Cryptography Protocols

Thierry Sans

Security goals vs attacker's model

Interception
Modification
Fabrication
Interruption

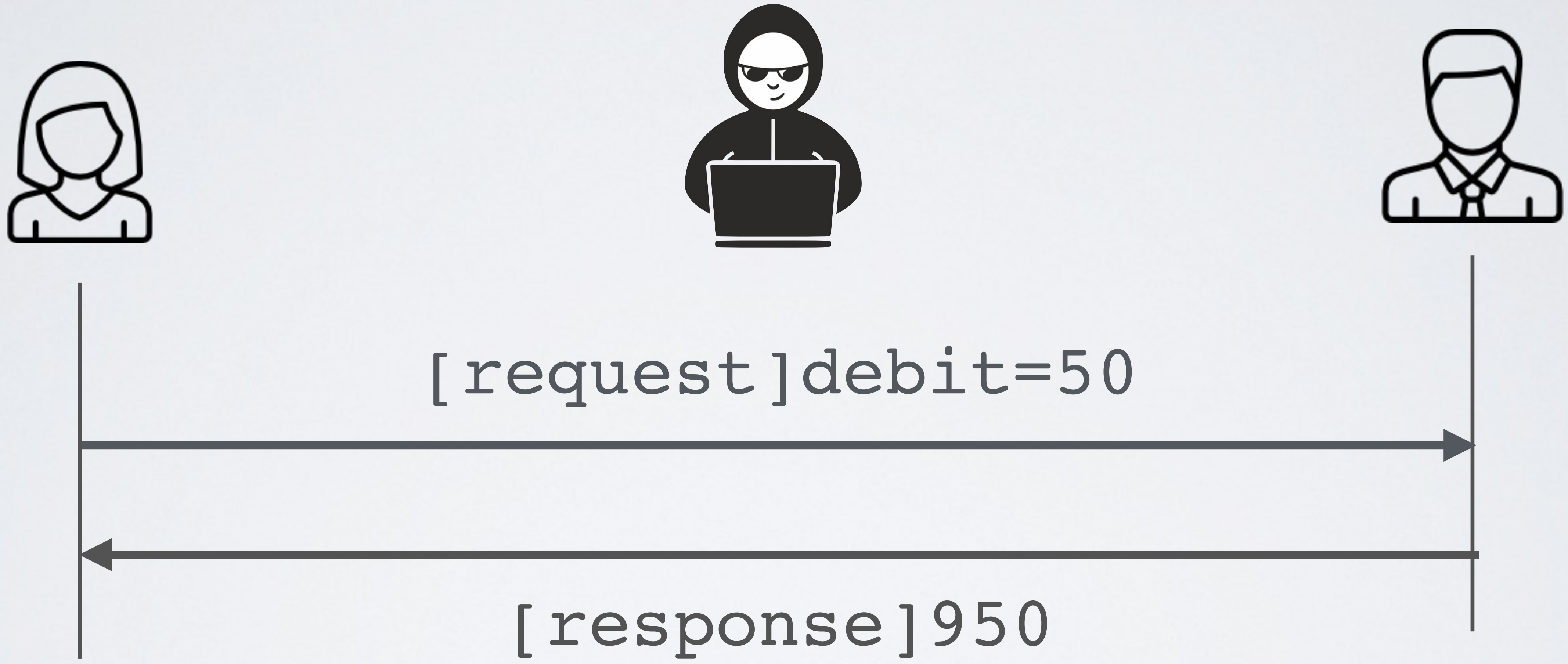
Let us consider **confidentiality, integrity** ~~**and availability**~~

Design of a cryptography protocol

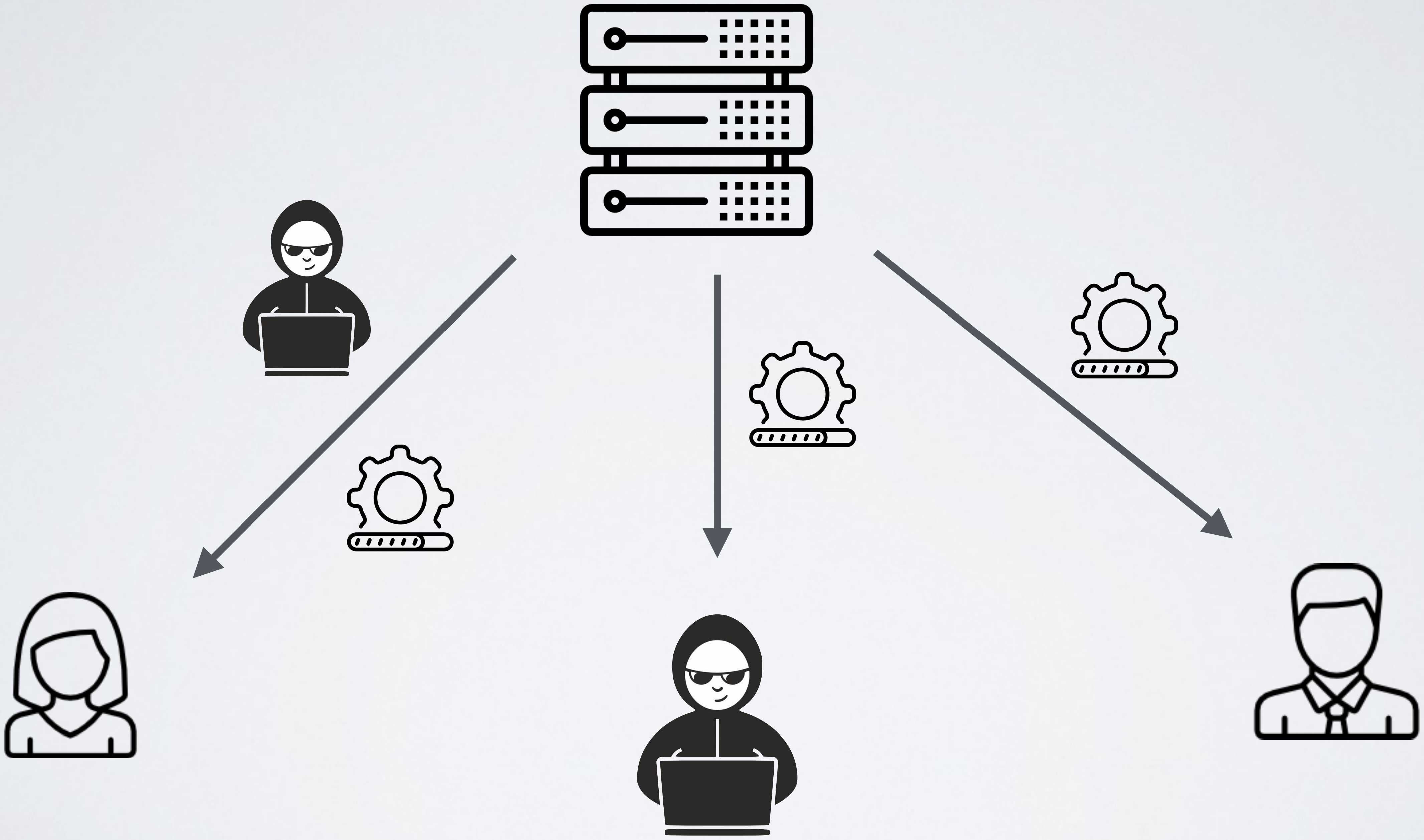
The hypothesis on the system

- **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

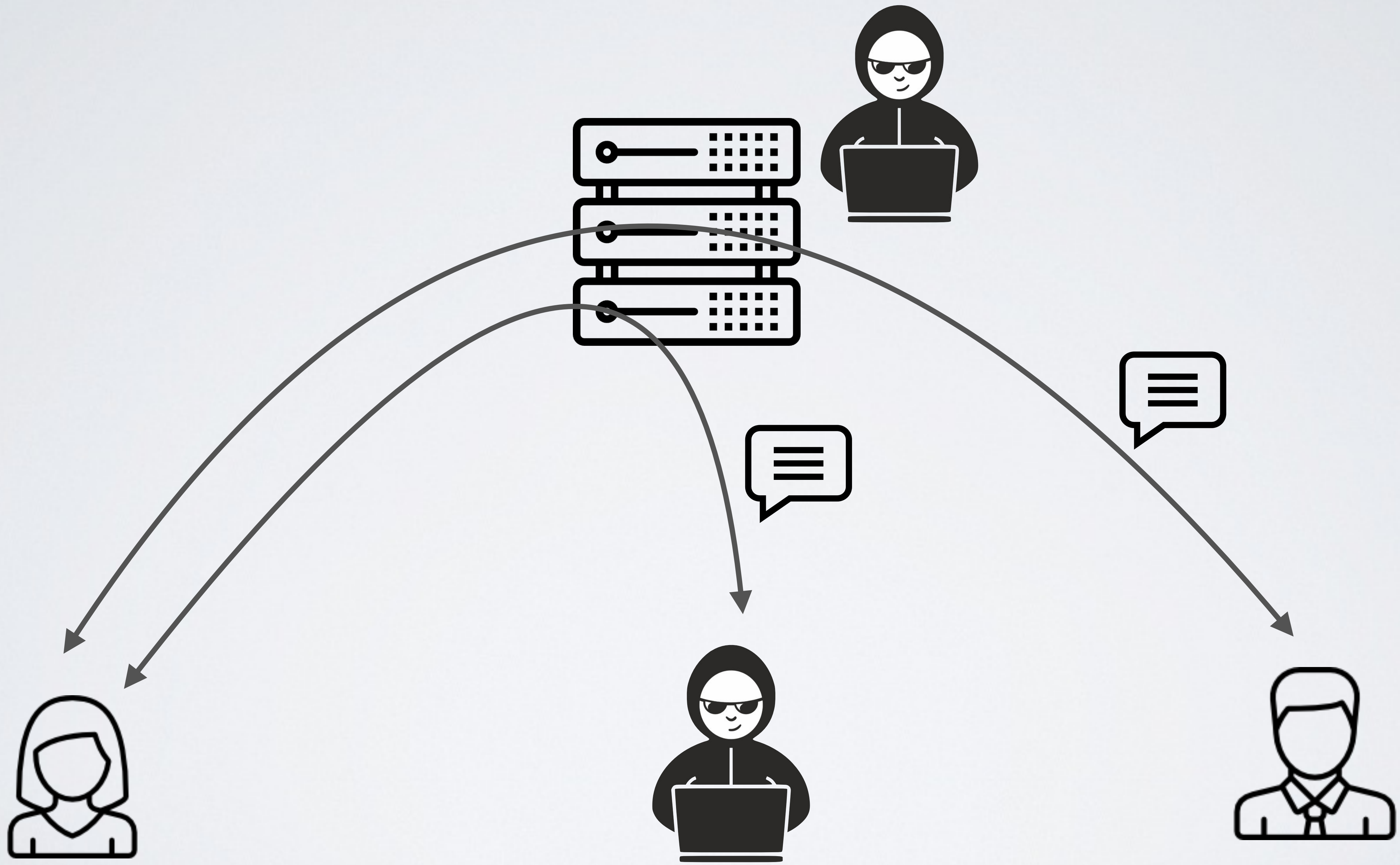
Example 1 - **Interactive Protocol**



Example 2 - **Distribution Centre**

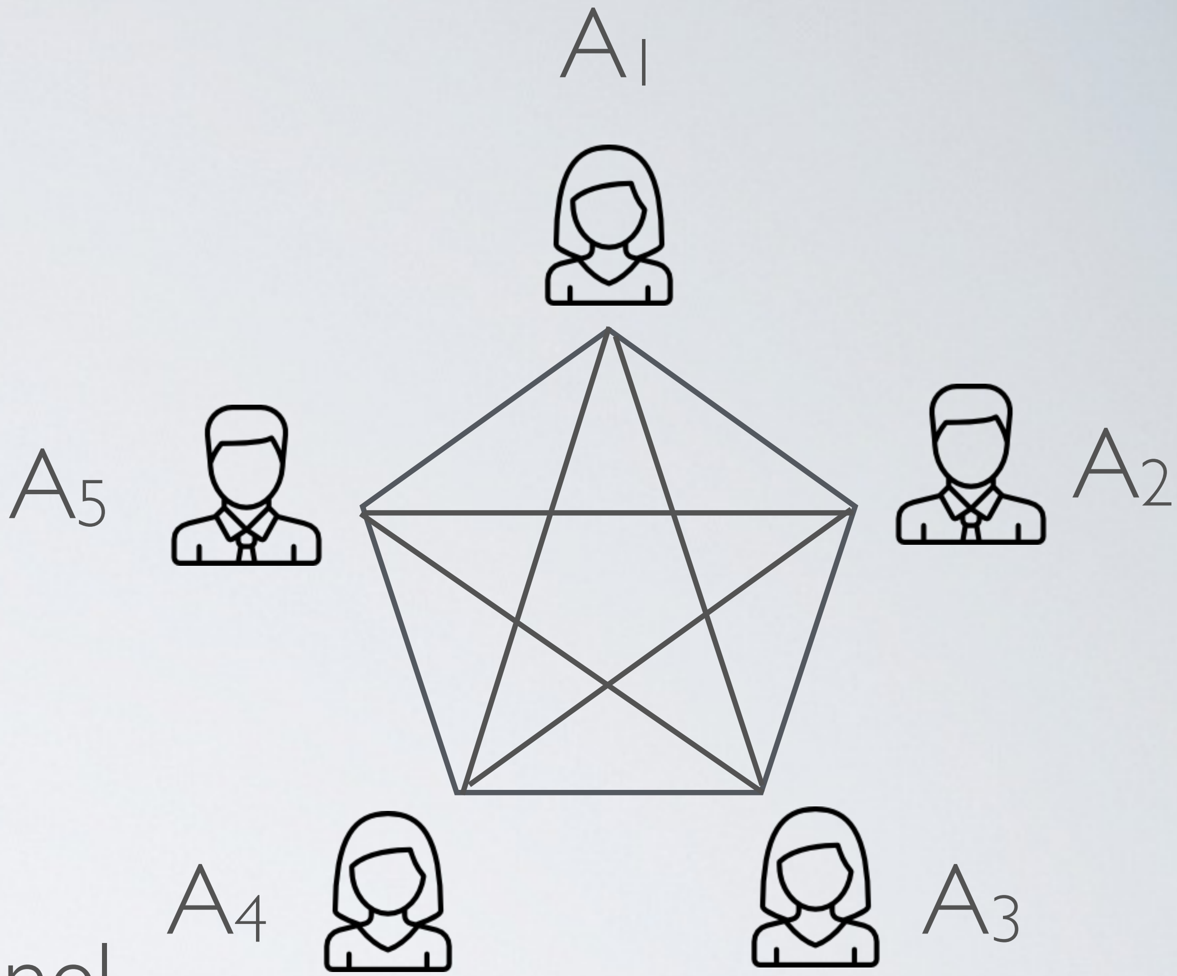


Example 3 - **Asynchronous Messaging**



Replay attacks

Interactive Protocol



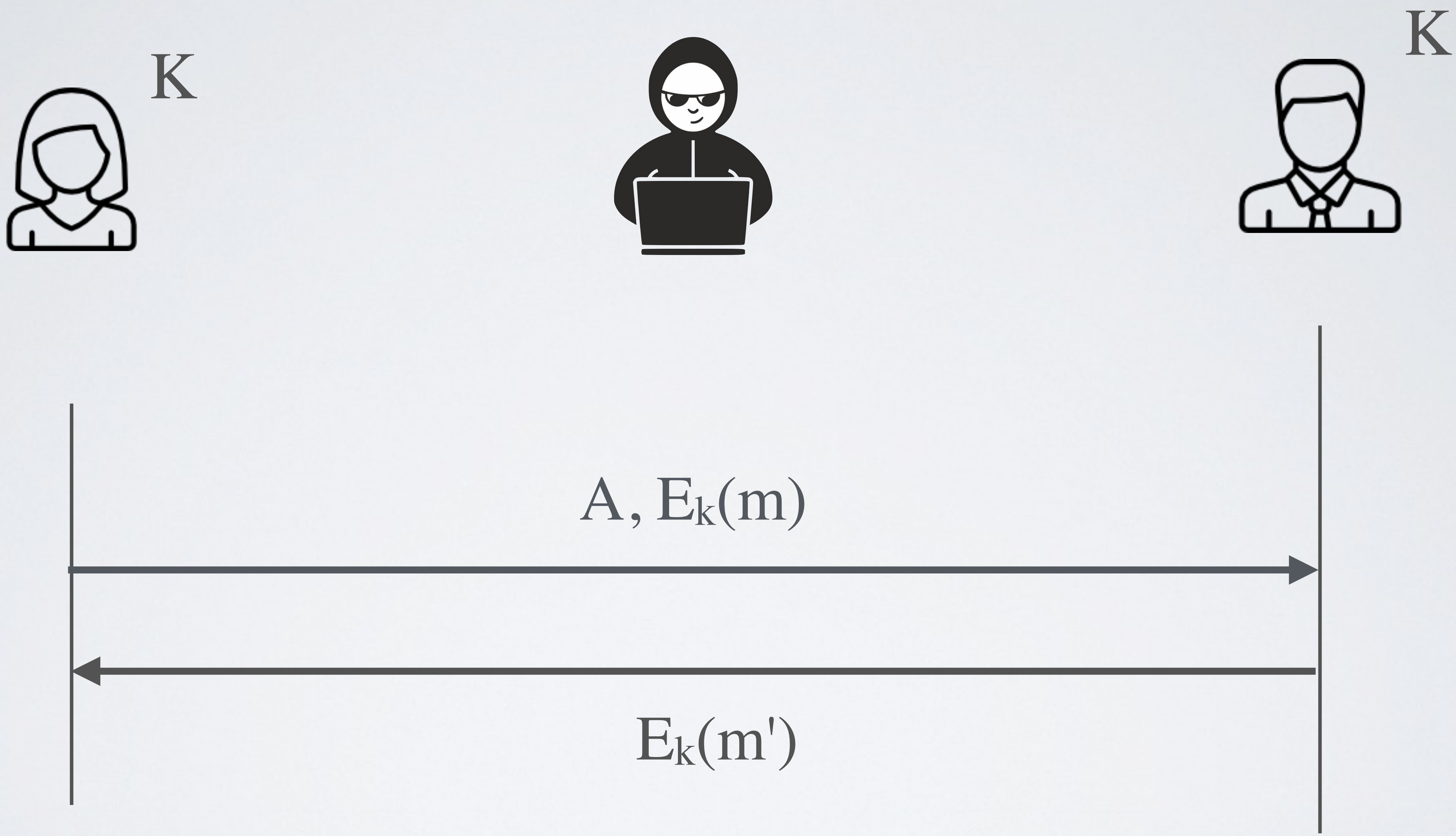
System Hypothesis

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

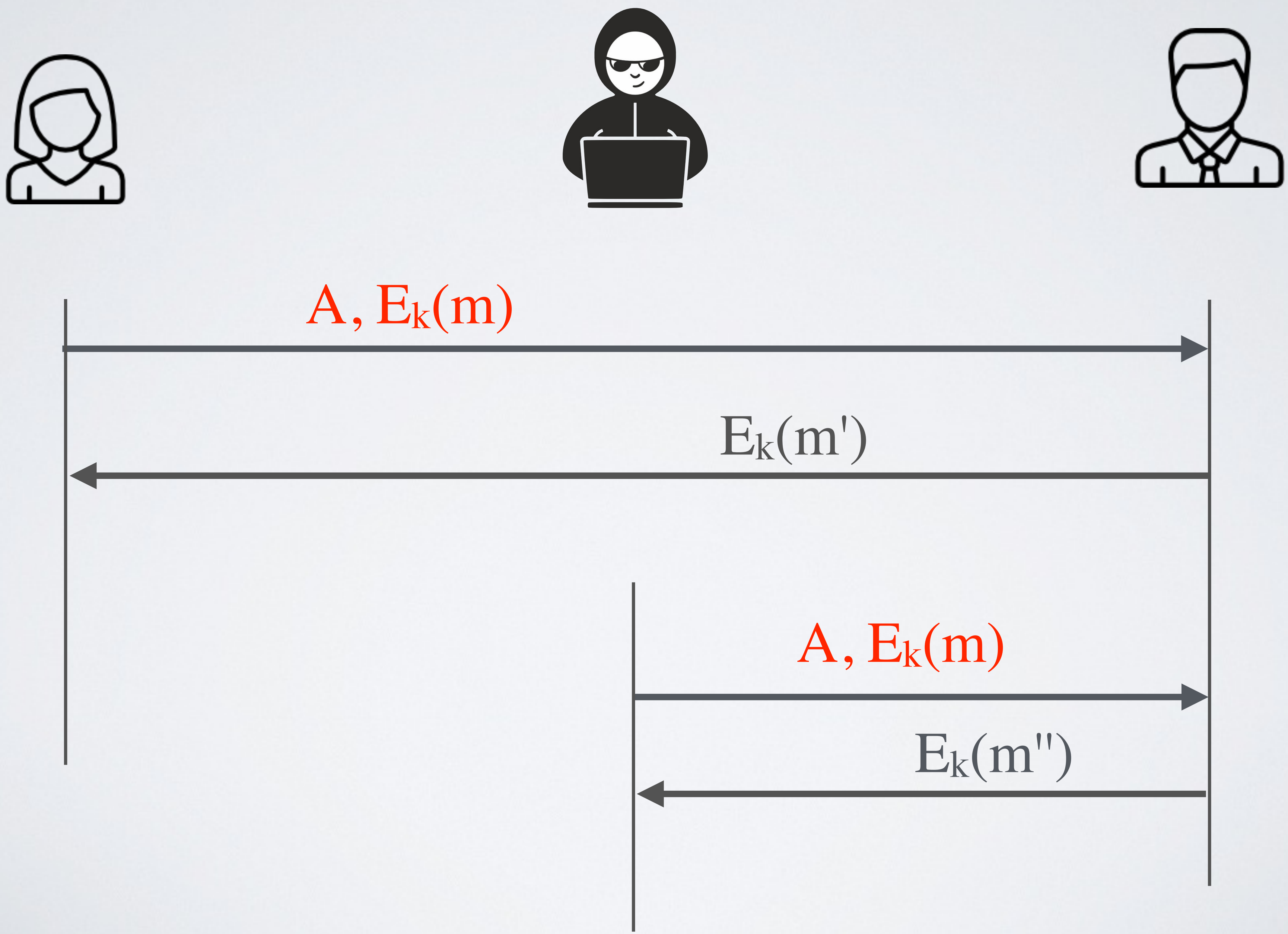
Goals

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

Using Authenticated Encryption



Problem : replay attack

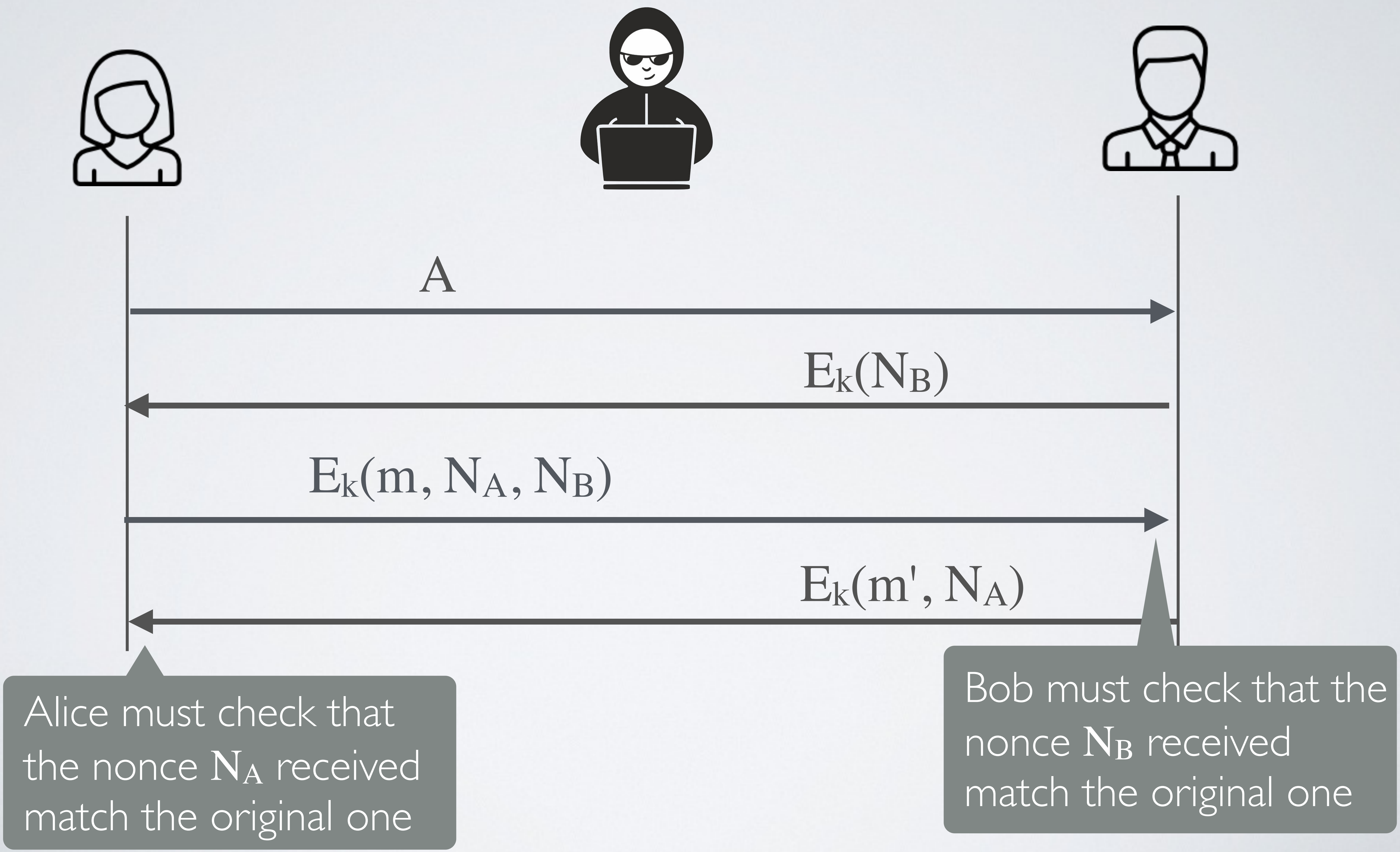


Counter replay attacks

- ✓ **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



Are we secure yet?

Two major issues:

1. **Key distribution**
   If A1, A2 … A5 want to talk, then **n . (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

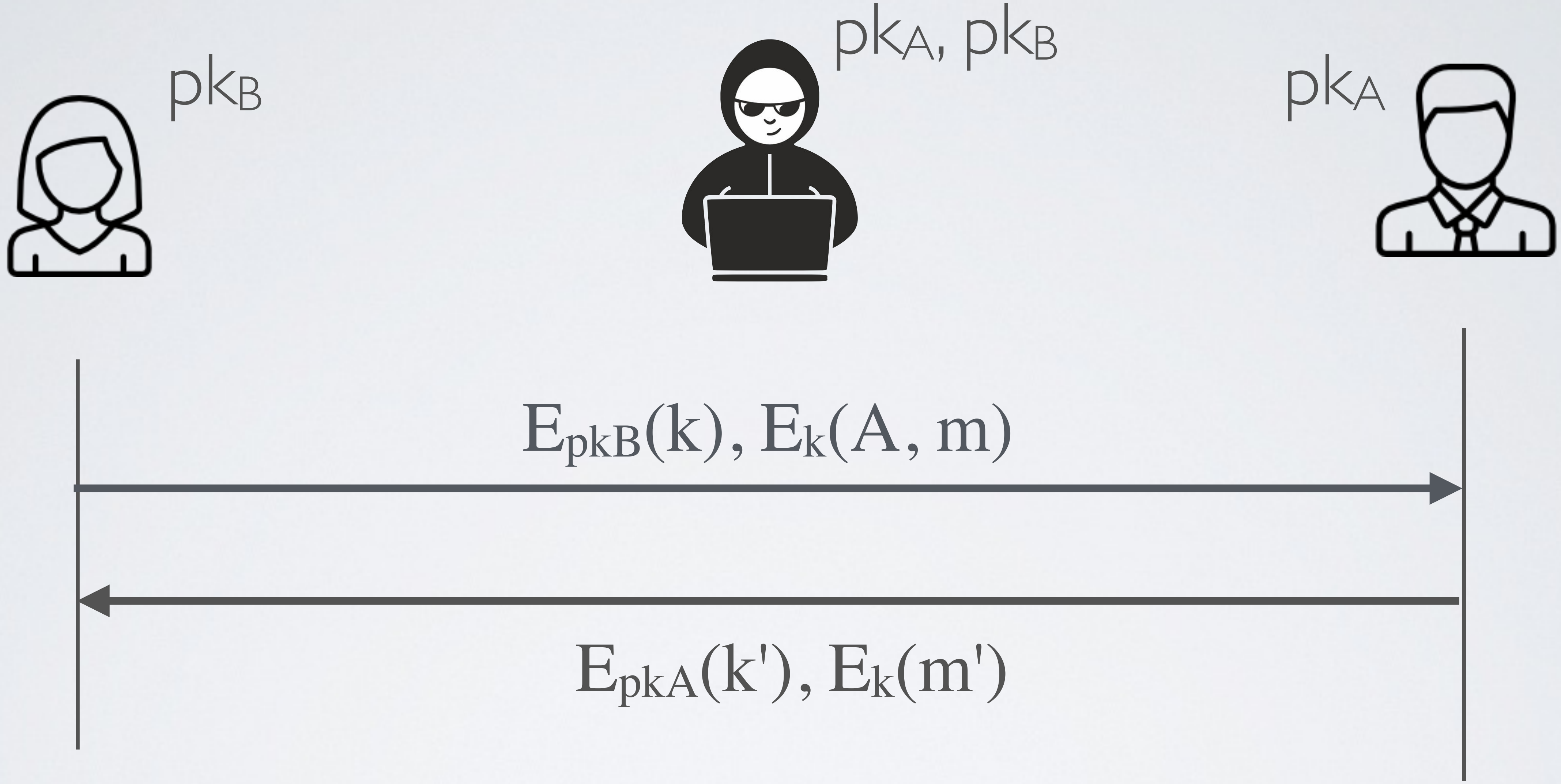
System Hypothesis

- 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

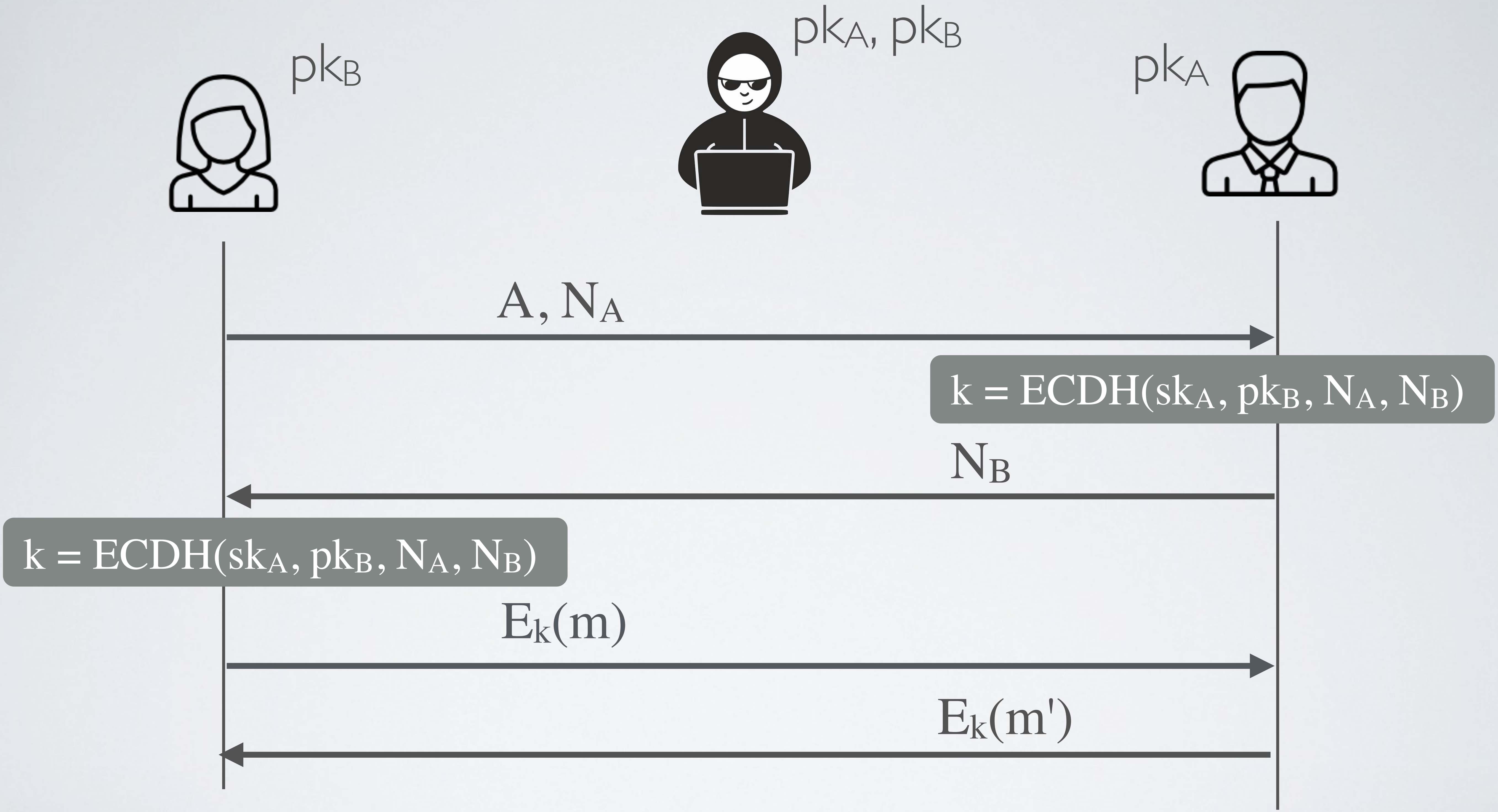
- When two participant exchange a message, the system should protect the confidentiality, integrity **and perfect forward secrecy** of the messages

[broken] Key Wrapping



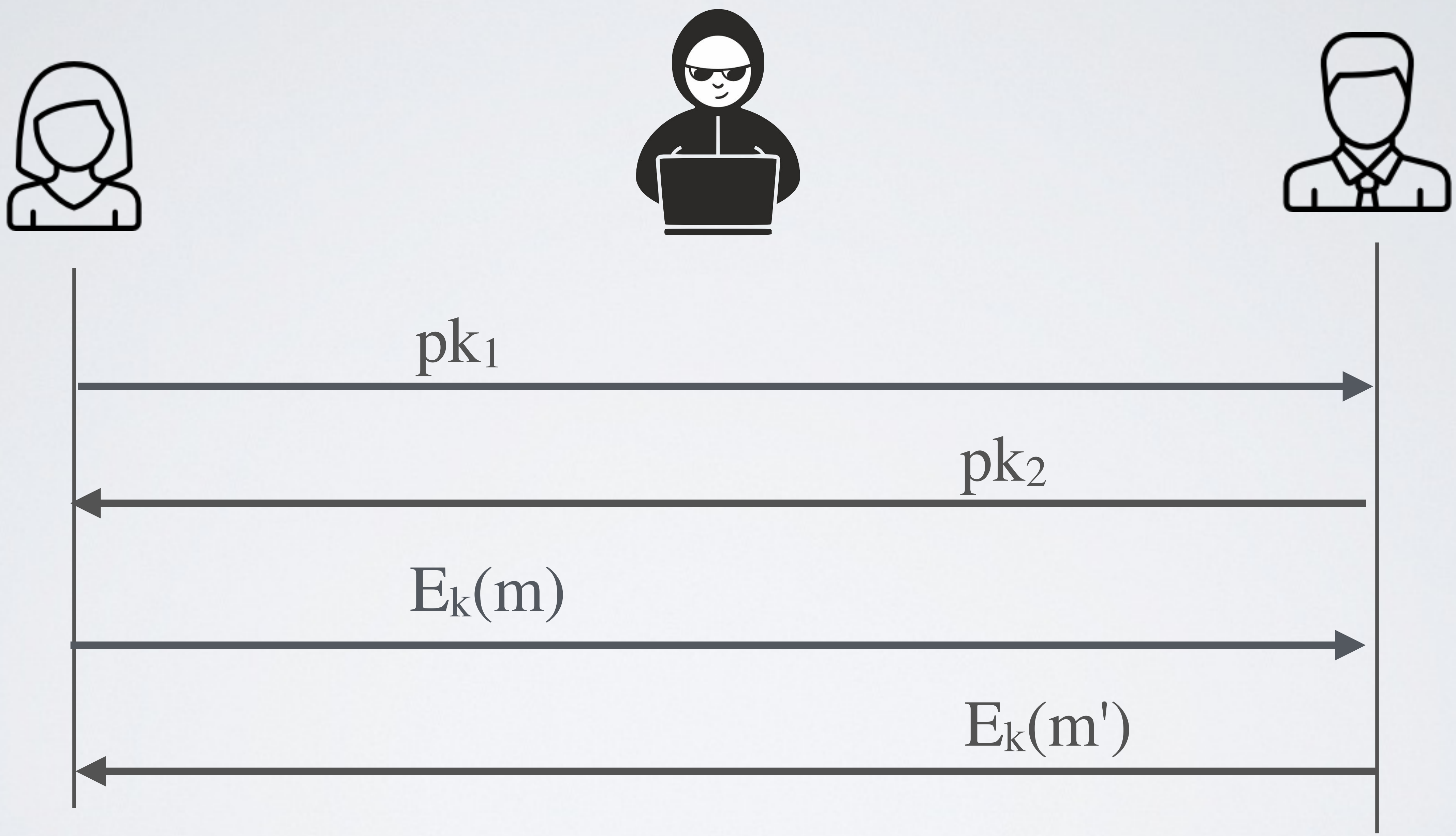
- Replay attack
- No Forward Secrecy
- Bad Authentication

[broken] Key Derivation using Long-Term Keys



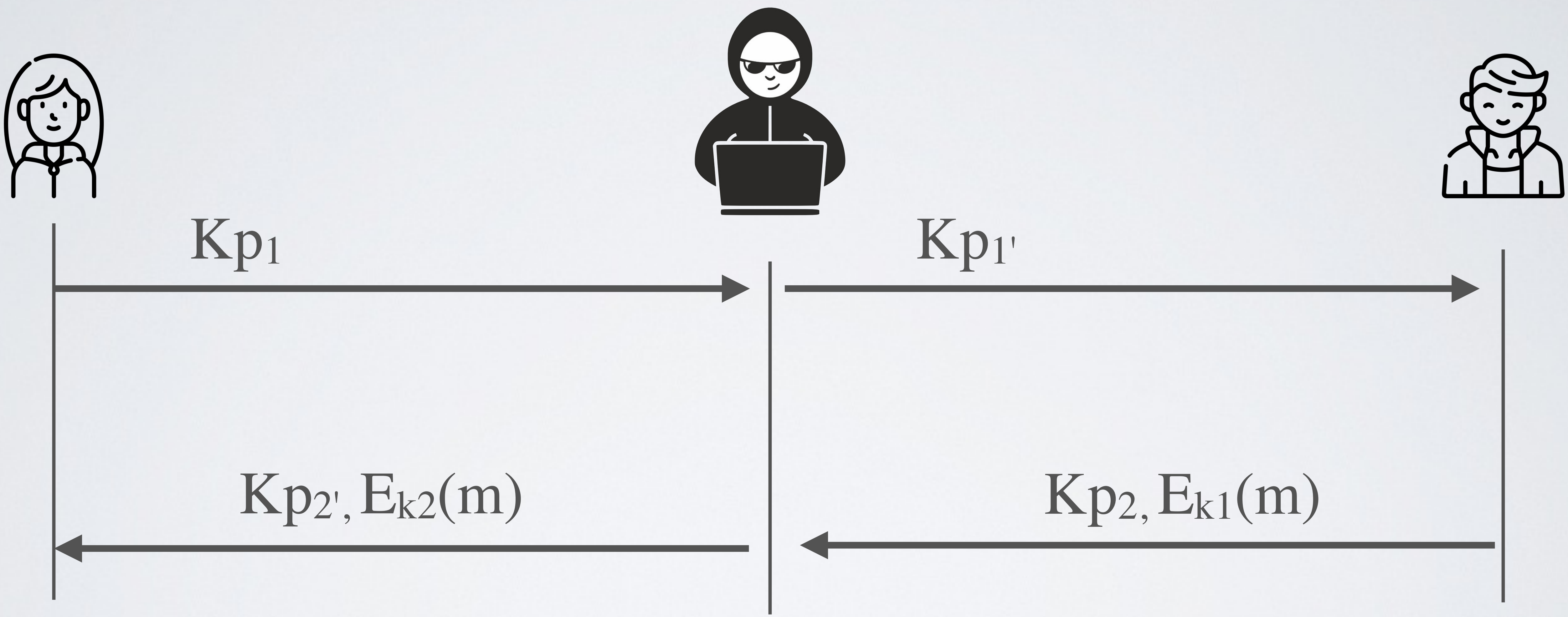
- No Forward Secrecy

[broken] Key Derivation using Short-Term Keys

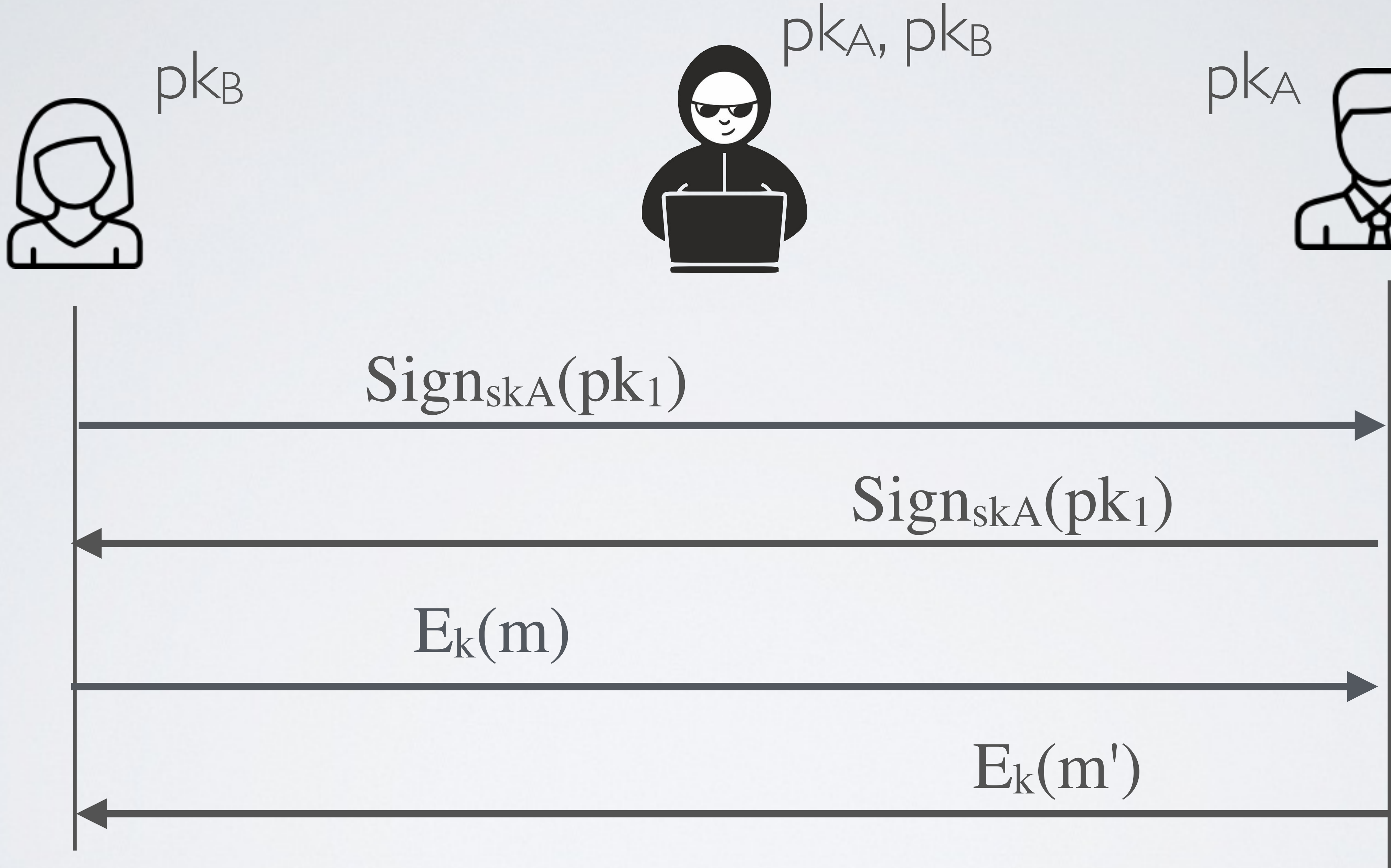


$$k = ECDH(sk_1, pk_2) = ECDH(sk_2, pk_1)$$

The key exchange is not authenticated



Key Derivation with Authenticated Short-Term Keys

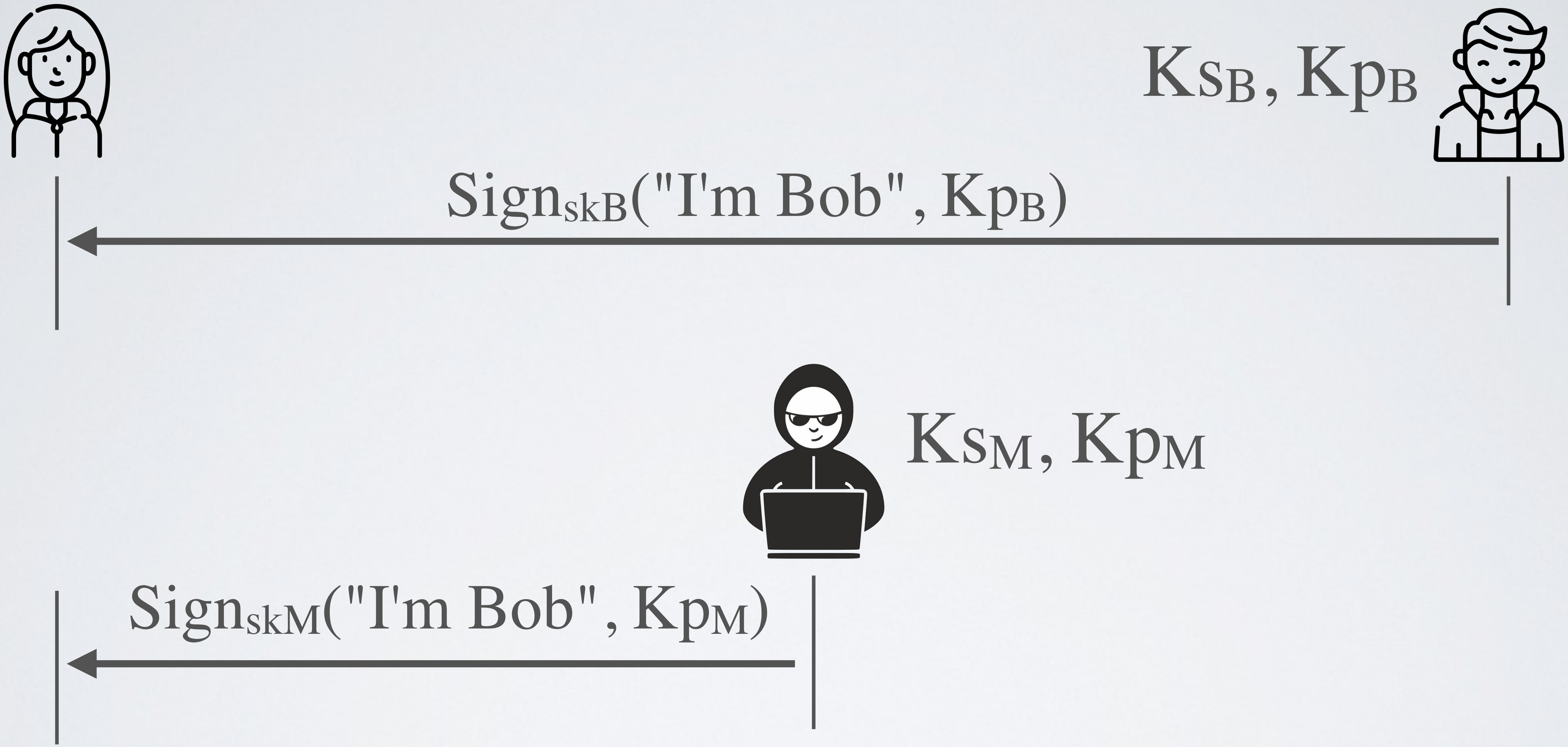


One major issue

Key distribution

If A1, A2 … A5 want to talk, then **n** public keys must be distributed physically to each user using a secure channel

Why not?



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 ?

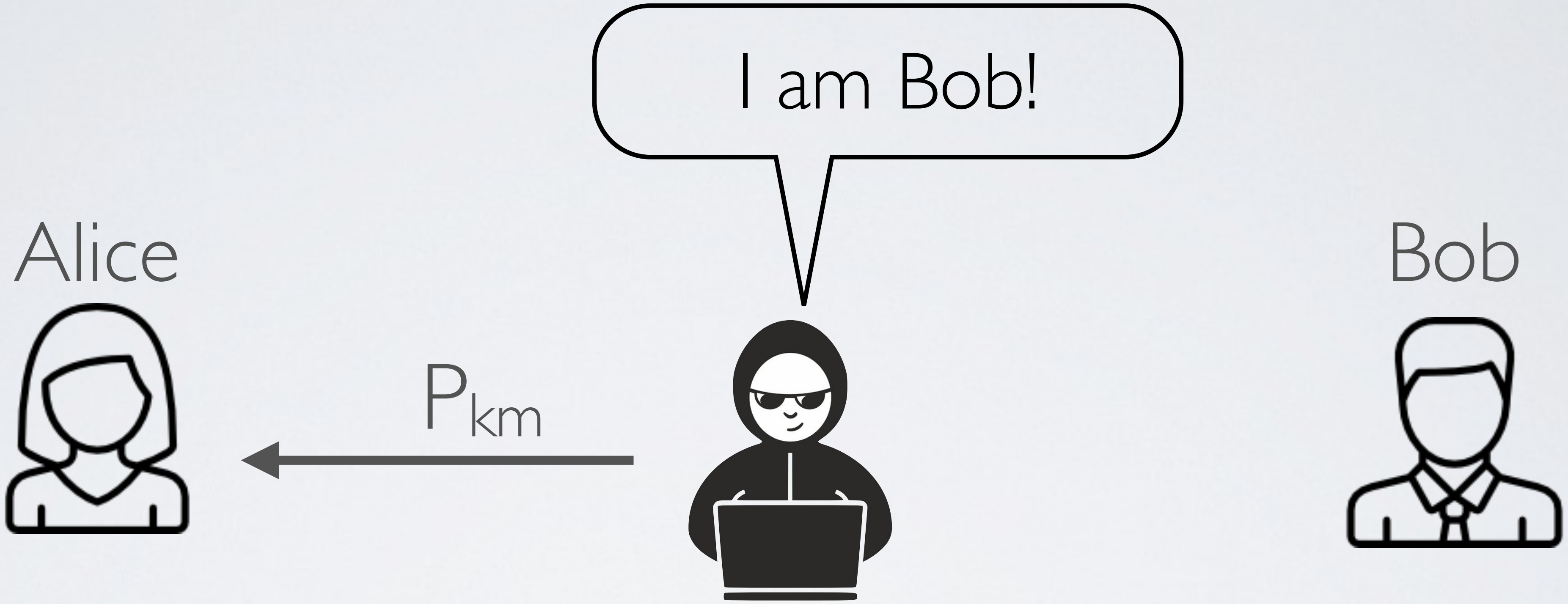
Decentralized trust model

➡ **Web of Trust**

Centralized trust model

➡ **PKI - Public Key Infrastructure**

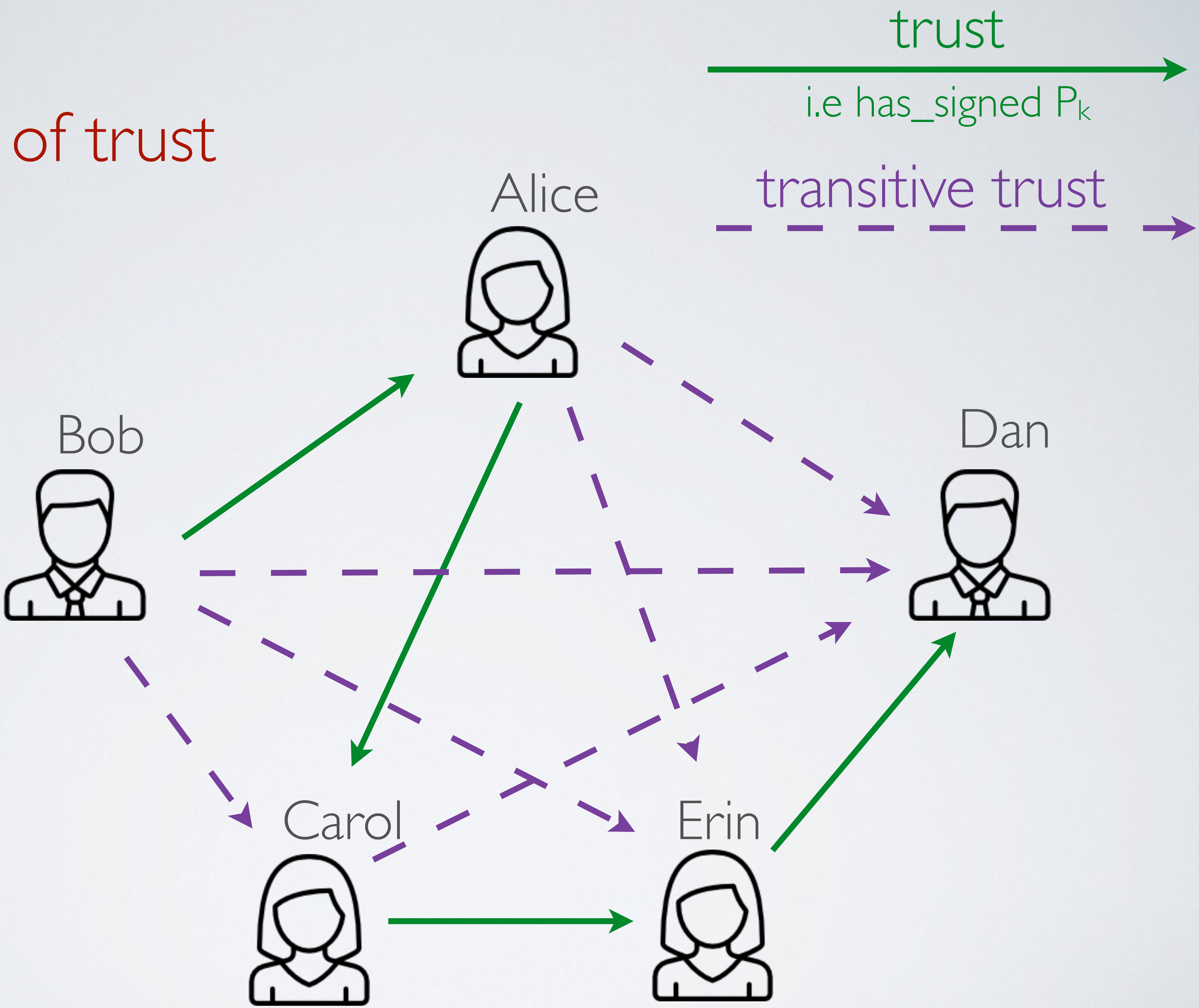
Do you trust the GPG key ?



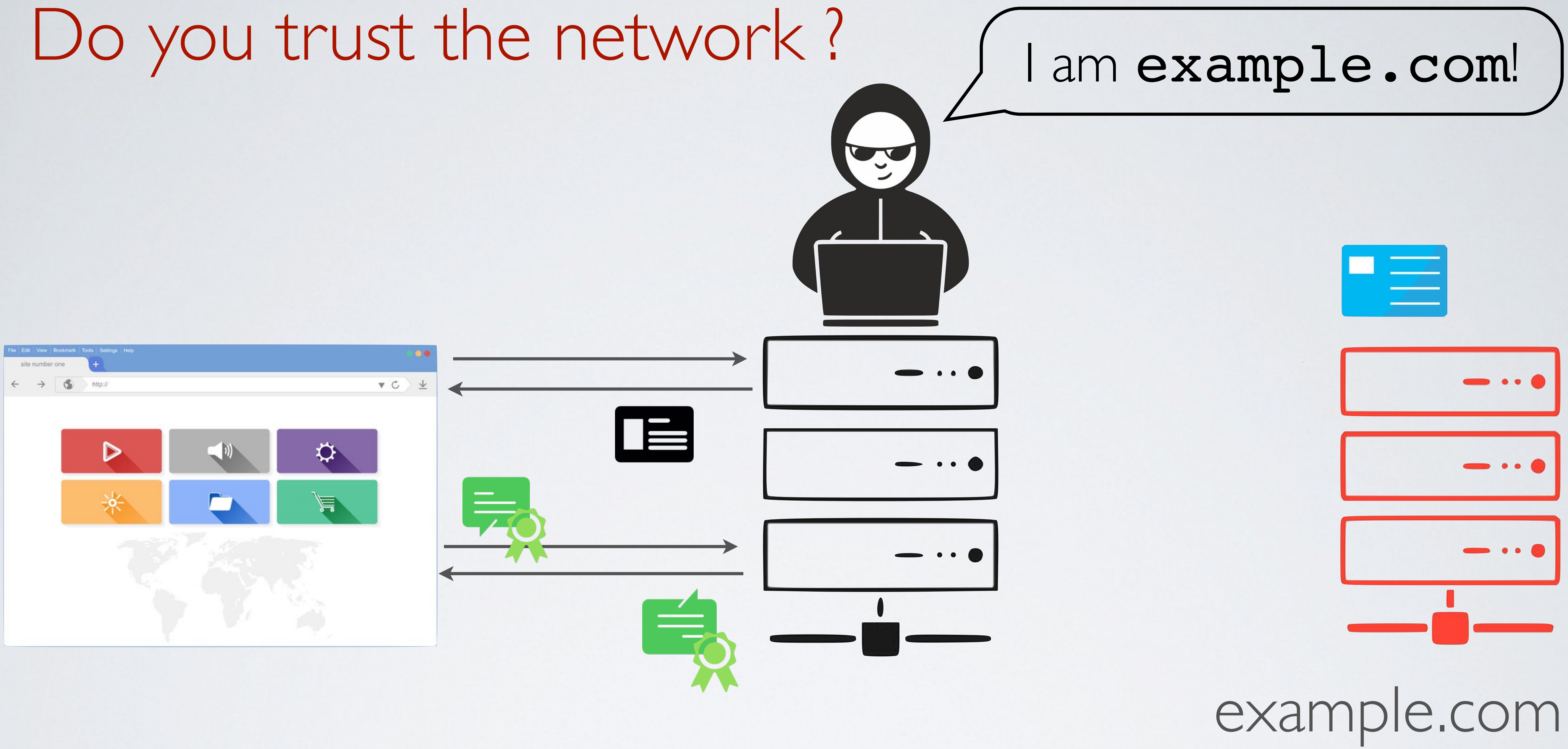
Alice should verify Bob's public key fingerprint

- 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 ?

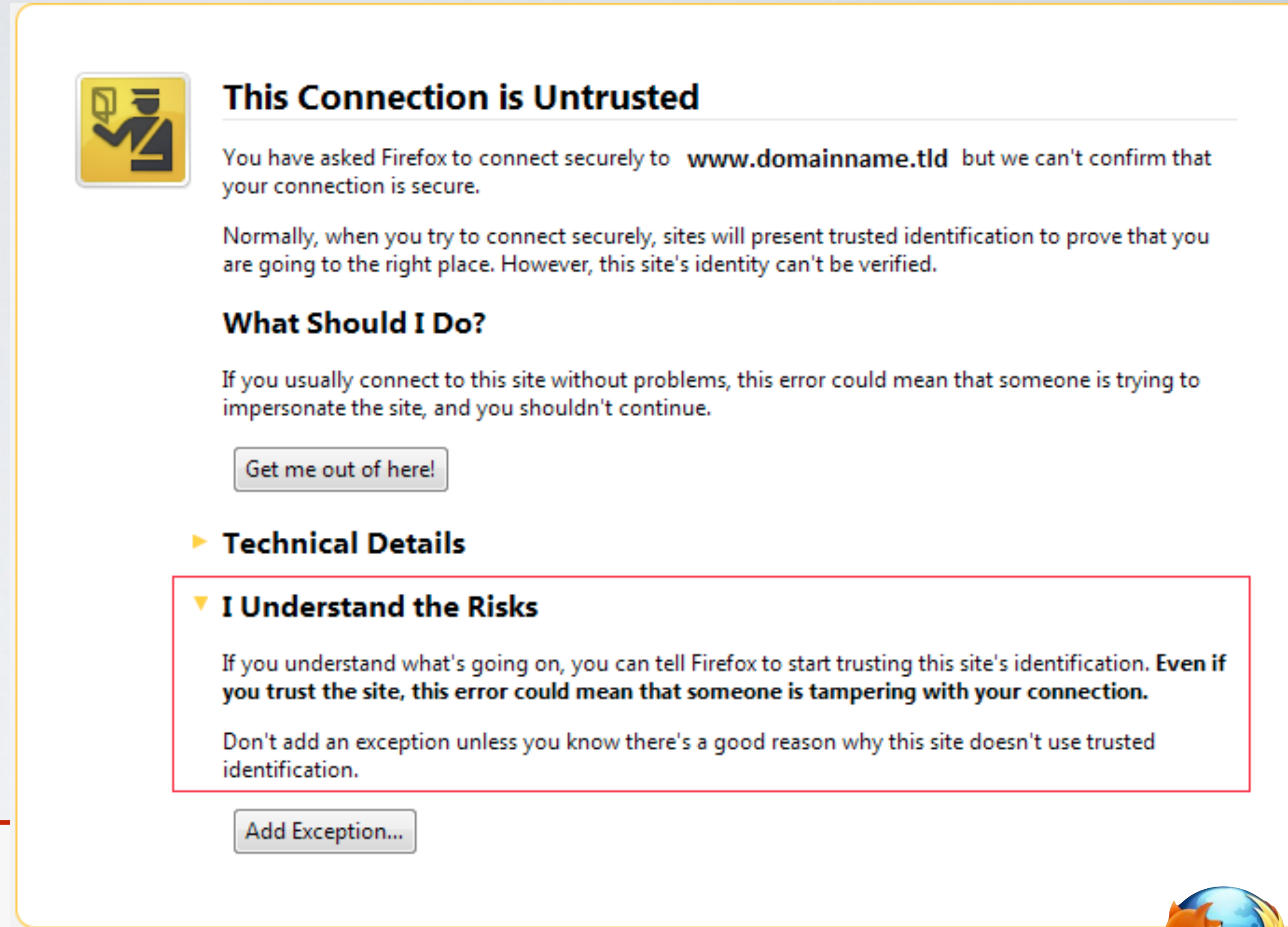


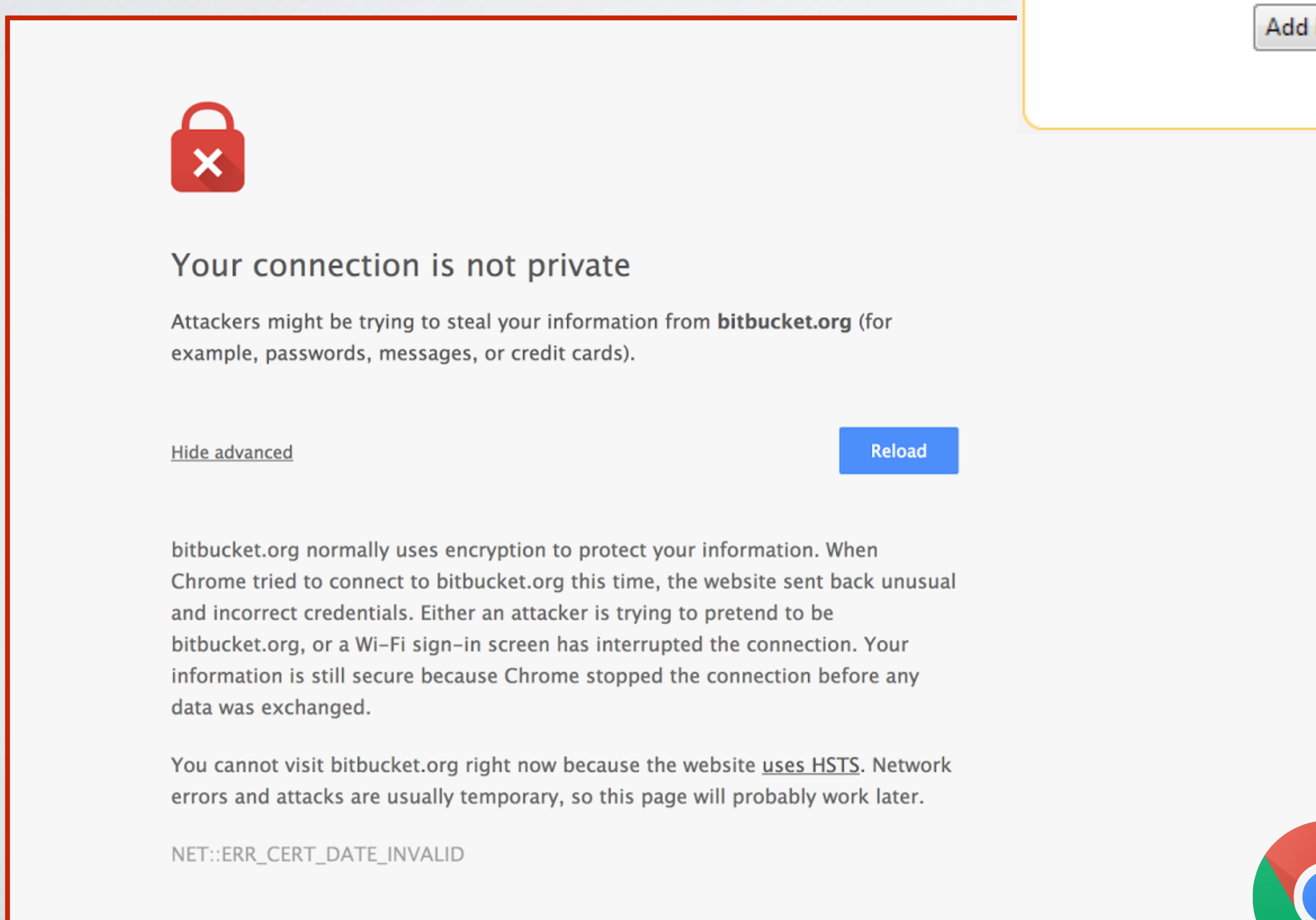
The browser should verify the certificate

➡ **PKI - Public Key Infrastructure**

Generating and using (self-signed) certificates

Self-signed certificates are not trusted by your browser

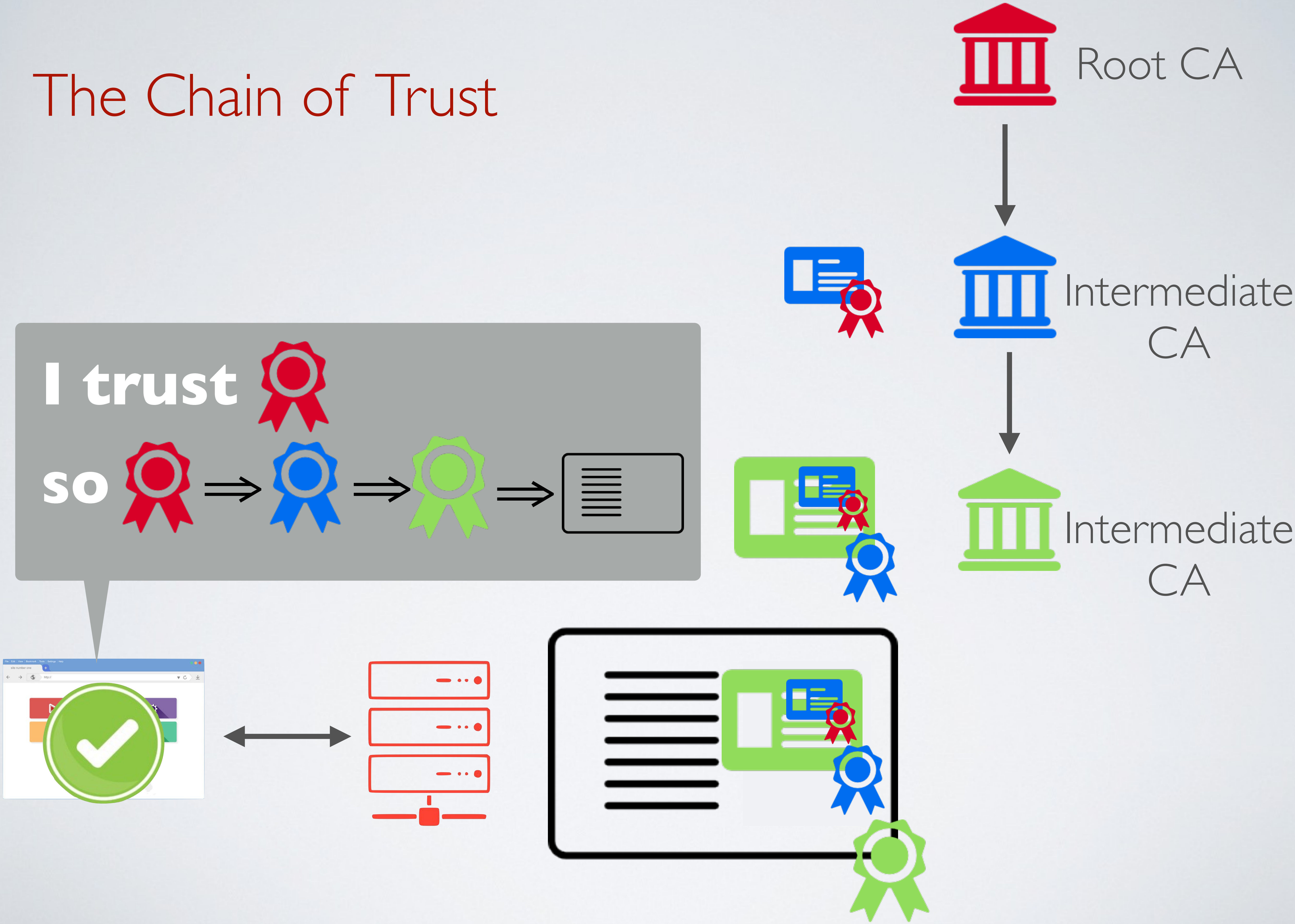




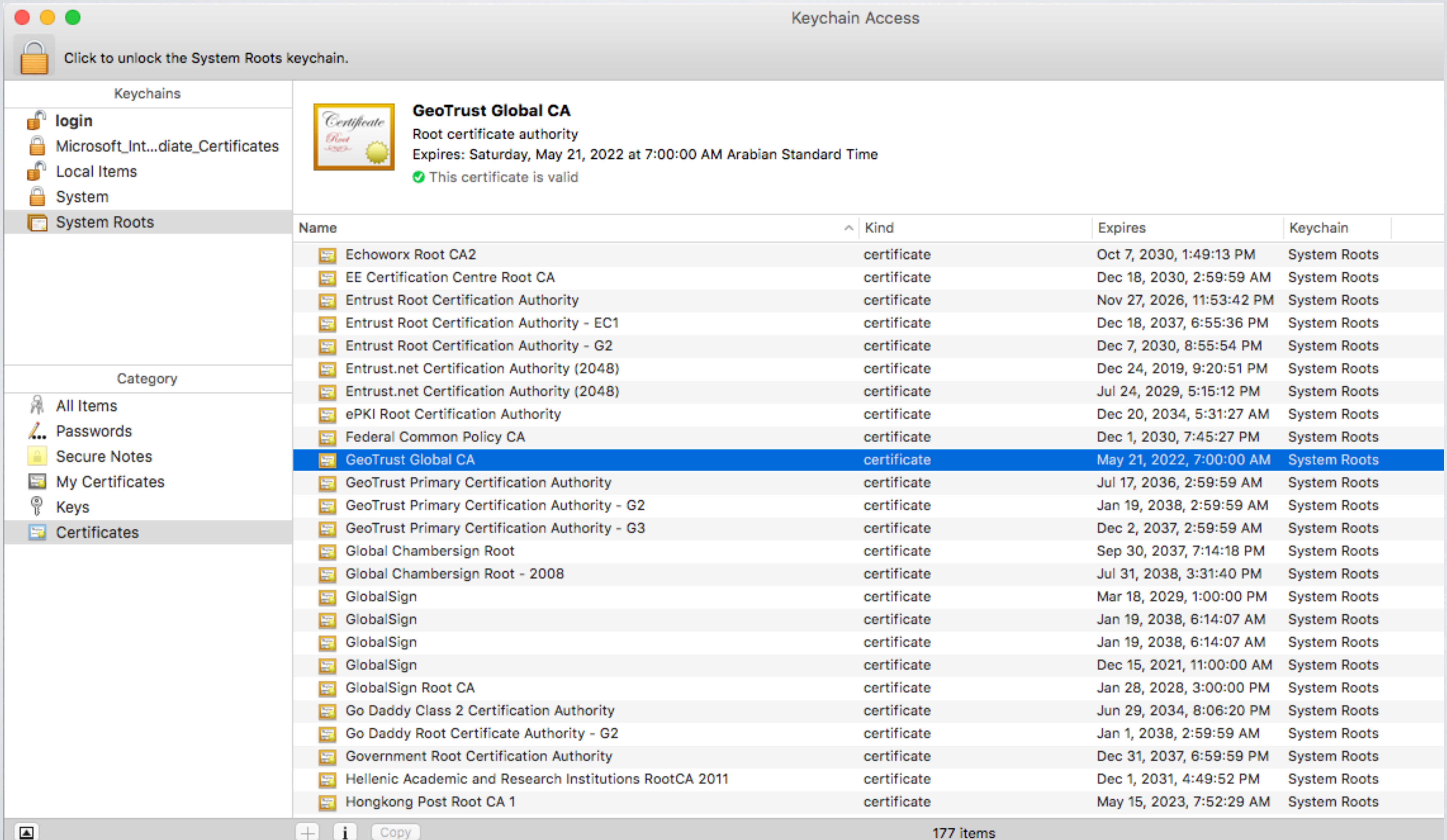
Signed Certificate

Certificate Authority (CA)

The Chain of Trust



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



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

Enhancing digital certificate security

January 3, 2013

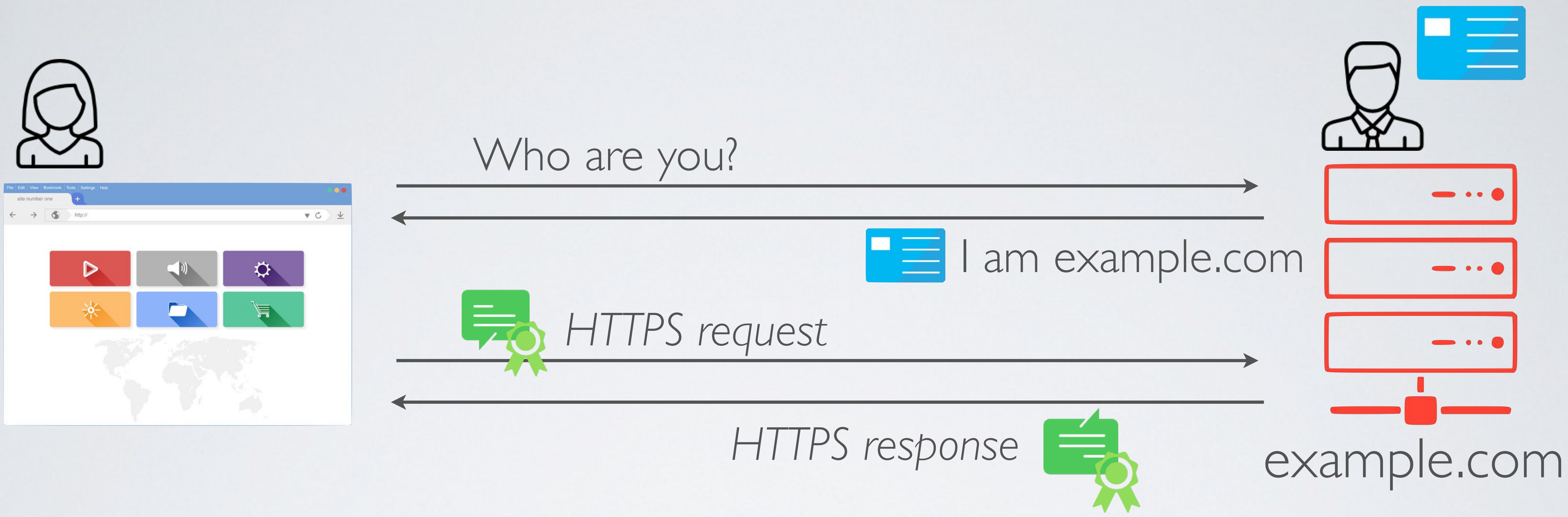
Posted by Adam Langley, Software Engineer

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

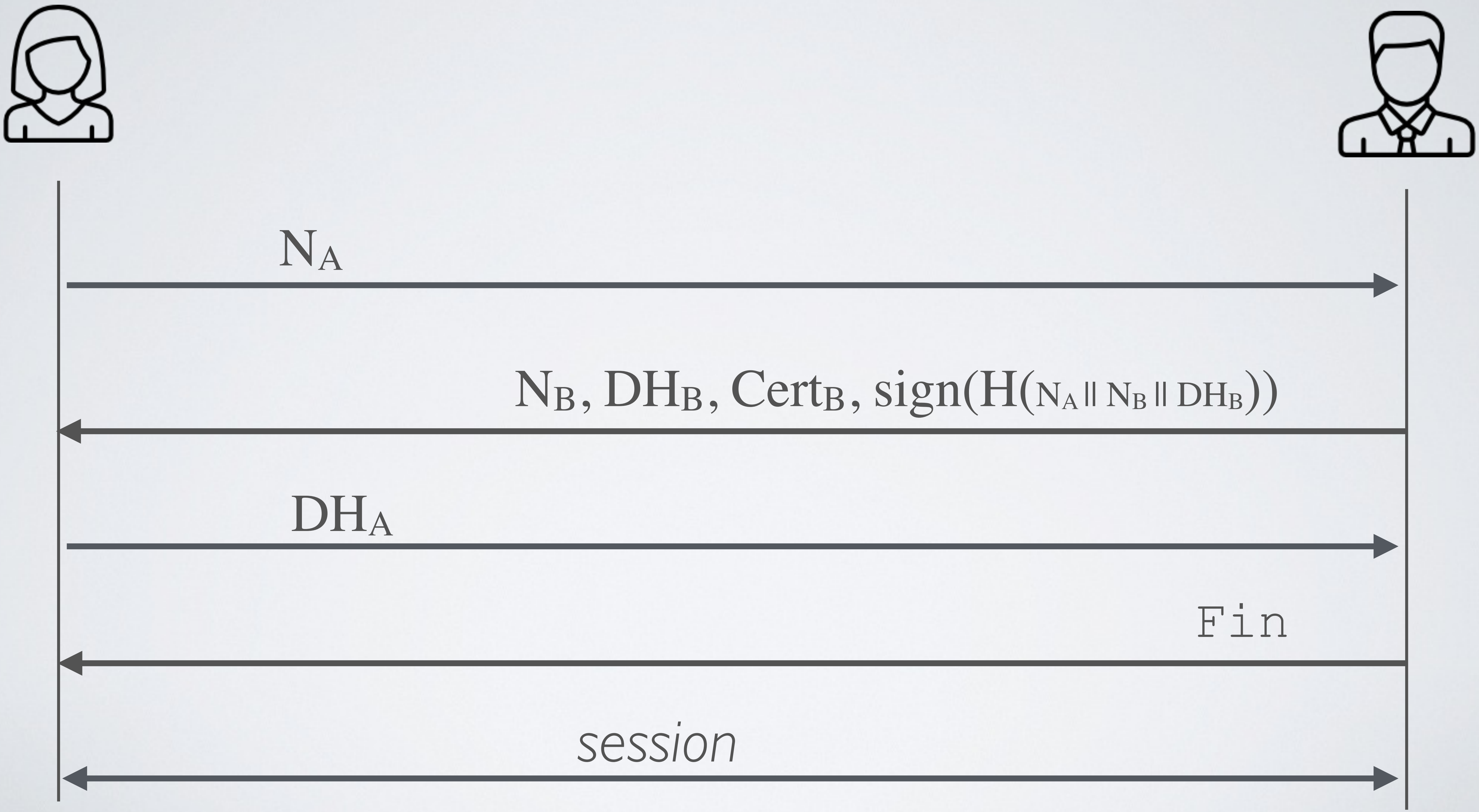


✓ **HTTPS = HTTP + TLS**

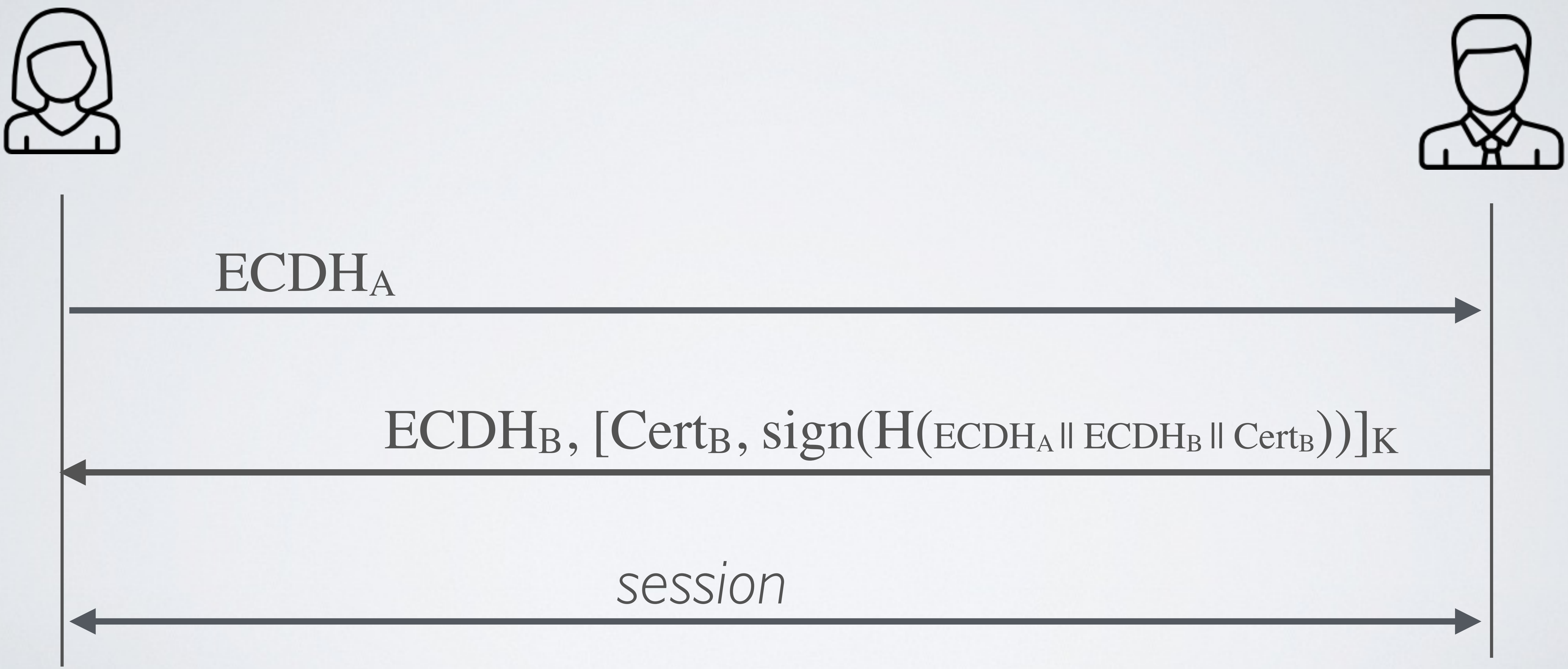
➡ TLS - Transport Layer Security (a.k.a SSL) provides

- **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
  (dos not prevent from implementation bugs)

The Signal Protocol

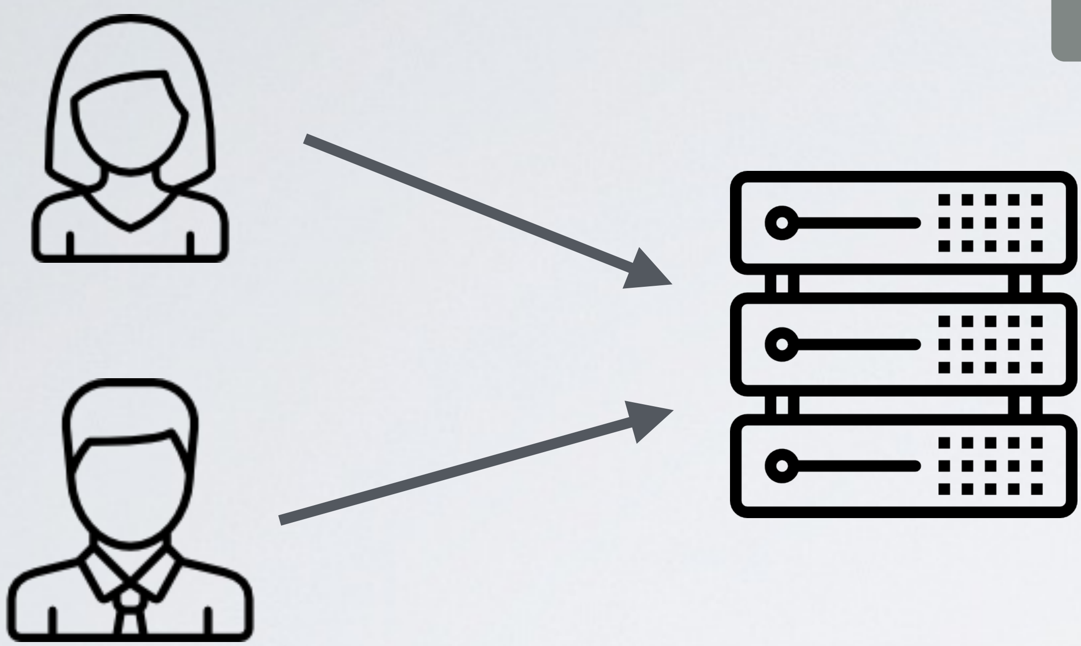
Signal in a Nutshell

➡ 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



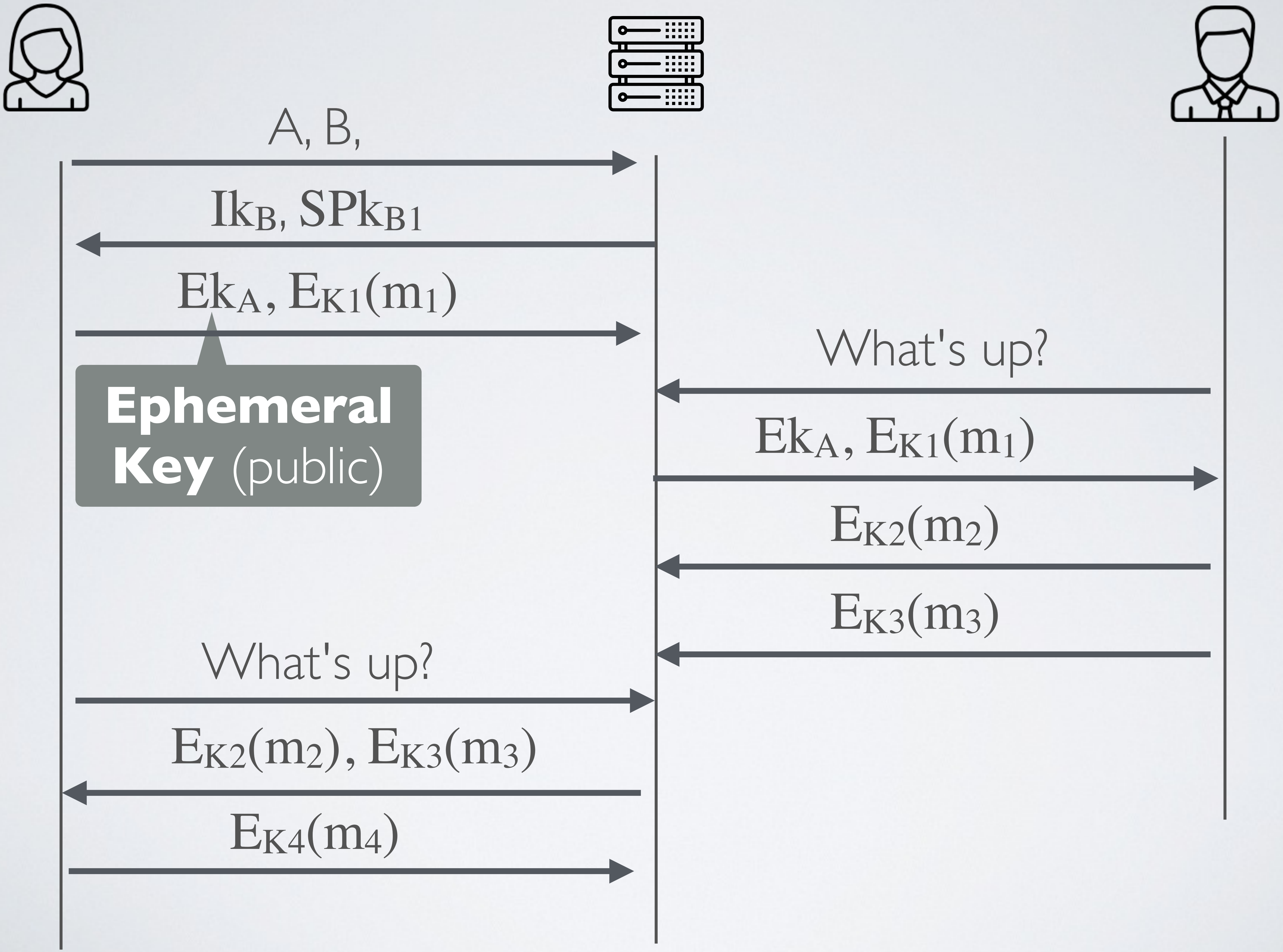
Identity key

Signed Pre-keys

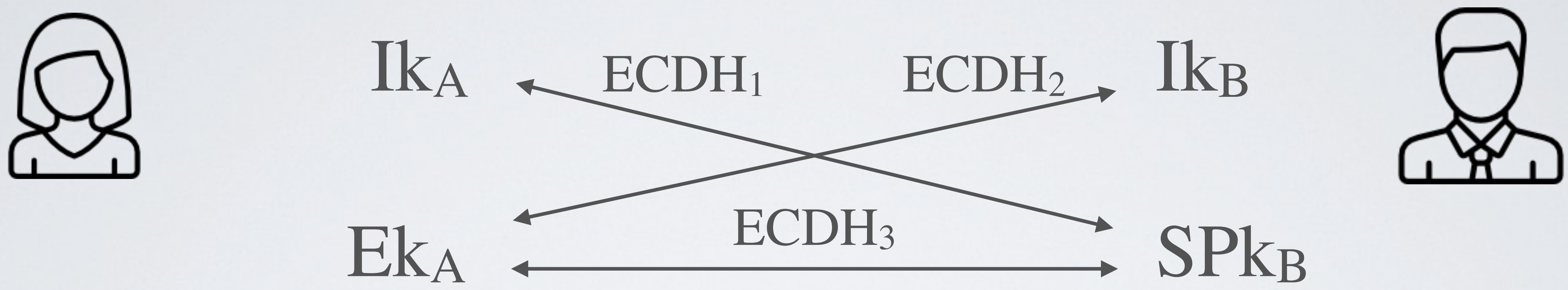
- Alice, Ik_A, $\boxed{SPk_{A1}, SPk_{A2}, SPk_{A3}}$
- Bob, Ik_B, SPk_{B1}, SPk_{B2}, SPk_{B3}
- Mallory, Ik_M, SPk_{M1}, SPk_{M2}, SPk_{M3}

- 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)



$ECDH(Ik_A, SPk_B)$

$ECDH(Ek_A, SPk_B)$

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

$ECDH(Ek_A, Ik_B)$

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 ... 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