replaces each occurrence of f_{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 (4.32).

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

$$EX\big((k, P_1, P_2),\ x\big) := E(k,\ x \oplus P_1) \oplus P_2\ , \qquad DX\big((k, P_1, P_2),\ y\big) := D(k,\ y \oplus P_2) \oplus P_1. \tag{4.33}$$

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 where as 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 (4.31) in the ideal permutation model and of the $\mathcal{E}X$ construction (4.33) 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 (4.33), where P_1 and P_2 are each uniformly distributed over a subset of $\mathcal{X}'$ 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*

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{E}X] \leq \frac{2Q_{\mathrm{s}}Q_{\mathrm{ic}}}{|\mathcal{K}||\mathcal{X}'|}. \quad \square \tag{4.34}$$

To understand the security benefit of the $\mathcal{E}X$ construction consider the following: modeling $\mathcal{E}$ as an ideal cipher gives $\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{E}] \leq Q_{\mathrm{ic}}/|\mathcal{K}|$ 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 (4.34). The advantage of this $\mathcal{A}$ is no smaller 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 (4.34). Indeed, to make the two types of queries interact with each other, the adversary has to make

$$(\mathsf{k} = k \text{ and } \mathsf{a} = x \oplus P_1) \text{ or } (\mathsf{k} = k \text{ and } \mathsf{b} = y \oplus P_2)$$

for some input/output pair (x, y) corresponding to a standard query and some input/output triple $(\mathsf{k}, \mathsf{a}, \mathsf{b})$ 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 $\Pi_\mu \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$ and $\overline{\Pi}_\nu \xleftarrow{R} \mathrm{Perms}[\mathcal{X}]$ 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 is 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 the 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

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, \mathcal{E}X] = |p_0 - p_1|. \tag{4.35}$$

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 $\mathcal{k} \in \mathcal{K}$, set $\Pi_{\mathcal{k}} \xleftarrow{\mathrm{R}} \mathrm{Perms}[\mathcal{X}]$
$k \xleftarrow{\mathrm{R}} \mathcal{K}$, choose P_1, P_2

standard EX-query x:
1. $a \leftarrow x \oplus P_1$
2. $b \leftarrow \Pi_k(a)$
3. $y \leftarrow b \oplus P_2$
4. return y

ideal cipher Π-query k, a:
1. $b \leftarrow \Pi_k(a)$
2. return b

ideal cipher Π^{-1}-query k, b:
1. $a \leftarrow \Pi_k^{-1}(b)$
2. return 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

$$\Pr[W_0] = p_0. \tag{4.36}$$

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 Π_k gets split into two independent permutations: $\Pi_{\mathrm{std},k}$, which is used by the challenger in responding to standard EX-queries, and $\Pi_{\mathrm{ic},k}$, which is used in responding to ideal cipher queries. In detail (changes from Game 0 are highlighted):

Initialize:
for each $k \in \mathcal{K}$, set $\Pi_{\mathrm{std},k} \xleftarrow{\mathrm{R}} \mathrm{Perms}[\mathcal{X}]$ and $\Pi_{\mathrm{ic},k} \xleftarrow{\mathrm{R}} \mathrm{Perms}[\mathcal{X}]$
$k \xleftarrow{\mathrm{R}} \mathcal{K}$, choose P_1, P_2

standard EX-query x:
1. $a \leftarrow x \oplus P_1$
2. $b \leftarrow \Pi_{\mathrm{std},k}(a)$ // add a to sampled domain of $\Pi_{\mathrm{std},k}$, add b to sampled range of $\Pi_{\mathrm{std},k}$
3. $y \leftarrow b \oplus P_2$
4. return y

ideal cipher Π-query k, a:
1. $b \leftarrow \Pi_{\mathrm{ic},k}(a)$ // add a to sampled domain of $\Pi_{\mathrm{ic},k}$, add b to sampled range of $\Pi_{\mathrm{ic},k}$
2. return b

ideal cipher Π^{-1}-query k, b:
1. $a \leftarrow \Pi_{\mathrm{ic},k}^{-1}(b)$ // add a to sampled domain of $\Pi_{\mathrm{ic},k}$, add b to sampled range of $\Pi_{\mathrm{ic},k}$
2. return 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 $k \in \mathcal{K}$, such that the sampled domains of $\Pi_{\mathrm{ic},k}$ and $\Pi_{\mathrm{std},k}$ overlap or the sampled ranges

of $\Pi_{\mathrm{ic},\mathcal{k}}$ and $\Pi_{\mathrm{std},\mathcal{k}}$ overlap. The Domain Separation Lemma says that

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z]. \tag{4.37}$$

In applying the Domain Separation Lemma, the "coalescing function" f maps from $\{\mathrm{std}, \mathrm{ic}\} \times \mathcal{K}$ to $\mathcal{K}$, sending the pair $(\cdot, \mathcal{k})$ to $\mathcal{k}$. 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 $\mathcal{k} = k$. 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 $(\mathcal{k}, \mathcal{a}, \mathcal{b})$ triple arising from a Π- or Π^{-1}-query, and for some input/output pair (x, y) arising from an *EX*-query, we have

$$(\mathcal{k} = k \text{ and } \mathcal{a} = x \oplus P_1) \text{ or } (\mathcal{k} = k \text{ and } \mathcal{b} = y \oplus P_2). \tag{4.38}$$

Using the union bound, we can therefore bound $\Pr[Z]$ as a sum of probabilities of $2Q_{\mathrm{s}}Q_{\mathrm{ic}}$ events, each of the form $\mathcal{k} = k$ and $\mathcal{a} = x \oplus P_1$, or of the form $\mathcal{k} = k$ and $\mathcal{b} = y \oplus P_2$. 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

$$\Pr[Z] \leq \frac{2Q_{\mathrm{s}}Q_{\mathrm{ic}}}{|\mathcal{K}||\mathcal{X}'|}. \tag{4.39}$$

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 $\Pi'(x) := \Pi_{\mathrm{std},k}(x \oplus P_1) \oplus P_2$ and this permutation is independent of the random permutations used in the Π- and Π^{-1}-queries. Thus,

$$\Pr[W_1] = p_1. \tag{4.40}$$

The bound (4.34) now follows from (4.35), (4.36), (4.37), (4.39), and (4.40). 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 $(\mathcal{a}, \mathcal{b})$. Each set Π represents a function that can be extended to a permutation on $\mathcal{X}$ that sends $\mathcal{a}$ to $\mathcal{b}$ for every $(\mathcal{a}, \mathcal{b})$ in Π. We call such a set Π a *partial permutation on* $\mathcal{X}$. Define

$$\begin{aligned} \mathrm{Domain}(\Pi) &= \{\mathcal{a} \in \mathcal{X} : (\mathcal{a}, \mathcal{b}) \in \Pi \text{ for some } \mathcal{b} \in \mathcal{X}\}, \\ \mathrm{Range}(\Pi) &= \{\mathcal{b} \in \mathcal{X} : (\mathcal{a}, \mathcal{b}) \in \Pi \text{ for some } \mathcal{a} \in \mathcal{X}\}. \end{aligned}$$

Also, for $\mathcal{a} \in \mathrm{Domain}(\Pi)$, define $\Pi(\mathcal{a})$ to be the unique $\mathcal{b}$ such that $(\mathcal{a}, \mathcal{b}) \in \Pi$. Likewise, for $\mathcal{b} \in \mathrm{Range}(\Pi)$, define $\Pi^{-1}(\mathcal{b})$ to be the unique $\mathcal{a}$ such that $(\mathcal{a}, \mathcal{b}) \in \Pi$.

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. $b \xleftarrow{R} \mathcal{X} \setminus \text{Range}(\overline{\Pi}_{f(\mu)})$
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 $a \leftarrow \overline{\Pi}^{-1}_{f(\mu)}(b)$, return a
2. $a \xleftarrow{R} \mathcal{X} \setminus \text{Domain}(\overline{\Pi}_{f(\mu)})$
3. add (a, b) to Π_μ
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 $f(\mu) = f(\mu')$. The 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. $b \xleftarrow{R} \mathcal{X} \setminus \text{Range}(\Pi_\mu)$
* 2b. if $b \in \bigcup_{\mu' \in [\mu]} \text{Range}(\Pi_{\mu'})$ then $b \xleftarrow{R} \mathcal{X} \setminus \bigcup_{\mu' \in [\mu]} \text{Range}(\Pi_{\mu'})$
3. add (a, b) to Π_μ
4. return b

Process query $(\mu, -1, b)$:
1a. if $b \in \text{Range}(\Pi_\mu)$ then $a \leftarrow \Pi^{-1}_\mu(b)$, return a
* 1b. if $b \in \text{Range}(\Pi_{\mu'})$ for some $\mu' \in [\mu]$ then $a \leftarrow \Pi^{-1}_{\mu'}(b)$, return a
2a. $a \xleftarrow{R} \mathcal{X} \setminus \text{Domain}(\Pi_\mu)$
* 2b. if $a \in \bigcup_{\mu' \in [\mu]} \text{Domain}(\Pi_{\mu'})$ then $a \xleftarrow{R} \mathcal{X} \setminus \bigcup_{\mu' \in [\mu]} \text{Domain}(\Pi_{\mu'})$
3. add (a, b) to Π_μ
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

$$\text{Domain}(\Pi_\mu) \cap \text{Domain}(\Pi_{\mu'}) = \emptyset \quad \text{and} \quad \text{Range}(\Pi_\mu) \cap \text{Range}(\Pi_{\mu'}) = \emptyset$$
$$\text{for all } \mu, \mu' \in f^{-1}(\nu) \text{ with } \mu \neq \mu'. \tag{4.41}$$

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 (4.41) 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 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 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.

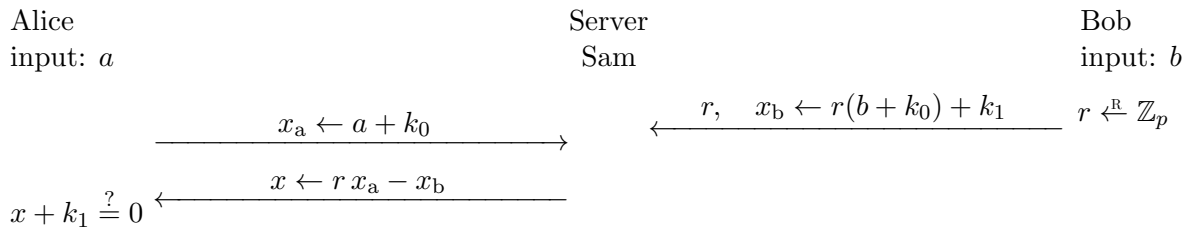


Figure 4.17: Comparing a and b without revealing them

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 $x \leftarrow r\, x_a - x_b$ 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_a, x_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:

$$(k_0, k_1) \leftarrow F(k,\ cnt_b). \tag{4.42}$$

He sends (r, x_b, cnt_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 (4.42) 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. 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 $F_1(k, x) = F(k, x) \,\|\, 0$ is not a secure PRF.

(b) Prove 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) Show that $F_4(k, x) := F(k, x) \,\|\, F(k, x \oplus 1^n)$ is insecure.

(e) Prove that $F_5(k, x) := F(F(k, 0^n), x)$ is a secure PRF.

(f) Show that $F_6(k, x) := F(F(k, 0^n), x) \,\|\, F(k, x)$ is insecure.

(g) Show that $F_7(k, x) := F(k, x) \,\|\, F\big(k, F(k, x)\big)$ 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

$$F_2\big(k,\ (x, y)\big) := F(k, x) \oplus F(k, y).$$

We showed in Exercise 4.1 part (b) that F_2 is not a secure PRF.

(a) Show that F_2 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_2 (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

$$\text{wPRF}\mathsf{adv}[\mathcal{A}, F_2] \leq \text{wPRF}\mathsf{adv}[\mathcal{B}, F] + (Q/|\mathcal{X}|)^4.$$

(b) Suppose F is a secure PRF. Show that F_2 is weakly secure even if we modify the weak PRF attack game and allow the adversary $\mathcal{A}$ to query F_2 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_2 is no longer secure if we modify the weak PRF attack game and allow the adversary $\mathcal{A}$ to query F_2 at two chosen points in addition to the Q random points.

4.3. 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 $\mathcal{X}' \subset \mathcal{X}$. 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. 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: $F'(k,x) = x[0 \ldots \ell - 1]$. 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. In Section 4.7.3 we discussed the Even-Mansour block cipher $(E_{\text{EM}}, D_{\text{EM}})$ built from a permutation $\pi : \mathcal{X} \to \mathcal{X}$ where $\mathcal{X} = \{0,1\}^n$. Recall that $E_{\text{EM}}\big((P_0, P_1),\ m\big) := \pi(m \oplus P_0) \oplus P_1$.

(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.

4.6 (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 [47, 43].

4.7 (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, $k \xleftarrow{R} \mathcal{K}$ and $f \xleftarrow{R} F(k,\cdot)$, while in Experiment 1, $f \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{Y}]$. 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.

4.8. 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

$$b \xleftarrow{R} \{0,1\},\ k \xleftarrow{R} \mathcal{K},\ y_1 \xleftarrow{R} \mathcal{Y}$$

Then $\mathcal{A}$ makes a series of queries to the challenger. There are two types of queries:

Encryption: 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 $x \in \mathcal{X}$ to the challenger, who computes $y_0 \leftarrow F(k, x)$ and responds with 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).

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 x_0 in $\mathcal{X}$ looks like a random element of $\mathcal{Y}$, even after seeing the value of the PRF at many other points of $\mathcal{X}$.

4.9 (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.
In contrast, we note that there are secure PRFs where $F(k_1 \oplus k_2,\ x) = F(k_1, x) \oplus F(k_2, x)$.

4.10. 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 $(\mathcal{K}, \{0,1\}^n)$:

$$E'(k, m) := D(k,\ t \oplus E(k, m)\ ) \quad \text{and} \quad D'(k, c) := E(k,\ t \oplus D(k, m)\ )$$

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.

4.11. 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 with key space $\mathcal{K}$.

(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) 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 [24].

4.12 (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 4.10). In applications of tweakable block ciphers, this strong security notion is more appropriate.

(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

$$E'\big((k_0,k_1),m,t\big) := \big\{p \leftarrow F(k_0,t);\ \ \text{output } p \oplus E(k_1,\ m \oplus p)\big\},$$

where (E,D) is a *strongly* secure block cipher and F is a secure PRF. In Exercise 7.12 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 7.12.

4.13 (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.

4.14 (Block cipher combiners). Continuing with Exercise 4.13, 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.

4.15. 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 $k_1 = k \,\|\, b$ 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.

4.16. 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.

4.17. In the tree construction for building a PRF from a PRG (see 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 at the root and using the key to trace through the tree is not a secure PRF.

4.18. Show how the tree construction in Section 4.6 gives a PRG with large expansion rate and which can be evaluated with a highly parallel algorithm.

4.19 (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:

$$G'(k, s_0, \ldots, s_{Q-1}) := (G(k, s_0), \ldots, G(k, s_{Q-1})) \ .$$

4.20 (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 $\pi : \mathcal{X} \to \mathcal{X}$:

$$E(k, x) := (k \,\|\, 0^m) \oplus \pi(k \,\|\, x) \tag{4.43}$$

$D(k, c)$ 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

$$\mathrm{BC}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, E] \le \frac{2Q_{\mathrm{ic}}}{|\mathcal{K}|} \ . \tag{4.44}$$

Here Q_{ic} is the number of queries $\mathcal{A}$ makes to Π- and Π^{-1}-oracles.

4.21 (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

$$F(k, x) := (k \,\|\, x) \oplus \pi(k \,\|\, x) \ . \tag{4.45}$$

This PRF is an abstraction of the PRF underlying the Salsa and ChaCha stream ciphers. Use Exercise 4.20 to show that if we model π as an ideal permutation Π, then for every PRF adversary $\mathcal{A}$ attacking F we have

$$\mathrm{PRF}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, F] \leq \frac{2Q_{\mathrm{ic}}}{|\mathcal{K}|} + \frac{Q_{\mathrm{F}}^2}{2|\mathcal{X}|} \tag{4.46}$$

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."

4.22 (Alternative proof of Theorem 4.6). Let $\mathbf{X}$ and $\mathbf{Y}$ 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 $\mathrm{PF}\mathsf{adv}[\mathcal{A}, \mathcal{X}] \leq \Delta[\mathbf{X}, \mathbf{Y}] \leq Q^2/2N$.

4.23 (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 $\Pr[W_0] \leq \rho[\mathbf{X}, \mathbf{Y}] \cdot \Pr[W_1] \leq 2\Pr[W_1]$.

4.24 (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 $\mathrm{Funs}[\mathcal{X}, \mathcal{Y}]$, 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 $\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F'] = n \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F]$.

4.25 (Universal attacker on PRFs). Let F be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{Y})$ where $|\mathcal{K}| < |\mathcal{X}|$. Let $Q < |\mathcal{K}|$. Show that there is a PRF adversary $\mathcal{A}$ that runs in time proportional to Q, makes one query to the PRF challenger, and has advantage

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F] \geq \left| \frac{Q}{|\mathcal{K}|} - \frac{Q}{|\mathcal{X}|} \right|.$$

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 may 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 will 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 2.3), 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 decrypt any one megabyte file stored on Alice's server! This email may even be caught in Alice's spam filter, and never actually 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 of 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 $k_i \xleftarrow{R} \mathcal{K}$, $c_i \xleftarrow{R} E(k_i, m_{ib})$, 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

$$\mathrm{MSSadv}[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - \Pr[W_1]|. \quad \square$$

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* $\mathrm{MSSadv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

As discussed in Section 2.3.5, 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* $\mathrm{MSSadv}^*[\mathcal{A}, \mathcal{E}]$ as $|\Pr[\hat{b} = b] - 1/2|$, and as usual, we have $\mathrm{MSSadv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \mathrm{MSSadv}^*[\mathcal{A}, \mathcal{E}]$.

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*
>
> $$\mathrm{MSSadv}[\mathcal{A}, \mathcal{E}] = Q \cdot \mathrm{SSadv}[\mathcal{B}, \mathcal{E}].$$

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:

$k_i \xleftarrow{R} \mathcal{K}$
if $i > j$ then $\quad c_i \xleftarrow{R} E(k_i, m_{i0}) \quad$ else $\quad c_i \xleftarrow{R} E(k_i, m_{i1})$.

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

$$\mathrm{MSSadv}[\mathcal{A}, \mathcal{E}] = |p_Q - p_0|. \tag{5.1}$$

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
    $k_i \xleftarrow{R} \mathcal{K}$, $c_i \xleftarrow{R} E(k_i, m_{i0})$
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 // $i < \omega$
    $k_i \xleftarrow{R} \mathcal{K}$, $c_i \xleftarrow{R} E(k_i, m_{i1})$.

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

$$\mathrm{MSSadv}[\mathcal{A}, \mathcal{E}] = Q \cdot \mathrm{SSadv}[\mathcal{B}, \mathcal{E}]. \tag{5.2}$$

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 (5.2) now follows from this observation, together with (5.1), via the usual telescoping sum calculation:

$$\begin{aligned}
\mathrm{SSadv}[\mathcal{B}, \mathcal{E}] &= |\Pr[W_1] - \Pr[W_0]| \\
&= \frac{1}{Q} \cdot \left| \sum_{j=1}^{Q} \Pr[W_1 \mid \omega = j] - \sum_{j=1}^{Q} \Pr[W_0 \mid \omega = j] \right| \\
&= \frac{1}{Q} \cdot |p_Q - p_0| \\
&= \frac{1}{Q} \cdot \mathrm{MSSadv}[\mathcal{A}, \mathcal{E}],
\end{aligned}$$

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 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 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 $k \xleftarrow{R} \mathcal{K}$.
- 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.

  The challenger computes $c_i \xleftarrow{R} E(k, m_{ib})$, 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

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

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 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* $\mathrm{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

As in Section 2.3.5, 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 $\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] := |\Pr[\hat{b} = b] - 1/2|$, and as usual, we have $\mathrm{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]$.

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 $\mathrm{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$. On then 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 use 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 $\mathrm{CPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 1$. $\square$

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. $\square$

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, its 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

$$E_0(k', m) := \quad x \xleftarrow{R} \mathcal{X}, \quad \text{output } (x,\ F(k', x) \oplus m)$$

- 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*
>
> $$\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] \leq \frac{Q^2}{2N} + \text{PR}\mathsf{Fadv}[\mathcal{B}_F, F] + Q \cdot \text{SS}\mathsf{adv}^*[\mathcal{B}_\mathcal{E}, \mathcal{E}]. \tag{5.3}$$

Proof idea. Suppose we start with an adversary that plays the bit-guessing version of 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. 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 bit-guessing version of Attack Game 5.1. Therefore, we can use the assumption that $\mathcal{E}$ is semantically secure, and hence (by Theorem 5.1) multi-key semantically secure, to argue that the adversary's advantage in the original CPA attack game must be negligible. □

Proof. Let $\mathcal{A}$ be an efficient CPA adversary that attacks $\mathcal{E}'$ as in the bit-guessing version of Attack Game 5.2. Assume that $\mathcal{A}$ makes at most Q queries to its challenger. Our goal is to show that $\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}']$ is negligible, assuming that F is a secure PRF, that N is super-poly, and that $\mathcal{E}$ is semantically secure.

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; 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 $\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] := |\Pr[W_0] - 1/2|$ is negligible.

Game 0. Let us begin by giving a detailed description of the challenger in Game 0 that is convenient for our purposes:

$b \xleftarrow{R} \{0, 1\}$
$k' \xleftarrow{R} \mathcal{K}'$
for $i \leftarrow 1$ to Q do
    $x_i \xleftarrow{R} \mathcal{X}$
    $k_i \leftarrow F(k', x_i)$
upon receiving the ith 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.

By construction, we have

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] = \Big|\Pr[W_0] - 1/2\Big|, \tag{5.4}$$

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:

$b \xleftarrow{R} \{0,1\}$
$f \xleftarrow{R} \mathrm{Funs}[\mathcal{X}, \mathcal{K}]$
for $i \leftarrow 1$ to Q do
    $x_i \xleftarrow{R} \mathcal{X}$
    $k_i \leftarrow f(x_i)$
upon receiving the ith 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.

We claim that

$$|\Pr[W_1] - \Pr[W_0]| = \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F], \tag{5.5}$$

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:

$b \xleftarrow{R} \{0,1\}$
for $i \leftarrow 1$ to Q do
    $x_i \xleftarrow{R} \mathcal{X}$
    $k_i \xleftarrow{R} f(x_i)$.

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

$$c_i \xleftarrow{R} E(k_i, m_{ib})$$

and sends (x_i, c_i) to $\mathcal{A}$.

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 $(*)$:

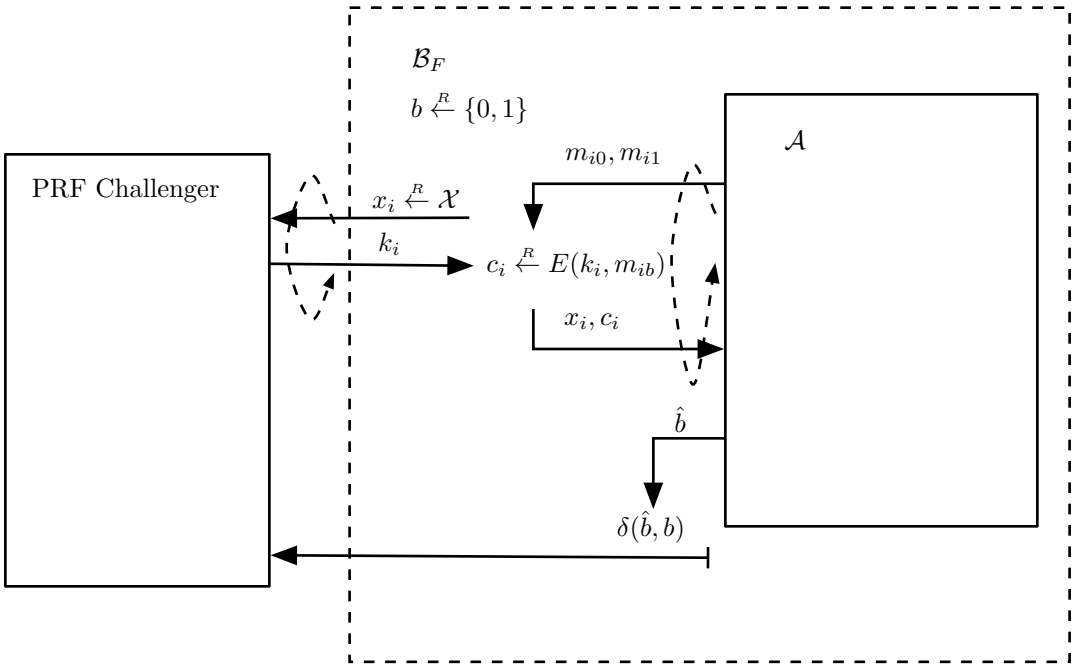


Figure 5.1: Adversary $\mathcal{B}_F$ in the proof of Theorem 5.2

$b \xleftarrow{R} \{0,1\}$
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 ith 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

$$\Pr[W_2] = \Pr[W_1]. \tag{5.6}$$

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 ith 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 (i.e., 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

$$|\Pr[W_3] - \Pr[W_2]| \leq \Pr[Z]. \tag{5.7}$$

Moreover, it is easy to see that

$$\Pr[Z] \leq \frac{Q^2}{2N}, \tag{5.8}$$

since Z is the union of less that $Q^2/2$ events, each of which occurs with probability $1/N$.

Observe that in Game 3, different, independent encryption keys k_i are used to encrypt each message. So next, we play our "semantic security card," claiming that

$$|\Pr[W_3] - 1/2| = \mathrm{MSSadv}^*[\bar{\mathcal{B}}_{\mathcal{E}}, \mathcal{E}], \tag{5.9}$$

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 (2.12)) that (5.9) holds. Moreover, by Theorem 5.1, we have

$$\mathrm{MSSadv}^*[\bar{\mathcal{B}}_{\mathcal{E}}, \mathcal{E}] = Q \cdot \mathrm{SSadv}^*[\mathcal{B}_{\mathcal{E}}, \mathcal{E}], \tag{5.10}$$

where $\mathcal{B}_{\mathcal{E}}$ is an efficient adversary playing the bit-guessing version of Attack Game 2.1 with respect to $\mathcal{E}$.

Putting together (5.4) through (5.10), we obtain (5.3). 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 (5.3) holds without assuming that $\mathcal{A}$ 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 bounded even if F is only weakly secure. □

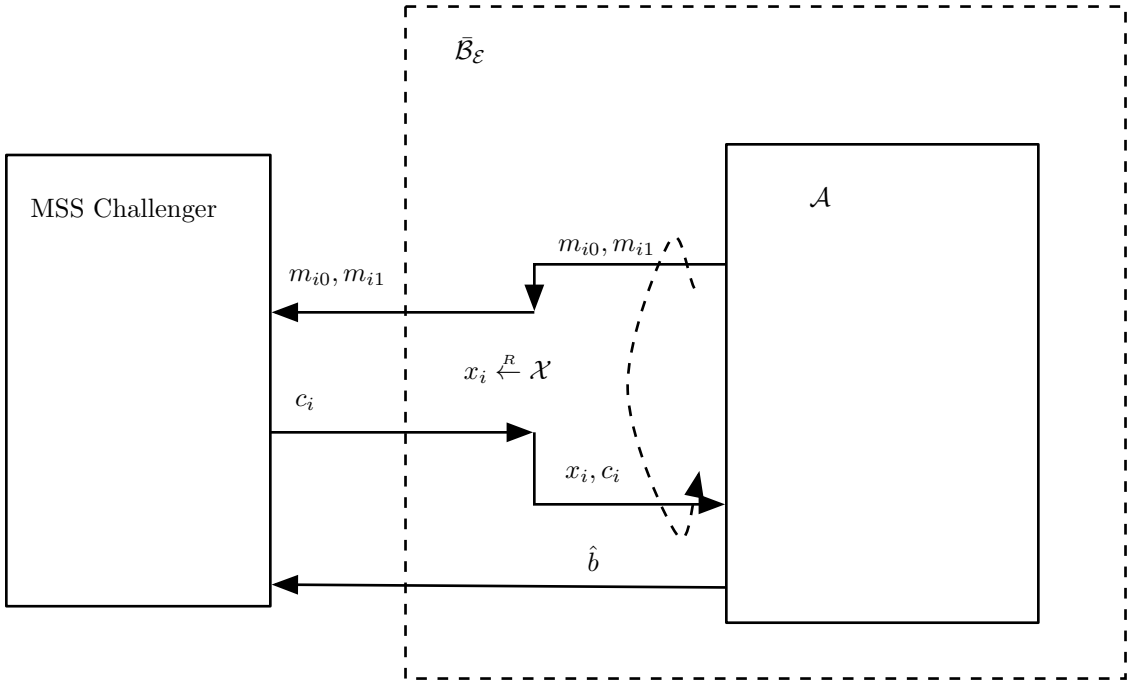


Figure 5.2: Adversary $\bar{\mathcal{B}}_{\mathcal{E}}$ in the proof of Theorem 5.2

5.4.2 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 $\mathcal{X} = \{0, \ldots, N-1\}$, 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 $0, \ldots, 2^n - 1$. As it happens, the decryption function of the block cipher is not needed at all in this construction.

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 $|m|$.

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

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] \leq \frac{2Q^2\ell}{N} + \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F]. \tag{5.11}$$

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. We shall work with the bit-guessing version of Attack Game 5.2 (again, see Section 2.3.5).

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:

$b \xleftarrow{R} \{0,1\}$
$k \xleftarrow{R} \mathcal{K}$
for $i \leftarrow 1$ to Q do
    $x_i \xleftarrow{R} \mathcal{X}$
    for $j \leftarrow 0$ to $\ell - 1$ do
        $x'_{ij} \leftarrow x_i + j \bmod N$
        $y_{ij} \leftarrow F(k, x'_{ij})$
upon receiving the ith 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 we have

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] = \Big|\Pr[W_0] - 1/2\Big|. \tag{5.12}$$

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

$$|\Pr[W_1] - \Pr[W_0]| = \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F], \tag{5.13}$$

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

$$\Pr[W_2] = \Pr[W_1]. \tag{5.14}$$

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 (i.e., 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

$$|\Pr[W_3] - \Pr[W_2]| \leq \Pr[Z]. \tag{5.15}$$

We claim that

$$\Pr[Z] \leq \frac{2Q^2\ell}{N}. \tag{5.16}$$

To prove this claim, we may assume that $N \geq 2\ell$ (this should anyway 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 (5.16) now follows.

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,

$$\Pr[W_3] = 1/2. \tag{5.17}$$

Putting together (5.12) through (5.17), 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 5.4. □

Case study: AES counter mode

The IPsec protocol uses a particular variants 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 encryption 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

An 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

$$
\begin{array}{l}
D'(k, c) := \\
\quad \text{compute } m \in \mathcal{X}^v \text{ as follows:} \\
\quad\quad \text{for } j \leftarrow 0 \text{ to } v - 1 \text{ do} \\
\quad\quad\quad m[j] \leftarrow D(k,\ c[j+1]) \oplus c[j] \\
\quad \text{output } m.
\end{array}
$$

See Fig. 5.3 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

$$\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] \leq \frac{Q^2 \ell^2}{N} + \text{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{5.18}$$

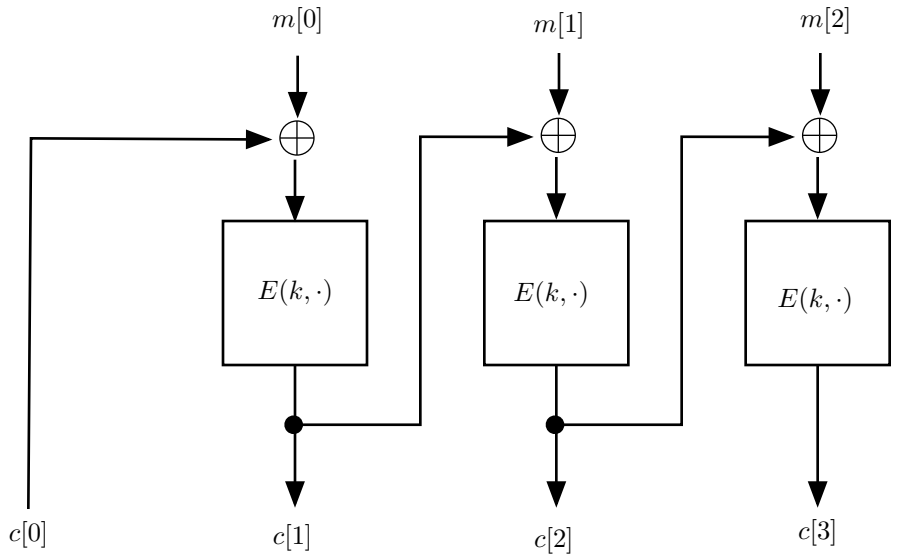
Proof idea. The basic idea of the proof is very similar to that of Theorem 5.3. We start with an adversary that plays the CPA attack game with respect to $\mathcal{E}'$. We then replace E by a truly random function f. Then we argue that except with negligible probability, the challenger never evaluates 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 the bit-guessing version of 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 $(\mathcal{K}, \mathcal{X}, \mathcal{X})$.

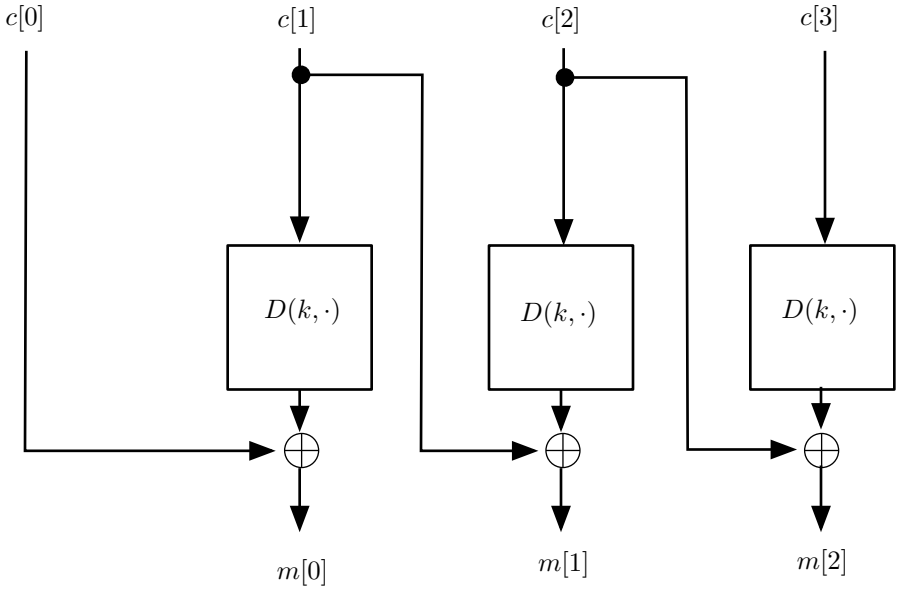
As usual, we define a sequence of games: Game 0, Game 1, Game 2, Game 3. Each of these games is played between $\mathcal{A}$ and a challenger. The challenger in Game 0 is the one from the bit-guessing version of Attack Game 5.2 with respect to $\mathcal{E}'$. In each of these games, b denotes the random 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 5.3: Encryption and decryption for CBC mode with $\ell = 3$

$b \xleftarrow{R} \{0,1\}$, $k \xleftarrow{R} \mathcal{K}$

upon receiving the ith 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

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] = \Big|\Pr[W_0] - 1/2\Big|. \tag{5.19}$$

Game 1. We now play the "PRF card," replacing $E(k, \cdot)$ by a truly random function $f \in \mathrm{Funs}[\mathcal{X}, \mathcal{X}]$. Our challenger in this game looks like this:

$b \xleftarrow{R} \{0,1\}$, $f \xleftarrow{R} \mathrm{Funs}[\mathcal{X}, \mathcal{X}]$

upon receiving the ith 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

$$|\Pr[W_1] - \Pr[W_0]| = \mathrm{PRF}\mathsf{adv}[\mathcal{B}, E], \tag{5.20}$$

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 $\mathrm{PRF}\mathsf{adv}[\mathcal{B}, E]$ 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 ith 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

$$\Pr[W_2] = \Pr[W_1]. \tag{5.21}$$

Game 3. Now we make gnome forgetful, removing the check in the line marked $(*)$:

$b \xleftarrow{R} \{0,1\}$
set $y_{ij} \xleftarrow{R} \mathcal{X}$ for $i = 1, \ldots, Q$ and $j = 0, \ldots, \ell$
upon receiving the ith 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
            $x_{ij} \leftarrow c_i[j] \oplus m_{ib}[j]$
            $c_i[j+1] \xleftarrow{R} y_{i(j+1)}$
    send c_i to the adversary.

To analyze the quantity $|\Pr[W_3] - \Pr[W_2]|$, we use the Difference Lemma (i.e., 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

$$|\Pr[W_3] - \Pr[W_2]| \leq \Pr[Z]. \tag{5.22}$$

We claim that

$$\Pr[Z] \leq \frac{Q^2 \ell^2}{2N}. \tag{5.23}$$

To prove this, let *Coins* denote the random choices made by $\mathcal{A}$. Observe that in Game 3, the values

$$\textit{Coins}, \quad b, \quad y_{ij} \ (i = 1, \ldots Q, \ j = 0, \ldots, \ell)$$

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 (5.23) now follows from an easy calculation.

Finally, we claim that

$$\Pr[W_3] = 1/2. \tag{5.24}$$

This follows from the fact that

$$\textit{Coins}, \quad b, \quad y_{ij} \ (i = 1, \ldots Q, \ j = 0, \ldots, \ell)$$

are independently distributed, and the fact that the adversary's output $\hat{b}$ is a function of

$$\textit{Coins}, \quad y_{ij} \ (i = 1, \ldots Q, \ j = 0, \ldots, \ell).$$

From this, we see that $\hat{b}$ and b are independent, and so (5.24) follows immediately.

Putting together (5.19) through (5.24), we have

$$\mathrm{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] \leq \frac{Q^2\ell^2}{2N} + \mathrm{PRF}\mathsf{adv}[\mathcal{B}, E].$$

By Theorem 4.4, we have

$$\Big|\mathrm{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}] - \mathrm{PRF}\mathsf{adv}[\mathcal{B}, E]\Big| \leq \frac{Q^2\ell^2}{2N},$$

and the theorem follows. □

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 $\mathcal{X}^{\leq \ell}$. When the domain is $\mathcal{X} = \{0,1\}^{128}$, as in the case of AES, this implies that the length of messages to be encrypted must be a multiple of 16 bytes. Since the length of messages in practice need not be a multiple of 16 we need a way augment CBC to handle messages whose length is not necessarily 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 somehow pad the message m so that its length in bytes is a multiple of 16. Clearly the padding function needs to be invertible so that during decryption the padding can be removed.

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 the content of each byte is 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 5.15. 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:

$$\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\text{ctr}}] \leq \frac{2Q^2\ell}{2^n} + \text{PRF}\mathsf{adv}[\mathcal{B}_F, F],$$
$$\text{CPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}_{\text{cbc}}] \leq \frac{Q^2\ell^2}{2^n} + \text{BC}\mathsf{adv}[\mathcal{B}_\mathcal{E}, \mathcal{E}].$$

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 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 message 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 as compared with CBC.

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 132). Pipelining enables multiple encryption instructions to execute at the same time. A parallel mode such as counter mode keeps the pipeline busy, where as 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 where as 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 5.11.

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 5.9). 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 m_1, then an encryption m_2, and so on, while Bob first decrypts the encryption of m_1, and 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 $k_i := F(x_i)$. 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 $\aleph$, called a *nonce*. The syntax for nonce-based encryption becomes

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

where $c \in \mathcal{C}$ is the ciphertext, $k \in \mathcal{K}$ is the key, $m \in \mathcal{M}$ is the message, and $\aleph \in \mathcal{N}$ is the nonce.

Moreover, the encryption algorithm E is required to be deterministic. Likewise, the decryption syntax becomes

$$m = D(k, c, \aleph).$$

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

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

for all $k \in \mathcal{K}$, $m \in \mathcal{M}$, and $\aleph \in \mathcal{N}$. 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.

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 $k \xleftarrow{R} \mathcal{K}$.
- 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 $\aleph_i \in \mathcal{N} \setminus \{\aleph_1, \ldots, \aleph_{i-1}\}$.

  The challenger computes $c_i \leftarrow E(k, m_{ib}, \aleph_i)$, 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

$$\text{nCPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - \Pr[W_1]|. \quad \square$$

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* $\text{nCPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

As usual, as in Section 2.3.5, Attack Game 5.3 can be recast as a "bit guessing" game, and we have $\text{nCPA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] = 2 \cdot \text{nCPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}]$, where $\text{nCPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] := |\Pr[\hat{b} = b] - 1/2|$ 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 $D'(k', c) := D(k, c)$, 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 that 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

$$\mathrm{nCPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F] + Q \cdot \mathrm{SS}\mathsf{adv}^*[\mathcal{B}_\mathcal{E}, \mathcal{E}]. \tag{5.25}$$

We leave the proof as an exercise for the reader. Note that the term $\frac{Q^2}{2N}$ in (5.3), which represent the probability of a collision on the input to F, is missing from (5.25), 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 $\varkappa$ to the PRF input $x := \varkappa\ell$. It is easy to see that for any two distinct nonces $\varkappa_1$ and $\varkappa_2$, for $x_1 := \varkappa_1\ell$ and $x_2 := \varkappa_2\ell$, 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

$$\text{nCPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}, F]. \tag{5.26}$$

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:

$(m_{10}, m_{11}, \aleph_1)$,
$(m_{20}, m_{21}, \aleph_2)$,

where

$$m_{10} = \aleph_1 \neq \aleph_2 = m_{20},\ m_{11} = m_{21}.$$

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 $\aleph$ and key k' as $c[0] := F(k', \aleph)$.

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

$$\text{nCPA}\mathsf{adv}^*[\mathcal{A}, \mathcal{E}'] \leq \frac{Q^2\ell^2}{N} + \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \text{BC}\mathsf{adv}[\mathcal{B}, \mathcal{E}]. \tag{5.27}$$

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 5.13).

5.6 A fun application: revocation schemes

To be written.

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. Generalize the definition of CPA-security to the multi-key setting, analogous to Definition 5.1. Using a hybrid argument, prove that CPA-security implies CPA-security in the multi-key setting.

5.3. 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

$$b \xleftarrow{R} \{0, 1\},\ k \xleftarrow{R} \mathcal{K}.$$

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 $c \xleftarrow{R} E(k, m)$. 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 $c \xleftarrow{R} E(k, m_b)$. 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. 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.

5.5. Let $\mathcal{E} = (E, D)$ be a block cipher defined over $(\mathcal{K}, \mathcal{M} \times \mathcal{R})$. Consider the probabilistic cipher $\mathcal{E}' = (E', D')$, where

$$E'(k, m) := \big\{ r \xleftarrow{R} \mathcal{R},\ c \xleftarrow{R} E\big(k,\ (m, r)\big),\ \text{output } c \big\}$$
$$D'(k, c) := \big\{ (m, r') \leftarrow D(k, c),\ \text{output } m \big\}$$

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.

5.6 (pseudo-random ciphertext security). In Exercise 3.5, 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 $k \xleftarrow{R} \mathcal{K}$ 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 psuedo-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 psuedo-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 psuedo-random multi-ciphertext secure.

(c) Show that a psuedo-random multi-ciphertext secure cipher is also CPA secure.

(d) Give an example of a CPA secure cipher that is not psuedo-random multi-ciphertext secure.

5.7 (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 $r \xleftarrow{R} \mathcal{R}$ (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:

$$E'\big((k,k'),\ m\big) \quad := E(k,m;\ F(k',m)),$$
$$D'\big((k,k'),\ c\big) \quad := D(k,c)\ .$$

Show that $\mathcal{E}'$ is deterministic CPA secure. This construction is known as the **Synthetic IV** (or **SIV**) construction.

5.8 (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.

5.9 (Ciphertext expansion vs. security). Let $\mathcal{E} = (E, D)$ be an encryption scheme 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 more than $\approx 2^{\ell/2}$ messages.

5.10 (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 make a series of queries; the ith query is a message m_i, to which the challenger' responds with $c_i \xleftarrow{R} E(k, m_i)$. 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.

5.11 (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 [**?**] exploits this defect to completely break CBC encryption in TLS 1.0.

5.12. Suppose the block cipher used for CBC encryption has a block size of n bits. Construct an attacker that wins the CPA game against CBC that makes $\approx 2^{n/2}$ queries to its challenger and gains an advantage $\approx 1/2$. Your answer explains why CBC cannot be used with a block cipher that has a small block size (e.g. $n = 64$ bits). This is one reason why AES has a block size of 128 bits.

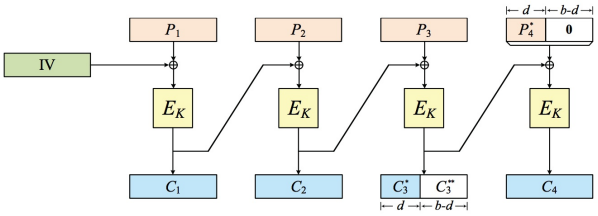
5.13. Consider the nonce-based CBC scheme $\mathcal{E}'$ described in Section 5.5.3. Suppose that the nonce space $\mathcal{N}$ is equal to block space $\mathcal{X}$ of the underlying block cipher $\mathcal{E} = (E, D)$, and the PRF F is just the encryption algorithm E. If the two keys k and k' in the construction are chosen independently, the scheme is secure. Your task is to show that if only one key k is chosen, and other key k' is just set to k, then the scheme is insecure.

5.14. 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**).

5.15. 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?

5.16. Suppose that one block of a CBC-encrypted ciphertext is corrupted, and is then decrypted. How many blocks of the decrypted plaintext are corrupted?

5.17 (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.

5.18. Suppose you are given an CBC encryption of a message $m \in \mathcal{X}^\ell$. You do not know m, but you are given $\Delta \in \mathcal{X}$. Show how to modify the ciphertext to obtain a new ciphertext that decrypts to m', where $m'[0] = m[0] \oplus \Delta$ and $m'[i] = m[i]$ for $i = 1, \ldots, \ell - 1$. That is, you can effectively flip any bit in the first block of the message, without affecting any bits in any of the other blocks.

5.19. 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 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:

$$E'(k, m) := \big\{ c \leftarrow E(k, m),\ t \leftarrow g(c),\ \text{output } (c, t) \big\}$$
$$D'\big(k,\ (c, t)\big) := D(k, c)$$

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) aattacks, 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 —

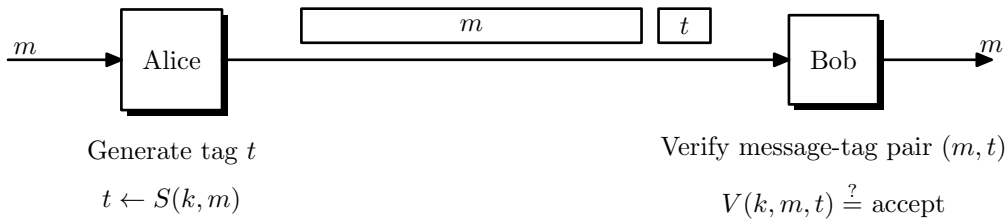


Figure 6.1: Short message integrity tag added to messages

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 what is a message integrity system based on a shared secret key between the sender and receiver. 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 $t \xleftarrow{R} S(k, m)$, 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 us 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

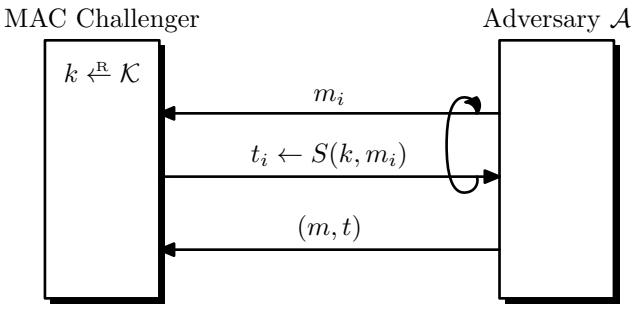


Figure 6.2: MAC attack game (Attack Game 6.1)

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 $m_i \in \mathcal{M}$. The challenger, given m_i, computes a tag $t_i \xleftarrow{R} S(k, m_i)$, and 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.,

$$(m, t) \notin \big\{(m_1, t_1), (m_2, t_2), \ldots\big\}.$$

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 $\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$, 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* $\mathrm{MACadv}[\mathcal{A}, \mathcal{I}]$ *is negligible.*

In case the adversary wins Attack Game 6.1, the pair (m, t) it sends 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 his 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 2.4. 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:*

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \text{and} \quad \mathbf{T} = \{\mathcal{T}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

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 a randomized 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 $m \in \mathcal{M}_{\lambda,\Lambda}$, runs in time bounded by a polynomial in λ, and outputs an element of $\mathcal{T}_{\lambda,\Lambda}$.*
4. *Algorithm V is a 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 $t \in \mathcal{T}_{\lambda,\Lambda}$, runs in time bounded by a polynomial in λ, and 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 his 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 $k \xleftarrow{R} \mathcal{K}$.
- $\mathcal{A}$ 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 $t_i \xleftarrow{R} S(k, m_i)$, 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.,
$$(\hat{m}_j, \hat{t}_j) \notin \big\{(m_1, t_1), (m_2, t_2), \ldots\big\}.$$
    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 $\mathrm{MAC}^{\mathrm{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$, 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*
> $$\mathrm{MAC}^{\mathrm{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq \mathrm{MAC}\mathsf{adv}[\mathcal{B}, \mathcal{I}] \cdot Q_{\mathrm{v}}.$$

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. Note that $\mathcal{A}$ 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 would let 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} verifcation queries, $\mathcal{B}$ will guess correctly with probability at least $1/Q_{\mathrm{v}}$. 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 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, $r_j =$ 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 1, 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, $r_j =$ 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 $\omega \xleftarrow{R} \{1, \ldots, Q_v\}$. 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

$$\Pr[W_2] \geq \Pr[W_1]/Q_v. \tag{6.3}$$

Evidently, by construction, we have

$$\Pr[W_2] = \mathrm{MACadv}[\mathcal{B}, \mathcal{I}]. \tag{6.4}$$

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 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 $\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}]$ 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 $\text{Funs}[\mathcal{X}, \mathcal{Y}]$ 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 his 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 ith 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 $\text{Funs}[\mathcal{X}, \mathcal{Y}]$. 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:

(∗) $f \xleftarrow{R} \text{Funs}[\mathcal{X}, \mathcal{Y}]$

We define W_1 to be the event that condition (6.5) holds in Game 1. It should be clear how to design the corresponding PRF adversary $\mathcal{B}$ such that:

$$|\Pr[W_1] - \Pr[W_0]| = \text{PRF}\mathsf{adv}[\mathcal{B}, F].$$

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 it all together, we obtain

$$\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}] = \Pr[W_0] \leq |\Pr[W_0] - \Pr[W_1]| + \Pr[W_1] \leq \text{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{1}{|\mathcal{Y}|}$$

as required. □

Concrete tag lengths. The theorem shows that to ensure $\mathsf{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}] < 2^{-128}$ 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.

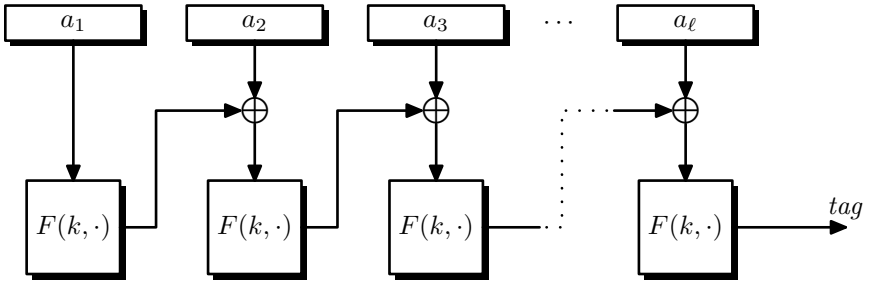
We solve this problem in three steps:

- 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*.

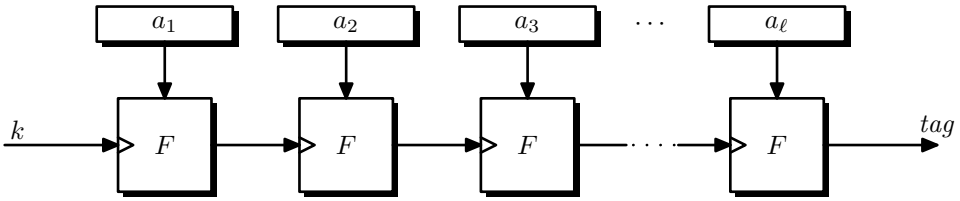
Prefix-free PRFs. We begin with two classic constructions for prefix-free secure PRFs. The **CBC** construction is shown in Fig. 6.3a. The **cascade** construction is shown in Fig. 6.3b. We show that when the underlying F is a secure PRF, both CBC and cascade are prefix-free secure PRFs.

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:



(a) The CBC construction $F_{\text{CBC}}(k, m)$



(b) The cascade construction $F^*(k, m)$

Figure 6.3: Two prefix-free secure PRFs

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. 5.3, 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

$$\text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{A}, F_{\text{CBC}}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{(Q\ell)^2}{2|\mathcal{X}|}. \tag{6.6}$$

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 (6.6) 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 (6.6) 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 $F_{\text{CBC}}(k, m)$, 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:

$$root \xrightarrow{a_1} p_1 \xrightarrow{a_2} p_2 \xrightarrow{a_3} \cdots \xrightarrow{a_v} p_v. \tag{6.7}$$

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 $\gamma_{root} := 0^n$, and for any non-root node q with parent p, if the corresponding edge in the tree is $p \xrightarrow{a} q$, 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 (6.7), then $\gamma_{p_v} = F_{\text{CBC}}(k, m)$.

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 $p \xrightarrow{a} q$ and $p' \xrightarrow{a'} q'$, 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 three closely related challengers in three 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 $\text{Funs}[\mathcal{X}, \mathcal{X}]$. Clearly, we have

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \text{PRF}\mathsf{adv}[\mathcal{B}, F] \tag{6.8}$$

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 is a particular way, using the "query tree" discussed above.

To this end, first let $B := Q\ell$, which represents an upper bound on how many points at which f will evaluated. Our challenger first prepares random values

$$\beta_i \xleftarrow{\text{R}} \mathcal{X} \qquad (i = 1, \ldots, B).$$

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:

$\gamma_q \leftarrow \beta_i$

$(*)$ if $\exists$ another edge $p' \xrightarrow{a'} q'$ with $\gamma_{p'} \oplus a' = \gamma_p \oplus a$ then $\gamma_q \leftarrow \gamma_{q'}$

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

$$\Pr[W_2] = \Pr[W_1]. \tag{6.9}$$

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 $\mathit{Coins}, \beta_1, \ldots, \beta_B$, 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

$$\big|\Pr[W_3] - \Pr[W_2]\big| \leq \Pr[Z]. \tag{6.10}$$

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

$$\gamma_{p'} \oplus \gamma_p = a' \oplus a. \tag{6.11}$$

There are two cases to consider.

Case 1: $p = p'$. Since the edges are distinct, we must have $a' \neq a$, and hence (6.11) 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 (6.11) holds with probability $1/|\mathcal{X}|$.

By the union bound, it follows that

$$\Pr[Z] \leq \frac{B^2}{2|\mathcal{X}|}. \tag{6.12}$$

Combining (6.8), (6.9), (6.10), and (6.12), we obtain

$$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, F_{\mathrm{CBC}}] = \big|\Pr[W_3] - \Pr[W_0]\big| \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F] + \frac{B^2}{2|\mathcal{X}|}. \tag{6.13}$$

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

$$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{A}, F^*] \leq Q\ell \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}, F]. \tag{6.14}$$

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 (6.14). 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 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 suprising about this adversary $\mathcal{B}$: it is essentially the universal PRF attacker from Exercise 4.25. Hence, (6.14) 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 (6.14) 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 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 4.24 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 rekeying 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

$$t' := F^*(k, \quad m \parallel m') \tag{6.15}$$

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 his 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:

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.

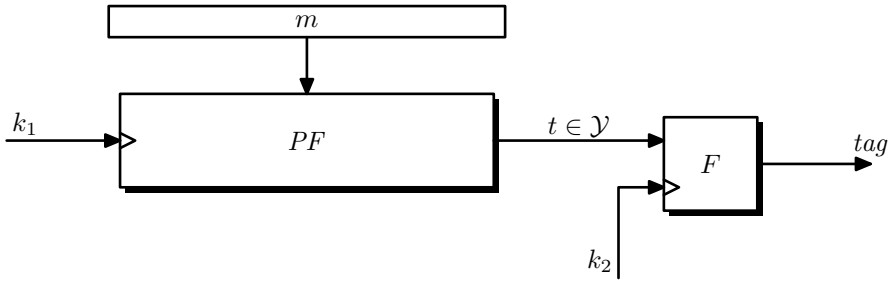


Figure 6.4: The encrypted PRF construction $EF(k, m)$

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 randomizaton.

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

$$EF\big((k_1, k_2),\ m\big) := F\big(k_2,\ PF(k_1,\ m)\big) \tag{6.16}$$

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:*

$$\text{if} \quad PF(k, x) = PF(k, y) \quad \text{then} \quad PF(k,\ x \,\|\, a) = PF(k,\ y \,\|\, a).$$

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 (6.16), 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*
>
> $$\text{PRF}\mathsf{adv}[\mathcal{A}, EF] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_1, F] + \text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{B}_2, PF] + \frac{Q^2}{2|\mathcal{Y}|}. \tag{6.17}$$

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 (6.17) 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 (6.17).

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 (6.16) to CBC and cascade.

The Encrypted-CBC PRF

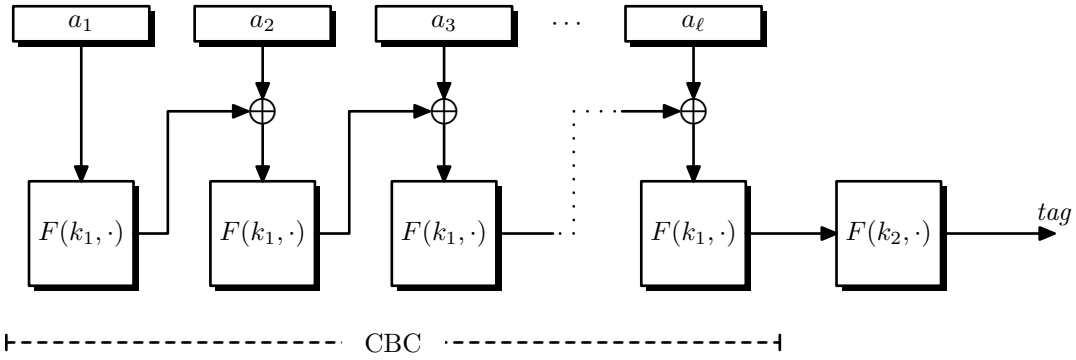
Applying *EF* to CBC results in a classic PRF (and hence a MAC) called **encrypted-CBC** or **ECBC** for short. This MAC is standardized by ANSI (see Section 6.9) and is used in the banking industry. The ECBC PRF uses the same underlying PRF F for both CBC and the final encryption. Consequently, ECBC is defined over $(\mathcal{K}^2,\ \mathcal{X}^{\leq \ell},\ \mathcal{X})$.

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 $(\mathcal{K}^2, \mathcal{X}^{\leq \ell}, \mathcal{X})$.*

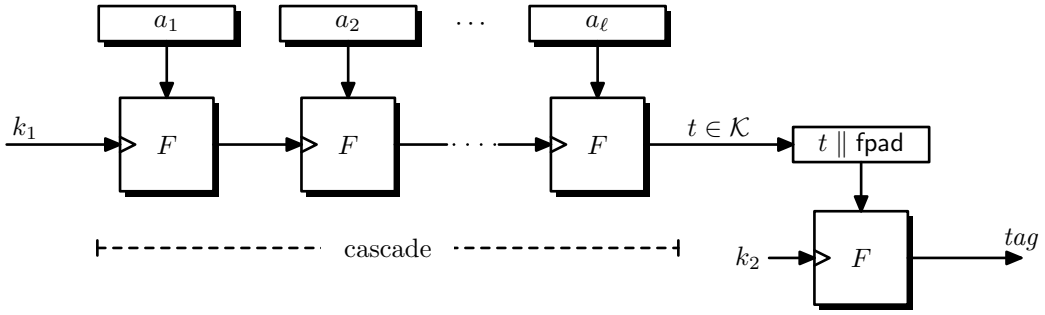
> *In particular, for every PRF adversary $\mathcal{A}$ that attacks* ECBC *as in Attack Game 4.2, and issues at most Q queries, there exist PRF adversaries $\mathcal{B}_1, \mathcal{B}_2$ that attack F as in Attack Game 4.2, and which are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\text{PRFadv}[\mathcal{A}, \text{ECBC}] \leq \text{PRFadv}[\mathcal{B}_1, F] + \text{PRFadv}[\mathcal{B}_2, F] + \frac{(Q(\ell+1))^2 + Q^2}{2|\mathcal{X}|}. \tag{6.18}$$

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.



(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

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 $g(t) := t \,\|\, \mathsf{fpad}$ 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 adversares $\mathcal{B}_1, \mathcal{B}_2$ that attack F as in Attack Game 4.2, and which are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, \mathrm{NMAC}] \leq (Q(\ell+1)) \cdot \mathrm{PRF}\mathsf{adv}[\mathcal{B}_1, F] + \mathrm{PRF}\mathsf{adv}[\mathcal{B}_2, F] + \frac{Q^2}{2|\mathcal{K}|}. \tag{6.19}$$

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 $\mathcal{X}^{\leq \ell}_{>0}$ denote the set of all non-empty strings over $\mathcal{X}$ of length at most ℓ. We say that a function $pf : \mathcal{M} \to \mathcal{X}^{\leq \ell}_{>0}$ is a **prefix-free encoding** if pf is injective (i.e., one-to-one) and the image of pf in is a prefix-free set.

Let PF be a prefix-free secure PRF defined over $(\mathcal{K}, \mathcal{X}^{\leq \ell}, \mathcal{Y})$ and $pf : \mathcal{M} \to \mathcal{X}^{\leq \ell}_{>0}$ 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

$$pf(m) := (\langle v \rangle, a_1, \ldots, a_v) \quad \in \mathcal{X}^{\leq \ell}_{>0}$$

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 $\mathcal{M} = \bar{\mathcal{X}}^{\leq \ell}_{>0}$. For $m = (a_1, \ldots, a_v) \in \mathcal{M}$, define

$$pf(m) := \big((a_1 \,\|\, 0),\ (a_2 \,\|\, 0),\ \ldots,\ (a_{v-1} \,\|\, 0),\ (a_v \,\|\, 1)\big) \quad \in \mathcal{X}^{\leq \ell}_{>0}$$

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 $rpf : \mathcal{K} \times \mathcal{M} \to \mathcal{X}^{\leq \ell}_{>0}$ 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 $\mathcal{M} := \mathcal{X}^{\leq \ell}_{>0}$. Define

$$rpf(k,\ (a_1, \ldots, a_v)) := \big(a_1, \ldots, a_{v-1}, (a_v \oplus k)\big) \in \mathcal{X}^{\leq \ell}_{>0}$$

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

$$m_0[v-1] \oplus k = m_1[v-1].$$

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

$$F\big((k, k_1),\ m\big) := PF\big(k, rpf(k_1, m)\big). \tag{6.20}$$

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 (6.20) 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

$$\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F] \leq \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_1, PF] + \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_2, PF] + Q^2\epsilon/2. \tag{6.21}$$

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.

$k \xleftarrow{\mathrm{R}} \mathcal{K}, \quad k_1 \xleftarrow{\mathrm{R}} \mathcal{K}_1$

upon receiving a signing query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) do:
  $x_i \leftarrow rpf(k_1, m_i) \in \mathcal{X}_{>0}^{\leq \ell}$
  $y_i \leftarrow PF(k, x_i)$
  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.

$k \xleftarrow{\mathrm{R}} \mathcal{K}, \quad k_1 \xleftarrow{\mathrm{R}} \mathcal{K}_1, \quad r_1, \ldots, r_Q \xleftarrow{\mathrm{R}} \mathcal{Y}$

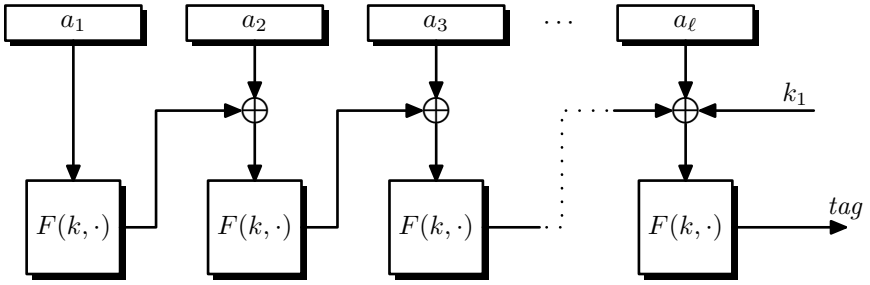
upon receiving a signing query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) do:
  $x_i \leftarrow rpf(k_1, m_i) \in \mathcal{X}_{>0}^{\leq \ell}$
(1) if $x_i \sim x_j$ for some $j < i$
    then $y_i \leftarrow r_i$
(2)   else $y_i \leftarrow PF(k, x_i)$
  send y_i to $\mathcal{A}$

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

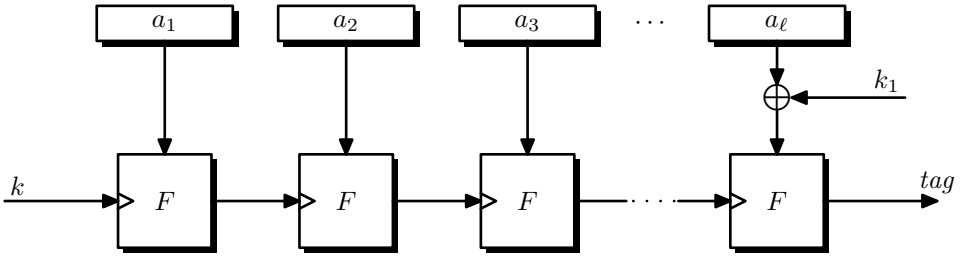
$$\big|\Pr[W_1] - \Pr[W_0]\big| \leq \Pr[Z_1]. \tag{6.22}$$

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



(a) *rpf* applied to CBC



(b) *rpf* applied to cascade

Figure 6.6: Secure PRFs using random prefix-free encodings

(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

$$|\Pr[Z_1] - \Pr[Z_2]| \leq \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_1, F] \tag{6.23}$$

and

$$|\Pr[W_1] - \Pr[W_2]| \leq \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}_2, F] \tag{6.24}$$

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

$$\Pr[Z_2] \leq Q^2\epsilon/2 \tag{6.25}$$

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

$$|\Pr[W_0] - \Pr[W_2]| = \mathrm{PRF}\mathsf{adv}[\mathcal{A}, F]. \tag{6.26}$$

Now combining (6.22)–(6.26) proves the theorem. □

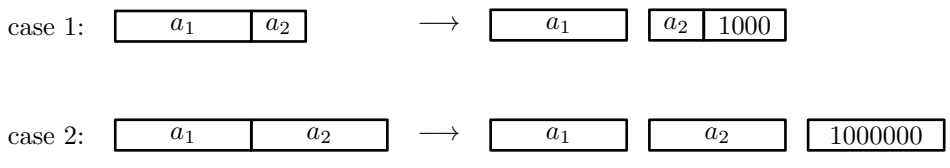


Figure 6.7: An injective function $inj : \{0,1\}^{\leq n\ell} \to \mathcal{X}^{\leq \ell+1}$

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:

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 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

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: CMAC signing algorithm

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 into a *bit-wise* secure PRF using a randomized prefix-free encoding $rpf : \mathcal{K} \times \mathcal{M} \to \mathcal{X}^{\leq \ell}_{>0}$ 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 positve 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.

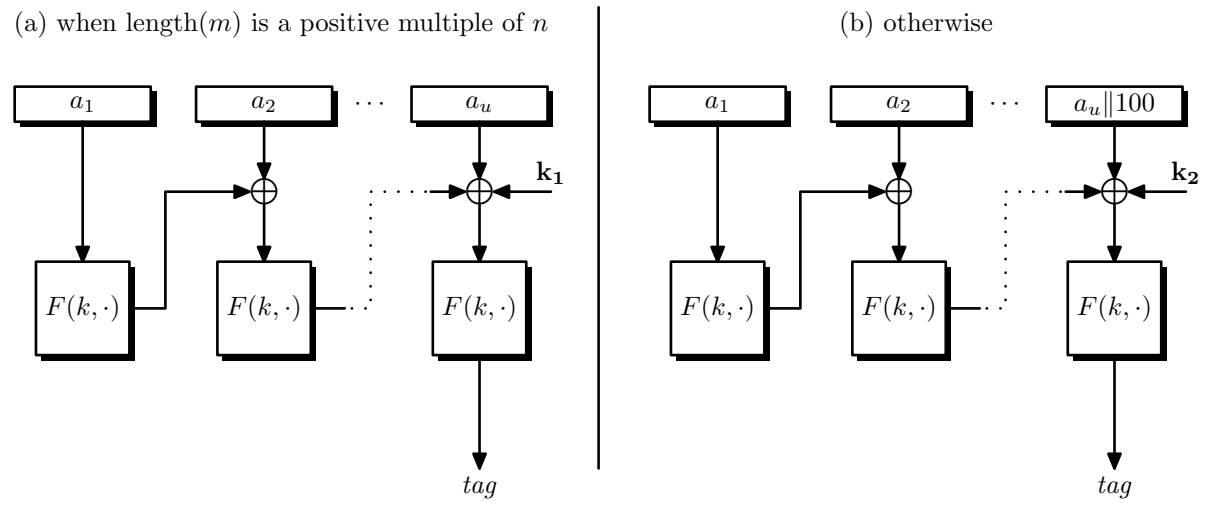


Figure 6.8: CMAC signing algorithm

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 $X \ll 1$ 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 msb$(L) = 0$ then $k_1 \leftarrow (L \ll 1)$ else $k_1 \leftarrow (L \ll 1) \oplus R_n$
(2) if msb$(k_1) = 0$ then $k_2 \leftarrow (k_1 \ll 1)$ else $k_2 \leftarrow (k_1 \ll 1) \oplus R_n$
output k_0, k_1, k_2.

where 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 GF(2^n). For a 128-bit block size the defining polynomial for GF(2^{128}) corresponding to R_{128} is $g(X) := X^{128} + X^7 + X^2 + X + 1$. Exercise 6.16 explores 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 [36].

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

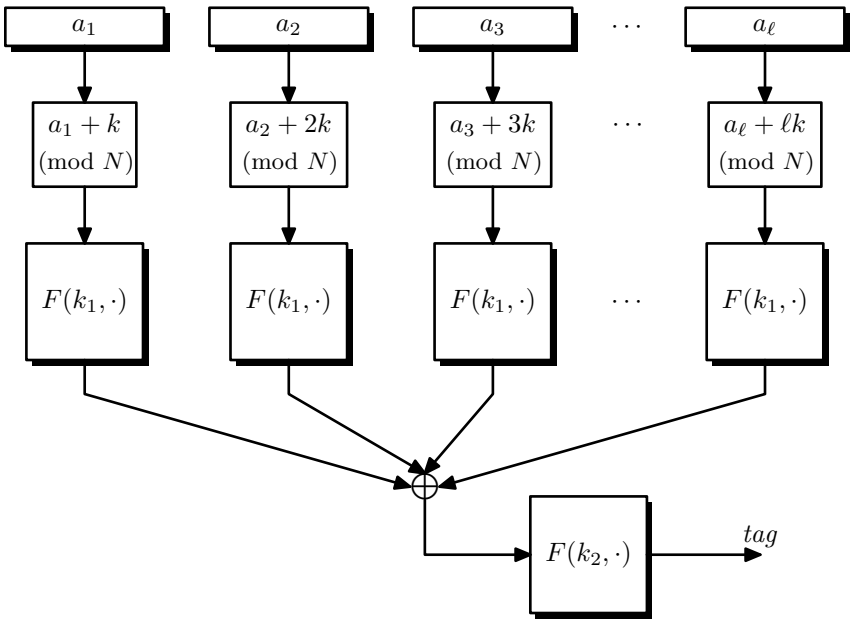


Figure 6.9: PMAC_0 construction

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 be a PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, where $\mathcal{X} := \{0,1\}^n$. Define $N := 2^n$. We can naturally interpret elements of $\mathcal{X}$ as numbers in $\{0, \ldots, N-1\}$, encoded in binary, and we will freely switch back and forth between these two interpretions.

We build a new PRF, called PMAC_0, that takes as input a key and a message in $\mathcal{X}^{\leq \ell}$ for some ℓ. It outputs a value in $\mathcal{X}$. A key for PMAC_0 consists of $k \in \{0, \ldots, N-1\}$ and $k_1, k_2 \in \mathcal{K}$. The PMAC_0 construction works as follows:

input: $m = (a_1, \ldots, a_v) \in \mathcal{X}^v$ for some $0 \leq v \leq \ell$, and
key $\vec{k} = (k, k_1, k_2)$ where $k \in \{0, \ldots, N-1\}$, $k_1 \in \mathcal{K}$, and $k_2 \in \mathcal{K}$
output: tag in $\mathcal{X}$

$\text{PMAC}_0(\vec{k}, m)$:
  $t \leftarrow 0^n \in \mathcal{X}$, $\quad mask \leftarrow 0$
  for $i \leftarrow 1$ to v do:
    $mask \leftarrow (mask + k) \bmod N \quad // \quad mask = (i \cdot k) \bmod N$
    $r \leftarrow (a_i + mask) \bmod N$
    $t \leftarrow t \oplus F\big(k_1, r\big)$
  output $F(k_2,\ t)$

The main loop adds the masks $k, 2k, 3k, \ldots$ to message blocks prior to evaluating the PRF. On a sequential machine this requires two additions modulo N per iteration. On a parallel machine each processor can independently compute $a_i + ik$ and then apply F. 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. *Let F be a secure PRF defined over $(\mathcal{K}, \mathcal{X}, \mathcal{X})$, where $\mathcal{X} = \{0,1\}^n$ and $N := 2^n$ is super-poly. Then $PMAC_0$ taking inputs in $\mathcal{X}^{\leq \ell}$ 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*
>
> $$\text{PRF}\mathsf{adv}[\mathcal{A}, PMAC_0] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_1, F] + \text{PRF}\mathsf{adv}[\mathcal{B}_2, F] + \frac{Q^2}{2N} + \frac{2Q^2\ell^2\lceil \log_2 \ell \rceil}{N}. \qquad (6.27)$$

As always, this block-wise PRF can be converted into a bit-wise PRF using the method discussed in Section 6.8.

PMAC: better than PMAC_0. Although PMAC_0 is well suited for a parallel architecture, there is room for improvement.

Fortunately, better implementations of the PMAC_0 approach are available. Examples include PMAC [14] and XECB [32], both of which are parallizable. PMAC, for example, provides the following improvements over PMAC_0:

- PMAC uses arithmetic in the finite field $\text{GF}(2^n)$ instead of arithmeic modulo N. Addition in $\text{GF}(2^n)$ is just an XOR. 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.

  By using arithmetic in $\text{GF}(2^n)$, one gets a somewhat better security bound (essentially, the $\log_2 \ell$ factor in the last term in (6.27) disappears).

- PMAC saves one application of $F(k_1, \cdot)$ using a trick described in Exercise 7.10.

- 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 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 character 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 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:

$t_1 \leftarrow D(k_2, t)$
$t_2 \leftarrow t_1 \ \oplus \ F(k_1,\ a_i + ik \bmod N) \ \oplus \ F(k_1,\ a'_i + ik \bmod N)$
$t' \leftarrow F_2(k_2, t_2)$

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; $\text{CRC32}(m)$ takes inputs $m \in \{0,1\}^{\leq \ell}$ and always outputs a 32-bit string. For this exercise, the only fact you need to know is that $\text{CRC32}(m_1) \oplus \text{CRC32}(m_2) = \text{CRC32}(m_1 \oplus m_2)$. Define the following MAC system (S, V):

$$S(k,m) := \{\ r \xleftarrow{R} \mathcal{R},\ t \leftarrow F(k,r) \oplus \text{CRC32}(m),\ \text{output } (r,t)\ \}$$
$$V(k,m,(r,t)) := \{\ \mathsf{accept} \text{ if } t = F(k,r) \oplus \text{CRC32}(m) \text{ and } \mathsf{reject} \text{ otherwise}\}$$

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 $\text{MAC}^{\text{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq \text{MAC}\mathsf{adv}[\mathcal{B}, \mathcal{I}] \cdot Q_\text{v}$. 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 $\text{MAC}\mathsf{adv}[\mathcal{B}, \mathcal{I}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}', F] + 1/|\mathcal{Y}|$. Putting these two statements together, we get

$$\text{MAC}^{\text{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq (\text{PRF}\mathsf{adv}[\mathcal{B}', F] + 1/|\mathcal{Y}|) \cdot Q_\text{v}$$

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

$$\text{MAC}^{\text{vq}}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}'', F] + Q_\text{v}/|\mathcal{Y}|.$$

6.3 (Multi-key MAC security). Just as we did for semantically secure encryption in Section 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 essentially 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 $k_1, \ldots, k_Q$.

(b) Show that any efficient adversary $\mathcal{A}$ that wins your new 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.

6.4. 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 messaage.

(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 collisions, 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 $k_1, \ldots, k_4$. Each user U_i verifies tags t_u and t_v, using its keys k_u, k_v, 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 cosntruct $\mathcal{I}$ 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. 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), $\ldots (x_Q, y_Q, z)$, 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). It is natural to define a weaker notion of security for a MAC in which we make it harder for the adversary to win; specifically, in order to win, the adversary must submit a valid tag on a new message. One can strengthen 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 $V(k, m, t) = \mathsf{accept}$ and m is not among the signing queries $m_1, m_2, \ldots$. In Attack Game 6.2, this means that for some verification query $(\hat{m}_j, \hat{t}_j)$, we have $V(k, \hat{m}_j, \hat{t}_j) = \mathsf{accept}$ and $\hat{m}_j$ is not among the signing queries $m_1, m_2, \ldots$ made prior to this verification query. These two modified winning conditions correspond to notions of security that we call *weak security without verification queries* and *weak security with verification queries*. Unfortunately, the analog of Theorem 6.1 does not hold relative to these weak security notions. In this exercise, you are to show this by giving an explicit counter-example. Assume the existence of a secure PRF (defined over any convenient input, output, and key spaces, of your choosing). Show how to "sabotage" this PRF to obtain a MAC that is weakly secure without verification queries but is not weakly secure with verification queries.

6.8. We showed that CBC is a prefix-free secure PRF but not a secure PRF. We showed that pre-pending 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. 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.

6.10. 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.

6.11. 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 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.

6.12. 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.

6.13. 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 4.7).

(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 casacde 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. 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. 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) $S_1(k,(a_1,a_2)) = S(k,a_1) \,\|\, S(k,a_2)$,

(b) $S_2(k,(a_1,a_2)) = S(k,a_1) \oplus S(k,a_2)$,

(c) $S_3((k_1,k_2),(a_1,a_2)) = S(k_1,a_1) \,\|\, S(k_2,a_2)$,

(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 duplicate an old record without being detected. To prevent addition or removal of records the data ownder 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.

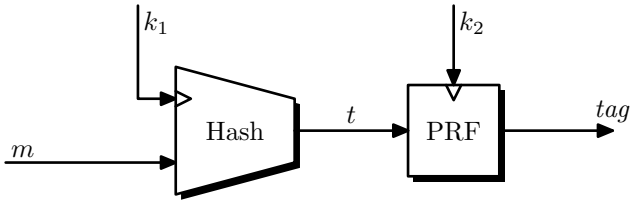


Figure 7.1: The hash-then-PRF paradigm

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 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 $\mathrm{UHF}\mathsf{adv}[\mathcal{A}, H]$, 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 hash function defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$, an alternative characterization of the ϵ-UHF property is the following (see Exercise 7.22):

$$\text{for every pair of distinct messages } m_0, m_1 \in \mathcal{M} \text{ we have } \Pr[H(k, m_0) = H(k, m_1)] \leq \epsilon \quad \text{where the probability is over the random choice of } k \in \mathcal{K}. \tag{7.1}$$

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 $k \xleftarrow{R} \mathcal{K}$ 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 any 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 randomly picks 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 $\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, H]$ and $\mathcal{B}$ will choose a colliding pair with probability at least $2/Q^2$. Hence, $\mathrm{UHF}\mathsf{adv}[\mathcal{B}, H]$ is at least $\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, H] \cdot (2/Q^2)$, 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 2.4.

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:*

$$\mathbf{K} = \{\mathcal{K}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \mathbf{M} = \{\mathcal{M}_{\lambda,\Lambda}\}_{\lambda,\Lambda}, \quad \text{and} \quad \mathbf{T} = \{\mathcal{T}_{\lambda,\Lambda}\}_{\lambda,\Lambda},$$

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 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 $m \in \mathcal{M}_{\lambda,\Lambda}$, runs in time bounded by a polynomial in λ, and outputs an element of $\mathcal{T}_{\lambda,\Lambda}$.*

In defining UHFs we parameterize Attack Game 7.1 by the security parameter λ. The advantage $\mathrm{UHF}\mathsf{adv}[\mathcal{A}, H]$ 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:

$$H_{\mathrm{poly}}\big(k,\ (a_1, \ldots, a_v)\big) := k^v + a_1 k^{v-1} + a_2 k^{v-2} + \ldots + a_{v-1} k + a_v \in \mathbb{Z}_p \tag{7.3}$$

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. $\quad t \leftarrow t \cdot k + a_i \ \in \mathbb{Z}_p$
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. $\quad t \leftarrow t \cdot k^4 + a_i \cdot k^3 + a_{i+1} \cdot k^2 + a_{i+2} \cdot k + a_{i+3} \ \in \mathbb{Z}_p$

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:

$$\begin{aligned} f(X) &:= X^u + a_1 X^{u-1} + a_2 X^{u-2} + \ldots + a_{u-1} X + a_u \\ g(X) &:= X^v + b_1 X^{v-1} + b_2 X^{v-2} + \ldots + b_{v-1} X + b_v \end{aligned} \tag{7.4}$$

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

$$H_{\text{fpoly}}\big(k,\ (a_1,\ldots,a_v)\big) := a_1k^{v-1} + a_2k^{v-2} + \cdots + a_{v-1}k + a_v \in \mathbb{Z}_p, \tag{7.5}$$

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 7.3 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. In contrast, the function H_{poly} defined in (7.3) is a statistical UHF for the input space $\mathbb{Z}_p^{\le\ell}$ containing messages of varying lengths.

Remark 7.1. The function H_{poly} takes inputs in $\mathbb{Z}_p^{\le\ell}$ 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 7.2 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 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

$$\mathcal{K}_{\lambda,p} = \mathbb{Z}_p, \quad \mathcal{M}_{\lambda,p} = \mathbb{Z}_p^{\le\ell(\lambda)}, \quad \text{and} \quad \mathcal{T}_{\lambda,p} = \mathbb{Z}_p,$$

where $\ell : \mathbb{Z} \to \mathbb{Z}^{\ge 0}$ is poly-bounded. By Lemma 7.2 we know that

$$\text{UHF}\mathsf{adv}[\mathcal{A}, H_{\text{poly}}](\lambda) \le \ell(\lambda)/2^{L(\lambda)}$$

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:

$$\text{if} \quad F(k,x) = F(k,y) \quad \text{then} \quad F(k,\ x \,\|\, a) = F(k,\ y \,\|\, a).$$

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 $f \xleftarrow{\mathrm{R}} \mathrm{Funs}[\mathcal{X}^{\leq \ell+1}, \mathcal{Y}]$.

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

$$\Pr\left[f(m_0) = f(m_1)\right] = \mathrm{UHF}\mathsf{adv}[\mathcal{A}, PF]$$

while in Experiment 1, since f is a random function and $m_0 \neq m_1$, we have

$$\Pr\left[f(m_0) = f(m_1)\right] = 1/|\mathcal{Y}|.$$

Hence, if $\mathcal{B}$ could query f at m_0 and m_1 it could distinguish between the two experiments with advantage $\left|\mathrm{UHF}\mathsf{adv}[\mathcal{A}, PF] - 1/|\mathcal{Y}|\right|$, 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, since $\mathcal{B}$ 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 $a \in \mathcal{X}$ so that $m_0 \,\|\, a$ is no longer a proper prefix of $m_1 \,\|\, a$. 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

$$PF(k,\ m_0) = PF(k,\ m_1) \qquad \Longrightarrow \qquad PF(k,\ m_0 \,\|\, a) = PF(k,\ m_1 \,\|\, a).$$

Since $m_0 \,\|\, a$ is no longer a proper prefix of $m_1 \,\|\, a$, our $\mathcal{B}$ is free to query f at both inputs and obtain 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 $a \neq a_{u+1}$
    $m'_0 \leftarrow m_0 \,\|\, a$ and $m'_1 \leftarrow m_1 \,\|\, a$
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

$$p_0 := \Pr\left[f(m'_0) = f(m'_1)\right] \geq \Pr\left[f(m_0) = f(m_1)\right] = \mathrm{UHF}\mathsf{adv}[\mathcal{A}, PF]. \tag{7.7}$$

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):

$$\mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}, PF] = |p_0 - p_1| \geq p_0 - p_1 \geq \mathrm{UHF}\mathsf{adv}[\mathcal{A}, PF] - 1/|\mathcal{Y}|,$$

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

$$\mathrm{MUHF}\mathsf{adv}[\mathcal{A}, PF] \leq \mathrm{PRF}^{\mathrm{pf}}\mathsf{adv}[\mathcal{B}, PF] + \frac{Q^2}{2|\mathcal{Y}|}. \tag{7.9}$$

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 $m'_i := m_i \,\|\, a$ 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 $t_j = t_j$ 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

$$p_0 \geq \text{MUHF}\mathsf{adv}[\mathcal{A}, PF].$$

In Experiment 1 the union bound implies that

$$p_1 \leq \frac{Q(Q-1)}{2|\mathcal{Y}|}.$$

Therefore,

$$\text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{B}, PF] = |p_0 - p_1| \geq p_0 - p_1 \geq \text{MUHF}\mathsf{adv}[\mathcal{A}, PF] - \frac{Q^2}{2|\mathcal{Y}|}$$

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

$$\text{MUHF}\mathsf{adv}[\mathcal{A}, F_{\text{CBC}}] \leq \text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{B}_1, F] + \frac{Q^2(\ell+1)^2 + Q^2}{2|\mathcal{Y}|} \quad \textit{and} \tag{7.10}$$

$$\text{MUHF}\mathsf{adv}[\mathcal{A}, F^*] \leq Q(\ell+1) \cdot \text{PRF}^{\text{pf}}\mathsf{adv}[\mathcal{B}_2, F] + \frac{Q^2}{2|\mathcal{Y}|}. \tag{7.11}$$

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 $(\mathcal{K}, \mathcal{X} \times \{1, \ldots, \ell\}, \mathcal{Y})$. The XOR-hash works as follows:

input: $k \in \mathcal{K}$ and $m = (a_1, \ldots, a_v) \in \mathcal{X}^{\leq \ell}$ 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.

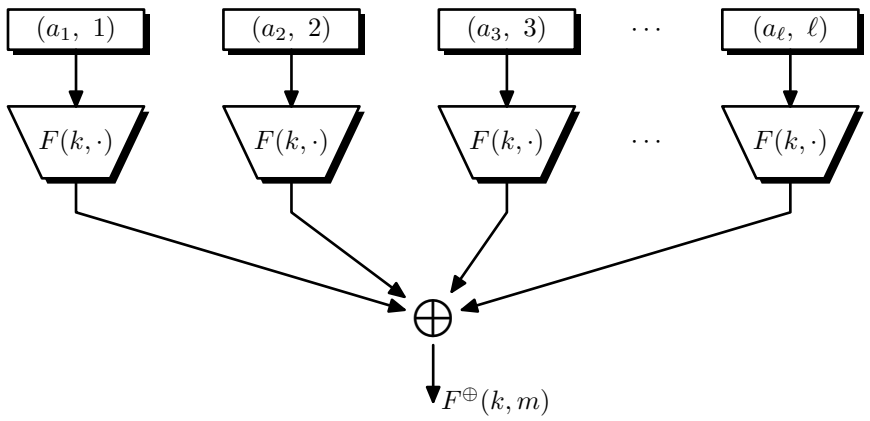


Figure 7.2: A parallel UHF from a small PRF

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

$$\mathrm{UHFadv}[\mathcal{A}, F^{\oplus}] \leq \mathrm{PRFadv}[\mathcal{B}, F] + \frac{1}{|\mathcal{Y}|}. \tag{7.12}$$

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 $\mathcal{X}^{\leq \ell}$. 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

$$\Pr[W_0] = \mathrm{UHFadv}[\mathcal{A}, F^{\oplus}]. \tag{7.14}$$

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

$$\left|\Pr[W_0] - \Pr[W_1]\right| \leq \mathsf{PRFadv}[\mathcal{B}, F] \tag{7.15}$$

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 $u \geq v$, 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 $U[i^*] \neq V[i^*]$, and this i^* does the job.

We can re-write (7.13) as

$$f(U[i^*], i^*) = \bigoplus_{i \neq i^*} f(U[i], i) \quad \oplus \quad \bigoplus_{j} f(V[j], j). \tag{7.16}$$

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 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.*

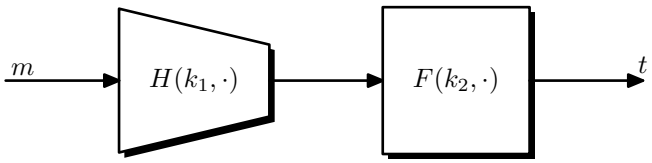


Figure 7.3: PRF-UHF composition: MAC signing

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

$$\mathrm{PRFadv}[\mathcal{A}, F'] \leq \mathrm{PRFadv}[\mathcal{B}_F, F] + (Q^2/2) \cdot \mathrm{UHFadv}[\mathcal{B}_H, H]. \tag{7.19}$$

More generally, there exists a Q-query UHF adversary $\mathcal{B}'_H$, which is an elementary wrapper around $\mathcal{A}$ such that

$$\mathrm{PRFadv}[\mathcal{A}, F'] \leq \mathrm{PRFadv}[\mathcal{B}_F, F] + \mathrm{MUHFadv}[\mathcal{B}'_H, H]. \tag{7.20}$$

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 7.27, 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 $\mathrm{PRFadv}[\mathcal{A}, F']$. 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 ith 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 $\mathrm{Funs}[\mathcal{X}, \mathcal{T}]$, which we implement as a faithful gnome (as in Section 4.4.2). The Game 1 challenger works as follows:

$k_1 \xleftarrow{R} \mathcal{K}_H, \quad t'_1, \ldots, t'_Q \xleftarrow{R} \mathcal{T}$
upon receiving the ith 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 $\mathrm{Funs}[\mathcal{X}, \mathcal{T}]$ — 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:

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \mathrm{PRFadv}[\mathcal{B}_F, F] \tag{7.21}$$

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 $\mathrm{MUHFadv}[\mathcal{B}'_H, H] = \Pr[Z]$.

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 $\mathrm{Funs}[\mathcal{M}, \mathcal{T}]$ and is therefore identical to an Experiment 1 PRF challenger with respect to F'. We obtain

$$\begin{aligned}\mathrm{PRF}\mathsf{adv}[\mathcal{A}, F'] = &\big|\Pr[W_2] - \Pr[W_0]\big| \leq \\ &\big|\Pr[W_2] - \Pr[W_1]\big| + \big|\Pr[W_1] - \Pr[W_0]\big| = \\ &\mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \mathrm{MUHF}\mathsf{adv}[\mathcal{B}'_H, H]\end{aligned}$$

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. Now, we may very well want to use for our PRF 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 digest space of the hash is equal to 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 that 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 7.25 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. To this end, we describe PMAC_0 as the composition of a PRF $\widehat{F}$ and a UHF $\widehat{H}$. The PRF $\widehat{F}$ is the final (bottom-most) application of F in Fig. 6.9 just prior to outputting the final tag. The UHF $\widehat{H}$ is the rest of Fig. 6.9. This $\widehat{H}$ is the XOR-hash construction (see Section 7.2.3) applied to a certain PRF F' described next.

Suppose F is defined over $(\mathcal{K},\ \{0,1\}^n,\ \{0,1\}^n)$ and set $N := 2^n$. The PRF F' is itself the result of applying PRF-UHF composition to the PRF F and a certain hash function H_1. This H_1 has key space $\{0,\ldots,N-1\}$, input space $\{0,\ldots N-1\} \times \{1,\ldots,\ell\}$, and output space $\{0,\ldots,N-1\}$. It is defined as follows:

$$H_1\big(k,\ (a,i)\big) := a + i \cdot k \bmod N.$$

The resulting PRF F', obtained by composing F with H_1, is shown in Fig. 7.4.

The fact that we are doing arithmetic modulo a number N that is not a prime means that we cannot just view H_1 as a special case of H_{poly}. Nevertheless, it is not hard to show that H_1 is a $(2\ell/N)$-UHF and you are asked to do this in Exercise 7.29. Then the PMAC_0 security theorem (Theorem 6.11) follows by the following chain of reasoning:

- first, PRF-UHF composition is applied to F and H_1 to show that F' is a secure PRF,
- next, applying the XOR-hash construction to F' proves that $\widehat{H}$ is a computational UHF,
- finally, PRF-UHF composition is used again, this time applied to F and $\widehat{H}$, to argue that PMAC_0 is a secure PRF.

While this analysis is easy, and is enough to show that PMAC_0 is secure, it does not quite yield the concrete security bound in (6.27): we get an additive error term of $2Q^2\ell^3/N$ instead of the $2Q^2\ell^2\lceil\log_2 \ell\rceil/N$ in (6.27). However, Exercise 7.29 describes a more refined analysis that yields this better security bound.

One can avoid this more refined analysis and get an even better security bound (essentially erasing the $\log_2 \ell$ factor in (6.27)) by working in a finite field: either working modulo a prime number that is close to N or working in $\mathrm{GF}(2^n)$. For example, one could choose a prime p a little bit smaller than N and define $H'(k,(a,i)) = (a + ik) \bmod p$. We could also choose a prime p a bit larger than N and define $H''(k,(a,i)) = (a + (ik \bmod p)) \bmod N$. There are tradeoffs among all of these options. Working directly modulo N as in PMAC_0 is the easiest to implement and the fastest on some architectures, but gives a slightly worse security bound (the $\log_2 \ell$ factor in (6.27)).

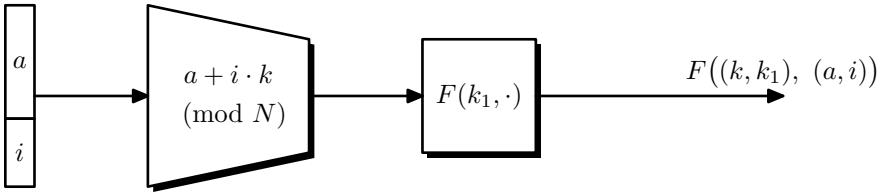


Figure 7.4: The PRF F' in the analysis of PMAC_0

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 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 $1/2^{127}$.

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 7.7 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 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. 7.5):

- For key (k_1, k_2) and message m we define

$$
\begin{array}{l}
S\big(\ (k_1, k_2),\ m\ \big) := \\
\quad r \xleftarrow{\text{R}} \mathcal{R} \\
\quad v \leftarrow H(k_1, m) + F(k_2, r) \quad \in \mathbb{Z}_N \quad // \quad \text{addition modulo } N \\
\quad \text{output } (r, v)
\end{array}
$$

- For key (k_1, k_2), message m, and tag (r, v) we define

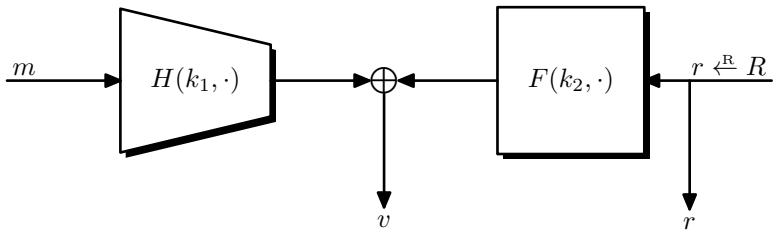


Figure 7.5: Carter-Wegman MAC signing algorithm

$V\big(\ (k_1, k_2),\ m,\ (r, v)\ \big) :=$
$\quad v^* \leftarrow H(k_1, m) + F(k_2, r) \quad \in \mathbb{Z}_N \quad //$ addition modulo N
$\quad$ 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

$$E(k, m) := \big\{ r \xleftarrow{R} \mathcal{R},\ \text{output } (r,\ m + F\big(k, r)\big) \big\} \qquad \text{and} \qquad D\big(k, (r, c)\big) := c - F(k, r)$$

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:

$$S\big((k_1, k_2),\ m\big) := E(k_2,\ H(k_1, m)\big)$$

$$V\big((k_1, k_2),\ m,\ t\big) := \begin{cases} \mathsf{accept} & \text{if } D(k_2, t) = H(k_1, m), \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

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 9.12 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 unpredicability). 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 $k \xleftarrow{R} \mathcal{K}$ 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 $\text{DUF}\mathsf{adv}[\mathcal{A}, H] \leq \epsilon$ 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 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 $\delta \in \mathcal{T}$ we have
$\Pr[H(k, m_1) - H(k, m_0) = \delta] \leq \epsilon$ *where 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:

$$H_{\text{xpoly}}(k, m) := k \cdot H_{\text{poly}}(k, m) = k^{v+1} + a_1 k^v + a_2 k^{v-1} + \ldots + a_v k \in \mathbb{Z}_p \tag{7.23}$$

Lemma 7.8. *The function H_{xpoly} over $(\mathbb{Z}_p,\ (\mathbb{Z}_p)^{\leq \ell},\ \mathbb{Z}_p)$ defined in (7.33) 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 an 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*
>
> $$\text{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \text{DUF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{Q^2}{2|\mathcal{R}|} + \frac{1}{|\mathcal{T}|}. \tag{7.24}$$

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 7.16 and 7.17. □

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 reviewing the challenger in the MAC Attack Game 6.1 with respect to $\mathcal{I}_{\mathrm{CW}}$. We implement the challenger in this game as follows:

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 ith signing query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) 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

At the end of the game, $\mathcal{A}$ outputs a message-tag pair $(m, (r, v))$ that is not among the signed message-tag pairs produced by the challenger. The winning condition in this game is defined to be the result of the following subroutine:

if $v = H(k_1, m) + F(k_2, r)$
  then return win
  else return lose

Then, by construction

$$\mathrm{MACadv}[\mathcal{A}, \mathcal{I}_{\mathrm{CW}}] = \Pr[W_0]. \tag{7.25}$$

Game 1. We next play the usual "PRF card," replacing the function $F(k_2, \cdot)$ by a truly random function f in $\mathrm{Funs}[\mathcal{R}, \mathcal{T}]$, 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 ith signing query $m_i \in \mathcal{M}$ (for $i = 1, 2, \ldots$) 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

Suppose $\mathcal{A}$ makes exactly $s \leq Q$ signing queries before outputting its forgery attempt $(m, (r, v))$. The subroutine for the winning condition becomes:

(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 return win
else return 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 $\text{Funs}[\mathcal{R}, \mathcal{T}]$. 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:

$$\big|\Pr[W_1] - \Pr[W_0]\big| = \text{PRFadv}[\mathcal{B}_F, F] \tag{7.26}$$

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) in the winning subroutine:

if $r_i = r_j$ for some $1 \leq i < j \leq s$ then return lose

Let Z to 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 $j = 1, \ldots, s$.

Thus, we have $W_2 = W'_2 \cup W''_2$, and it suffices to analyze these events separately, since

$$\Pr[W_2] \leq \Pr[W'_2] + \Pr[W''_2]. \tag{7.28}$$

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

$$\Pr[W_2'] \leq \mathrm{DUF}\mathsf{adv}[\mathcal{B}_H, H] \tag{7.30}$$

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 $(m, (r, v))$, $\mathcal{B}_H$ determines if $r = r_j$ and $m \neq m_j$ for some $j = 1, \ldots, s$; if so, $\mathcal{B}_H$ outputs the triple $(m_j, m, v_j - v)$. The bound (7.30) is now clear.

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 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 7.25 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 7.24), although this is less convenient, because inputs to the hash will have to 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 $\mathrm{GF}(2^n)$, as discussed in Remark 7.4.

7.5 Nonce-based MACs

In the Carter-Wegman construction in Section 7.4, the only essential property we need for these randomizers are that they are distinct. Similar to what we did in Section 5.5, we can study nonce-based MACs: 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 $\mathcal{N} \in \mathcal{N}$:

$$\Pr\left[V(k,\ m,\ S(k,\ m,\ \mathcal{N}),\ \mathcal{N}\ ) = \mathsf{accept}\right] = 1.$$

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*

$$\mathrm{nMACadv}[\mathcal{A},\mathcal{I}_{\mathrm{CW}}] \le \mathrm{PRFadv}[\mathcal{B}_F,F] + \mathrm{DUFadv}[\mathcal{B}_H,H] + \frac{1}{|\mathcal{T}|}.$$

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 7.15 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 7.16 and 7.17.

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 $k \xleftarrow{R} \mathcal{K}$ 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, it is also an ϵ-UHF; moreover, if $\mathcal{T} = \mathbb{Z}_N$ for some N, then it is also 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 7.18). 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

$$\mathrm{PUF}\mathsf{adv}[\mathcal{A}, H] \leq \mathrm{DUF}\mathsf{adv}[\mathcal{B}, H] \leq \epsilon. \quad \square$$

In particular, Lemma 7.11 shows how to convert the function H_{xpoly}, defined in (7.33), into a an $(\ell + 1)/p$-PUF. We obtain the following hash function defined over $(\mathbb{Z}_p^2, \mathbb{Z}_p^{\leq \ell}, \mathbb{Z}_p)$:

$$H(k, m) := H_{\mathrm{xpoly}}(k_1, m) + k_2 = k_1^{v+1} + a_1 k_1^v + a_2 k_1^{v-1} + \ldots + a_v k_1 + k_2 \in \mathbb{Z}_p \tag{7.33}$$

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* $\mathrm{MAC}_1\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq \epsilon$.

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.

7.8 Notes

Citations to the literature to be added.

7.9 Exercises

7.1. Let H_1 be an ϵ_1-UHF defined over $(\mathcal{K}_1, \mathcal{X}, \mathcal{Y})$. Let H_2 be an ϵ_2-UHF 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 $H((k_1, k_2), x) := H_2(k_2, H(k_1, x))$. Show that H is an $(\epsilon_1 + \epsilon_2)$-UHF.

7.2. 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 insecure, and in particular an attacker can find two messages m_0, m_1 each of length two blocks that are guaranteed to collide.

7.3. 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

$$H_{\text{fxpoly}}(k, (a_1, \ldots, a_\ell)) := k \cdot H_{\text{fpoly}}(k, (a_1, \ldots, a_\ell)) = a_1 k^\ell + a_2 k^{v-1} + \cdots + a_\ell k \in \mathbb{Z}_p$$

is an (ℓ/p)-DUF.

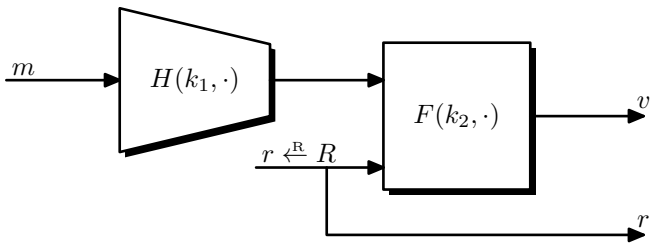


Figure 7.6: Randomized PRF-UHF composition: MAC signing

7.4. Let H be a hash function defined over $(\mathcal{K}, \mathcal{M}, \mathbb{Z}_N)$. We construct a new 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.

7.5. Show that if F is a secure PRF against non-adaptive adversaries (see Exercise 4.7), 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.

7.6. 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. Show how to modify the PRF so that it remains a secure MAC, but the MAC-UHF composition is insecure.

7.7 (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 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

$$S\big((k_1, k_2),\ m\big) := \big\{r \xleftarrow{R} \mathcal{R},\ x \leftarrow H(k_1, m),\ v \leftarrow F\big(k_2, (r, x)\big),\ \text{output } (r, v)\big\} \quad \text{(see Fig. 7.6)}$$

$$V\big((k_1, k_2),\ m,\ (r, v)\big) := \begin{cases} \mathsf{accept} & \text{if } x \leftarrow H(k_1, m),\ v = F\big(k_2, (r, x)\big) \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

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

$$\begin{aligned} \mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}] \leq \mathrm{PRF}\mathsf{adv}[\mathcal{B}_F, F] + \mathrm{UHF}\mathsf{adv}[\mathcal{B}_H, H] + \frac{Q^2}{2|\mathcal{R}|}\mathrm{UHF}\mathsf{adv}[\mathcal{B}'_H, H] \\ + \frac{Q^2}{2|\mathcal{R}||\mathcal{T}|} + \frac{1}{|\mathcal{T}|}. \end{aligned} \tag{7.34}$$

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

$$\text{MACadv}[\mathcal{A}, \mathcal{I}] \leq \text{PRFadv}[\mathcal{B}_F, F] + 4\epsilon. \tag{7.35}$$

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**.

7.8 (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 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.

7.9 (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).

7.10. This exercise shows how to save one application of the underlying PRF F in the PMAC_0 construction. With notation as in Section 6.11, we can write the definition of PMAC_0 as

$$\text{PMAC}_0((k,k_1,k_2),(a_1,\ldots,a_v)) := F\Big(\ k_2,\ \bigoplus_{i=1}^{v} F\big(k_1,(a_i+ik) \bmod N\big)\ \Big).$$

Using part (a) of Exercise 7.9, show that

$$H'((k,k_1),(a_1,\ldots,a_v)) := \bigoplus_{i=1}^{v-1} F\big(k_1,(a_i+ik) \bmod N)\big) \oplus a_v$$

is a computational UHF (for zero-length inputs, we define the output of H' to be zero). From this, conclude that

$$\text{PMAC}'_0((k,k_1,k_2),(a_1,\ldots,a_v)) := F\Big(\ k_2,\ \bigoplus_{i=1}^{v-1} F\big(k_1,(a_i+ik) \bmod N)\big) \oplus a_v\ \Big).$$

is a secure PRF.

7.11. 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.

7.12 (Tweakable block ciphers). Continuing with Exercise 4.12, 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 5.15) to handle sectors whose bit length is not a multiple of n.

7.13. 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.

7.14. 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

$$\text{MAC}^{\text{vq}}\mathsf{adv}[\mathcal{A},\mathcal{I}_{\text{CW}}] \le \text{PRF}\mathsf{adv}[\mathcal{B}_F,F] + Q_{\text{v}}\cdot\text{DUF}\mathsf{adv}[\mathcal{B}_H,H] + \frac{Q_{\text{s}}^2}{2|\mathcal{R}|} + \frac{Q_{\text{v}}}{|\mathcal{T}|}.$$

7.15. 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

$$\text{nMACadv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{PRFadv}[\mathcal{B}_F, F] + \text{DUFadv}[\mathcal{B}_H, H] + \frac{1}{|\mathcal{T}|},$$

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

$$\text{nMACadv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{BCadv}[\mathcal{B}_{\mathcal{E}}, \mathcal{E}] + \frac{Q^2}{2N} + \text{DUFadv}[\mathcal{B}_H, H] + \frac{1}{N},$$

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 4.23, show that assuming $Q^2 < N$, we have the following security bound:

$$\text{nMACadv}[\mathcal{A}, \mathcal{I}_{\text{CW}}] \leq \text{BCadv}[\mathcal{B}_{\mathcal{E}}, \mathcal{E}] + 2 \cdot \left(\text{DUFadv}[\mathcal{B}_H, H] + \frac{1}{N}\right).$$

7.16 (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 $\aleph$ and the tag on a different one-block message m_2 using *the same* nonce $\aleph$, then the MAC system becomes insecure: the adversary can forge the MAC an any message of his 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 that 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, note that 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.

7.17 (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 $\aleph$, we define

$$S((k_1, k_2, k_3), m, \aleph) := E(k_3,\ H(k_1, m) + E(k_2, \aleph)\ )$$

- For key (k_1, k_2, k_3), message m, tag v, and nonce $\aleph$, we define

$V((k_1, k_2, k_3), m, v, \aleph) :=$
$\quad v^* \leftarrow E(k_3,\ H(k_1, m) + E(k_2, \aleph)\ )$
$\quad$ if $v = v^*$ output accept; otherwise output 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 7.15, 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 $\mathrm{BCadv}[\mathcal{B}, \mathcal{E}] + 2(\epsilon + 1/N)$, 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 $\mathrm{BCadv}[\mathcal{B}, \mathcal{E}] + (Q+1)^2\epsilon + 2/N$, 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 4.23 and 7.4 may be helpful.

7.18. Show that H_{xpoly} defined in (7.33) is not a good PUF by exhibiting an adversary that wins Attack Game 7.5 with probability 1.

7.19. 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 $\mathrm{PRFadv}[\mathcal{A}, F] \leq \epsilon$. If $\epsilon = 0$, we call F a **pairwise independent function**, or **PIF**.

(a) 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

$$\Pr[F(k, x_0) = y_0 \quad \wedge \quad F(k, x_1) = y_1] = \frac{1}{|Y|^2},$$

where the probability is over the random choice of $k \in \mathcal{K}$. Show that F is a PIF.

(b) Let p be a prime. Let H be the hash function defined over $(\mathcal{K}, \mathbb{Z}_p^\ell, \mathbb{Z}_p)$ as follows:

$$H((k_0, k_1, \ldots, k_\ell), (a_1, \ldots, a_\ell)) := k_0 + \sum_i a_i k_i \in \mathbb{Z}_p.$$

Show that H is a PIF.

(c) For positive integer m, let $I_m := \{0, \ldots, m-1\}$. Let n be a positive integer and set $N := 2^n$. Consider the hash function H defined over $(I_{N^2}^{\ell+1}, I_N^\ell, I_N)$ as follows:

$$H((k_0, k_1, \ldots, k_\ell), (a_1, \ldots, a_\ell)) := \left\lfloor \left(\left(k_0 + \sum_i a_i k_i\right) \bmod N^2 \right) / N \right\rfloor.$$

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.

(d) 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.

(e) Show that any ϵ-APIF is an $(\epsilon + |\mathcal{Y}|)$-PUF.

(f) Using an appropriate APIF, show how to construct a probabilistic cipher that is unconditionally CPA secure provided the adversary can make at most two queries in Attack Game 5.2.

7.20 (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 shown in Fig. **??**:

$$\begin{array}{rl} H(k,(a_1,\ldots,a_v)) := & h \leftarrow k \\ & \text{for } i \leftarrow 1 \text{ to } v \text{ do:} \quad h \leftarrow \pi(a_i \oplus h) \\ & \text{output } h \end{array}$$

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 $\alpha \in \mathcal{X}^{\leq \ell}_{>0}$. 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.

7.21. 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:

$$\begin{aligned} S'((k_1,k_2),m) &:= \{\ r \xleftarrow{\text{R}} \mathcal{R};\ t \xleftarrow{\text{R}} S(k_1,m);\ t' \leftarrow F(k_2,r) \oplus t;\ \text{ output } (r,t')\ \} \\ V'((k_1,k_2),m,(r,t')) &:= \{\ t \leftarrow F(k_2,r) \oplus t';\ \text{ output } V(k_1,m,t)\ \} \end{aligned}$$

Show that $\mathcal{I}'$ is a secure (many time) MAC.

7.22. Let H be a 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 $\mathcal{A}$ will inevitably be *very* inefficient.

7.23. For positive integer m, let $I_m := \{0, \ldots, m-1\}$ and $I_m^* := \{1, \ldots, m-1\}$.

(a) Let N be a positive integer and p be a prime. Consider the 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), and Exercise 7.1 and 7.3, you are to design a hash function with "nearly optimal" collision probability, but with much smaller keys.

Let N and ℓ be positive integers. Let α be a number with $0 < \alpha < 1$. Design a $(1+\alpha)/N$-UHF H with message space I_N^ℓ and output space I_N. The keys for H should be short: encoded as a bit string, each key should be of length $O(\log(N\ell/\alpha))$.

7.24. 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 m, let $I_m := \{0, \ldots, m-1\}$.

Let p and N be integers greater than 1. Let H be a hash function defined over $(\mathcal{K}, \mathcal{M}, I_p)$. Let H' be the hash function defined over $(\mathcal{K}, \mathcal{M}, I_N)$ as follows: $H'(k, m) := H(k, m) \bmod N$.

(a) Show that if $p < N/2$ and H is an ϵ-DUF modulo p, then H' is an ϵ-DUF modulo N.

(b) Suppose that $p \geq N$ 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 $\epsilon' = 4\alpha/N$.

7.25. 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 m, we let $I_m := \{0, \ldots, m-1\}$.

Let $1 < N \leq p$, where p is prime.

(a) H^*_{fxpoly} is the hash function defined over (I_p, I_N^ℓ, I_N) as follows:

$$H^*_{\text{fxpoly}}(k, (a_1, \ldots, a_\ell)) := \Big((a_1 k^\ell + \cdots + a_\ell k) \bmod p \Big) \bmod N.$$

Show that H^*_{fxpoly} is a $4\ell/N$-DUF modulo N.

(b) H^*_{xpoly} is the 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 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 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 1. 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.

7.26. With notation as in the previous exercise, show that if $(3/2)N \leq p < 2N$, the 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 $\ell = 2$.

7.27. 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, when $\ell = 2$, then 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 $\hat{k} = k$.

7.28 (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, where each query is of the form $(\cdot, j)$ for some $j = 1, \ldots, \ell$, and in addition, for each $j = 1, \ldots, \ell$, at most Q of these queries are of the form $(\cdot, j)$. Let us call such an adversary $\mathcal{B}$ a (Q, ℓ)-**adversary**.

7.29 (PMAC$_0$ analysis). This exercise develops an analysis of the PMAC$_0$ scheme, presented in Section 6.11 Throughout, we assume that n and ℓ are positive integers with $\ell \leq 2^n$, and define $N := 2^n$ and $\mathcal{L} := \{1, \ldots, \ell\}$. For any nonzero integer s, write $s = \alpha(s) \cdot \beta(s)$, where $\alpha(s)$ is a positive power of 2 and $\beta(s)$ is odd. For completeness, we define $\alpha(0) = 0$.

(a) Let H_0 be the keyed hash function defined over $(\mathbb{Z}_N, \mathcal{L}, \mathbb{Z}_N)$ by $H_0(k, j) := j \cdot k \in \mathbb{Z}_N$. Show that if j and j' are distinct elements of $\mathcal{L}$, and $\delta \in \mathbb{Z}_n$, then if we pick $k \in \mathbb{Z}_N$ at random, we have $\Pr[H_0(j_1) - H_0(j_0)] \leq 2\alpha(j_1 - j_0)/N$. In particular, show that H_0 is an $(2\ell/N)$-DUF.

(b) Let H_1 be the keyed hash function defined over $(\mathbb{Z}_N, \mathbb{Z}_N \times \mathcal{L}, \mathbb{Z}_N)$ by $H_1(k, (a, j)) := a + j \cdot k \in \mathbb{Z}_N$. Combine part (a) with the result of Exercise 7.4 to show that H_1 is an $2\ell/N$-UHF. Using this result, one can prove Theorem 6.11, but not the security bound stated in (6.27).

Note the dependence on ℓ in the collision probability: in general, without a restriction of ℓ, this hash is completely insecure (see Exercise 7.2 below).

(c) Show that any (Q, ℓ)-adversary (see Exercise 7.28) attacking H_1 as in the multi-query Attack Game 7.2 has advantage at most $2Q^2T/N$, where $T := \sum_{j_0 \in \mathcal{L}} \sum_{j_1 \in \mathcal{L}} \alpha(j_1 - j_0)$.

(d) For non-negative integer m, define $S(m) := \sum_{i=1}^{2^m - 1} \alpha(i)$. Show that $S(m) = m2^{m-1}$. From this, deduce that $T \leq \ell^2 \lceil \log_2 \ell \rceil$.

(e) Using parts (c) and (d) and the result of Exercise 7.28, prove the security bound (6.27). For this, you will also want to use the fact (which is fairly obvious from the proof) that in Theorem 7.7, if $\mathcal{A}$ is a (Q, ℓ)-adversary, then the adversary $\mathcal{B}'_H$ in (7.20) is also a (Q, ℓ)-adversary.

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, **SHA-256** is a function that hashes long messages into 256-bit digests. It is believed that finding collisions for SHA-256 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 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 then apply S to

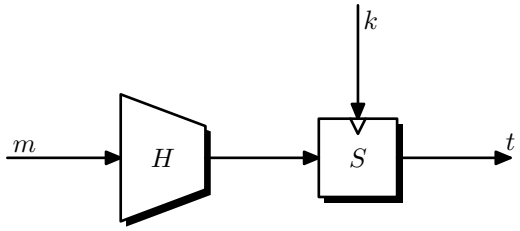


Figure 8.1: Hash-then-MAC construction

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 7.6. 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 a file on disk that is readable by multiple users. The 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 message n times, once for each user. See also Exercise 6.4 for more on this problem.

File integrity. Another application for collision resistance is file integrity also discussed in the introduction of Chapter 6. Consider a set of n critical files that change infrequently, such as certain operating system files. We want a method to verify that these files are not 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 read-only memory so that this storage area only

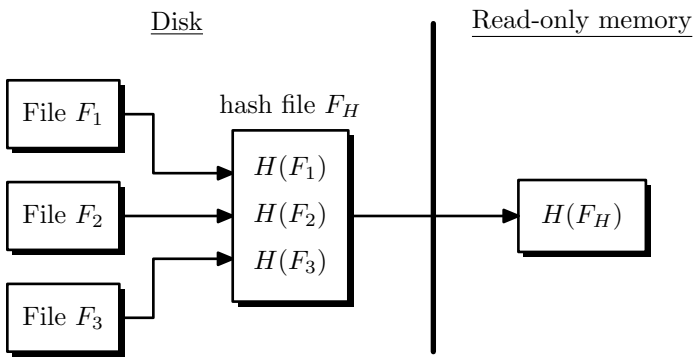


Figure 8.2: File integrity using small read-only memory

contains n short hashes. We can then check integrity of a file F by rehashing 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. The *TripWire* malware protection system [39] uses this mechanism to protect critical system files.

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. When there are many files to protect the amount of read-only memory needed could become large. We can greatly reduce the size of read-only memory by viewing the entire set of file 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. Then read-only memory contains a single hash value. To verify file 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. We describe a more efficient solution using authentication trees in Section **??**.

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 8.10.2 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 is just 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 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 $\mathsf{CRadv}[\mathcal{A}, H]$, 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 $\mathsf{CRadv}[\mathcal{A}, H]$ 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 2.4.

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:*

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

such that

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

[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.

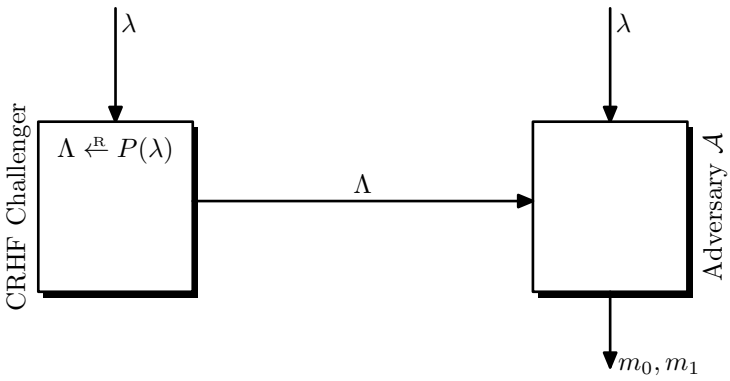


Figure 8.3: Asymptotic version of Attack Game 8.1

2. *Algorithm H is a deterministic algorithm that on input $\lambda \in \mathbb{Z}_{\geq 1}$, $\Lambda \in \mathrm{Supp}(P(\lambda))$, and $m \in \mathcal{M}_{\lambda,\Lambda}$, runs in time bounded by a polynomial in λ, and 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 $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H]$ is then a function of λ. Definition 8.1 should be read as saying that $\mathrm{CR}\mathsf{adv}[\mathcal{A}, H](\lambda)$ 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. SHA-256, 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:

$$S'(k, m) := S(k,\ H(m)\ ) \quad \text{and} \quad V'(k, m) := V(k,\ H(m)\ ) \tag{8.1}$$

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

$$\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}'] \leq \mathrm{MAC}\mathsf{adv}[\mathcal{B}_{\mathcal{I}}, \mathcal{I}] + \mathrm{CR}\mathsf{adv}[\mathcal{B}_H, H].$$

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:

$$\mathrm{MAC}\mathsf{adv}[\mathcal{A}, \mathcal{I}'] \ = \ \Pr[X] \ \leq \ \Pr[X \wedge \neg Y] + \Pr[Y] \ = \ \Pr[Z] + \Pr[Y] \tag{8.2}$$

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:

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)

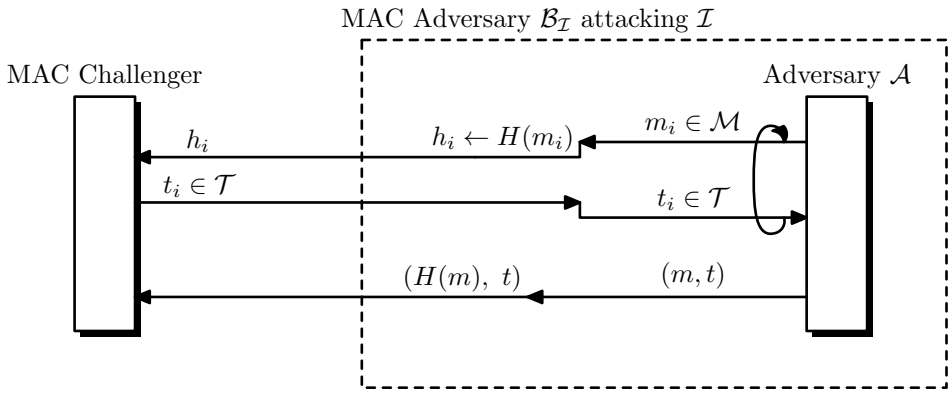


Figure 8.4: Adversary $\mathcal{B}_\mathcal{I}$ in the proof of Theorem 8.1

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, $\mathsf{CRadv}[\mathcal{B}_H, H] = \Pr[Y]$ 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, $\mathsf{MACadv}[\mathcal{B}_\mathcal{I}, \mathcal{I}] = \Pr[Z]$, 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 SHA-256. 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 colli-

sion 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, $m_1, \ldots, m_s \xleftarrow{R} \mathcal{M}$, 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 8.7, 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 SHA-256 output a 256-bit digest. Other hash functions, such as SHA-384 and SHA-512, 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 where $\mathcal{Y} := \{0,1\}^\ell$ and $\mathcal{X} := \{0,1\}^n$. 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):

input: $M \in \{0,1\}^{\leq L}$
output: a tag in $\mathcal{X}$

$\hat{M} \leftarrow M \,\|\, \text{PB}$ // 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

$$\hat{M} = m_1 \,\|\, m_2 \,\|\, \cdots \,\|\, m_s \quad \text{where} \quad m_1, \ldots, m_s \in \{0,1\}^\ell$$

$t_0 \leftarrow \text{IV} \in \mathcal{X}$
for $i = 1$ to s do:
    $t_i \leftarrow h(t_{i-1}, m_i)$
output t_s

The function SHA-256 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 $\text{IV} = 0^n$, but usually the IV is set to some complicated string. For example, SHA-256 uses a 256-bit IV whose value in hex is

$$\text{IV} := \text{6A09E667 BB67AE85 3C6EF372 A54FF53A 510E527F 9B05688C 1F83D9AB 5BE0CD19}.$$

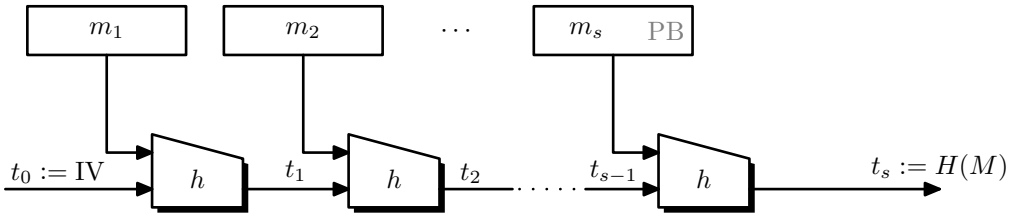


Figure 8.5: The Merkle-Damgård iterated hash function

- 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 '$100\ldots00$' 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 '$100\ldots00$' 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 $(\{0,1\}^{\leq L}, \mathcal{X})$, 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*
>
> $$\text{CRadv}[\mathcal{A}, H_{\text{MD}}] = \text{CRadv}[\mathcal{B}, h].$$

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 $\{0,1\}^{\leq L}$ 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 $t'_0, t'_1, \ldots, t'_s \in \mathcal{X}$ 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 $t_{u-1} \neq t'_{v-1}$ or $m_u \neq m'_v$ 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 $t_{u-2} \neq t'_{u-2}$ or $m_{u-1} \neq m'_{u-1}$ 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 8.8 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 [15] 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 $H_{12}(M) := H_1(M) \,\|\, H_2(M) \in \{0,1\}^{2n}$. 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 $M = m_0 m_1 \ldots m_{b-1}$. 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 $m_0, \ldots, m_{b-1}$ by the corresponding blocks in $m_0', \ldots, m_{b-1}'$ without changing $H(M)$. As a result we obtain $s = 2^b$ messages

$$\begin{array}{c} m_0 m_1 \ldots m_{b-1} \\ m_0' m_1 \ldots m_{b-1} \\ m_0 m_1' \ldots m_{b-1} \\ m_0' m_1' \ldots m_{b-1} \\ \vdots \\ m_0' m_1' \ldots m_{b-1}' \end{array}$$

that all hash to 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.

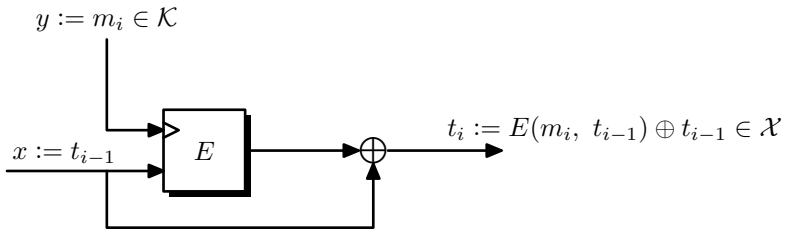


Figure 8.6: The Davies-Meyer compression function

- Compressions 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 and let g and h be integers in the range $[1, p-1]$. Consider the following simple compression function that takes two integers in $[1, p-1]$ and outputs an integer in $[1, p-1]$:

$$h(x, y) = g^x h^y \bmod p.$$

We will show in Exercise **??** 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 not often used in practice.

8.5.2 Davies-Meyer compression functions

In Chapter 4 we spent the effort to build secure block ciphers like AES. It is natural to ask whether we can leverage these constructions to build fast compression functions. The Davies-Meyer method enables us to do just that, but security can only be shown in the ideal cipher model.

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

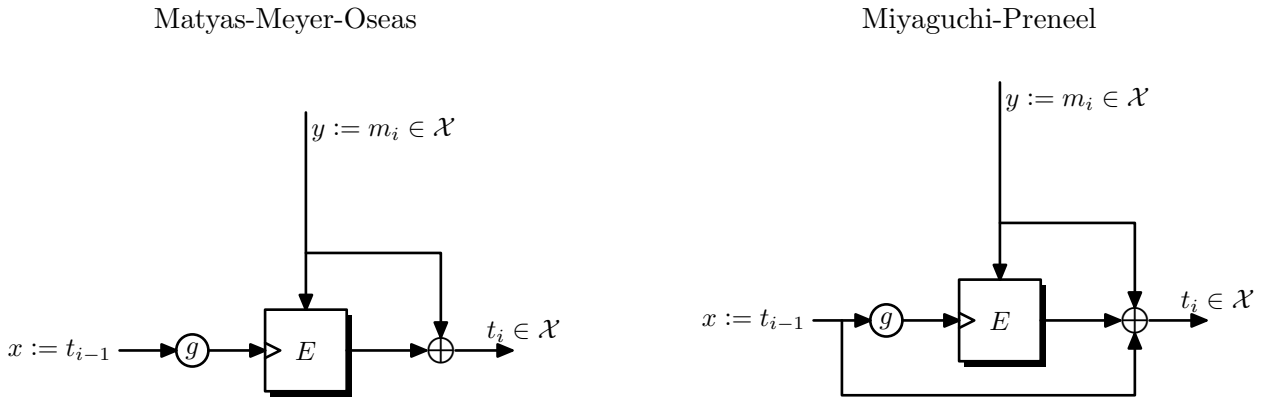


Figure 8.7: Other block cipher compression functions

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, one could use

Matyas-Meyer-Oseas:	$h_1(x, y) := E(x,\ y) \oplus y$
Miyaguchi-Preneel:	$h_2(x, y) := E(x,\ y) \oplus y \oplus x$
Or even:	$h_3(x, y) := E(x \oplus y,\ y) \oplus y$

or many other such variants. Preneel et al. [57] 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 8.14.

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 8.10).

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": 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 we will assume $\mathcal{A}$ is reasonable and makes at most q' ideal cipher queries from now on.

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 $a_i := \Pi^{-1}_{k_i}(b_i)$. 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

$$\mathrm{CR}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, h_{\mathrm{DM}}] \leq \Pr[Z].$$

Our goal is to show

$$\Pr[Z] \leq \frac{q'(q'-1)}{2^n}, \tag{8.3}$$

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

$$\Pr[Z] \leq \sum_{j=1}^{q'} \sum_{i=1}^{j-1} \Pr[a_i \oplus b_i = a_j \oplus b_j] \leq \sum_{j=1}^{q'} \frac{j-1}{2^n - j + 1} \leq \sum_{j=1}^{q'} \frac{j-1}{2^n - q'} = \frac{q'(q'-1)}{2(2^n - q')}. \tag{8.4}$$

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, (8.3) holds for all q'. □

8.6 Case study: SHA-256

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 **SHA-0**, 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 **SHA-1**. NIST gave no explanation for this change, but it was later found that this extra instruction is crucial for collision resistance. SHA-1 became the de-facto standard for collision resistant hashing and is very widely deployed.

The birthday attack can find collisions for SHA-1 using an expected 2^{80} evaluations of the function. In 2002 NIST added [FIPS 180-2] two new hash functions to the SHA family: **SHA-256** and **SHA-512**. They output larger digests (256 and 512-bit digests respectively) and therefore provide better protection against the birthday attack. NIST also approved SHA-224 and SHA-384

Name	year	digest size	message block size	Speed[2] MB/sec	best known attack time
SHA-0	1993	160	512		2^{39}
SHA-1	1995	160	512	153	2^{63}
SHA-224	2004	224	512		
SHA-256	2002	256	512	111	
SHA-384	2002	384	1024		
SHA-512	2002	512	1024	99	
MD4	1990	128	512		2^{1}
MD5	1992	128	512	255	2^{30}
Whirpool	2000	512	512	57	

Table 8.1: Merkle-Damgård collision resistant hash functions

which are obtained from SHA-256 and SHA-512 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 a variety of hash functions. In particular, Wang, Yao, and Yao [70] presented a collision finder for SHA-1 that uses 2^{63} evaluations of the function — far less than the birthday attack. As a result SHA-1 is no longer considered collision resistant. The current recommended practice is to use SHA-256 which we describe here.

The SHA-256 function. SHA-256 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 SHA-256 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 SHA-256 with the minor difference that PB encodes the number of *bits* in the message. Hence, SHA-256 can hash messages that are at most $2^{64} - 1$ bits long. The Merkle-Damgård Initial Value (IV) in SHA-256 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 SHA-256 can be truncated to obtain shorter digests at the cost of reduced security. This is, in fact, how the SHA-224 hash function works — it is identical to SHA-256 with two exceptions: (1) SHA-224 uses a different initialization vector IV, and (2) SHA-224 truncates the output of SHA-256 to its left most 224 bits.

Next, we describe the SHA-256 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, SHA-256 uses addition modulo 2^{32}. That is, let

$$x_0, x_1, \ldots, x_7 \in \{0,1\}^{32} \qquad \text{and} \qquad y_0, y_1, \ldots, y_7 \in \{0,1\}^{32}$$

[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.

and set

$$x := x_0 \,\|\, \cdots \,\|\, x_7 \;\; \in \{0,1\}^{256} \qquad \text{and} \qquad y := y_0 \,\|\, \cdots \,\|\, y_7 \;\; \in \{0,1\}^{256}.$$

Define: $x \boxplus y := (x_0 + y_0) \,\|\, \cdots \,\|\, (x_7 + y_7) \;\; \in \{0,1\}^{256}$ where all additions are modulo 2^{32}. Then the SHA-256 compression function h is defined as:

$$h(t, m) \;\; := \;\; E_{\text{SHA256}}(m, t) \;\boxplus\; t \;\; \in \{0,1\}^{256}.$$

Our ideal cipher analysis of Davies-Meyer (Theorem 8.4) applies equally well to this modified function.

The SHA-256 block cipher. To complete the description of SHA-256 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.

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:

$$k := k_0 \,\|\, k_1 \,\|\, \cdots \,\|\, k_{15} \;\; \in \{0,1\}^{512}$$
$$t := t_0 \,\|\, t_1 \,\|\, \cdots \,\|\, t_7 \;\; \in \{0,1\}^{256}$$

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 SHA-256 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 SHA-1 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 SHA-256 is already implemented, thus reducing the overall size of the crypto code.

8.6.1 Other Merkle-Damgård hash functions

MD4 and MD5. Two cryptographic hash functions designed by Rivest in 1990–1 [58, 59]. 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 should no longer be used.

Whirpool. Whirlpool was designed by Barreto and Rijmen in 2000 and was adopted as an ISO/IEC standard in 2004. Whirpool is a Merkle-Damgård hash function. Its compression function 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.

For x, y, z in $\{0,1\}^{32}$ define:

$$\begin{array}{rcll}
\mathrm{SHR}^n(x) & := & (x >> n) & \text{(Shift Right)} \\
\mathrm{ROTR}^n(x) & := & (x >> n) \vee (x << 32 - n) & \text{(Rotate Right)} \\
\mathrm{Ch}(x,y,z) & := & (x \wedge y) \oplus (\neg x \wedge z) & \\
\mathrm{Maj}(x,y,z) & := & (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & \\
\Sigma_0(x) & := & \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x) & \\
\Sigma_1(x) & := & \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x) & \\
\sigma_0(x) & := & \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) & \\
\sigma_1(x) & := & \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x) &
\end{array}$$

Table 8.2: Functions used in the SHA-256 block cipher

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 $K_0, K_1, \ldots, K_{63} \in \{0,1\}^{32}$

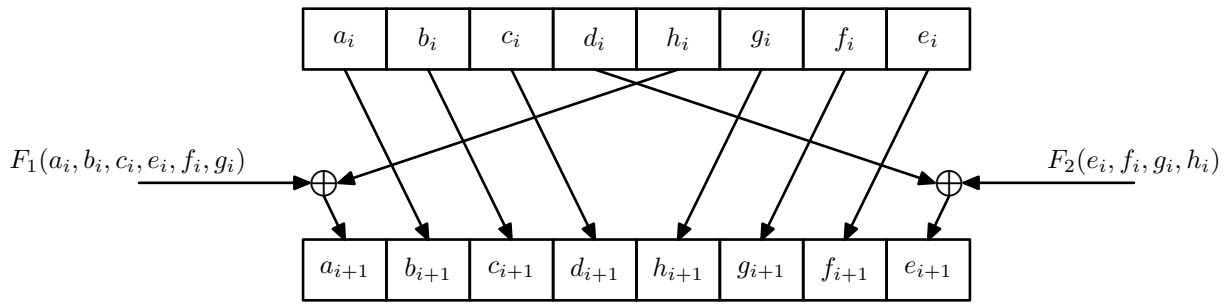
Key setup: Construct 64 round keys $W_0, \ldots, W_{63} \in \{0,1\}^{32}$:

$$\begin{cases} \text{for } i = 0, 1, \ldots, 15 & \text{set} \quad W_i \leftarrow k_i, \\ \text{for } i = 16, 17, \ldots, 63 & \text{set} \quad W_i \leftarrow \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} \end{cases}$$

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 SHA-256 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 SHA-256 block cipher

Others. Many other Merkle-Damgård hash functions were proposed in the literature. Some examples include Tiger/192 [10] and RIPEMD-160 to name a few.

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 SHA1 and SHA-256 are very 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 SHA-256 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, SHA-256 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 SHA-256, 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 8.17).

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 8.18).

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 8.16.

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 psuedorandomness 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 $k_1 \parallel M \parallel \text{PB}_\text{i}$, 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 $k_2 \parallel t \parallel \text{PB}_\text{o}$. We shall assume that n is significantly smaller than ℓ, so that $t \parallel \text{PB}_\text{o}$ 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:

$$h_{\text{bot}}(k, m) := h(k, m) \qquad \text{and} \qquad h_{\text{top}}(k, m) := h(m, k). \tag{8.5}$$

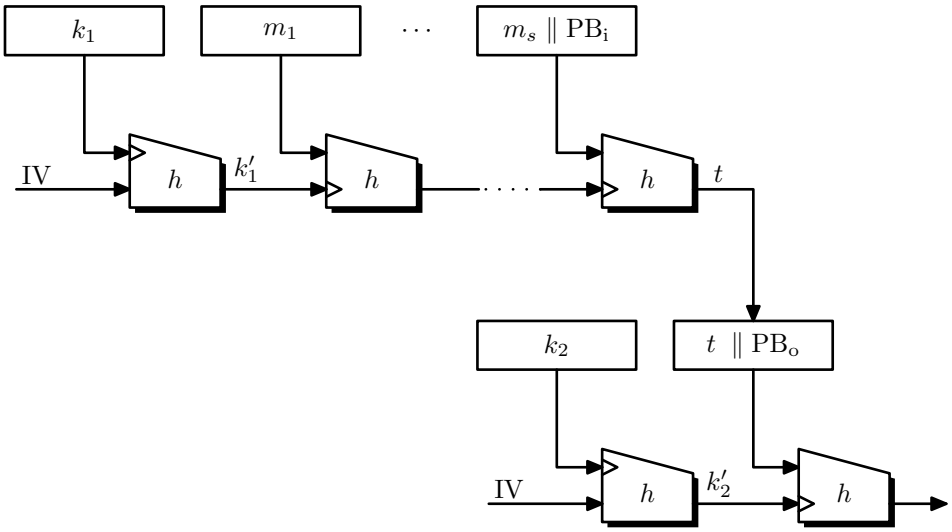


Figure 8.9: The two-key nest

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}.

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 $k_1' = h_{\text{top}}(k_1, \text{IV})$ and $k_2' = h_{\text{top}}(k_2, \text{IV})$. 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 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.

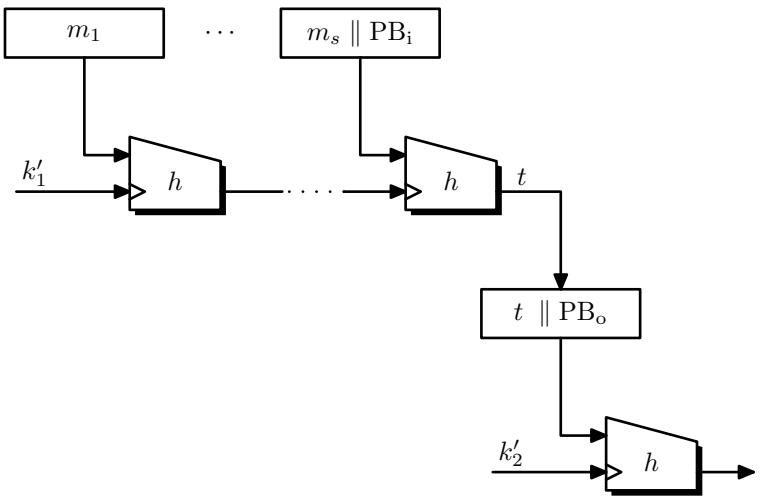


Figure 8.10: A bit-wise version of NMAC

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 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:

$$\begin{array}{rcl} \mathsf{ipad} & = & \text{the byte 0x36 repeated } B \text{ times} \\ \mathsf{opad} & = & \text{the byte 0x5C repeated } B \text{ times} \end{array}$$

HMAC implemented using a hash function H is denoted HMAC-H. The most common HMACs used in practice are HMAC-SHA1 and HMAC-SHA-256. 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 [5]). 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 (8.5) 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 $\{\Pi_k\}_{k \in \mathcal{K}}$. 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 (k, a) to which the challenger responds with $b := \Pi_k(a)$; a Π^{-1}-query is a pair (k, b) to which is the challenger responds with $a := \Pi_k^{-1}(b)$. 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

$$\mathrm{PRF}^{\mathrm{ic}}\mathsf{adv}[\mathcal{A}, h_{\mathrm{bot}}] = |p_0 - p_1|. \tag{8.6}$$

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 idea cipher model. We can write the logic of the challenger as follows:

Initialize:
    for each $k \in \mathcal{K}$, set $\Pi_k \xleftarrow{\mathrm{R}} \mathrm{Perms}[\mathcal{X}]$
    $k \xleftarrow{\mathrm{R}} \mathcal{X}$

standard h_{bot}-query m:
1. $c \leftarrow \Pi_m(k)$
2. $v \leftarrow c \oplus k$
3. 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 Π-query k, a:
1. $b \leftarrow \Pi_k(a)$
2. return b

ideal cipher Π^{-1}-query k, b:
1. $a \leftarrow \Pi_k^{-1}(b)$
2. return 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

$$\Pr[W_0] = p_0. \tag{8.7}$$

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 (changed from Game 0 are highlighted):

Initialize:
    for each $k \in \mathcal{K}$, set $\Pi_{\mathrm{std},k} \xleftarrow{\mathrm{R}} \mathrm{Perms}[\mathcal{X}]$ and $\Pi_{\mathrm{ic},k} \xleftarrow{\mathrm{R}} \mathrm{Perms}[\mathcal{X}]$
    $k \xleftarrow{\mathrm{R}} \mathcal{X}$

standard h_{bot}-query m:
1. $c \leftarrow \Pi_{\mathrm{std},m}(k)$ // add k to sampled domain of $\Pi_{\mathrm{std},m}$, add c to sampled range of $\Pi_{\mathrm{std},m}$
2. $v \leftarrow c \oplus k$
3. 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 Π-query $\mathcal{k}, \mathcal{a}$:

1. $\mathcal{b} \leftarrow \Pi_{\mathrm{ic},\mathcal{k}}(\mathcal{a})$ // add $\mathcal{a}$ to sampled domain of $\Pi_{\mathrm{ic},\mathcal{k}}$, add $\mathcal{b}$ to sampled range of $\Pi_{\mathrm{ic},\mathcal{k}}$
2. return $\mathcal{b}$

ideal cipher Π^{-1}-query $\mathcal{k}, \mathcal{b}$:

1. $\mathcal{a} \leftarrow \Pi^{-1}_{\mathrm{ic},\mathcal{k}}(\mathcal{b})$ // add $\mathcal{a}$ to sampled domain of $\Pi_{\mathrm{ic},\mathcal{k}}$, add $\mathcal{b}$ to sampled range of $\Pi_{\mathrm{ic},\mathcal{k}}$
2. return $\mathcal{a}$

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 2. 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

$$\Pr[W_1] = p_1. \tag{8.8}$$

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 $\Pi_{\mathrm{ic},\mathcal{k}}$'s, or the sampled range of one of the $\Pi_{\mathrm{std},m}$'s overlaps with the sampled range of one of the $\Pi_{\mathrm{ic},\mathcal{k}}$'s. The Domain Separation Lemma tells us that

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z]. \tag{8.9}$$

If Z occurs, then for some input/output triple $(\mathcal{k}, \mathcal{a}, \mathcal{b})$ corresponding to an ideal cipher query, $\mathcal{k} = m$ was the input to a standard query with output v, and either

(i) $\mathcal{a} = k$, or

(ii) $\mathcal{b} = v \oplus k$.

For any fixed triple $(\mathcal{k}, \mathcal{a}, \mathcal{b})$, by the independence of k, conditions (i) and (ii) each hold with probability $1/|\mathcal{X}|$, and so by the union bound

$$\Pr[Z] \leq \frac{2Q_{\mathrm{ic}}}{|\mathcal{X}|}. \tag{8.10}$$

The theorem now follows from (8.6)–(8.10). □

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 $F_{\text{pre}}(k, M) := H(k \,\|\, M)$, does not work when H 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 8.9). 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 specifying a permutation $\pi : \{0,1\}^n \to \{0,1\}^n$, we need to specify two positive integers numbers 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:

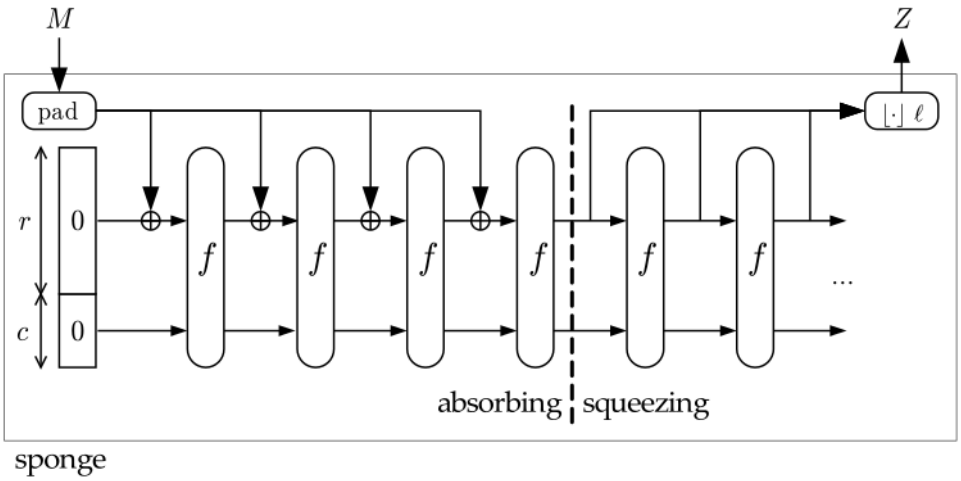


Figure 8.11: The sponge construction

Input: $M \in \{0,1\}^{\leq L}$ and $\ell > 0$
Output: a tag $h \in \{0,1\}^v$

// Absorbing stage
Pad M and break into r-bit blocks $m_1, \ldots, m_s$
$h \leftarrow 0^n$
for $i \leftarrow 1$ to s do
    $m'_i \leftarrow m_i \,\|\, 0^c \;\in \{0,1\}^n$
    $h \leftarrow \pi(h \oplus m'_i)$

// Squeezing stage
$z \leftarrow h[0\,..\,r-1]$
for $i \leftarrow 1$ to $\lceil v/r \rceil$ do
    $h \leftarrow \pi(h)$
    $z \leftarrow z \,\|\, (h[0\,..\,r-1])$
output $z[0\,..\,v-1]$

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 8.20).

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 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 notion of 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 $i = 1, \ldots, s$.

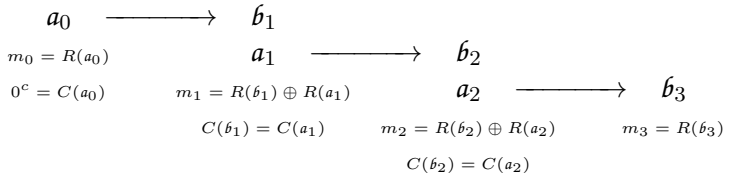
For such a path p, the **message** of p is defined as $(m_0, \ldots, m_{s-1})$, where

$$m_0 := R(a_0) \quad \text{and} \quad m_i := R(b_i) \oplus R(a_i) \ \text{ for } \ i = 1, \ldots, s-1.$$

and the **result** of p is defined to be $m_s := R(\mathfrak{b}_s)$. 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

$$m_0|\mathfrak{a}_0 \longrightarrow \mathfrak{b}_1|m_1|\mathfrak{a}_1 \longrightarrow \cdots \longrightarrow \mathfrak{b}_{s-2}|m_{s-2}|\mathfrak{a}_{s-2} \longrightarrow \mathfrak{b}_{s-1}|m_{s-1}|\mathfrak{a}_{s-1} \longrightarrow \mathfrak{b}_s|m_s. \tag{8.11}$$

The following diagram illustrates a C-path of length 3.



The path has message (m_0, m_1, m_2) and result m_3. Using the notation in (8.11), we write this path as

$$m_0|\mathfrak{a}_0 \longrightarrow \mathfrak{b}_1|m_1|\mathfrak{a}_1 \longrightarrow \mathfrak{b}_2|m_2|\mathfrak{a}_2 \longrightarrow \mathfrak{b}_3|m_3.$$

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 $v \le r$). 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 $\mathfrak{a}_{s-1} \to \mathfrak{b}_s$ and $\mathfrak{a}'_{t-1} \to \mathfrak{b}'_t$. Let us call such a pair of paths *problematic* if

(i) $\mathfrak{a}_{s-1} = \mathfrak{a}'_{t-1}$, 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

$$\Pr[W] \le \Pr[Z] + \Pr[W \text{ and not } Z]. \tag{8.12}$$

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 $\mathfrak{a}$ and $\mathfrak{a}'$, yielding outputs $\mathfrak{b}$ and $\mathfrak{b}'$ such that $V(\mathfrak{b}) = V(\mathfrak{b}')$. 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 $\mathfrak{a}$, yielding $\mathfrak{b} = \Pi(\mathfrak{a})$, and the j-th query is also a Π-query on some other value $\mathfrak{a}' \ne \mathfrak{a}$, yielding $\mathfrak{b}' = \Pi(\mathfrak{a}')$ such that $V(\mathfrak{b}) = V(\mathfrak{b}')$. 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 $\mathfrak{a}$, $\mathfrak{b}$, and $\mathfrak{a}'$ are all fixed, but the value $\mathfrak{b}'$ is uniformly distributed over a set of size at least $2^n - j + 1$. To get $V(\mathfrak{b}) = V(\mathfrak{b}')$, the value of $\mathfrak{b}'$ must be equal to one of the 2^{n-v} strings whose first v bits agree with that of $\mathfrak{b}$, and so we have

$$\Pr[X_{ij}] \le \frac{2^{n-v}}{2^n - j + 1}.$$

A simple calculation like that done in (8.4) in the proof of Theorem 8.4 yields

$$\Pr[W \text{ and not } Z] \leq \frac{q(q-1)}{2^v}. \tag{8.13}$$

Second, we bound $\Pr[Z]$, the probability that $\mathcal{A}$ finds a pair of problematic paths. The technical heart of the 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 b such that $C(\Pi^{-1}(b)) = 0^c$, or (ii) a Π query on some value a such that $C(\Pi(a)) = 0^c$,

and (E2) means $\mathcal{A}$ made pair of queries of the form:

(i) a Π-query on some value a and a Π^{-1} query on some value b, such that $C(\Pi(a)) = C(\Pi^{-1}(b))$, or (ii) Π-queries on two distinct values a and a' such that $C(\Pi(a)) = C(\Pi(a'))$.

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 (8.11). 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 b_1, or
- $s > 1$ and (E2) occurs with the Π^{-1} query on b_s and the Π-query on 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 (8.11) and write p' as

$$m'_0|a'_0 \longrightarrow b'_1|m'_1|a'_1 \longrightarrow \cdots \longrightarrow b'_{t-2}|m'_{t-2}|a'_{t-2} \longrightarrow b'_{t-1}|m'_{t-1}|a'_{t-1} \longrightarrow b'_t|m'_t.$$

We are assuming that $(m_0, \ldots, m_{s-1}) \neq (m'_0, \ldots, m'_{t-1})$ but $a_{s-1} = a'_{t-1}$, 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 $m_0|a_0 \rightarrow b_1|m_1$ and $m'_0|a'_0 \rightarrow b'_1|m'_1$, and we cannot have both $m_0 \neq m'_0$ and $a_0 = a'_0$.
2. $s = 1$ and $t \geq 2$. In this case, we have $a_0 = b'_{t-1}$, and so (E1) occurs on the Π-query on a'_{t-2}.

3. $s \geq 2$ and $t \geq 2$. Consider the penultimate edges, which are forward edges:

$$a_{s-2} \to b_{s-1}|m_{s-1}|a_{s-1}$$

and

$$a'_{t-2} \to b'_{t-1}|m'_{t-1}|a'_{t-1}.$$

We are assuming $a_{s-1} = a'_{t-1}$. Therefore, the C-parts of b_{s-1} and 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 $b_{s-1} = b'_{t-1}$, and therefore $a_{s-2} = a'_{t-2}$, while the truncated messages $(m_0, \ldots, m_{s-2})$ and $(m'_1, \ldots, m'_{t-2})$ 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 b_{s-1} and b'_{t-1} are the same, but their R-parts differ, and therefore, $a_{s-1} \neq a'_{t-2}$. Thus, (E2) occurs on the Π-queries on a_{s-2} and 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 (8.12), and earlier in the proof of Theorem 8.4. The only difference from (8.12) 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:

$$\Pr[Z] \leq \frac{q(q+1)}{2^c}. \tag{8.14}$$

The theorem now follows from (8.12)–(8.14). □

8.8.2 Case study: SHA3, SHAKE256, and SHAKE512

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 Key derivation and the random oracle model

Although hash functions like SHA-256 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 SHA-256 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 SHA-256 (or, as we will ultimately recommend, some function built out of SHA-256), 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.

8.9.1 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 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 as 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 8.2 (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 $\mathsf{Guessadv}[\mathcal{A}, P, I]$. □

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 $\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P, I, H]$ 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 $\mathsf{Dist}\mathsf{adv}[\mathcal{A}, P, I, H]$ is not too much larger than $\mathsf{Guess}\mathsf{adv}[\mathcal{A}, P, I]$. In fact, in certain situations it *is* possible prove such a theorem. We will discuss this result later, in Section 8.9.4 — 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 SHA-256 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, and 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 encrypting 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 SHA-256.

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.

8.9.2 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 8.9.3 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 $\mathsf{Xadv}[\mathcal{A}, \mathcal{S}]$, 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 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 8.3 (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:

- $\mathcal{O} \xleftarrow{\text{R}} \text{Funs}[\mathcal{M}, \mathcal{T}]$.
- The challenger selects $f \in \text{Funs}[\mathcal{X}, \mathcal{Y}]$ as follows:

  if $b = 0$: $k \xleftarrow{\text{R}} \mathcal{K}$, $f \leftarrow F^{\mathcal{O}}(k, \cdot)$;
  if $b = 1$: $f \xleftarrow{\text{R}} \text{Funs}[\mathcal{X}, \mathcal{Y}]$.

- The adversary submits a sequence of queries to the challenger.
  - 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 8.3. *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 8.9.3.

Theorem 8.7. *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 8.3, that makes at most Q_{H} oracle queries, then*
>
> $$\text{PRF}^{\text{ro}}\mathsf{adv}[\mathcal{A}, F_{\text{pre}}] \leq Q_{\text{H}}/|\mathcal{K}|$$

Note that Theorem 8.7 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_{H} queries the adversary succeeds in issuing a relevant query is at most $Q_{\text{H}}/|\mathcal{K}|$. □

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 8.3, 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* mapping from $\mathcal{M}$ 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 $Map : \mathcal{M} \to \mathcal{T}$
    $k \xleftarrow{\text{R}} \mathcal{K}$
Upon receiving an F_{pre}-query on $x \in \{0,1\}^{\leq L}$ do:
    $t \xleftarrow{\text{R}} \mathcal{T}$
(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:
    $t \xleftarrow{\text{R}} \mathcal{T}$
    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 8.3. 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 8.3, 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

$$\Pr[Z] \leq Q_{\text{H}}/|\mathcal{K}|.$$

That completes the proof. □

Key derivation in the random oracle model. Let us now return to the key derivation problem introduced in Section 8.9.1. 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 8.2 to allow the adversary to output a list of guesses $\hat{s}_1, \ldots, \hat{s}_Q$, where and 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 8.8. *If H is modeled as a random oracle, then for every distinguishing adversary $\mathcal{A}$ that makes at most Q_{H} random oracle queries, there exists a list guessing adversary $\mathcal{B}$, which is an*

elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{Dist}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, P, I, H] \leq \mathrm{ListGuess}\mathsf{adv}[\mathcal{B}, P, I]$$

and $\mathcal{B}$ outputs a list of size at most Q_{H}. In particular, there exists a guessing adversary $\mathcal{B}'$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{Dist}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}, P, I, H] \leq Q_{\mathrm{H}} \cdot \mathrm{Guess}\mathsf{adv}[\mathcal{B}', P, I].$$

Proof. The proof is almost identical to that of Theorem 8.7. 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
  $t \xleftarrow{\mathrm{R}} \mathcal{T}$
$(*)$ $Map[s] \leftarrow t$
  send $(I(s), t)$ to $\mathcal{A}$
Upon receiving an $\mathcal{O}$-query $\hat{s} \in \mathcal{S}$ do:
  $\hat{t} \xleftarrow{\mathrm{R}} \mathcal{T}$
  if $\hat{s} \in \mathrm{Domain}(Map)$ 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 $\mathrm{Domain}(Map)$ — 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

$$\mathrm{Dist}^{\mathrm{ro}}\mathsf{adv}[\mathcal{A}] = |\Pr[W_0] - \Pr[W_1]| \leq \mathrm{ListGuess}\mathsf{adv}[\mathcal{B}]. \quad \square$$

8.9.3 Random oracles: safe modes of operation

We have already seen that $F_{\mathrm{pre}}(k, x) := H(k \parallel x)$ 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 safe HMAC with a zero key:

$$\mathrm{HMAC}_0(m) := \mathrm{HMAC}(0^\ell, m) = H(\mathsf{opad} \parallel H(\mathsf{ipad} \parallel m)).$$

While this construction foils the obvious extension attacks, why should we have any confidence at all 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 permutation. This is undoubtedly a rather quixotic task — using heuristics to justify heuristics — but we shall sketch the basic ideas.

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 $\langle \mathbf{C}, \mathcal{A} \rangle$, and 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 model 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* $\mathcal{B}$*, 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 $\mathcal{A}$ breaks the scheme with $F[\rho]$, then the adversary $\mathcal{B}$ above will break the scheme with $\mathcal{O}$.

The upshot. 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. 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, where c is the capacity is super-poly.) 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.

8.9.4 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 $\mathbf{s}$ is a random variable taking values in a set $\mathcal{S}$, we define the **guessing probability of $\mathbf{s}$** to be $\max_{x \in \mathcal{S}} \Pr[\mathbf{s} = x]$.

Theorem 8.9 (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 $\mathbf{k}, \mathbf{s}_1, \ldots, \mathbf{s}_m$ be mutually independent random variables, where $\mathbf{k}$ is uniformly distributed over $\mathcal{K}$, and each $\mathbf{s}_i$ has guessing probability at most γ. Let δ be the statistical difference between*

$$(\mathbf{k}, H(\mathbf{k}, \mathbf{s}_1), \ldots, H(\mathbf{k}, \mathbf{s}_m))$$

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 7.23 — 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 8.9.1 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. Using cryptographic hash functions, while only heuristically secure for key derivation, are also more forgiving.

8.9.5 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 SHA-256.

Th 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

$$t \leftarrow \text{HMAC}(salt, s).$$

Using the intermediate key t, along with *info*, the expand (or sub-key derivation) stage computes L bytes of output data, as follows:

$q \leftarrow \lceil L/HashLen \rceil$ // *HashLen* is the output length (in bytes) of H
initialize z_0 to the empty string
for $i \leftarrow 1$ to q do:
    $z_i \leftarrow \text{HMAC}(t, z_{i-1} \parallel info \parallel Octet(i))$ // $Octet(i)$ 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 8.9.3. As discussed there, HMAC_0 can heuristically be viewed as a random oracle, and so we can use the analysis in Section 8.9.2 to show that this is a secure key derivation procedure in the random oracle model. This, if s is hard to guess, then t is indistinguishable from random.

Users of HKDF have the option of providing non-zero salt. The salt plays a role akin to the random hash key used in the Leftover Hash Lemma (see Section 8.9.4); 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 8.9.1. 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.

8.10 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.

8.10.1 Second preimage resistance

We start by defining two classic security properties for non-keyed hash functions. 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 $H(m) = H(m')$.

Definition 8.4. *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 $\mathcal{A}$.*

Similarly, 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* $\mathsf{SPRadv}[\mathcal{A}, H]$ *is negligible for every efficient adversary $\mathcal{A}$.*

We mention some trivial relations between these notions when $\mathcal{M}$ is at least twice the size of $\mathcal{T}$. Under this condition we have the following implications:

$$H \text{ is collision resistant} \quad \Rightarrow \quad H \text{ is 2nd-preimage resistant} \quad \Rightarrow \quad H \text{ is one-way}$$

as shown in Exercise 8.22. The converse is not true. A hash function can be 2nd-preimage resistant, but not collision resistant. For example, SHA-1 is believed to be 2nd-preimage resistant even though SHA-1 is not collision resistant. Similarly, a hash function can be one-way, but not be 2nd-preimage resistant. For example, the function $h(x) := x^2 \bmod N$ for a large odd composite N is believed to be one-way. In other words, it is believed that given $x^2 \bmod N$ it is difficult to find x (as long as the factorization of N is unknown). However, this function H is trivially not 2nd-preimage resistant: given $x \in \{1, \ldots, N\}$ as input, the value $-x$ is a second preimage since $x^2 \bmod N = (-x)^2 \bmod N$.

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.

8.10.2 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. 8.12.

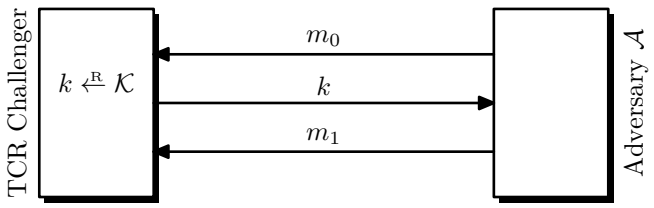


Figure 8.12: TCR Attack Game

Attack Game 8.4 (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:

- $\mathcal{A}$ 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 $\mathrm{TCR}\mathsf{adv}[\mathcal{A}, H]$, as the probability that $\mathcal{A}$ wins the game. □

Definition 8.5. *We say that a keyed hash function H over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ is target collision resistant if $\mathrm{TCR}\mathsf{adv}[\mathcal{A}, H]$ 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.

8.10.3 TCR from 2nd-preimage resistance

We show how to build a keyed TCR hash function from a keyless 2nd-preimage resistant function such as SHA-1. 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:

$$H_{\mathrm{tcr}}(k, m) = H(k \oplus m) \tag{8.15}$$

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 8.10. *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 8.4, there exists a 2nd-preimage finder $\mathcal{B}$, which is an elementary wrapper around $\mathcal{A}$, such that

$$\mathrm{TCR}\mathsf{adv}[\mathcal{A}, H_{tcr}] \leq \mathrm{SPR}\mathsf{adv}[\mathcal{B}, H].$$

Proof. The proof is a simple direct reduction. Adversary $\mathcal{B}$ emulates the challenger in Attack Game 8.4 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 $\text{SPRadv}[\mathcal{B}, H] = \text{TCRadv}[\mathcal{A}, H_{\text{tcr}}]$. 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 8.4. It follows that $\mathcal{B}$ perfectly emulates the challenger in Attack Game 8.4 and consequently $\Pr[W] = \text{TCRadv}[\mathcal{A}, H_{\text{tcr}}]$.

By definition of H_{tcr}, we also have the following:

$$\begin{aligned} H_{\text{tcr}}(k, m_0) &= H((m \oplus m_0) \oplus m_0) = H(m) \\ H_{\text{tcr}}(k, m_1) &= H(m_1 \oplus k) = H(m') \end{aligned} \tag{8.16}$$

Now, suppose event W happens. Then $H_{\text{tcr}}(k, m_0) = H_{\text{tcr}}(k, m_1)$ and therefore, by (8.16), 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:

$$\text{SPRadv}[\mathcal{B}, H] \geq \Pr[W] = \text{TCRadv}[\mathcal{A}, H_{\text{tcr}}]$$

as required. □

Target collision resistance for long inputs. The function H_{tcr} in (8.15) shows that a 2nd-preimage resistant function directly gives a TCR function. If we assume that the SHA-256 compression function h is 2nd-preimage resistant (a weaker assumption than assuming that h is collision resistant) then, by Theorem 8.10 we obtain a TCR hash for inputs of length $512 + 265 = 768$ bits. The length of the required key is also 768 bits.

We will often need TCR functions for much longer inputs. Using the SHA-256 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 (8.15).

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.

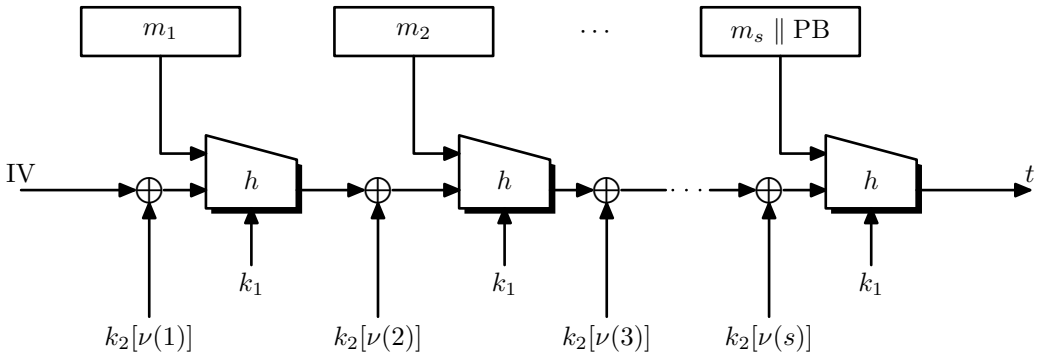


Figure 8.13: Extending the domain of a TCR hash

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. 8.13):

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 ℓ-bit blocks so that
$$M = m_1 \parallel m_2 \parallel \cdots \parallel m_s \quad \text{where} \quad m_1, \ldots, m_s \in \{0,1\}^\ell$$
$t_0 \leftarrow \text{IV}$
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. We refer to [63, 48] for the proof of this theorem.

Theorem 8.11. *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 8.4). Then there exists a TCR adversary $\mathcal{B}$ (whose running times are about the same as that of $\mathcal{A}$) such that

$$\text{TCRadv}[\mathcal{A}, H] \leq L \cdot \text{TCRadv}[\mathcal{B}, h].$$

As in Merkle-Damgård this construction is inherently sequential. A tree-based construction similar to Exercise 8.8 gives a TCR hash using logarithmic size keys that is more suitable for a parallel machine. We refer to [6] for the details.

8.10.4 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.

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 8.4 — 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 8.24) we can reduce the additional read-only memory used to a small constant, but still non-zero.

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:

$$
\begin{array}{ll}
S'(k,\ m) := & V'(k,\ m,\ (t,r)\ ) := \\
\quad r \xleftarrow{R} \mathcal{K}_H & \quad h \leftarrow H(r, m) \\
\quad h \leftarrow H(r, m) & \quad \text{Output } V(k,\ (r,h),\ t) \\
\quad t \leftarrow S(k,\ (r,h)\ ) & \\
\quad \text{Output } (t, r) &
\end{array}
\tag{8.17}
$$

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 (8.17) 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 8.12. *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 (8.17) 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*
>
> $$\mathrm{MACadv}[\mathcal{A}, \mathcal{I}'] \leq \mathrm{MACadv}[\mathcal{B}_{\mathcal{I}}, \mathcal{I}] + q \cdot \mathrm{TCRadv}[\mathcal{B}_H, H].$$

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

$$\mathrm{MACadv}[\mathcal{A}, \mathcal{I}'] \ = \ \Pr[X] \ \leq \ \Pr[X \wedge \neg Y] + \Pr[Y] \ = \ \Pr[Z] + \Pr[Y] \tag{8.18}$$

To prove the theorem we construct a TCR adversary $\mathcal{B}_H$ and a MAC adversary $\mathcal{B}_{\mathcal{I}}$ such that

$$\Pr[Y] \leq q \cdot \mathrm{TCRadv}[\mathcal{B}_H, H] \quad \text{and} \quad \Pr[Z] = \mathrm{MACadv}[\mathcal{B}_{\mathcal{I}}, \mathcal{I}].$$

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 ith 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,

$$\text{TCR}\mathsf{adv}[\mathcal{B}_H, H] = \Pr[Y \wedge (j = u)].$$

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,

$$\text{TCR}\mathsf{adv}[\mathcal{B}_H, H] = \Pr[Y \wedge (j = u)] = \Pr[Y] \cdot \Pr[j = u] \geq \Pr[Y]/q$$

as required. □

8.11 A fun application: commitment schemes

To be written.

8.12 Notes

Citations to the literature to be added.

8.13 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 $H''(x) := H'(x)[0\,..\,n-2]$ 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. 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.

8.4. 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.

8.5 (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.

8.6 (Finding multi-collisions). Continuing with Exercise 8.5, 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.

8.7 (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 $\mathrm{Funs}[\mathcal{M}, \mathcal{T}]$ 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 $j \le i$.

  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.

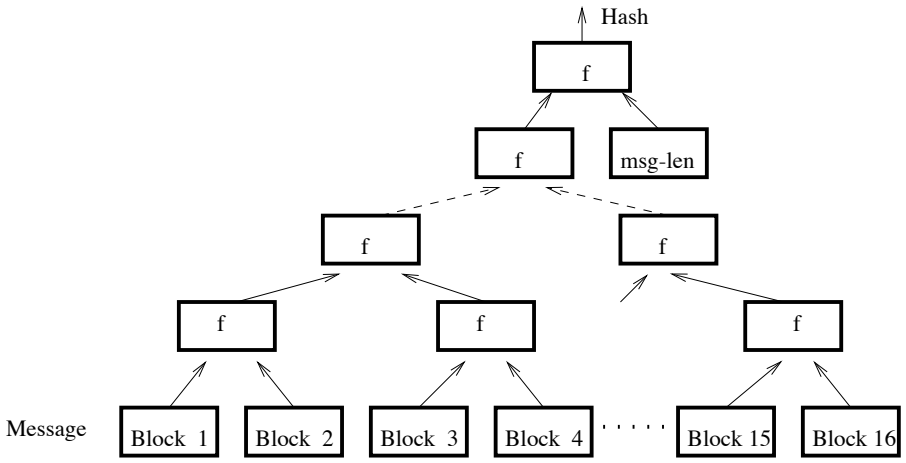


Figure 8.14: Tree-based Merkle-Damgård

8.8. 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. 8.14 is a parallelizable approach. Prove that the resulting hash function is collision resistant assuming f is collision resistant.

8.9. Prove that the h_1, h_2, and h_3 variants of Davies-Meyer defined on page 298 are collision resistant in the ideal cipher model.

8.10. Show that the h_4 and h_5 variants of Davies-Meyer defined on page 299 are not collision resistant.

8.11. 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 $\mathcal{K} = \mathcal{X} = \{0,1\}^n$. Suppose that

$$E(\bar{k}, \bar{x}) = \overline{E(k, x)}$$

for all keys $k \in \mathcal{K}$ and all $x \in \mathcal{X}$, where $\bar{y}$ denotes the bit-wise complement of y. The DES block cipher has precisely this property.

(a) Show that Davies-Meyer is not collision resistant when instantiated with algorithm E.

(b) Let (E', D') be the Even-Mansour block cipher built from a permutation $\pi : \mathcal{X} \to \mathcal{X}$. Show that Davies-Meyer is not collision resistant when instantiated with algorithm E'.

8.12 (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.

8.13 (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 8.4. 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.

8.14 (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 $h_{\mathrm{DM}}(t, m) = t$. 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.

8.15 (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 8.13. 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 $h(\mathrm{IV}, x)$ 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 second preimage resistance takes about 2^n time. This exercise shows that for Merkle-Damgård functions, breaking second preimage resistance can be done much faster, taking only about $2^{n/2}$ time.

8.16 (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 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 7.8; 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).

8.17. 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.

8.18. 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.

8.19 (Symmetric 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 **symmetric PRF**. Let F be a secure PRF defined over $(\mathcal{X}, \mathcal{X}, \mathcal{Y})$. We wish to build a new PRF $\hat{F}$ that is symmetric. Then $\hat{F}$ can be used as a building block for HMAC.

- 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 symmetric PRF. Hint: start from a secure PRF F' and the sabotage it to get the required F.
- Let $G : \{0,1\}^n \to \{0,1\}^{2n}$ be a secure PRG. Let $G_0(s)$ be the left n bits of $G(s)$ and $G_1(s)$ be the right n bits of $G(s)$. Define

$$\hat{F}(x, y) := F(\ G_0(x),\ G_0(y)\ ) \quad \oplus \quad F(\ G_1(y),\ G_1(x)\ ).$$

  Prove that $\hat{F}$ that is a symmetric PRF.

8.20 (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.

8.21 (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 8.9.3) and Theorem 8.7. However, you are to give a *direct proof* of this fact. Hint: use the same domain splitting strategy as outlined in Exercise 7.20.

8.22 (Relations among definitions). Let H be a hash function over $(\mathcal{M}, \mathcal{T})$ where $|\mathcal{M}| \geq 2|\mathcal{T}|$. We say that an element $m \in \mathcal{M}$ has a second preimage if there exists a different $m' \in \mathcal{M}$ such that $H(m) = H(m')$.

(a) Show that at least half the elements of $\mathcal{M}$ have a second preimage.

(b) Use part (a) to show that a 2nd-preimage hash must be one-way.

(c) Show that a collision resistant hash must be 2nd-preimage resistant.

8.23. Let H be a TCR hash defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$. Choose a random $r \in \mathcal{M}$. Prove that $f(x) := H(r, x) \,\|\, r$ is 2nd-preimage resistant, where r is treated as a system parameter.

8.24. The file integrity construction in Section 8.10.4 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 (1) to show how to reduce read-only overhead to constant size independent of $|F|$. The extra information stored on disk is still of size $O(\log |F|)$.

8.25 (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) Let H be a strong-SPR. Use H to construct a collision resistant hash function H' defined over $(\mathcal{X} \times \mathcal{Y},\ \mathcal{T})$.

(b) 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.

(c) Show that $H_{TCR}(k, (x, y)) := H((k \oplus x), y)$ is a TCR hash function assuming H is a strong-SPR hash function.

8.26. 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 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 $H(k, m) = H(k', m')$.

(a) Let H be a strong-SPR hash function over $(\mathcal{X} \times \mathcal{Y}, \mathcal{T})$, as defined in Exercise 8.25, where $\mathcal{X} := \{0,1\}^n$. Show that $H'(k,(x,y)) := H((k \oplus x), y)$ is an enhanced-TCR hash function.

(b) 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:

$$S'(k,m) := \{\ r \xleftarrow{R} \mathcal{K}_H,\ t \leftarrow S(k, H(r,m)),\ \text{output } (r,t)\}$$
$$V'\big(k,m,(r,t)\big) := \{\ \mathsf{accept} \text{ if } t = V(k, H(r,m))\}$$

8.27 (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.

Chapter 9

Authenticated Encryption

Our discussion of encryption in Chapters 2 to 8 leads up to this point. In this chapter we, construct systems that ensure both data secrecy (confidentiality) and data integrity, even against very aggressive attackers that can interact with the sender and receiver quite maliciously and arbitrarly. Such systems are said to provide **authenticated encryption** or are simply said to be AE-secure. This chapter concludes our discussion of symmetric encryption. It is the culmination of our symmetric encryption story.

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 comprises 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.

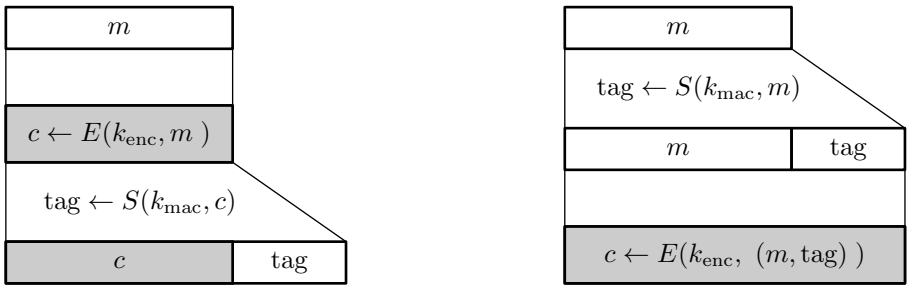
In this chapter we develop the notion of authenticated encryption and construct several AE systems. There are two general paradigms for construction AE systems. The first, called **generic composition**, is to combine a CPA-secure cipher with a secure MAC. There are many ways to combine these two primitives 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, $c \xleftarrow{R} E(k_{\text{enc}}, m)$, then MAC the ciphertext, tag $\xleftarrow{R} S(k_{\text{mac}}, c)$; 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 9.6).

MAC-then-encrypt MAC the message, tag $\xleftarrow{R} S(k_{\text{mac}}, m)$, then encrypt the message-tag pair, $c \xleftarrow{R} E\big(k_{\text{enc}},\ (m, t)\big)$; 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 lead to many attacks on widely deployed systems.

The second paradigm for building authenticated encryption is to build them directly from a block cipher or a PRF without first constructing either a standalone cipher or MAC. These are sometimes called **integrated schemes**. The OCB encryption mode is the primary example in this category (see Exercise 9.14). Other examples include IAPM, XCBC, CCFB, and others.

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 9.6). 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 propertes. 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 $k \xleftarrow{R} \mathcal{K}$.

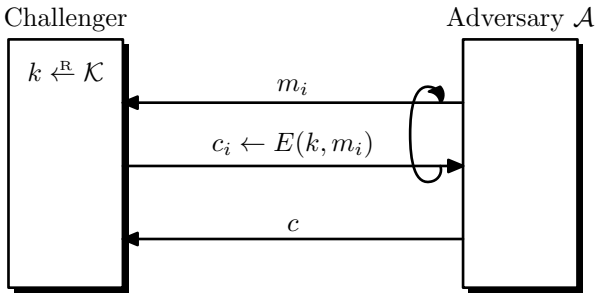


Figure 9.2: Ciphertext integrity game (Attack Game 9.1)

- $\mathcal{A}$ 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 $c_i \xleftarrow{R} E(k, m_i)$, 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 insideous 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.

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 provide 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* $\mathsf{CIadv}[\mathcal{A}, \mathcal{E}]$ *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 Chosen ciphertext attacks

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** or CCA for short. As we will see, an AE-secure cipher provides mesage secrecy and message integrity even against such a powerful attack.

To motivate CCA 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 Eve begins with `To:eve@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:eve@mail.com` are written to Eve's inbox.

Eve, the attacker in this story, wants to read the email that Alice sent to Bob. Unfortunately for Eve, 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. Eve 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 Eve can quite easily obtain the email contents. Here is how: Eve 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:eve@mail.com`, but is otherwise the same as the original message. Eve 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 Eve's inbox where Eve can easily read it.

To successfully carry out this attack, Eve must first solve the following problem: given an encryption c of some message $(u \,\|\, m)$ 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 $(v \,\|\, m)$, where v is some other prefix (in our case $v :=$ `To:eve@mail.com`).

Let us show that Eve 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. Eve 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). Eve 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 $\hat{c}$ Eve was able to deduce the decryption of c. As we just saw, a CPA-secure system can become completely insecure when an attacker can decrypt certain ciphertexts, even if she cannot directly decrypt a ciphertext that interests her. Put another way, the lack of ciphertext integrity can completely compromise secrecy — even if plaintext integrity is not a 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 Eve intercepts a ciphertext $c := E(k, m)$. She 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 consider a more general chosen ciphertext attack: the attacker intercepts a ciphertext c and wants to decrypt it. It can request the decryption of multiple ciphertexts $\hat{c}_1, \ldots, \hat{c}_Q$ of its choice, possibly derived from c. However, it cannot directly request the decryption of c. The attacker "wins" if using these decryption queries it can learn something about the decryption of c. We will show that if $\mathcal{E}$ is AE-secure then even these powerful chosen ciphertext queries do not help the attacker break semantic security. The fact that authenticated encryption foils the email attack above is a special case of this more general result.

9.2.2 Chosen ciphertext attacks: definition

In the previous section we saw an example where the adversary could obtain the decryption of some ciphertexts, but not others. We showed that this enabled the adversary to decrypt any ciphertext of his choice. Here, we take this idea to the extreme:

- Suppose the adversary intercepts a number of ciphertexts $L := \{c_1, c_2, \ldots\}$ all encrypted under the same key k. It wants to learn something about the decryption of these ciphertexts.
- We allow the adversary to request the decryption (under the key k) of any ciphertext $\hat{c}$ of his choice as long as $\hat{c}$ is not one of the challenge ciphertexts in L.

We say that the system is **chosen ciphertext secure** if the adversary cannot learn any information about the decryption of ciphertexts in L. Achieving this level of security would appear to be a daunting task, and yet we will show that any AE-secure system is chosen ciphertext secure. This provides yet another justification for insisting that only AE-secure systems be used for general-purpose applications: it prevents chosen ciphertext attacks that frequently come up in practice. We start with a definition of chosen ciphertext security using the following 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 $k \xleftarrow{R} \mathcal{K}$.
- The adversary then makes a series of queries to the challenger, each of which is either an **encryption query** or a **decryption query**. The adversary is allowed to make many encryption and decryption queries interleaved in any order.
  - upon receiving an encryption query $(m_0, m_1) \in \mathcal{M}^2$ from the adversary, the challenger computes $c \xleftarrow{R} E(k, m_b)$ and sends c to the adversary.
  - upon receiving a ciphertext $\hat{c} \in \mathcal{C}$, with $\hat{c} \notin \{c_1, \ldots, c_i\}$ where $c_1, \ldots, c_i$ are all the responses to encryption queries so far. The challenger computes $\hat{m} \leftarrow D(k, \hat{c})$, and sends $\hat{m}$ to the adversary.
- 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 and define $\mathcal{A}$'s **advantage** with respect to $\mathcal{E}$ as

$$\text{CCA}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := \Big|\Pr[W_0] - \Pr[W_1]\Big|. \quad \square$$

Definition 9.5 (CCA **security).** *A cipher $\mathcal{E}$ is called* ***semantically secure against a chosen ciphertext and plaintext 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). *A cipher $\mathcal{E}$ is **semantically secure against chosen ciphertext attack**, or simply, 1CCA-**secure**, if for all efficient adversaries $\mathcal{A}$ that make at most a single encryption query in Attack Game 9.2 the value* $\mathrm{CCAadv}[\mathcal{A}, \mathcal{E}]$ *is negligible.*

9.2.3 Authenticated encryption implies chosen ciphertext security

We now show that every AE-secure system is also CCA-secure. Similarly, every one-time AE-secure system is 1CCA-secure. Our concrete security bounds and proofs work using the bit-guessing versions of the CCA and CCA attack games (see Section 2.3.5).

Theorem 9.1. *Let $\mathcal{E} = (E, D)$ be an authenticated encryption system. Then $\mathcal{E}$ is* CCA*-secure.*

> *In particular, suppose $\mathcal{A}$ is a* CCA*-adversary for $\mathcal{E}$ that makes at most Q_{d} encryption queries and Q_{e} decryption queries. Then there exist a* CPA*-adversary $\mathcal{B}_1$ and a* CI*-adversary $\mathcal{B}_2$, where $\mathcal{B}_1$ and $\mathcal{B}_2$ are elementary wrappers around $\mathcal{A}$, such that*
>
> $$\mathrm{CCAadv}^*[\mathcal{A}, \mathcal{E}] \leq \mathrm{CPAadv}^*[\mathcal{B}_1, \mathcal{E}] + Q_{\mathrm{e}} \cdot \mathrm{CIadv}[\mathcal{B}_2, \mathcal{E}].$$
>
> *Moreover, $\mathcal{B}_1$ and $\mathcal{B}_2$ both make at most Q_{d} 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 9.11. 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 9.8 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. To make this proof idea rigorous it is convenient to work with the bit-guessing version of the CCA Attack Game 9.2 (as discussed in Section 2.3.5). The challenger in this game, called Game 0, works as follows:

> $b \xleftarrow{R} \{0, 1\}$ // $\mathcal{A}$ will try to guess b
> $k \xleftarrow{R} \mathcal{K}$, $L \leftarrow \emptyset$
>
> upon receiving an encryption query (m_0, m_1) from $\mathcal{A}$ do:
> Send $c \leftarrow E(k, m_b)$ to $\mathcal{A}$ and add c to the set L, that is, $L \leftarrow L \cup \{c\}$
>
> upon receiving a decryption query $\hat{c} \notin L$ from $\mathcal{A}$ do:
> (1) Send $D(k, \hat{c})$ 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

$$\mathrm{CCAadv}^*[\mathcal{A}, \mathcal{E}] = |\Pr[W_0] - 1/2|. \tag{9.1}$$

Game 1. We now modify line (1) in the challenger as follows:

upon receiving a decryption query $\hat{c} \notin L$ from $\mathcal{A}$ do:
(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 $\hat{c}$ such that $D(k, c) \neq \mathsf{reject}$. 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 $|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z]$.

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 $\mathcal{B}_2$ that wins the CI attack game with probability at least $\Pr[Z]/Q_{\mathrm{e}}$. Note that in Game 2, the decryption algorithm is not used at all. Adversary $\mathcal{B}_2$'s strategy is simply to guess a random number $\omega \in \{1, \ldots, Q_{\mathrm{e}}\}$, and then to play the role of challenger to $\mathcal{A}$:

- when $\mathcal{A}$ makes an encryption query, $\mathcal{B}_2$ forwards this to its own challenger, and returns the response to $\mathcal{A}$;
- when $\mathcal{A}$ makes a decryption query $\hat{c}$, $\mathcal{B}_2$ simply sends reject to $\mathcal{A}$, except that if this is the ωth such request, $\mathcal{B}_2$ outputs $\hat{c}$ and halts.

It is not hard to see that $\mathrm{CI}\mathsf{adv}[\mathcal{B}_2, \mathcal{E}] \geq \Pr[Z]/Q_{\mathrm{e}}$, and so

$$|\Pr[W_0] - \Pr[W_1]| \leq \Pr[Z] \leq Q_{\mathrm{e}} \cdot \mathrm{CI}\mathsf{adv}[\mathcal{B}_2, \mathcal{E}]. \tag{9.2}$$

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 $\mathcal{B}_1$ that plays the role of challenger to $\mathcal{A}$ as follows:

- when $\mathcal{A}$ makes an encryption query, $\mathcal{B}_1$ forwards this to its own challenger, and returns the response to $\mathcal{A}$;
- when $\mathcal{A}$ makes a decryption query $\hat{c}$, $\mathcal{B}_1$ simply sends reject to $\mathcal{A}$.

At the end of the game, $\mathcal{B}_1$ simply outputs the bit $\hat{b}$ that $\mathcal{A}$ outputs. Clearly,

$$|\Pr[W_1] - 1/2| = \mathrm{CPA}\mathsf{adv}^*[\mathcal{B}_1, \mathcal{E}] \tag{9.3}$$

Putting equations (9.1)–(9.3) together 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 or not 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 placeholder 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 placeholders 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 he 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 "placeholders" 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 5.10.

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 $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ and let $\mathcal{I} = (S, V)$ be a MAC defined over $(\mathcal{K}, \mathcal{C}, \mathcal{T})$. To keep the notation simple we assume that $\mathcal{E}$ and $\mathcal{I}$ use keys in the same key space $\mathcal{K}$. The **encrypt-then-MAC** system $\mathcal{E}_{\mathrm{EtM}} = (E_{\mathrm{EtM}}, D_{\mathrm{EtM}})$, or EtM for short, is defined as follows:

$$
\begin{aligned}
E_{\mathrm{EtM}}(\ (k_{\mathrm{e}}, k_{\mathrm{m}}),\ m) := \quad & c \xleftarrow{\mathrm{R}} E(k_{\mathrm{e}}, m), \quad t \xleftarrow{\mathrm{R}} S(k_{\mathrm{m}}, c) \\
& \text{Output } (c, t) \\
\\
D_{\mathrm{EtM}}((k_{\mathrm{e}}, k_{\mathrm{m}}),\ (c, t)\ ) := \quad & \text{if } V(k_{\mathrm{m}}, c, t) = \mathsf{reject} \text{ then output } \mathsf{reject} \\
& \text{otherwise, output } D(k_{\mathrm{e}}, c)
\end{aligned}
$$

The EtM system is defined over $(\mathcal{K}^2,\ \mathcal{M},\ \mathcal{C} \times \mathcal{T})$. We emphasize that no plaintext data should be output before the integrity tag over the entire message is verified. The following theorem shows that $\mathcal{E}_{\mathrm{EtM}}$ provides authenticated encryption.

Theorem 9.2. *Let $\mathcal{E} = (E, D)$ be a cipher and let $\mathcal{I} = (S, V)$ be a MAC. Then $\mathcal{E}_{\text{EtM}}$ is* AE*-secure assuming $\mathcal{E}$ is CPA-secure and $\mathcal{I}$ is a secure MAC.*

> *In particular, for every ciphertext integrity adversary $\mathcal{A}_{\text{ci}}$ that attacks $\mathcal{E}_{\text{EtM}}$ as in Attack Game 9.1 there exists a MAC adversary $\mathcal{B}_{\text{mac}}$ that attacks $\mathcal{I}$ as in Attack Game 6.1, where $\mathcal{B}_{\text{mac}}$ is an elementary wrapper around $\mathcal{A}_{\text{ci}}$, such that*
>
> $$\text{CI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{E}_{\text{EtM}}] = \text{MAC}\mathsf{adv}[\mathcal{B}_{\text{mac}}, \mathcal{I}].$$
>
> *For every CPA adversary $\mathcal{A}_{\text{cpa}}$ that attacks $\mathcal{E}_{\text{EtM}}$ as in Attack Game 5.2 there exists a CPA adversary $\mathcal{B}_{\text{cpa}}$ that attacks $\mathcal{E}$ as in Attack Game 5.2, where $\mathcal{B}_{\text{cpa}}$ is an elementary wrapper around $\mathcal{A}_{\text{cpa}}$, such that*
>
> $$\text{CPA}\mathsf{adv}[\mathcal{A}_{\text{cpa}}, \mathcal{E}_{\text{EtM}}] = \text{CPA}\mathsf{adv}[\mathcal{B}_{\text{cpa}}, \mathcal{E}]$$

Proof. Let us first show that $\mathcal{E}_{\text{EtM}}$ provides ciphertext integrity. The proof is by a straight forward reduction. Suppose $\mathcal{A}_{\text{ci}}$ is a ciphertext integrity adversary attacking $\mathcal{E}_{\text{EtM}}$. We construct a MAC adversary $\mathcal{B}_{\text{mac}}$ attacking $\mathcal{I}$.

Adversary $\mathcal{B}_{\text{mac}}$ plays the role of adversary in a MAC attack game for $\mathcal{I}$. It interacts with a MAC challenger $\mathbf{C}_{\text{mac}}$ that starts by picking a random $k_{\text{m}} \xleftarrow{\text{R}} \mathcal{K}$. Adversary $\mathcal{B}_{\text{mac}}$ works by emulating a $\mathcal{E}_{\text{EtM}}$ ciphertext integrity challenger for $\mathcal{A}_{\text{ci}}$, as follows:

> $k_{\text{e}} \xleftarrow{\text{R}} \mathcal{K}$
> upon receiving a query $m_i \in \mathcal{M}$ from $\mathcal{A}_{\text{ci}}$ do:
> $c_i \xleftarrow{\text{R}} E(k_{\text{e}}, m_i)$
> Query $\mathbf{C}_{\text{mac}}$ on c_i and obtain $t_i \xleftarrow{\text{R}} S(k_{\text{m}}, c_i)$ in response
> Send (c_i, t_i) to $\mathcal{A}_{\text{ci}}$ // then $(c_i, t_i) = E_{\text{EtM}}(\,(k_{\text{e}}, k_{\text{m}}),\ m_i)$
> eventually $\mathcal{A}_{\text{ci}}$ outputs a ciphertext $(c, t) \in \mathcal{C} \times \mathcal{T}$
> output the message-tag pair (c, t)

It should be clear that $\mathcal{B}_{\text{mac}}$ responds to $\mathcal{A}_{\text{ci}}$'s queries as in a real ciphertext integrity attack game. Therefore, with probability $\text{CI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{E}_{\text{EtM}}]$ adversary $\mathcal{A}_{\text{ci}}$ outputs a ciphertext (c, t) that makes it win Attack Game 9.1 so that $(c, t) \notin \{(c_1, t_1), \ldots\}$ and $V(k_{\text{m}}, c, t) = \mathsf{accept}$. It follows that (c, t) is a message-tag pair that lets $\mathcal{B}_{\text{mac}}$ win the MAC attack game and therefore $\text{CI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{E}_{\text{EtM}}] = \text{MAC}\mathsf{adv}[\mathcal{B}_{\text{mac}}, \mathcal{I}]$, as required.

It remains to show that if $\mathcal{E}$ is CPA-secure then so is $\mathcal{E}_{\text{EtM}}$. 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 $\mathcal{E}_{\text{EtM}}$. This is straightforward and is left as an easy exercise (see Exercise 5.19). □

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.

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 9.4. In the proof of Theorem 9.2 we relied on the fact that the two keys k_e and k_m are chosen independently in $\mathcal{K}$.

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 $(r, c) \xleftarrow{R} E(k, c)$ where r is a random IV. When implementing encrypt-then-MAC $\mathcal{E}_{\text{EtM}} = (E_{\text{EtM}}, D_{\text{EtM}})$ the encryption algorithm is incorrectly defined as

$$E_{\text{EtM}}\big(\ (k_e, k_m),\ m\big) := \big\{\ (r, c) \xleftarrow{R} E(k_e, m),\ t \xleftarrow{R} S(k_m, c),\ \text{output } (r, c, t)\ \big\}.$$

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. The first edition of the ISO 19772 standard from 2009 for authenticated encryption made precisely this mistake [**?**].

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 $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ and let $\mathcal{E} = (E, D)$ be a cipher defined over $(\mathcal{K},\ \mathcal{M} \times \mathcal{T},\ \mathcal{C})$. The **MAC-then-encrypt** system $\mathcal{E}_{\text{MtE}} = (E_{\text{MtE}}, D_{\text{MtE}})$, or MtE for short, is defined as follows:

$$\begin{aligned}
E_{\text{MtE}}(\ (k_e, k_m),\ m) := \quad & t \xleftarrow{R} S(k_m, m), \quad c \xleftarrow{R} E(k_c,\ (m, t)\ ) \\
& \text{Output } c \\
\\
D_{\text{EtM}}((k_e, k_m),\ c\ ) := \quad & (m, t) \leftarrow D(k_e, c) \\
& \text{if } V(k_m, m, t) = \textsf{reject} \text{ then output } \textsf{reject} \\
& \text{otherwise, output } m
\end{aligned}$$

The MtE system is defined over $(\mathcal{K}^2,\ \mathcal{M},\ \mathcal{C})$.

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** [16].

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 $\mathcal{E}_{\text{MtE}} = (E_{\text{MtE}}, D_{\text{MtE}})$ obtained from applying MtE to randomized CBC mode encryption and a secure MAC works as follows:

- $E_{\text{MtE}}(\ (k_{\text{e}}, k_{\text{m}}),\ m)$: First use the MAC signing algorithm to compute a fixed-length tag $t \xleftarrow{\text{R}} S(k_{\text{m}}, m)$ 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:

  (9.4)

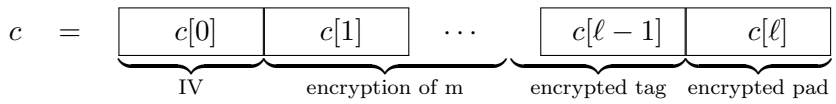
  Notice that the tag t does not protect the integrity of the pad. We will exploit this to break CPA security using a chosen-ciphertext attack.

- $D_{\text{MtE}}(\ (k_{\text{e}}, k_{\text{m}}),\ c)$: Run CBC decryption to obtain the plaintext data in (9.4). Next, remove the pad p by reading the last byte in (9.4) 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 5.11, 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 $\mathcal{E}_{\text{MtE}}$ that lets the adversary decrypt any ciphertext of its choice. It follows that $\mathcal{E}_{\text{MtE}}$ 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 $c := E_{\text{MtE}}(\ (k_{\text{e}}, k_{\text{m}}),\ m)$ 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:



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 $\mathcal{E}_{\text{MtE}}$. The attacker prepares a chosen ciphertext query $\hat{c}$

by replacing the last block of c with $c[1]$. That is,

$$\hat{c} \quad := \quad \boxed{c[0]}\boxed{c[1]} \quad \cdots \quad \boxed{c[\ell-1]}\underbrace{\boxed{c[1]}}_{\text{encrypted pad?}} \tag{9.5}$$

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 $\mathcal{E}_{\mathrm{MtE}}$ 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:

GET `path` cookie: `cookie`

where `path` identifies the name of a resource being requested from `bank.com`. The browser only inserts the cookie into requests 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`

$$\boxed{\text{GET /AA} \quad \text{cookie: } \texttt{cookie}} \tag{9.6}$$

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 (9.5), 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 (9.6) 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 implementations 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 [4]. It is quite challenging to implement TLS 1.0 decryption in way that hides the timing information exploited by the Lucky13 attack.

Informative error messages. To make matter worse, the TLS 1.0 specification [1] 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 9.5 we show that the same holds for randomized CBC, assuming there is no message padding.

Surprisingly, Theorem 9.3 shows that MAC-then-encrypt with randomized counter mode is AE-secure even if the MAC is a one-time MAC. 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 encrypt-then-MAC presented in the previous section suggest that it should not be used.

Let $\mathcal{E} = (E, D)$ be an instance of randomized counter-mode built from a secure PRF F taking inputs in $\mathcal{X} := \{0,1\}^{\ell}$. Let $\mathcal{I} = (S, V)$ be a secure one-time MAC defined over $(\mathcal{K}, \mathcal{M}, \mathcal{T})$ where $\mathcal{M} := \{0,1\}^{\ell_m}$ and $\mathcal{T} := \{0,1\}^{\ell_t}$. The MAC-then-encrypt cipher $\mathcal{E}_{\text{MtE}} = (E_{\text{MtE}}, D_{\text{MtE}})$ built from F and $\mathcal{I}$ and taking messages in $\mathcal{M}$ is defined as follows:

$$E_{\text{MtE}}\big((k_e, k_m),\ m\big) := \big\{\ t \xleftarrow{R} S(k_m, m),\quad c \xleftarrow{R} E\big(k_e,\ (m \,\|\, t)\big),\quad \text{output } c\ \big\}$$

$$D_{\text{EtM}}\big((k_e, k_m),\ c\big) := \left\{ \begin{array}{l} (m \,\|\, t) \leftarrow D(k_e, c) \\ \text{if } V(k_m, m, t) = \mathsf{reject} \text{ then output } \mathsf{reject} \\ \text{otherwise, output } m \end{array} \right\} \tag{9.7}$$

Theorem 9.3. *The cipher $\mathcal{E}_{\text{MtE}} = (E_{\text{MtE}}, D_{\text{MtE}})$ in (9.7) 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 $\mathcal{A}_{\text{ci}}$ that attacks $\mathcal{E}_{\text{MtE}}$ as in Attack Game 9.1 there exists two MAC adversaries $\mathcal{B}_R Mmac$ and $\mathcal{B}'_{\text{mac}}$ that attack $\mathcal{I}$ as in Attack Game 6.1, and a PRF adversary $\mathcal{B}_{\text{prf}}$ that attacks F as in Attack Game 4.2, each of which is an elementary wrapper around $\mathcal{A}_{\text{ci}}$, such that

$$\begin{aligned} \text{CI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{E}_{\text{MtE}}] \le\ & \text{PRF}\mathsf{adv}[\mathcal{B}_{\text{prf}}, F] + \\ & Q \cdot \text{MAC}_1\mathsf{adv}[\mathcal{B}_{\text{mac}}, \mathcal{I}] + \text{MAC}_1\mathsf{adv}[\mathcal{B}'_{\text{mac}}, \mathcal{I}] + \frac{2Q^2(\ell_m + \ell_t)}{|\mathcal{X}|}. \end{aligned} \tag{9.8}$$

For every CPA adversary $\mathcal{A}_{\text{cpa}}$ that attacks $\mathcal{E}_{\text{EtM}}$ as in Attack Game 5.2 there exists a CPA adversary $\mathcal{B}_{\text{cpa}}$ that attacks $\mathcal{E}$ as in Attack Game 5.2, which is an elementary wrapper around $\mathcal{A}_{\text{cpa}}$, such that

$$\text{CPA}\mathsf{adv}[\mathcal{A}_{\text{cpa}}, \mathcal{E}_{\text{MtE}}] = \text{CPA}\mathsf{adv}[\mathcal{B}_{\text{cpa}}, \mathcal{E}]$$

Proof idea. CPA security of the system follows immediately from CPA security of randomized counter mode. The challenge is to prove ciphertext integrity for $\mathcal{E}_{\text{MtE}}$. To do so we let $\mathcal{A}_{\text{ci}}$ interact with the following challengers:

- First, we replace the pseudo-random pads in the counter-mode cipher by truly independent one-time pads. We show that since F is a secure PRF and $1/|\mathcal{X}|$ is negligible, the adversary will not notice the difference.
- The adversary requests a number of ciphertexts $c_1, \ldots, c_Q$. Each ciphertext c_i given to $\mathcal{A}_{\text{ci}}$ is a pair (IV_i, e_i) where IV_i is an initial value and e_i is a one-time pad encryption of a message-tag pair. At the beginning of the game the challenger picks a random $\omega \xleftarrow{R} \{1, \ldots, Q\}$ and hopes that the final forgery $c = (\text{IV}, e)$ from $\mathcal{A}_{\text{ci}}$ uses the same IV as in c_ω, namely $\text{IV} = \text{IV}_\omega$. We show that this happens with probability at least $1/Q$.

- Finally, in ciphertexts $c_1, \ldots, c_Q$ given to $\mathcal{A}_{\text{ci}}$, the challenger need not embed a valid MAC. Indeed, since the tags are encrypted using a one-time pad the adversary cannot distinguish a ciphertext containing a valid tag from a ciphertext with an invalid tag. The only exception is the critical ciphertext c_ω where the challenger must embed a valid tag so that the final forgery c from $\mathcal{A}_{\text{ci}}$ is well defined.

This final challenger only needs a single valid tag throughout its interaction with $\mathcal{A}_{\text{ci}}$. It therefore, implies a one-time MAC adversary that succeeds in breaking the one-time MAC $\mathcal{I}$ whenever $\mathcal{A}_{\text{ci}}$ produces a valid ciphertext forgery. We note that this proof is also the solution to Exercise 7.21. □

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 $\mathcal{E}_{\text{MtE}}$. Then $\Pr[W_0] = \text{CI}\mathsf{adv}[\mathcal{A}_{\text{ci}}, \mathcal{E}_{\text{MtE}}]$.

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.

$k_{\text{mac}} \xleftarrow{\text{R}} \mathcal{K}$ // pick random MAC key
$\omega \xleftarrow{\text{R}} \{1, \ldots, Q\}$ // this ω will be used in Game 2
upon receiving the ith query $m_i \in \{0,1\}^{\ell_m}$ for $i = 1, 2, \ldots$ do:
(1) $t_i \leftarrow S(k_{\text{mac}}, m_i) \in \mathcal{T}$ // compute the tag for m_i
$\text{IV}_i \xleftarrow{\text{R}} \mathcal{X}$ // Pick a random IV
$r_i \xleftarrow{\text{R}} \{0,1\}^{\ell_m + \ell_t}$ // pick a sufficiently long truly random one-time pad
$e_i \leftarrow (m_i \,\|\, t_i) \oplus r_i, \quad c_i \leftarrow (\text{IV}_i, e_i)$ // build ciphertext
send c_i to the adversary

At the end of the game, $\mathcal{A}_{\text{ci}}$ outputs $c = (\text{IV}, e)$, which is not among $c_1, \ldots, c_Q$, and the winning condition is evaluated as follows:

// decrypt ciphertext c
(2) if $\text{IV} = \text{IV}_j$ for some j then $(m \,\|\, t) \leftarrow e \oplus r_j$
(3) otherwise, $r \xleftarrow{\text{R}} \{0,1\}^{\ell_m + \ell_t}$ and $(m \,\|\, t) \leftarrow e \oplus r$
$\mathcal{A}_{\text{ci}}$ wins if $V(k_{\text{mac}}, m, t) = \mathsf{accept}$ // check resulting message-tag pair

The analysis in Theorem 5.3 shows that there exists a PRF adversary $\mathcal{B}_{\text{prf}}$ whose running time is about the same as that of $\mathcal{A}_{\text{ci}}$ such that:

$$|\Pr[W_1] - \Pr[W_0]| \leq \text{PRF}\mathsf{adv}[\mathcal{B}_{\text{prf}}, F] + 2Q^2(\ell_m + \ell_t)/|\mathcal{X}| \tag{9.9}$$

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 $\mathcal{A}_{\text{ci}}$ during the game. In particular, we replace line (3) with

(3) otherwise, the adversary loses in Game 2.

Let Z_1 be the event that the final ciphertext c from $\mathcal{A}_{\text{ci}}$ is valid despite using a previously unused $\text{IV} \in \mathcal{X}$. We know that the two games proceed identically, unless event Z_1 happens. When even Z_1

happens in Game 1 then the resulting pair (m, t) is uniformly random in $\{0,1\}^{\ell_m+\ell_t}$. Such a pair is unlikely to form a valid message-tag pair. In particular, there is a trivial MAC adversary $\mathcal{B}'_{\text{mac}}$ such that $\Pr[Z_1] \leq \text{MAC}_1\mathsf{adv}[\mathcal{B}'_{\text{mac}}, \mathcal{I}]$. Adversary $\mathcal{B}'_{\text{mac}}$ simply outputs a random pair in $\{0,1\}^{\ell_m+\ell_t}$. Hence, by the difference lemma, we have that

$$|\Pr[W_2] - \Pr[W_1]| \leq \text{MAC}_1\mathsf{adv}[\mathcal{B}'_{\text{mac}}, \mathcal{I}] \tag{9.10}$$

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 $\mathcal{A}_{\text{ci}}$. Here ω is a random number in $\{1, \ldots, Q\}$ chosen by the challenger. The only change to the winning condition of Game 2 is that line (2) now becomes:

(2) if $\text{IV} = \text{IV}_\omega$ then $(m \parallel t) \leftarrow e \oplus r_\omega$
(3) otherwise, the adversary loses in Game 2.

Since ω is independent of $\mathcal{A}_{\text{ci}}$'s view we know that

$$\Pr[W_3] \geq (1/Q) \cdot \Pr[W_2] \tag{9.11}$$

Game 4. Finally, we change the challenger so that it only computes a valid tag for query number ω issued by $\mathcal{A}_{\text{ci}}$. 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 change line (1) as follows:

(1) $t_i \leftarrow 0^{\ell_t} \in \mathcal{T}$
if $i = \omega$ then $t_i \leftarrow S(k_{\text{m}}, m_i) \in \mathcal{T}$ // only compute correct tag for m_ω

Since the adversary's view in this game is identical to its view in Game 3 we have

$$\Pr[W_4] = \Pr[W_3] \tag{9.12}$$

Final reduction. We claim that there is a one-time MAC forger $\mathcal{B}_{\text{mac}}$ so that

$$\Pr[W_4] = \text{MAC}_1\mathsf{adv}[\mathcal{B}_{\text{mac}}, \mathcal{I}] \tag{9.13}$$

Adversary $\mathcal{B}_{\text{mac}}$ interacts with a MAC challenger $\mathbf{C}$ and works as follows:

$\omega \xleftarrow{\text{R}} \{1, \ldots, Q\}$
upon receiving the ith query $m_i \in \{0,1\}^{\ell_m}$ for $i = 1, 2, \ldots$ do:
  $t_i \leftarrow 0^{\ell_t} \in \mathcal{T}$
  if $i = \omega$ then query $\mathbf{C}$ for the tag on m_i and let $t_i \in \mathcal{T}$ be the response
  $\text{IV}_i \xleftarrow{\text{R}} \mathcal{X}$ // Pick a random IV
  $r_i \xleftarrow{\text{R}} \{0,1\}^{\ell_m+\ell_t}$ // Pick a sufficiently long random one-time pad
  $e_i \leftarrow (m_i \parallel t_i) \oplus r_i, \qquad c_i \leftarrow (\text{IV}_i, e_i)$
  send c_i to the adversary

when $\mathcal{A}_{\text{ci}}$ outputs $c = (\text{IV}, e)$ from $\mathcal{A}_{\text{ci}}$ do:
  if $\text{IV} = \text{IV}_\omega$ then
    $(m \parallel t) \leftarrow e \oplus r_\omega$
    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 $\mathcal{A}_{\text{ci}}$ wins Game 4 we know that $\mathcal{B}_{\text{mac}}$ does not abort and outputs a pair (m, t) that lets it win the one-time MAC attack game. It follows that $\Pr[W_4] = \text{MAC}_1\mathsf{adv}[\mathcal{B}_{\text{mac}}, \mathcal{I}]$ as required. In summary, putting equations (9.9)–(9.13) together proves the theorem. □

Choosing keys. As we discussed at the end of Section 9.4.1, the proof of Theorem 9.3 relies heavily on the fact that the two keys k_e and k_m are chosen independently. Setting $k_e = k_m$ will invalidate the proof.

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 does is not AE-secure. Moreover, MtE is difficult to implement correctly due to a potential timing side-channel that leads to serious chosen ciphertext attacks. 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 9.6. 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 9.6), 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 $\mathcal{N} \in \mathcal{N}$ is the nonce. Moreover, the encryption algorithm E is required to be deterministic.

Likewise, the decryption syntax becomes

$$D(k, c, d, \aleph)$$

which outputs a message m or reject. We say that the nonce-based AD cipher is defined over $(\mathcal{K}, \mathcal{M}, \mathcal{D}, \mathcal{C}, \mathcal{N})$. 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.* The cipher must satisfy the following **correctness property**: for all keys k, all messages m, all associated data d and all nonces $\aleph \in \mathcal{N}$:

$$\Pr\left[D\big(k,\ E(k,\ m,\ d,\ \aleph),\ d,\ \aleph\big) = m\right] = 1.$$

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 $\aleph_i \in \mathcal{N} \setminus \{\aleph_1, \ldots, \aleph_{i-1}\}$.
>
> The challenger computes $c_i \leftarrow E(k, m_{ib}, d_i, \aleph_i)$, 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 $\aleph_i$, 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

$$\mathrm{nCPA}_{\mathrm{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}] := |\Pr[W_0] - \Pr[W_1]|. \quad \square$$

Definition 9.7 (nonce-based 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 $(c, d, \aleph)$ 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 $(\mathcal{K}, \mathcal{M}, \mathcal{D}, \mathcal{C}, \mathcal{N})$, and a given adversary $\mathcal{A}$, the attack game runs as follows:

- The challenger chooses a random $k \xleftarrow{\mathrm{R}} \mathcal{K}$.
- $\mathcal{A}$ 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 $\aleph_i \in \mathcal{N} \setminus \{\aleph_1, \ldots, \aleph_{i-1}\}$. The challenger computes $c_i \xleftarrow{\mathrm{R}} E(k, m_i, d_i, \aleph_i)$, and gives c_i to $\mathcal{A}$.

- Eventually $\mathcal{A}$ outputs a candidate triple $(c, d, \aleph)$ where $c \in \mathcal{C}$, $d \in \mathcal{D}$, and $\aleph \in \mathcal{N}$ that is not among the triples it was given, i.e.,

$$(c, d, \aleph) \notin \{(c_1, d_1, \aleph_1),\ (c_2, d_2, \aleph_2),\ \ldots\}.$$

We say that $\mathcal{A}$ wins the game if $D(k, c, d, \aleph) \neq \mathsf{reject}$. We define $\mathcal{A}$'s advantage with respect to $\mathcal{E}$, denoted $\mathrm{nCI}_{\mathrm{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$, 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 $\mathrm{nCI}_{\mathrm{ad}}\mathsf{adv}[\mathcal{A}, \mathcal{E}]$ 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) with a nonce-based secure MAC (S, V) as follows:

$$\begin{aligned} E_{\mathrm{EtM}}(\ (k_{\mathrm{e}}, k_{\mathrm{m}}),\ m, d, \aleph) := \quad & c \xleftarrow{\mathrm{R}} E(k_{\mathrm{e}}, m, \aleph), \quad t \xleftarrow{\mathrm{R}} S(k_{\mathrm{m}}, (c, d), \aleph) \\ & \text{Output } (c, t) \\ D_{\mathrm{EtM}}((k_{\mathrm{e}}, k_{\mathrm{m}}),\ (c, t),\ d, \aleph) := \quad & \text{if } V(k_{\mathrm{m}},\ (c, d),\ t,\ \aleph) = \mathsf{reject} \text{ then output } \mathsf{reject} \\ & \text{otherwise, output } D(k_{\mathrm{e}}, c, d, \aleph) \end{aligned}$$

The EtM system is defined over $(\mathcal{K}^2,\ \mathcal{M},\ \mathcal{C} \times \mathcal{T})$. 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. Then $\mathcal{E}_{\mathrm{EtM}}$ is a nonce-based AEAD cipher assuming $\mathcal{E}$ is CPA-secure and $\mathcal{I}$ is a secure MAC.*

The proof of Theorem 9.4 is essentially the same as the proof of Theorem 9.2.

9.6 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 $\aleph$ 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 $\mathcal{N} \in \{0,1\}^{96}$

$k_{\mathrm{m}} \leftarrow E(k, 0^n)$ // first, generate the key for GHASH (a variant of H_{xpoly})

Compute the initial value of the counter in counter mode encryption:

$x \leftarrow (\mathcal{N} \,\|\, 0^{31}1) \in \{0,1\}^{128}$

$x' \leftarrow x + 1$ // initial value of counter

$c \leftarrow$ {encryption of 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:

(∗) $h \leftarrow \mathrm{GHASH}\Big(k_{\mathrm{m}},\ \big(d' \,\|\, c' \,\|\, \mathrm{length}(d) \,\|\, \mathrm{length}(c)\big)\Big) \ \in \{0,1\}^{128}$

$t \leftarrow h \oplus E(k, x) \ \in \{0,1\}^{128}$

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 $x \leftarrow \mathrm{GHASH}\big(k_{\mathrm{m}},\ (\mathcal{N}' \,\|\, \mathrm{length}(\mathcal{N}))\big)$ 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), associate data d and a nonce $\mathcal{N}$. It operates as in encrypt-then-MAC: it first derives $k_{\mathrm{m}} \leftarrow E(k, 0^n)$ 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 and GHASH is essentially the same thing. Recall that $H_{\mathrm{xpoly}}(k, z)$ works by evaluating a polynomial derived from z at the point k. We described H_{xpoly} using arithmetic modulo a prime p so that both 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 $\mathrm{GF}(2)$ 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, while multiplication

is a bit more complicated, but still not too difficult (see below — many modern computers provide direct hardware support).

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 $\mathrm{GF}(2^{128})$:

$$\mathrm{GHASH}(k, z) := z[0]k^v + z[1]k^{v-1} + \ldots + z[v-1]k \ \in \mathrm{GF}(2^{128}) \tag{9.14}$$

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 just 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, which 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 9.10.

GCM has no nonce re-use resistance. If a nonce is accidentally re-used on two different messages then all secrecy for those message is lost. Even worse, the GHASH secret key k_{m} is exposed (Exercise 7.16) 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 (9.14) 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 recently 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 AES in counter mode. However, due to improvements in hardware implementations of 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 extra overhead of GHASH. However, upcoming improvements in the implementation of PCLMULQDQ will likely make GCM just slightly more expensive than pure counter mode, which is the best one can hope for.

We should point out that it already is possible to implement secure authenticated encryption at a cost that is not much more than the cost of AES counter mode — this can be achieved using an integrated scheme such as OCB (see Exercise 9.14).

9.7 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 do 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 [1]. TLS 1.0 is a minor variation of SSL 3.0 and is often referred to as SSL version 3.1. TLS is supported by most major browsers and web servers and TLS 1.3 is the recommended protocol to use. We will mostly focus on TLS 1.3 here.

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 **??**. 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 8.9.5) 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: a concatenation of three fields: the encrypting party's 64-bit write sequence number, a 1-byte record type (a value of 23 means application data), and a 2-byte protocol version (set to 3.1 in TLS 3.1).

- 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 2-byte protocol version set to 3.1 for TLS 3.1, `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 chosen at random? Why not simply set it 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 associate data d. This opens up the system to an exhaustive search attack for the key k using a *time-space tradeoff* discussed in Chapter 18. 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. [20] 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 sequence number to discard 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 record will be discarded because the AEAD decryption algorithm will be given the wrong nonce as input.

9.8 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 9.6) in an attempt to construct a secure AEAD cipher. Specifically, SSHv2 encryption works as follows (Fig. 9.3):

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 5.11.

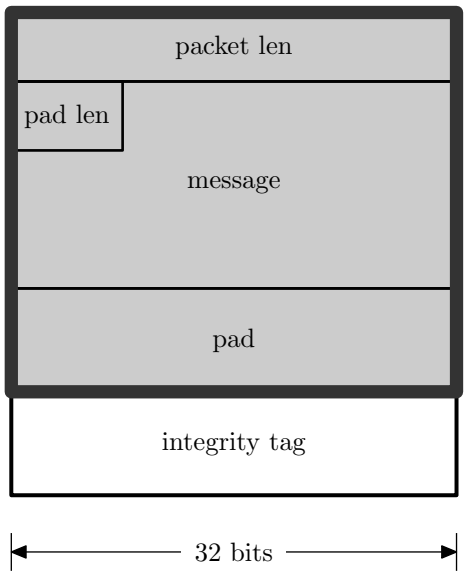


Figure 9.3: An SSHv2 packet

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 9.6 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 [18], 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 [2] 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 [18] 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 [65]. 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 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.

9.9 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

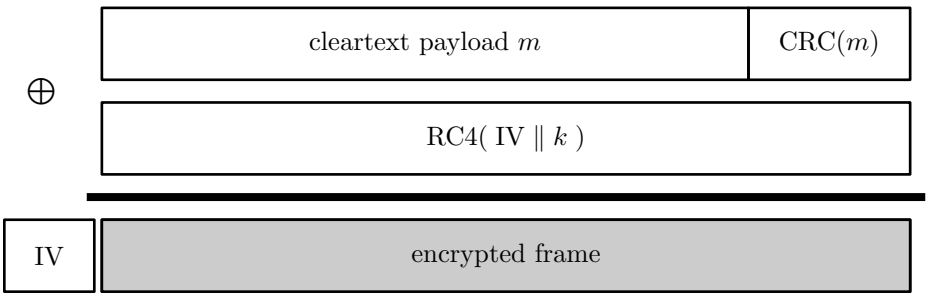


Figure 9.4: WEP Encryption

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:

$$c \leftarrow \big(m \,\|\, \text{CRC}(m)\big) \;\oplus\; \text{RC4}(\text{IV} \,\|\, k)$$
$$c_{\text{full}} \leftarrow (\text{IV},\; c)$$

The WEP encryption process is shown in Fig. 9.4. The receiver decrypts by first computing $c \oplus \text{RC4}(\text{IV} \,\|\, k)$ to obtain a pair (m, s). The receiver accepts the frame if $s = \text{CRC}(m)$ 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 $k_{\text{f}} := \text{IV} \,\|\, k$. 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 $1 \,\|\, k,\ 2 \,\|\, k, \ldots$ 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 [27] 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 [68] 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

$$c = \big(m, \text{CRC}(m)\big) \ \oplus \ \text{RC4}(\text{IV} \,\|\, k)$$

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

$$\begin{aligned} c' \ = \ & \text{RC4}(\text{IV} \,\|\, k) \ \oplus \ \big(m,\ \text{CRC}(m)\big) \ \oplus \ \big(\Delta,\ L(\Delta)\big) = \\ & \text{RC4}(\text{IV} \,\|\, k) \ \oplus \ \big(m \oplus \Delta,\ \ \text{CRC}(m) \oplus L(\Delta)\big) = \\ & \text{RC4}(\text{IV} \,\|\, k) \ \oplus \ \big(m \oplus \Delta,\ \ \text{CRC}(m \oplus \Delta)\big) \end{aligned}$$

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 9.13.

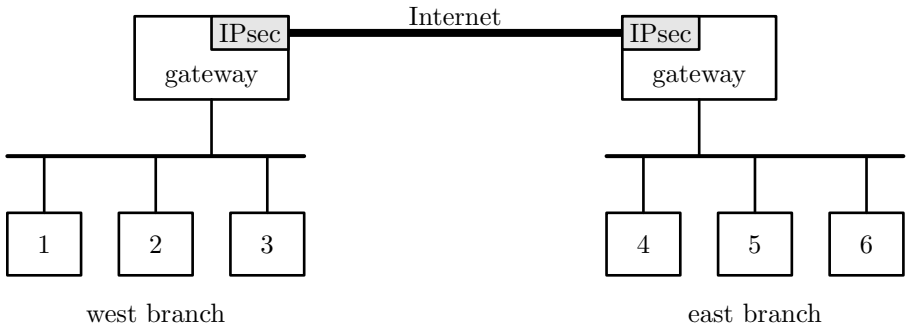


Figure 9.5: A virtual private network (VPN) between east and west office branches

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 allocates 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 [54].

9.10 Case study: IPsec

The IPsec protocol provides confidentiality and integrity for Internet IP packets. The protocol was first published in 1998 and was subsequently updated in 2005. The IPsec protocol consists of many sub-protocols that are not relevant for our discussion here. In this section we will focus on the most commonly used IPsec protocol called **encapsulated security payload** (ESP) in tunnel mode.

Virtual private networks (VPNs) are an important application for IPsec. A VPN enables two office branches to communicate securely over a public Internet channel, as shown in Fig. 9.5. Here, packets from machines 1,2,3 are encrypted at the west gateway using IPsec and transmitted over the public channel. The east gateway decrypts each received packet and forwards it to its destination inside the east branch, namely, one of 4,5,6. We note that all packets sent from west to east are encrypted using the same cryptographic key $k_{w \to e}$. Packets sent from east to west are processed similarly, but encrypted using a different key, $k_{e \to w}$. We will use this VPN example as our motivating example for IPsec.

To understand IPsec one first needs a basic understanding of the IP protocol. Here we focus on IP version 4 (IPv4), which is currently widely deployed. The left side of Fig. 9.6 shows a (cleartext)

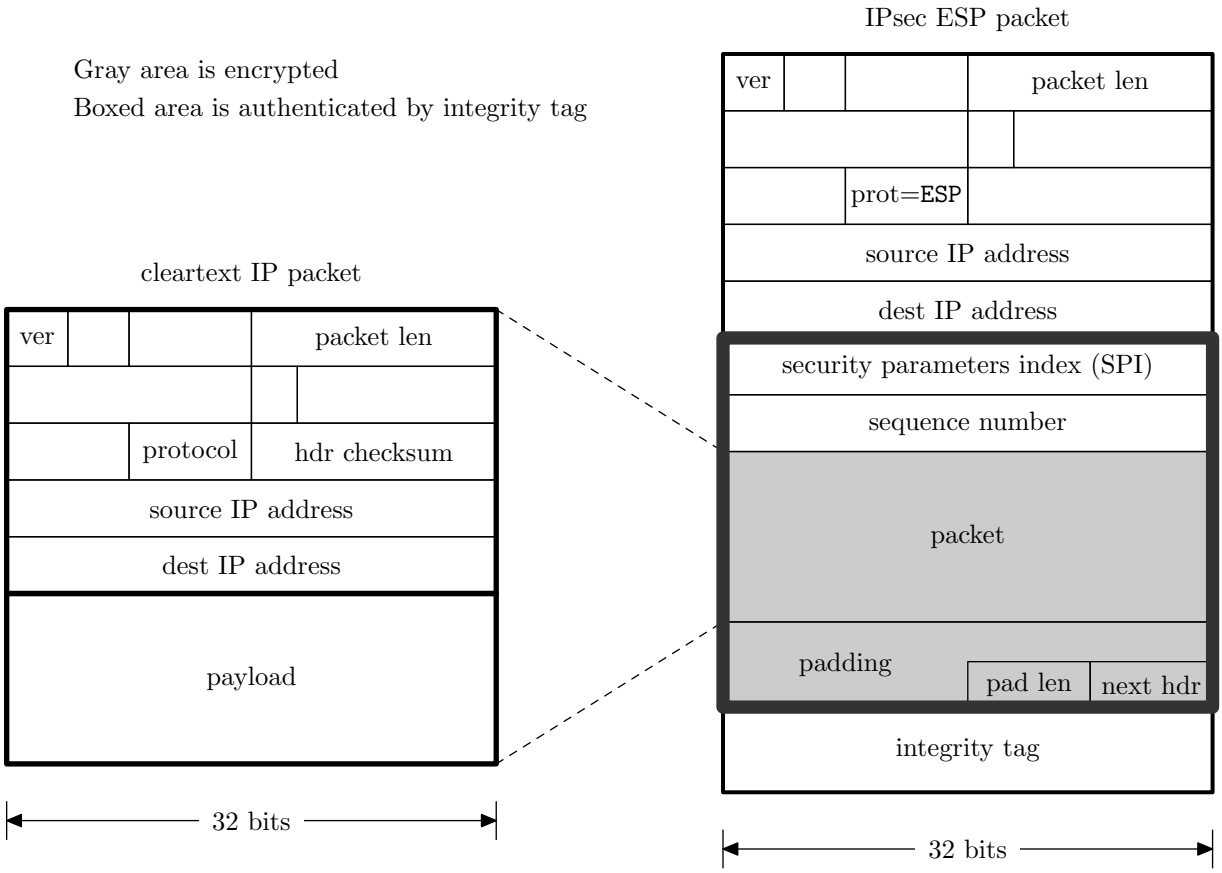


Figure 9.6: Cleartext IPv4 packet and an IPsec ESP packet

IPv4 packet. The packet consists of a packet header and a packet payload. The header contains a bunch of fields, but only a few are relevant to our discussion:

- The first four bits indicate the **version** number which is set to 4 for IPv4.
- The 2-byte **packet length** field contains the length in bytes of the entire packet including the header.
- The 1-byte **protocol** field describes the packet payload For example, protocol = 6 indicates a TCP payload.
- the 2-byte **header checksum** contains a checksum of all header bytes (excluding the checksum field). The checksum is used to detect random transmission errors in the header. Packets with an invalid checksum are dropped at the recipient. The checksum can be computed by anyone and consequently provides no integrity against an attacker. In fact, Internet routers regularly change fields in the packet header as the packet moves from router to router and recompute the checksum.
- The source and destination IP indicate the source and destination addresses for the packet.
- The **payload** contains the packet contents and is variable length.

IPsec encapsulated security payload (ESP). The right side of Fig. 9.6 shows the result of encrypting a packet with ESP in tunnel mode. We first describe the fields in the encrypted packet and then describe the encryption process.

IPsec key management — the SPI field. Every ESP endpoint maintains a **security association database** (SAD). A record in the SAD is called a **security association** (SA) and is identified by a 32 bit identifier called a **security parameters index** (SPI). A SAD record (an SA) contains many connection-specific parameters, such as the ESP encryption algorithm (e.g. 3DES-CBC or AES-CBC), the ESP secret key (e.g. $k_{w\to e}$ or $k_{e\to w}$), the source and destination IP addresses, the SPI, and various key-exchange parameters.

When the east branch gateway sends out a packet, it uses the packet's destination IP address and other parameters to choose a security association (SA) in its security association database (SAD). The gateway embeds the 32-bit SPI of the chosen SA in the packet header and encrypts the packet using the secret key specified in the SA. When the packet arrives at its destination, the recipient locates an appropriate SA in its own SAD using the following algorithm:

1. First, look for an SA matching the received (SPI, dest address, source address);
2. If no match is found, the recipient looks for a match based on the (SPI, dest address) pair;
3. Otherwise, it looks for a match based on the SPI only.

If no SA exists for the received packet, the packet is discarded. Otherwise, the gateway decrypts the packet using the secret key specified in the chosen SA. Most often an SA is used for transmitting packets in one direction, e.g., from east to west. A bi-directional TCP connection between east and west uses two separate SAs — one for packets from east to west and one for packets from west to east. Generally, an ESP endpoint maintains two SAD records for each peer.

The SAD at a particular host is managed semi-manually. Some parameters are managed manually while others are negotiated between the communicating hosts. In particular, an SA secret

key can be set manually at both endpoints or it can be negotiated using an IPsec key exchange protocol called IKE [**?**]. We will not discuss SAD management here.

ESP anti-replay — the sequence number field. The sequence number enables the recipient to detect and discard duplicate packets. Duplication can result from a network error or can be caused by an attacker who is deliberately replaying old packets. Every ESP end point maintains a **sequence number** for each security association. By default the sequence number is 64 bits long (called an extended sequence number), although older versions of ESP use a shorter 32 bit sequence number. The sequence number is initialized to zero when the security association is created and is incremented by one for each packet sent using the SA. The entire 64 bits are included in the MAC calculation. However, only the 32 least significant bits (LSB) are included in the ESP packet header. In other words, ESP endpoints maintain 64-bit counters, of which the 32 MSBs are implicit while the 32 LSBs are explicit in the packet header.

For our discussion of sequence numbers, we assume that there is at most a single host sending packets for each security association (SA). Hence, for a particular SA there is no danger of two hosts sending a packet with the same sequence number. Note that multiple hosts can receive packets for a particular SA, as in the case of multicast. We only disallow multiple hosts from sending packets using a single SA.

For a particular SA, the recipient must discard any packet that contains a 32-bit sequence number that was previously contained in an earlier packet. Since packets can arrive out of order, verifying sequence number unicity at the recipient takes some effort. RFC 4303 recommends that the recipient maintain a window (e.g. bit vector) of size 32. The "right" edge of the window represents the highest, validated sequence number value received on this SA. Packets that contain sequence numbers lower than the "left" edge of the window are discarded. Received packets falling within the window are checked against the list of received packets within the window, and are discarded if their sequence number was already seen. The window shifts whenever a valid packet with a sequence number on the "right" of the current window is received. Consequently, the receiver recovers gracefully from a long sequence of lost packets

If more than 2^{32} consecutive packets are lost, then the 64-bit sequence numbers at the sender and receiver will go out of sync — the 32 MSBs implicitly maintained by the two will differ. As a result, all further packets will be rejected due to MAC validation failure. This explains why the designers of ESP chose to include 32 bits in the packet header — a loss of 2^{32} packets in unlikely. Including fewer bits (e.g. 16 bits) would have greatly increased the chance of communication failure.

Padding and the next header field. ESP first appends a pad to ensure that the length of the data to encrypt is a multiple of the block length of the chosen encryption algorithm (e.g. a multiple of 16 bytes for AES-CBC). It also ensures that the resulting ciphertext length is a multiple of four bytes. The pad length is anywhere from 0 to 255 bytes. An additional pad-length byte is appended to indicate the number of padding bytes preceding it. Finally, a next header (`next-hdr`) byte, is appended to indicate the payload type. Most often the payload type is an IPv4 packet in which case `next-hdr`=4.

ESP supports an optional **traffic flow confidentiality** (TFC) service where the sender attempts to hide the length of the plaintext packet. To do so, the sender appends dummy (unspecified) bytes to the payload before padding takes place. The length of the TFC pad is arbitrary. The packet length field in the plaintext IP header indicates the beginning of the TFC pad. The TFC pad is removed after decryption.

ESP also supports "dummy" packets to defeat traffic analysis. The goal is to prevent an observer

from telling when the sender transmits data. For example, one can instruct the sender to transmit a packet every millisecond, whether it has data to send or not. When no data is available, the sender transmits a "dummy" packet which is indicated by setting `next-hdr`=59. Since the `next-hdr` field is encrypted an observer cannot tell dummy packets from real packets. However, at the destination, all dummy packets are discarded immediately after decryption.

The encryption process. ESP implements the encrypt-then-MAC method in four steps. We discuss each step in turn.

1. **Pad.** The pad, including the optional TFC pad and next header field, are appended to the plaintext IP packet.

2. **Encrypt.** The gray area in Fig. 9.6 is encrypted with the algorithm and key specified by the SA. ESP supports a variety of encryption algorithms, but is required to support 3DES-CBC, AES-CBC, and AES counter mode. For CBC modes the IV is prepended to the encrypted payload and is sent in the clear. The encryption algorithm can be set to NULL in which case no encryption takes place. This is used when ESP provides integrity but no confidentiality.

3. **MAC.** An integrity tag is computed using an algorithm and key specified in the SA. The tag is computed over the following data

   $$\text{SPI} \parallel \text{64-bit sequence number} \parallel \text{ciphertext}$$

   where ciphertext is the result of Step 2. Note that the tag is computed over the 64 bit sequence number even though only 32 bits are embedded in the packet. The resulting tag is placed in the integrity tag field following the ciphertext. ESP supports a variety of MAC algorithms, but is required to support HMAC-SHA1-96, HMAC-MD5-96, and AES-XCBC-MAC-96 (XCBC-MAC is a variant of CMAC). The integrity tag field is optional and is omitted if the encryption algorithm already provides authenticated encryption, as in the case of GCM.

4. **Encapsulate.** Finally, an IPv4 packet header is prepended to obtain an ESP packet as shown on the right side of Fig. 9.6. The protocol field in the IPv4 header is set to 50 indicating an ESP payload.

Decryption follows a similar process. The recipient first checks the 32-bit sequence number. If the value is repeated or outside the allowed window, the packet is dropped. Next, the recipient checks the tag field, and rejects the packet if MAC verification fails. The packet is then decrypted and the padding removed. If the packet is a dummy packet (i.e. the next header field is equal to 59), the packet is discarded. Finally, the original cleartext packet is reconstructed and sent to the destination. Note that in principle, the sequence number field could have been encrypted. The designers of ESP chose to send the field in the clear so as to reduce the time until a duplicate packet is rejected.

Security. IP packets can arrive at any order, be duplicated, and even modified. By relying on encrypt-then-MAC and on the sequence number, ESP ensures that the recipient sees a data stream identical to the one transmitted by the sender. One issue that haunts ESP is a setting that provides CPA-secure encryption without an integrity check. RFC 4303 states that

ESP allows encryption-only SAs because this may offer considerably better performance and still provide adequate security, e.g., when higher-layer authentication/integrity protection is offered independently.

Relying on a higher application layer for integrity is highly risky. On the sender side the application layer processes data before passing it to the IP layer. Hence, this implements MAC-then-encrypt which from a theoretical point view we know can be insecure. More importantly, in practice it is dangerous to assume that the higher layer will protect the entire IP packet. For example, a higher layer such as SSL may provide integrity without encryption. Combining encryption-only ESP and integrity-only SSL will be insecure since the SSL layer will not provide integrity for the encrypted packet header. As a result, an attacker can tamper with the destination IP field in the encrypted packet. The recipient's IPsec gateway will decrypt the packet and forward the result to an unintended destination, thus causing a serious privacy breach. This and other dangers of the ESP encryption-only mode are discussed in [7, 55].

We note, however, that when the cipher used provides authenticated encryption (such as GCM mode) it is perfectly fine to use encryption without an integrity check, since the cipher already provides authenticated encryption.

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. Let (E, D) be an AE-secure cipher. Consider the following derived ciphers:

(a) $E_1(k,m) := \big(E(k,m),\ E(k,m)\big)$; $\quad D_2\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } D(k,c_1) = D(k,c_2) \\ \mathsf{reject} & \text{otherwise} \end{cases}$

(b) $E_2(k,m) := \big\{c \leftarrow E(k,m),\ \text{output } (c,c)\big\}$; $\quad D_2\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } c_1 = c_2 \\ \mathsf{reject} & \text{otherwise} \end{cases}$

Show that part (b) is AE-secure, but part (a) is not.

9.2. 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:

$$E_1(k,m) := \big(E(k,m),\ H_1(m)\big); \quad D_1\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } H_1(D(k,c_1)) = c_2 \\ \mathsf{reject} & \text{otherwise} \end{cases}$$

$$E_2(k,m) := \big(E(k,m),\ H_2(c)\big); \quad D_2\big(k,\ (c_1,c_2)\big) := \begin{cases} D(k,c_1) & \text{if } H_2(c_1) = c_2 \\ \mathsf{reject} & \text{otherwise} \end{cases}$$

Show that both ciphers are not AE-secure.

9.3. Let (E, D) be an AE-secure cipher defined over $(\mathcal{K}, \mathcal{M}, \mathcal{C})$ where $D(k, \mathsf{reject}) := \mathsf{reject}$ for all keys $k \in \mathcal{K}$. Consider the following derived cipher:

$$E'\big((k_1, k_2), m\big) := E(k_2, E(k_1, m)); \quad D'\big((k_1, k_2),\ c\big) := D(k_1,\ D(k_2, c))$$

(a) Show that (E', D') is AE-secure 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.

9.4. 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 RawCBC built from (E, D) as the secure MAC. This MAC is secure in this context because it is only being applied 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 is not CPA secure.

9.5 (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. For concreteness, if 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 the message that is CBC-encrypted is the block sequence $m \,\|\, t$.

9.6 (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.

$$E_1\big((k_e, k_m),\ m,\ d,\ \nu\big) := \big\{ t \leftarrow F\big(k_m,\ (m, d, \nu)\big),\ c \xleftarrow{R} E(k_c, m; t),\ \text{output } (c, t) \big\}$$

$$D_1\big((k_e, k_m),\ (c, t),\ d,\ \nu\big) := \left\{ \begin{array}{l} m \leftarrow D(k_e, c; t) \\ \text{if } F\big(k_m,\ (m, d, \nu)\big) \neq t \text{ output } \mathsf{reject}\text{, otherwise output } m \end{array} \right\}$$

This method is loosely called enrypt-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 9.7) 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.

9.7 (Authenticated SIV). We discuss a modification of the SIV construction, introduced in Exercise 5.7, 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 5.7, 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 very 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 5.7).

9.8. Let (E, D) be a block cipher defined over $(\mathcal{K},\ \mathcal{M} \times \mathcal{R})$.

(a) As in Exercise 5.5, let (E', D') be defined as

$$E'(k, m) := \big\{r \xleftarrow{R} \mathcal{R},\ c \xleftarrow{R} E\big(k,\ (m, r)\big),\ \text{output } c\big\}$$
$$D'(k, c) := \big\{(m, r') \leftarrow D(k, c),\ \text{output } m\big\}$$

Show that (E', D') is CCA-secure provided (E, D) is a *strongly secure* block cipher and $1/|\mathcal{R}|$ is negligible.

Note: 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.

9.9. 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.

9.10 (GCM analysis). Give a complete security analysis of GCM (see Section 9.6). 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.

9.11. 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 that the MAC-then-Encrypt counter-example (Section 9.4.2) shows that a system that is CPA and PI secure is not necessarily 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.

9.12 (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

$$S\big((k_1, k_2),\ m\big) := E\big(k_1,\ H(k_2, m)\big)$$

$$V\big((k_1, k_2),\ m,\ c\big) := \begin{cases} \mathsf{accept} & \text{if } H(k_2, m) = D(k_1, c), \\ \mathsf{reject} & \text{otherwise.} \end{cases}$$

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.

9.13 (chop-chop attack). A parity 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 $m' = m \,\|\, b$ 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 random errors.

Consider a cipher where encryption is done using randomized counter mode without any padding. Messages are variable length bit strings and ciphertexts are bit strings of the same length as plaintext. No MAC is used, but before the plaintext is encrypted, the sender appends a parity bit to the end of the plaintext. After receiver decrypts, he checks the parity bit and returns either the plaintext or reject.

Design a chosen-ciphertext attack that recovers the complete plaintext of every encrypted message. Hint: use the fact that the system encrypts variable length messages. 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.

9.14 (Simplified OCB mode). OCB is an elegant and efficient AE cipher built from a tweakable block cipher (as defined in Exercise 4.12). 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 $\mathcal{T} := \mathcal{N} \times \{-\ell, \ldots, \ell\}$. 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.

$E'(k, m, \nu) :=$

create (uninitialized) $c \in \mathcal{X}^{|m|}$
checksum $\leftarrow 0^n$
for $i = 0, \ldots, |m| - 1$:
  $c[i] \leftarrow E\big(k,\ m[i],\ (\nu, i+1)\big)$
  checksum $\leftarrow$ checksum $\oplus\ m[i]$
$t \leftarrow E\big(k,\ \text{checksum},\ (\nu, -|m|)\big)$
output (c, t)

$D'(k, (c, t), \nu) :=$

create (uninitialized) $m \in \mathcal{X}^{|c|}$
checksum $\leftarrow 0^n$
for $i = 0, \ldots, |c| - 1$:
  $m[i] \leftarrow D\big(k,\ c[i],\ (\nu, i+1)\big)$
  checksum $\leftarrow$ checksum $\oplus\ m[i]$
$t' \leftarrow E\big(k,\ \text{checksum},\ (\nu, -|c|)\big)$
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 $t \leftarrow E\big(k,\ \text{checksum},\ (\nu, 0)\big)$ then the scheme would be insecure: it would have no ciphertext integrity.

9.15 (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 for 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., $k' \xleftarrow{R} K(k)$), 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 $c \xleftarrow{R} E(k, m)$ and $k' \xleftarrow{R} K(k)$.

(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 $D'(k', \cdot)$, and the adversary knows nothing about k, and

  (iii) the cipher provides ciphertext integrity with respect to the decryption algorithm $D(k, \cdot)$, and the adversary is given a sub-key $k' \xleftarrow{R} K(k)$, 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

Part III

Protocols

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 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: $3^4 \bmod 5 = 81 \bmod 5 = 1$

3. The *inverse* of $x \in \mathbb{Z}_p$ is an element a satisfying $a \cdot x = 1$.
   The inverse of x in $\mathbb{Z}_p$ is denoted by x^{-1}.
   Example: 1. 3^{-1} in $\mathbb{Z}_5$ is 2 since $2 \cdot 3 = 1 \bmod 5$.
   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: $x^{-1} = x^{p-2} \bmod p$.
   Indeed, $x^{p-2} \cdot x = x^{p-1} = 1 \bmod p$.

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 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 $\mathbb{Z}_7^*$: $\langle 3 \rangle = \{1, 3, 3^2, 3^3, 3^4, 3^5, 3^6\} = \{1, 3, 2, 6, 4, 5\} \pmod 7 = \mathbb{Z}_7^*$.

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:
   for $g \in \mathbb{Z}_p^*$ 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, $x^r \bmod p$ can be computed in time $(\log_2 r)O(n^2)$ where p is n bits long. Note, the algorithm takes linear time in the length of r.

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: $a \cdot x = b \bmod n$.
   Solution: $x = b \cdot a^{-1}$ 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: $\varphi(15) = \left|\{1, 2, 4, 7, 8, 11, 13, 14\}\right| = 8 = 2 * 4$.

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 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) $\Pr[C] \geq 1 - e^{-k(k-1)/2n} \geq \min\left\{\frac{k(k-1)}{4n}, 0.63\right\}$, *and*

(ii) $\Pr[C] \leq 1 - e^{-k(k-1)/n}$ *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 $x \in [0, 1]$. □

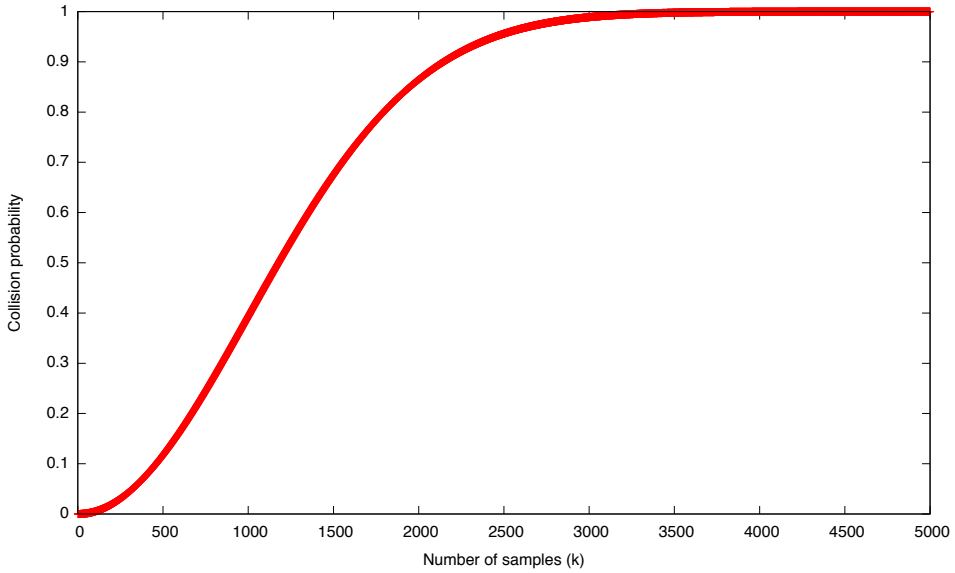
Most frequently we will use the lower bound to say that a collision happens with *at least* a certain probability. But occasionally we will use the upper bound to argue that collisions do not happen.

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 $T_x \xleftarrow{R} X$ and $T_y \xleftarrow{R} Y$. 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] The TLS protocol version 1.0. Internet RFC 2246, 1999.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In *Proceedings of Crypto '97*, volume 1294 of *LNCS*. Springer-Verlag, 1997.

[7] S. Bellovin. Problem areas for the IP security protocols. In *Proceedings of the Sixth Usenix Security Symposium*, pages 1–16, 1996.

[8] S. Bellovin and M. Merritt. Limitations of the kerberos authentication system. In *Proceedings of USENIX '91*, 1991.

[9] S. M. Bellovin. Frank miller: Inventor of the one-time pad. *Cryptologia*, 35(3):203–222, 2011.

[10] 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.

[11] A. Biryukov and J. Großschädl. Cryptanalysis of the full AES using gpu-like special-purpose hardware. *Fundam. Inform.*, 114(3-4):221–237, 2012.

[12] A. Biryukov and D. Khovratovich. Related-key cryptanalysis of the full aes-192 and aes-256. In *Advances in Cryptology–ASIACRYPT 2009*, pages 1–18. 2009.

[13] A. Biryukov, J. Lano, and B. Preneel. Cryptanalysis of the alleged securid hash function. IACR eprint archive, Report 162, 2003, 2003.

[14] 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.

[15] 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.

[16] K. K. Bodo Möller, Thai Duong. This poodle bites: Exploiting the ssl 3.0 fallback. Technical report, 2014.

[17] A. Bogdanov, D. Khovratovich, and C. Rechberger. Biclique cryptanalysis of the full aes. In *Advances in Cryptology–ASIACRYPT 2011*, pages 344–371. 2011.

[18] 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.

[19] 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.

[20] 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.

[21] D. Coppersmith. The data encryption standard and its strength against attack. *IBM Journal of Research and Development*, 38(3), 1994.

[22] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer, 2002.

[23] 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.

[24] I. Dinur, O. Dunkelman, N. Keller, and A. Shamir. Efficient dissection of composite problems, with applications to cryptanalysis, knapsacks, and combinatorial search problems. In *Advances in Cryptology–CRYPTO 2012*, pages 719–740. 2012.

[25] 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.

[26] D. Feldmeier and P. Karn. UNIX password security – 10 years later. In *Proceedings of Crypto '89*, pages 44–63, 1989.

[27] 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.

[28] 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.

[29] A. M. Frieze, R. Kannan, and J. C. Lagarias. Linear congruential generators do not produce random sequences. In *FOCS*, pages 480–484, 1984.

[30] S. Garfinkel and A. Shelat. Remembrance of data passed: A study of disk sanitization practices. *IEEE Security and Privacy*, January 2003.

[31] D. Genkin, A. Shamir, and E. Tromer. RSA key extraction via low-bandwidth acoustic cryptanalysis. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference*, pages 444–461, 2014.

[32] 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.

[33] 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.

[34] T. Guneysu, T. Kasper, M. Novotny, C. Paar, and A. Rupp. Cryptanalysis with copacobana. *Computers, IEEE Transactions on*, 57(11):1498–1513, 2008.

[35] J. Haynes and H. Klehr. *Venona: Decoding Soviet Espionage in America*. Yale University Press, 1999.

[36] 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.

[37] M. Joye and M. Tunstall, editors. *Fault Analysis in Cryptography*. Information Security and Cryptography. Springer, 2012.

[38] P. Junod. On the complexity of matsui's attack. In *Selected Areas in Cryptography (SAC)*, pages 199–211, 2001.

[39] G. Kim and E. Spafford. The design and implementation of tripwire: a file system integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and communications security*, pages 18–29, 1994.

[40] P. C. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. *J. Cryptographic Engineering*, 1(1):5–27, 2011.

[41] A. Lenstra. *Key lengths*. Wiley, 2005.

[42] L. Levin. One-way functions and pseudorandom generators. *Combinatorica*, 7(4):357–363, 1987.

[43] S. Lucks. Attacking triple encryption. In *Proceedings of Fast Software Encryption 1998*, LNCS, pages 239–253, 1998.

[44] I. Mantin and A. Shamir. A practical attack on broadcast rc4. In *Proceedings of FSE 2001*. Springer-Verlag, 2001.

[45] M. Matsui. The first experimental cryptanalysis of the data encryption standard. In *Proceedings of Crypto'94*, pages 1–11, 1994.

[46] M. Matsui. Linear cryptanalysis method for des cipher. In *Proceedings of Eurocrypt'93*, pages 386–397, 1994.

[47] R. Merkle and M. Hellman. On the security of multiple encryption. *Communications of the ACM*, 24:465–467, 1981.

[48] I. Mironov. Hash functions: From merkle-damgard to shoup. In *Proceedings of Eurocrypt '01*, volume 2045 of *LNCS*, pages 166–181. Springer-Verlag, 2001.

[49] T. Moran, M. Naor, and G. Segev. An optimally fair coin toss. In *TCC*, pages 1–18, 2009.

[50] R. Morris and K. Thompson. Password security: A case history. *Communications of the ACM*, 22(11):594–497, 1979.

[51] 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.

[52] M. Naor. Bit commitment using pseudo-randomness. In *CRYPTO*, pages 128–136, 1989.

[53] Nist recommendation for key management part 1: General, 2005.

[54] Recommendation for block cipher modes of operation: The ccm mode for authentication and confidentiality, 2004.

[55] K. Paterson and A. Yau. Cryptography in theory and practice: The case of encryption in ipsec. In *Proceedings of Eurocrypt '06*, volume 4004 of *LNCS*, pages 12–29. Springer-Verlag, 2006.

[56] A. Prado, N. Harris, and Y. Gluck. SSL, gone in 30 seconds: a BREACH beyond CRIME, 2013.

[57] 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.

[58] R. Rivest. The MD4 message digest algorithm. In *Proceedings of Crypto '90*, volume 537 of *LNCS*. Springer-Verlag, 1990.

[59] R. Rivest. The MD5 message digest algorithm. Internet RFC 1321, 1992.

[60] J. Rizzo and T. Duong. The CRIME attack. Presentation at Ekoparty 2012, 2012.

[61] B. Schneier. Real-world passwords. Crypto-gram, Dec. 2006.

[62] 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.

[63] 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.

[64] T. J. Smedinghoff. Online transactions: The rules for ensuring enforceability in a global environment. *The Computer & Internet Lawyer*, pages 6–17, April 2006. `http://www.bakernet.com/ecommerce/7-etran-req.pdf`.

[65] D. Song, D. Wagner, and X. Tian. Timing analysis of keystrokes and timing attacks on SSH. In *10th USENIX Security Symposium*, 2001.

[66] F.-X. Standaert, G. Piret, and J.-J. Quisquater. Cryptanalysis of block ciphers: A survey. *UCL Crypto Group*, 2003.

[67] J. Steiner, C. Neuman, and J. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of USENIX '88*, pages 191–202, 1988.

[68] 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.

[69] B. Vallée. Gauss' algorithm revisited. *J. Algorithms*, 12(4):556–572, 1991.

[70] X. Wang, A. Yao, and F. Yao. New collision search for SHA-1. Rump Session Crypto'05, 2005.

[71] T. Wu. A real-world analysis of kerberos password security. In *In proceedings of NDSS '99*, 1999.