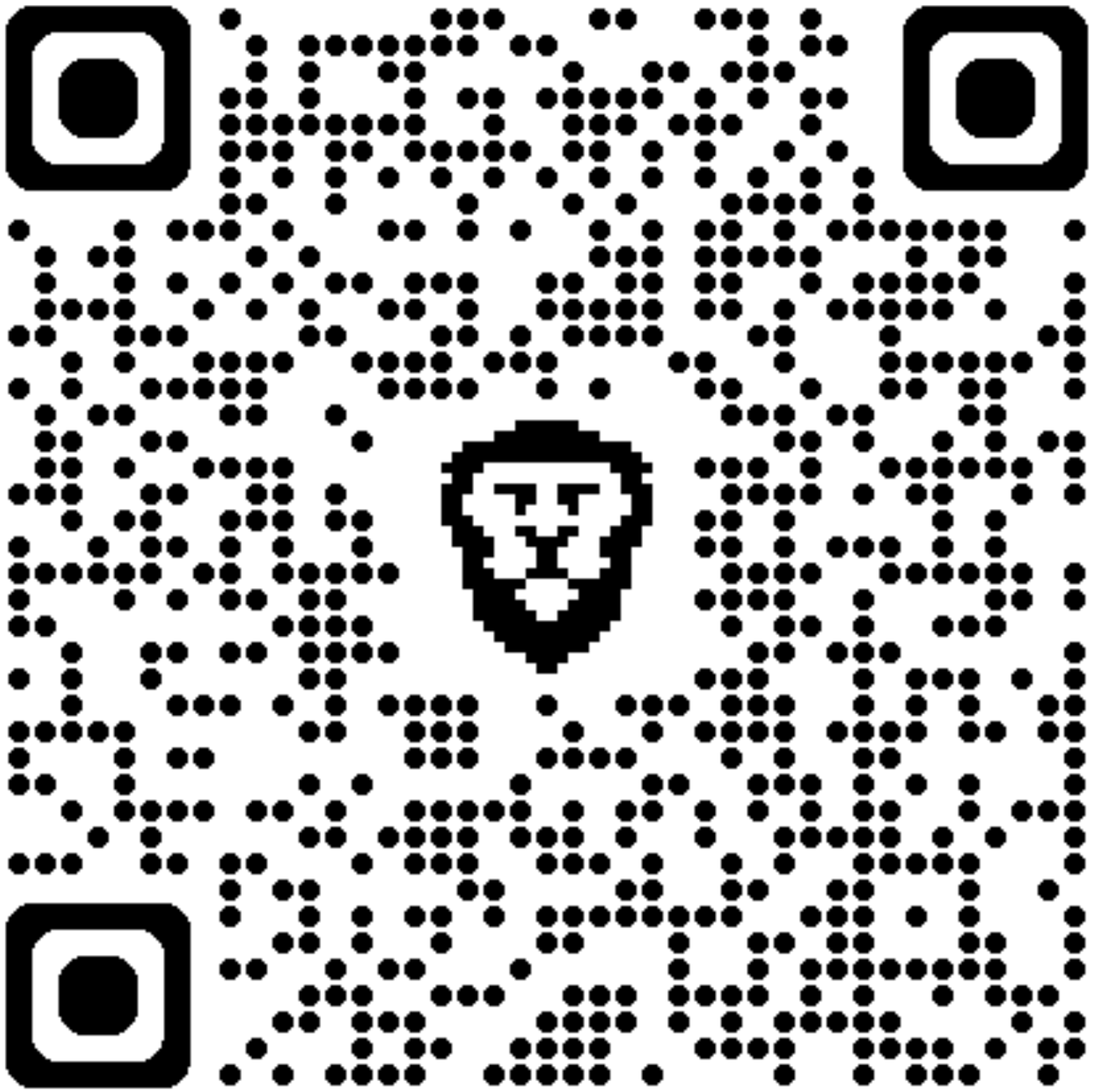


Get the slides!

Thierry Sans
Associate Professor - Teaching Stream

Teaching

- Computer and Network Security
- Blockchain and Decentralized Applications
- Web Development
- Operating Systems

Research

- Cryptography Protocols
- Zero-Knowledge Proofs
- Blockchain Protocols and Applications

Understanding the "s" in "https://"

An introduction to modern cryptography

Thierry Sans

You are using cryptography everyday

- **Browsing the web** with `HTTPS://`
- **Text messaging** apps
- **Authentication** (passwords, biometrics, one-time code and so on)
- Connecting to a remote server **using SSH**
- **Bitcoin** and other cryptocurrencies

The Internet

1980's - few hosts connected : government institutions and universities

➡ Trustworthy environment

2025 - ~ 5.6 billion internet users : network of networks

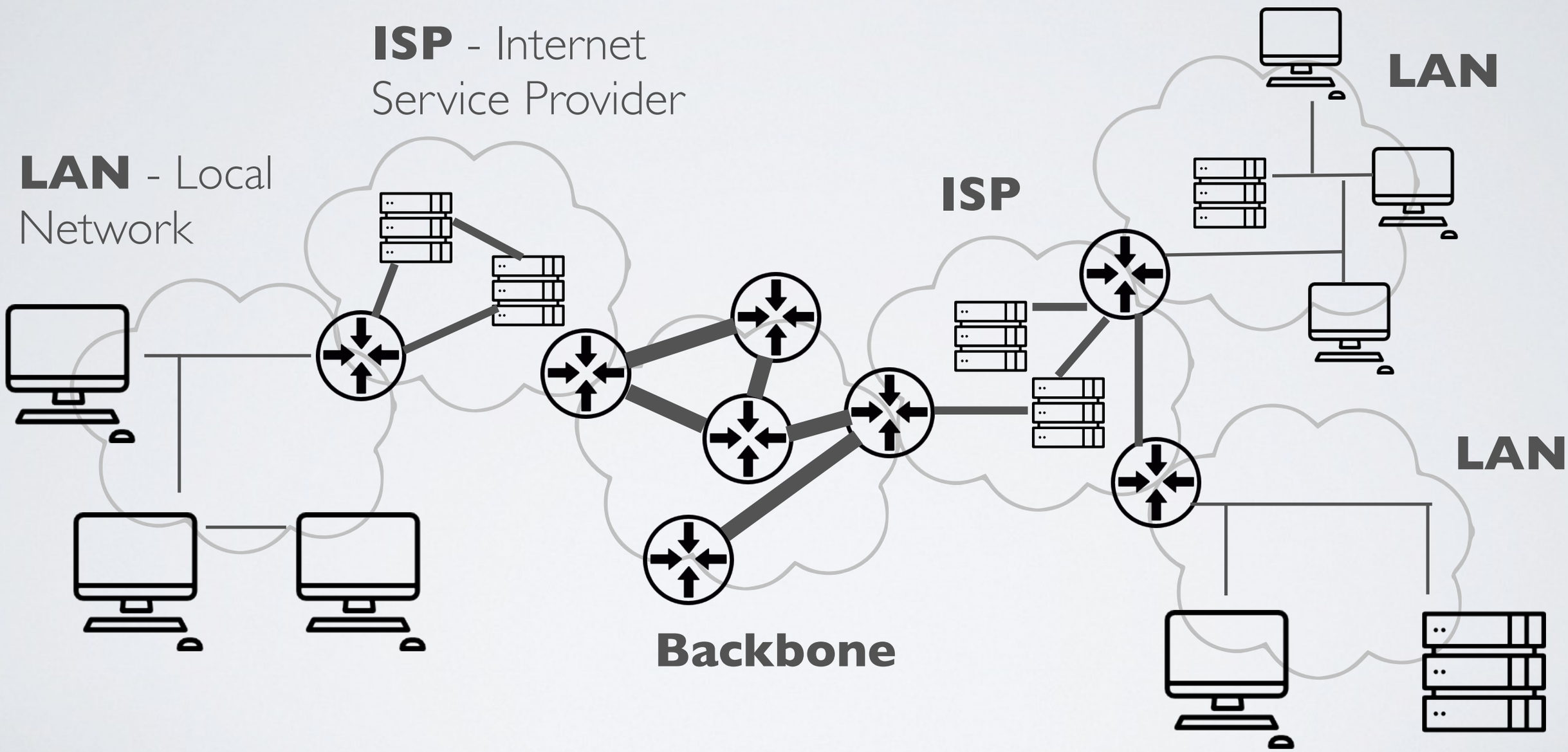
➡ Untrustworthy environment

➡ Internet (and its protocols) was
not designed for untrustworthy environment

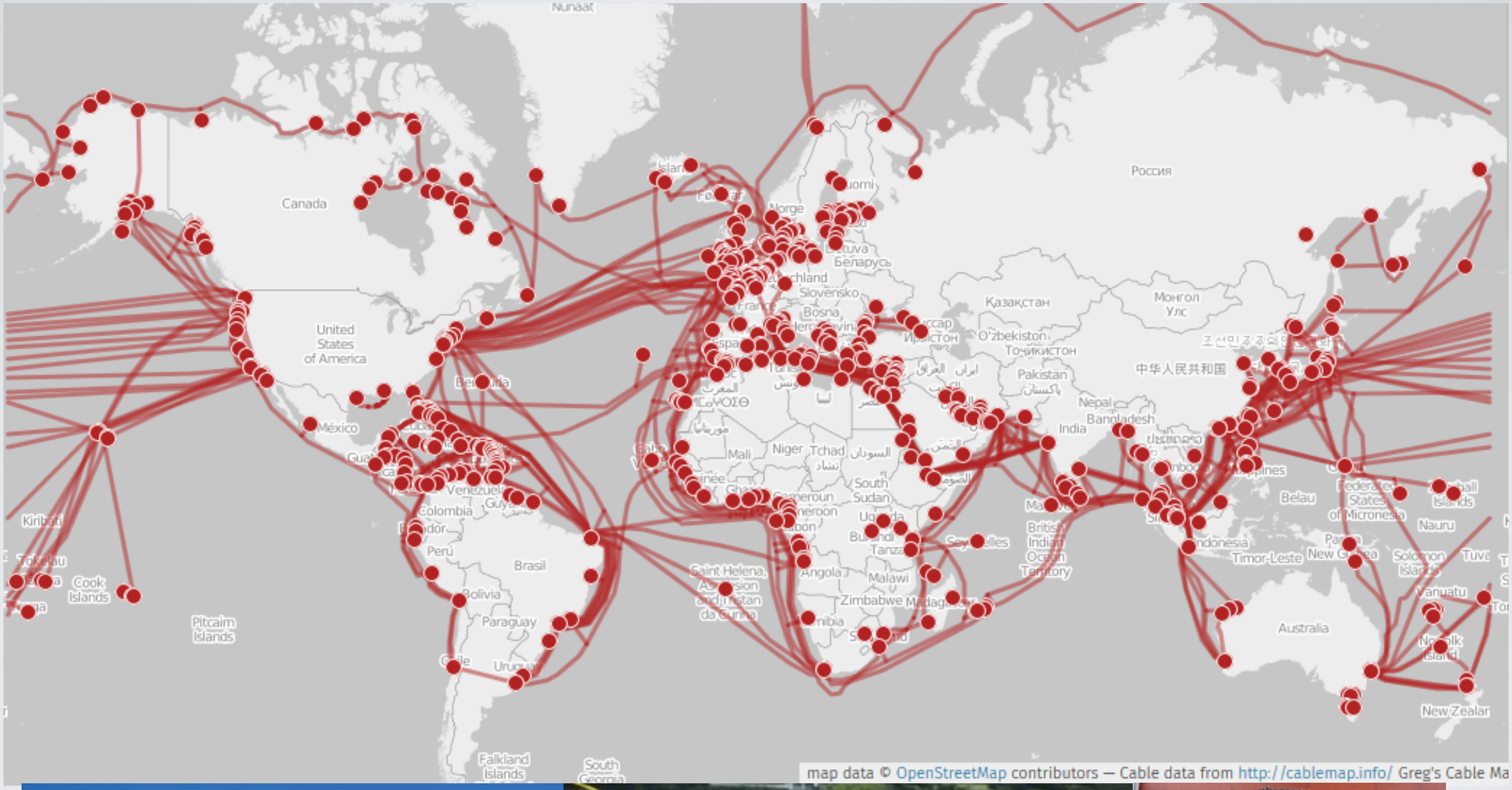
A network of networks

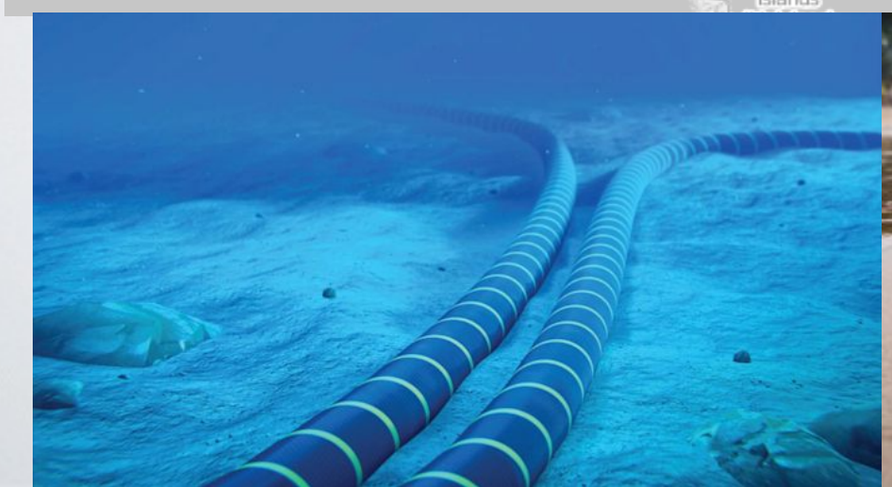
ISP - Internet Service Provider

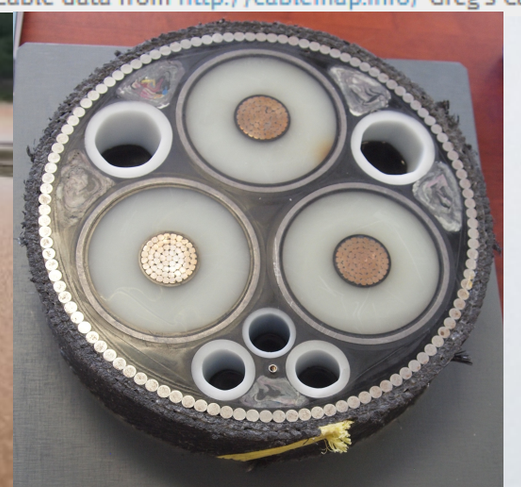
LAN - Local Network



The Internet Backbone



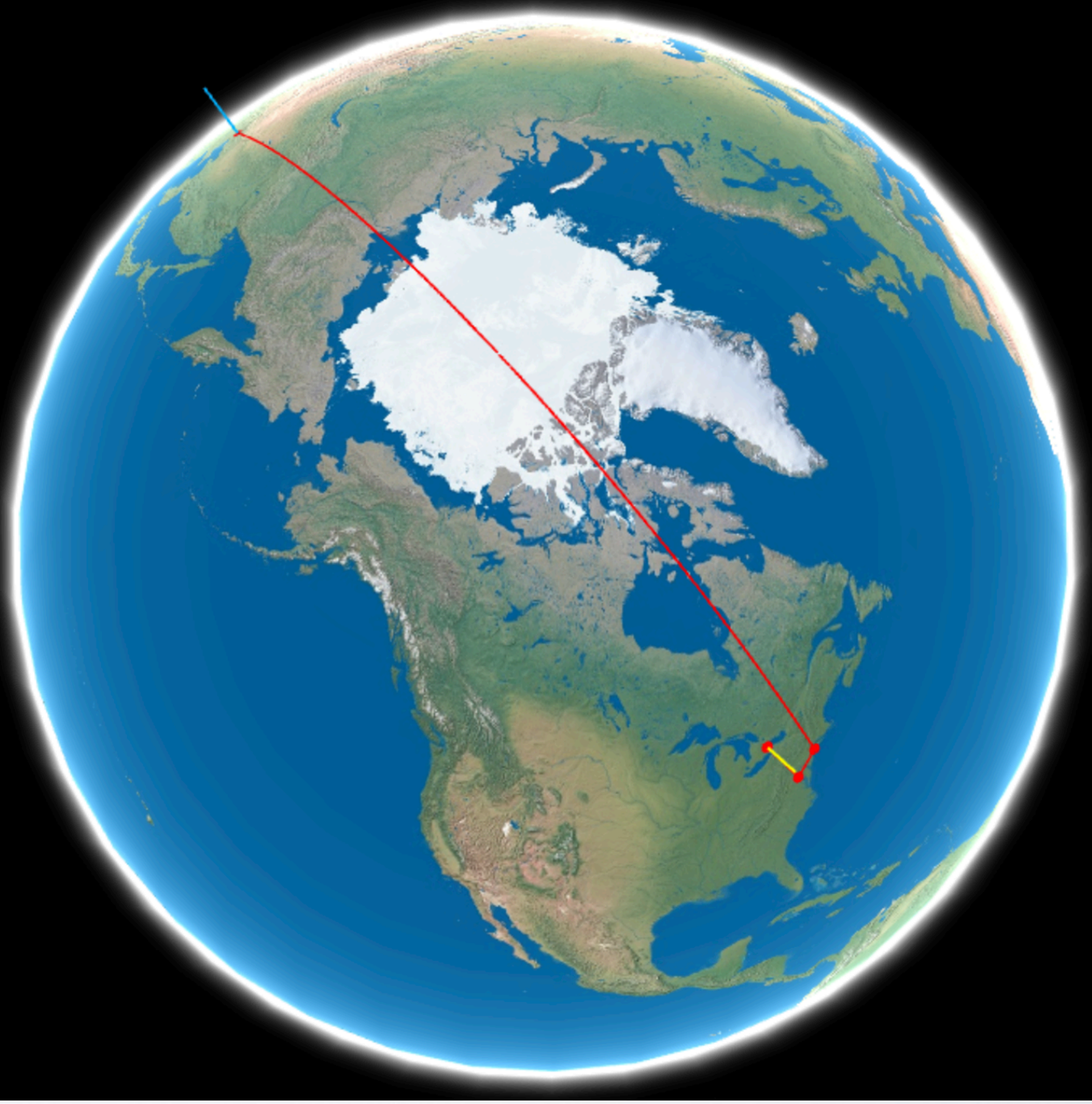




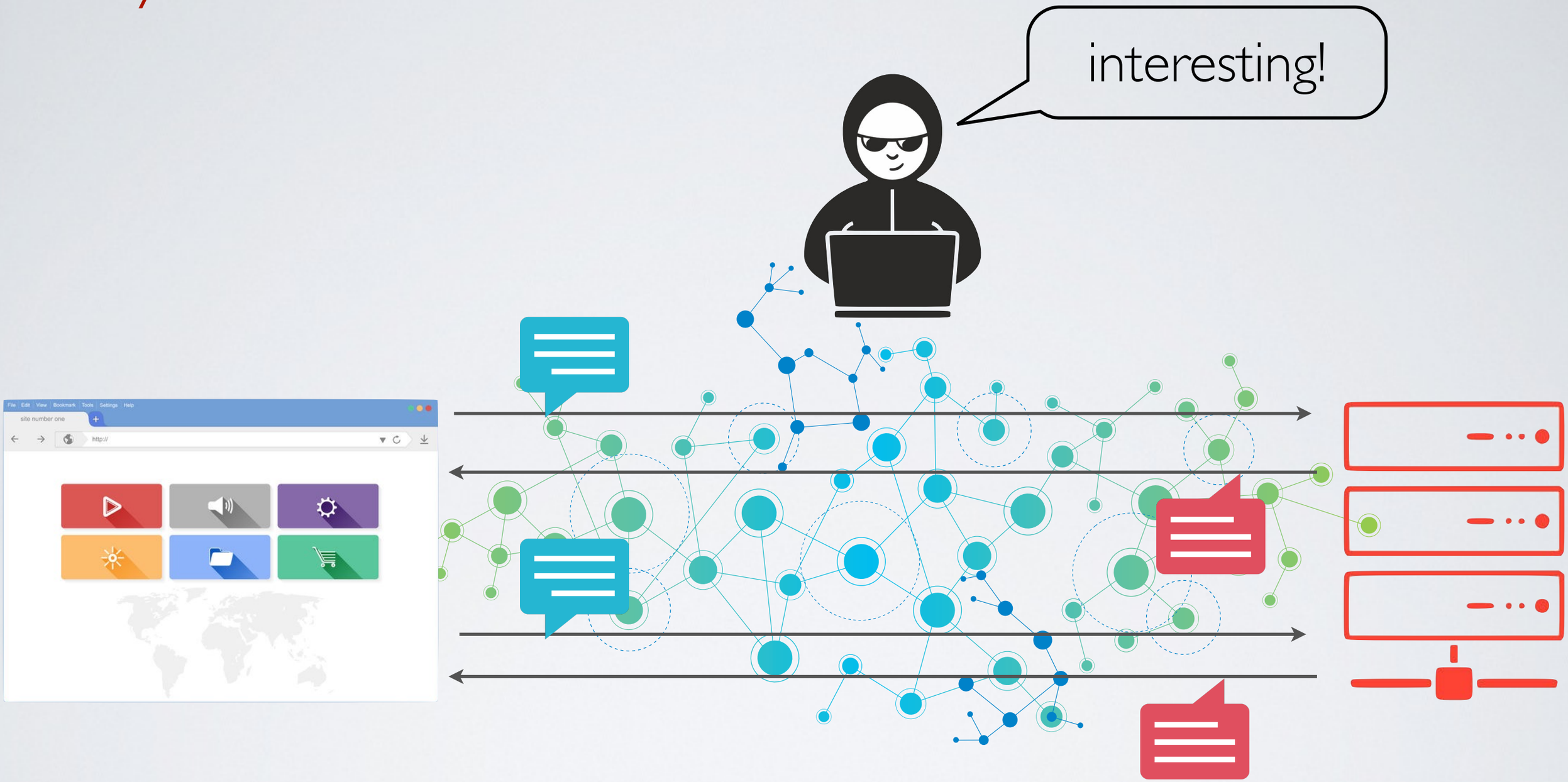
GeoTraceroute to:

baidu.com

#1 CA - Toronto (0 km)
159.203.1.207 [AS14061] (0 ms)
143.244.192.138 [AS0] (1 ms)
143.244.224.37 [AS0] (1 ms)
157.238.227.100 [AS2914] (1 ms)
#2 US - Washington (564 km)
129.250.2.141 [AS2914] (15 ms)
#3 US - New York (340 km)
129.250.3.213 [AS2914] (16 ms)
129.250.9.222 [AS2914] (22 ms)
#4 CN - Beijing (11012 km)
202.97.63.98 [AS4134] (71 ms)
202.97.41.49 [AS4134] (218 ms)
202.97.12.49 [AS4134] (218 ms)
36.110.251.78 [AS23724] (222 ms)
106.38.196.190 [AS23724] (222 ms)
182.61.255.66 [AS38365] (221 ms)
182.61.255.45 [AS38365] (221 ms)
182.61.201.211 [AS38365] (244 ms)

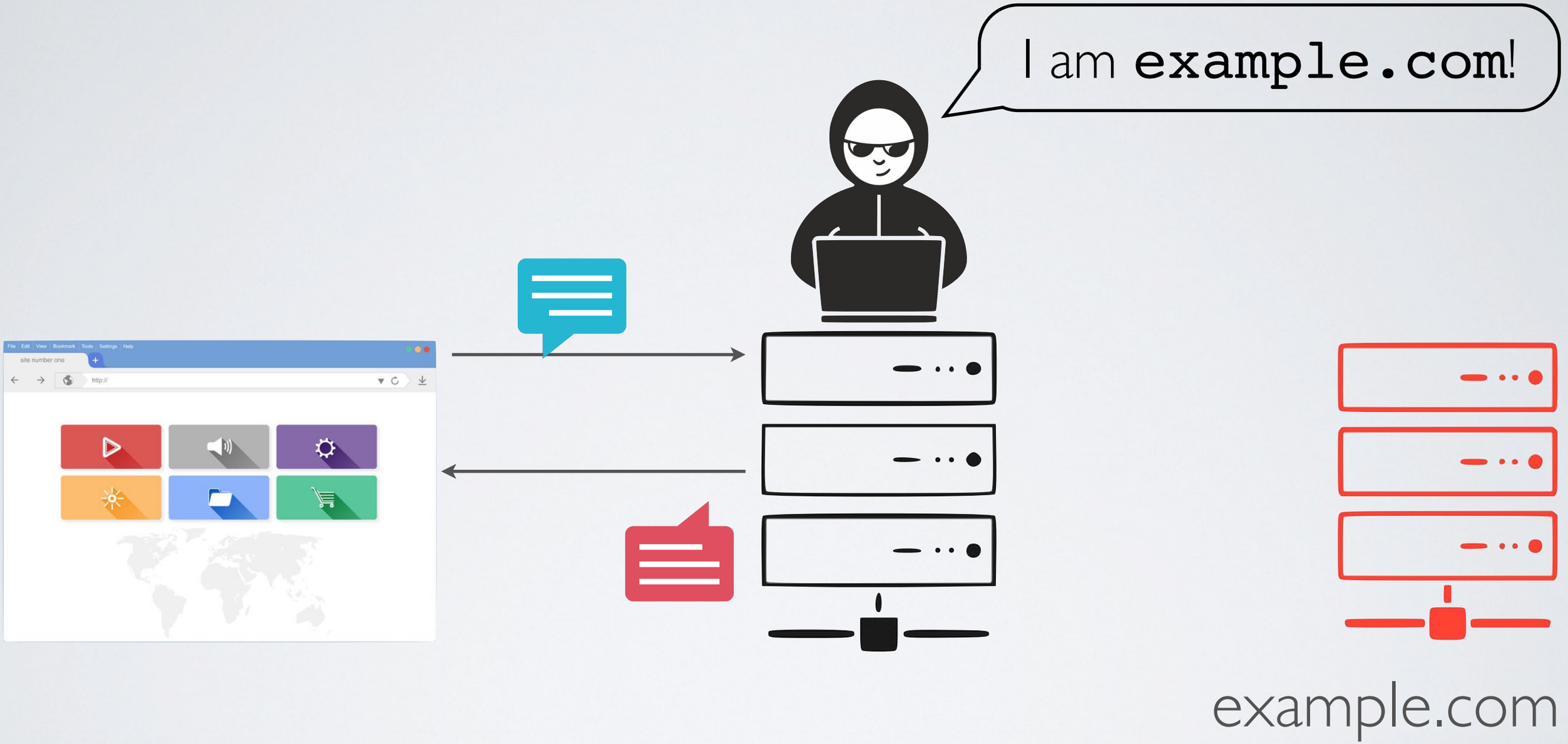


Do you trust the network?



- Threat 1 : an attacker **can eavesdrop** messages sent back and forth

Do you *really* trust the network?



- Threat 2: an attacker **can tamper with** messages sent back and forth

Confidentiality and Integrity

- Threat 1: an attacker **can eavesdrop** messages sent back and forth

  Confidentiality: How do exchange information secretly?

- Threat 2: an attacker **can tamper** with messages sent back and forth

  Integrity: How do we exchange information reliably?

HTTPS = HTTP + TLS

➡ Transport Layer Security (formerly SSL) provides

- **confidentiality:** end-to-end secure channel
- **integrity:** authentication handshake

✓ Prevents all kinds of eavesdropping and tampering protecting many internet protocols

An introduction to cryptography

Protecting confidentiality

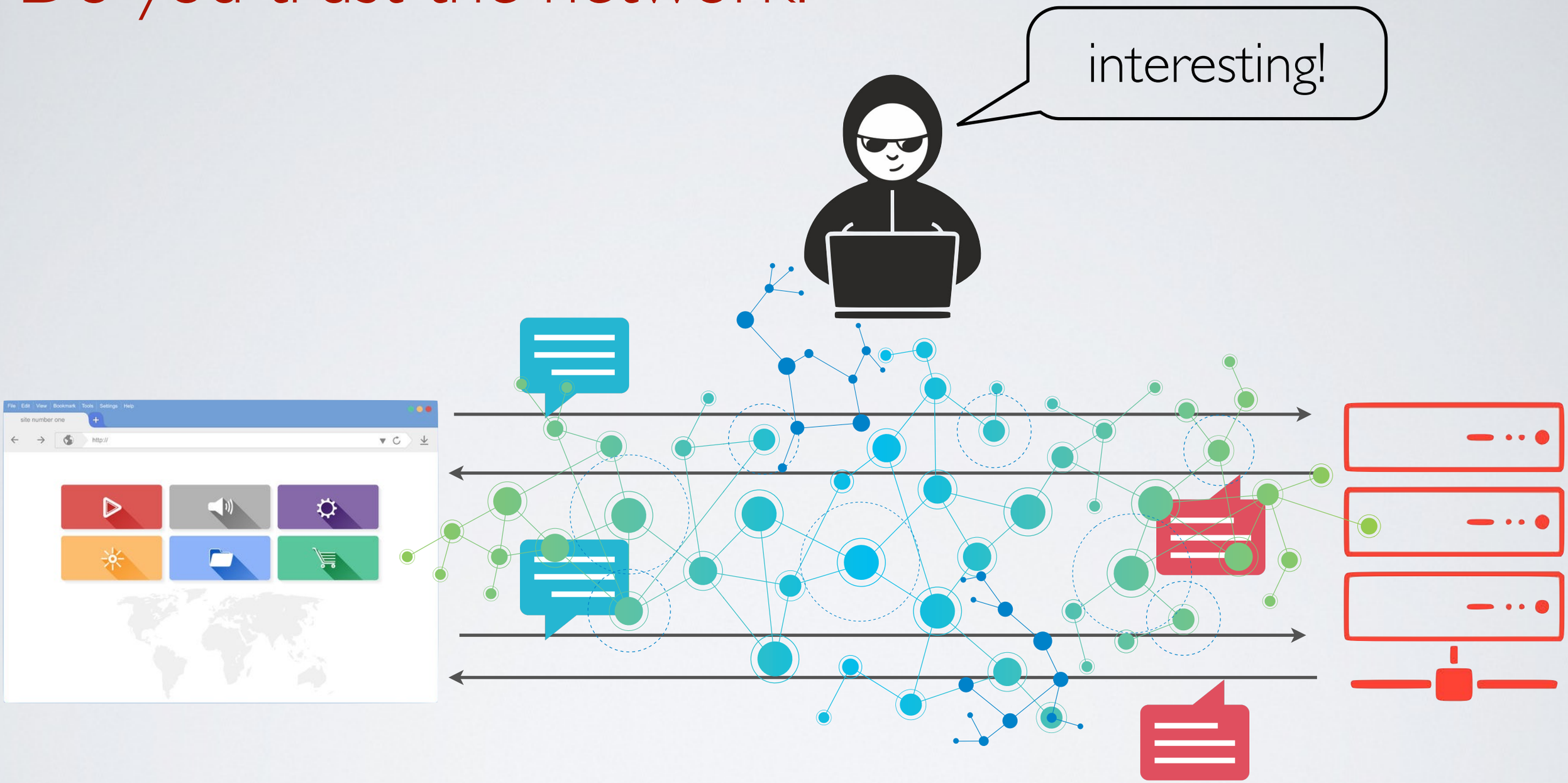
- Symmetric Encryption
- Asymmetric Encryption
- Key Exchange

Protecting Integrity

- Digital Signatures
- Public Key Infrastructure

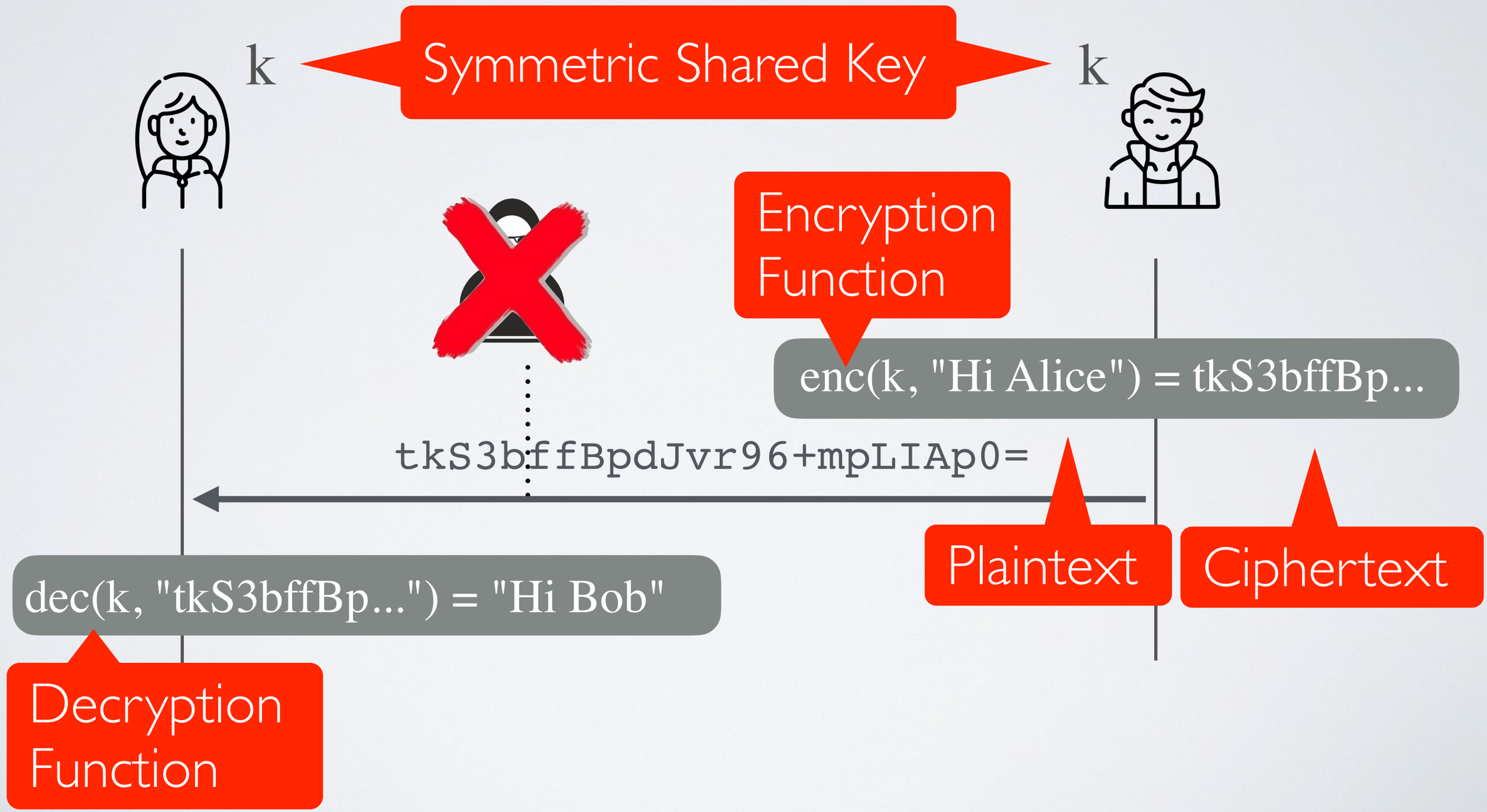
Protecting Confidentiality

Do you trust the network?



- Threat 1 : an attacker **can eavesdrop** messages sent back and forth

Symmetric Encryption to protect confidentiality



Definitions

Plaintext
The message in its original form (a.k.a "in clear")

Ciphertext
The message in its encrypted form

Encryption Function
Transform a plaintext into ciphertext

Decryption Function
Transform a ciphertext into a plaintext

Definitions

Cryptographic algorithm

The method to do encryption and decryption

✓ The cryptographic algorithm is public

Cryptographic key

An input variable used by the algorithm for the transformation

◉ The key must remain secret between the two parties

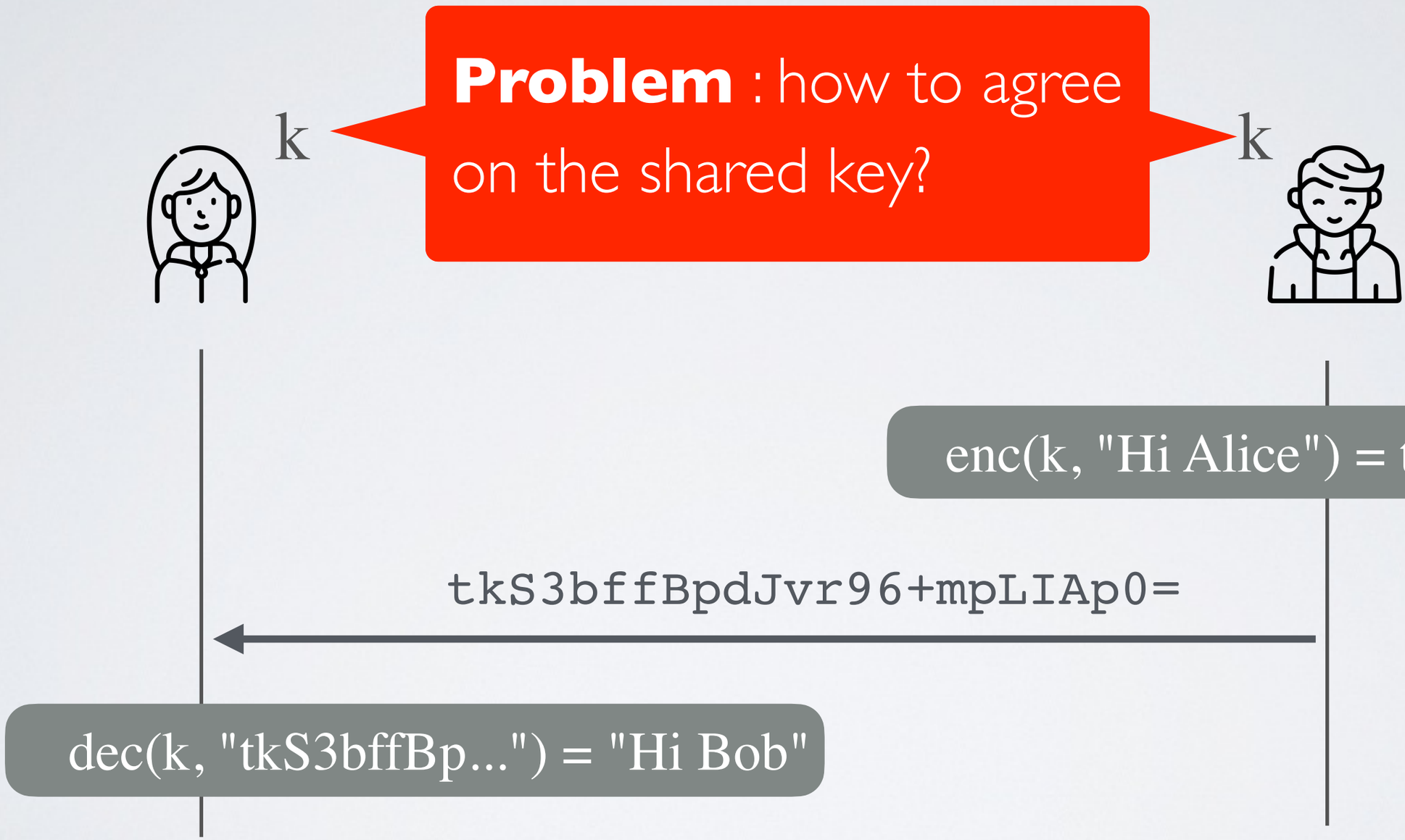
The Kerckhoffs' principle (1883)

"The enemy knows the system" - the security of a communication should **not** rely on the fact that the algorithms are secrets

➡ A cryptosystem should be secure even if **everything about the system, except the key, is public knowledge**

No security by obscurity

The big challenge with symmetric encryption



Asymmetric Encryption a.k.a Public Key Cryptography

Ks_A, Kp_A

Kp_A

Kp_A

Generate Ks_A, Kp_A

Kp_A

Alice generates a pair of asymmetric keys

- Ks_A is the secret key that Alice keeps for herself
- Kp_A is the public key that Alice gives to everyone (even Mallory)

➡ These two keys Ks_A and Kp_A work together

Asymmetric encryption **for confidentiality**

Ks_A, Kp_A

Kp_A

Kp_A

Generate Ks_A, Kp_A

Kp_A

aenc(Kp_A, "Hi Alice") = tkS3bffBp...

`tkS3bffBpdJvr96+mpLIAp0=`

adec(Ks_A, "tkS3bffBp...") = "Hi Bob"

➡ Nobody can decrypt $\mathbf{m}$, except Alice with her private key Ks_A

✓ Confidentiality without the need to exchange a secret key

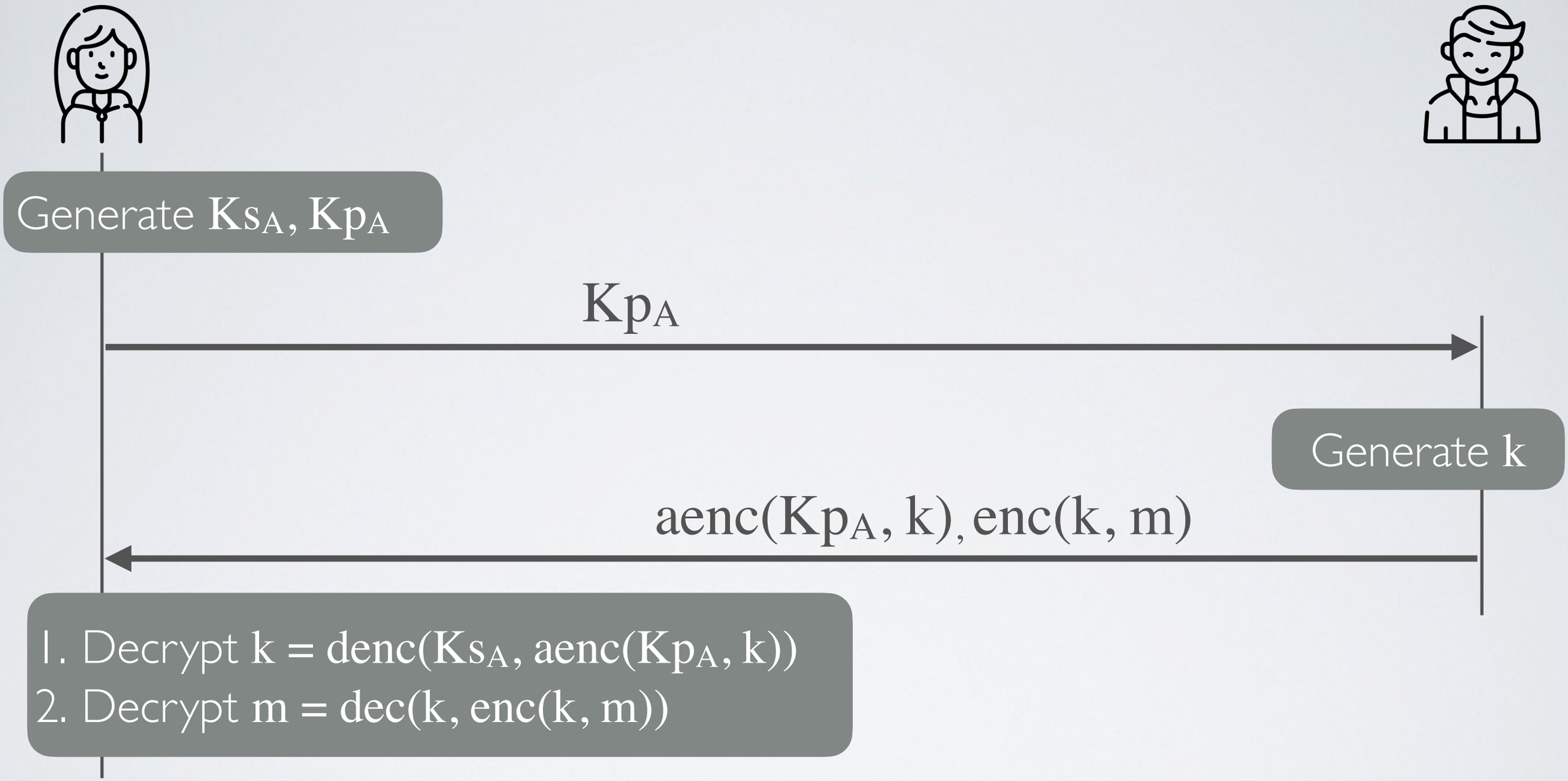
Asymmetric vs Symmetric

	Symmetric	Asymmetric
pro	Faster	No key agreement
cons	Key agreement	Slower and works on small messages only

The best of both worlds

- ➡ Use asymmetric encryption to encrypt a shared key
- ➡ Use symmetric encryption to encrypt message

(naive) key exchange using asymmetric encryption



- Protecting the shared key is the **responsibility of Alice only**
- Generating the shared key is the **responsibility of Bob only**

(better) key exchange using Diffie-Hellman-Merkle key exchange protocol

Generate Ks_A, Kp_A

Kp_A

1. Generate Ks_B, Kp_B
2. Derive $k = kex(Ks_B, Kp_A)$

Kp_B, $enc(k, m)$

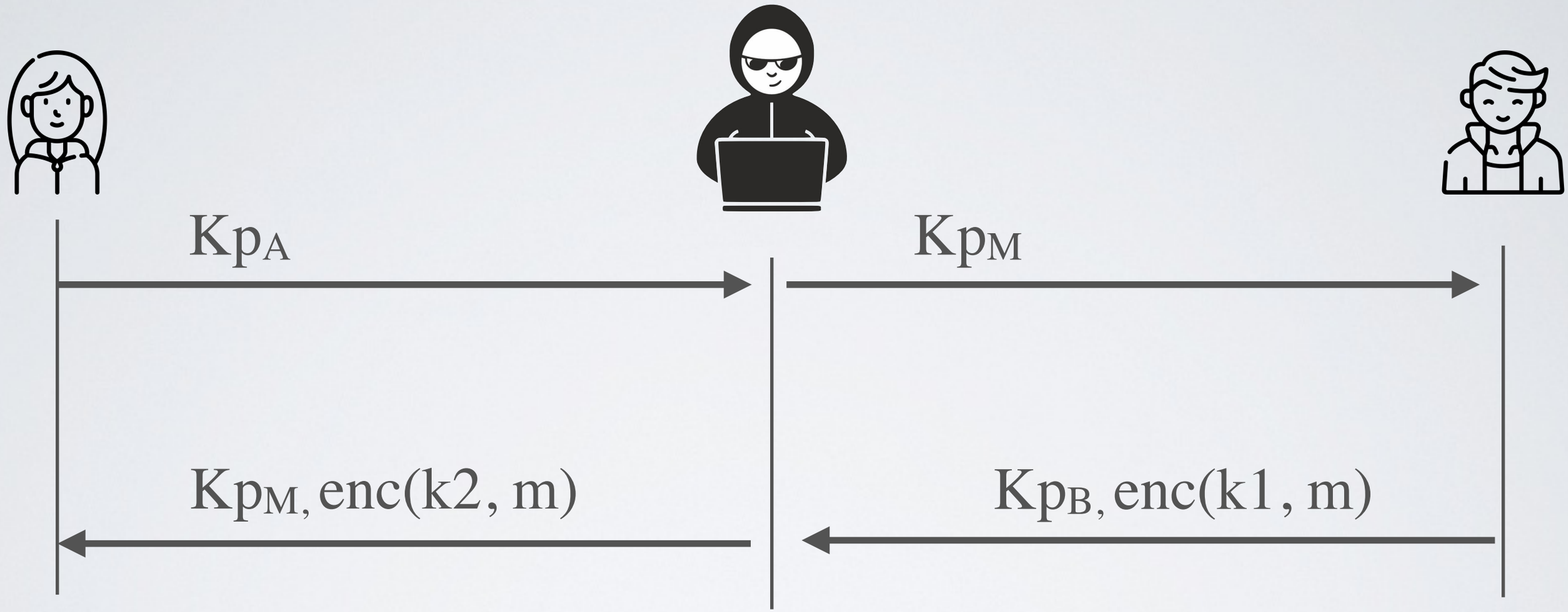
1. Derive $k = kex(Ks_A, Kp_B)$
2. Decrypt $m = dec(k, enc(k, m))$

Diffie-Hellman-Merkle provides a way to generate a shared key from two asymmetric key pairs

$$kex(Ks_A, Kp_B) = kex(Ks_B, Kp_A) = k$$

- ✓ Mutual contribution to the key generation
- ✓ No need to send the encrypted shared key

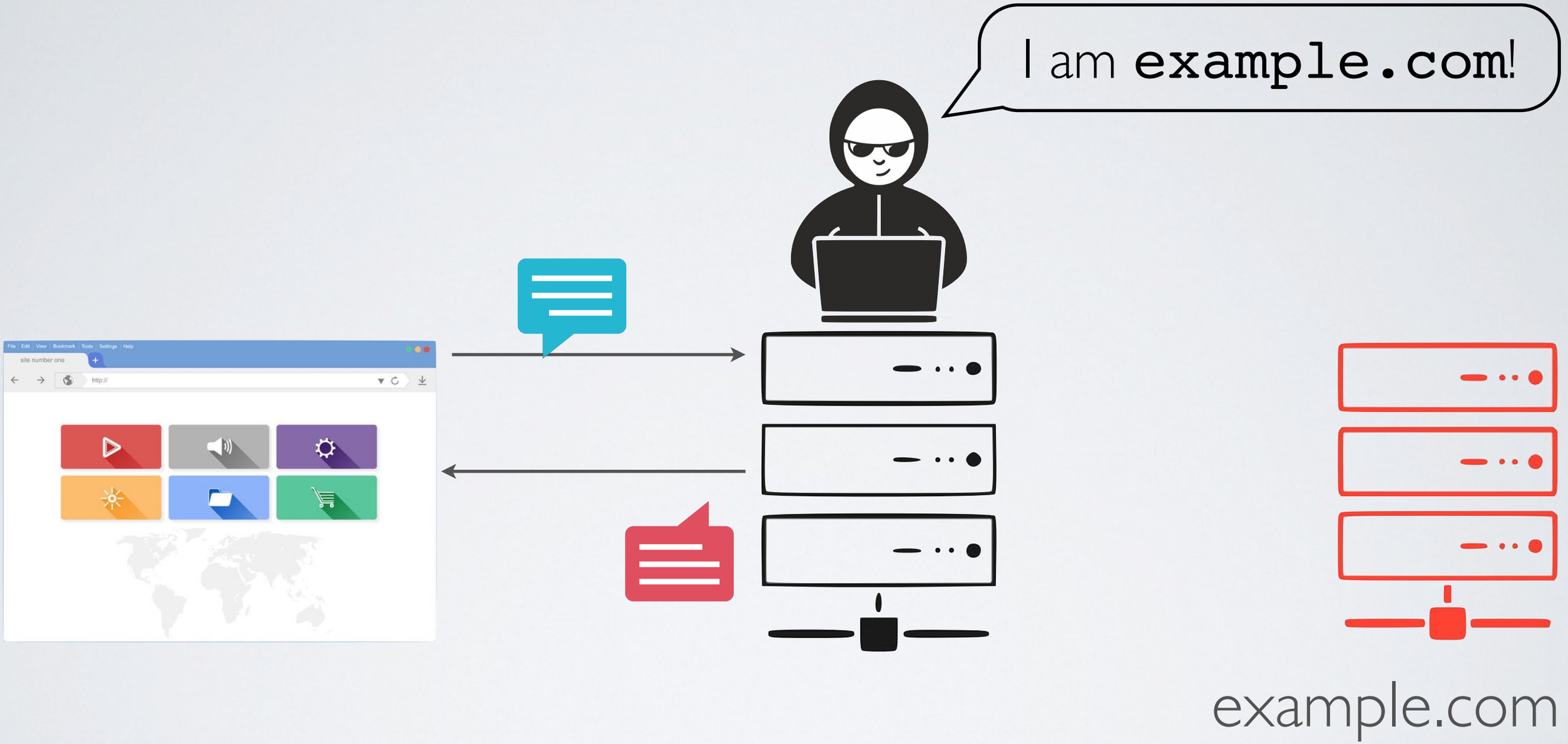
Are we done yet?



- ✓ Encryption and key exchange protects against confidentiality ...
- ◉ ... but not **does not protect integrity**

Protecting Integrity

Do you *really* trust the network?



- Threat 2: an attacker **can tamper with** messages sent back and forth

Asymmetric encryption **for Integrity**

Kp_B

Kp_B

Ks_B, Kp_B

aenc(Ks_B, "I'm Bob") = tkS3bffBp...

`tkS3bffBpdJvr96+mpLIAp0=`

adec(Kp_B, "tkS3bffBp...") = "I'm Bob"

➡ Everybody can decrypt $\mathbf{m}$ using Bob's public key $\mathbf{Kp_B}$

✓ Authentication with non-repudiation (a.k.a Digital Signature)

Digital Signatures

Kp_B

Kp_B

Ks_B, Kp_B

sign $sig_B = sign("I'm\ Bob", Ks_B)$

"I'm Bob", sig_B

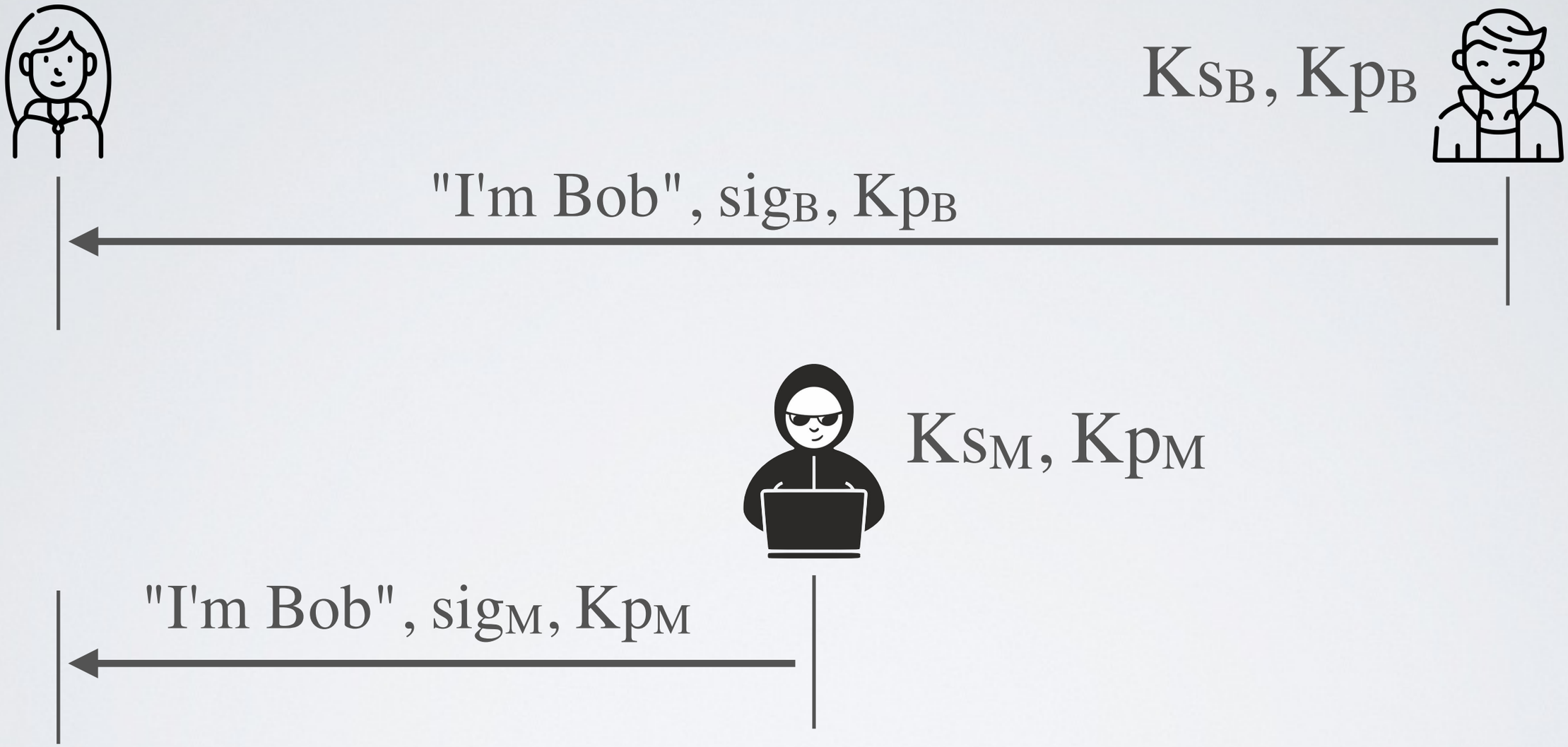
check $verify("I'm\ Bob", sig_B, Kp_B)$

Only Bob can sign a message $\mathbf{m}$ with his secret key $\mathbf{Ks_B}$

➡ Everybody can verify $\mathbf{m}$ and its $\mathbf{sig}$ using Bob's public key $\mathbf{Kp_B}$

Problem :

How does everyone knows Bob's public key?



- If Alice does not know Bob in the first place, Bob cannot just send his key and ask Alice to trust it

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

Solution 1: Online Public Key Registry

Kp_C

Ks_B, Kp_B

"I'm Bob", sig_B

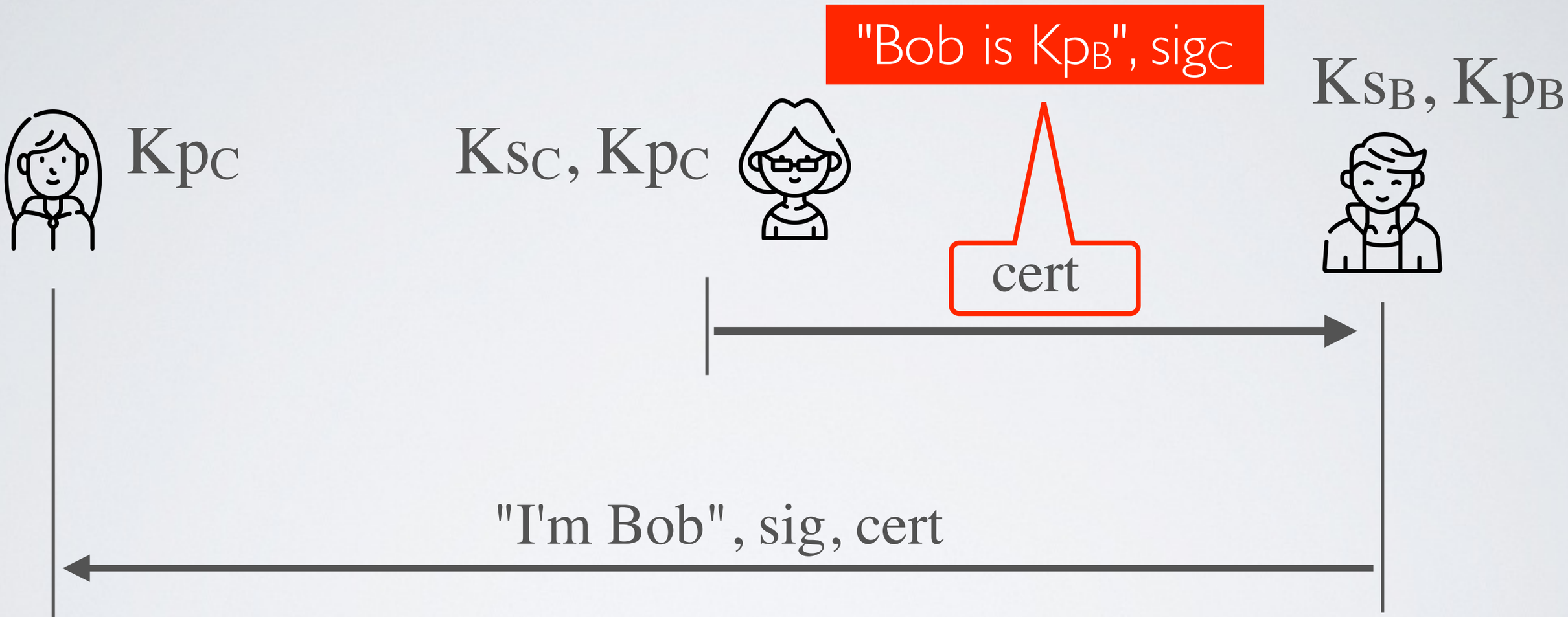
Ks_C, Kp_C

"Who is Bob?"

"Bob is Kp_B", sig_C

1. check verify("Bob is Kp_B", sig_C, Kp_C)
2. check verify("I'm Bob", sig_B, Kp_B)

Solution 2 : Offline Public Key Certification



1. check verify("Bob is Kp_B", sig_C, Kp_C)
2. check verify("I'm Bob", sig_B, Kp_B)

Putting everything together

Understanding HTTPS

HTTPS = HTTP + TLS

➡ Transport Layer Security (formerly SSL) provides

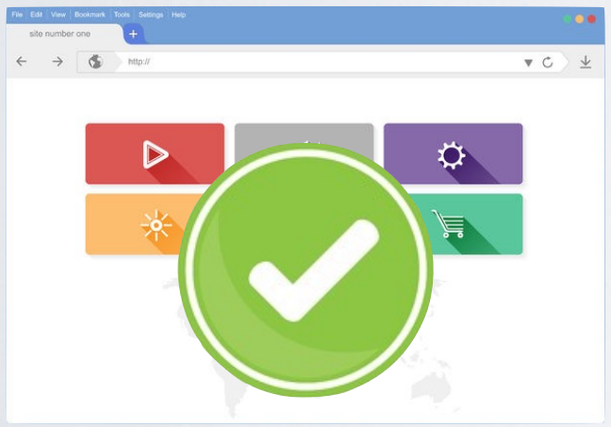
- **confidentiality:** end-to-end secure channel
- **integrity:** authentication handshake

✓ Prevents all kinds of eavesdropping and tampering protecting many internet protocols

How HTTPS works

Certificate Authority (CA)

Web Browser

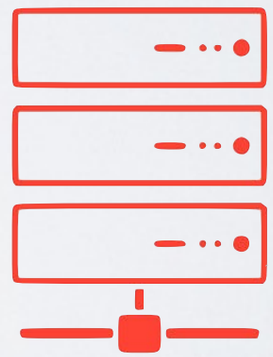


TLS request

TLS response

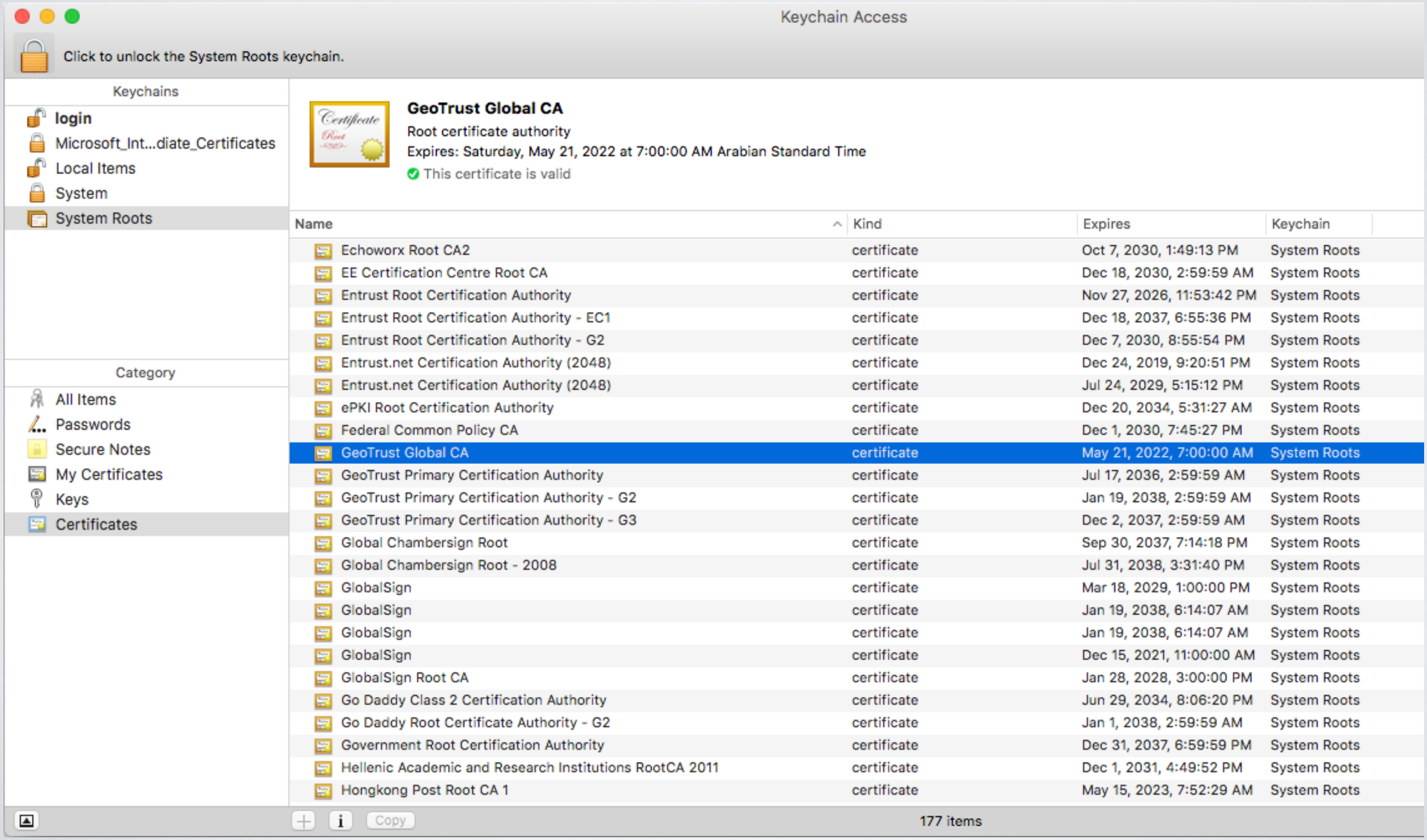
HTTP request

HTTP response



Server

Your browser trusts many CAs **by default**



Simplified TLS 1.3 (2018)

Key exchange and encryption only

Generate Ks_1, Kp_1

Kp_1

1. Generate Ks_2, Kp_2
2. Derive $k = ECDH(Ks_2, Kp_1)$

Kp_2

Derive $k = ECDH(Ks_1, Kp_2)$

enc(k, HTTP_Request)

enc(k, HTTP_Response)

Simplified TLS 1.3 (2018)

Adding one-way authentication

Kp_C

Ks_C, Kp_C

"Bob is Kp_B", sig_C

cert

Ks_B, Kp_B

Generate Ks_1, Kp_1

Kp_1

1. Generate Ks_2, Kp_2
2. Derive $k = ECDH(Ks_2, Kp_1)$
3. **Sign** $sig_B = sign(Kp_B, Kp_1 + Kp_2 + cert)$
4. **Encrypt** $enc(k, sign_B + cert)$

Kp_2, **$enc(k, sign_B + cert)$**

1. Derive $k = ECDH(Ks_1, Kp_2)$
2. **Decrypt** $sign_B + cert$ **using** k
3. **Check** $cert$ **using** Kp_C
4. **Check** $verify(Kp_1 + Kp_2 + cert, sign_B, Kp_B)$

enc(k, HTTP_Request)

enc(k, HTTP_Response)

This afternoon,
let's implement HTTPS in Python