Intercepting iCloud Keychain

Alex Radocea
Longterm Security, Inc.

Alex Radocea

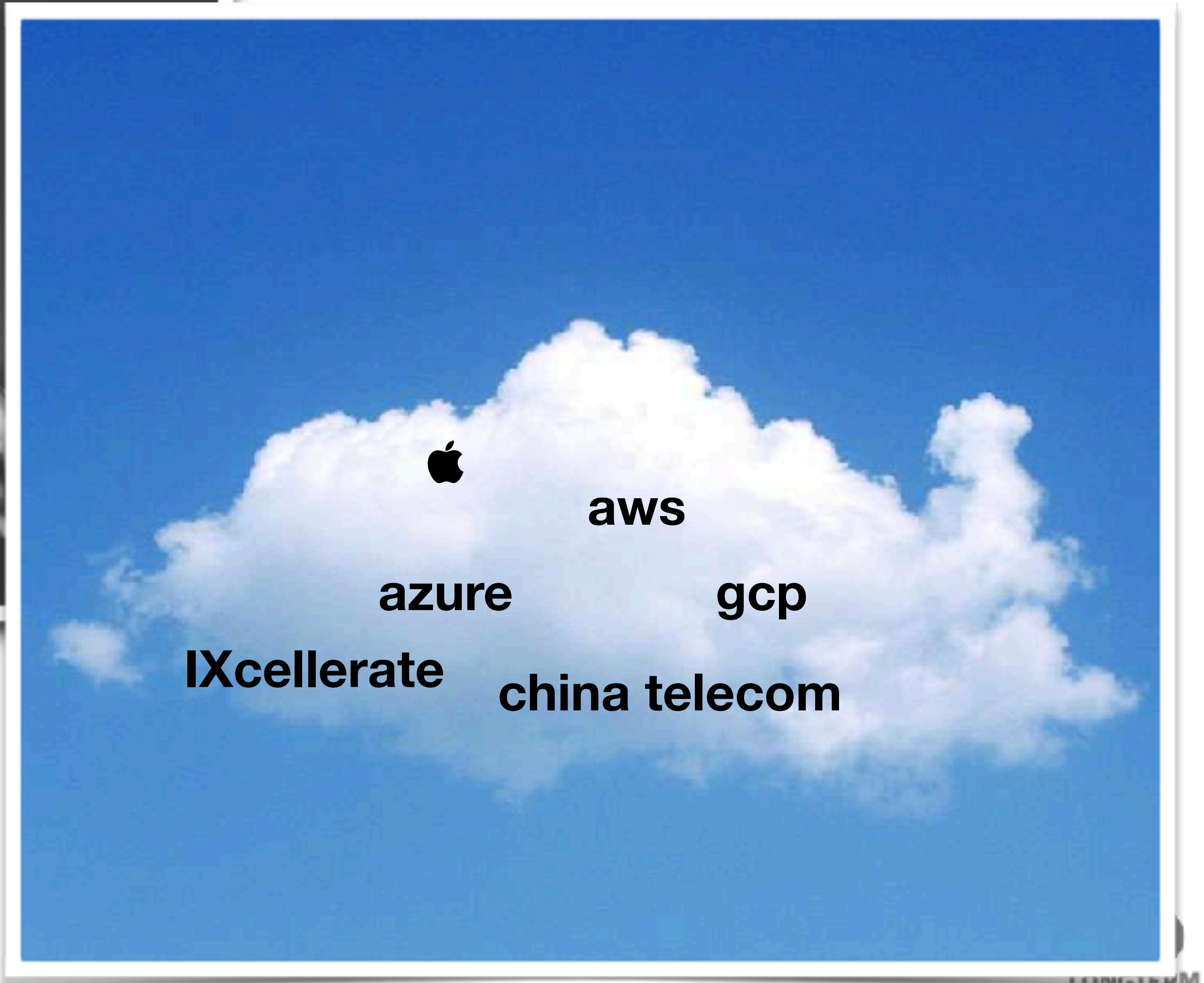
Co-founder at Longterm Security, Inc.

Recovering CTF Addict

contact@longterm.io
@defendtheworld

What is iCloud Keychain?

Secret Syncing & Recovery in the Cloud



Designed to be Highly Secure

- Strong end-to-end cryptography
- Resilient against a compromised backend, rogue insiders
- Resilient when an attacker has obtained a target's Apple ID password
    - Need an additional password or a trusted device

Critical Flaws Now Fixed

- We found critical flaws in undocumented, open-source compoments of the protocol

- Agenda: We'll describe previous work, how iCloud Keychain Syncing works, and the flaws in detail

Prior Work & Presentations Covering iCloud Keychain

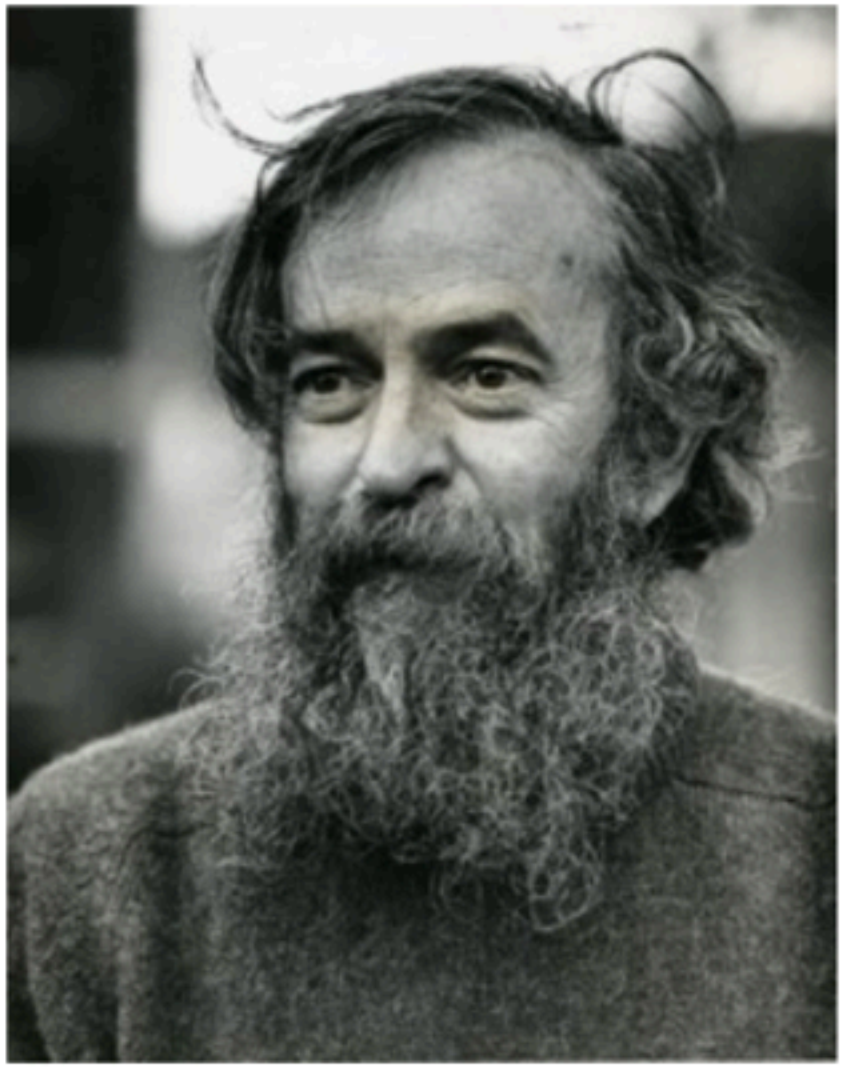
- **Andrey Belenko/ViaForensics** - https://speakerdeck.com/belenko/on-the-security-of-the-icloud-keychain

- **Andrey Belenko/ViaForensics** - CVE-2015-1065 - buffer overflows in keychain sync with MITM capability

- **Ivan Krstic/ -** Behind the Scenes with iOS Security: Secret Synchronization - https://www.blackhat.com/docs/us-16/materials/us-16-Krstic.pdf

- **Vladamir Katalov/Elcomsoft** - https://conference.hitb.org/hitbsecconf2017ams/sessions/commsec-when-two-factor-authentication-is-a-foe-breaking-apples-icloud-keychain/

- **iOS 10 Security Guide** - https://www.apple.com/business/docs/iOS_Security_Guide.pdf



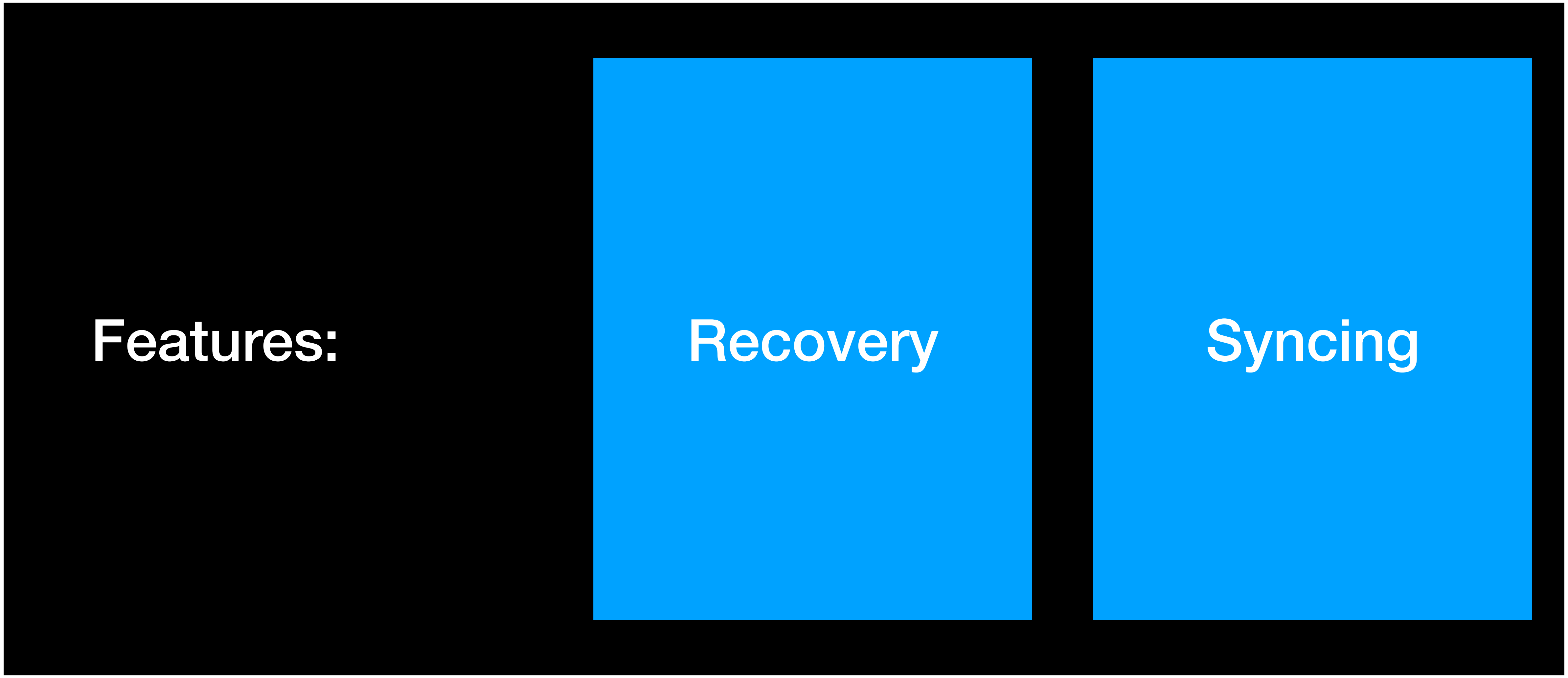
Robert Morris

(Harry Naltchayan/THE WASHINGTON POST)

“Never underestimate the attention, risk, money and time that an opponent will put into reading traffic.”

LONGTERM

iCloud Keychain Components



Features:

Recovery

Syncing

Features:

Recovery

Syncing

HSM-Based
Escrow
System

Features:

Recovery

Syncing

HSM-Based Escrow System

SMS Verification

Features: Recovery Syncing

HSM-Based Escrow System

SMS Verification

iCloud Security Code (iCSC) or Device Passcode required

Features:

Recovery

Syncing

HSM-Based Escrow System

SMS Verification

iCloud Security Code (iCSC) or Device Passcode required

Secure Remote Protocol (SRP) Code Verification

Features:

Recovery

Syncing

Sync'd Secrets Across All Trusted Devices

Features:

Recovery

Syncing

Sync'd Secrets Across All Trusted Devices

End to End Encryption

Features:

Recovery

Syncing

Sync'd Secrets Across All Trusted Devices

End to End Encryption

Circle of Trust

Features:

Recovery

Syncing

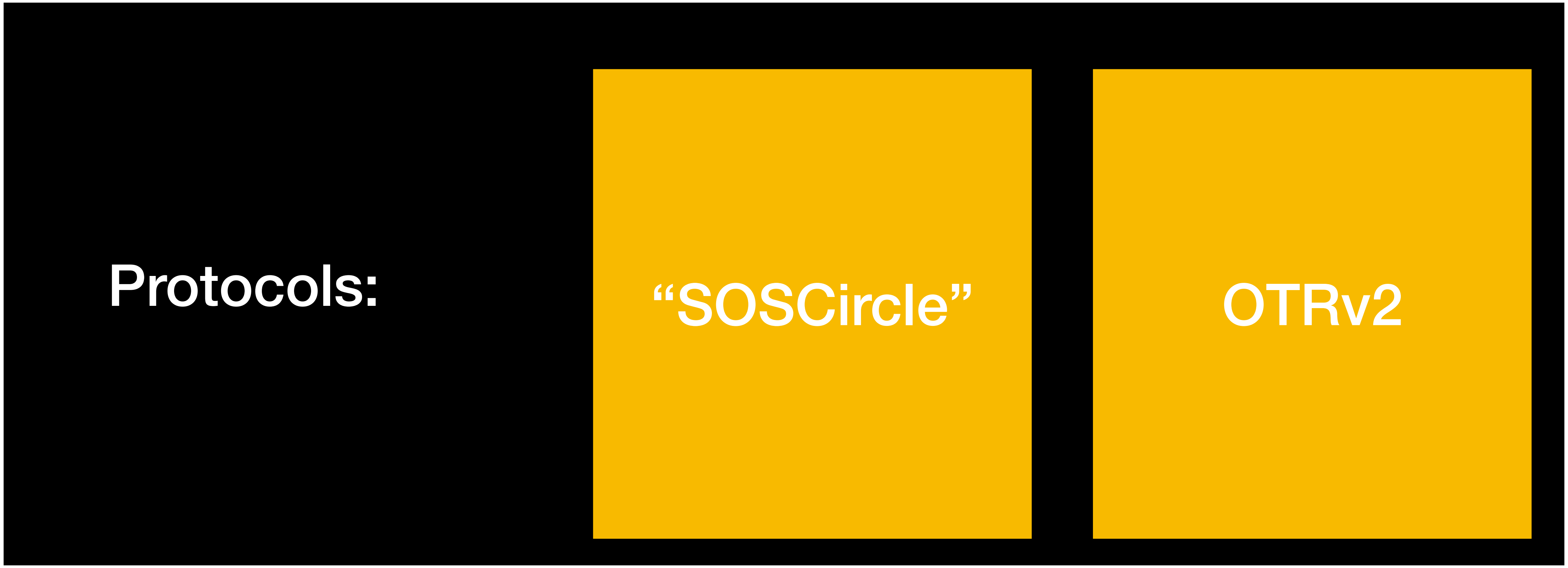
Sync’d Secrets Across All Trusted Devices

End to End Encryption

Circle of Trust

Approval or Two-Factor/ iCSC required to join

iCloud Keychain Sync



Protocols:

“SOSCircle”

OTRv2

Protocols: "SOSCircle" OTRv2

Signed Syncing Circle Establishes Trusted Devices

Join Circle with Apple ID Password and Trusted Device Approval

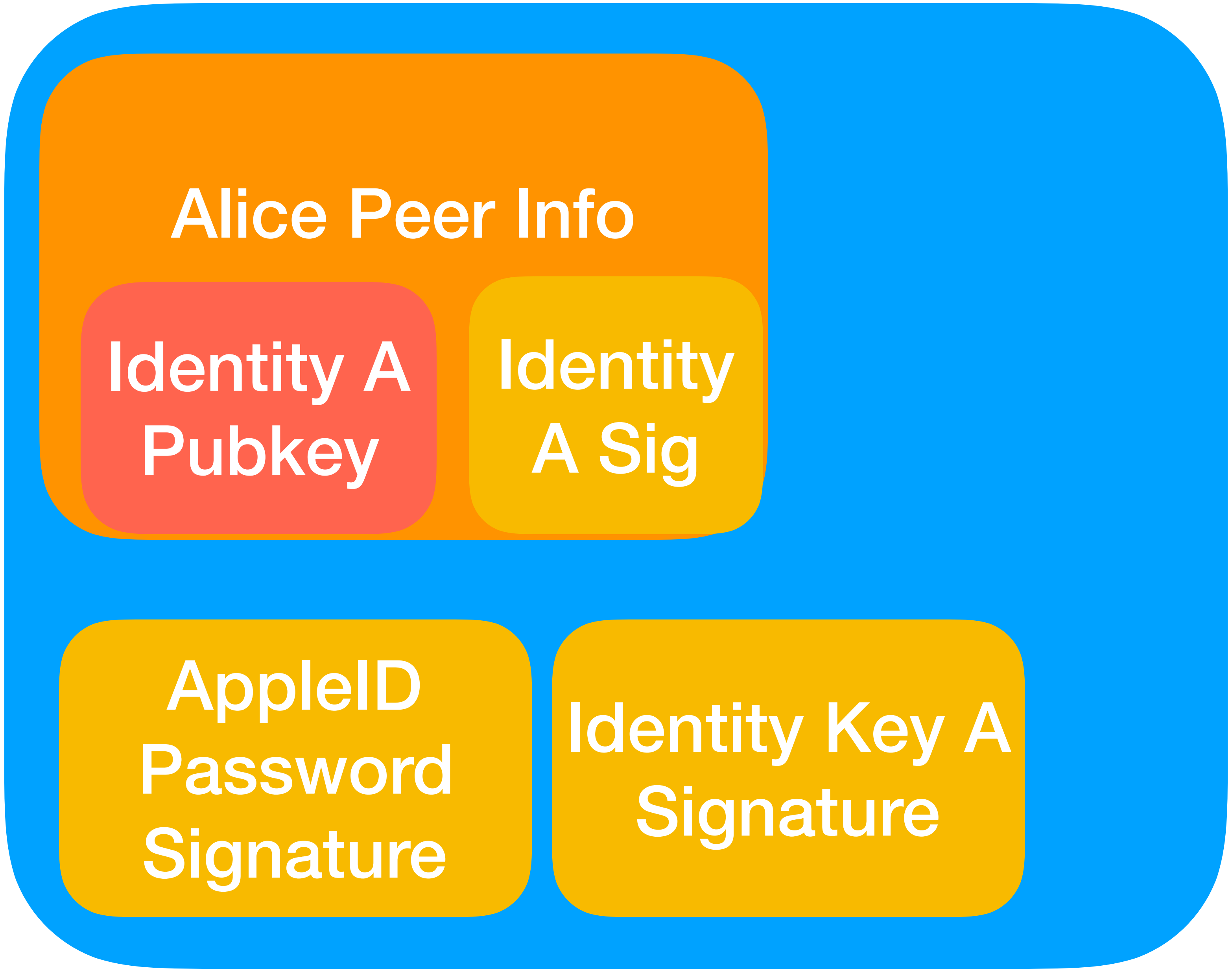
Join Circle with Apple ID Password and iCSC/Device Passcode

256-bit ECDSA on secp256r1 with SHA256

LONGTERM

Circle Protocol Illustrated

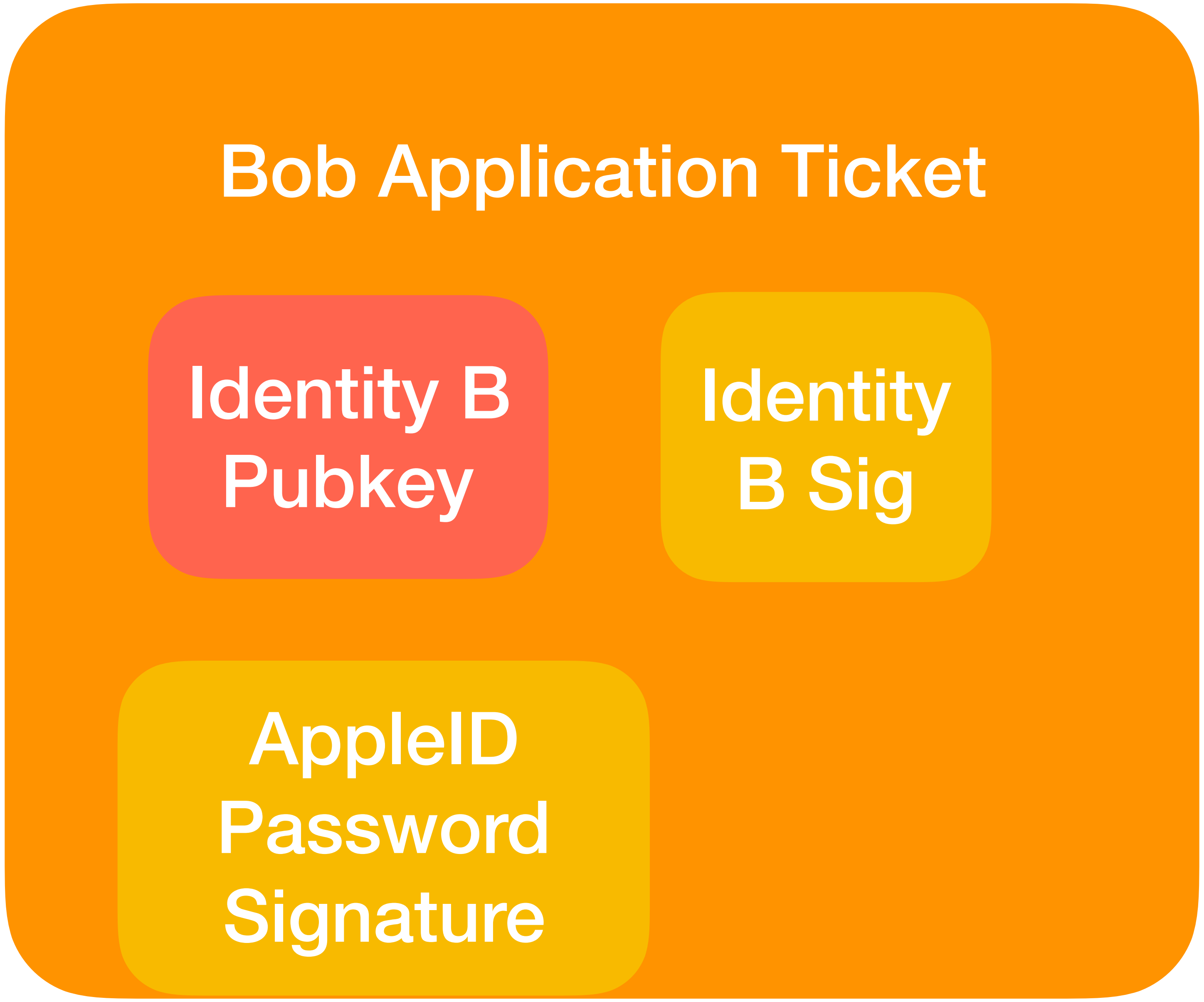
1. **A** creates a Circle



LONGTERM

Circle Protocol Illustrated

2. **B** requests to join



Bob Application Ticket

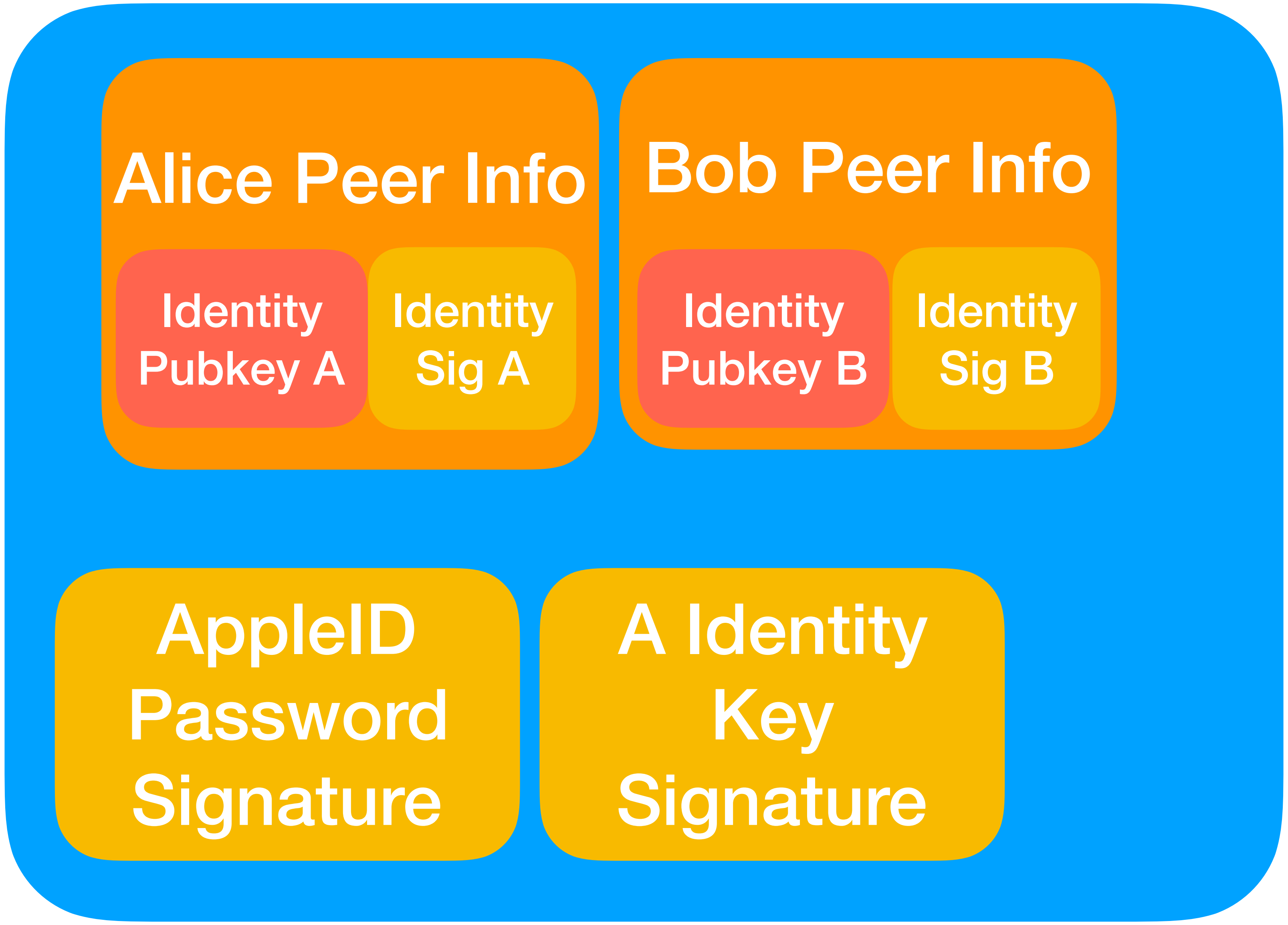
Identity B Pubkey

Identity B Sig

AppleID Password Signature

Circle Protocol Illustrated

3. **A** approves



Alice Peer Info

Identity Pubkey A

Identity Sig A

Bob Peer Info

Identity Pubkey B

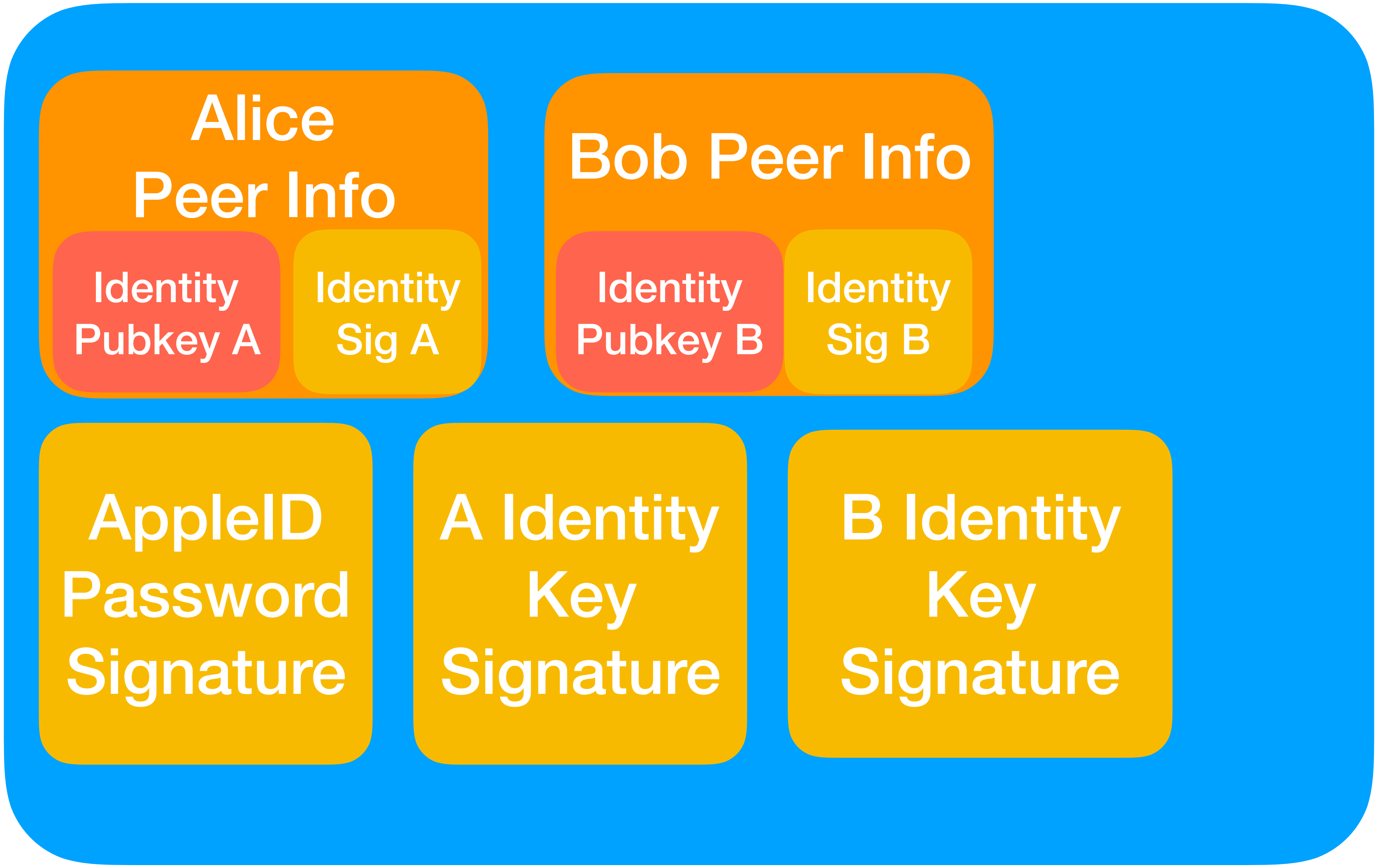
Identity Sig B

AppleID Password Signature

A Identity Key Signature

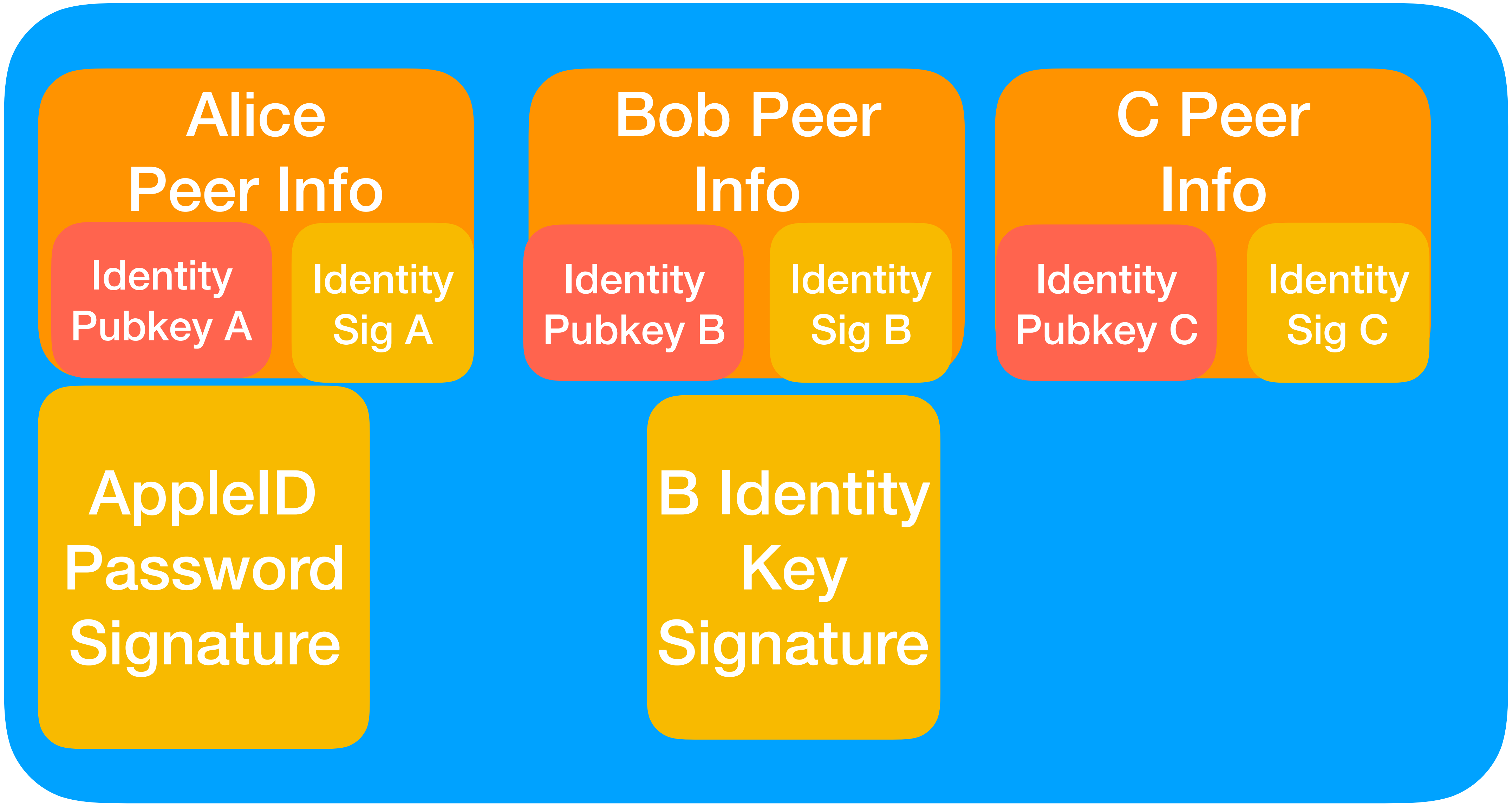
Circle Protocol Illustrated

4. **B** countersigns



Circle Protocol Illustrated

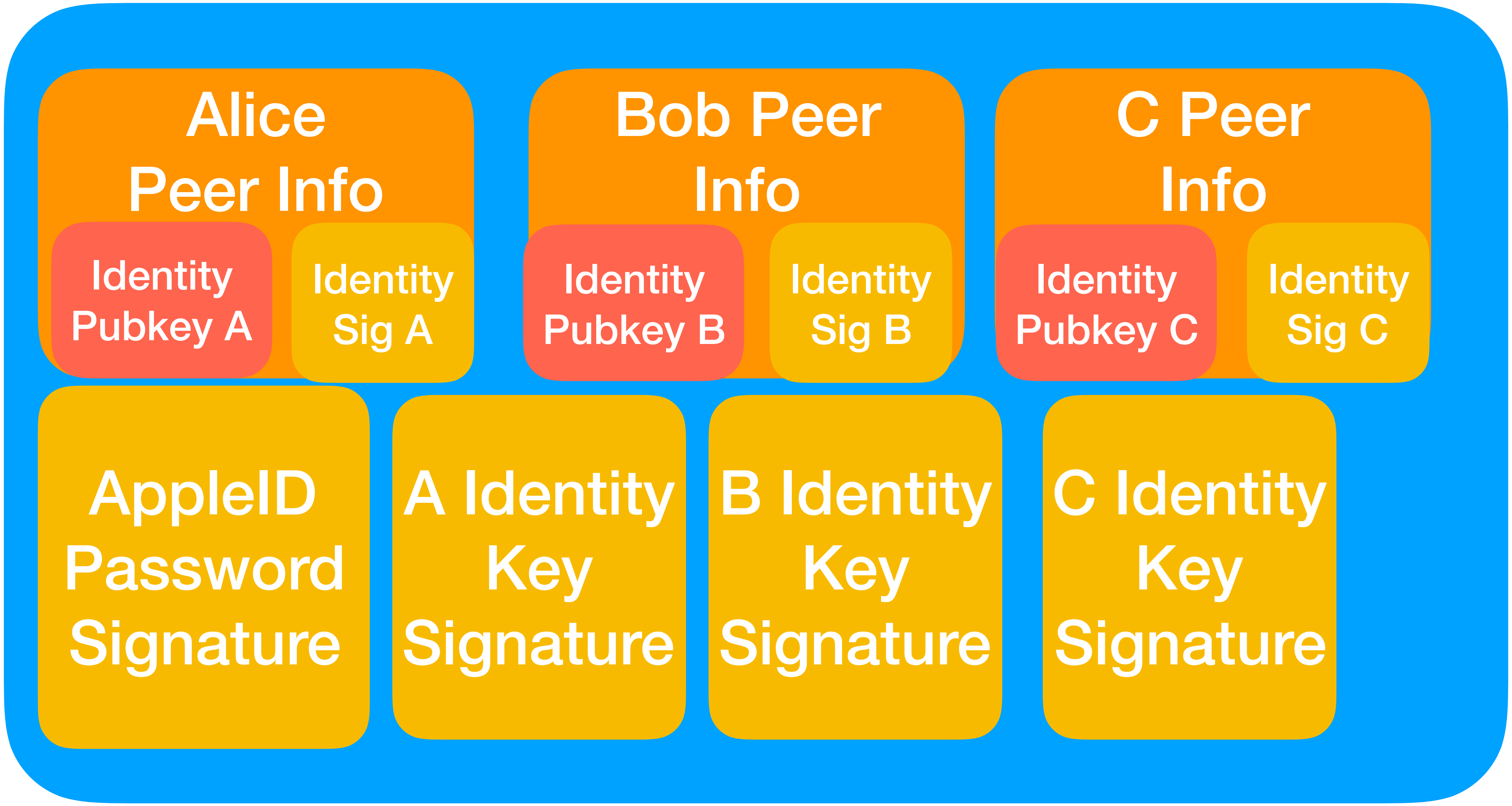
5. Third device approved by **B**…



LONGTERM

Circle Protocol Illustrated

6. Countersigned by all parties



Alice Peer Info

Identity Pubkey A

Identity Sig A

Bob Peer Info

Identity Pubkey B

Identity Sig B

C Peer Info

Identity Pubkey C

Identity Sig C

AppleID Password Signature

A Identity Key Signature

B Identity Key Signature

C Identity Key Signature

What happens when devices are lost while traveling?

iCloud Keychain Passwords Overview

	Join Circle With Approving Device	Join Circle Without Approving Device
Default	Apple ID Password	Apple ID Password + SMS + iCloud Security Code (iCSC)
Two-Factor Enabled Account	Apple ID Password + 2FA Approval	Apple ID Password + SMS + Device Passcode

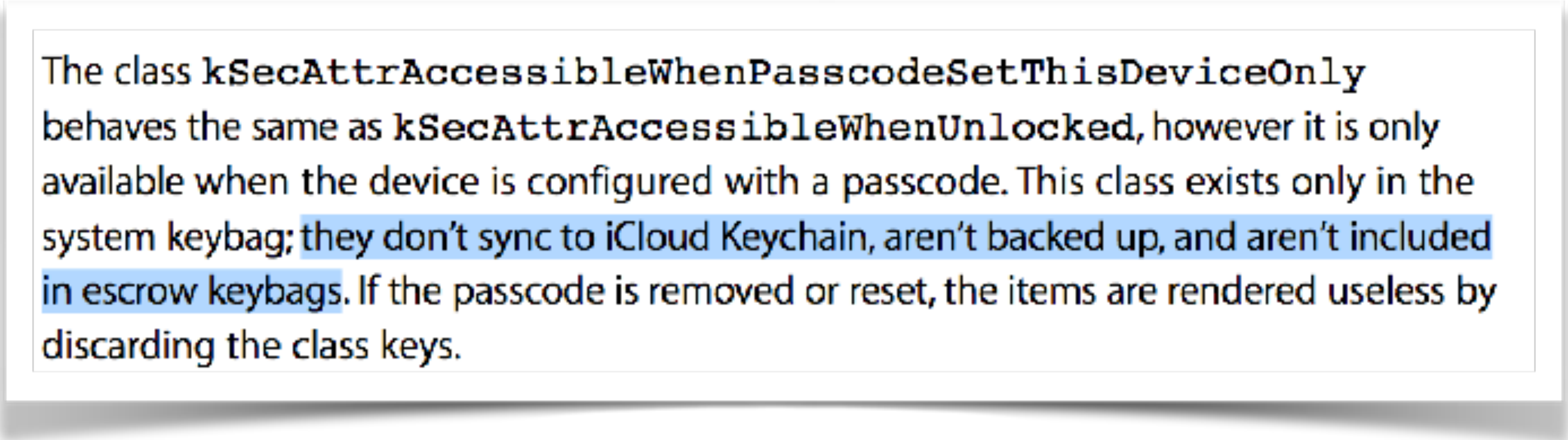
LONGTERM

How Does A New Device Join Without Approval?

- Circle does not reset when this happens

- Joining the circle requires a trusted device to sign the updated circle with an identity key…

- And Identity Keys not in the escrow **kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly**

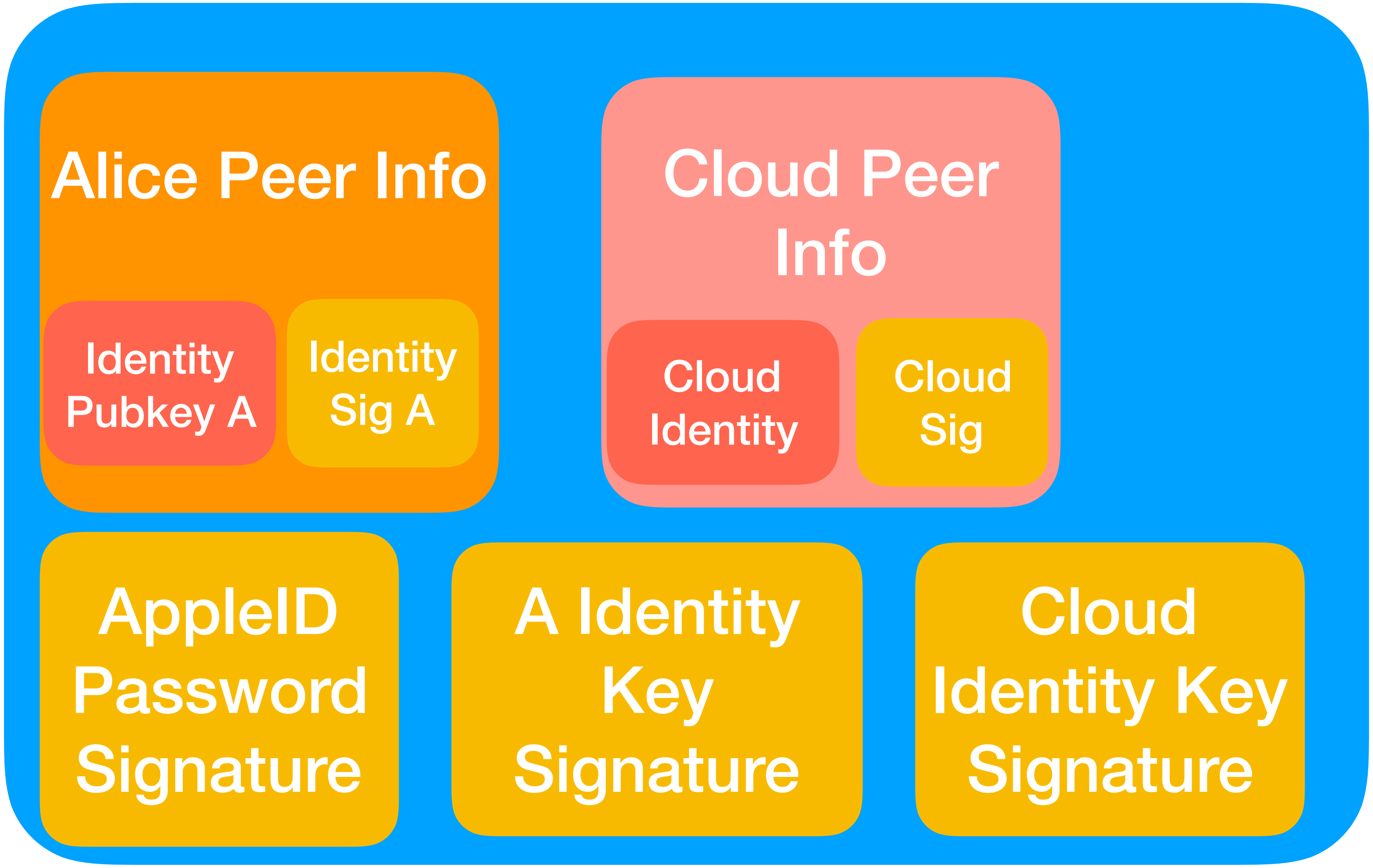


The class `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` behaves the same as `kSecAttrAccessibleWhenUnlocked`, however it is only available when the device is configured with a passcode. This class exists only in the system keybag; they don't sync to iCloud Keychain, aren't backed up, and aren't included in escrow keybags. If the passcode is removed or reset, the items are rendered useless by discarding the class keys.

Uncovering a hidden peer

- Undocumented, speculating this is for streamlining usability
- When a Circle is first established an “iCloud Identity” Key is also created as a “hidden” peer
    - Key is created with **kSecAttrAccessibleWhenUnlocked**, **kSecAttrSynchronizable**
- Available from iCloud Keychain Recovery
- Can be used to update the Syncing Circle, and trigger automatic coutersigning from all peers

Updated Circle Illustration - One Peer



Alice Peer Info

Identity Pubkey A

Identity Sig A

Cloud Peer Info

Cloud Identity

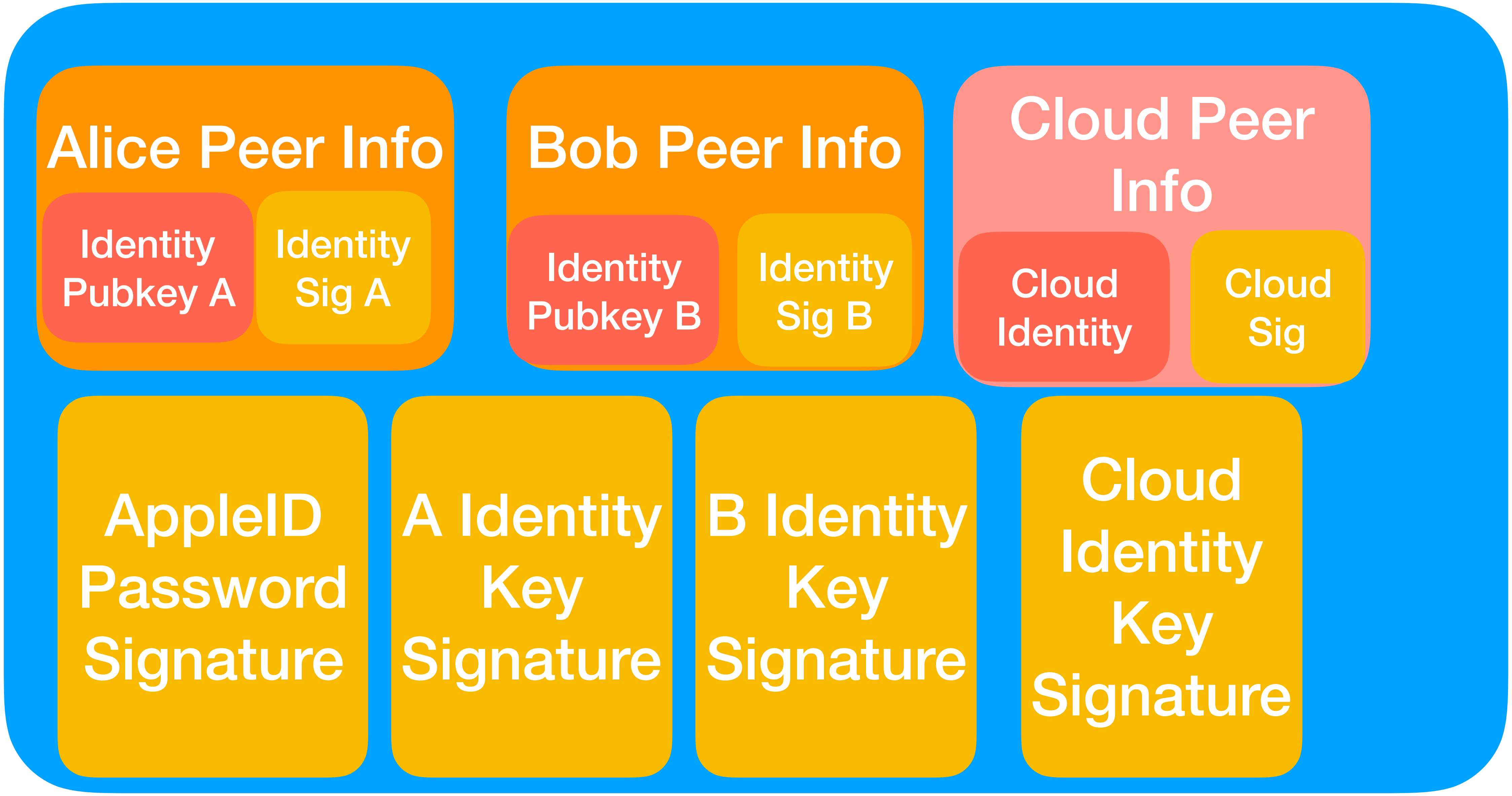
Cloud Sig

AppleID Password Signature

A Identity Key Signature

Cloud Identity Key Signature

Updated Circle Illustration - Two Peers



Alice Peer Info

Identity Pubkey A

Identity Sig A

Bob Peer Info

Identity Pubkey B

Identity Sig B

Cloud Peer Info

Cloud Identity

Cloud Sig

AppleID Password Signature

A Identity Key Signature

B Identity Key Signature

Cloud Identity Key Signature

Which Backups Contain the Cloud Identity Key?

- Cloud Peer Backup sounds tricky, seems okay

- If available in iCloud Backup Keybags…

    - UID Key wrapping prevents Apple/Malicious Insider from accessing the data

- iCloud Keychain Escrow contains Cloud Identity Keys (**kSecAttrSynchronizable)**

    - Not available without SMS and either iCSC or passcode with two-factor authentication

Protocols:

“SOSCircle”

OTRv2

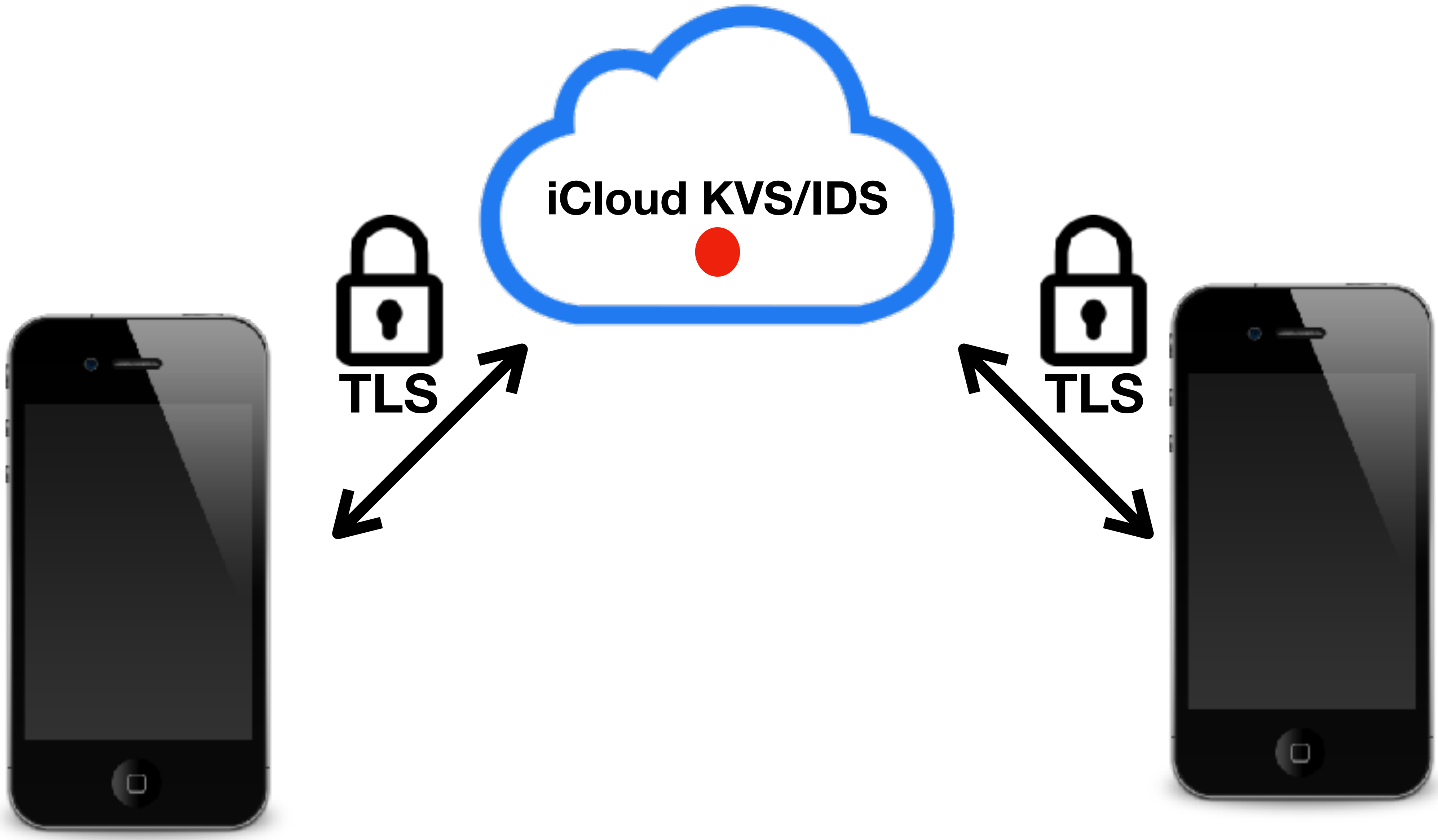
End to End Encryption

Forward Secrecy & Deniability

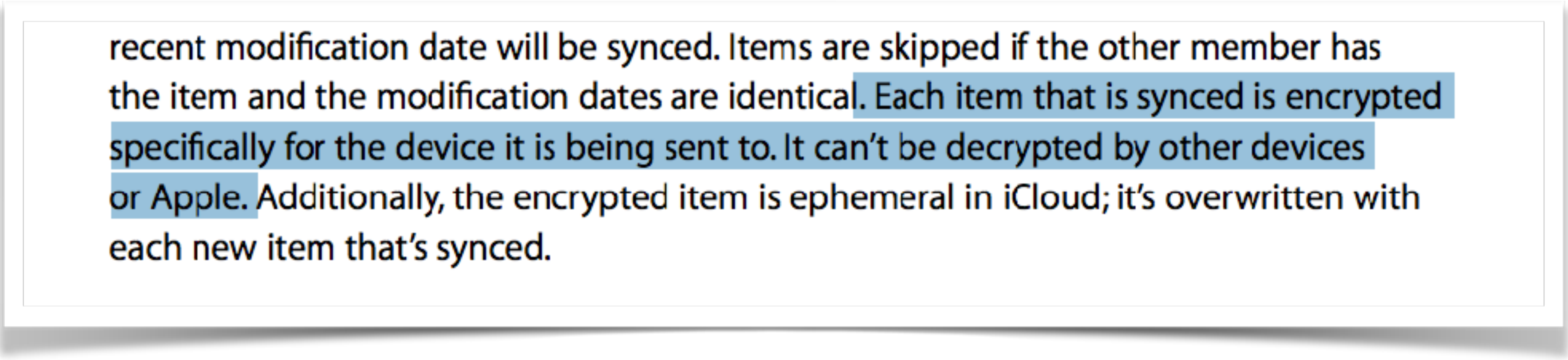
ECDH Key Exchange, Verified with Peer Identity Keys

128-AES-CTR Encryption w/ Rotating Keys

iCloud Keychain Sync Transmits Data Across Apple Services

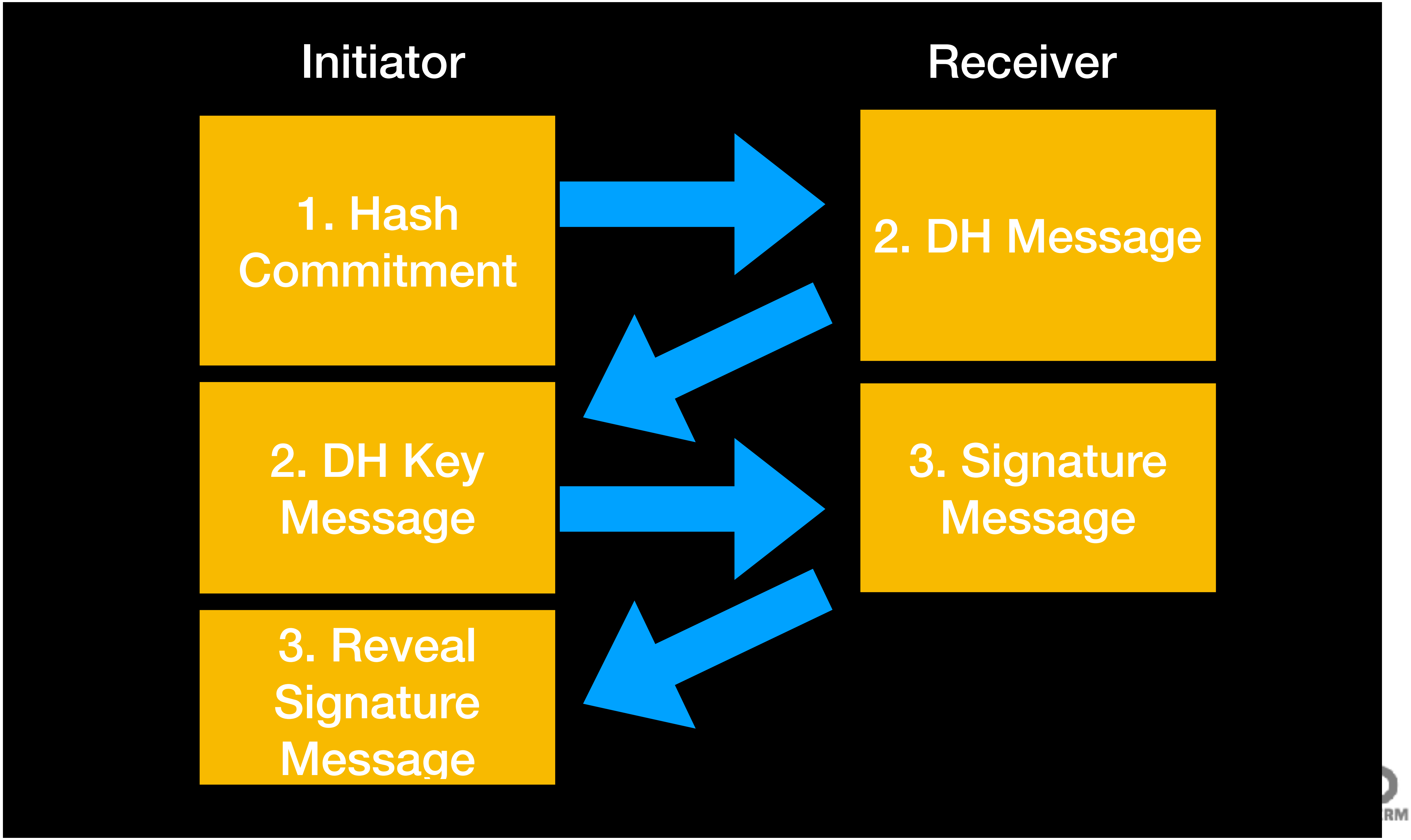


E2E: Plaintext material only available on trusted devices



recent modification date will be synced. Items are skipped if the other member has the item and the modification dates are identical. Each item that is synced is encrypted specifically for the device it is being sent to. It can't be decrypted by other devices or Apple. Additionally, the encrypted item is ephemeral in iCloud; it's overwritten with each new item that's synced.

OTR KEX Messages



Initiator

Receiver

1. Hash Commitment

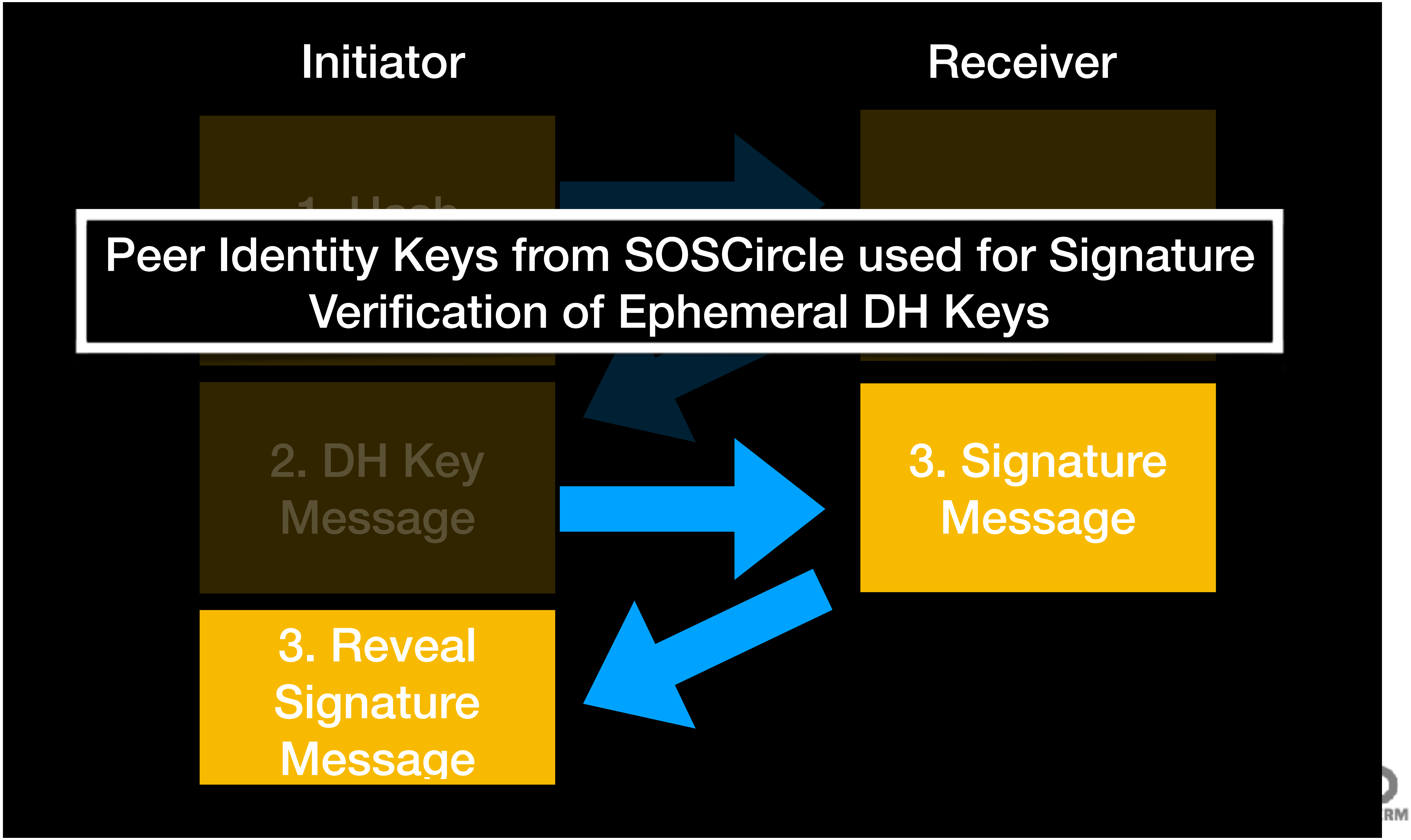
2. DH Message

2. DH Key Message

3. Signature Message

3. Reveal Signature Message

OTR KEX Messages



Initiator

Receiver

1. Hash

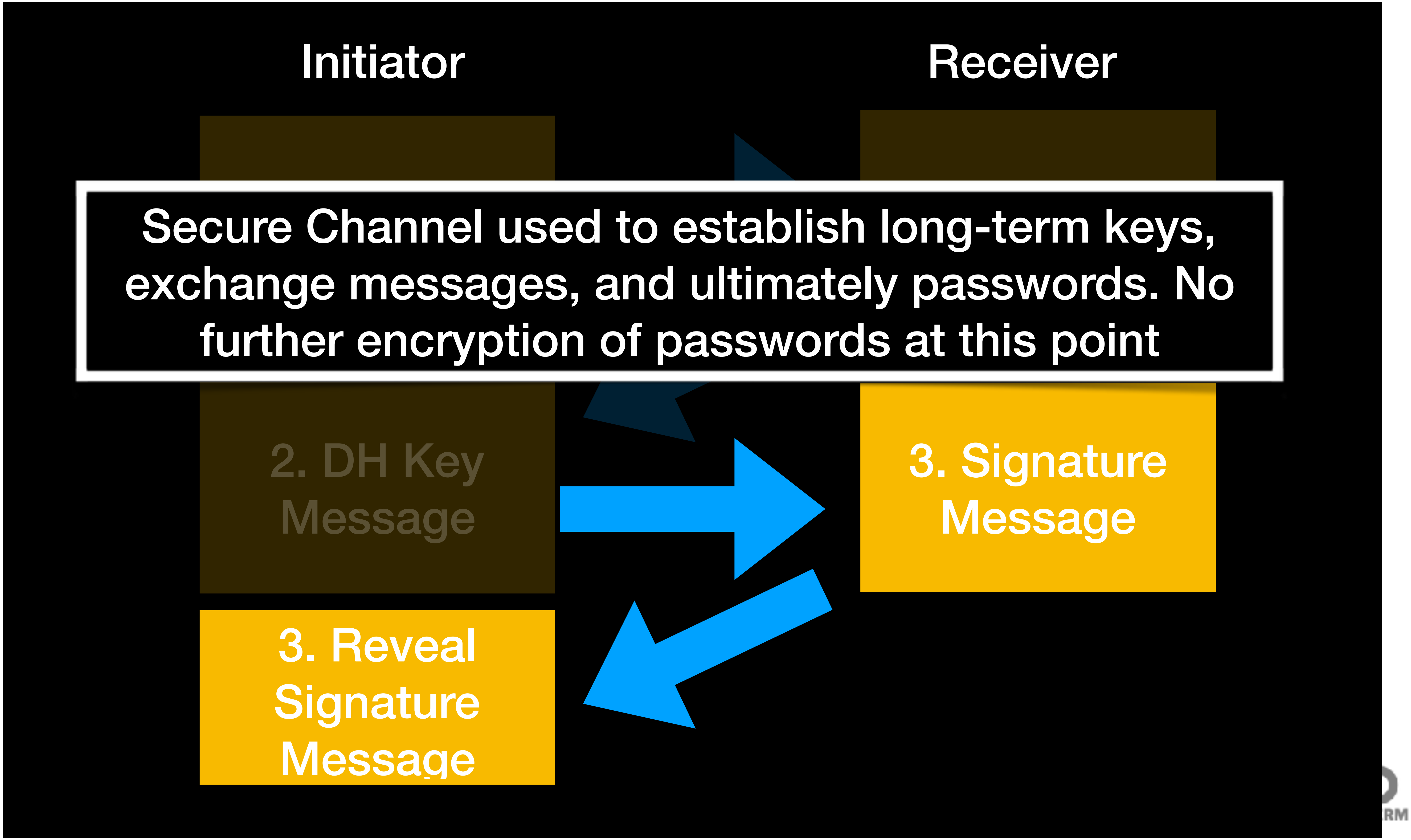
Peer Identity Keys from SOSCircle used for Signature Verification of Ephemeral DH Keys

2. DH Key Message

3. Signature Message

3. Reveal Signature Message

OTR KEX Messages



Initiator

Receiver

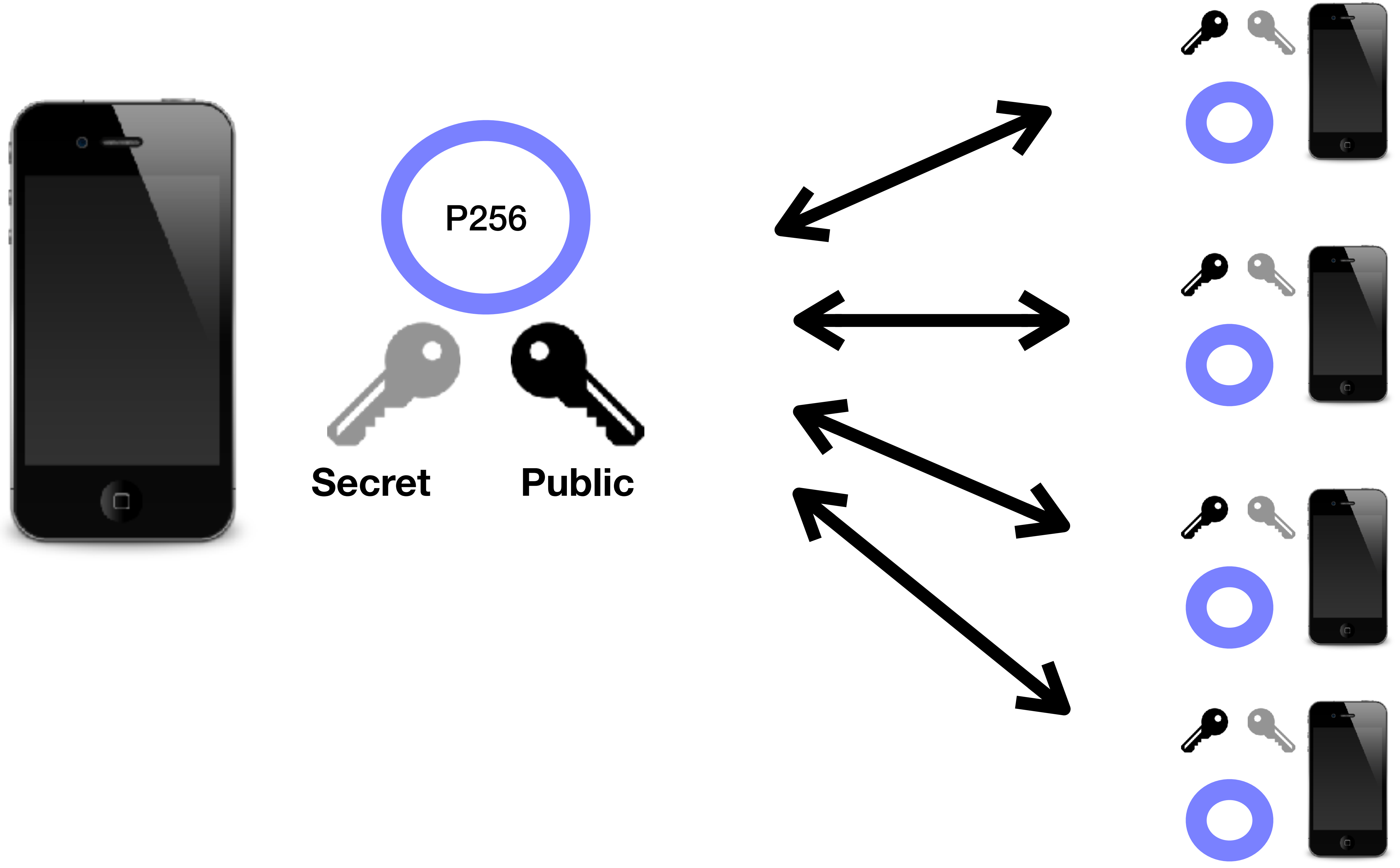
Secure Channel used to establish long-term keys, exchange messages, and ultimately passwords. No further encryption of passwords at this point

2. DH Key Message

3. Signature Message

3. Reveal Signature Message

Pairwise, Fanout Negotiation

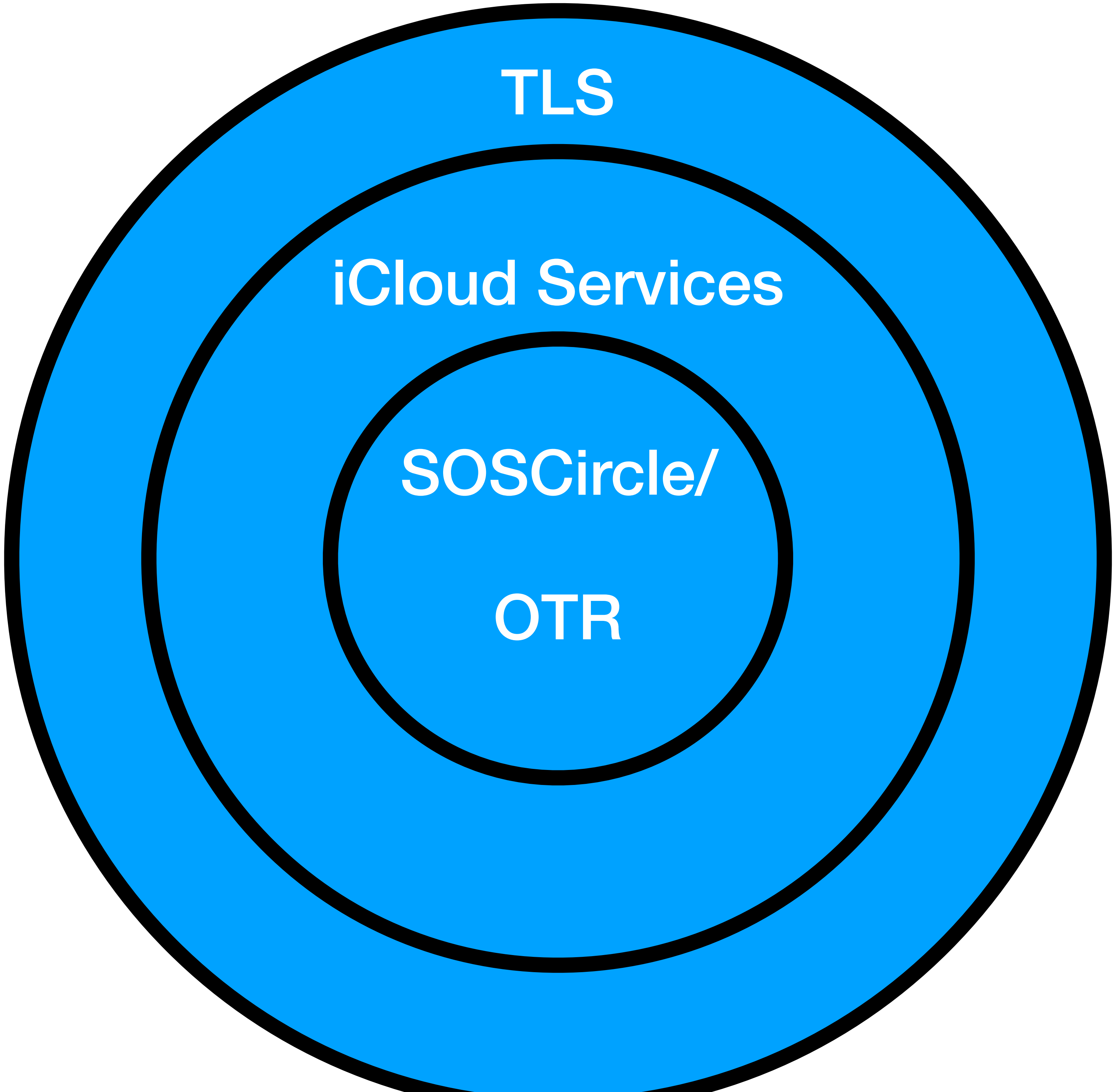


Apple’s iCloud Keychain Security Goals

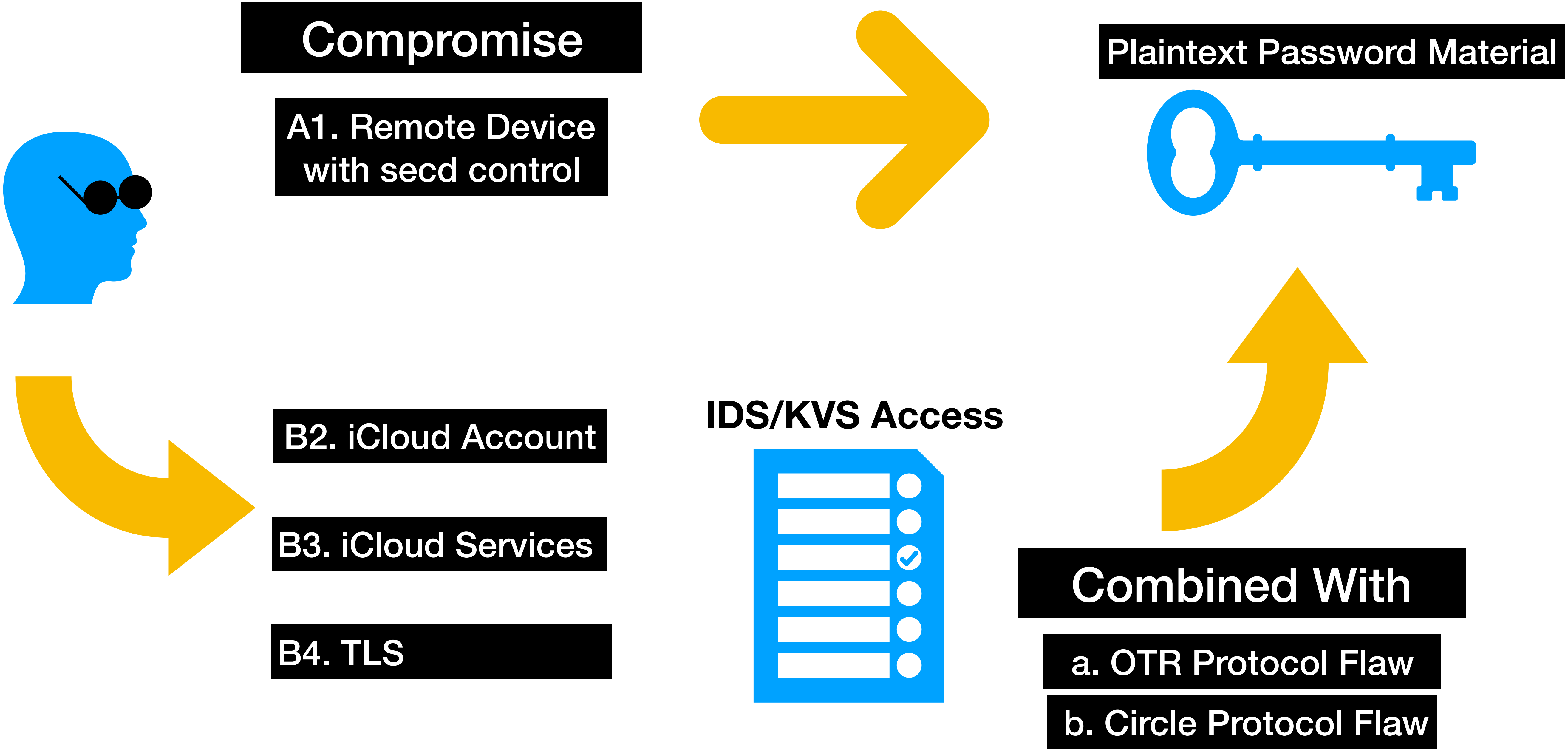
- “Sync passwords between iOS devices and Mac computers without exposing that information to Apple”
- Also protect password material:
    - When the iCloud account is compromised
    - When iCloud is compromised by a rogue insider or external attackers
    - When third parties access user accounts

LONGTERM

iCloud Keychain Sync Security Layers



iCloud Keychain Sync Remote Attack Graph



Compromise

A1. Remote Device with secd control

Plaintext Password Material

B2. iCloud Account

B3. iCloud Services

B4. TLS

IDS/KVS Access

Combined With

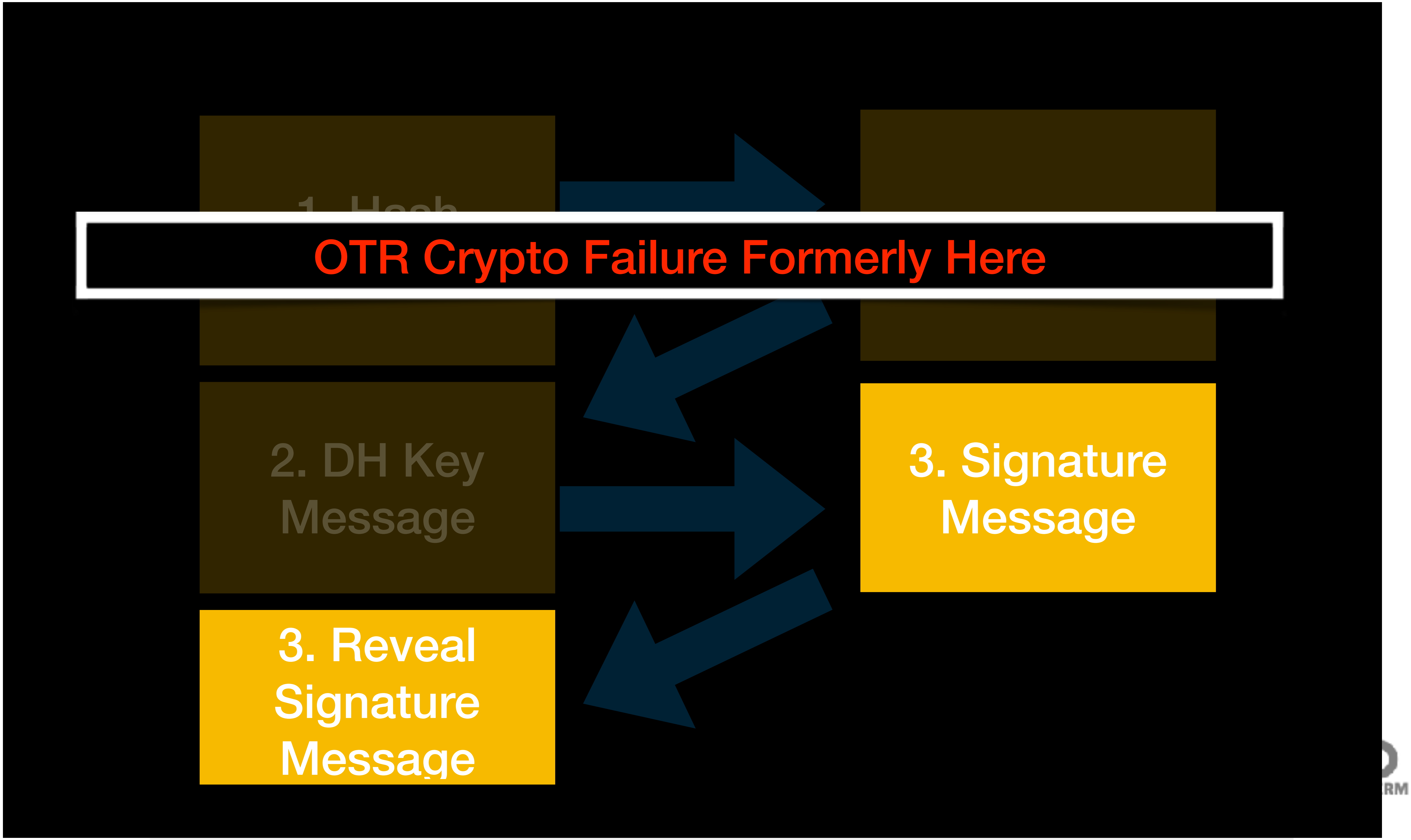
a. OTR Protocol Flaw

b. Circle Protocol Flaw

OTR Flaws

- CVE-2017-2448 - OTR Cryptographic Failure
- CVE-2017-2451 - OTR Memory Corruption
- Exacerbated by lack of TLS key pinning on KVS communications

CVE-2017-2448 - Goto Fail Redux



1. Hash

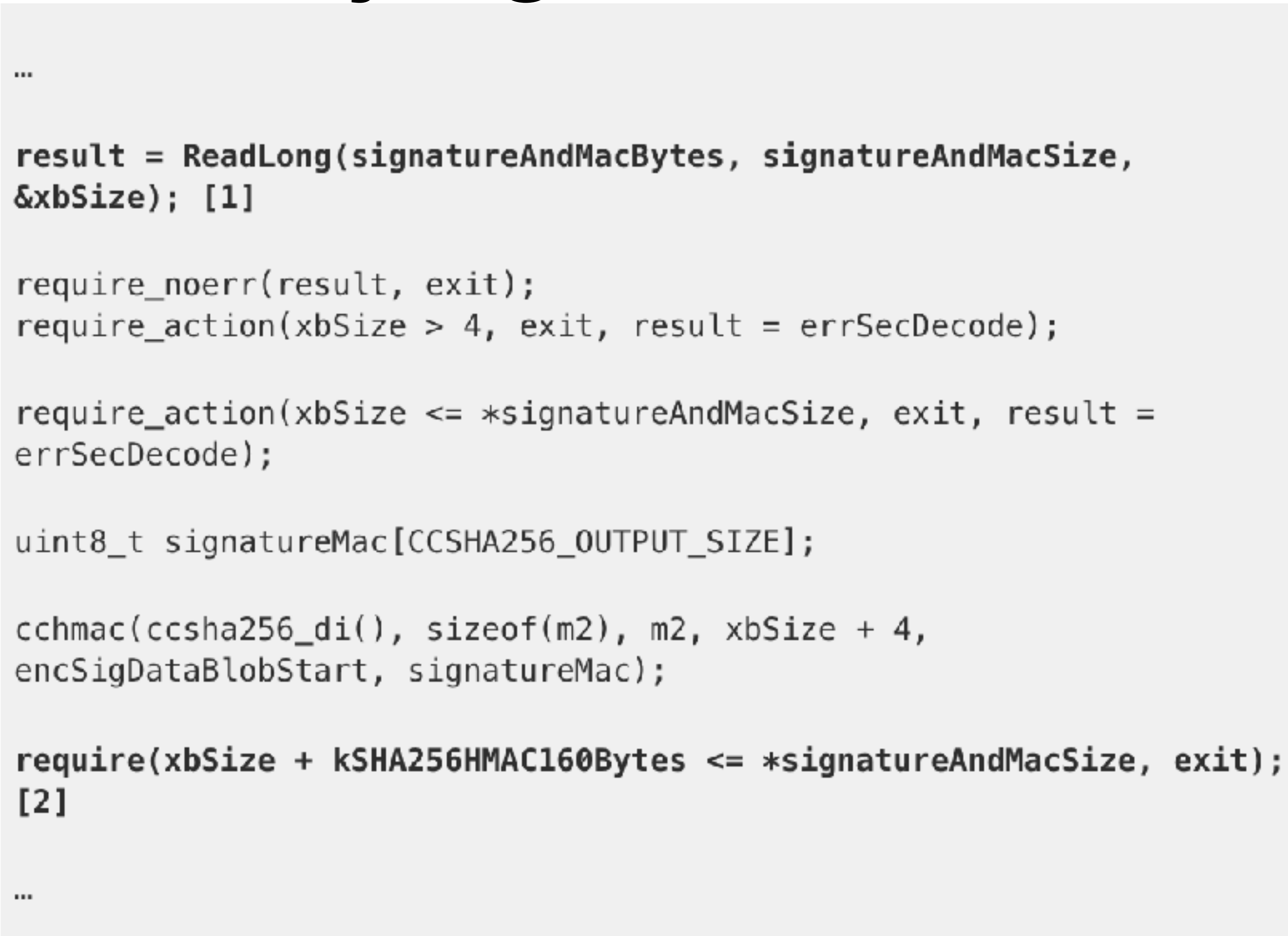
OTR Crypto Failure Formerly Here

2. DH Key Message

3. Signature Message

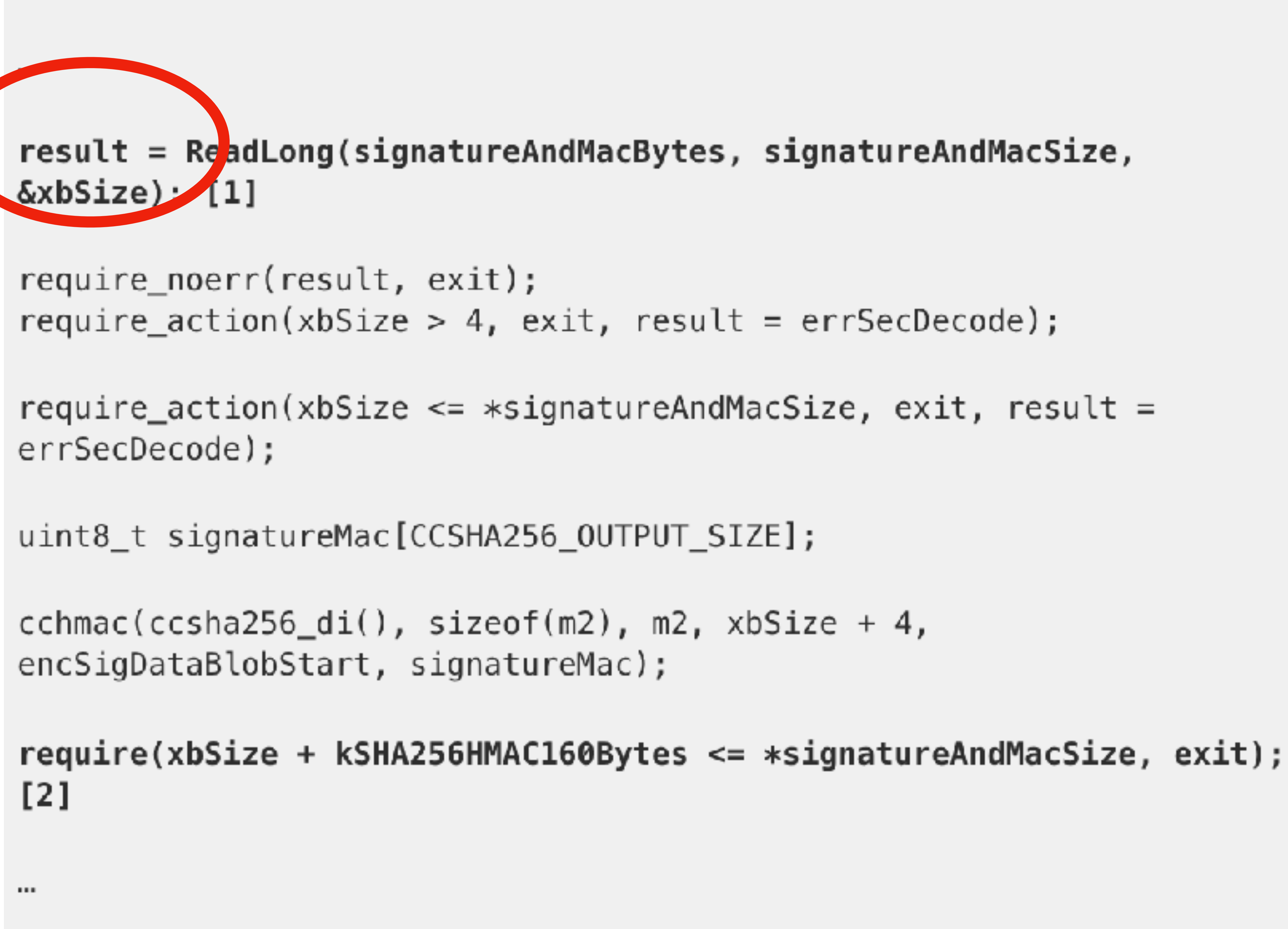
3. Reveal Signature Message

CVE-2017-2448 - SecVerifySignatureAndMac



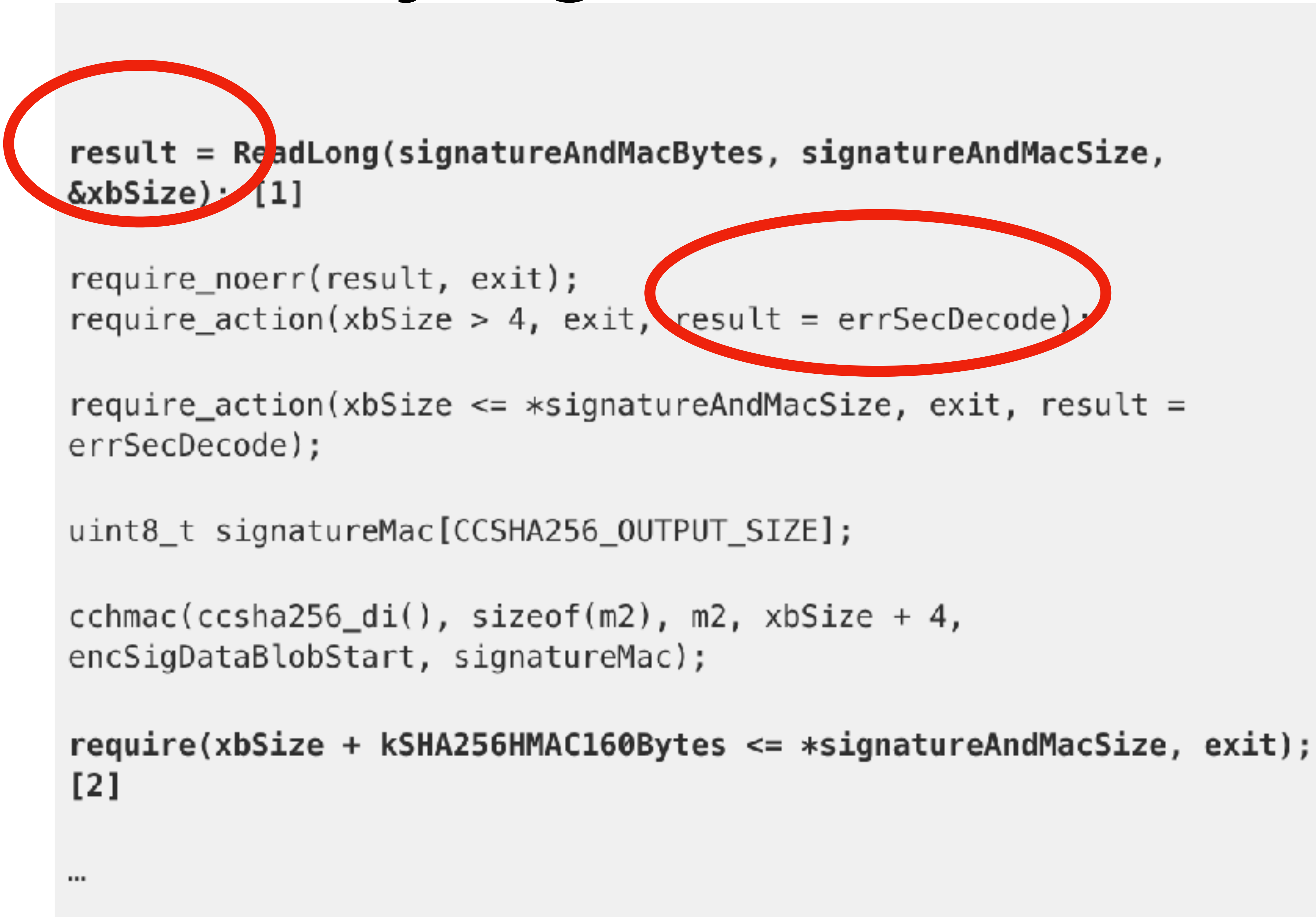
```
…

result = ReadLong(signatureAndMacBytes, signatureAndMacSize,
&xbSize); [1]

require_noerr(result, exit);
require_action(xbSize > 4, exit, result = errSecDecode);

require_action(xbSize <= *signatureAndMacSize, exit, result =
errSecDecode);

uint8_t signatureMac[CCSHA256_OUTPUT_SIZE];

cchmac(ccsha256_di(), sizeof(m2), m2, xbSize + 4,
encSigDataBlobStart, signatureMac);

require(xbSize + kSHA256HMAC160Bytes <= *signatureAndMacSize, exit);
[2]

…
```

CVE-2017-2448 - SecVerifySignatureAndMac



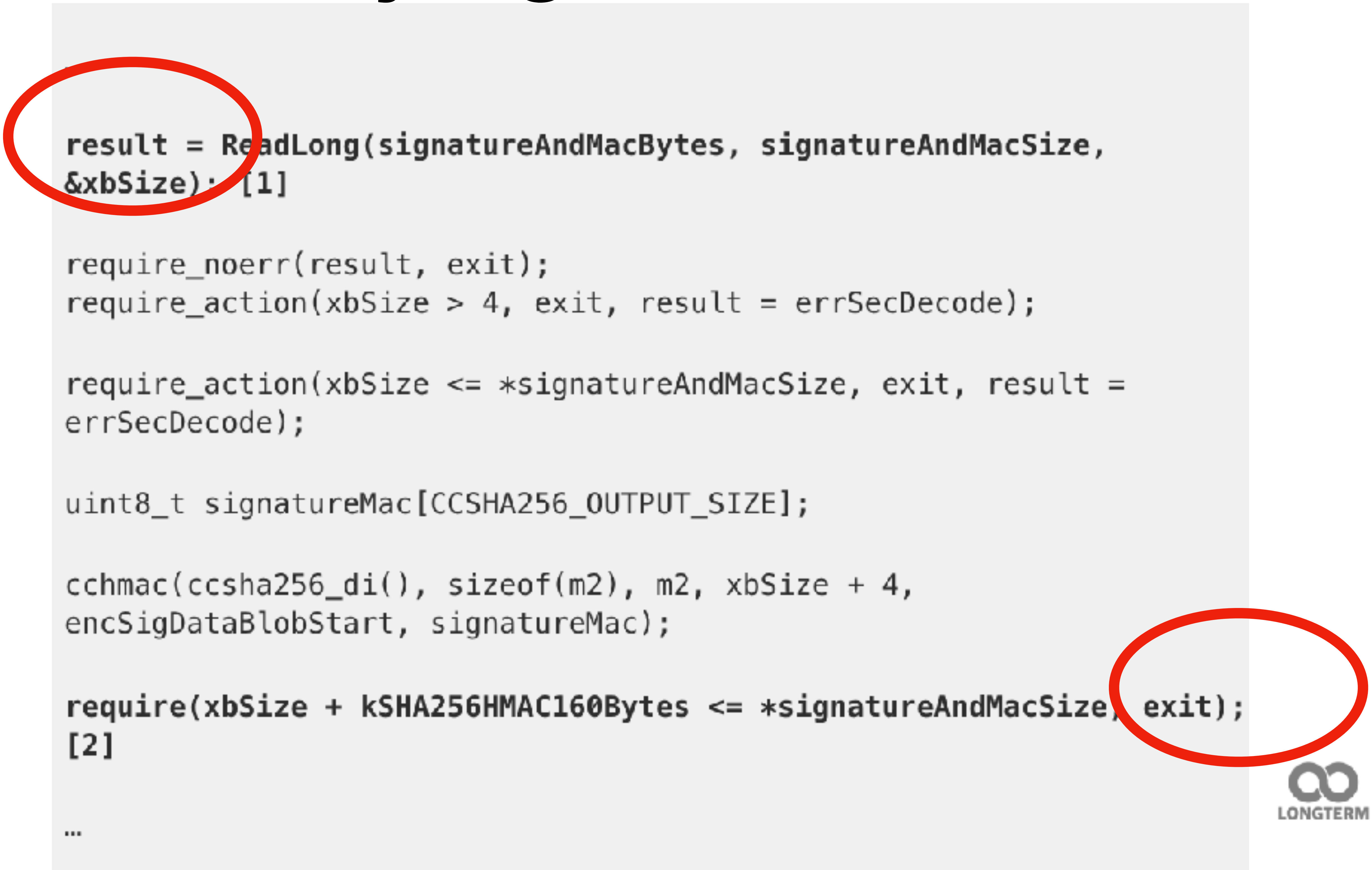
```
…

result = ReadLong(signatureAndMacBytes, signatureAndMacSize,
&xbSize); [1]

require_noerr(result, exit);
require_action(xbSize > 4, exit, result = errSecDecode);

require_action(xbSize <= *signatureAndMacSize, exit, result =
errSecDecode);

uint8_t signatureMac[CCSHA256_OUTPUT_SIZE];

cchmac(ccsha256_di(), sizeof(m2), m2, xbSize + 4,
encSigDataBlobStart, signatureMac);

require(xbSize + kSHA256HMAC160Bytes <= *signatureAndMacSize, exit);
[2]

…
```

CVE-2017-2448 - SecVerifySignatureAndMac



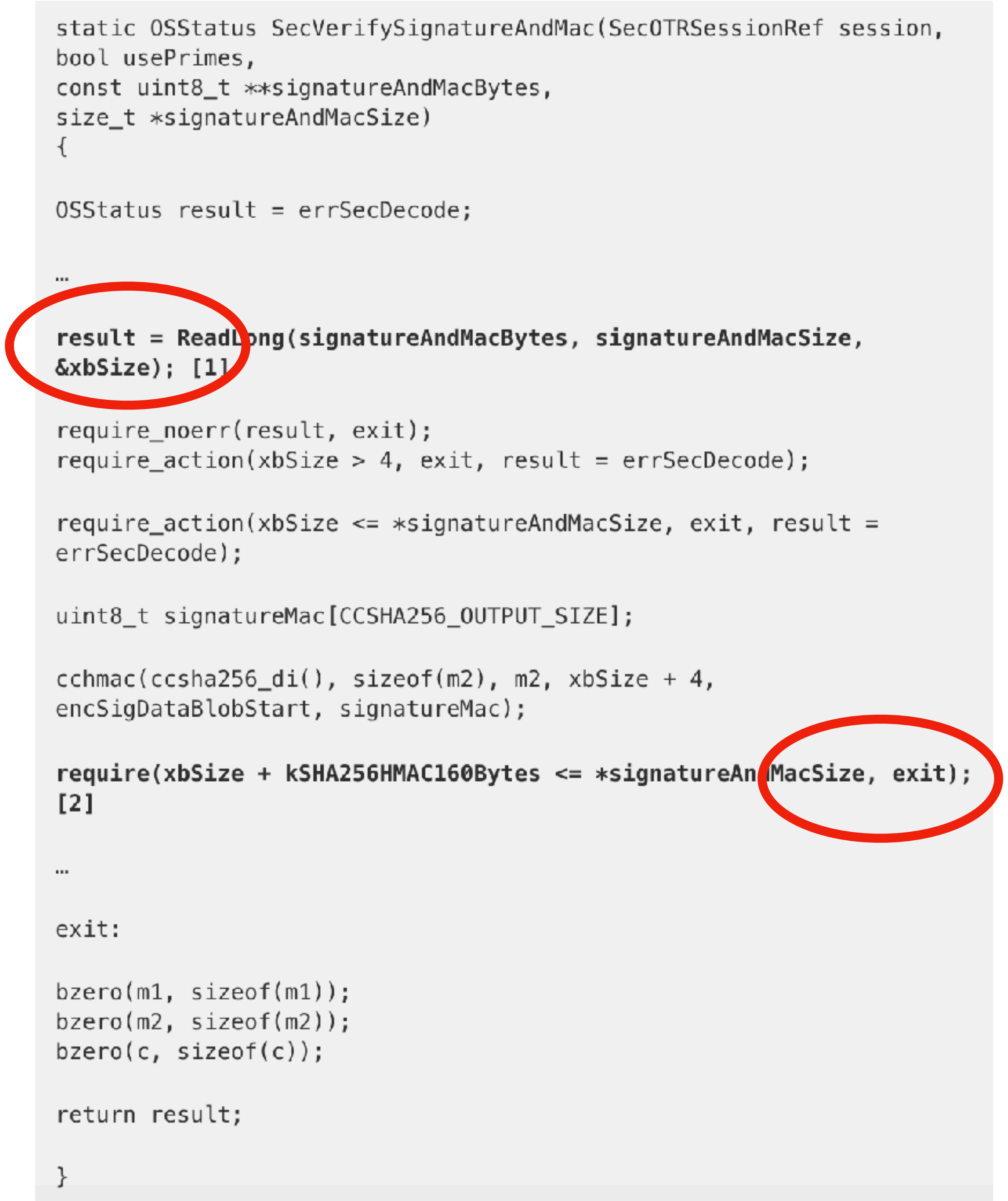
```
result = ReadLong(signatureAndMacBytes, signatureAndMacSize,
&xbSize); [1]

require_noerr(result, exit);
require_action(xbSize > 4, exit, result = errSecDecode);

require_action(xbSize <= *signatureAndMacSize, exit, result =
errSecDecode);

uint8_t signatureMac[CCSHA256_OUTPUT_SIZE];

cchmac(ccsha256_di(), sizeof(m2), m2, xbSize + 4,
encSigDataBlobStart, signatureMac);

require(xbSize + kSHA256HMAC160Bytes <= *signatureAndMacSize, exit);
[2]

…
```

CVE-2017-2448 - SecVerifySignatureAndMac



```
result = ReadLong(signatureAndMacBytes, signatureAndMacSize,
&xbSize); [1]

require_noerr(result, exit);
require_action(xbSize > 4, exit, result = errSecDecode);

require_action(xbSize <= *signatureAndMacSize, exit, result =
errSecDecode);

uint8_t signatureMac[CCSHA256_OUTPUT_SIZE];

cchmac(ccsha256_di(), sizeof(m2), m2, xbSize + 4,
encSigDataBlobStart, signatureMac);

require(xbSize + kSHA256HMAC160Bytes <= *signatureAndMacSize, exit);
[2]

…
```

CVE-2017-2448 - Goto Fail Redux



```
static OSStatus SecVerifySignatureAndMac(SecOTRSessionRef session,
bool usePrimes,
const uint8_t **signatureAndMacBytes,
size_t *signatureAndMacSize)
{

OSStatus result = errSecDecode;

…

result = ReadLong(signatureAndMacBytes, signatureAndMacSize,
&xbSize); [1]

require_noerr(result, exit);
require_action(xbSize > 4, exit, result = errSecDecode);

require_action(xbSize <= *signatureAndMacSize, exit, result =
errSecDecode);

uint8_t signatureMac[CCSHA256_OUTPUT_SIZE];

cchmac(ccsha256_di(), sizeof(m2), m2, xbSize + 4,
encSigDataBlobStart, signatureMac);

require(xbSize + kSHA256HMAC160Bytes <= *signatureAndMacSize, exit);
[2]

…

exit:

bzero(m1, sizeof(m1));
bzero(m2, sizeof(m2));
bzero(c, sizeof(c));

return result;

}
```

- Error handling erroneously returns successfully on parsing failure

- Encoding an invalid size in an OTR packet establishes a DH key exchange and bypasses signature verification

CVE-2017-2448 - Sample Trigger in 32 Bytes

00	0x00 0x02 0x12 0x00
04	0x00 0x00 0x00 0x18
08	0x41 0x41 0x41 0x41
0C	....
0x1c	0x41 0x41 0x41 0x41

```
int i = 0;
payload[i++] = 0x00;
payload[i++] = 0x02; //version 2

payload[i++] = kSignatureMessage; // packet type

payload[i++] = 0; //xbsize
payload[i++] = 0; //xbsize
payload[i++] = 0; //xbsize
payload[i++] = N-8; //xbsize

payload_length = N;
```

?OTR:AAISAAAAGEFBQUFBQUFBQUFBQUFBQUFBQUFBQUFBQUE=.

Signature Bypass Attack Impact

- MITM Attacker could impersonate existing peers to negotiate secrets

- OTR protocol encrypts using ephemeral keys, verified with the peer identity keys

- Silent attack on targets with 100% reliability

Apple’s iCloud Keychain Security Goals (without OTR fix)

- “Sync passwords between iOS devices and Mac computers without exposing that information to Apple”
- Also protect password material:
  - When the iCloud account is compromised ✗
  - When iCloud is compromised by a rogue insider or external attackers ✗
  - When third parties access user accounts

CVE-2017-2451 - Stack Clash

```
result = ReadLong(signatureAndMacBytes,
signatureAndMacSize, &xbSize);
...
uint8_t xb[xbSize];
```

- Same Routine as CVE-2017-2448
- MITM attacker controls stack allocation size
- Long OTR packet results in data being allocated in adjacent thread’s stack

Stack Overlap Attack Impact

- Potential sandbox escape into secd (as root)
- Malicious local application could potentially gain access to device keychains
- Remotely triggerable as well
- Tricky to exploit due to guard pages, trigger races against a crash

Wrapping up

- Exciting to see strong and usable end-to-end encryption for the masses
- We covered the Keychain Sync Protocol in depth
- We reviewed a critical vulnerability in OTR that undermined the End to End Encryption

Next Steps for the Security Industry

- Should this have been discovered after Goto Fail?
  - Strikingly similar, same code project.
  - See Crypto Testing Talk
- Are the protocol details sufficiently transparent to users?
  - Mostly open source, but we're still the first to discuss OTR publicly
- More research needed on the two-factor implementation, and its interface with iCloud Keychain Recovery and iCloud Keychain Syncing

Questions?

LONGTERM

Appendix

LONGTERM

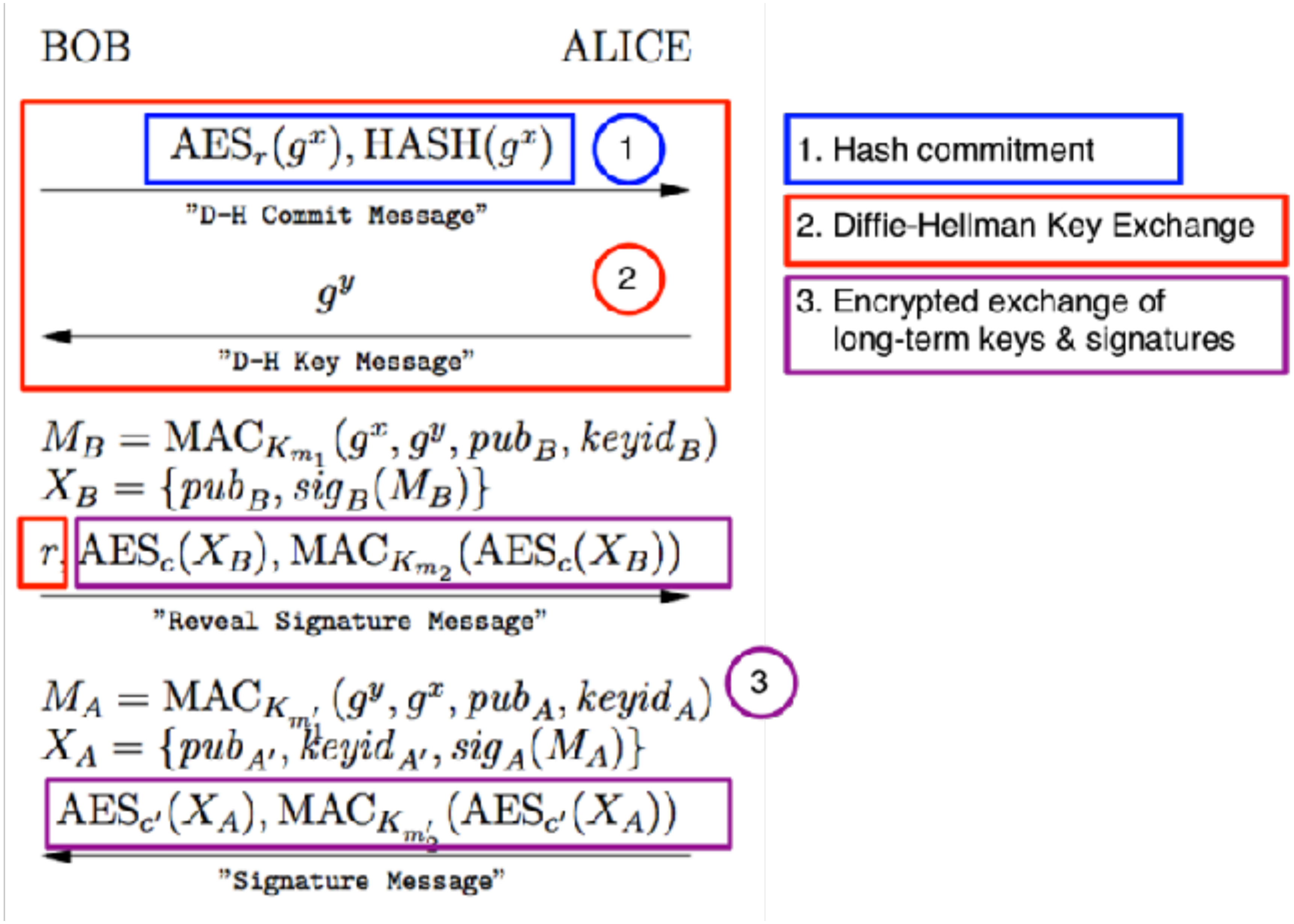
Circle Protocol Parameters

- Apple ID Password converted to ECC keypair using PBKDF2 and X9.63
- Identity Keys are 256-bit keys on the secp256r1 curve
  - Stored in Keychain with **kSecAttrAccessibleWhenUnlockedThisDeviceOnly** protection class
  - Cloud Identity Key **kSecAttrAccessibleWhenUnlocked** and synchronizable

OTR Encryption Parameters

- NIST Curve (secp256r1)
- ECDH with ephemeral keys over secp256r1
- ECDSA signatures over secp256r1 with SHA-256
- SHA256-HMAC-160
- 128-bit AES-CTR used for encryption

OTR Asynchronous Key Exchange



$\mathrm{AES}_r(g^x), \mathrm{HASH}(g^x)$

g^y

$$M_B = \mathrm{MAC}_{K_{m_1}}(g^x, g^y, pub_B, keyid_B)$$
$$X_B = \{pub_B, sig_B(M_B)\}$$
$$r, \mathrm{AES}_c(X_B), \mathrm{MAC}_{K_{m_2}}(\mathrm{AES}_c(X_B))$$

$$M_A = \mathrm{MAC}_{K_{m_1'}}(g^y, g^x, pub_A, keyid_A)$$
$$X_A = \{pub_{A'}, keyid_{A'}, sig_A(M_A)\}$$
$$\mathrm{AES}_{c'}(X_A), \mathrm{MAC}_{K_{m_2'}}(\mathrm{AES}_{c'}(X_A))$$

1. Hash commitment

2. Diffie-Hellman Key Exchange

3. Encrypted exchange of long-term keys & signatures

https://blog.cryptographyengineering.com/2016/03/21/attack-of-week-apple-imessage/