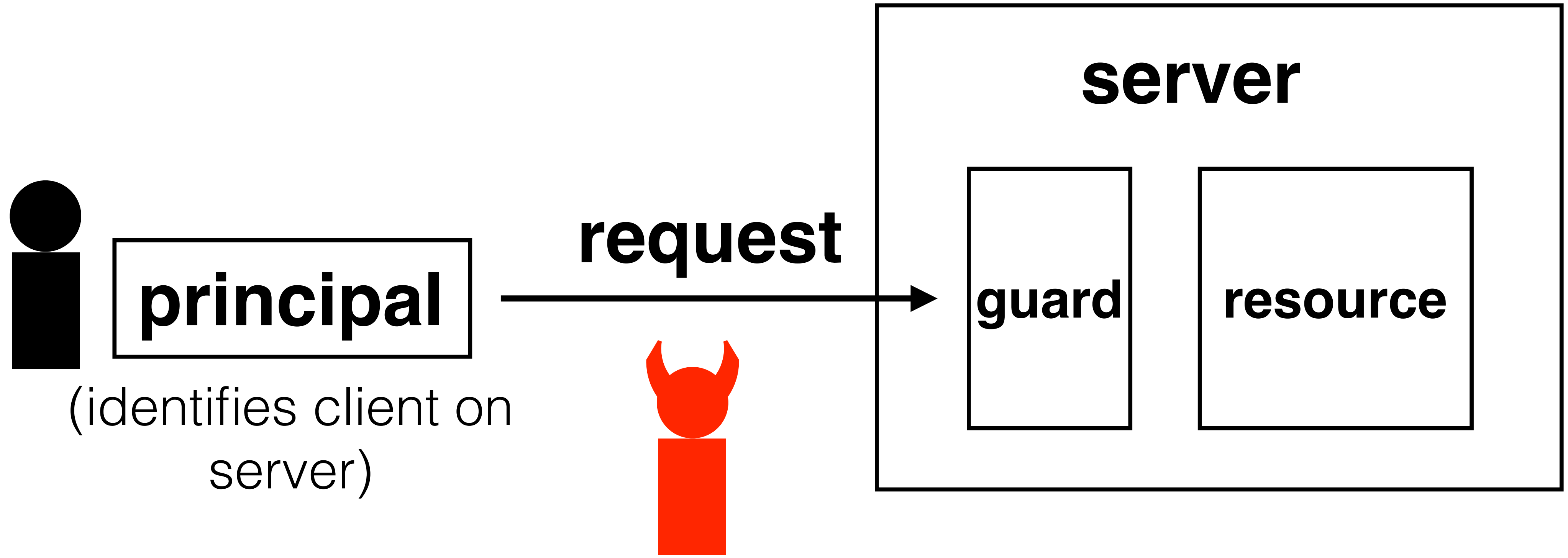
6.033 Spring 2015

Lecture #23

- **Combating network adversaries**
  - **Secure Channels**
  - **Signatures**



confidentiality: adversary cannot learn message contents

integrity: adversary cannot tamper with message contents
(if they do, client and/or server will detect it)

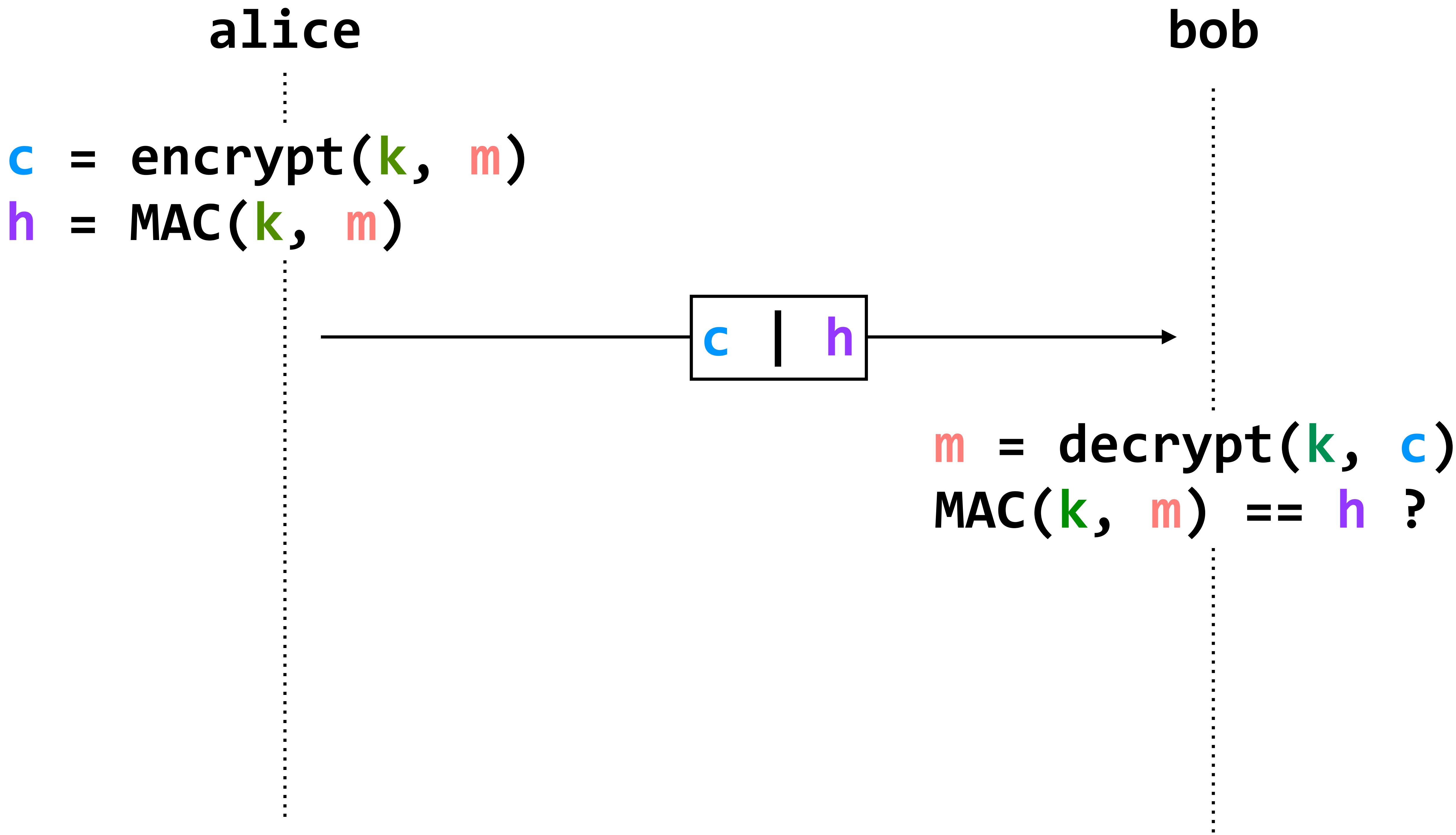
```
encrypt(key, message) → ciphertext
decrypt(key, ciphertext) → message
```

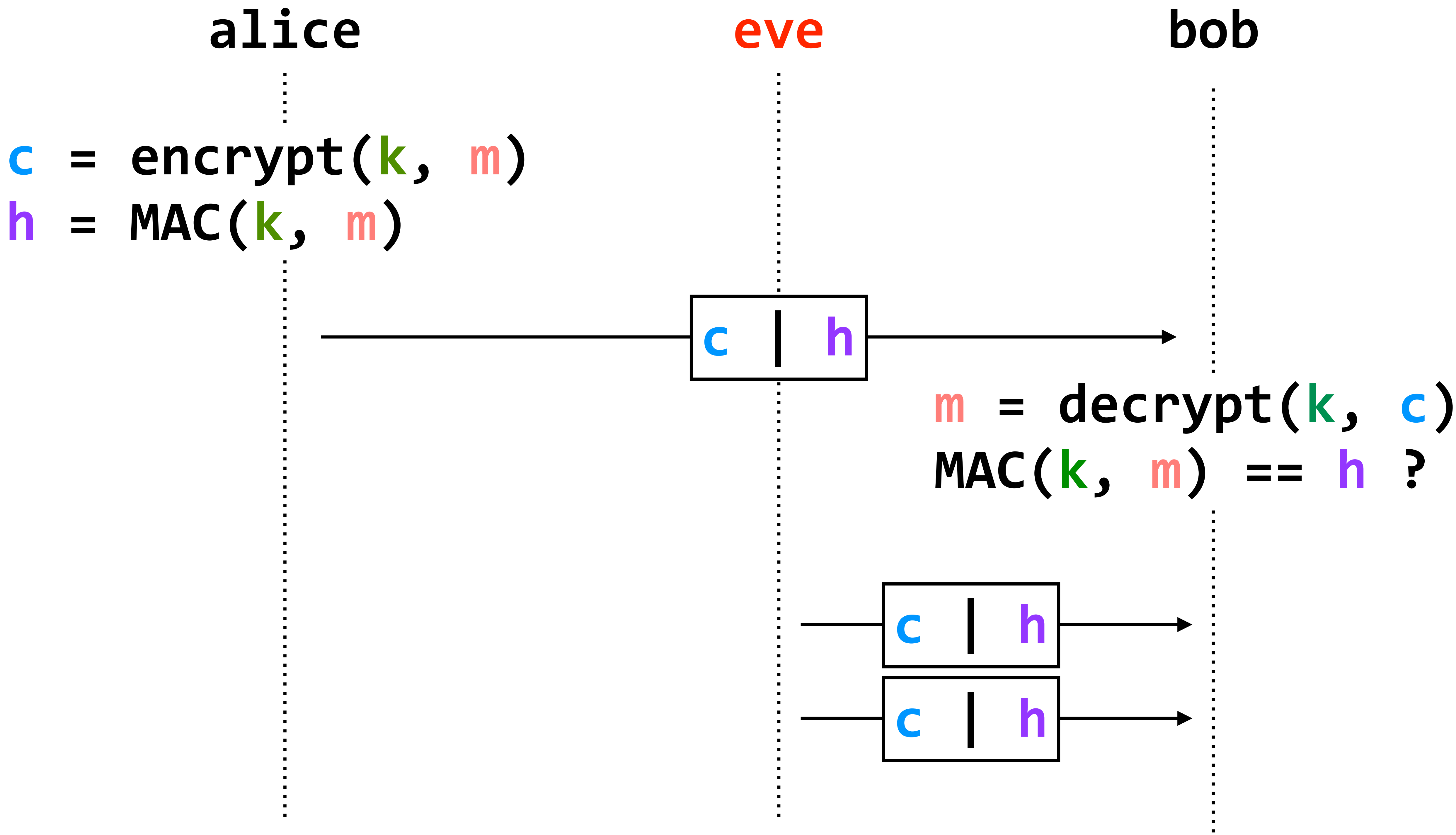
property: given the `ciphertext`, it is (virtually) impossible to obtain the `message` without knowing the `key`

```
MAC(key, message) → token
```

property: given the `message`, it is (virtually) impossible to obtain the `token` without knowing the `key`

(it is also impossible to go in the reverse direction)


alice
bob
c = encrypt(k, m)
h = MAC(k, m)
c | h
m = decrypt(k, c)
MAC(k, m) == h ?



problem: replay attacks

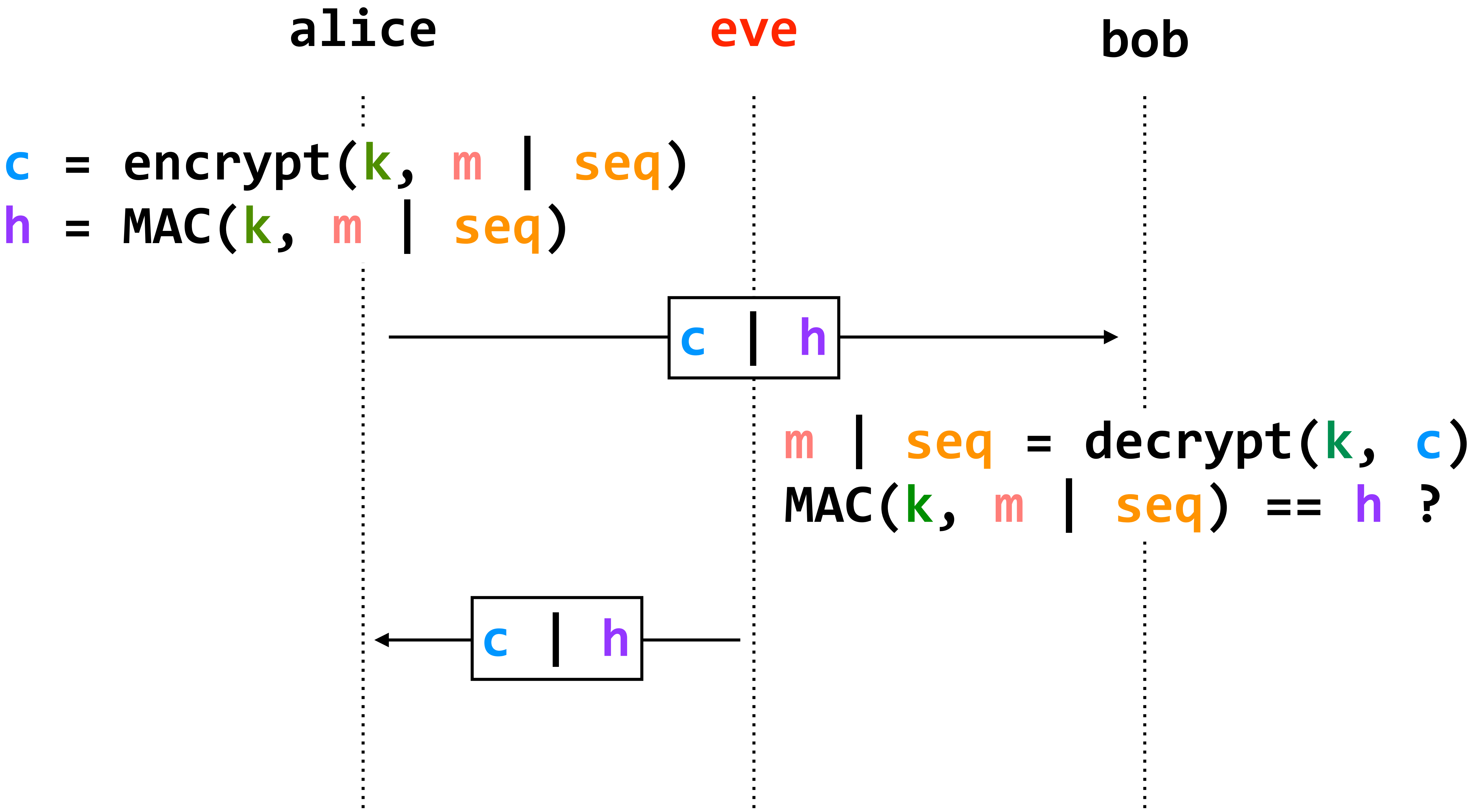
(adversary could intercept a message, re-send it at a later time) alice

bob

```
c = encrypt(k, m | seq)
h = MAC(k, m | seq)
```

c | h

```
m | seq = decrypt(k, c)
MAC(k, m | seq) == h ?
```



problem: reflection attacks

(adversary could intercept a message, re-send it at a later time in the opposite direction)

alice **bob**

$c_a = \text{encrypt}(k_a,\ m_a \mid seq_a)$

$h_a = \text{MAC}(k_a,\ m_a \mid seq_a)$

$\xrightarrow{\boxed{c_a \mid h_a}}$

$m_a \mid seq_a = \text{decrypt}(k_a,\ c_a)$

$\text{MAC}(k_a,\ m_a \mid seq_a) == h_a\ ?$

$c_b = \text{encrypt}(k_b,\ m_b \mid seq_b)$

$h_b = \text{MAC}(k_b,\ m_b \mid seq_b)$

$\xleftarrow{\boxed{c_b \mid h_b}}$

$m_b \mid seq_b = \text{decrypt}(k_b,\ c_b)$

$\text{MAC}(k_b,\ m_b \mid seq_b) == h_b\ ?$

problem: how do the parties know the keys?

known: `p (prime), g`

property: given $g^r \bmod p$, it is (virtually) impossible to determine **r** *even if* you know **g** and **p**

alice		**bob**
pick random a		pick random b
	$g^a \bmod p$ →	
	← $g^b \bmod p$	
calculate $(g^b)^a \bmod p$		calculate $(g^a)^b \bmod p$

key $= g^{ab} \bmod p$

alice **eve** **bob**

pick random a pick random e pick random b

$g^a \bmod p$

$g^e \bmod p$

$g^e \bmod p$

$g^b \bmod p$

$k_1 = (g^e)^a \bmod p$

$k_2 = (g^e)^b \bmod p$

eve can calculate k_1 and k_2

encrypt(k_1, m)

decrypt m

encrypt(k_2, m)

problem: alice and bob don't know they're not communicating directly

cryptographic signatures

allow users to verify identities using public-key cryptography

```
sign(secret_key, message) → sig
verify(public_key, message, sig) → yes/no
```

TLS handshake

client **server**

ClientHello {version, seq_c, session_id, cipher suites, compression func} →

← ServerHello {version, seq_s, session_id, cipher suite, compression func}

← {server certificate, CA certificates}

← ServerHelloDone

client verifies authenticity of server

ClientKeyExchange {encrypt(server_pub_key, pre_master_secret)} →

compute

```
master_secret = PRF(pre_master_secret, “master secret”, seqc | seqs)
    key_block = PRF(master_secret, “key expansion”, seqc | seqs)
              = {client_MAC_key,
                 server_MAC_key,
                 client_encrypt_key,
                 server_encrypt_key,
                 ...}
```

Finished {sign(client_MAC_key, encrypt(client_encrypt_key, MAC(master_secret, previous_messages)))} →

← Finished {sign(server_MAC_key, encrypt(server_encrypt_key, MAC(master_secret, previous_messages)))}

- **Secure channels** protect us from adversaries that can observer and tamper with packets in the network.
- Encrypting with **symmetric keys** provides secrecy, and using **MACs** provides integrity. **Diffie-Hellman key exchange** lets us exchange the symmetric key securely.
- To verify identities, we use **public-key cryptography** and cryptographic **signatures**. We often distribute public keys with **certificate authorities**, though this method is not perfect.