

Cryptography Protocols

Thierry Sans

Security goals vs attacker's model



Let us consider **confidentiality, integrity and availability**

Design of a cryptography protocol

The hypothesis on the system

- **What is the network model?**

bandwidth, latency, reliability, message ordering, synchronous vs asynchronous

- **What trusted setup is assumed?**

pre-shared keys, key generation

- **How powerful are the parties vs. attacker?**

computing power, source of randomness

- **Which adversary model is considered?**

outsider vs insider, passive vs active, man-in-the-middle, man-at-the-end, corruption

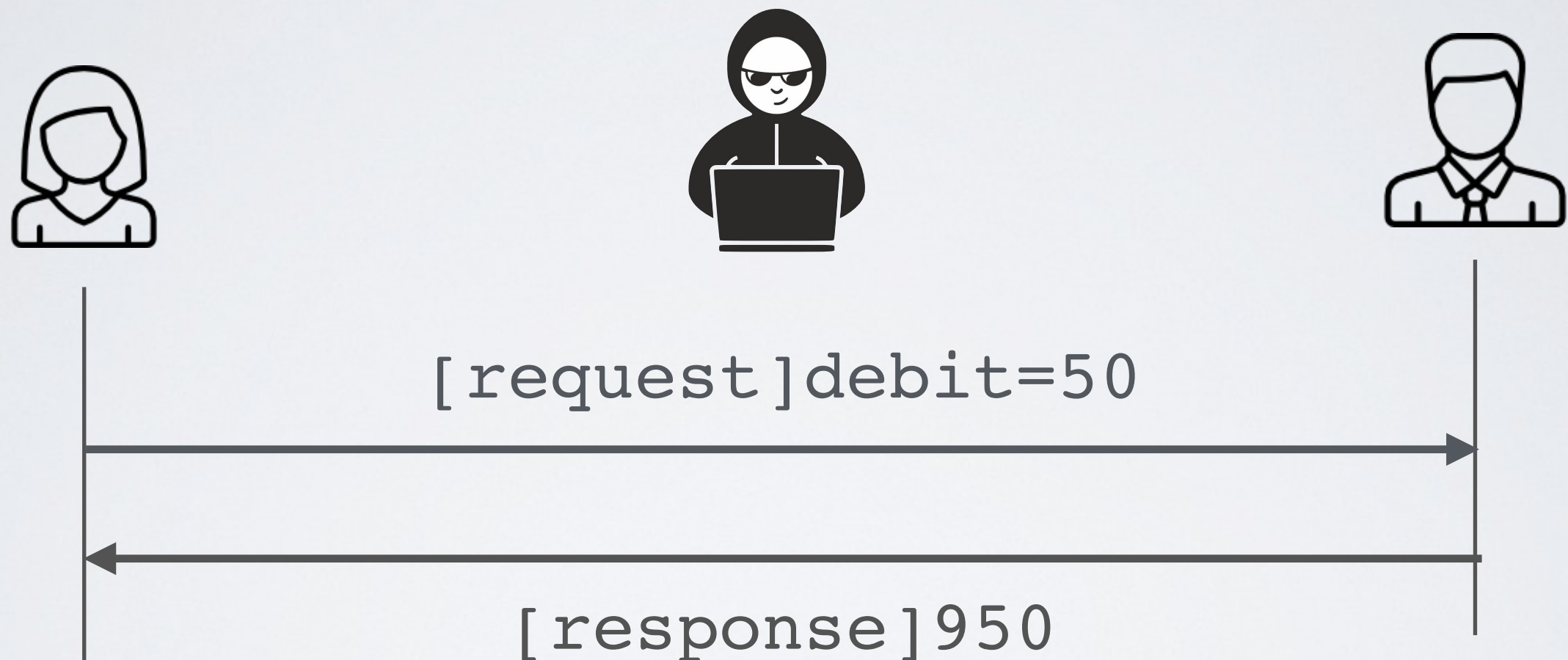
- **What kinds of failures are tolerated?**

crash faults, byzantine faults

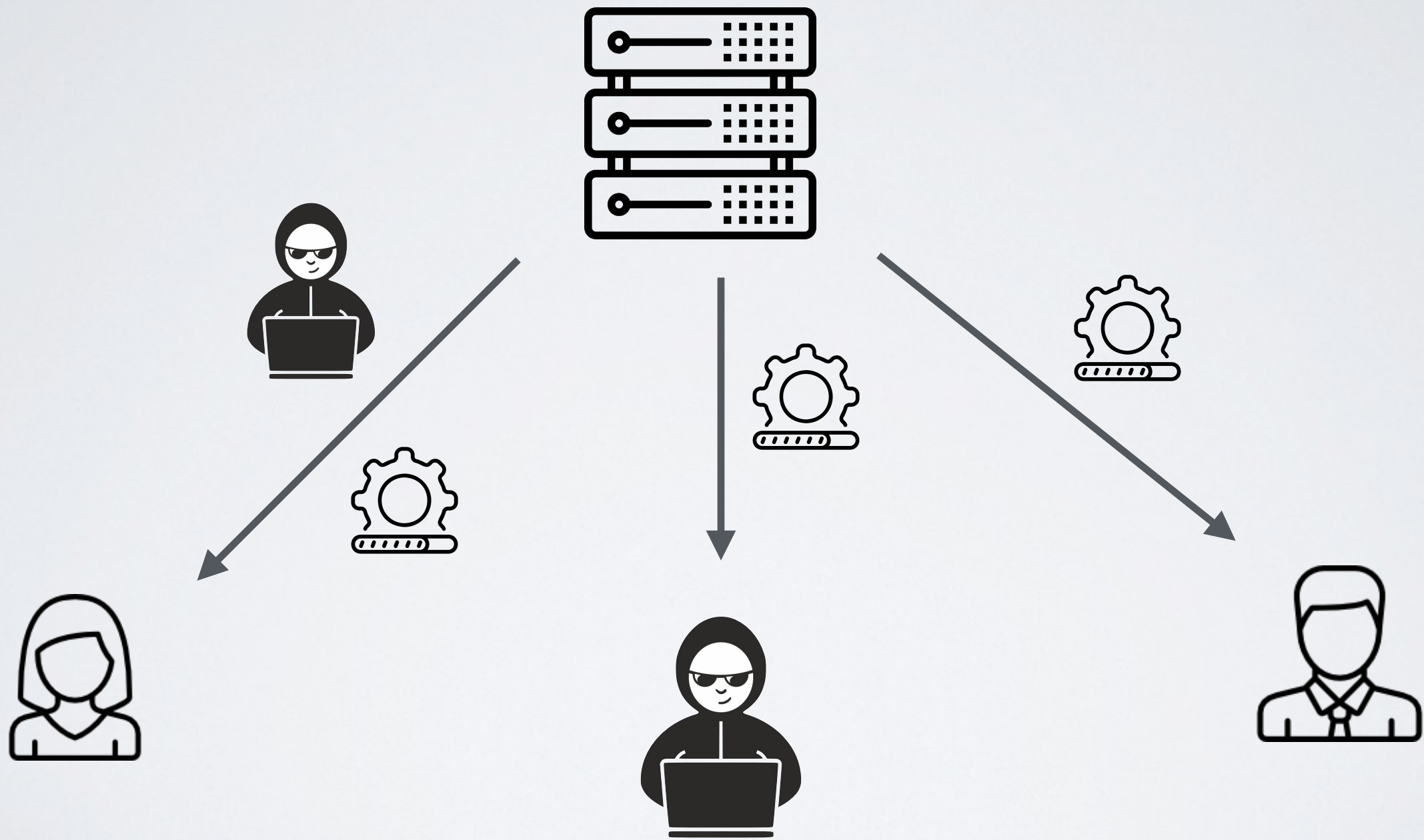
- **What exact security properties are being claimed?**

confidentiality, integrity, authentication, non-repudiation, forward secrecy

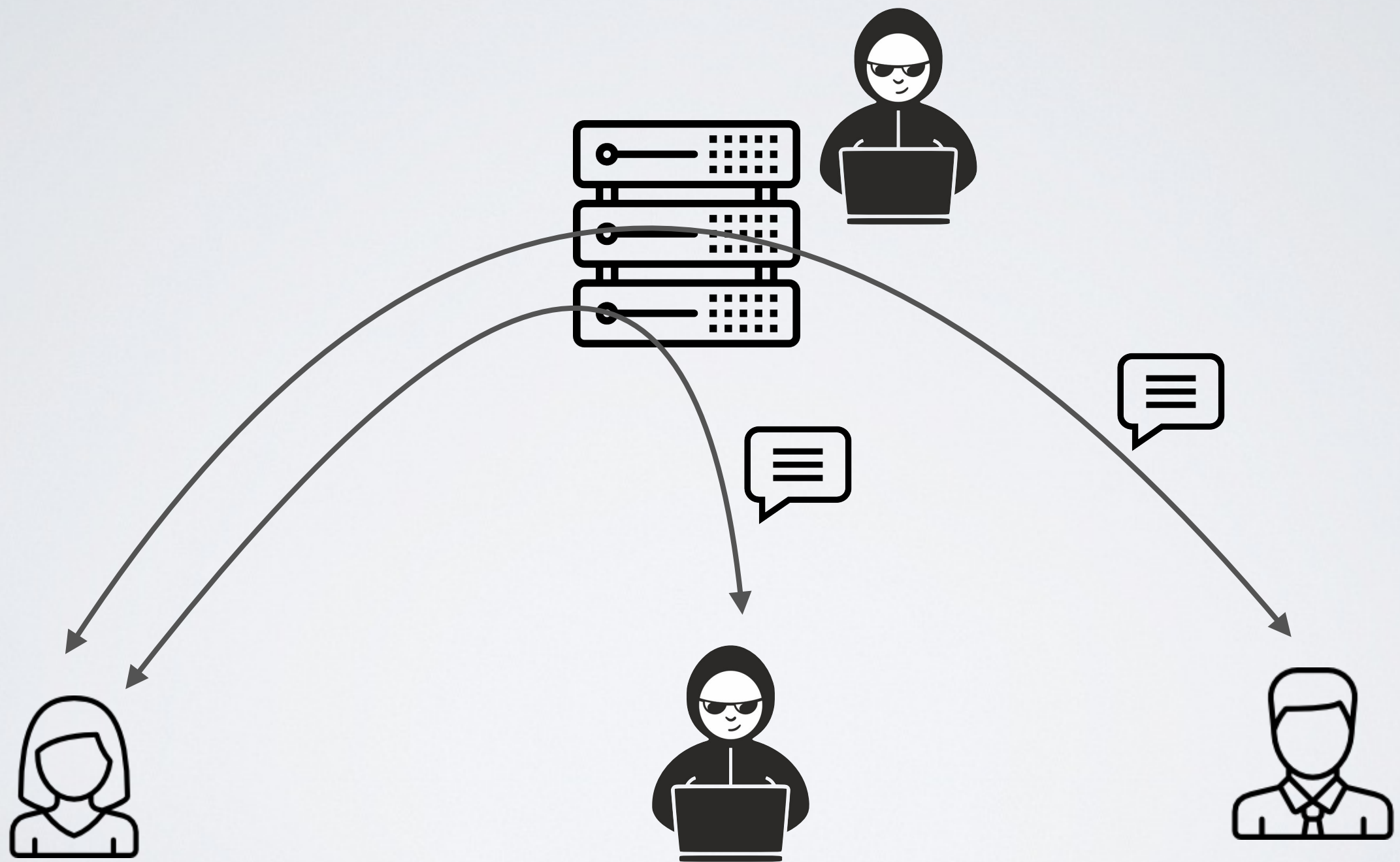
Example 1 - **Interactive Protocol**



Example 2 - **Distribution Centre**

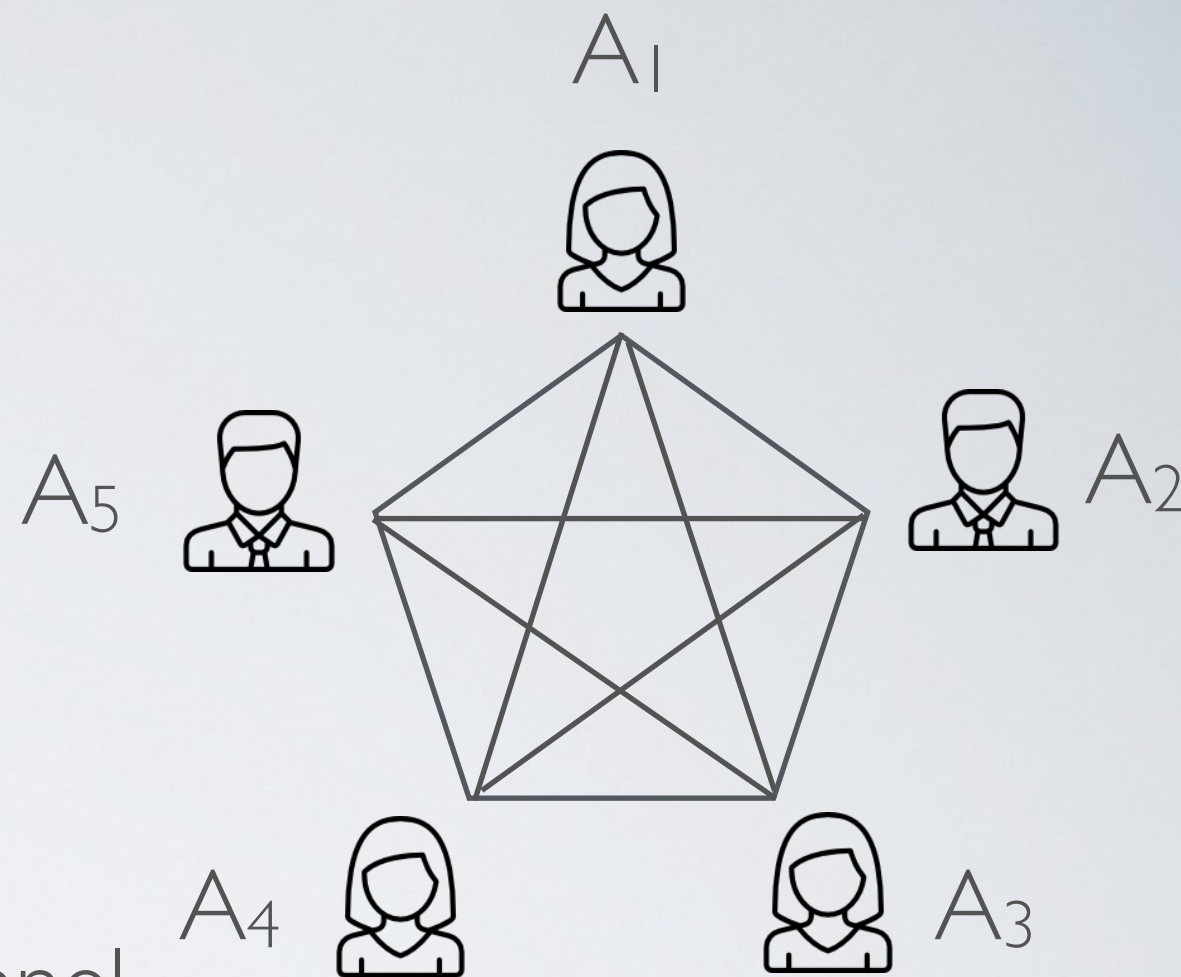


Example 3 - **Asynchronous Messaging**



Replay attacks

Interactive Protocol



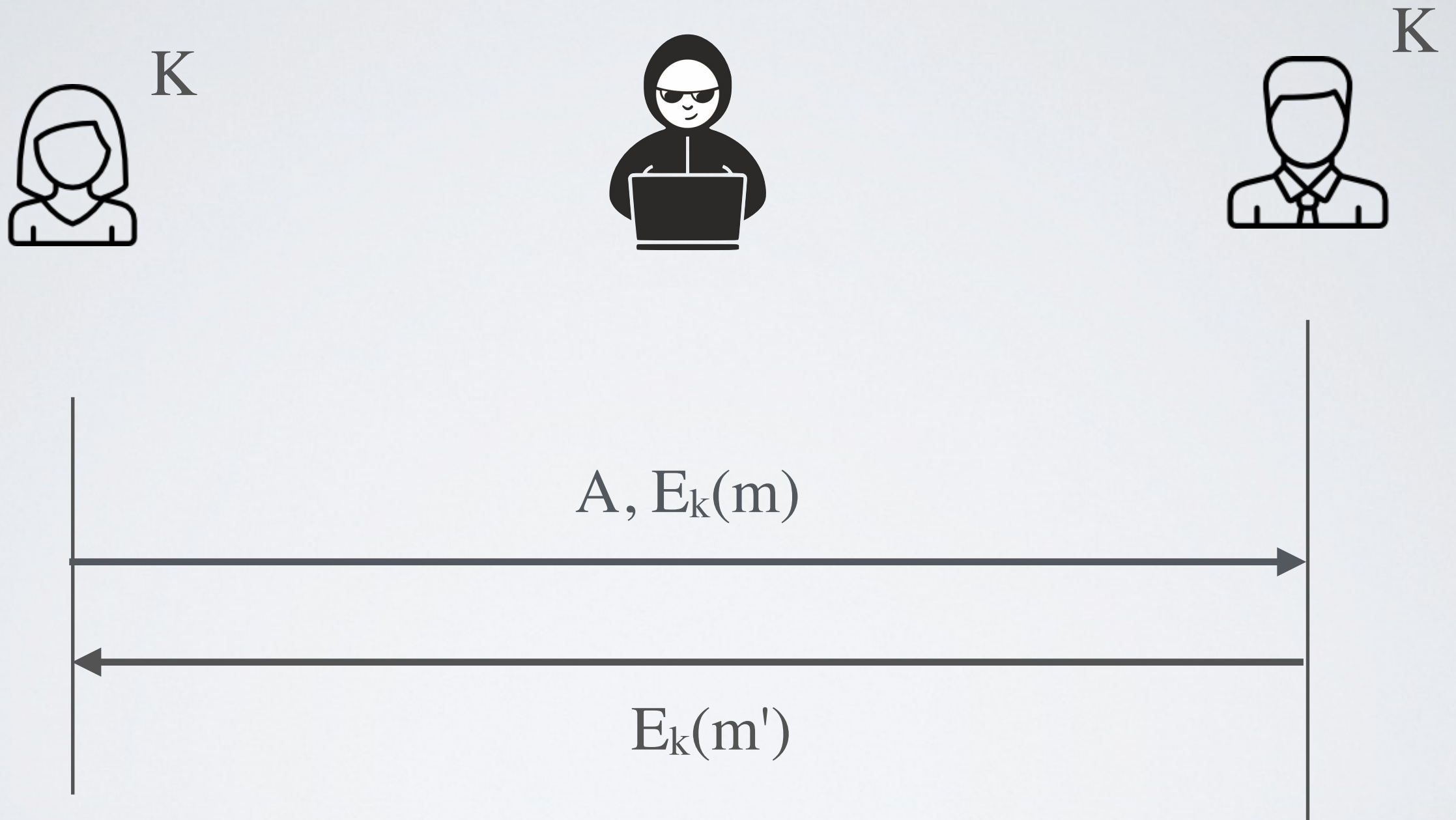
System Hypothesis

- Synchronous communication channel
- Each participant share a unique symmetric key with each other
- Mallory can read, modify and forge message send over the network

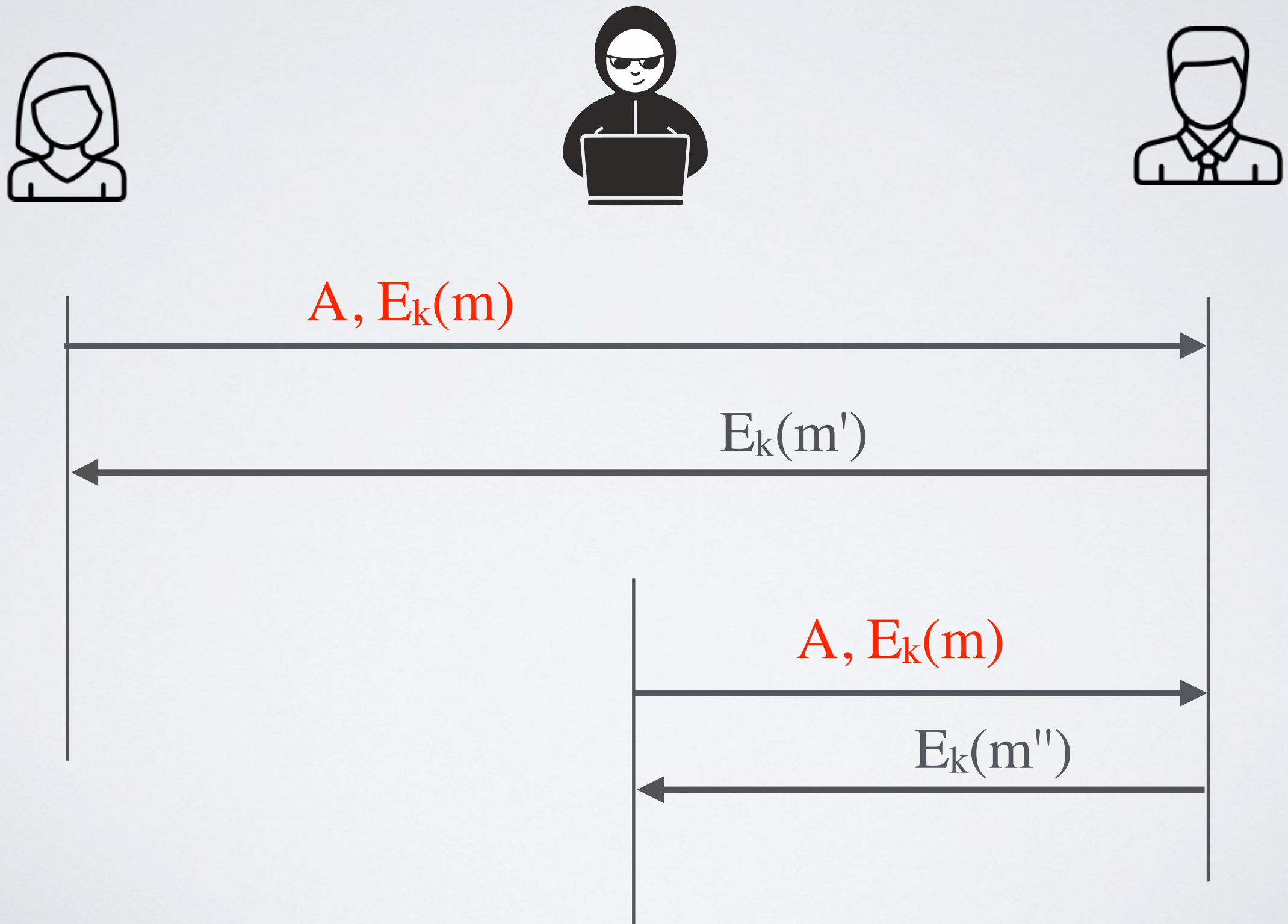
Goals

- When two participant exchange a message, the system should protect the confidentiality and integrity of the message

Using Authenticated Encryption



Problem : replay attack



Counter replay attacks

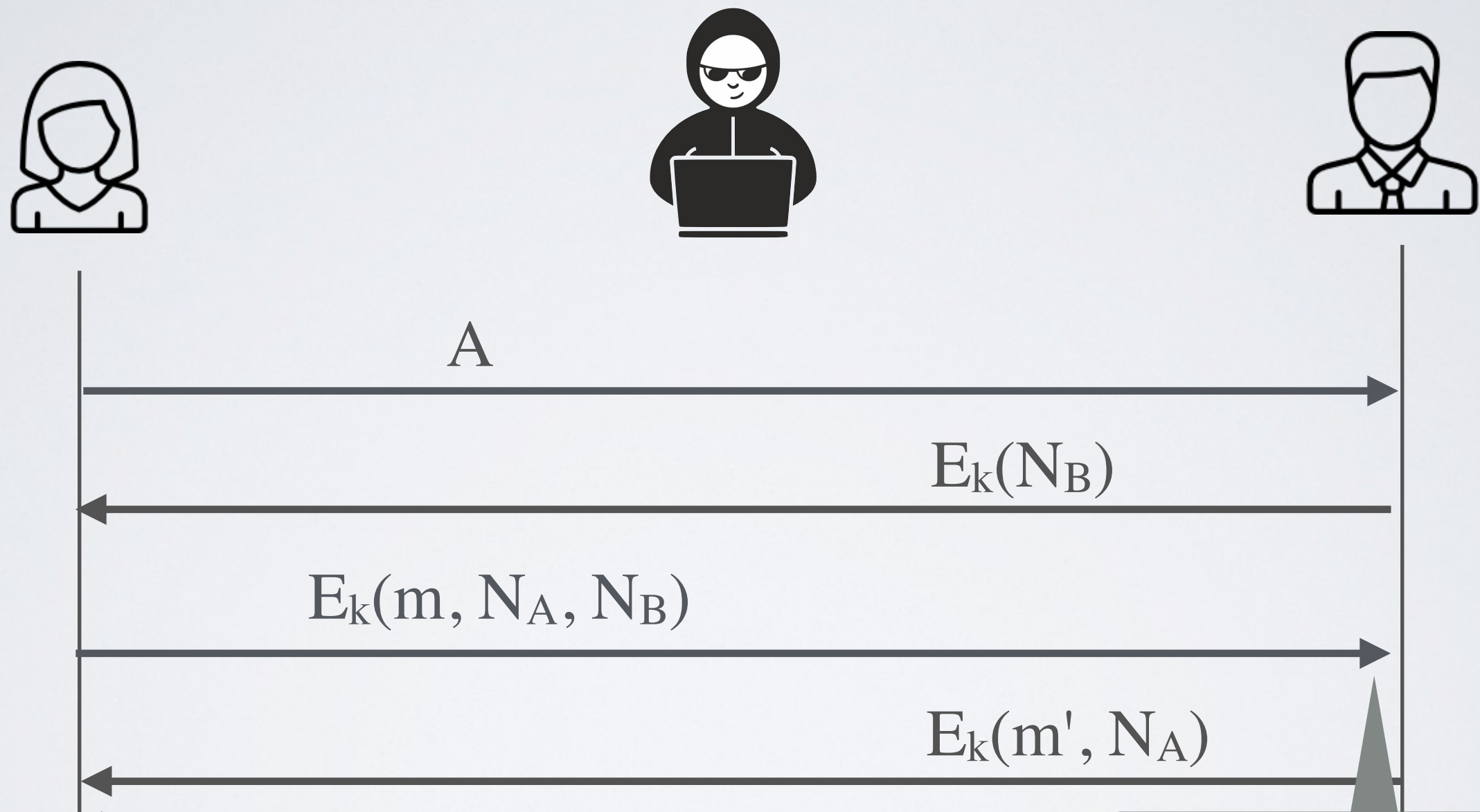
✓ **Storage-based solution**

Store the message entirely (log), or ID or encryption nonce or timestamp and check whether the same message has been replayed

✓ **Protocol-based solution**

Add a nonce in the interaction and verify that the nonce is sent back

Double Nonce Protocol



Alice must check that the nonce N_A received match the original one

Bob must check that the nonce N_B received match the original one

Are we secure yet?

Two major issues:

1. **Key distribution**

If $A_1, A_2 \dots A_5$ want to talk, then $n \cdot (n-1) / 2$ keys must be exchanged physically using a secure channel

2. Does not ensure **Perfect-Forward Secrecy**

If somehow Mallory is able to compromise one of the participant at some point in time, she can decrypt all previous communications between Alice and Bob

Session Keys

Interactive Protocol

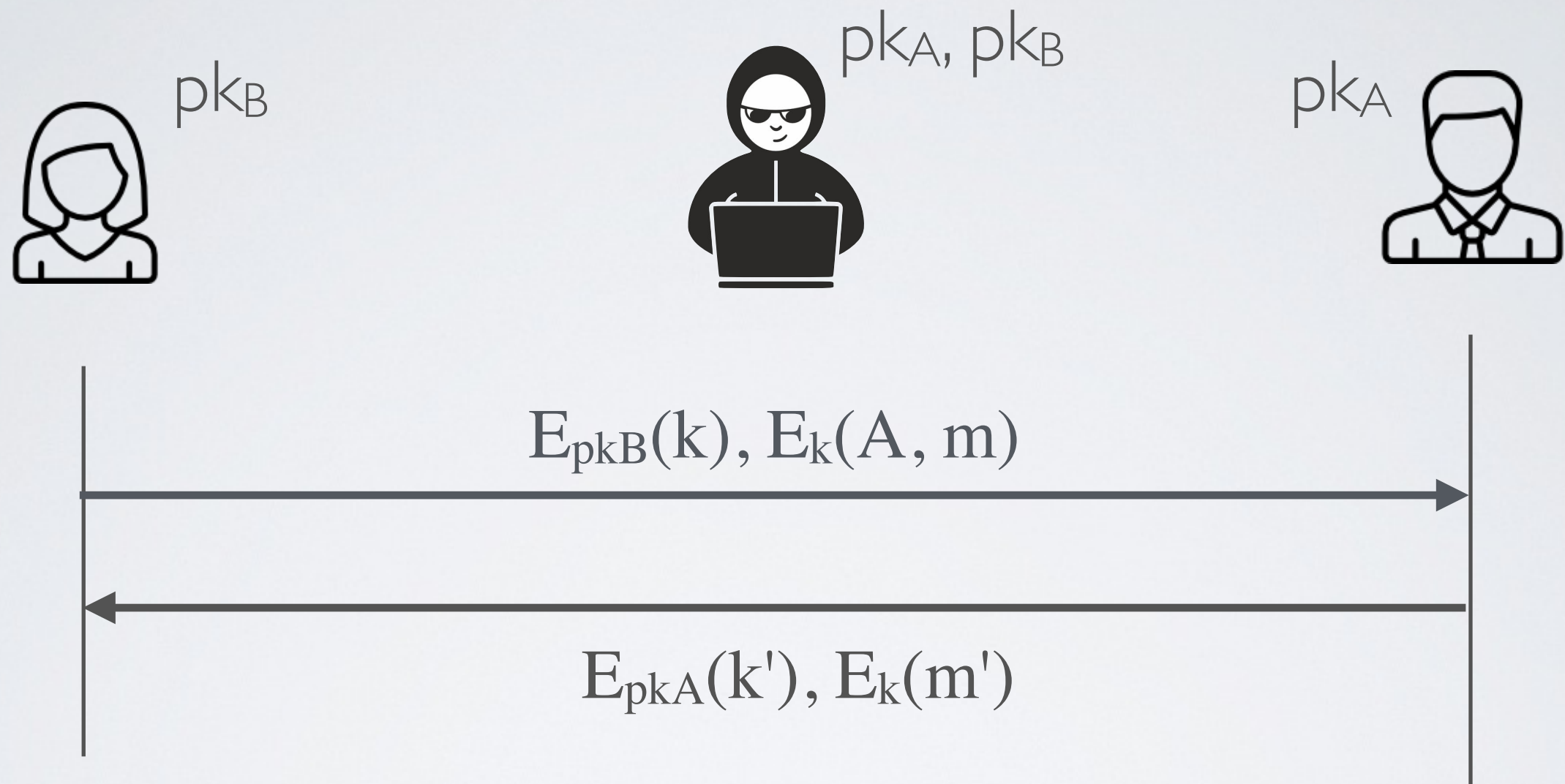
System Hypothesis

- Synchronous communication channel
- **Each participant has a public key pair and everybody knows everyone's public keys**
- Mallory can read, modify and forge message send over the network

Goals

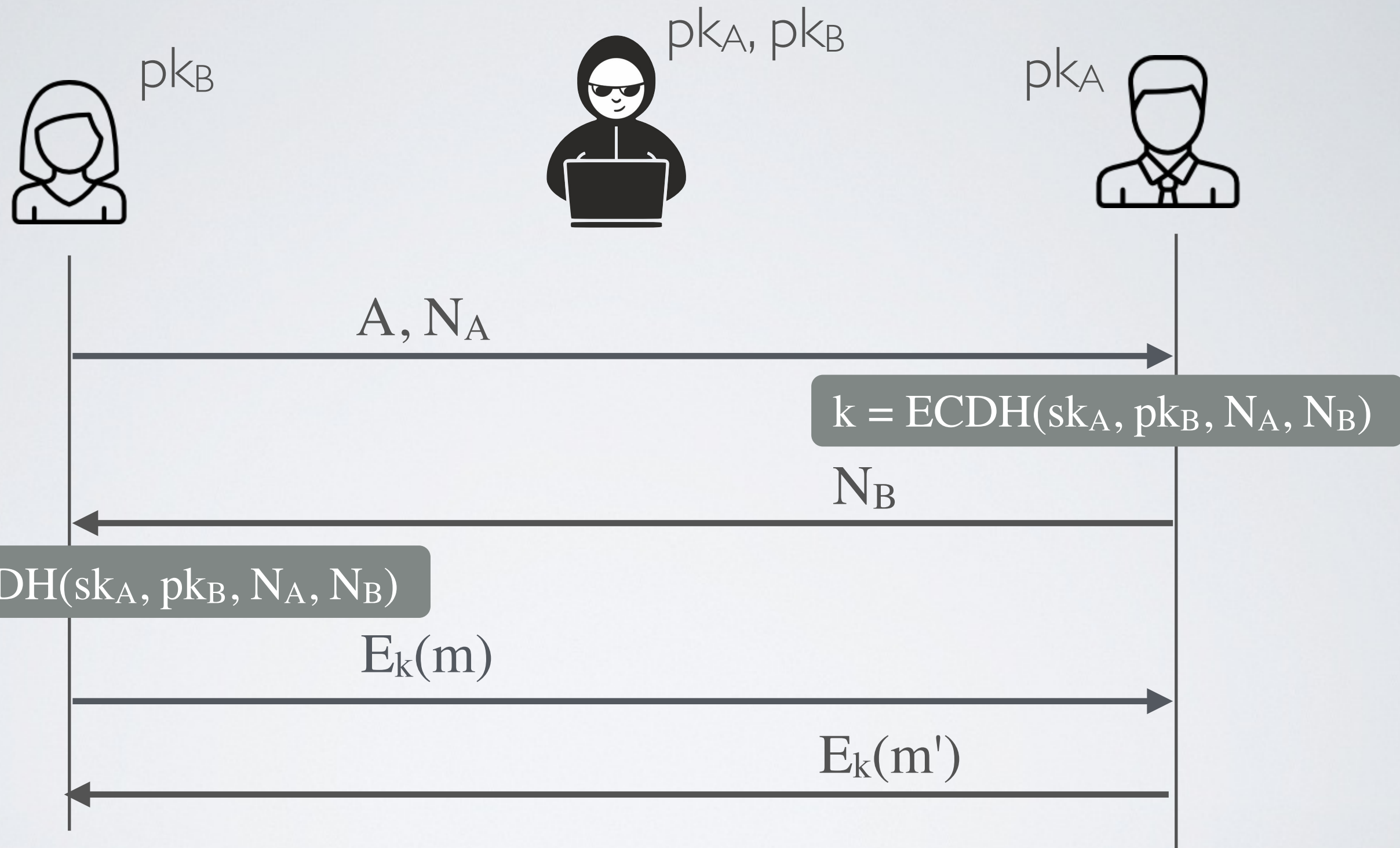
- When two participant exchange a message, the system should protect the confidentiality, integrity **and perfect forward secrecy** of the messages

[broken] Key Wrapping



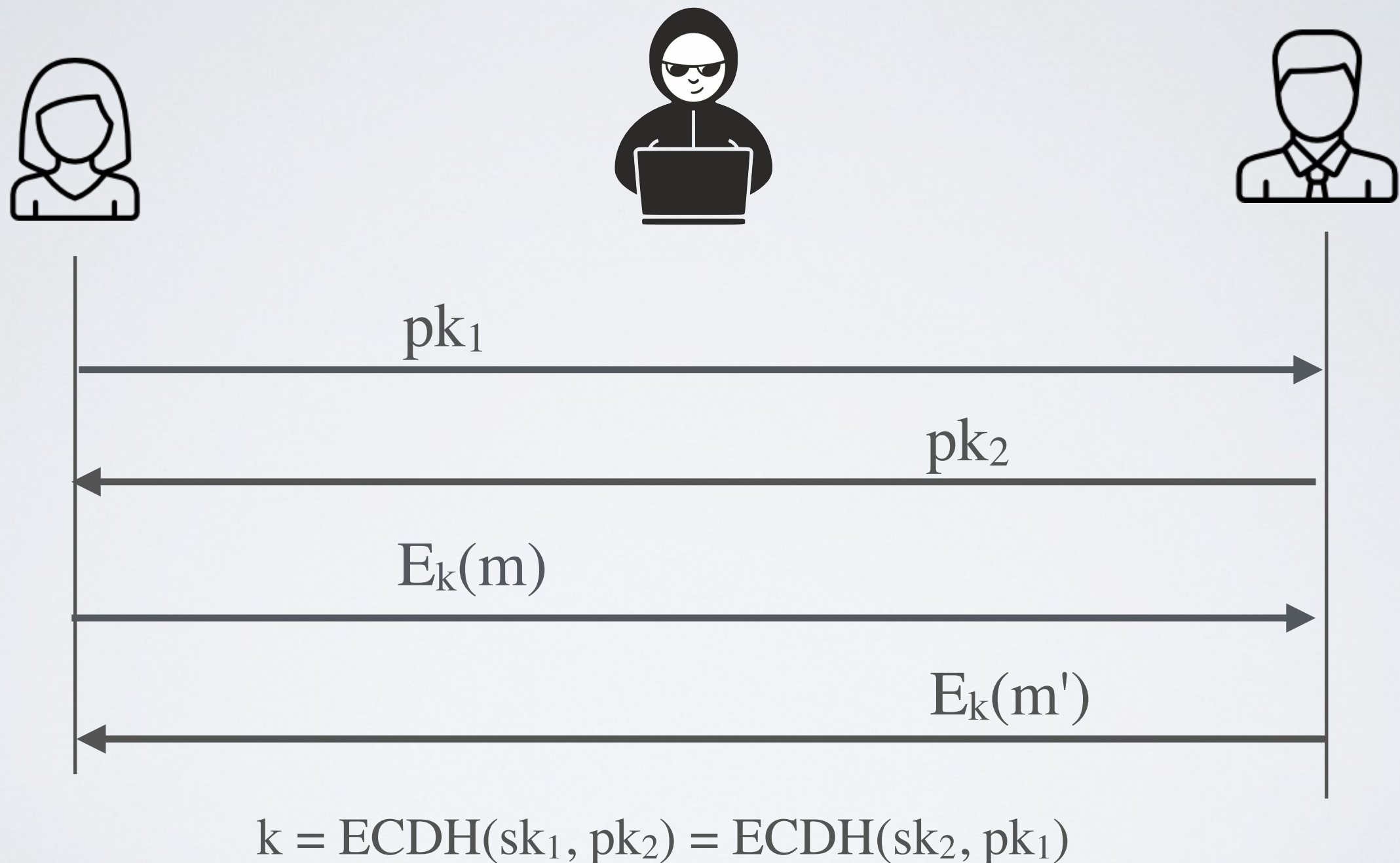
- Replay attack
- No Forward Secrecy
- Bad Authentication

[broken] Key Derivation using Long-Term Keys

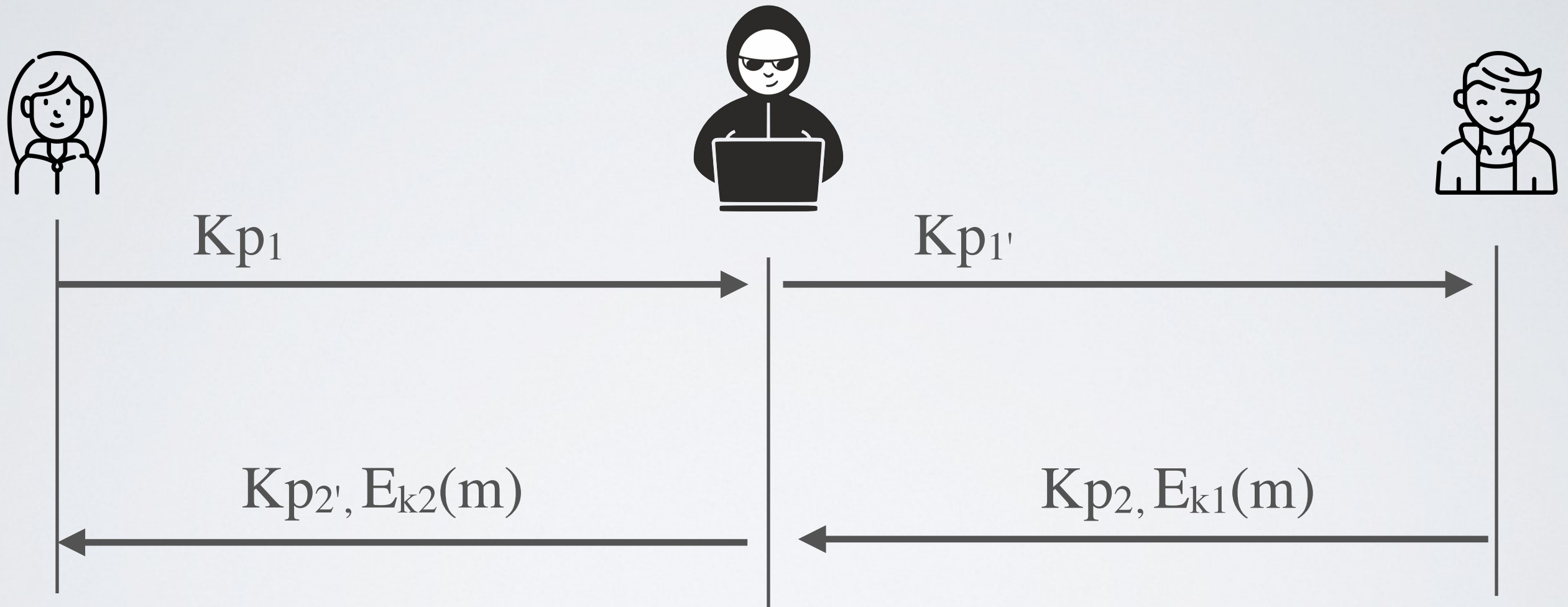


● No Forward Secrecy

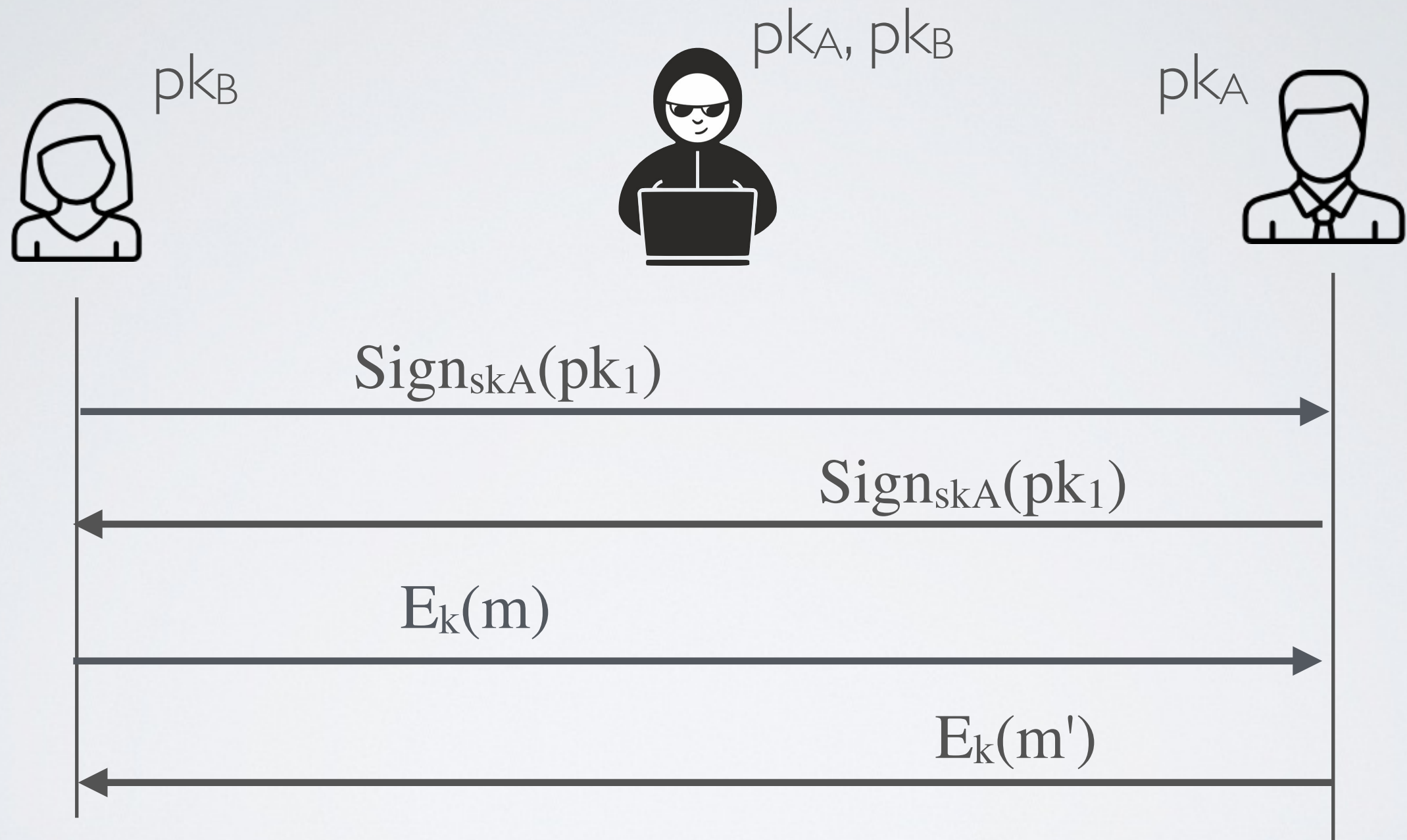
[broken] Key Derivation using Short-Term Keys



The key exchange is not authenticated



Key Derivation with Authenticated Short-Term Keys

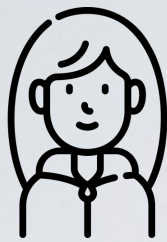


One major issue

Key distribution

If $A_1, A_2 \dots A_5$ want to talk, then n public keys must be distributed physically to each user using a secure channel

Why not?



K_{SB}, K_{pB}

$\text{Sign}_{skB}(\text{"I'm Bob"}, K_{pB})$



K_{SM}, K_{pM}

$\text{Sign}_{skM}(\text{"I'm Bob"}, K_{pM})$

Trust Models

Transitive Trust

If Alice does not know Bob maybe **she knows Charlie that knows Bob** and can vouch for him

✓ Charlie is a Trusted-Third Party for Alice

Two trust models

How to establish the authenticity of the binding between someone's identity and its public key ?

Decentralized trust model

➔ **Web of Trust**

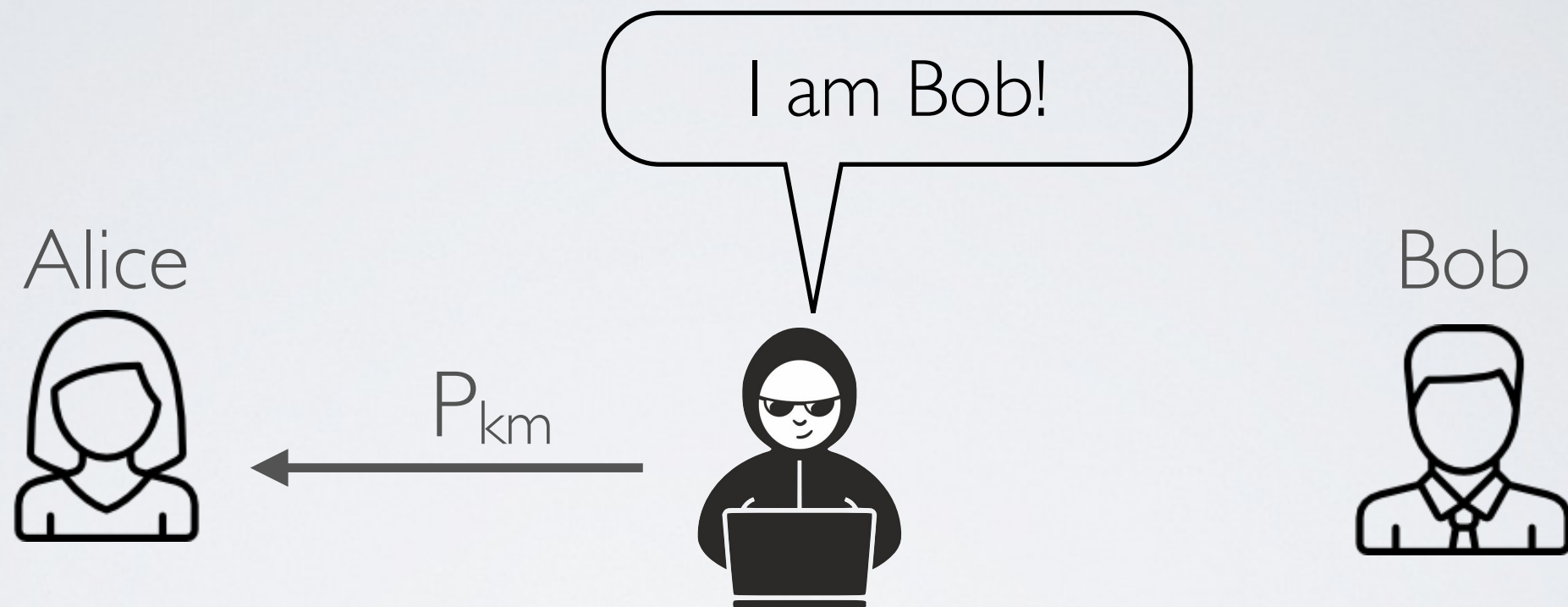


Centralized trust model

➔ **PKI - Public Key Infrastructure**



Do you trust the GPG key ?

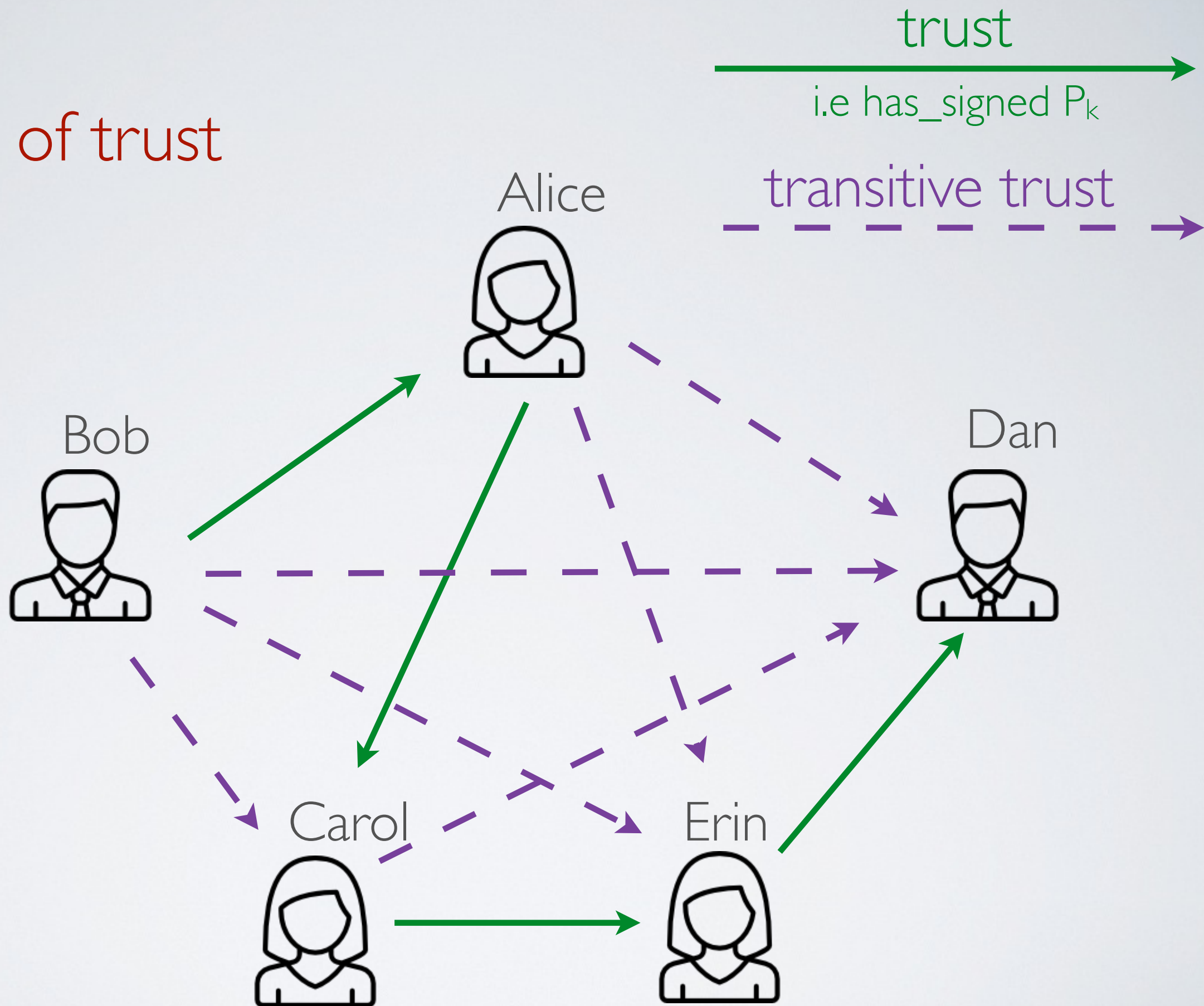


Alice should verify Bob's public key fingerprint

- either by communicating with Bob over another channel
- or by trusting someone that already trusts Bob

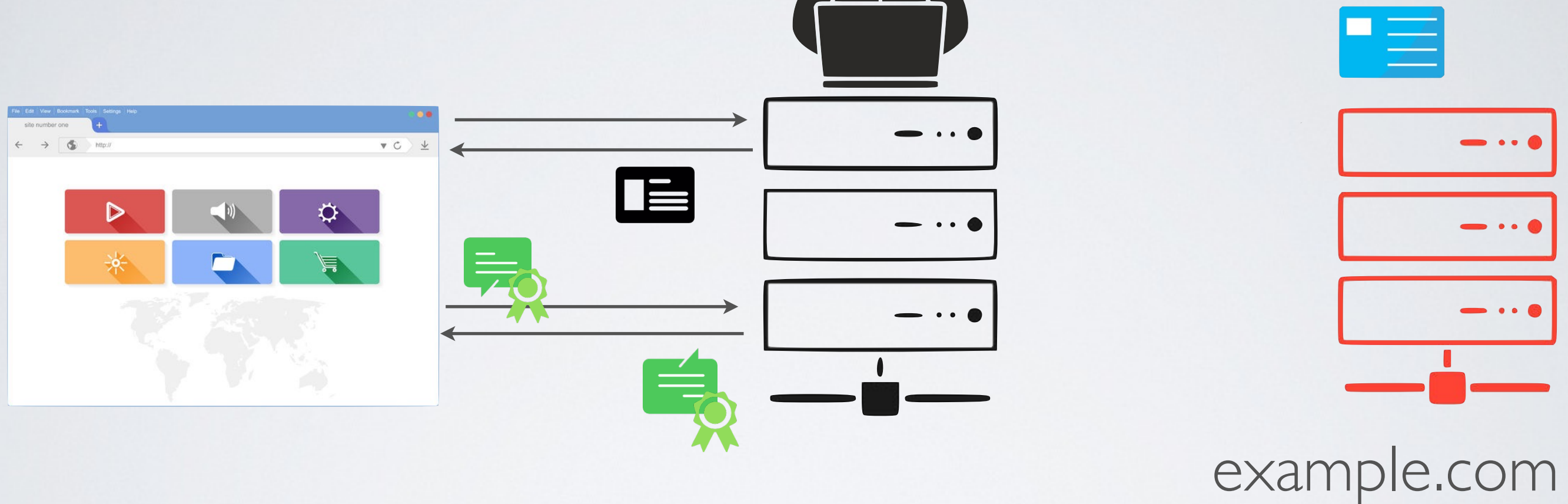
➡ **the web of trust**

The web of trust



Do you trust the network ?

I am `example.com`!

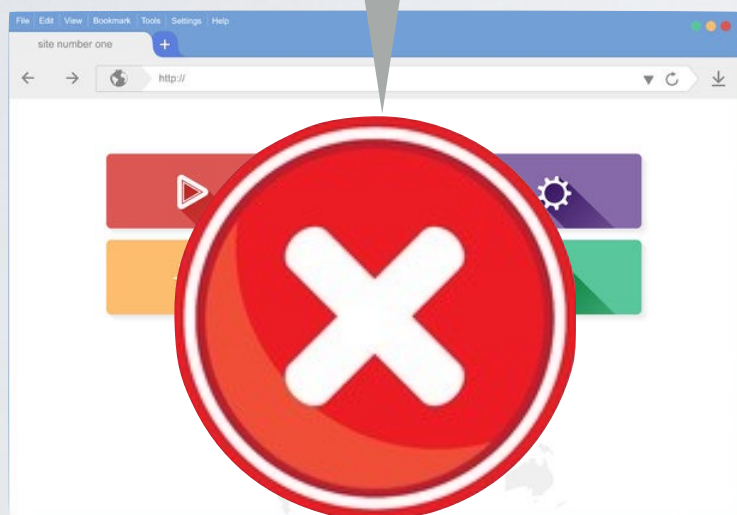


The browser should verify the certificate

➔ **PKI - Public Key Infrastructure**

Generating and using (self-signed) certificates

I don't know



Who are you?



I am example.com



Self-signed certificates are not trusted by your browser



Your connection is not private

Attackers might be trying to steal your information from **bitbucket.org** (for example, passwords, messages, or credit cards).

[Hide advanced](#)

[Reload](#)

bitbucket.org normally uses encryption to protect your information. When Chrome tried to connect to bitbucket.org this time, the website sent back unusual and incorrect credentials. Either an attacker is trying to pretend to be bitbucket.org, or a Wi-Fi sign-in screen has interrupted the connection. Your information is still secure because Chrome stopped the connection before any data was exchanged.

You cannot visit bitbucket.org right now because the website uses HSTS. Network errors and attacks are usually temporary, so this page will probably work later.

NET::ERR_CERT_DATE_INVALID



This Connection is Untrusted

You have asked Firefox to connect securely to **www.domainname.tld** but we can't confirm that your connection is secure.

Normally, when you try to connect securely, sites will present trusted identification to prove that you are going to the right place. However, this site's identity can't be verified.

What Should I Do?

If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

[Get me out of here!](#)

► Technical Details

▼ I Understand the Risks

If you understand what's going on, you can tell Firefox to start trusting this site's identification. **Even if you trust the site, this error could mean that someone is tampering with your connection.**

Don't add an exception unless you know there's a good reason why this site doesn't use trusted identification.

[Add Exception...](#)

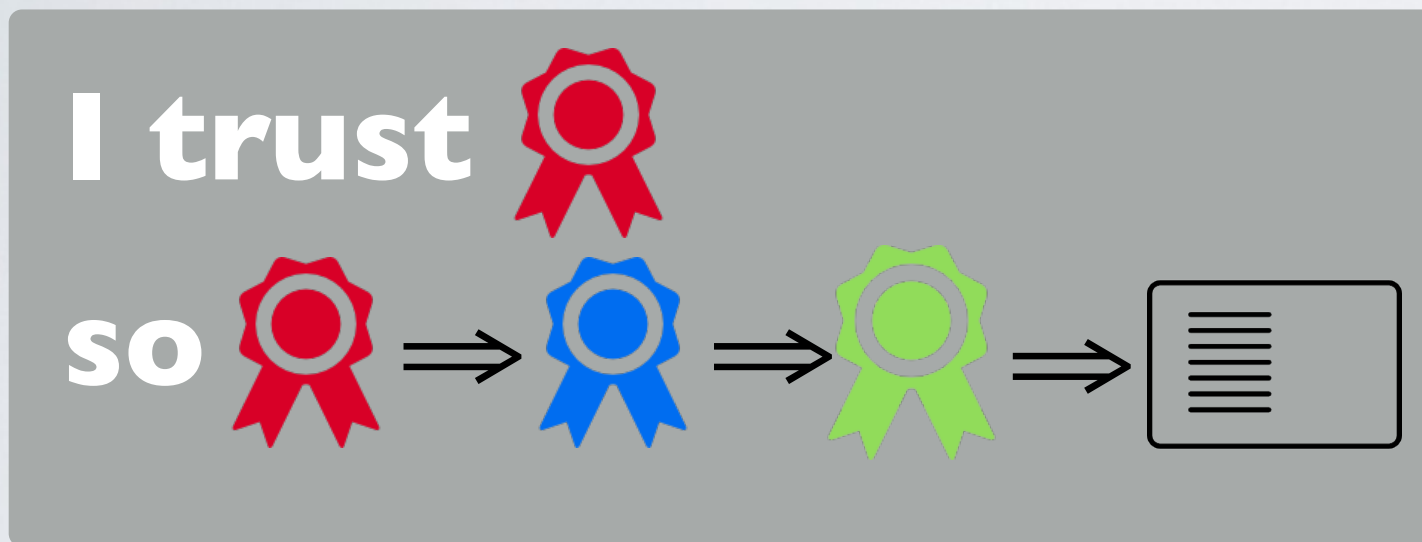


Signed Certificate

Certificate Authority (CA)



The Chain of Trust



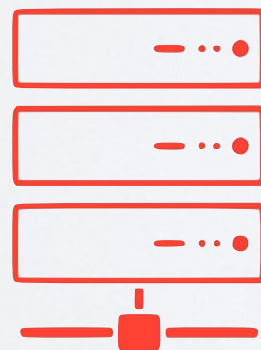
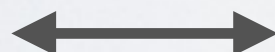
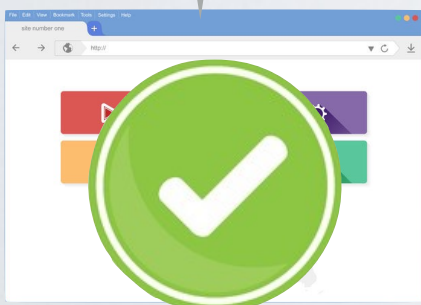
Root CA



Intermediate
CA



Intermediate
CA



Your browser trusts many root CAs **by default**

Keychain Access

Click to unlock the System Roots keychain.

Keychains

- login
- Microsoft_Int...diate_Certificates
- Local Items
- System
- System Roots**

Category

- All Items
- Passwords
- Secure Notes
- My Certificates
- Keys
- Certificates**

GeoTrust Global CA
Root certificate authority
Expires: Saturday, May 21, 2022 at 7:00:00 AM Arabian Standard Time
✓ This certificate is valid

Name	Kind	Expires	Keychain
Echoworx Root CA2	certificate	Oct 7, 2030, 1:49:13 PM	System Roots
EE Certification Centre Root CA	certificate	Dec 18, 2030, 2:59:59 AM	System Roots
Entrust Root Certification Authority	certificate	Nov 27, 2026, 11:53:42 PM	System Roots
Entrust Root Certification Authority - EC1	certificate	Dec 18, 2037, 6:55:36 PM	System Roots
Entrust Root Certification Authority - G2	certificate	Dec 7, 2030, 8:55:54 PM	System Roots
Entrust.net Certification Authority (2048)	certificate	Dec 24, 2019, 9:20:51 PM	System Roots
Entrust.net Certification Authority (2048)	certificate	Jul 24, 2029, 5:15:12 PM	System Roots
ePKI Root Certification Authority	certificate	Dec 20, 2034, 5:31:27 AM	System Roots
Federal Common Policy CA	certificate	Dec 1, 2030, 7:45:27 PM	System Roots
GeoTrust Global CA	certificate	May 21, 2022, 7:00:00 AM	System Roots
GeoTrust Primary Certification Authority	certificate	Jul 17, 2036, 2:59:59 AM	System Roots
GeoTrust Primary Certification Authority - G2	certificate	Jan 19, 2038, 2:59:59 AM	System Roots
GeoTrust Primary Certification Authority - G3	certificate	Dec 2, 2037, 2:59:59 AM	System Roots
Global Chambersign Root	certificate	Sep 30, 2037, 7:14:18 PM	System Roots
Global Chambersign Root - 2008	certificate	Jul 31, 2038, 3:31:40 PM	System Roots
GlobalSign	certificate	Mar 18, 2029, 1:00:00 PM	System Roots
GlobalSign	certificate	Jan 19, 2038, 6:14:07 AM	System Roots
GlobalSign	certificate	Jan 19, 2038, 6:14:07 AM	System Roots
GlobalSign	certificate	Dec 15, 2021, 11:00:00 AM	System Roots
GlobalSign Root CA	certificate	Jan 28, 2028, 3:00:00 PM	System Roots
Go Daddy Class 2 Certification Authority	certificate	Jun 29, 2034, 8:06:20 PM	System Roots
Go Daddy Root Certificate Authority - G2	certificate	Jan 1, 2038, 2:59:59 AM	System Roots
Government Root Certification Authority	certificate	Dec 31, 2037, 6:59:59 PM	System Roots
Hellenic Academic and Research Institutions RootCA 2011	certificate	Dec 1, 2031, 4:49:52 PM	System Roots
Hongkong Post Root CA 1	certificate	May 15, 2023, 7:52:29 AM	System Roots

177 items

Real attacks

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

An update on attempted man-in-the-middle attacks

August 29, 2011

Posted by Heather Adkins, Information Security Manager

Today we received reports of attempted SSL man-in-the-middle (MITM) attacks against Google users, whereby someone tried to get between them and encrypted Google services. The people affected were primarily located in Iran. The attacker used a fraudulent SSL certificate issued by DigiNotar, a root certificate authority that should not issue certificates for Google (and has since revoked it).

Google Chrome users were protected from this attack because Chrome was able to [detect](#) the fraudulent certificate.

Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Enhancing digital certificate security

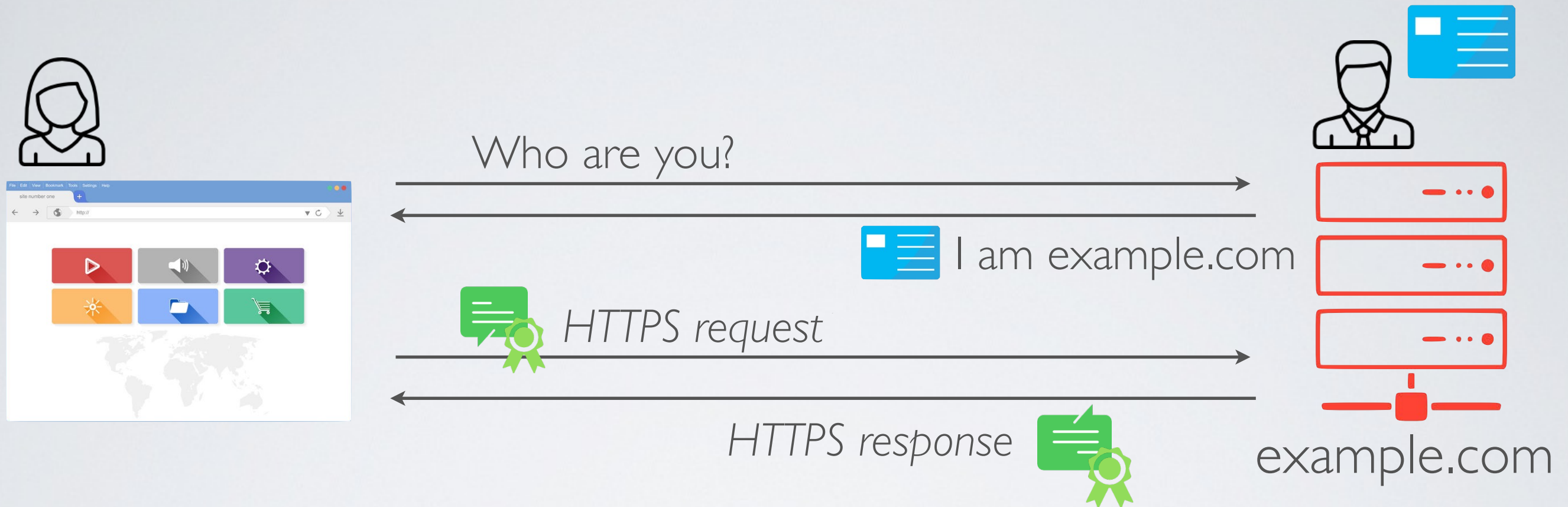
January 3, 2013

Posted by Adam Langley, Software Engineer

Late on December 24, Chrome detected and blocked an unauthorized digital certificate for the "*.google.com" domain. We investigated immediately and found the certificate was issued by an [intermediate certificate authority](#) (CA) linking back to TURKTRUST, a Turkish certificate authority. Intermediate CA certificates carry the full authority of the CA, so anyone who has one can use it to create a certificate for any website they wish to impersonate.

TLS - Transport Layer Security
a.k.a SSL - Secure Sockets Layer

This how HTTPS works

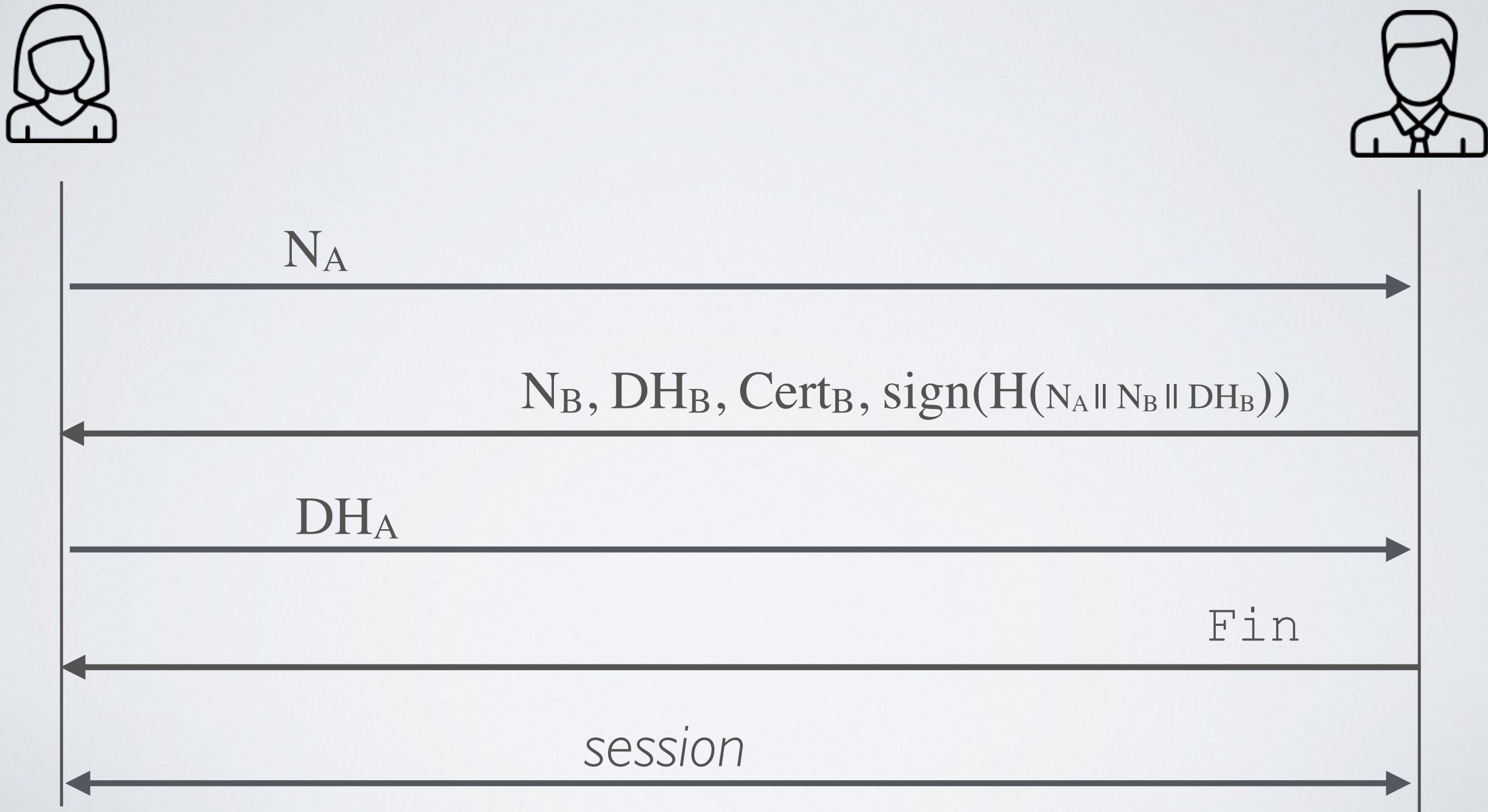


✓ **HTTPS = HTTP + TLS**

- ➔ TLS - Transport Layer Security (a.k.a SSL) provides
- **confidentiality** : end-to-end secure channel
 - **integrity** : one-way authentication handshake

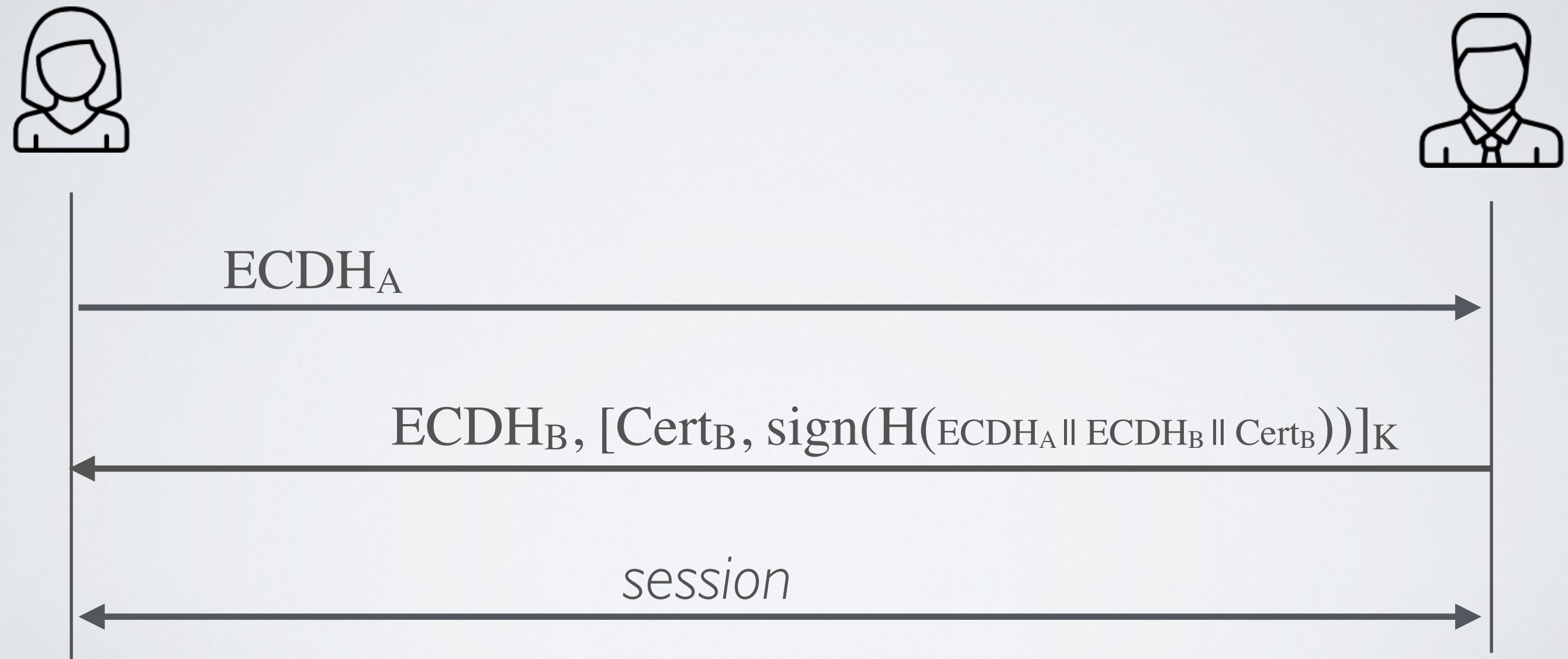
simplified and one-way authentication

TLS 1.2 (2008)



simplified and one-way authentication

TLS 1.3 (2018)



TLS 1.3 is much better than TLS 1.2

- ✓ Only one round in the handshake (vs 2 with TLS 1.2)
- ✓ Faster (use of elliptic curves)
- ✓ Certificate is encrypted (better confidentiality)
- ✓ Protocol has been formally proven
(does not prevent from implementation bugs)

The Signal Protocol

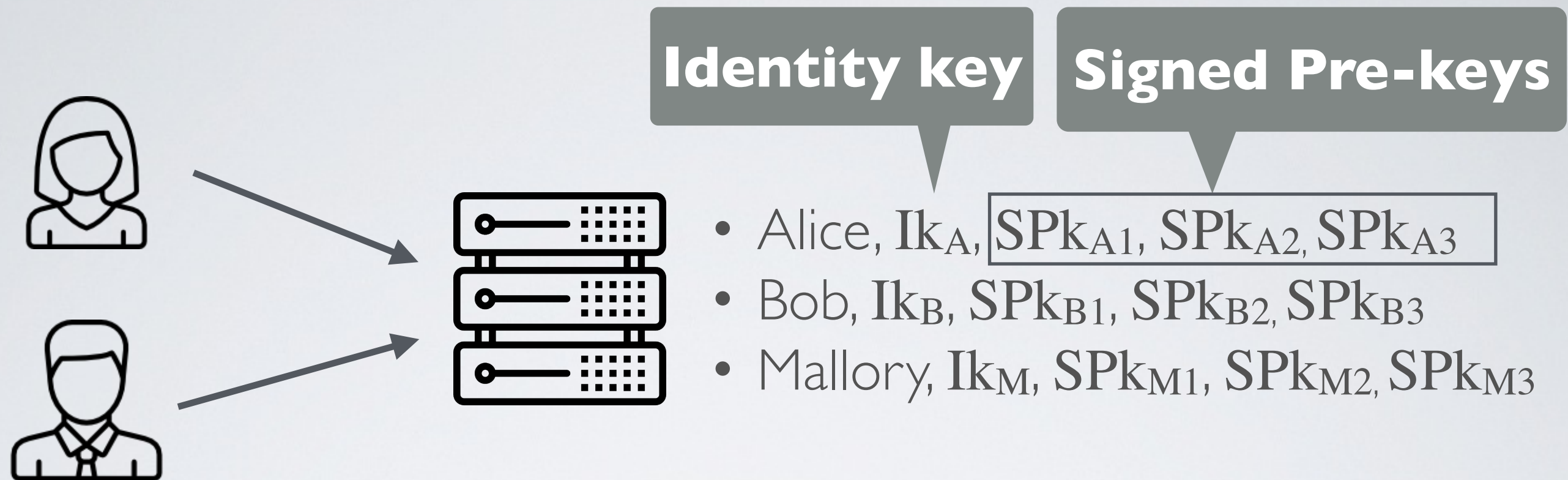
Signal in a Nutshell

➔ Asynchronous Protocol

Two phases

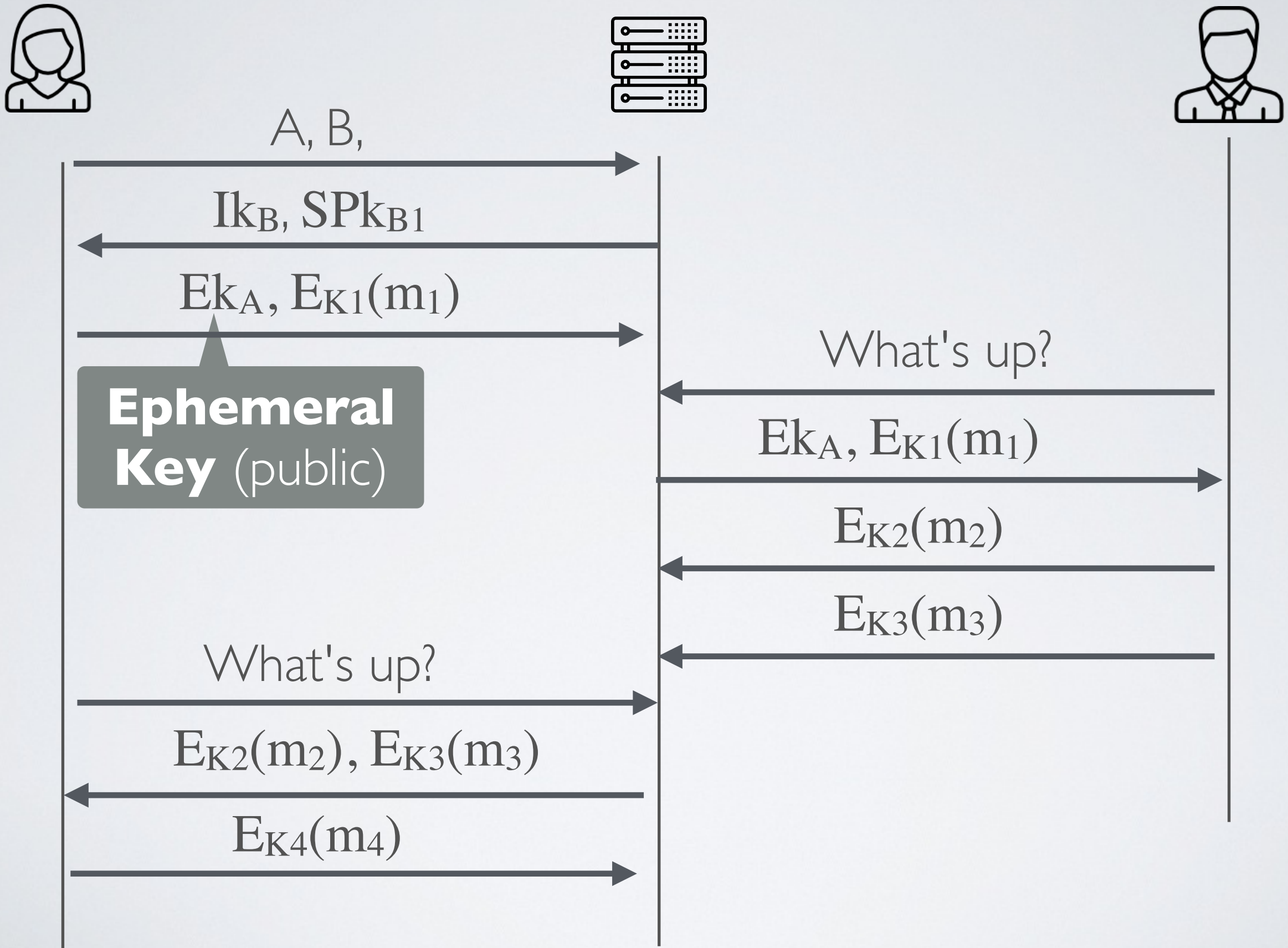
- **Triple Diffie-Hellman Key Exchange (3DH)**
Open a communication channel between Alice and Bob
- **Double Ratchet Protocol**
Exchange message exchange with forward secrecy

Signing-up with Signal



- Users sign-up with the signal server that verifies their identity (email, phone number, 2FA and so on)
- Users upload an identity key (public) and several pre-keys (public) signed with their identify key

Opening a channel between Alice and Bob



Triple Diffie-Hellman Key Exchange (3DH)



Ik_A

$ECDH_1$

$ECDH_2$

Ik_B



Ek_A

$ECDH_3$

SPk_B

$ECDH(Ik_A, SPk_B)$

$ECDH(Ek_A, SPk_B)$

$k_{root} = HKDF(ECDH_1, ECDH_2, ECDH_2)$

$ECDH(Ek_A, Ik_B)$

Double Ratchet Protocol

- The root key k_{root} **does not encrypt messages**
 - Use **for deriving a series of keys** $k_1, k_2, k_3 \dots$ that will be used to encrypt (send) and decrypt (receive) each message m_1, m_2, m_3
- ➔ **Double Ratchet Key Derivation Protocol**
(more details in bonus challenge)

Conclusion

Limitation of secure channels

