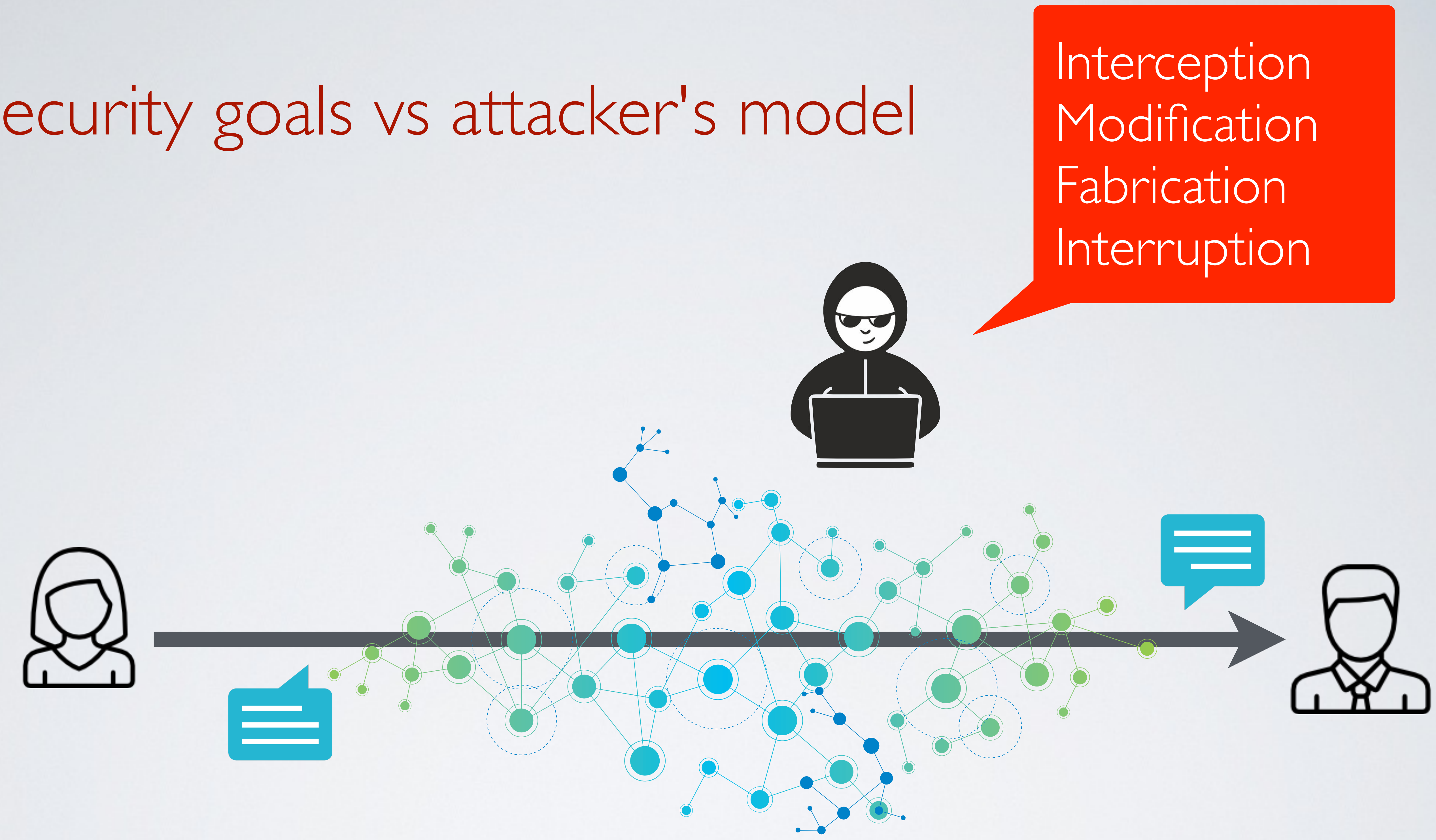
Cryptography Protocols

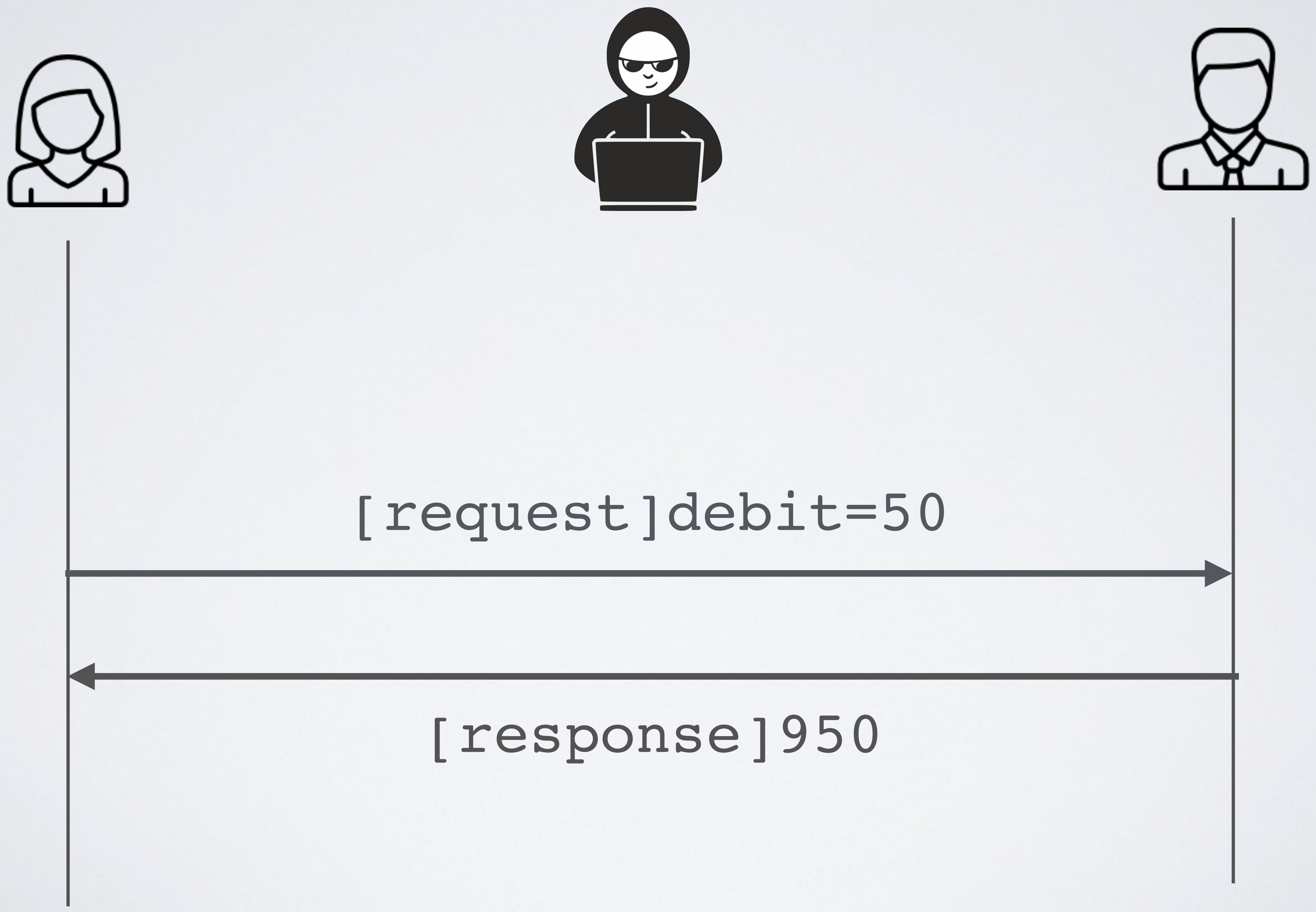
Thierry Sans

Security goals vs attacker's model

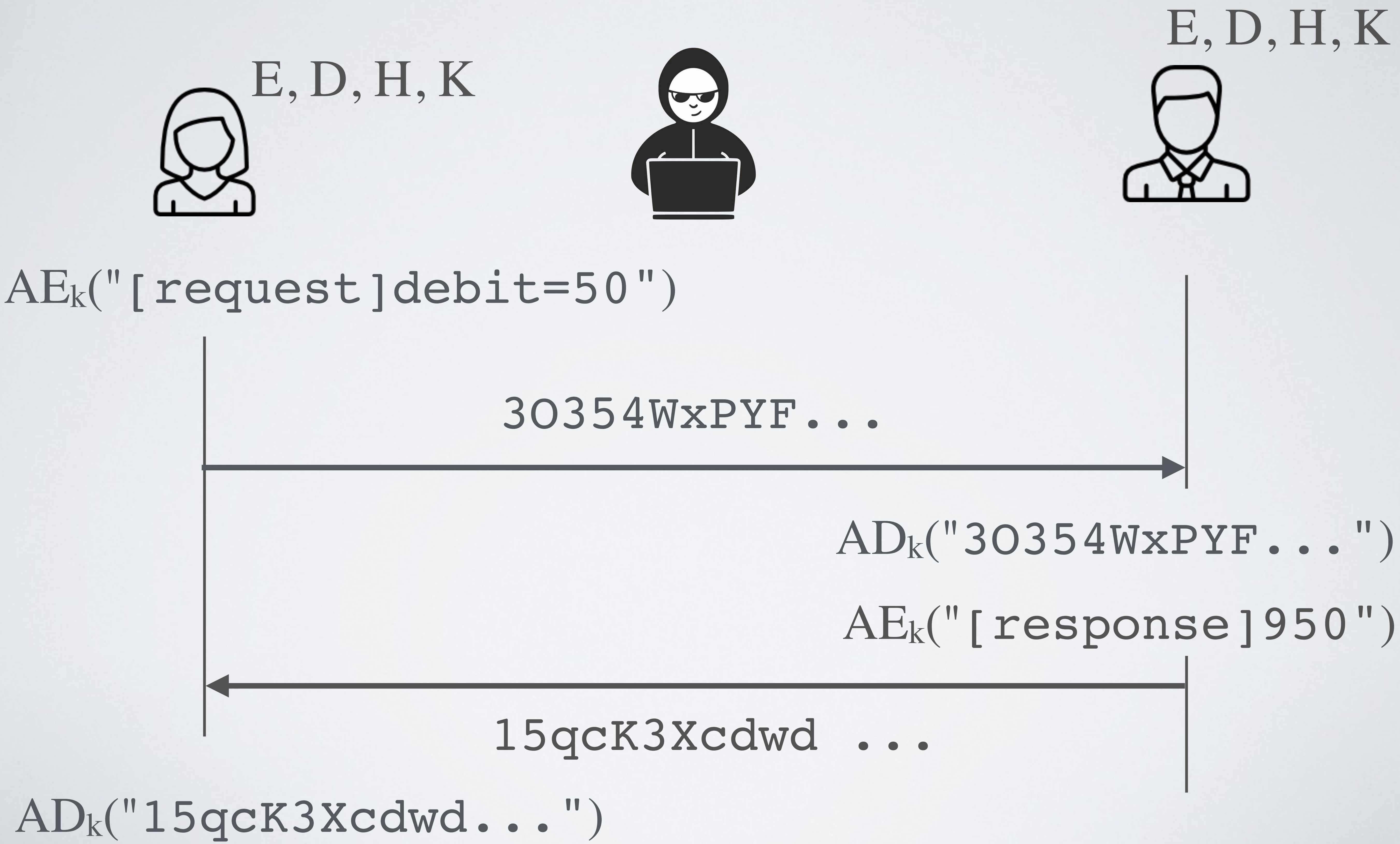


Let us consider **confidentiality, integrity ~~and availability~~**

Example



Ensuring confidentiality and integrity with Authenticated Encryption

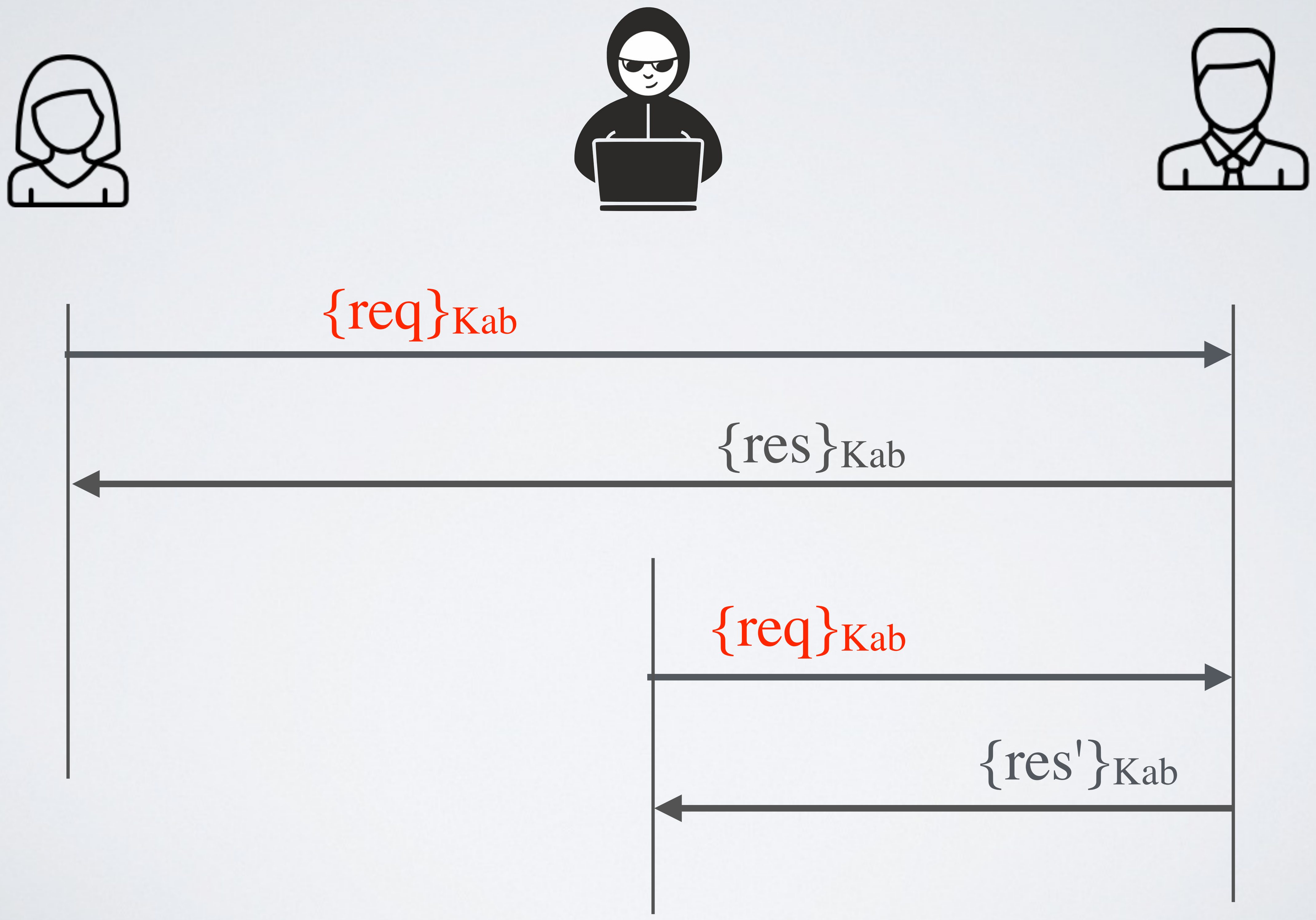


Overview

- Replay attacks in interactive protocols
- A symmetric protocol for key exchange (Needham–Schroeder Key Exchange Protocol)
- A symmetric protocol for key exchange (Diffie-Hellman-Merkle Protocol)
- The challenge of authentication (Needham–Schroeder Authentication Protocol)
- Putting it all together (TLS)
- Trust models (PKI)

Replay attacks

Replay attack

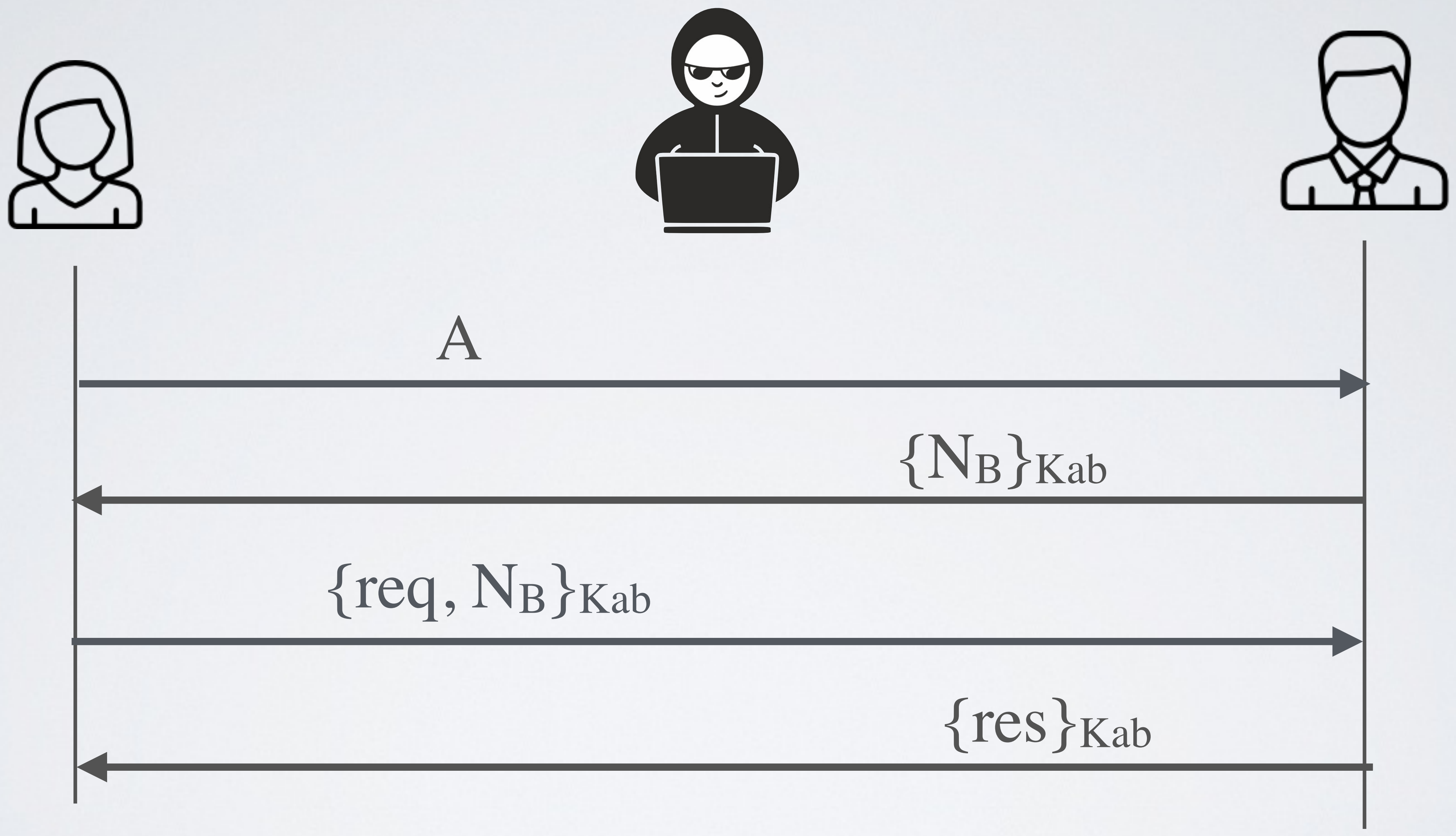


Counter replay attacks

Several solutions:

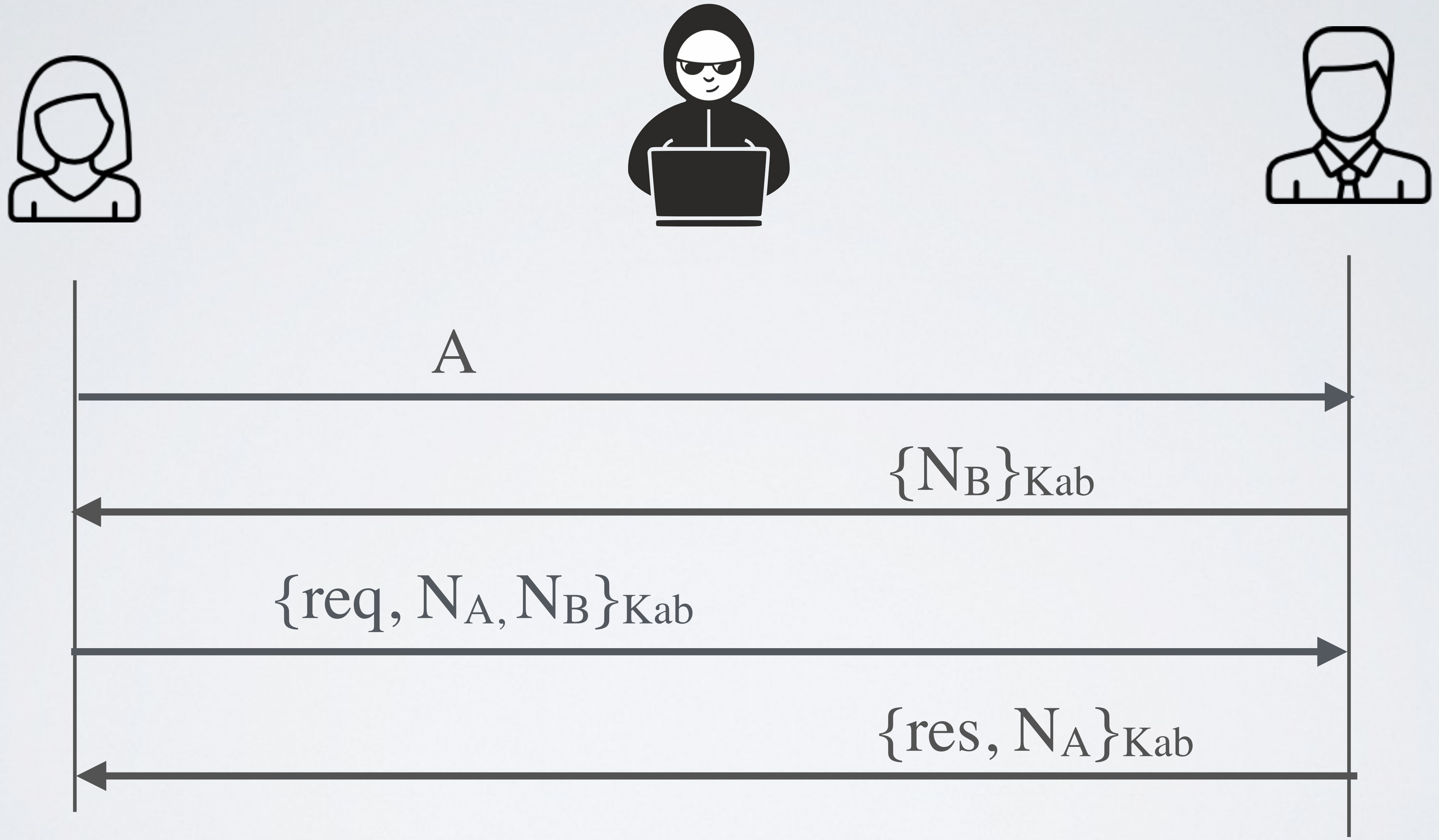
- **use a nonce (random number)**
- use sequence numbers
- use timestamps
- have fresh key for every transaction
  (key exchange problem)

Defeat replay attack with a nonce (not fully secured)



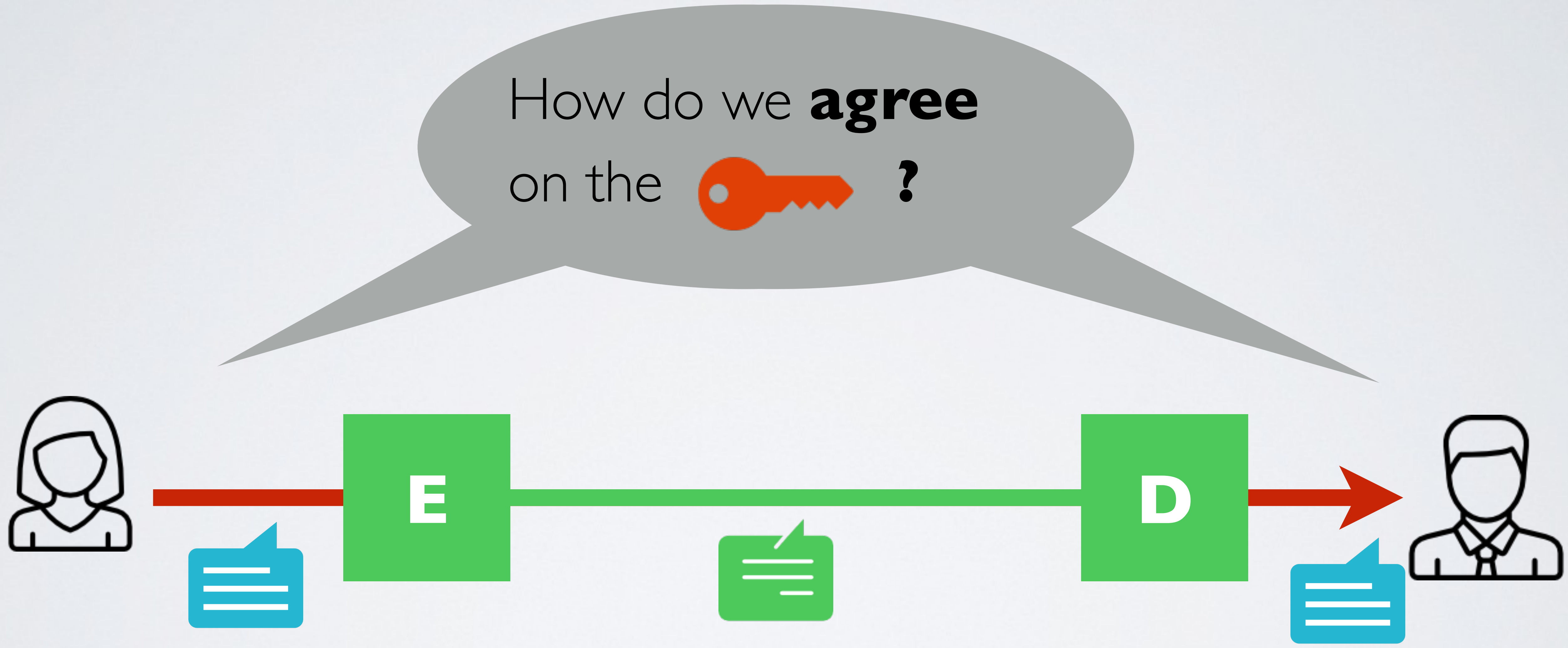
Replay attack on the response!

Defeat replay attack with a double nonce

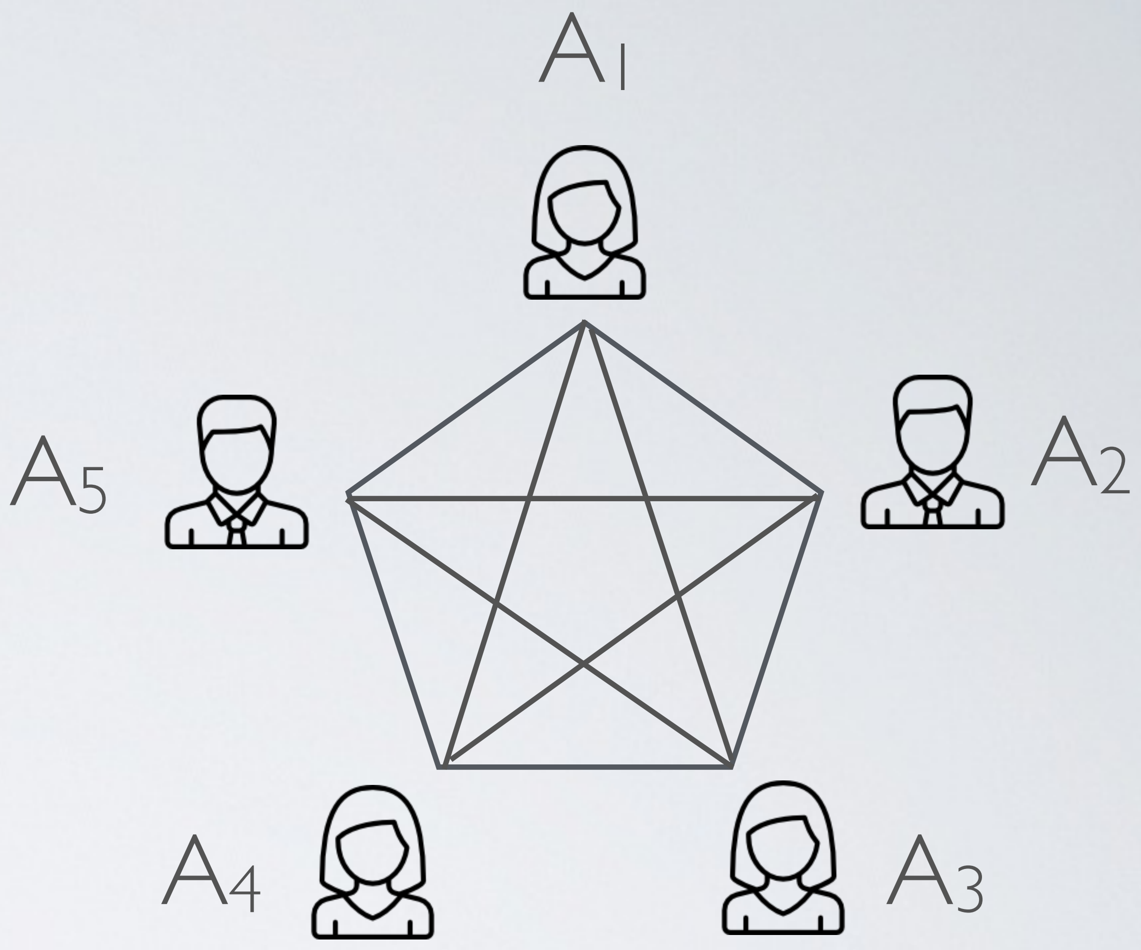


A symmetric protocol for key exchange

The big challenge with symmetric cryptosystems?



Naive Key Management

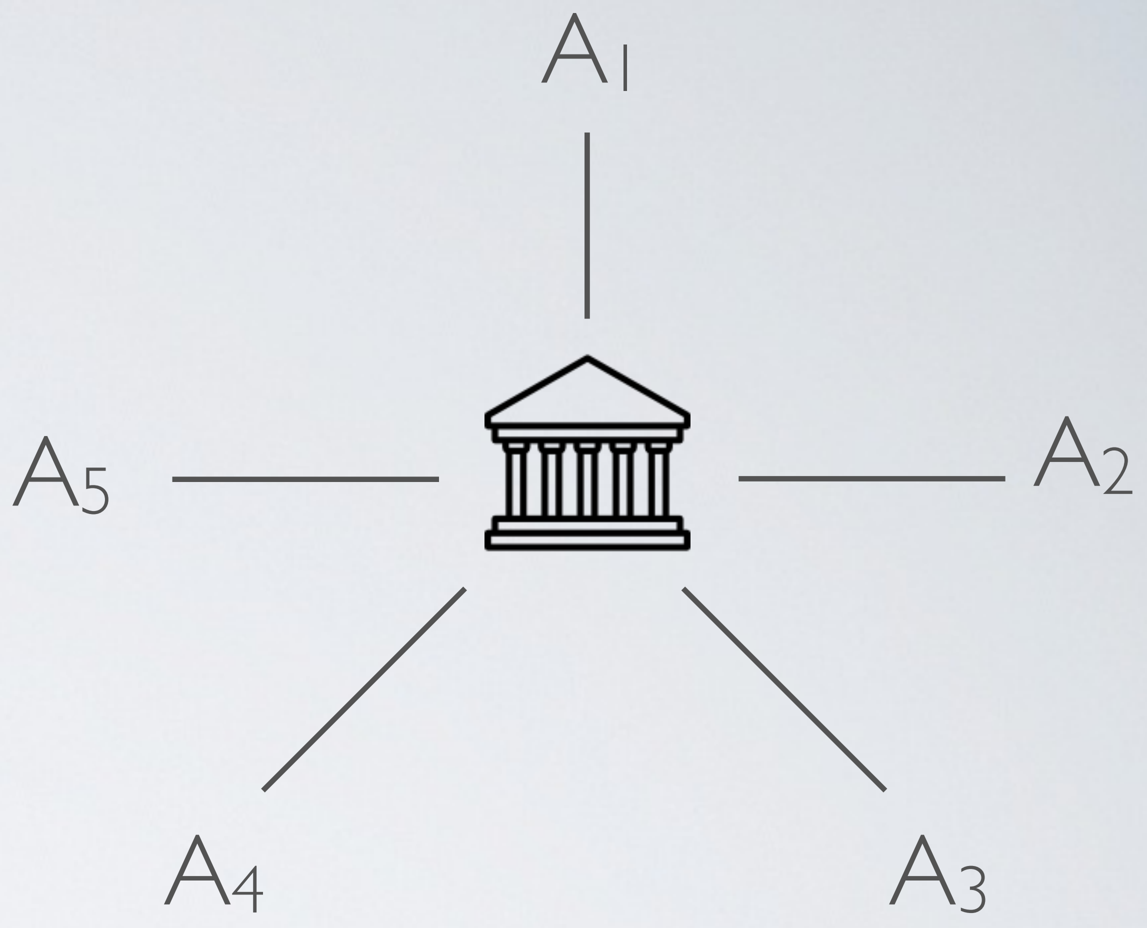


$A_1, A_2 \ldots A_5$ want to talk

➡ Each pair needs a key : **n (n-1) / 2** keys

◉ Keys must be exchanged physically using a secure channel

(Better) centralized solution



$A_1, A_2 \ldots A_5$ can talk to the KDC (Key Distribution Center)

- ➡ When A_i and A_j want to talk, the KDC can generate a new key and distribute it to them
- ◉ We still have n keys to distribute somehow using a secure channel
- ◉ The KDC must be trusted
- ◉ The KDC is a single point of failure
- ➡ The is how *Kerberos* works

The Needham-Shroeder symmetric protocol for key exchange

Assumptions

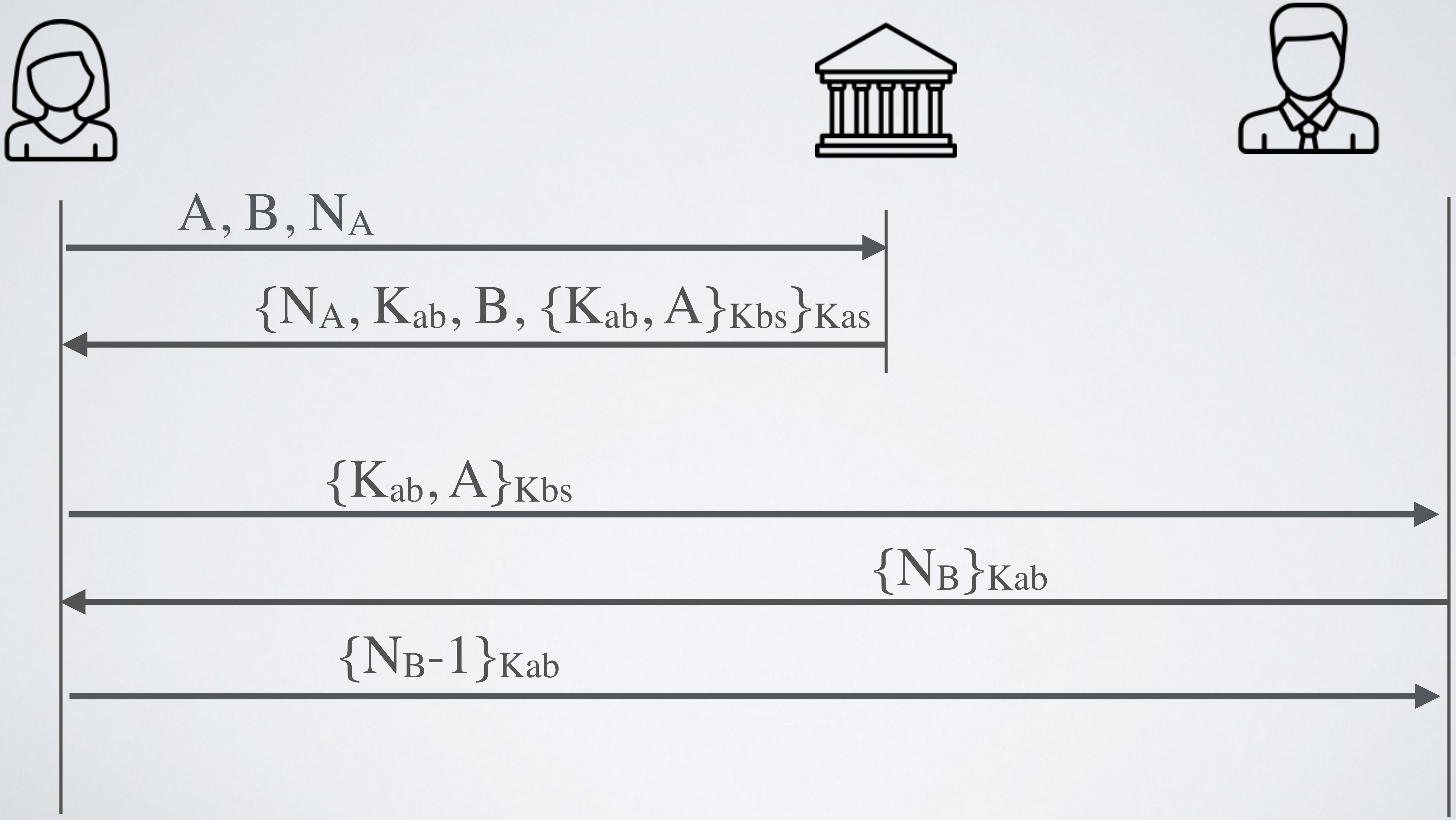
- 4 principals : **A**lice, **B**ob, **M**allory, Key Distribution **S**erver
- S shares a key with A, B and M respectively K_{as}, K_{bs}, K_{ms}
- A, B, M and S talk to each other using the same protocol

Goals

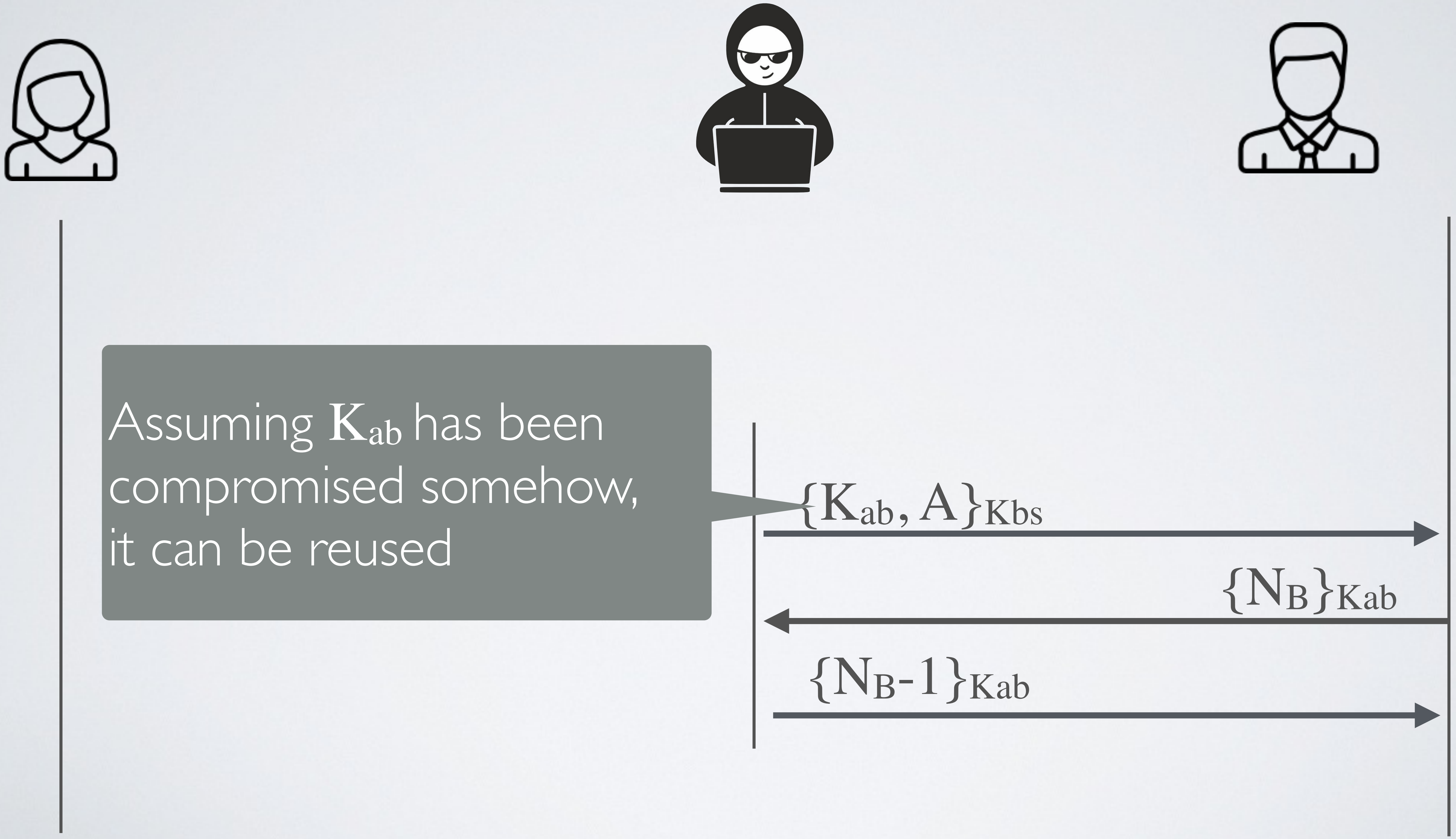
When two parties want to engage in the communication, they want to

1. make sure that they talk to the right person (authentication)
2. establish a session key

The vulnerable version of the protocol (1978)

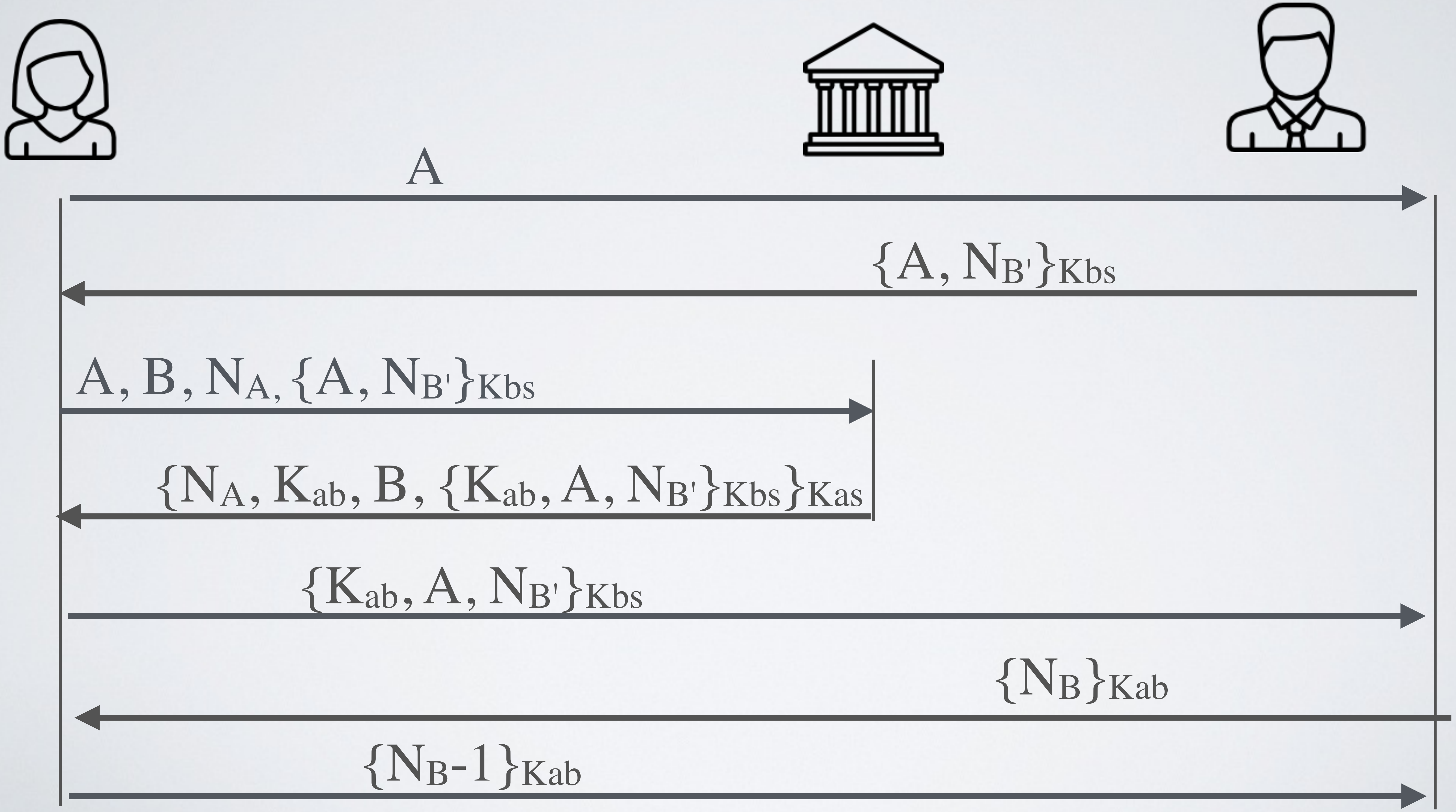


Replay attack (1981)



Assuming $\mathbf{K}_{ab}$ has been compromised somehow, it can be reused

The fix (1987)



A
$\{A, N_{B'}\}_{Kbs}$
$A, B, N_{A}, \{A, N_{B'}\}_{Kbs}$
$\{N_A, K_{ab}, B, \{K_{ab}, A, N_{B'}\}_{Kbs}\}_{Kas}$
$\{K_{ab}, A, N_{B'}\}_{Kbs}$
$\{N_B\}_{Kab}$
$\{N_B-1\}_{Kab}$

Kerberos

The Needham-Shroeder symmetric protocol is the basis for **the Kerberos Protocol**

➡ Use by Microsoft Windows for key exchange between machines on the same domain manage by the Active Directory

An asymmetric protocol for key exchange

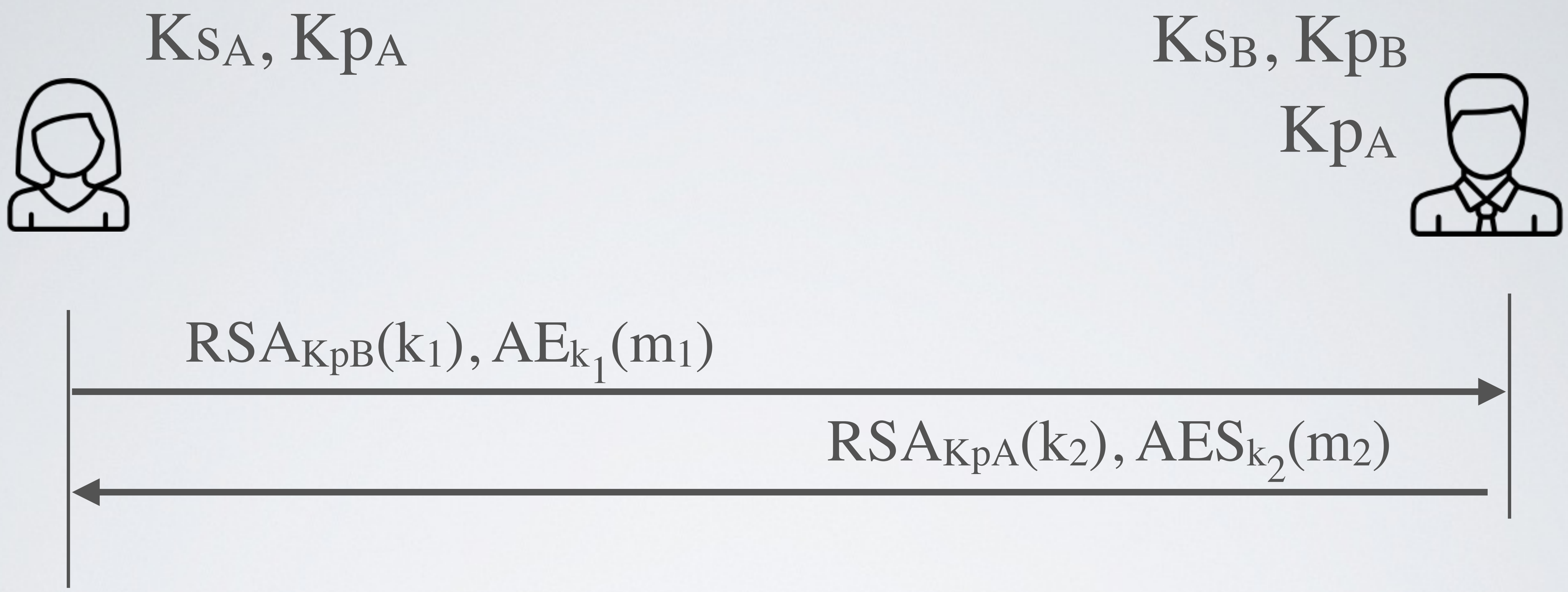
Limitations of using a key distribution centre

The key distribution server is a bootleneck and weak link

- The attacker could record the key exchange and the encrypted session, if one day either **Kas** or **Kbs** is broken, the attacker can decrypt the session

➡ Having a KDC does not ensure **Perfect Forward Secrecy**

Key exchange using asymmetric encryption



Ks_A, Kp_A
Ks_B, Kp_B
Kp_A
$RSA_{KpB}(k_1), AE_{k_1}(m_1)$
$RSA_{KpA}(k_2), AES_{k_2}(m_2)$

- The attacker could record the encrypted session, if one day either $\mathbf{Ks_A}$ or $\mathbf{Ks_B}$ is broken, the attacker can decrypt part of the session

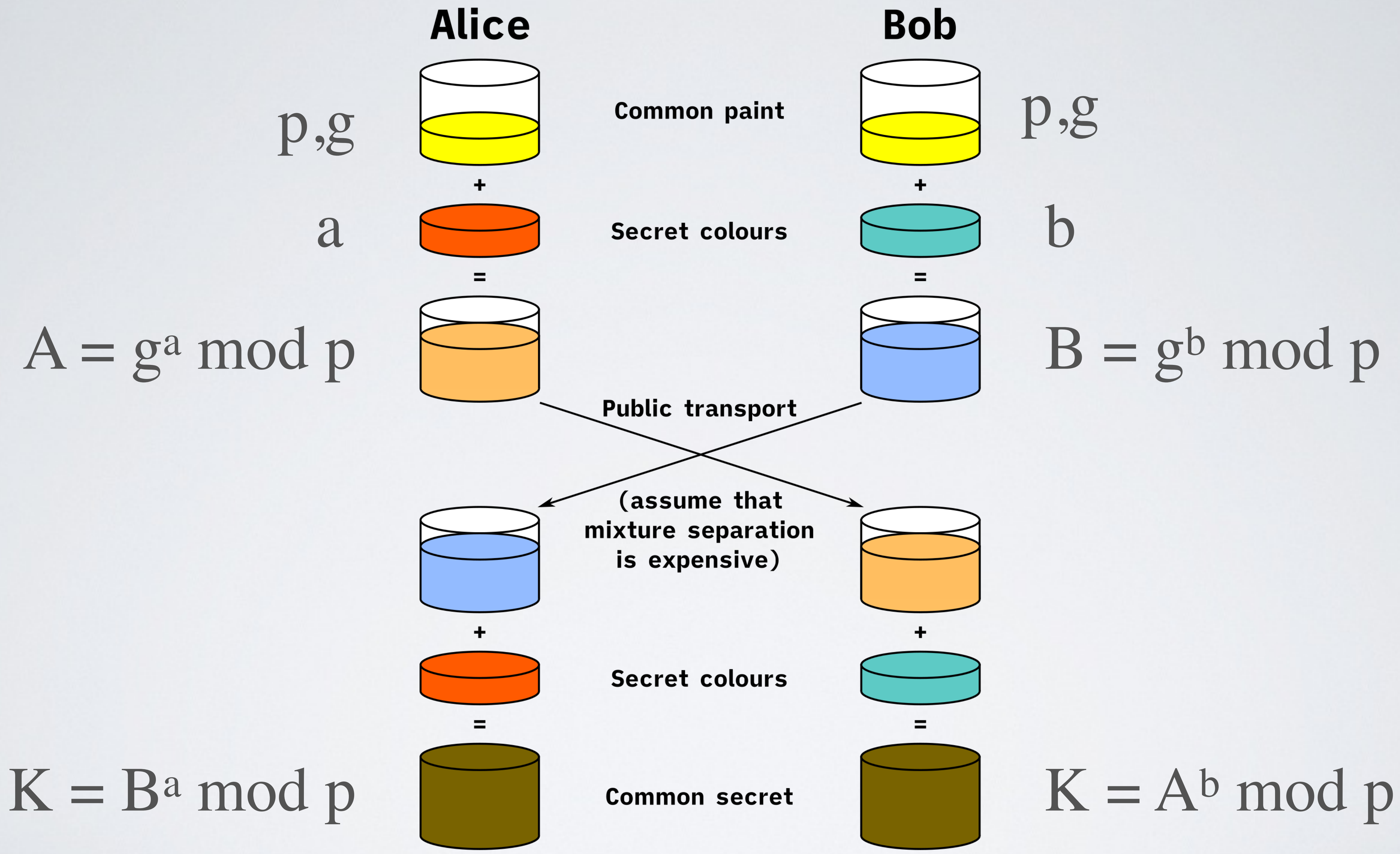
➡ Using asymmetric encryption for key exchange does not ensure **Perfect Forward Secrecy**

What is the solution?

Could Alice and Bob could magically come up with a key without exchanging it over the network?

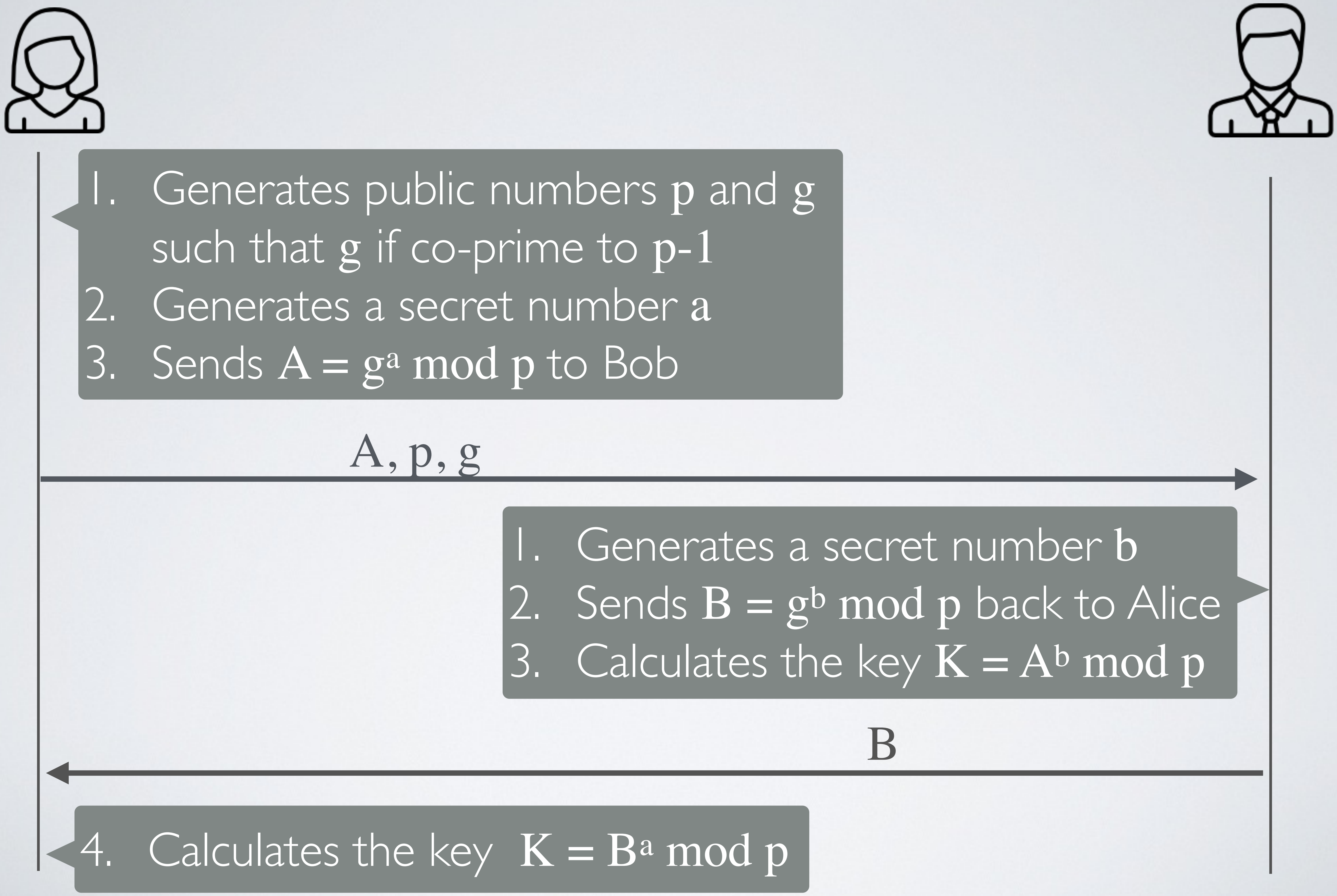
➡ The magic is called **Diffie-Hellman-Merkle Protocol**

The Diffie-Hellman-Merkel key exchange protocol



$$\boxed{K = g^{ab} \bmod p = (g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p}$$

The Diffie-Hellman-Merkel key exchange protocol



1. Generates public numbers $\mathbf{p}$ and $\mathbf{g}$ such that $\mathbf{g}$ if co-prime to $\mathbf{p-1}$
2. Generates a secret number $\mathbf{a}$
3. Sends $\mathbf{A = g^a \bmod p}$ to Bob

1. Generates a secret number $\mathbf{b}$
2. Sends $\mathbf{B = g^b \bmod p}$ back to Alice
3. Calculates the key $\mathbf{K = A^b \bmod p}$

4. Calculates the key $\mathbf{K = B^a \bmod p}$

Diffie-Hellman-Merkle in practice

- $\mathbf{g}$ is small (either 3, 5 or 7 and fixed in practice)
- $\mathbf{p}$ is at least 2048 bits (and fixed in practice)
- private keys $\mathbf{a}$ and $\mathbf{b}$ are 2048 bits as well

➡ So the public values $\mathbf{A}$ and $\mathbf{B}$
and the master key $\mathbf{k}$ are 2048 bits

➡ Use $\mathbf{k}$ to derive an AES key using a Key Derivation Function
(usually HKDF - the HMAC-based Extract-and-Expand key derivation function)

A widely used key exchange protocol

Diffie-Hellman-Merkle is in many protocols

- SSH
- TLS (used by HTTPS)
- Signal (used by most messaging apps like Whatsapp)
- and so on . . .

✓ It is fast and requires two exchanges only

✓ Solves the problem of having a key distribution server

✓ Ensures Perfect Forward Secrecy

◉ But how to make sure Alice is talking to Bob and vice-versa?
Diffie-Hellman-Merkle alone **does not ensure authentication**

Another Challenge

1. Establish a session key to exchange data while ensuring Perfect Forward Secrecy

   ✓ Use the Diffie-Hellman key exchange protocol

2. Ensure one-way or mutual authentication

   ✓ Use asymmetric encryption

The Needham-Schroeder public-key protocol for mutual authentication

Assumptions and Goals

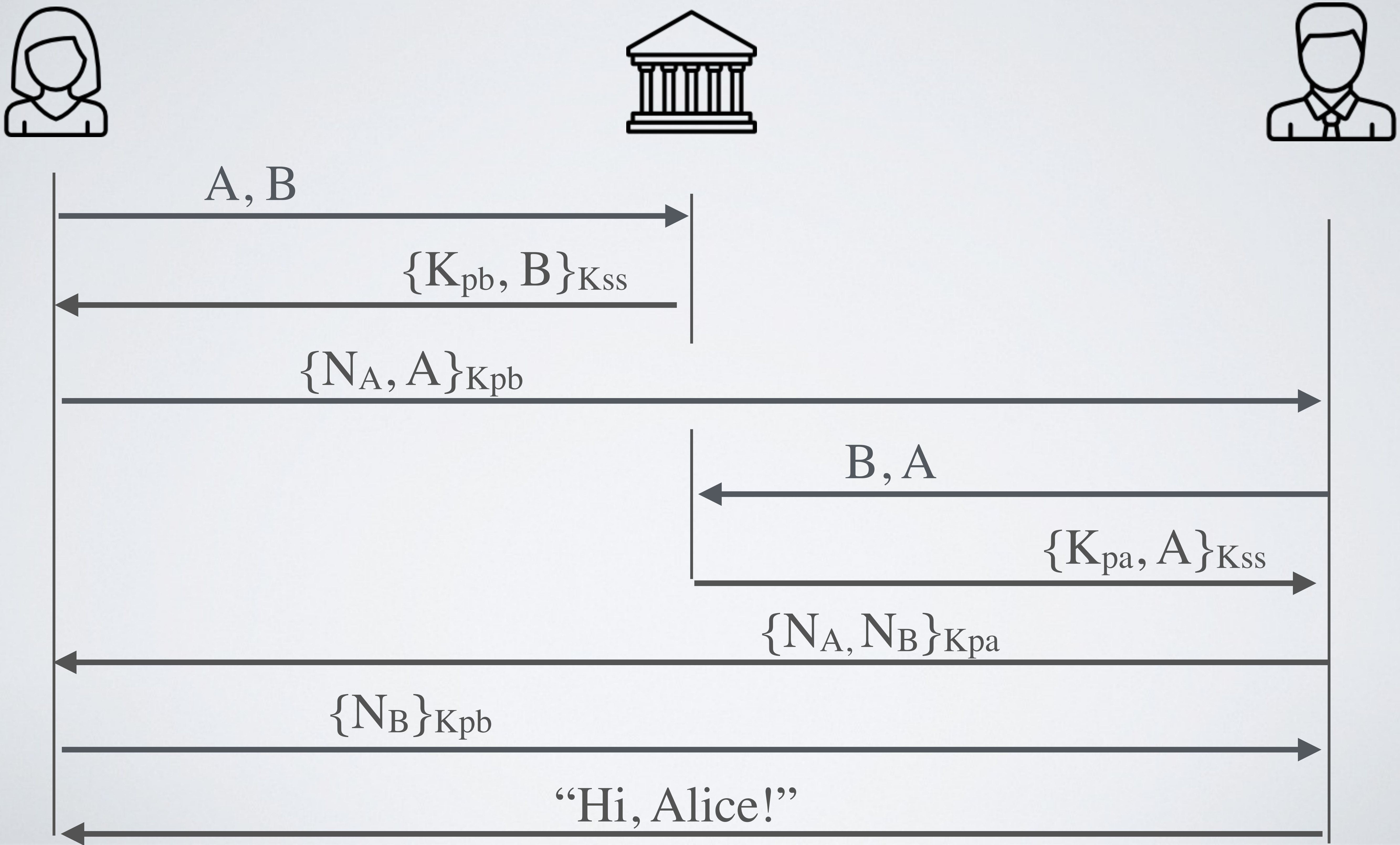
Assumptions

- 4 principals : Alice, Bob, Mallory and a Public-Key Server
- Alice, Bob, Mallory and the Server have generated their own public/private key pair
- Alice, Bob and Mallory know the Server's public key **Kps**
- **A**, **B** , **M** and **S** talk to each other using the same protocol

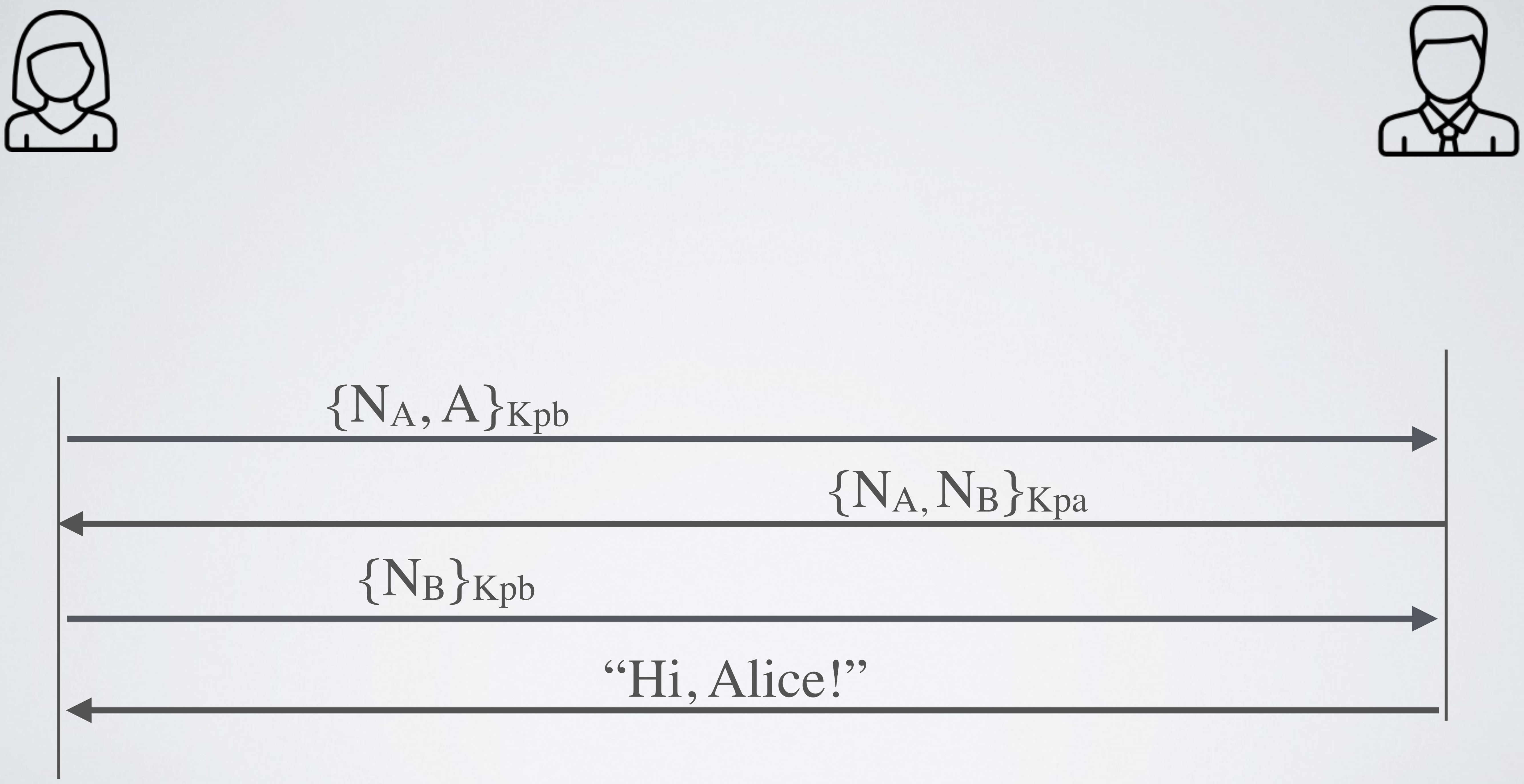
Goals

When two parties want to engage in the communication, they want to make sure that they talk to the right person (authentication)

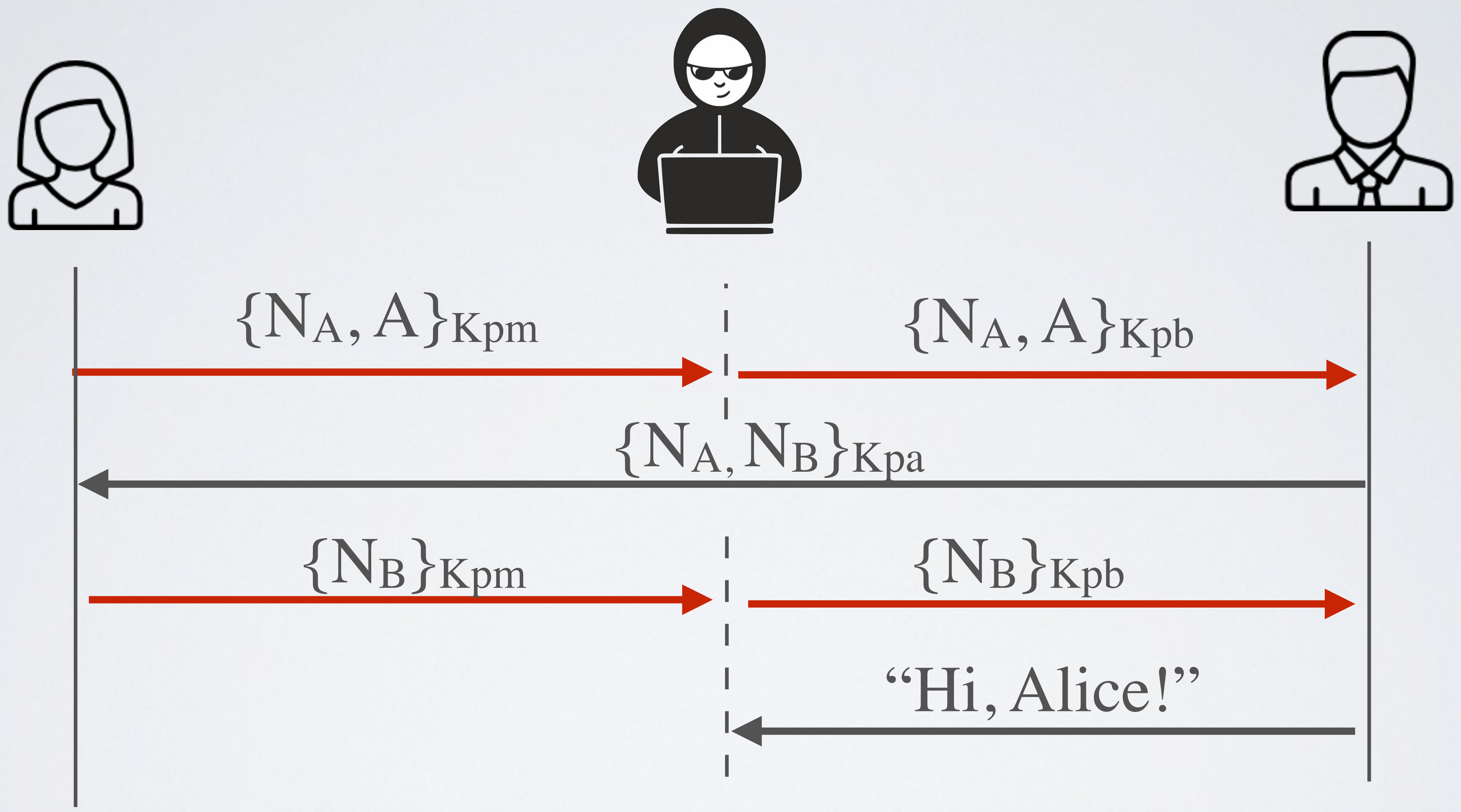
The vulnerable version (1978)



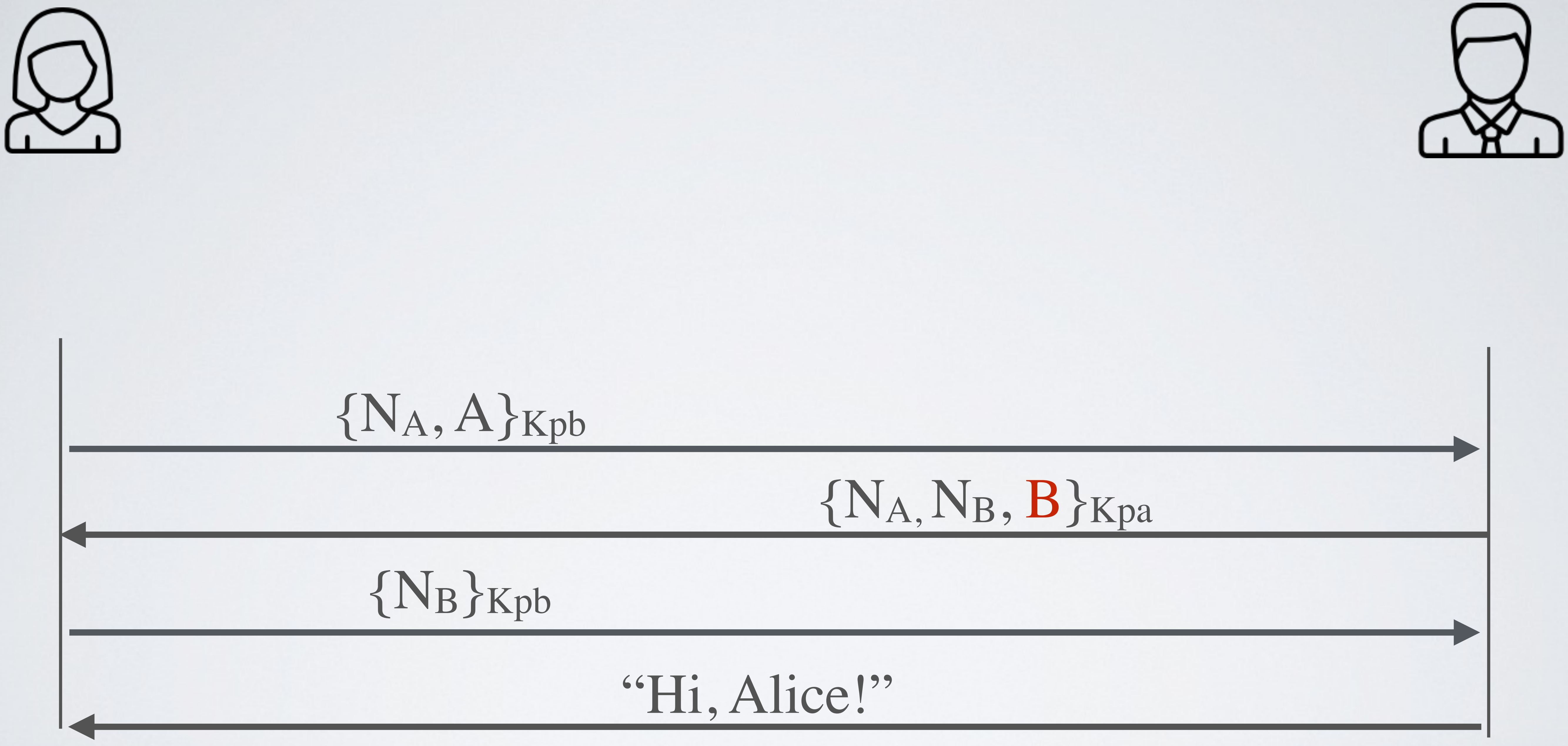
Simplified (but still vulnerable) version (1978)



Man-in-the-middle attack (Lowe's 1995)



Lowe's fix (1995)

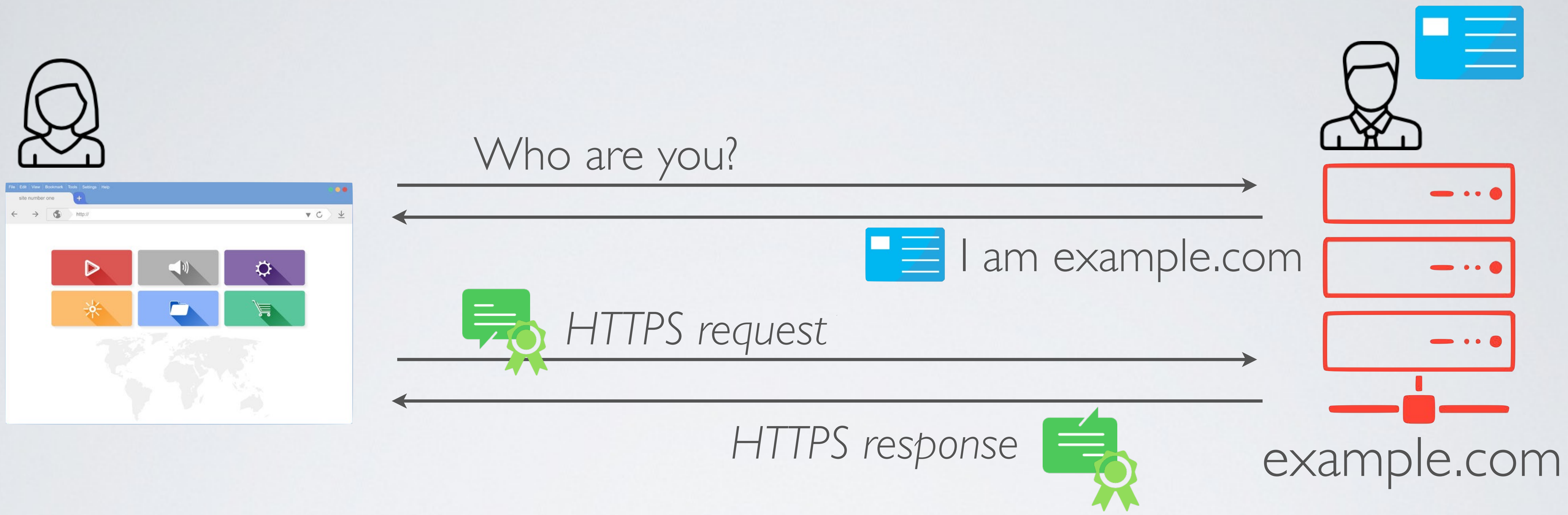


Not a perfect protocol yet

- ✓ Does authenticate Alice and bob
- ✓ Does prevent replay attacks
- ✓ Does ensure the authenticity of the public keys
- ◉ But the Public Key Server is a <u>single point of failure</u>

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

This how HTTPS works

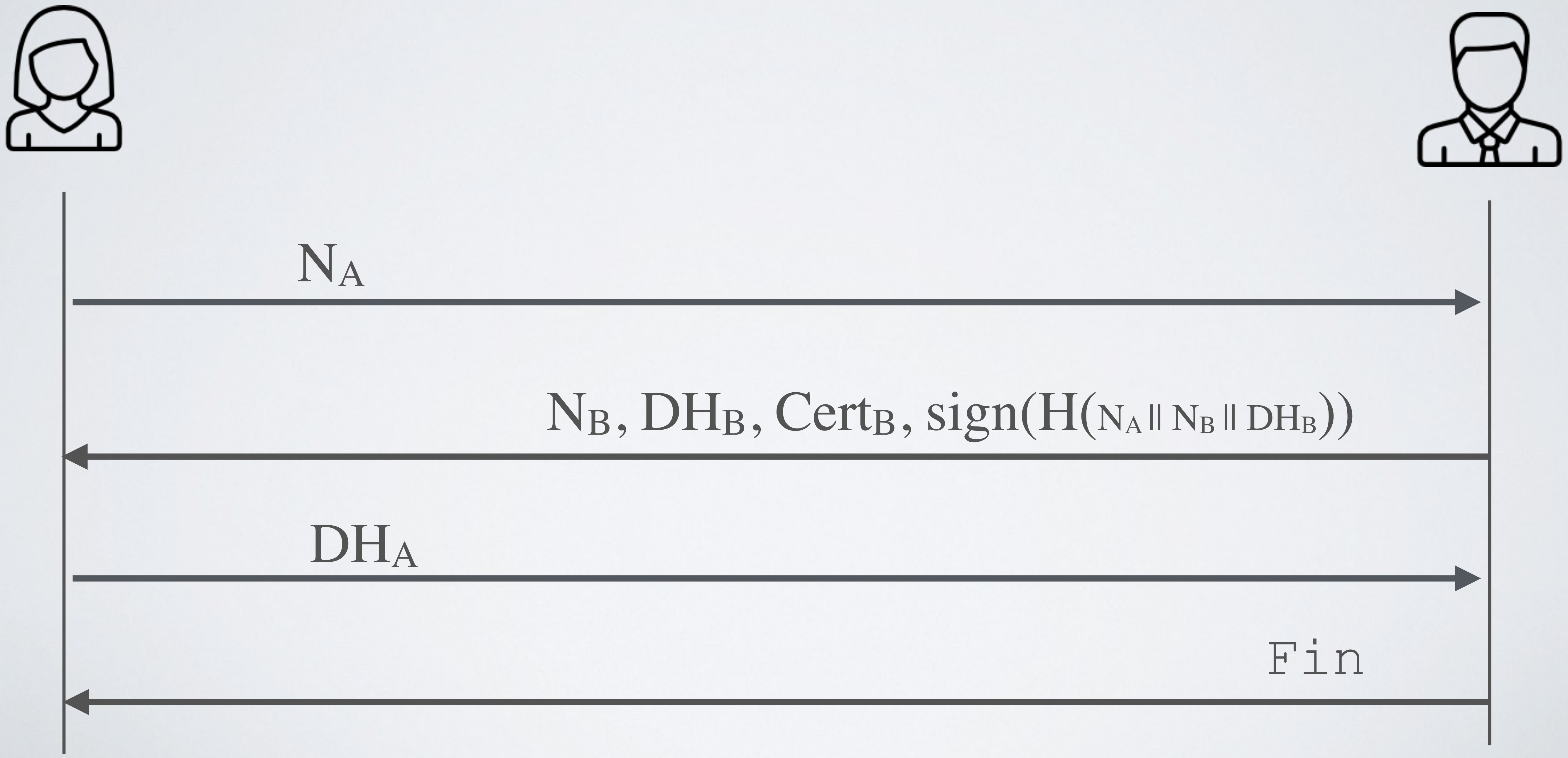


✓ **HTTPS = HTTP + TLS**

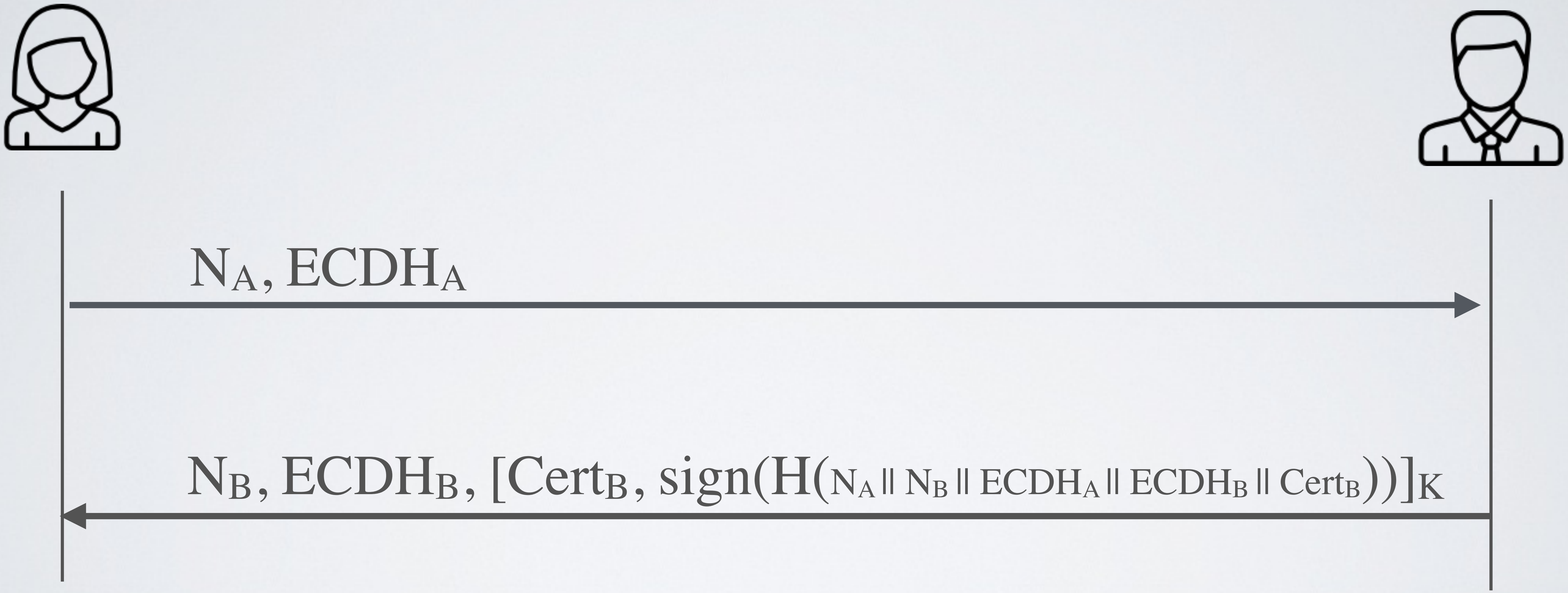
➡ TLS - Transport Layer Security (a.k.a SSL) provides

- **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)


N_A, $ECDH_A$
N_B, $ECDH_B$, $[Cert_B$, $sign(H(N_A \| N_B \| ECDH_A \| ECDH_B \| Cert_B))]_K$

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)

Almost there . . .

- ✓ Does ensure the confidentiality of the communication
- ✓ Does authenticate Alice and bob
- ✓ Does prevent replay attacks
- ➡ But how to ensure the authenticity of the public keys without using a Public Key Server ?

Trust Models

Two trust models

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

<u>Decentralized trust model</u>

➡ **Web of Trust**

GnuPG

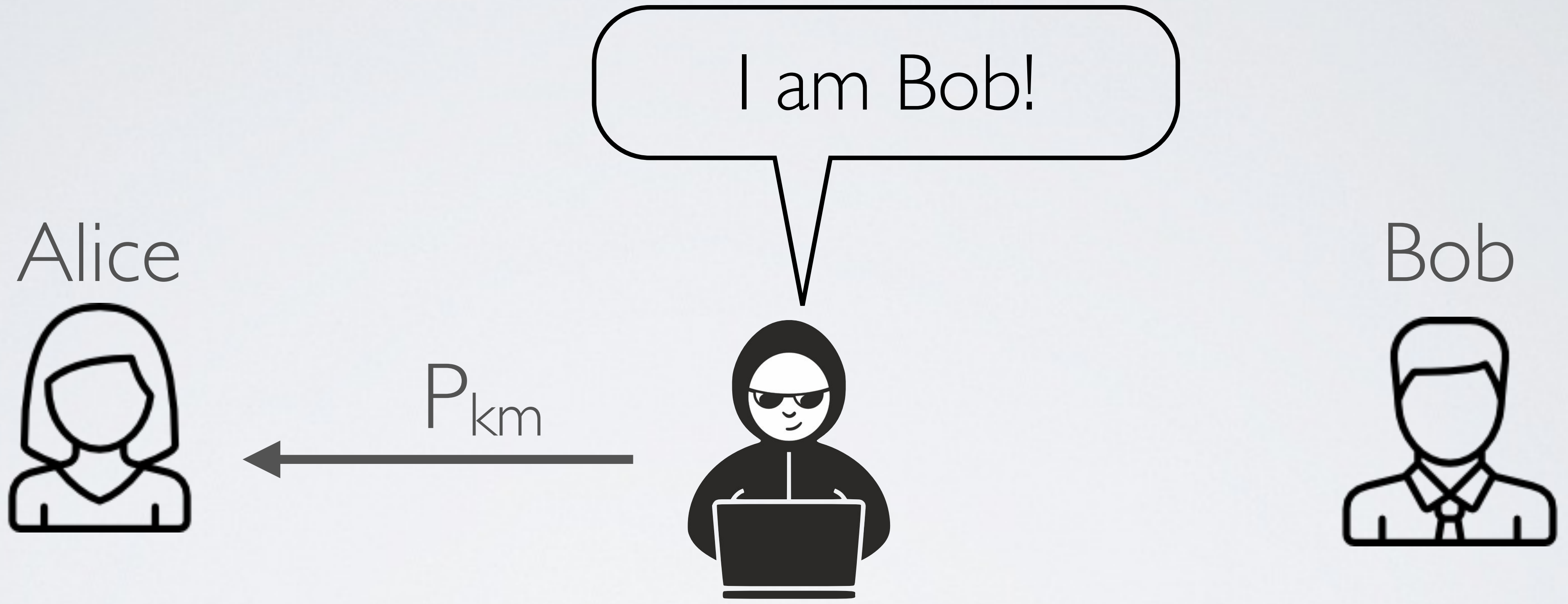
<u>Centralized trust model</u>

➡ **PKI - Public Key Infrastructure**

https://

TLS (SSL) Encryption

Do you trust the GPG key ?

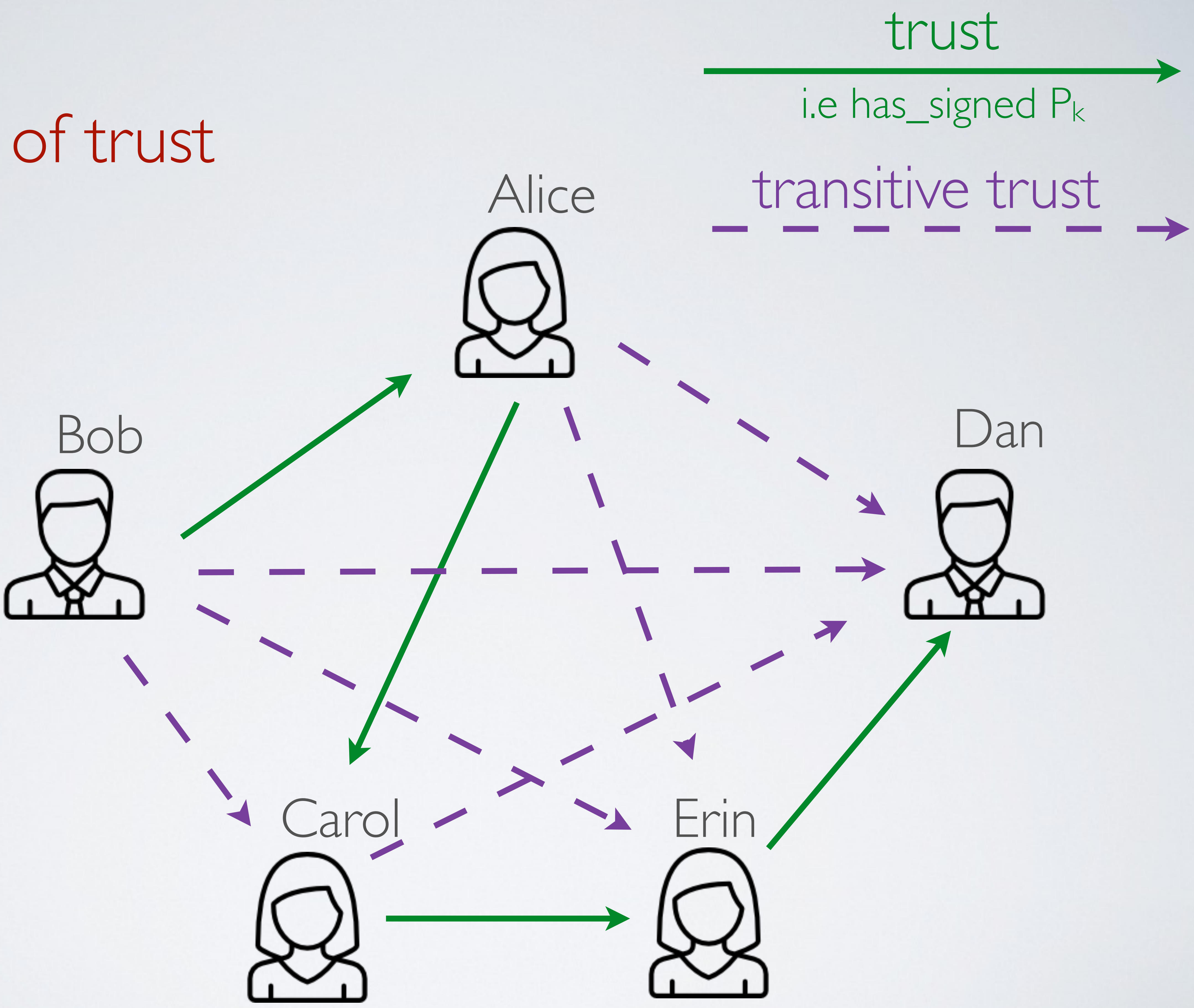


Alice should verify Bob’s public key fingerprint

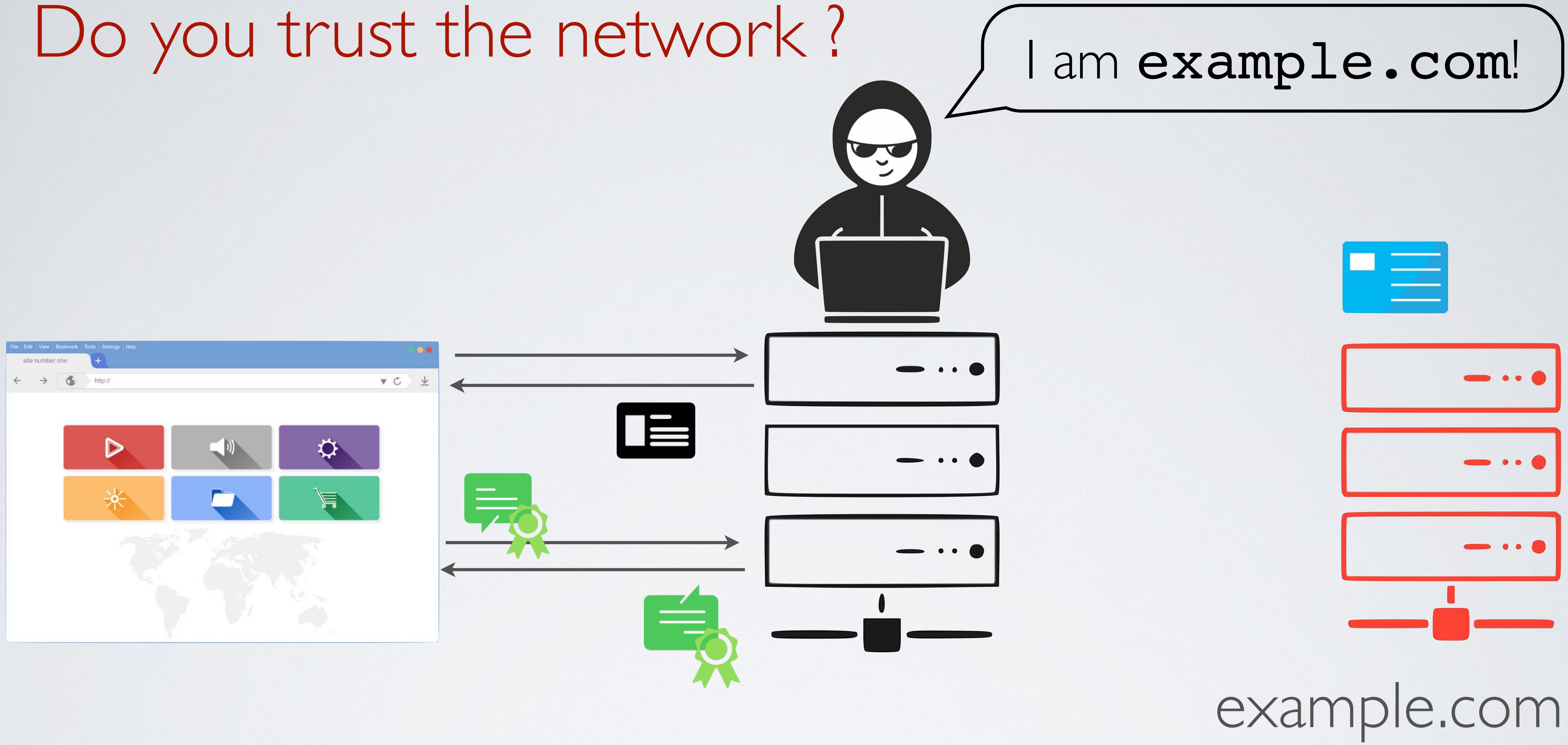
- either by communicating with Bob over another channel
- or by trusting someone that already trusts Bob

➡ **the web of trust**

The web of trust



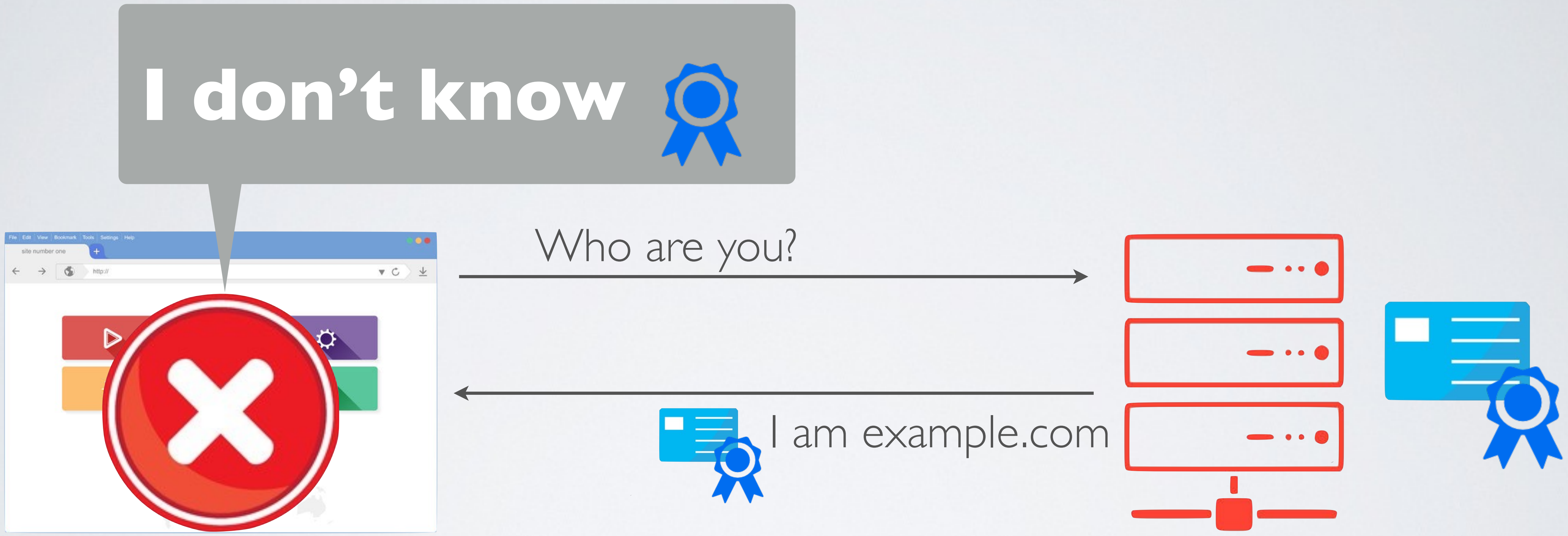
Do you trust the network ?



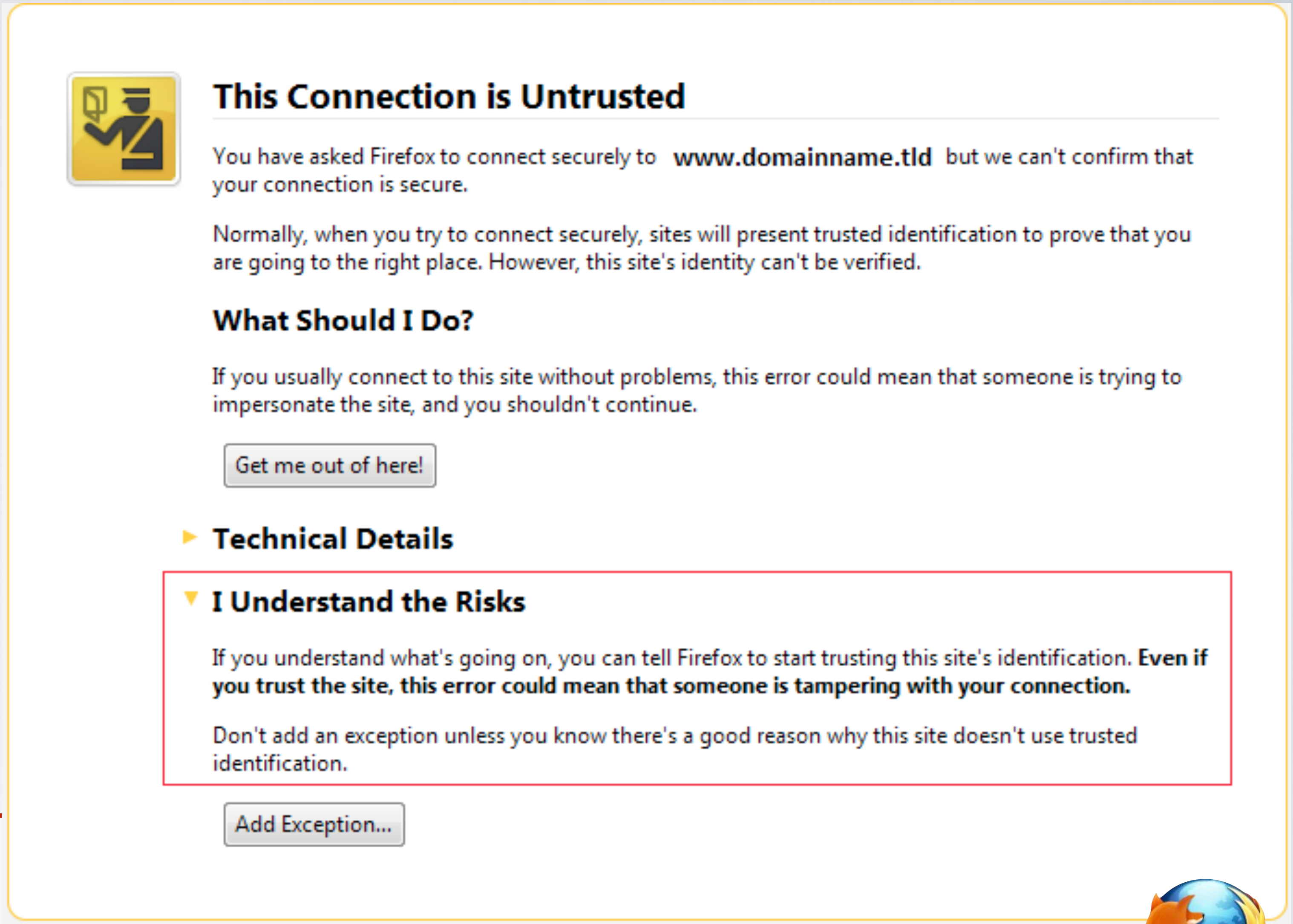
The browser should verify the certificate

➡ **PKI - Public Key Infrastructure**

Generating and using (self-signed) certificates



Self-signed certificates are not trusted by your browser


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?

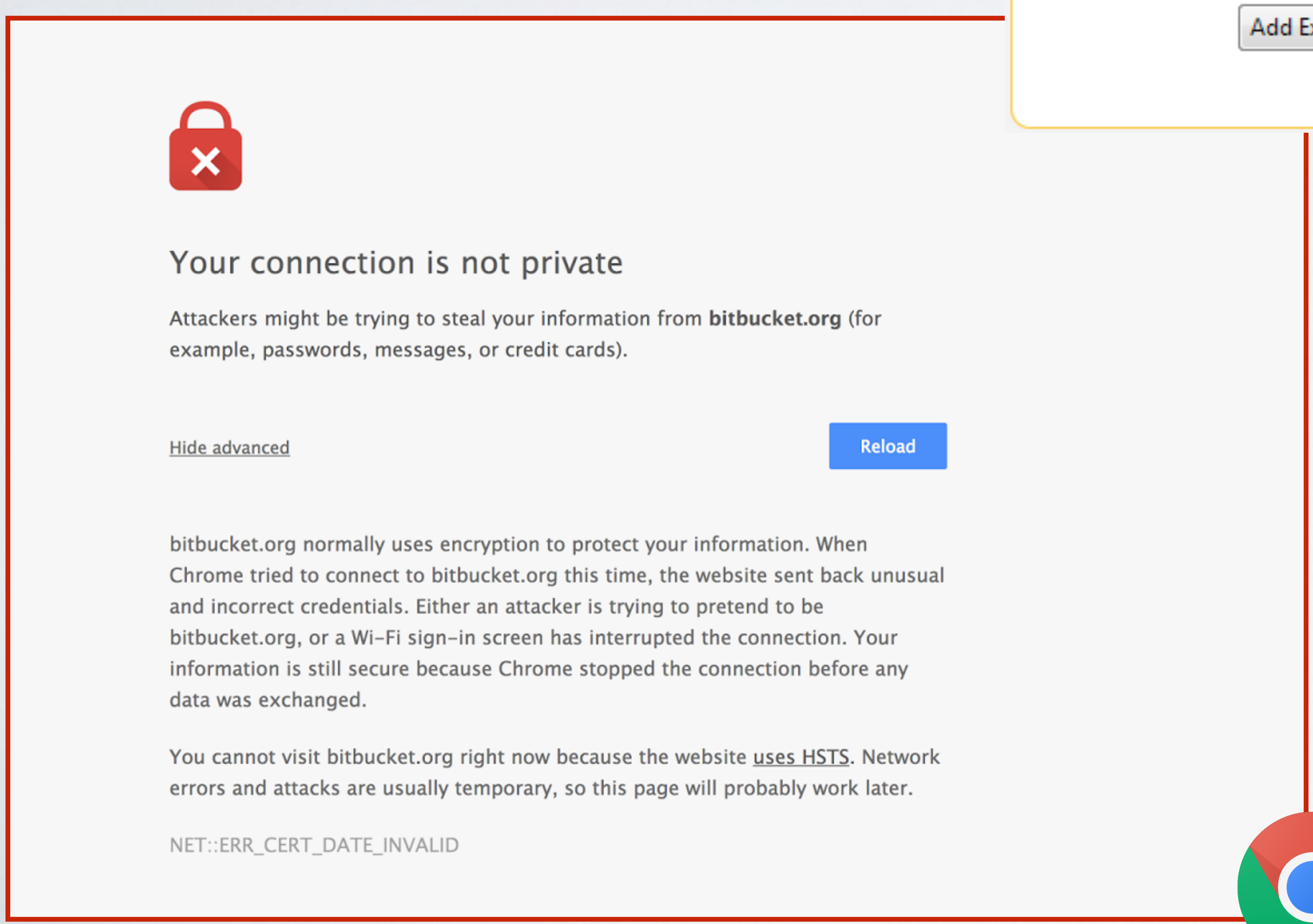
If you usually connect to this site without problems, this error could mean that someone is trying to impersonate the site, and you shouldn't continue.

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.


Your connection is not private

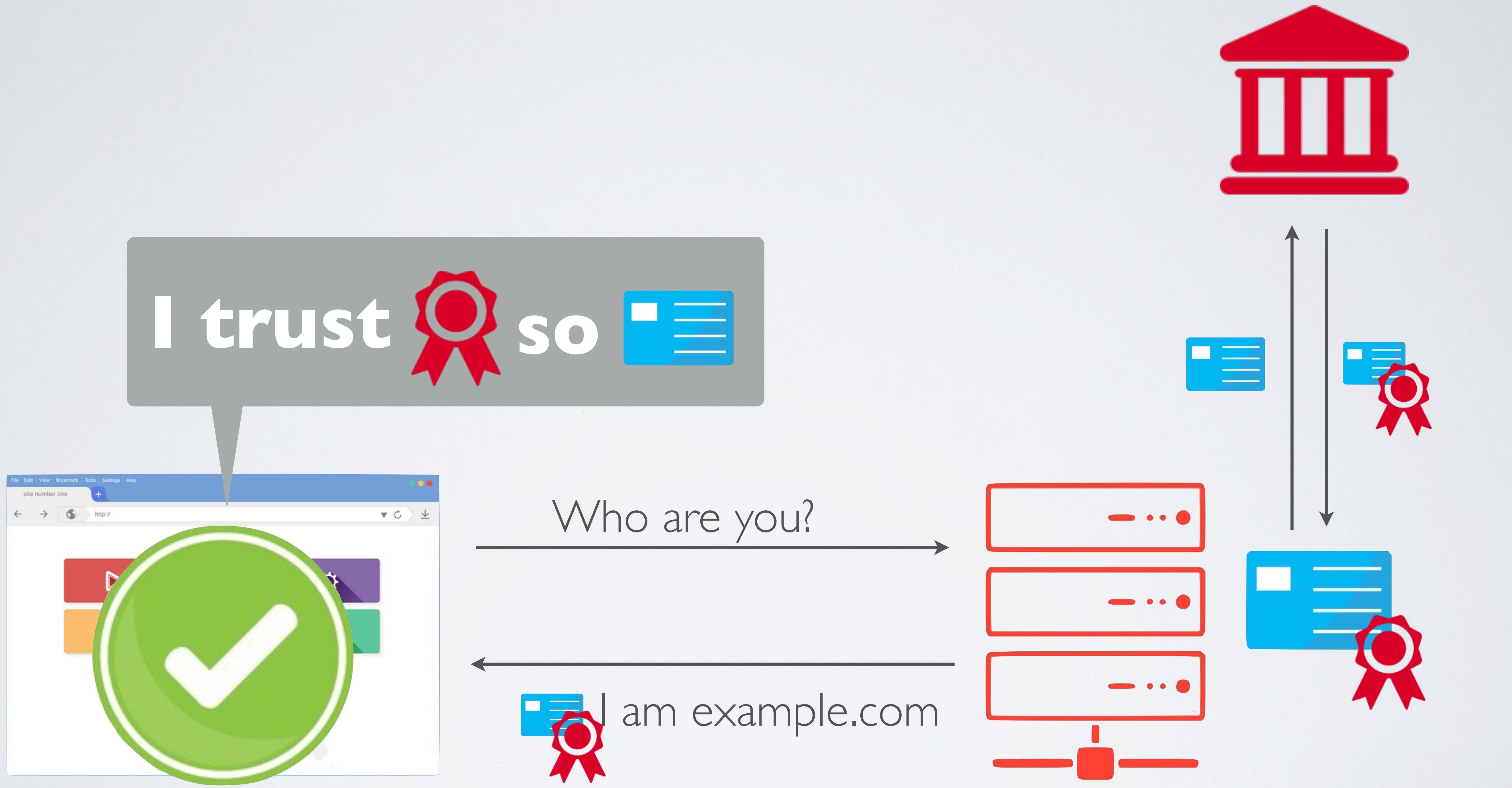
Attackers might be trying to steal your information from **bitbucket.org** (for example, passwords, messages, or credit cards).

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.

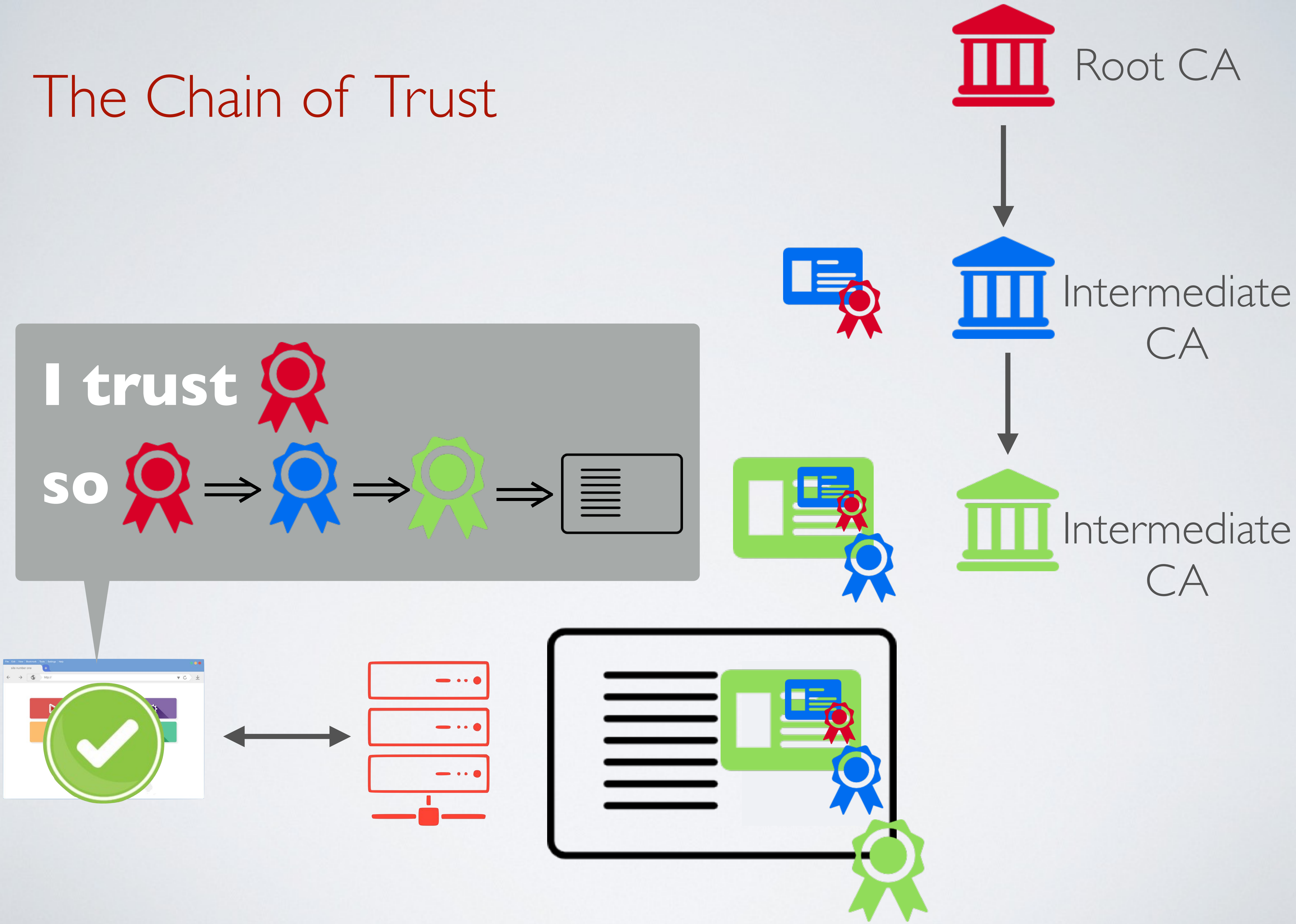
You cannot visit bitbucket.org right now because the website uses HSTS. Network errors and attacks are usually temporary, so this page will probably work later.

NET::ERR_CERT_DATE_INVALID

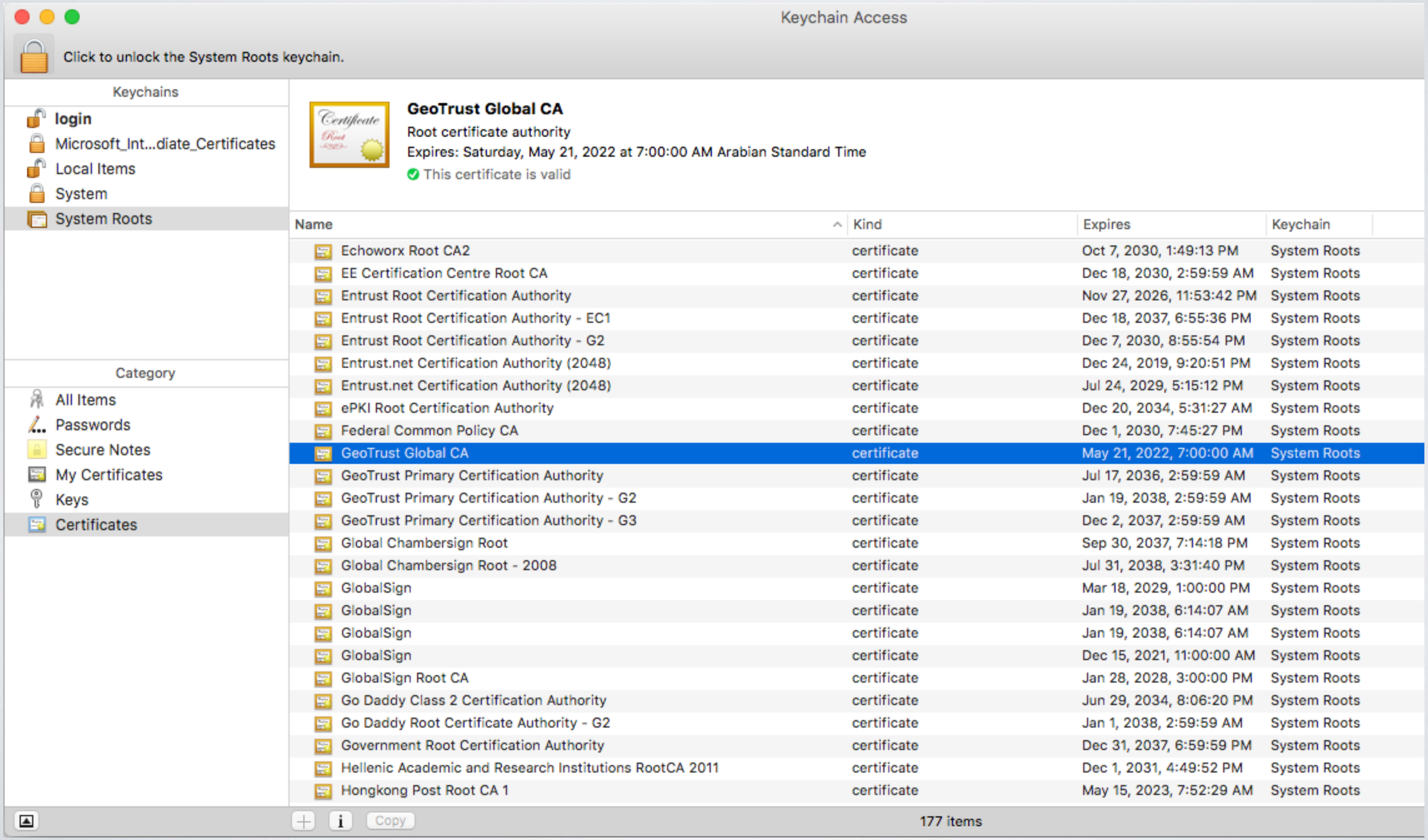
Signed Certificate



The Chain of Trust



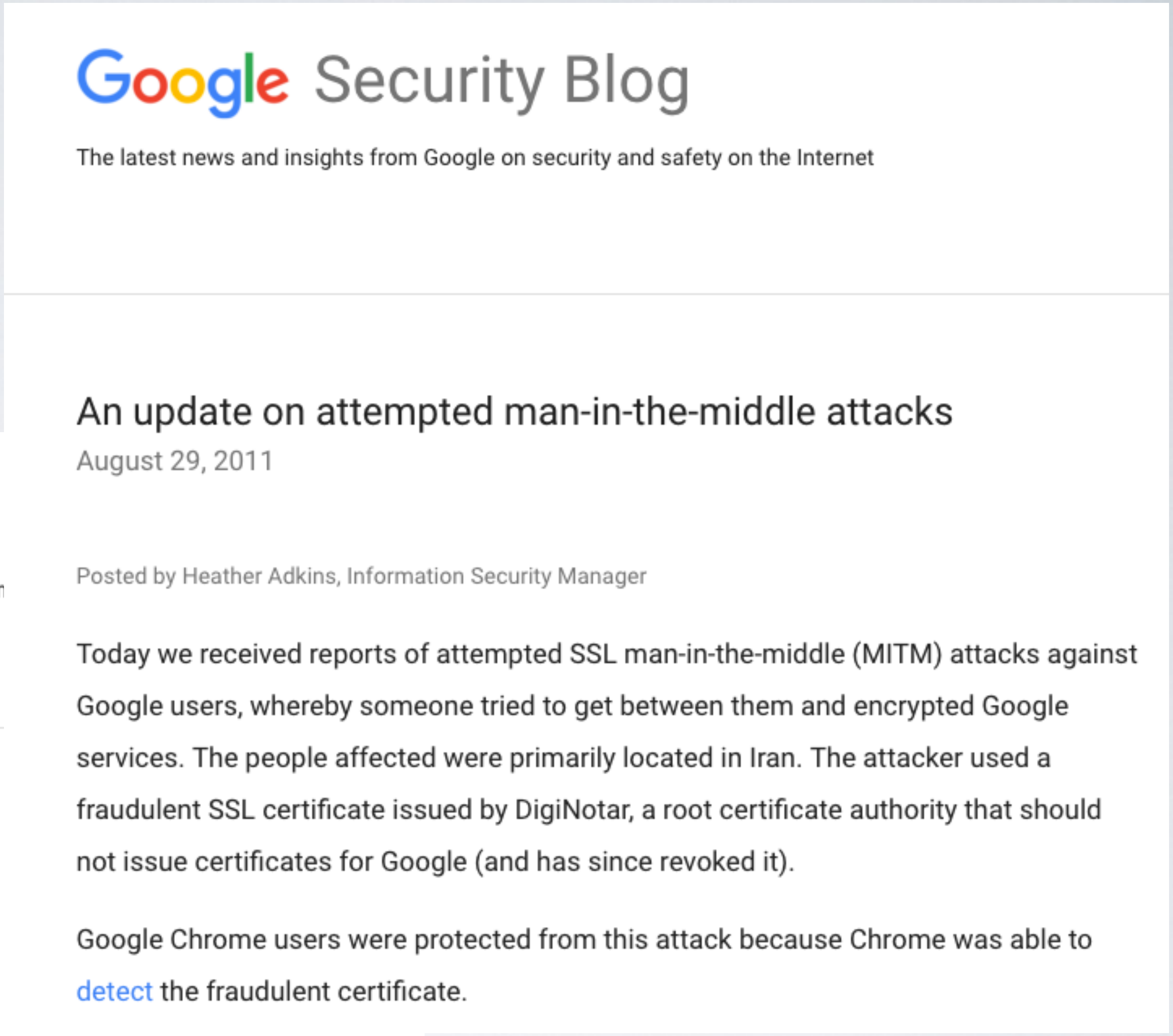
Your browser trusts many root CAs **by default**



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

Real attacks


Google Security Blog

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

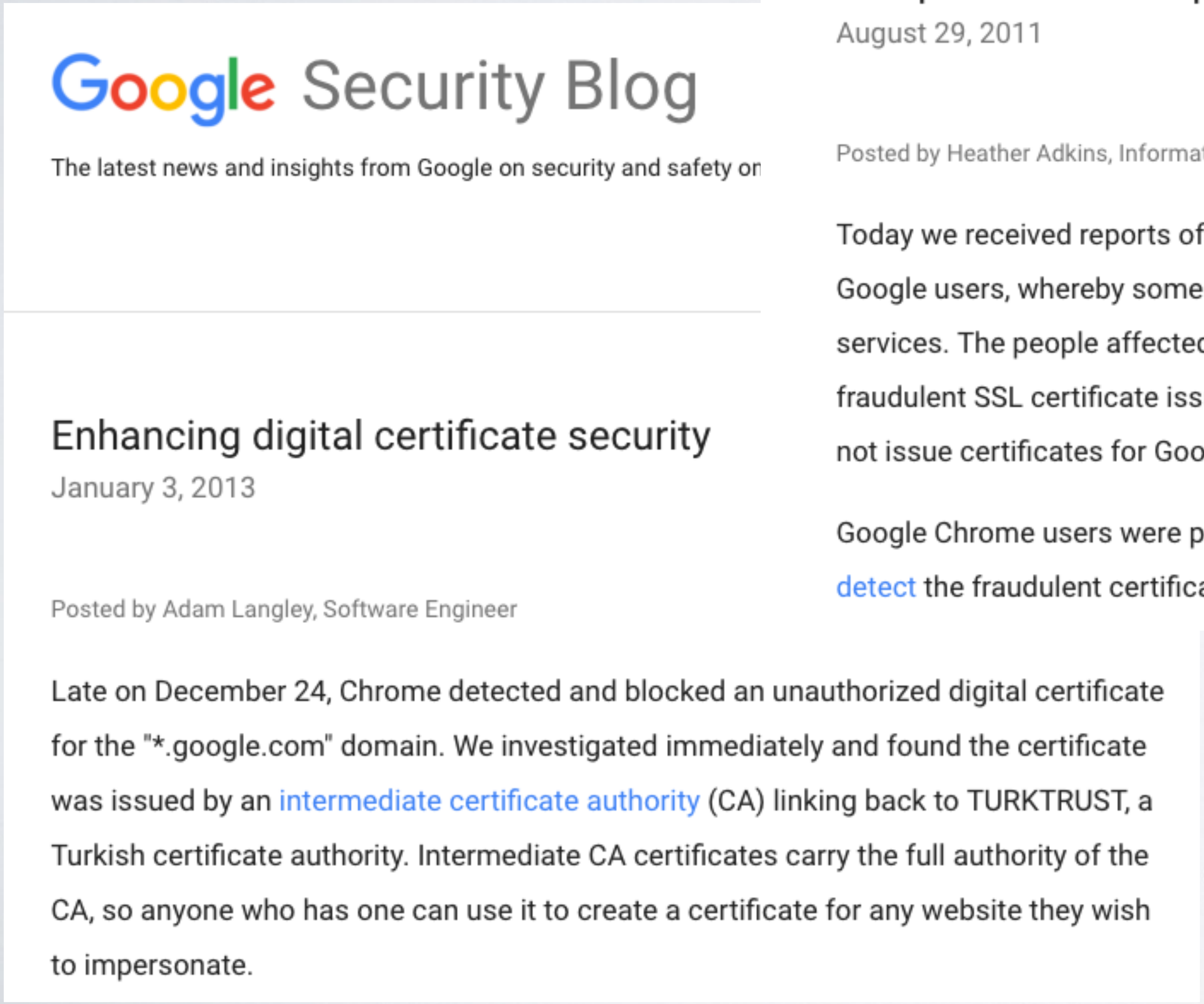
An update on attempted man-in-the-middle attacks

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

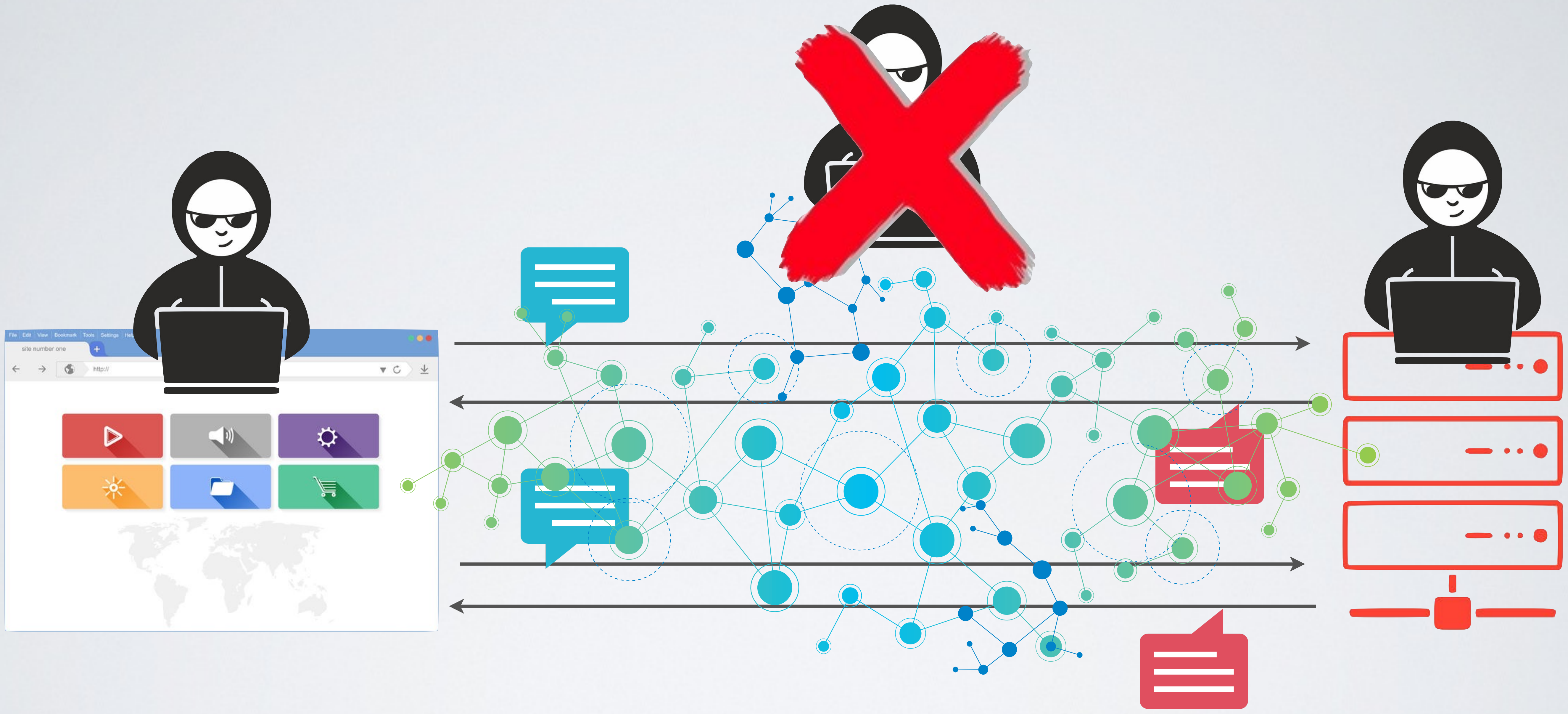
Enhancing digital certificate security

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.

Limitation of secure channels


File Edit View Bookmark Tools Settings
site number one
http://