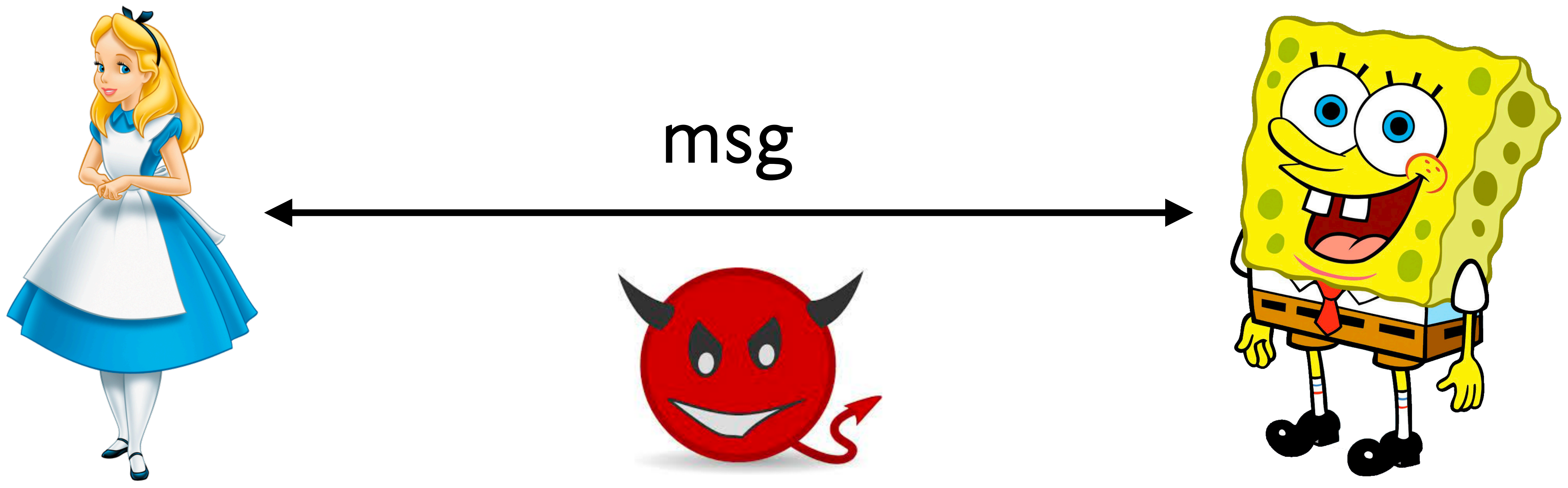
CS 350S: Privacy-Preserving Systems

Cryptography basics

Today: Outline

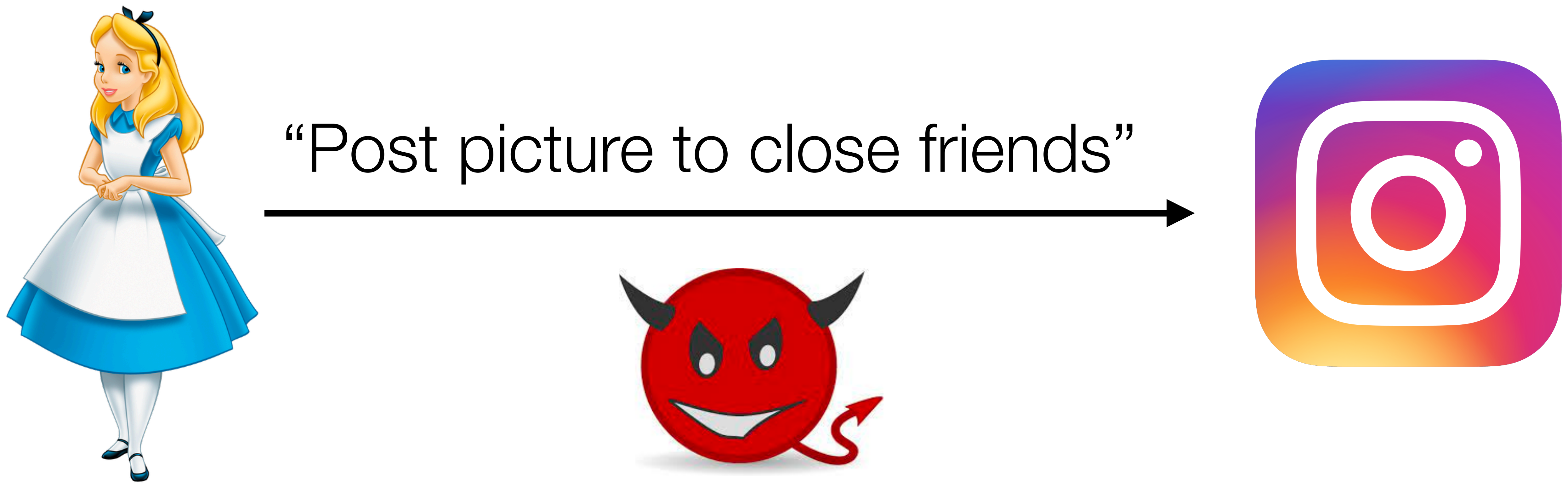
1. Symmetric-key cryptography
    - Message integrity
2. Public-key cryptography
3. Web public-key infrastructure
4. Authenticated data structures

From last time: message integrity



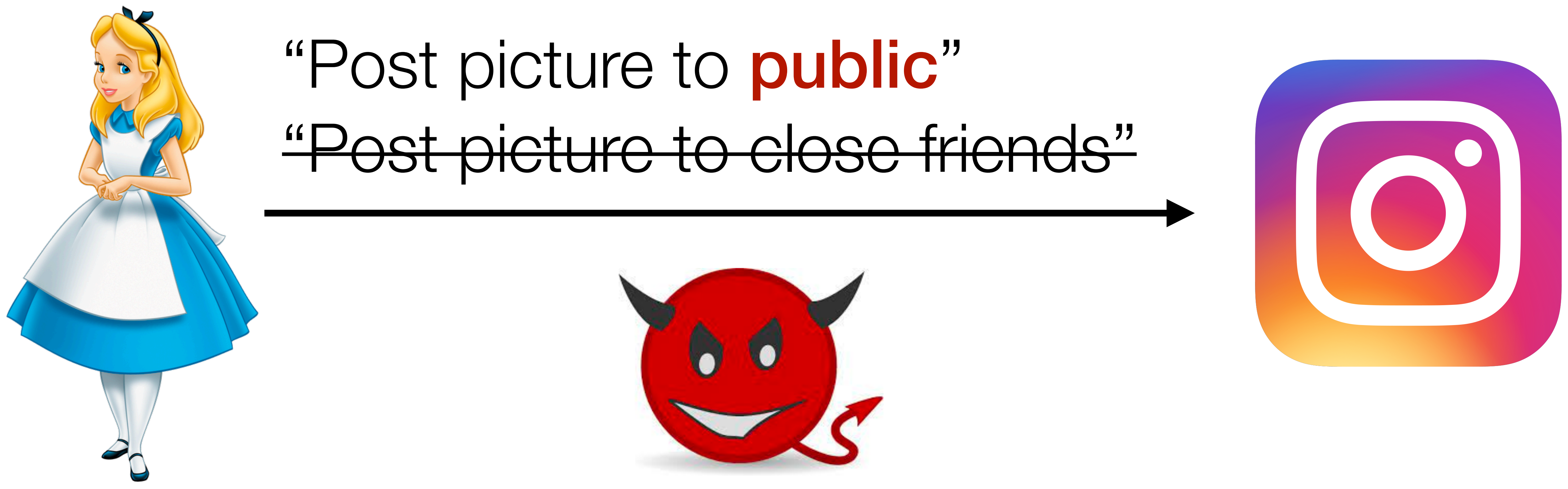
Integrity: Alice and Bob want to ensure that the attacker has not tampered with the message

Confidentiality and integrity go together



Depending on application, attacker can change a ciphertext and cause application-level damage

Confidentiality and integrity go together



Depending on application, attacker can change a ciphertext and cause application-level damage

Takeaway: use authenticated encryption (e.g., AES in GCM mode)

Collision-resistant hash functions

Input space $\{0,1\}^*$
Output space $\{0,1\}^{256}$

Hash function $H : \{0,1\}^* \rightarrow \{0,1\}^{256}$

A hash function H is collision-resistant if no “efficient” adversary can find a collision, i.e. $H(a) = H(b)$ for $a \neq b$

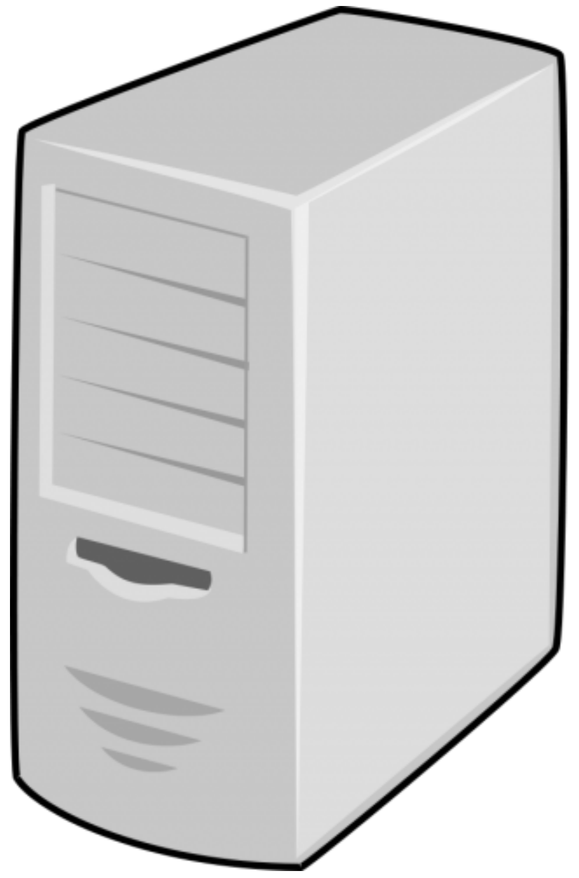
Note: many collisions exist, just hard to find

Example of collision-resistant hash function: SHA256

Application: file storage

$d = H(f)$

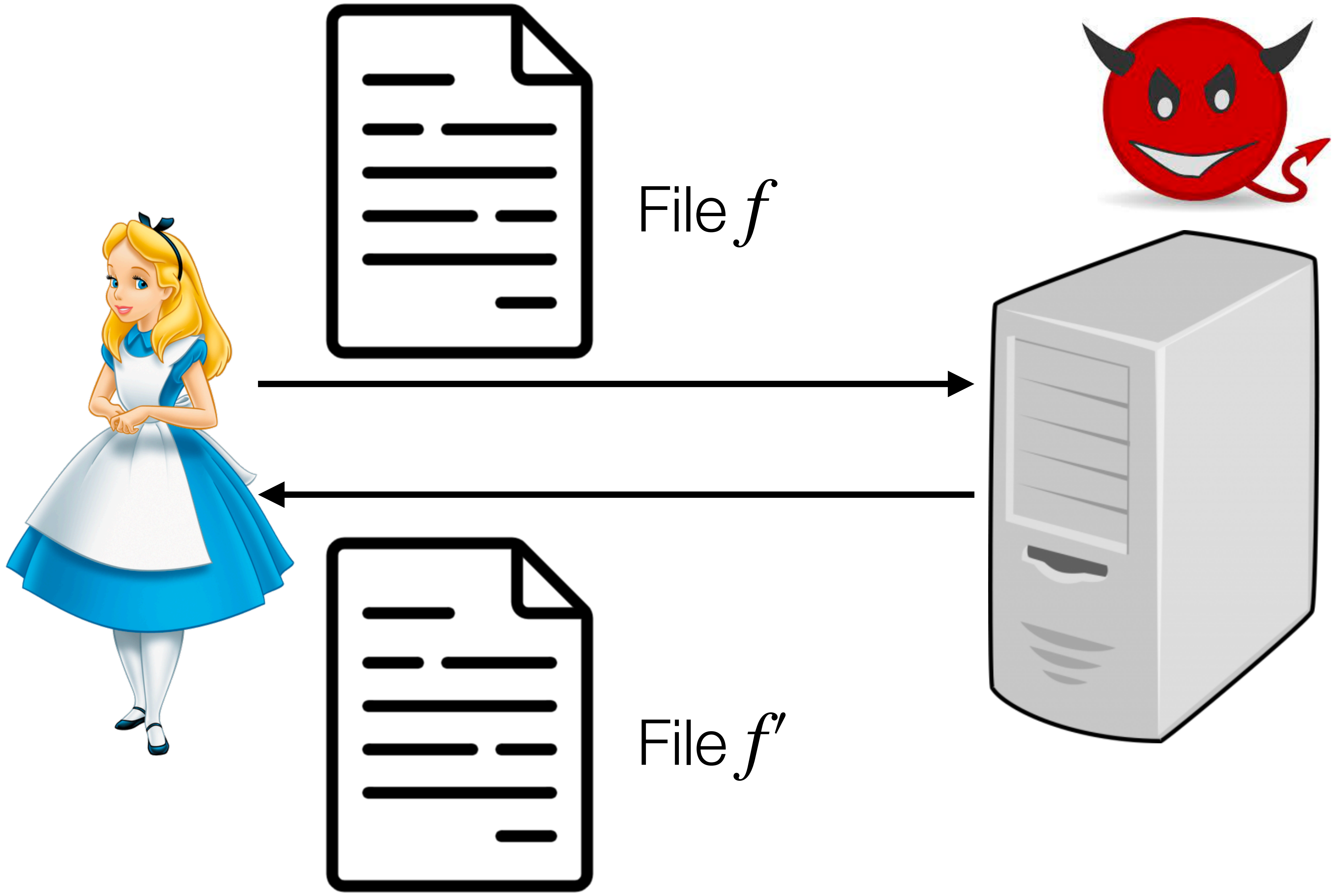
File f



Application: file storage

$d = H(f)$

Check: $d = H(f')$?



Next: Can we avoid the client storing a hash digest for each file?

Message authentication codes (MACs)

Key space K

Message space M

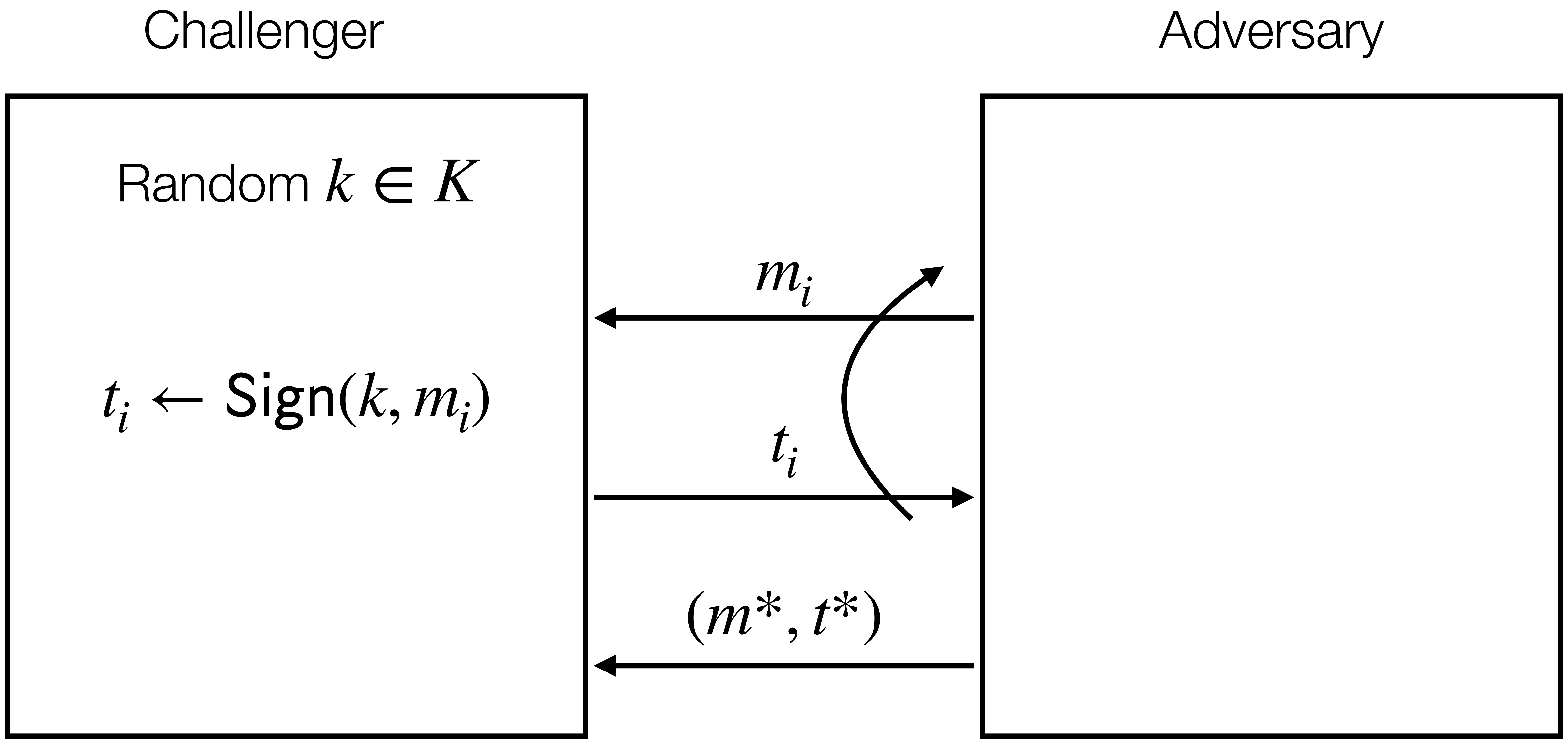
Tag space T

Message authentication codes:

- $\mathsf{Sign} : K \times M \to T$
- $\mathsf{Verify} : K \times M \times T \to \{0,1\}$

What is the right way to define security?

Chosen message attack security



Adversary wins if:

- $(m^*, t^*) \notin \{(m_1, t_1), (m_2, t_2), \ldots\}$
- $\mathsf{Verify}(k, m^*, t^*) = 1$ ("accept")

MACs from PRFs

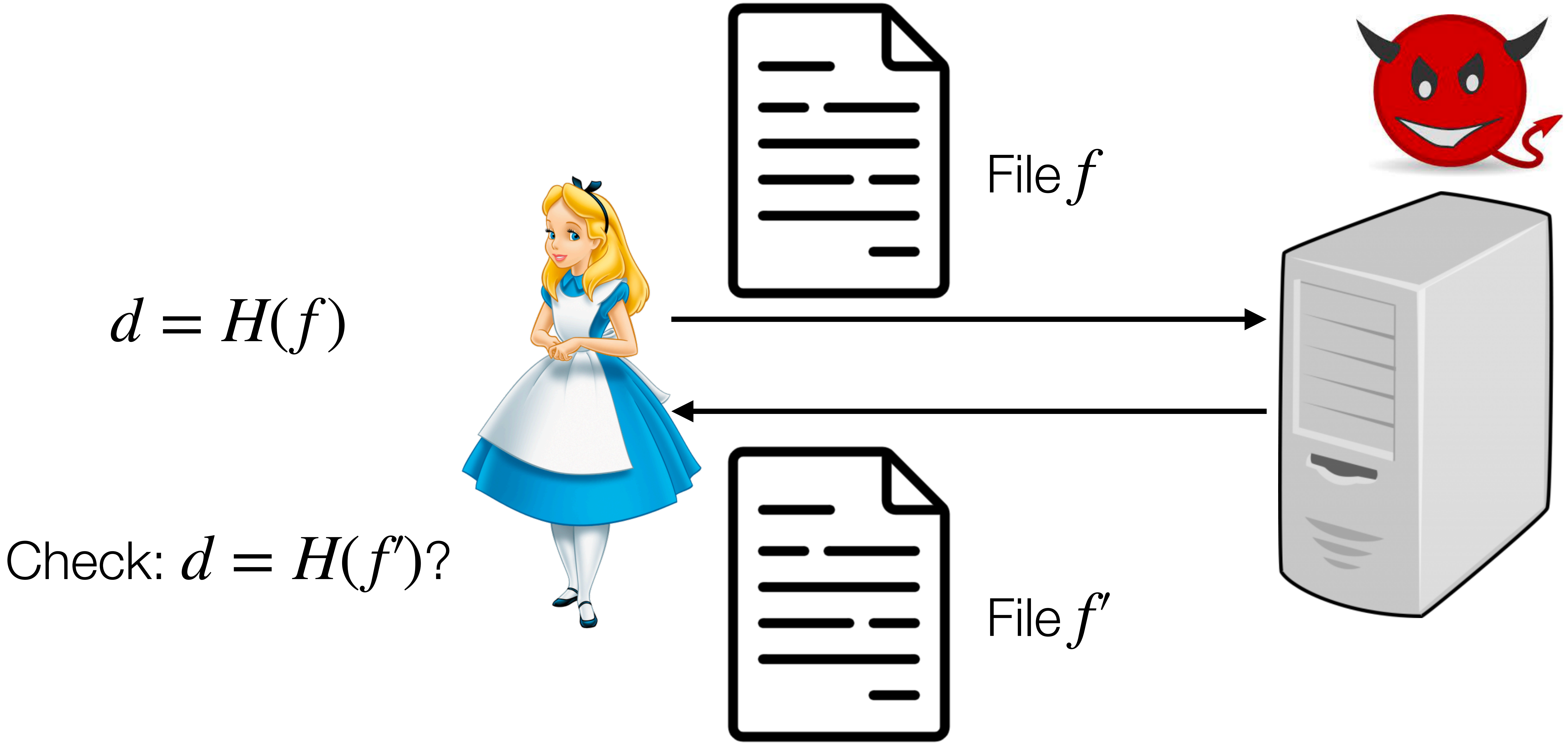
Given PRF F:

- $\mathsf{Sign}(k, m)$: $F(k, m)$
- $\mathsf{Verify}(k, m, t)$: Output 1 (“accept”) if $F(k, m) = t$, 0 (“reject”) otherwise

Note: this MAC construction is deterministic

Another example of MAC: HMAC (from SHA256)

Revisiting file storage

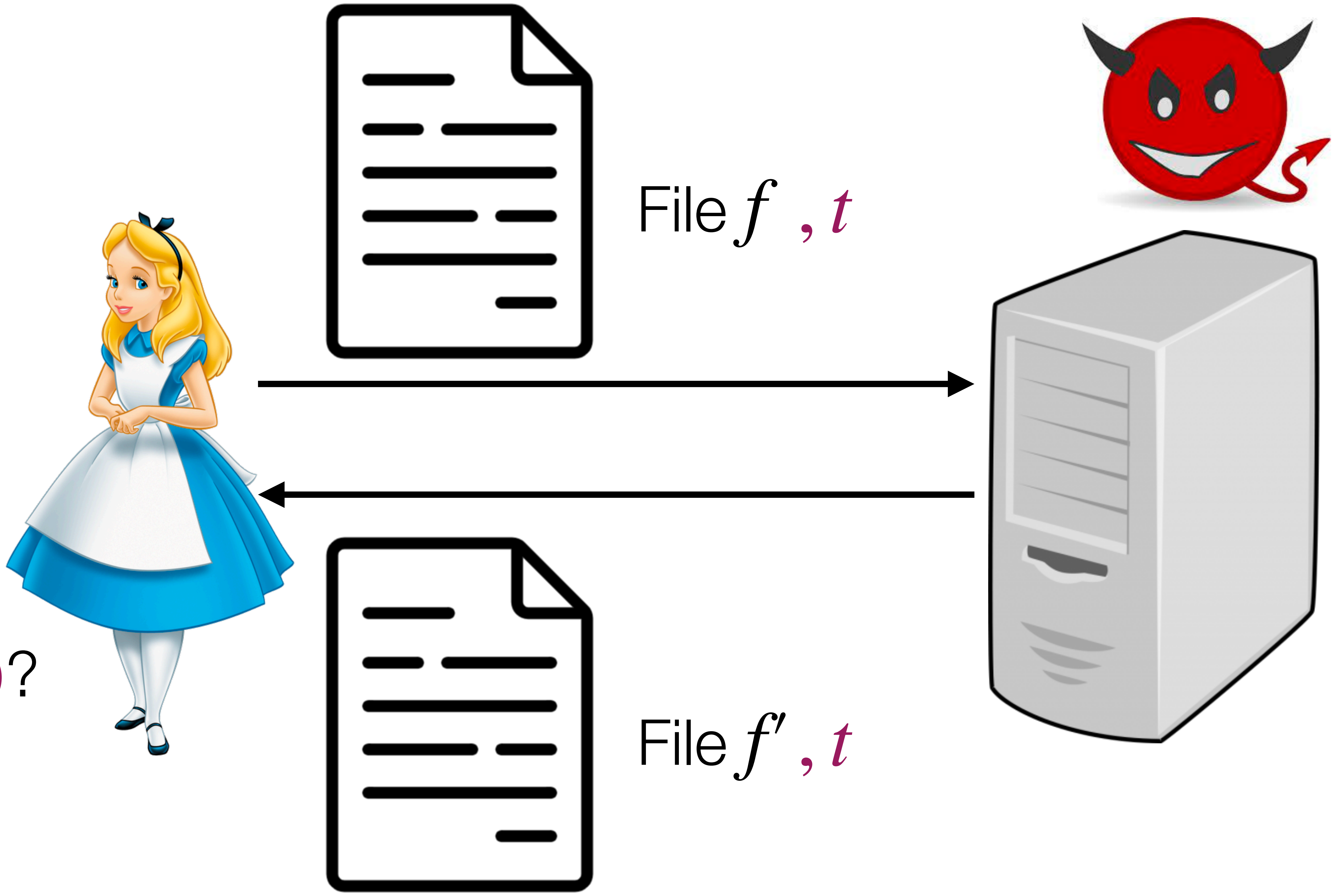


Can we avoid the client storing a hash digest for each file?

Revisiting file storage

Key k

$t = \text{Sign}(k, f)$



Check: $\text{Verify}(k, f', t)$?

Alice can outsource many files and only store 1 key

Q: other applications for MACs?

Public-key cryptography

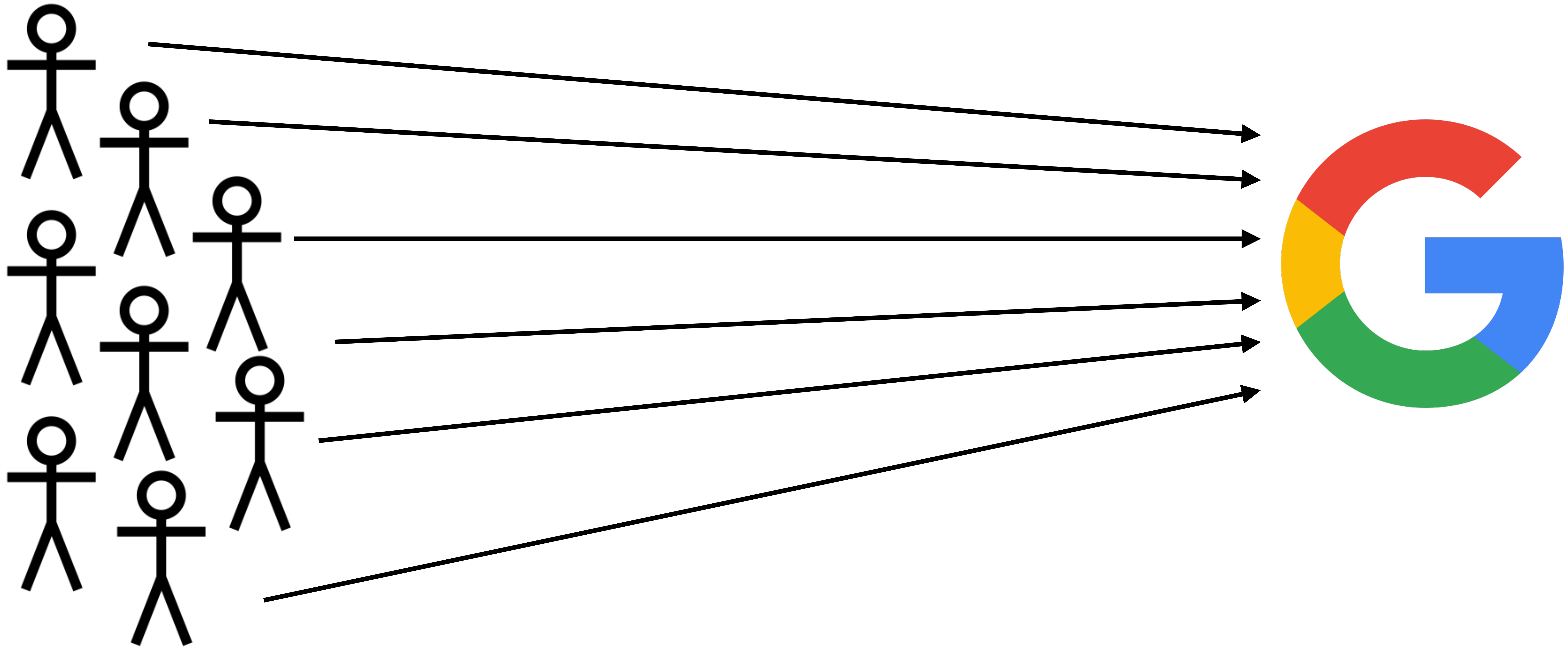
Symmetric key vs. public key cryptography

Symmetric-key cryptography
Same key for:
- Encryption + decryption
- Signing + verifying MAC tags

Public-key cryptography
Different keys for:
- Encryption + decryption
- Signing + verifying signatures

Public-key encryption

Public key **pk**



Secret key **sk**

Public-key encryption

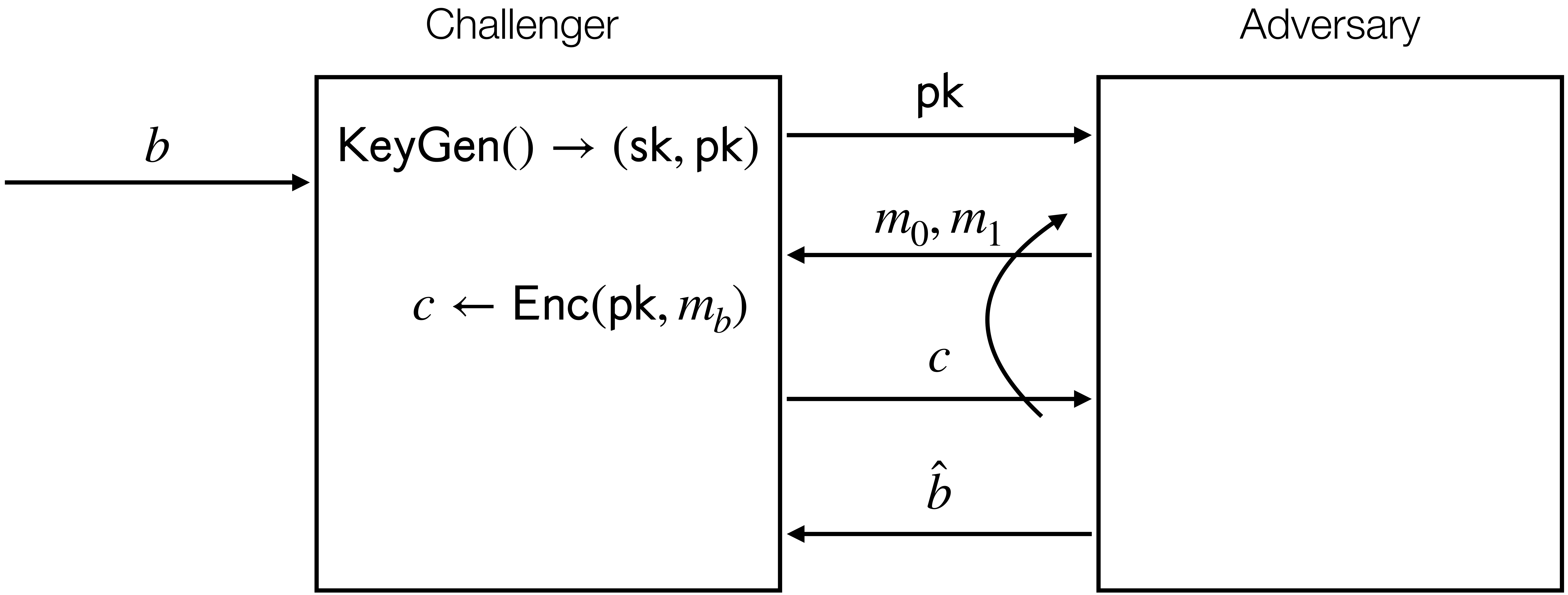
$\mathsf{KeyGen}() \to (\mathsf{sk}, \mathsf{pk})$
$\mathsf{Enc}(\mathsf{pk}, m) \to c$
$\mathsf{Dec}(\mathsf{sk}, c) \to m$

Notes:

- Public key does not make it possible to decrypt
- Clients can generate encryptions without being able to decrypt
- More expensive than symmetric-key cryptography (but still fast enough to encrypt web traffic)

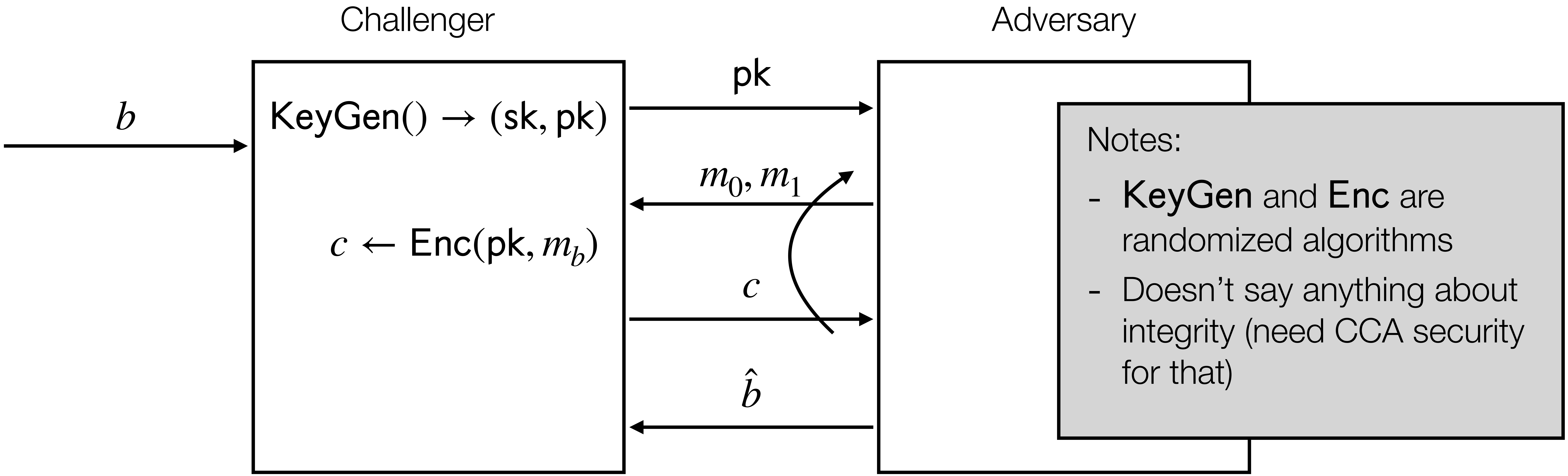
Examples: ElGamal, RSA, …

Chosen plaintext attack security for public-key encryption



An encryption scheme is semantically secure if the adversary's probability of guessing $\hat{b} = b$ is "very close" to 1/2.

Semantic security for public-key encryption



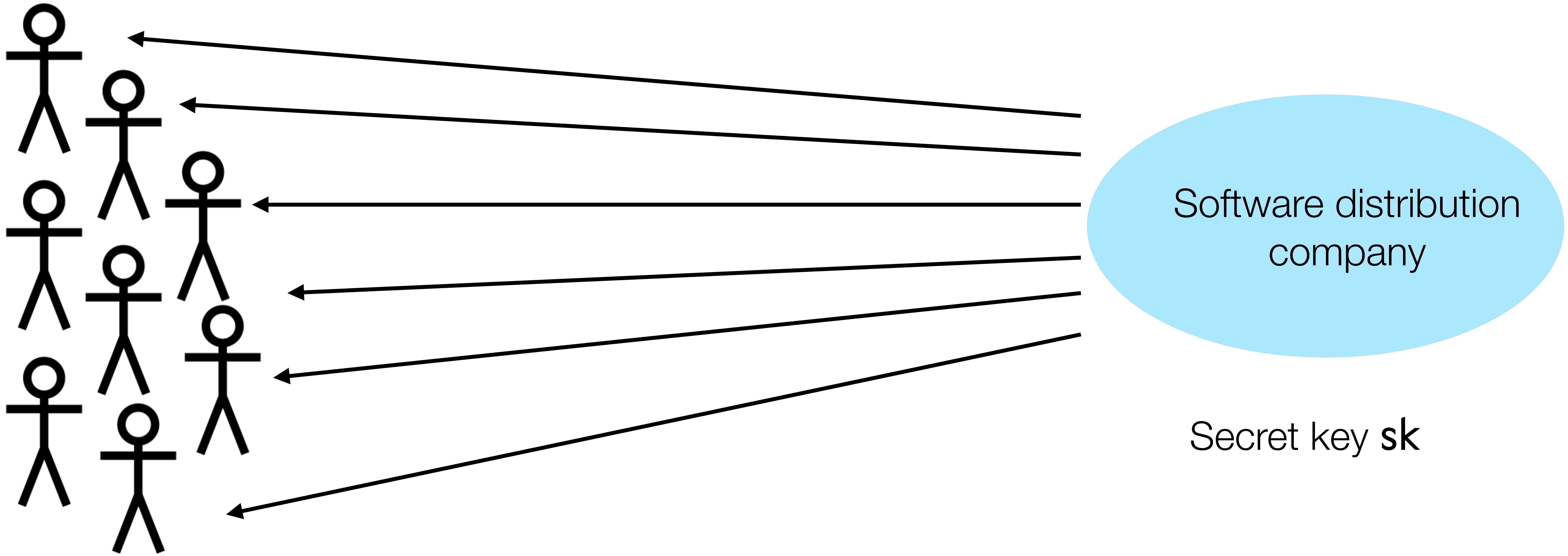
Notes:

- **KeyGen** and **Enc** are randomized algorithms
- Doesn't say anything about integrity (need CCA security for that)

An encryption scheme is semantically secure if the adversary's probability of guessing $\hat{b} = b$ is "very close" to 1/2.

Digital signatures

Public key **pk**



Secret key **sk**

Digital signatures

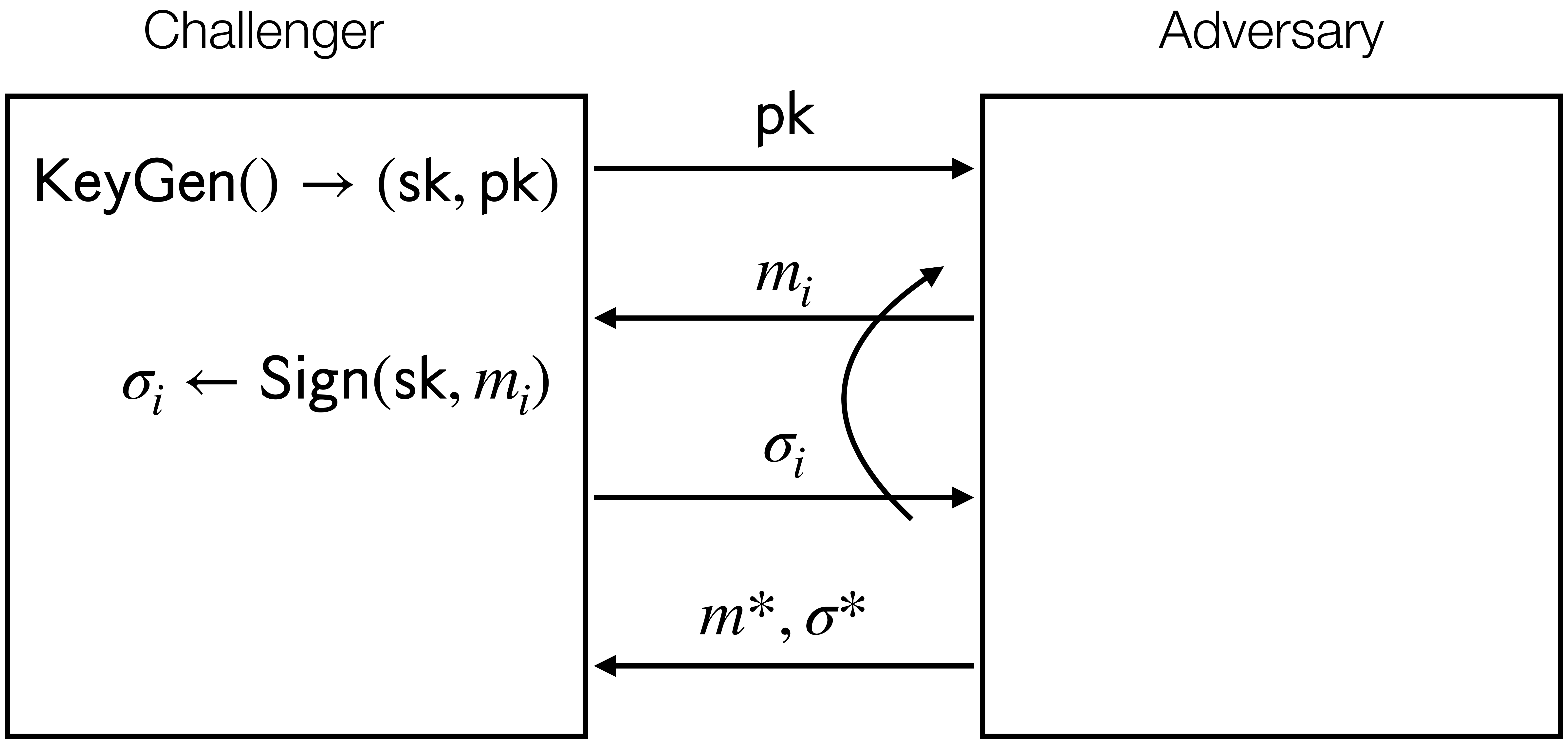
$\mathsf{KeyGen}() \rightarrow (\mathsf{sk}, \mathsf{pk})$
$\mathsf{Sign}(\mathsf{sk}, m) \rightarrow \sigma$
$\mathsf{Verify}(\mathsf{pk}, m, \sigma) \rightarrow \{0,1\}$

Notes:

- Public key only needed for verification
- Public-key analogue of message authentication codes

Examples: ECDSA, Schnorr, …

Digital signature security

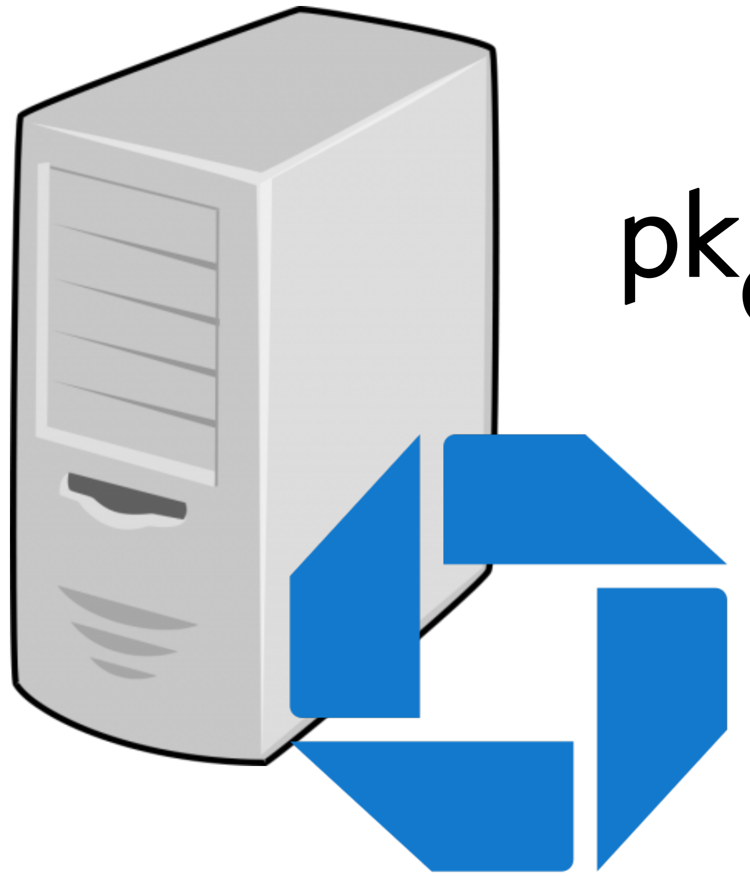


Adversary wins if:

- $(m^*, \sigma^*) \notin \{(m_1, \sigma_1), (m_2, \sigma_2), \ldots\}$
- $\mathsf{Verify}(\mathsf{pk}, m^*, \sigma^*) = 1$ ("accept")

Browsing the web

How to securely establish a web connection?

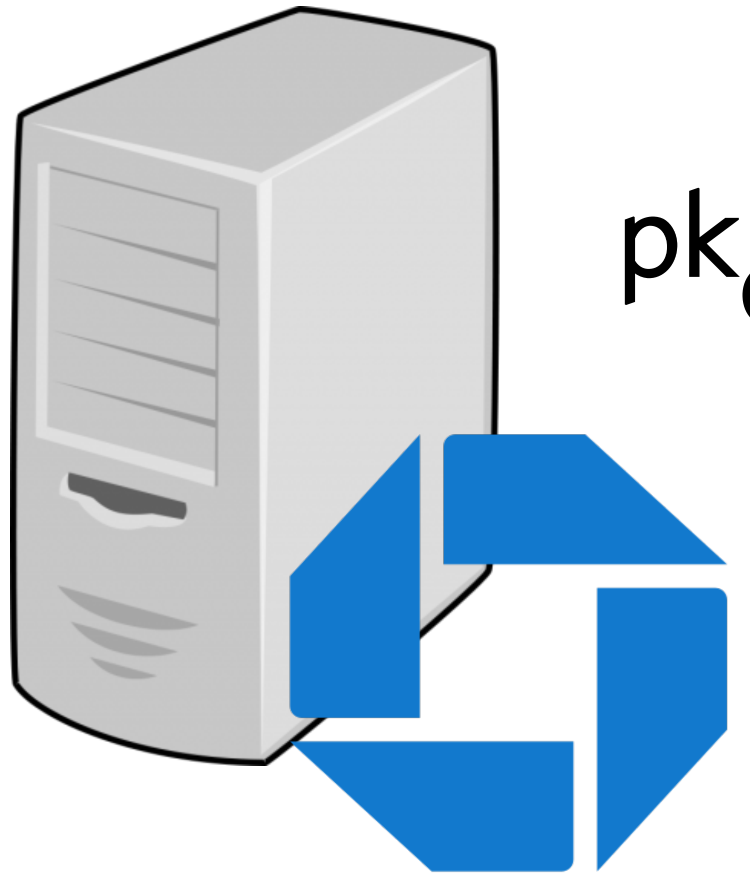


pk_{Chase}

How to securely establish a web connection?

Establish shared key k

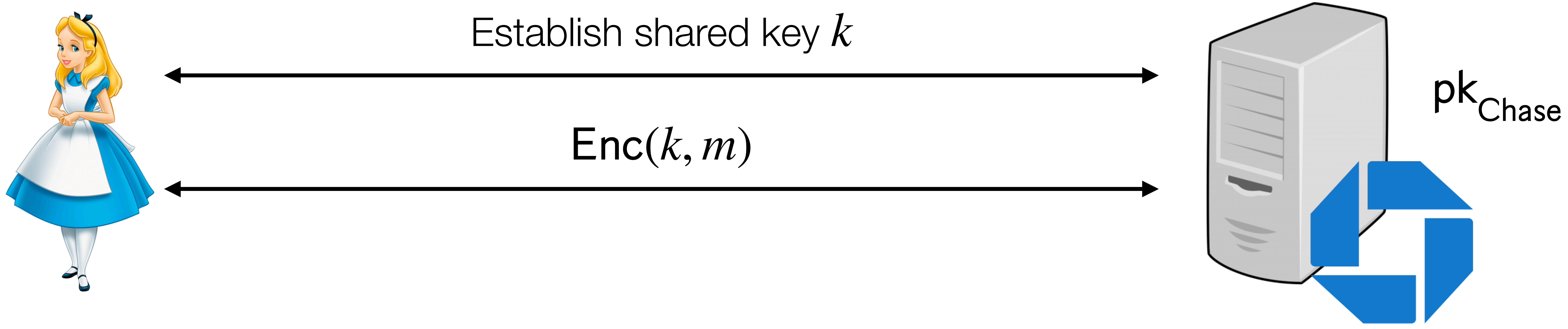
$\text{Enc}(k, m)$



pk_{Chase}

*Enc is an authenticated encryption scheme

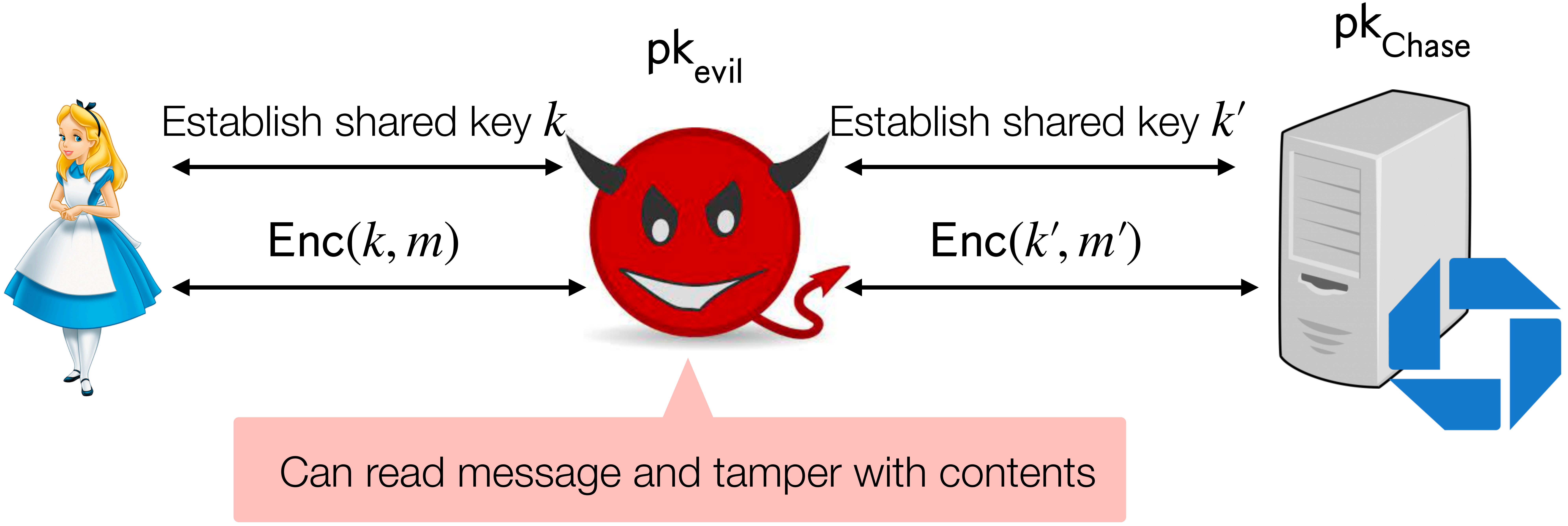
How to securely establish a web connection?



How does Alice know she is talking to the “right” $\mathsf{pk}_{\mathsf{Chase}}$?

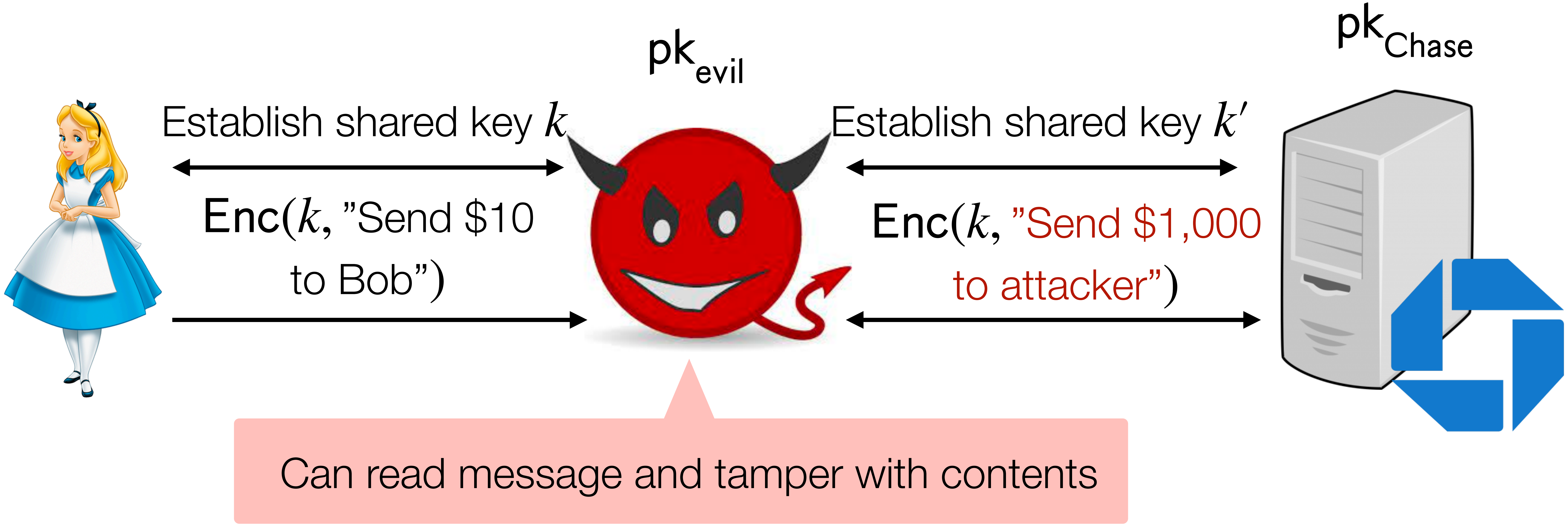
*Enc is an authenticated encryption scheme

Man-in-the-middle attack



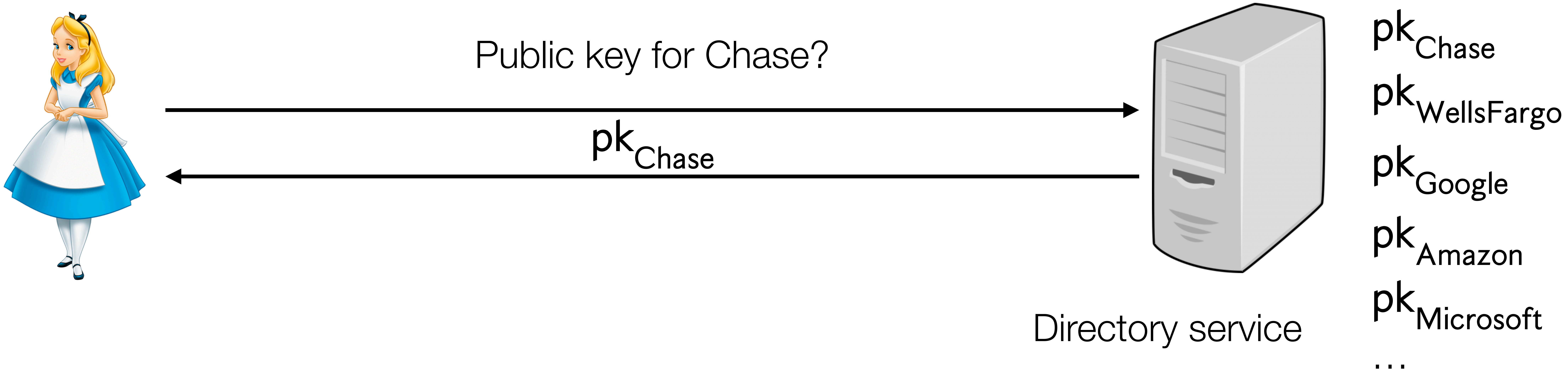
Alice cannot tell that she's talking to the attacker instead of Chase

Man-in-the-middle attack



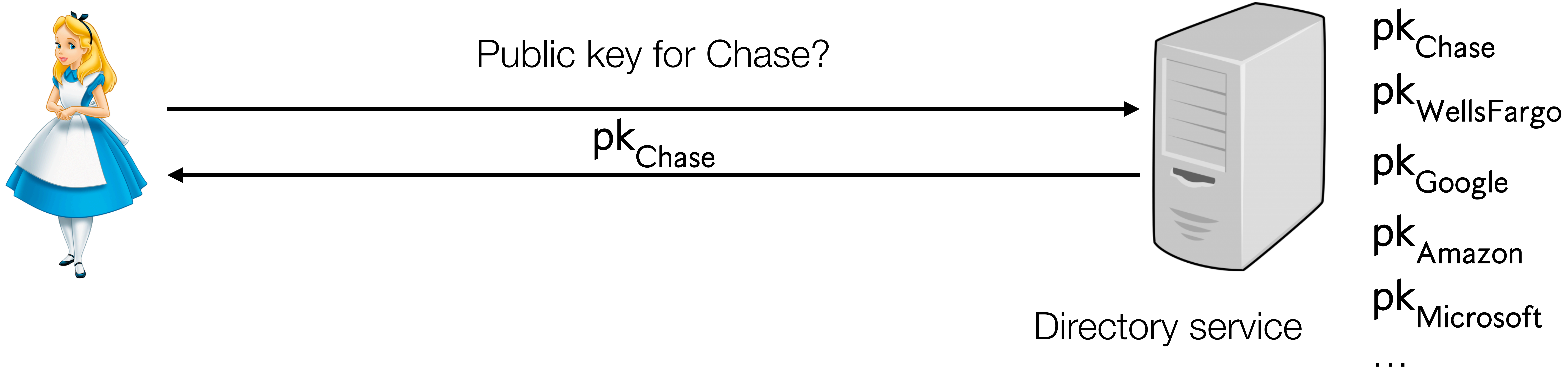
Alice cannot tell that she's talking to the attacker instead of Chase

One approach: trusted directory service



Drawbacks?

One approach: trusted directory service

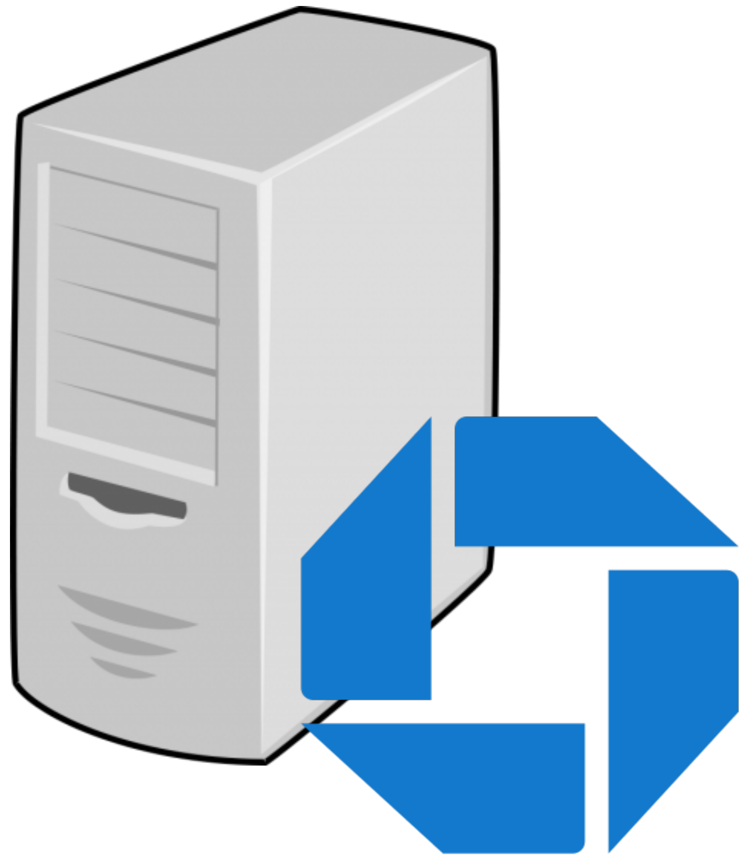


Drawbacks:

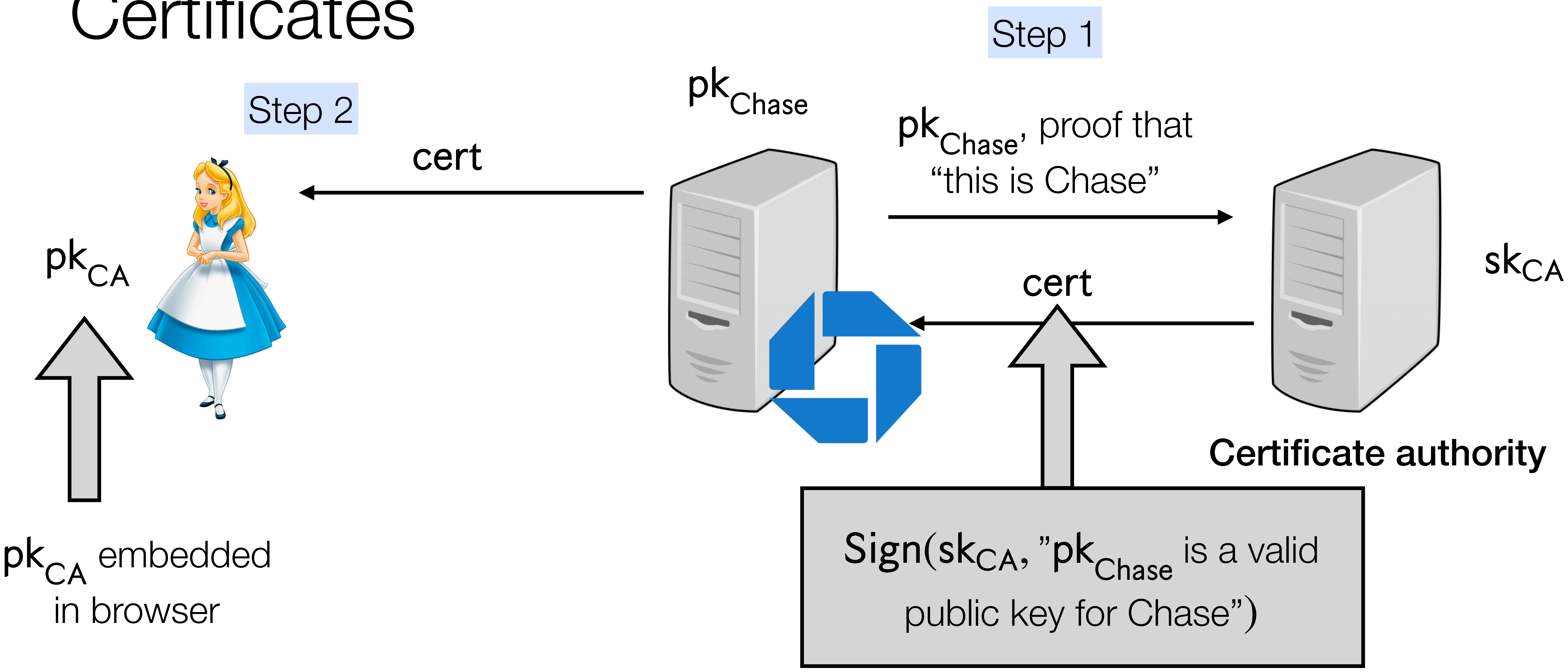
- A malicious directory service can provide the wrong public key
- Alice must be able to access the directory service at all times:
  - Must be online and can become a scalability bottleneck

The approach we use: certificates

pk_{Chase}

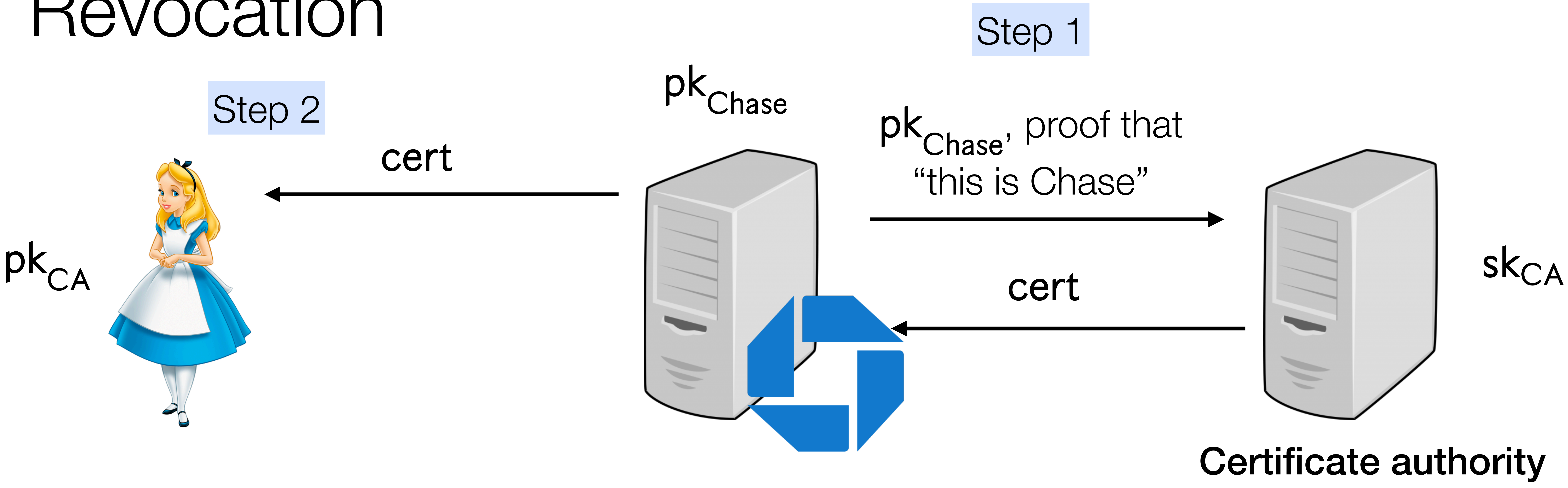


Certificates



Unlike directory service, Alice does not have to contact the CA

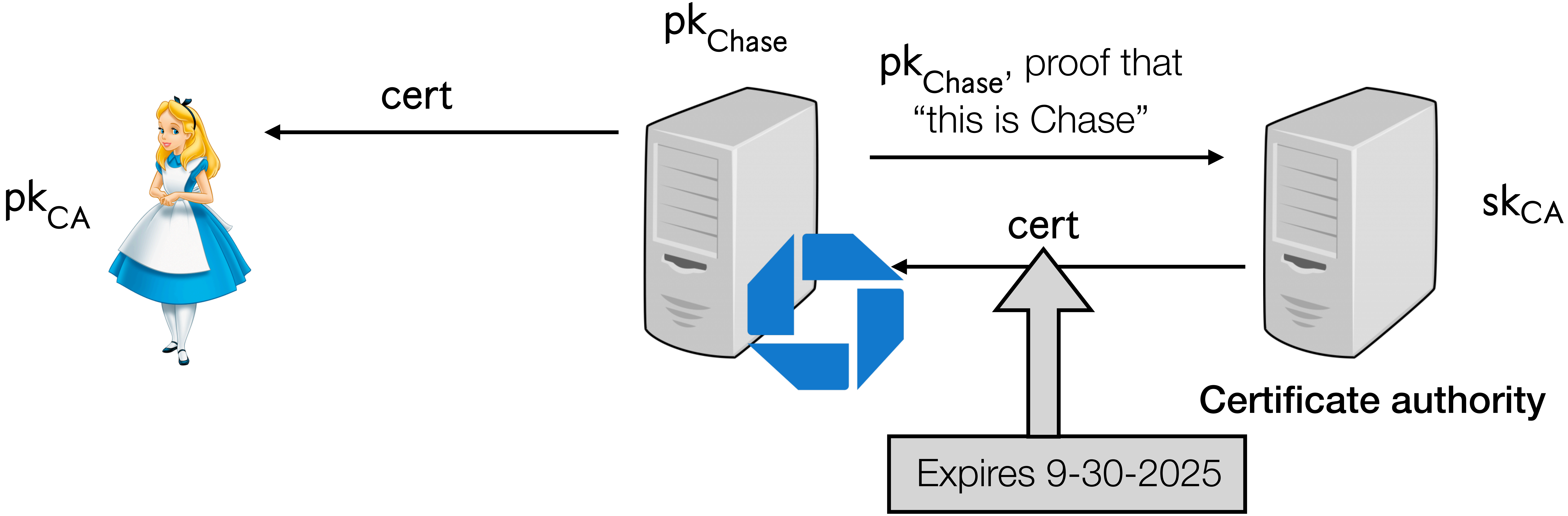
Revocation



The CA might make a mistake when issuing a certificate, or a website may need to update its public key

How to support updates with certificates?

Revocation approach #1: validity periods

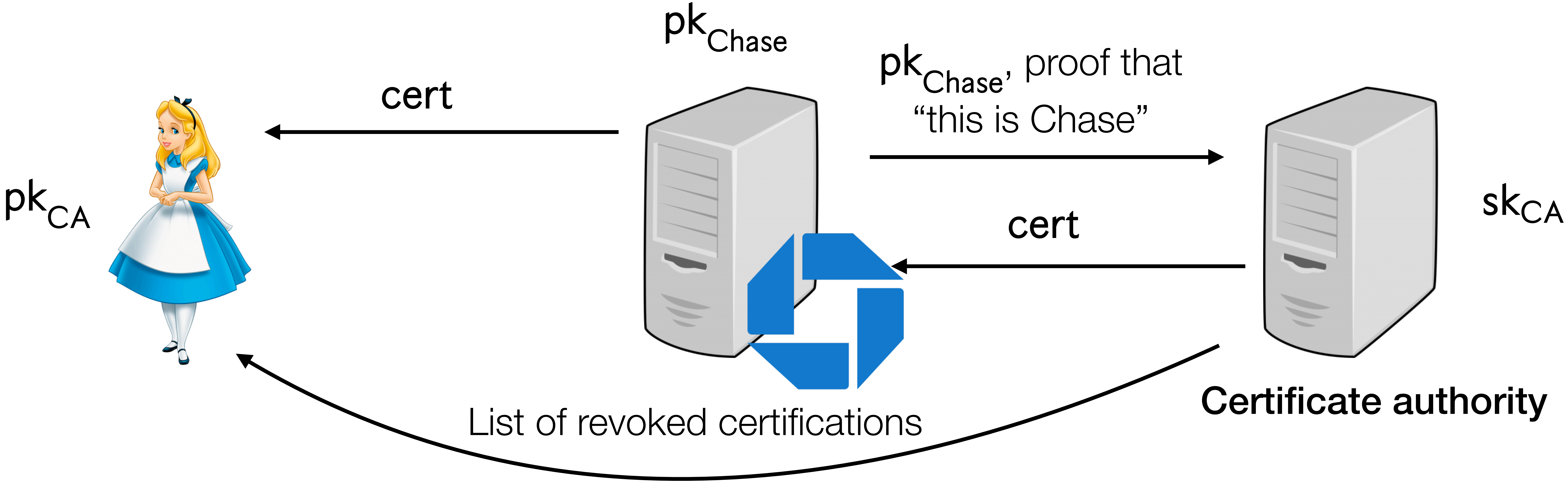


The CA might make a mistake when issuing a certificate, or a website may need to update its public key

How to support updates with certificates?

1. Validity periods: tradeoff between efficiency and speed of revocation

Revocation approach #2: revocation lists

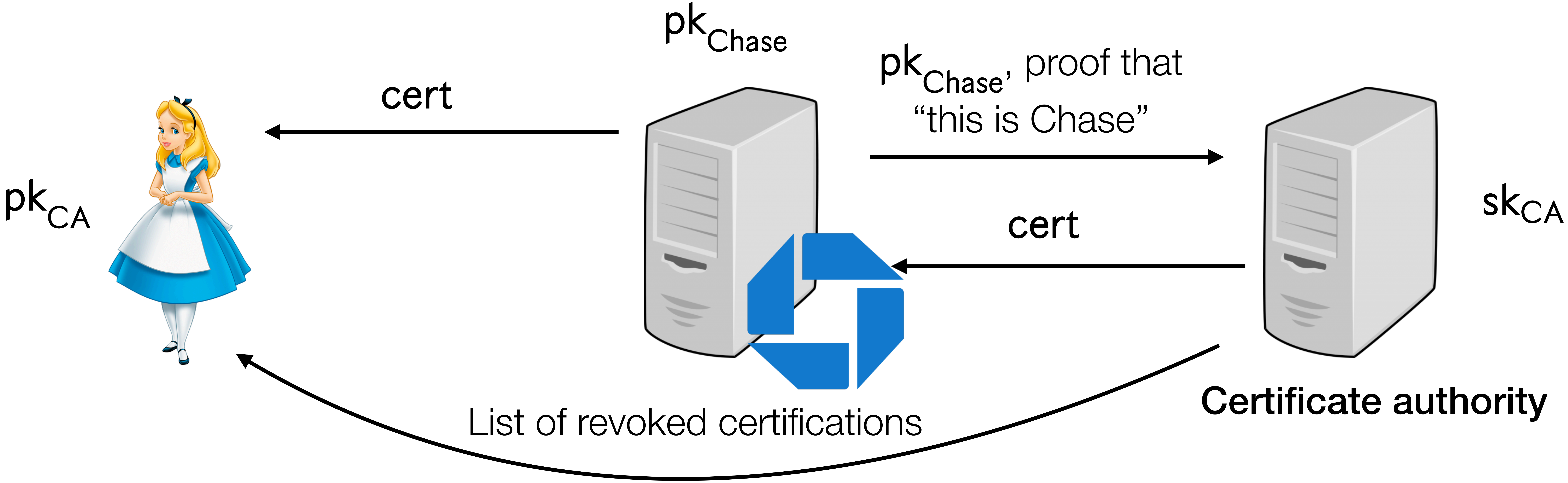


The CA might make a mistake when issuing a certificate, or a website may need to update its public key

How to support updates with certificates?

1. Validity periods: tradeoff between efficiency and speed of revocation
2. Revocation list: client periodically fetches list of revoked certificates

Revocation approach #2: revocation lists



Next class: how to protect against incorrectly issued certificates

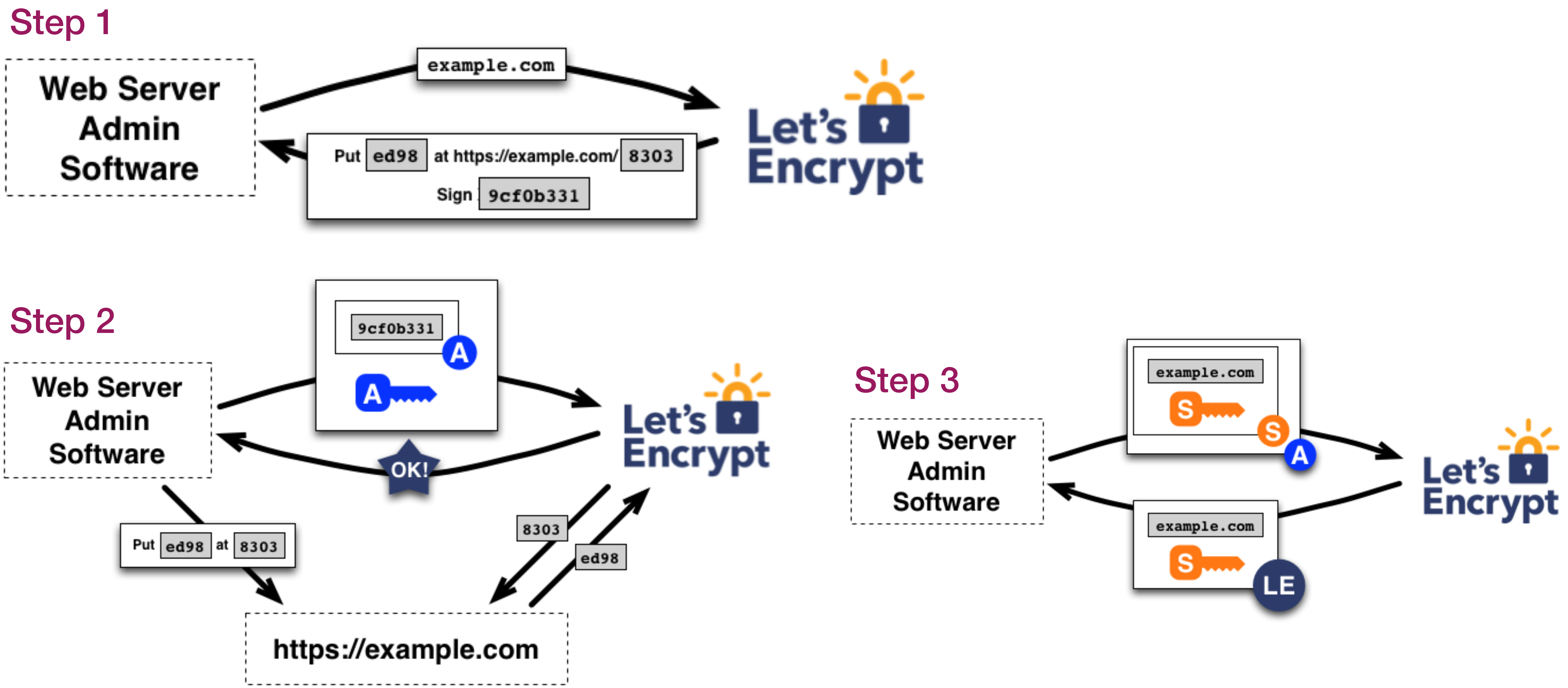
Getting a certificate with Let's Encrypt

Free certificate authority that makes it easy to obtain certificates

How to prove that you "own" a domain?

https://letsencrypt.org/how-it-works/

Getting a certificate with Let's Encrypt



https://letsencrypt.org/how-it-works/

Authenticated data structures

Goals for authenticated data structures

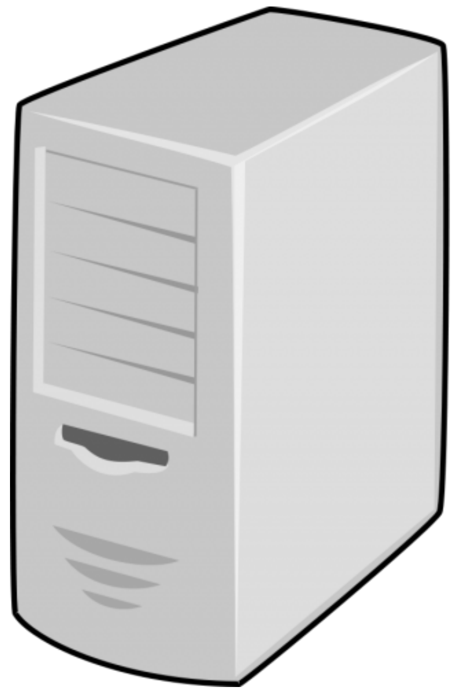
Limited storage

cm

Stores a short *commitment* to a list

Check if the commitment includes/excludes element v_i

Lots of storage



$v_1, v_2, v_3, \ldots$

Stores the entire list

Tool: collision-resistant hash function

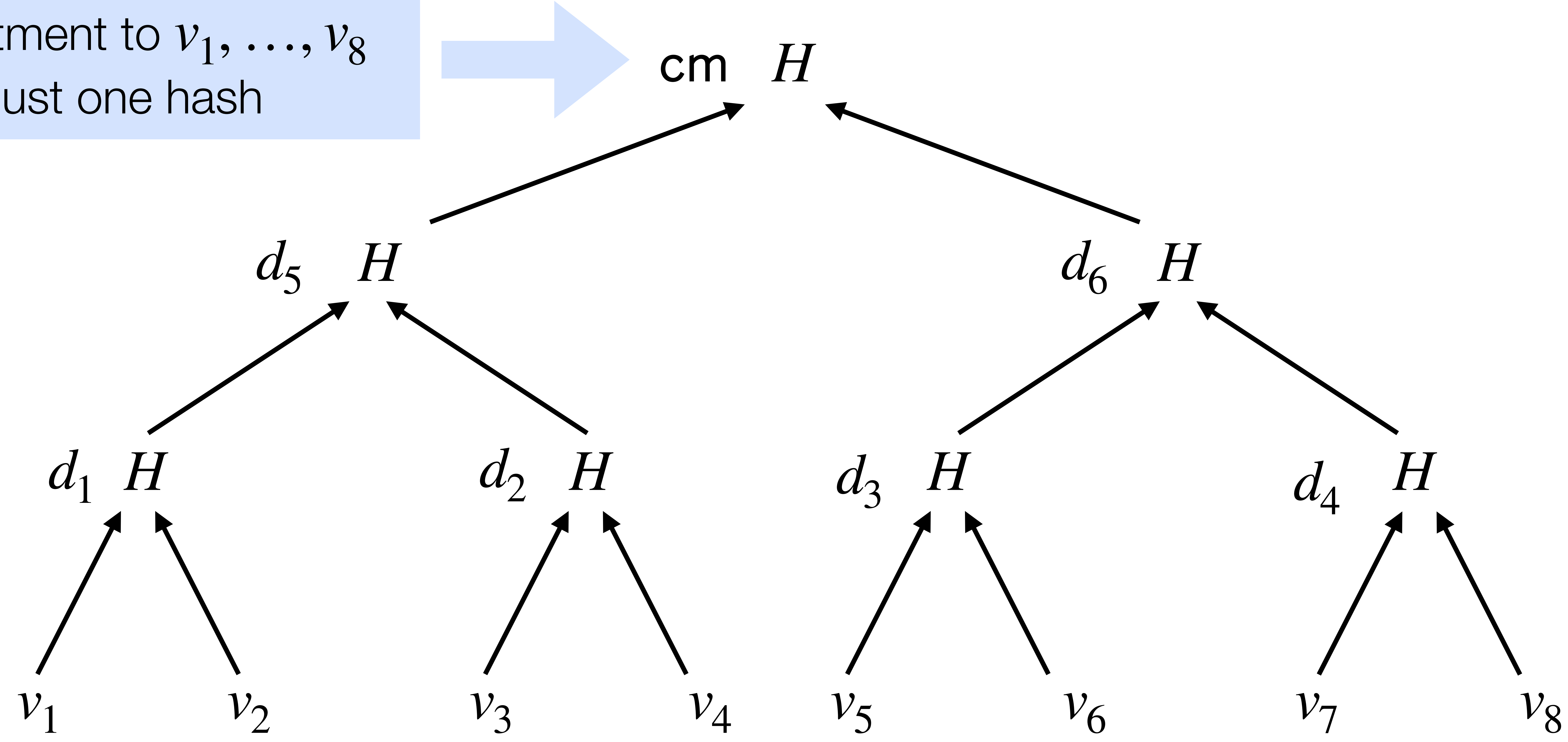
Input space $\{0,1\}^*$
Output space $\{0,1\}^{256}$

Hash function $H : \{0,1\}^* \to \{0,1\}^{256}$

A hash function H is collision-resistant if no "efficient" adversary can find a collision, i.e.
$H(a) = H(b)$ for $a \neq b$

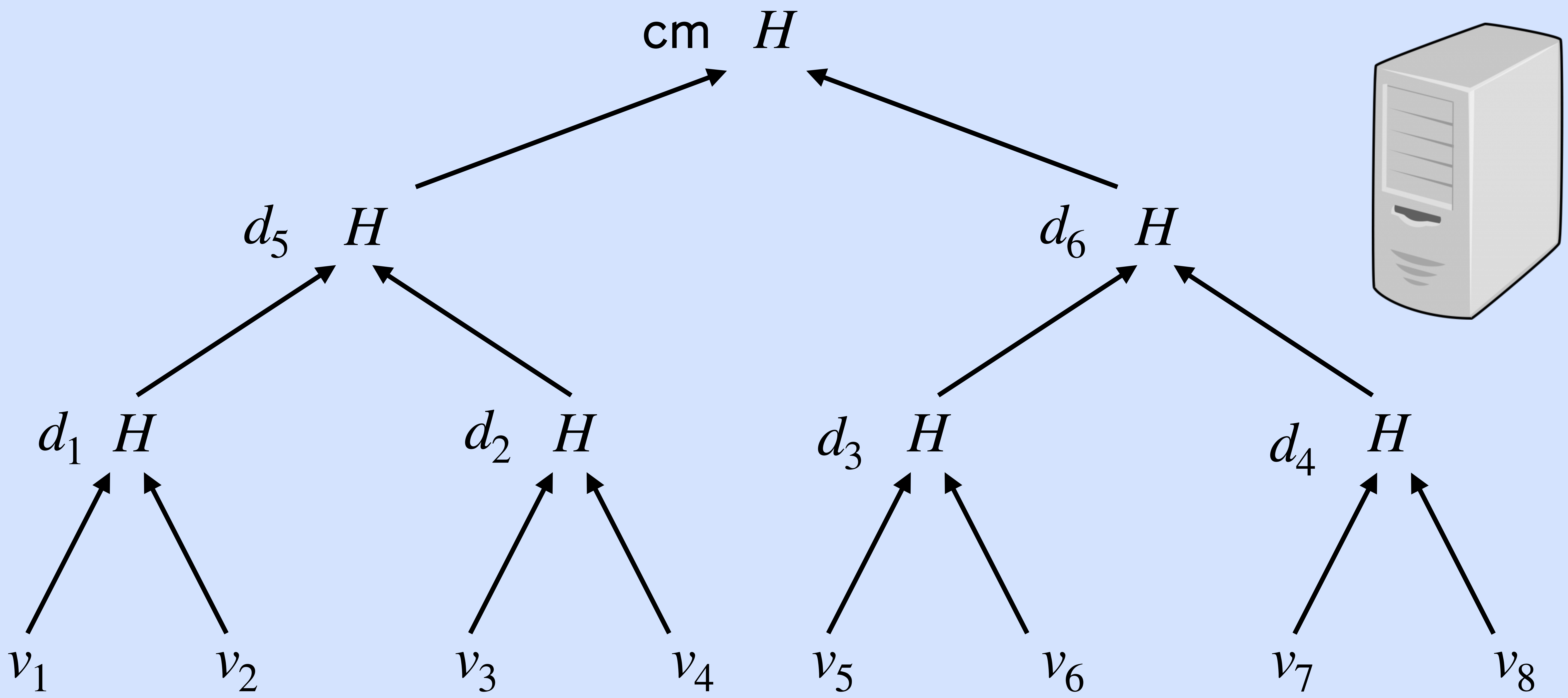
Merkle tree

Commitment to $v_1, \ldots, v_8$ is just one hash



Merkle tree

cm



Merkle tree

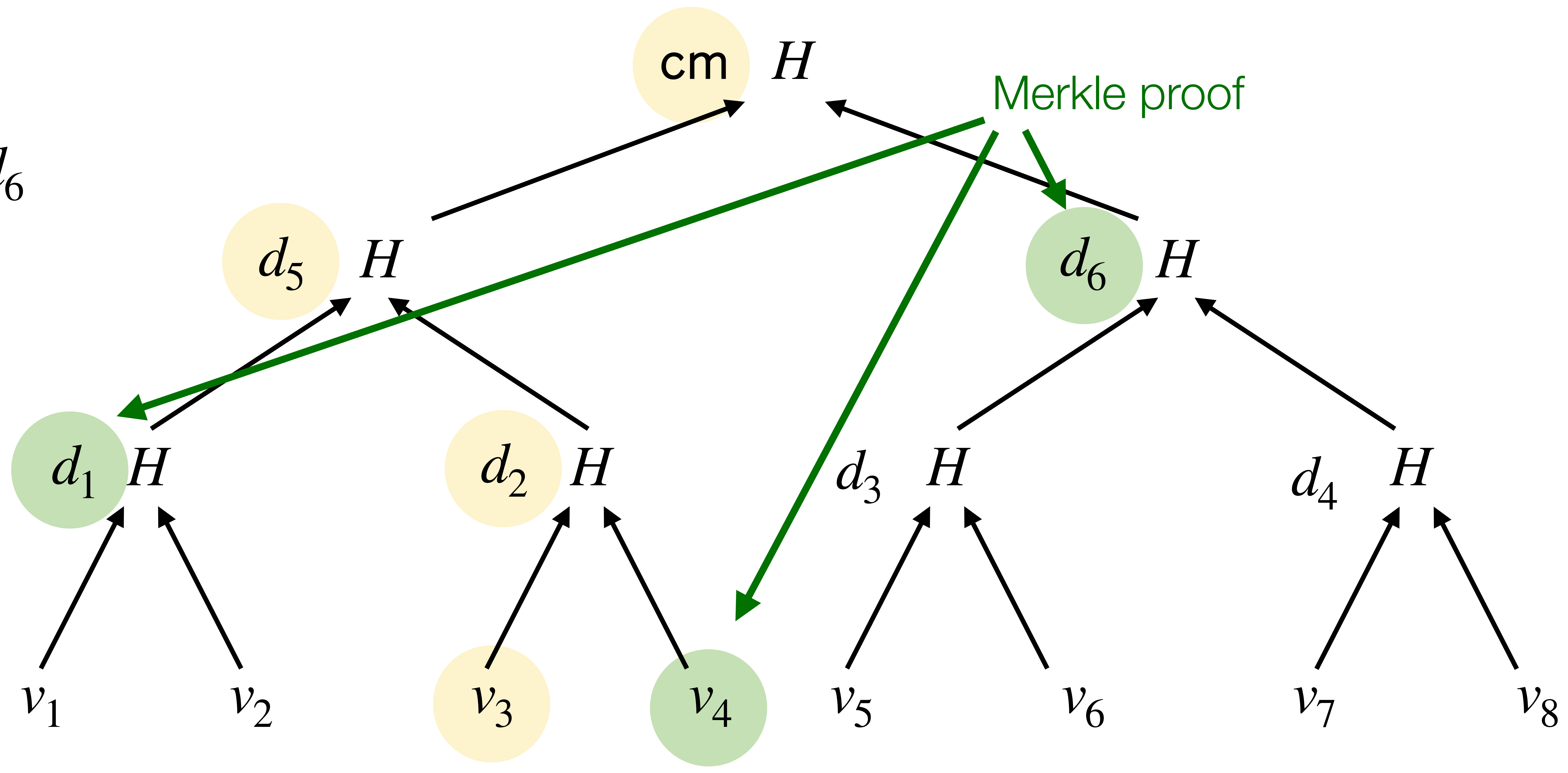
Is v_3 included?

Fetch proof $\pi = v_4, d_1, d_6$

Compute:

- $\hat{d}_2 \leftarrow H(v_3, v_4)$
- $\hat{d}_5 \leftarrow H(d_1, \hat{d}_2)$

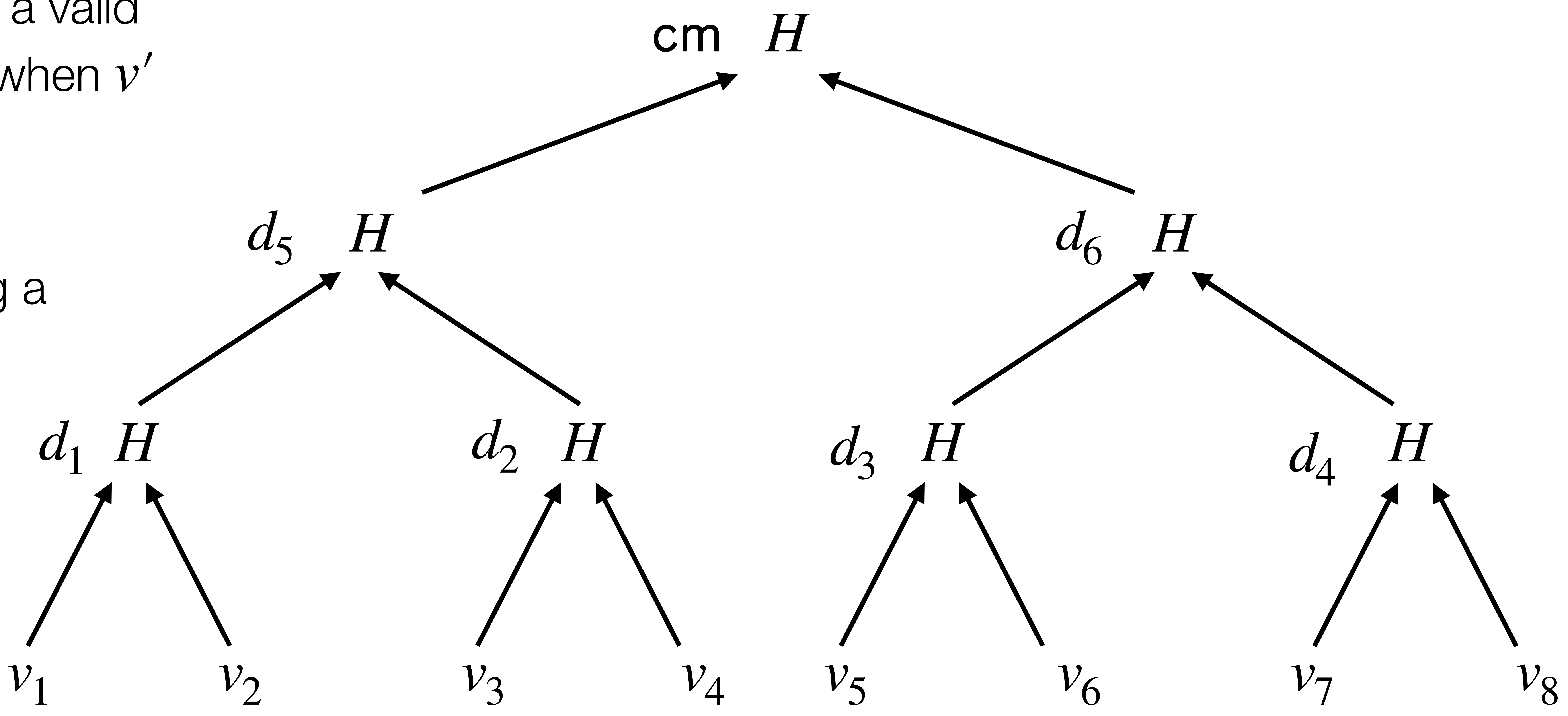
Check: $\textbf{cm} = H(\hat{d}_5, d_6)$



Merkle tree

Is it possible to generate a valid proof of inclusion for v', when v' is not included?

No, would require finding a hash function collision

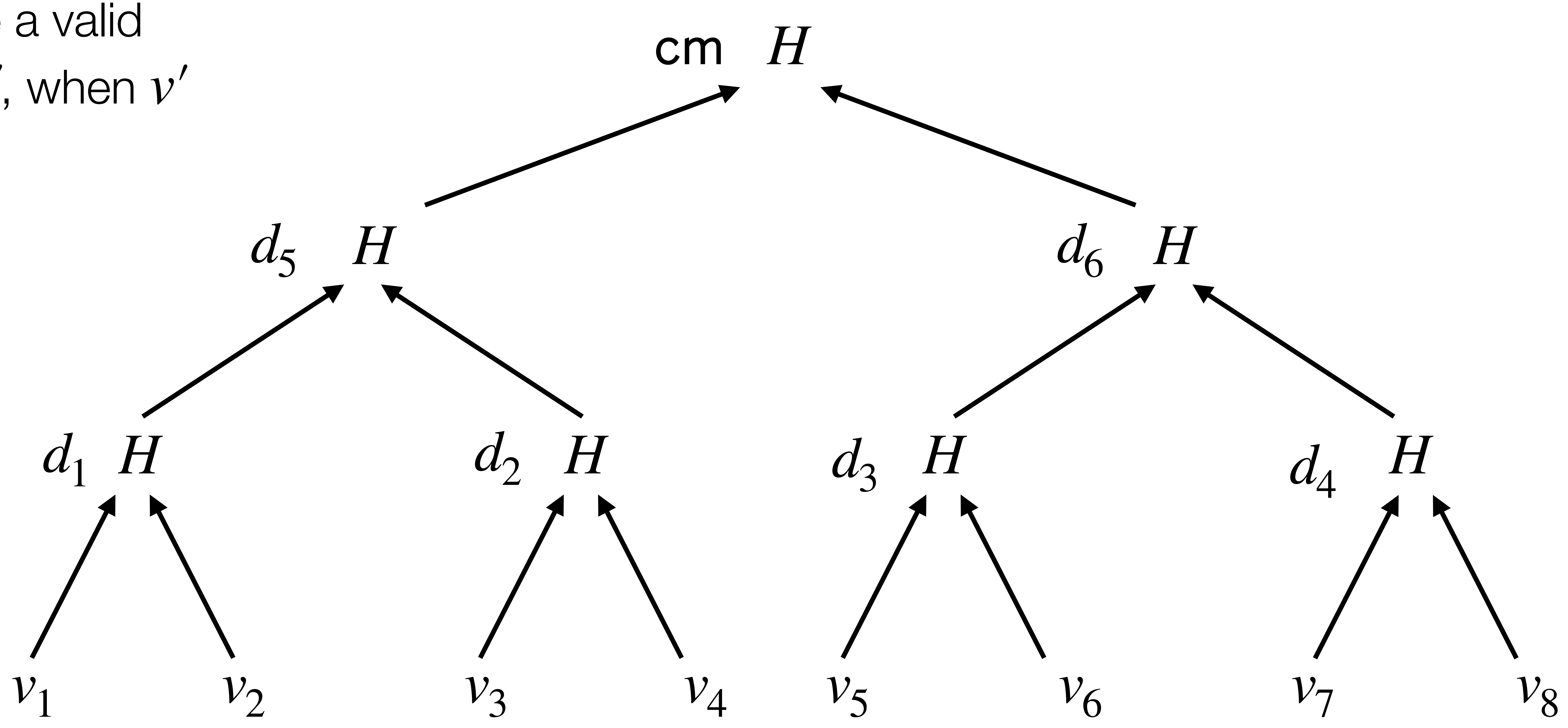


Authenticated data structures

Is it possible to generate a valid proof of **exclusion** for v', when v' is not included?

Idea:

1. Sort the values
2. Generate proofs of the values to the right and left of where v' would go



Authenticated data structures

Sort values when creating Merkle tree

To generate proof of non-inclusion for v' relative to cm:

- Find adjacent v_i, v_{i+1} such that $v_i < v' < v_{i+1}$
- Generate Merkle proofs π_1, π_2 for v_i, v_{i+1}

To check proof of non-inclusion for v':

- Check proofs π_1, π_2 relative to cm
- Check that $v_i < v' < v_{i+1}$ and v_i, v_{i+1} are adjacent

Requires list to be correctly sorted

Other applications of authenticated data structures

- Checking if files are correctly stored on a disk
- Blockchains
- Certificate transparency (this class)
- Key transparency (this class)

Next time: certificate transparency

- Two readings on certificate transparency
- Reading questions due at 3PM on Tuesday via Gradescope
- Fill out the form on Ed by the end of the week to sign up for paper presentations

References

Stanford CS 255, CS 251
MIT 6.1600
Berkeley CS 161
Boneh-Shoup cryptography book