Cryptography II

Instructor: Paruj Ratanaworabhan

Credits:
J. Kubiatowicz, UC Berkeley
R. Lee, Princeton U.
bitcoinbook.cs.princeton.edu

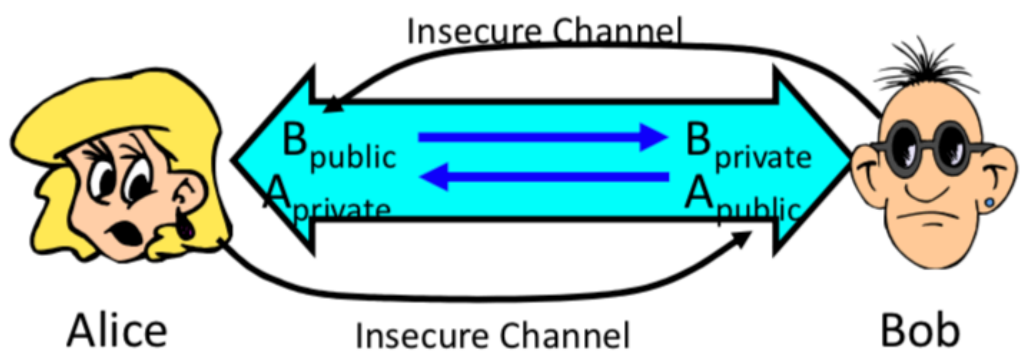
Public key encryption

- Another way of distributing keys without an authentication server is using a cryptographic system that each user possesses two keys, public and private keys
  - Asymmetric-key cryptography
- Example of public-key crypto system
  - RSA: from Rivest, Shamir, and Adleman, inventors of the scheme
  - ECC: Elliptic Curve Cryptography

Public key encryption

- Require two keys, K_{public} and $K_{private}$
  - Both keys are mathematically related, but it is extremely hard to derive K_{public} from $K_{private}$ and vice versa
- Forward encryption:
  - Encrypt: $(\text{plaint text})^{Kpublic} = \text{cipher text}_1$
  - Decrypt: $(\text{cipher text}_1)^{Kprivate} = \text{plain text}$
- Reverse encryption:
  - Encrypt: $(\text{plain text})^{Kprivate} = \text{cipher text}_2$
  - Decrypt: $(\text{cipher text}_2)^{Kpublic} = \text{plain text}$
- $\text{cipher text}_1 \neq \text{cipher text}_2$

Public key encryption

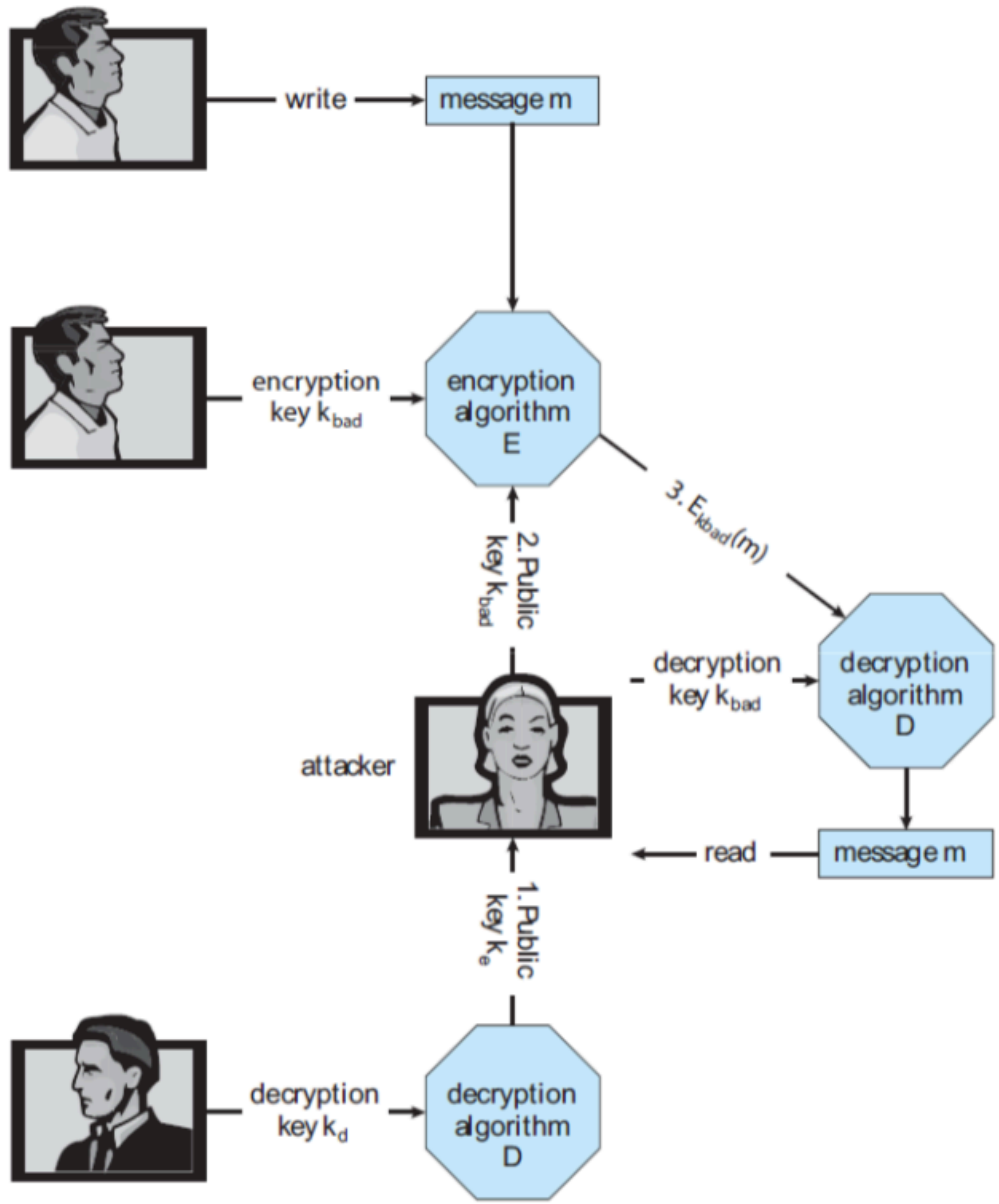


- K_{public} is to be made public, but $K_{private}$ **<u>must absolutely be kept secret</u>**
- Provide data privacy as the only person that can unlock the secret must have the private key in possession
- Can be used for authentication, e.g., Alice wants to ascertain to Bob that she is, indeed, Alice:

Alice→Bob: [(I'm Alice)^{Aprivate} Rest of message]^{Bpublic}

The main problem in public key encryption: How does Alice know that B_{public} actually belongs to Bob, not someone masquerading as Bob

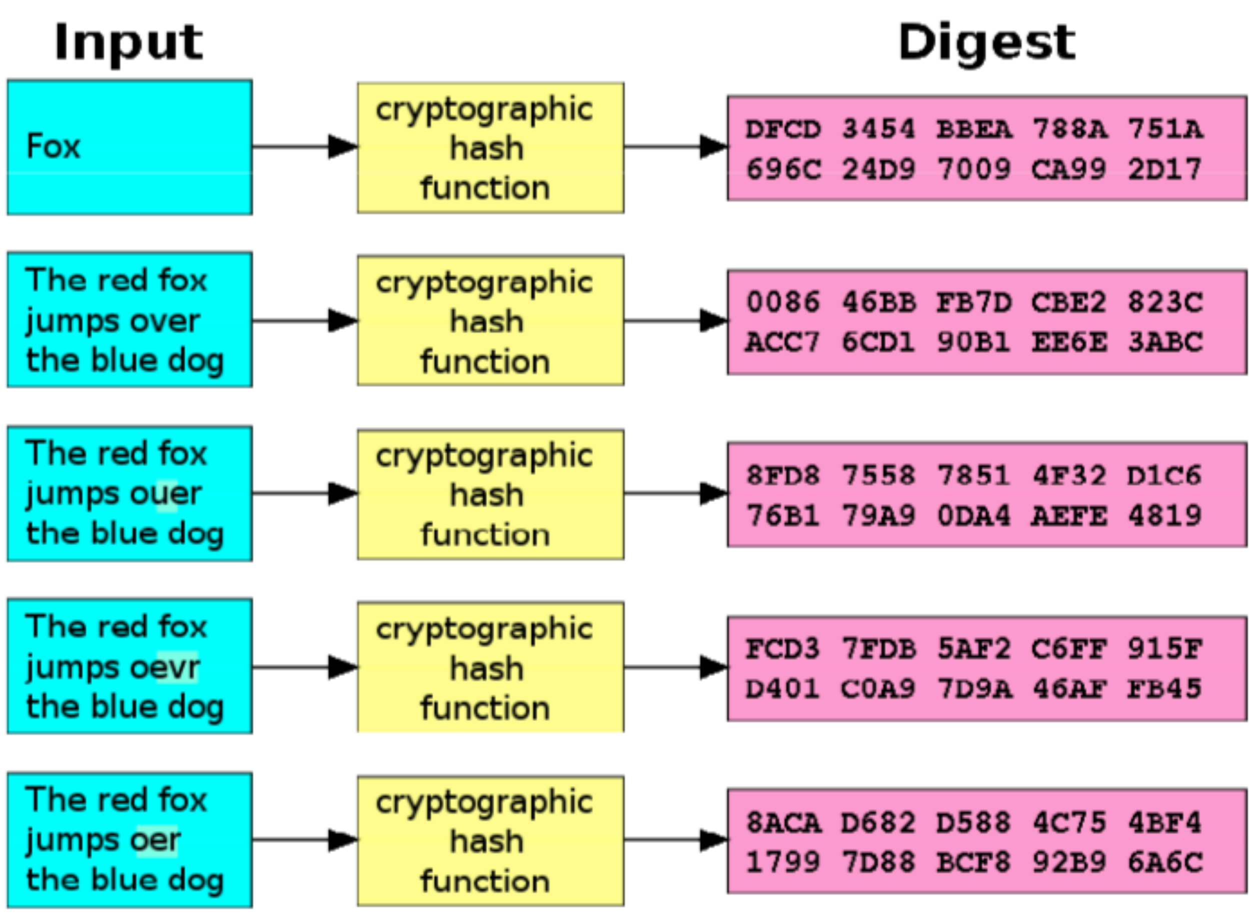
Man-in-the-middle (MIM) attack in asymmetric key crypto system



We will come to save this MIM later, but we need to introduce a few cryptographic constructs firs, starting with hash functions

Secure hash function

- Output of a secure hash function represents a summary of the input message
  - E.g., h_1=H(M_1) is a hash of M_1 such that h_1 is a string of fixed size (e.g., 256 bits), even though M_1 size may vary
  - h_1 is called the “digest” of M_1



Security Properties

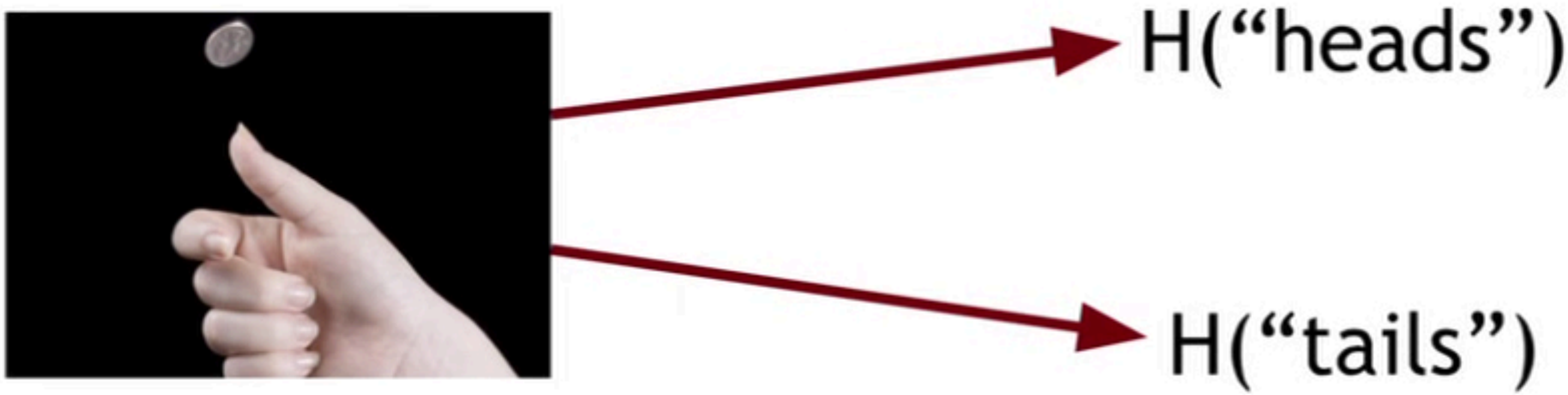
- Collision-free
- Hiding
- Puzzle-friendly

Collision-Free

- Want: no one should be able to find two different x and y values such that H(x) = H(y)
- But, collisions do exist according to the Pigeonhole principle
- For some hash functions, collisions can easily be found, e.g., modulo function
- But, for secure cryptographic hash functions, collisions must be extremely hard, if not impossible, to find

Hiding

- Want: given H(x), it is impossible to reverse the process to discover the input x
- However, if x is derived from a small domain, this hiding property may not be easily obtained
- So, we concatenate this x with r, where r is some random bit strings from a very large and uniformly distributed domain (e.g., r is a random string of 256 bits)
- Then, we calculate the hash H(r l x)



Hiding is difficult if inputs are derived from a small domain, need the r value to enforce hiding

Puzzle-Friendly

- Want: for every output, y, if k is derived from a uniformly distributed and large domain, it is impossible to find x such that H(k l x) = y

Bitcoin uses this property in its Proof-of-Work scheme (more on this in later lectures)

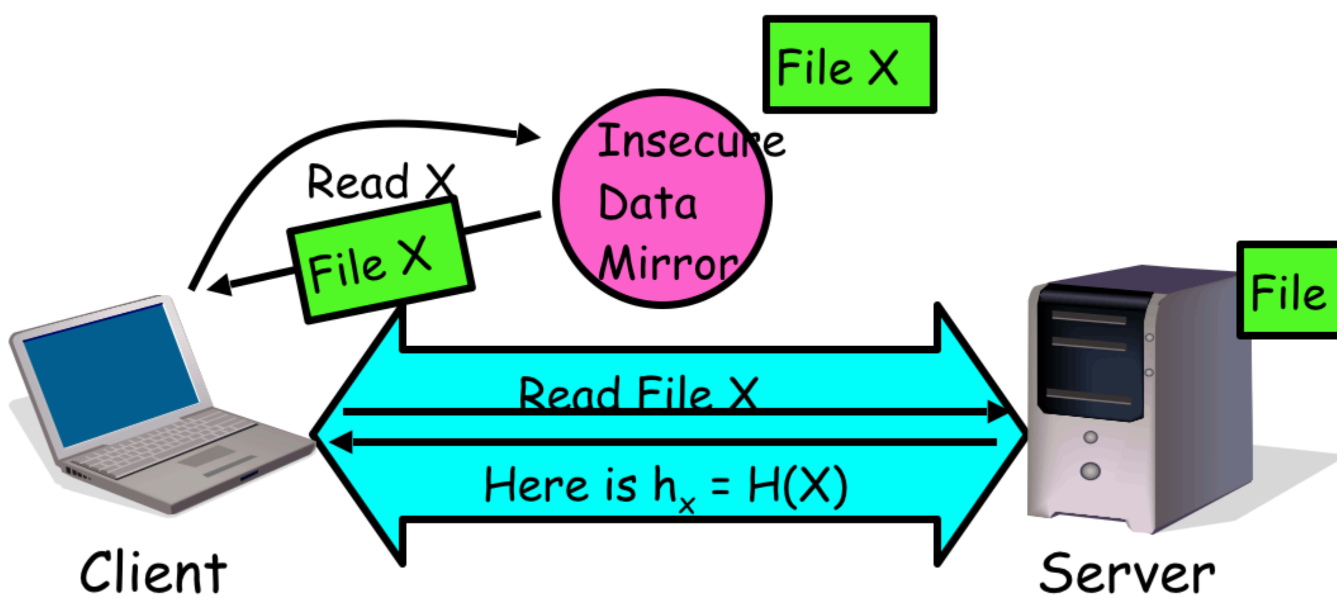
Hash function ยอดนิยม

- MD5
  - 128-bit output
  - Chinese researchers breaks the collision resistance property back in 2004
- SHA1
  - 160-bit output
  - Security researchers has not recommended SHA1 for secure hash function since 2005
  - In 2017, a team of Google researchers breaks the collision resistance property
    - security.googleblog.com/2017/02/announcing-first-sha1-collision.html
- SHA2 (SHA-224 SHA-256 SHA-384 SHA-512)
  - Has digest sizes of 224, 256, 384, and 512 bits, respectively
  - No security problems encountered so far

Usage of Hash Functions

Secure download

- Downloading securely from untrusted mirror sites
  - Mirror sites are to reduce the burden on the main secured server
- Before downloading from a mirror site, contact the main secured server to obtain the hash of the file to download
- With collision-free property, it is impossible to find another file with a different content that can generate the same hash output



Commitment

- Analogous to sealing a secret inside of an envelope to be opened at a later time, e.g., asking prices in a bidding process

this is "com" that is revealed to everyone

Commitment API

commit(*msg*) := (H(*key* | *msg*), H(*key*))
where *key* is a random 256-bit value

verify(*com*, *key*, *msg*) := (H(*key* | *msg*) == *com*)

Security properties:

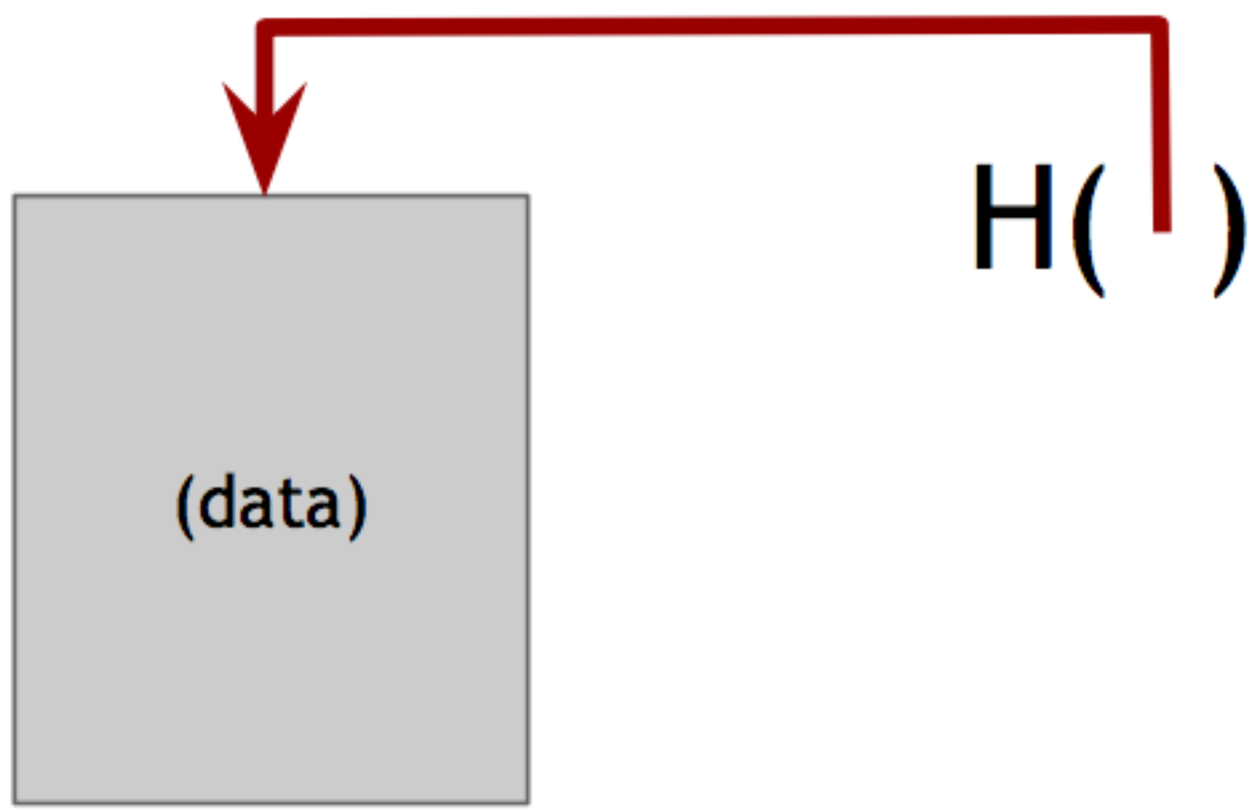
Hiding: Given H(*key* | *msg*), infeasible to find *msg*.

Binding: Infeasible to find *msg* != *msg'* such that
H(*key* | *msg*) == H(*key* | *msg'*)

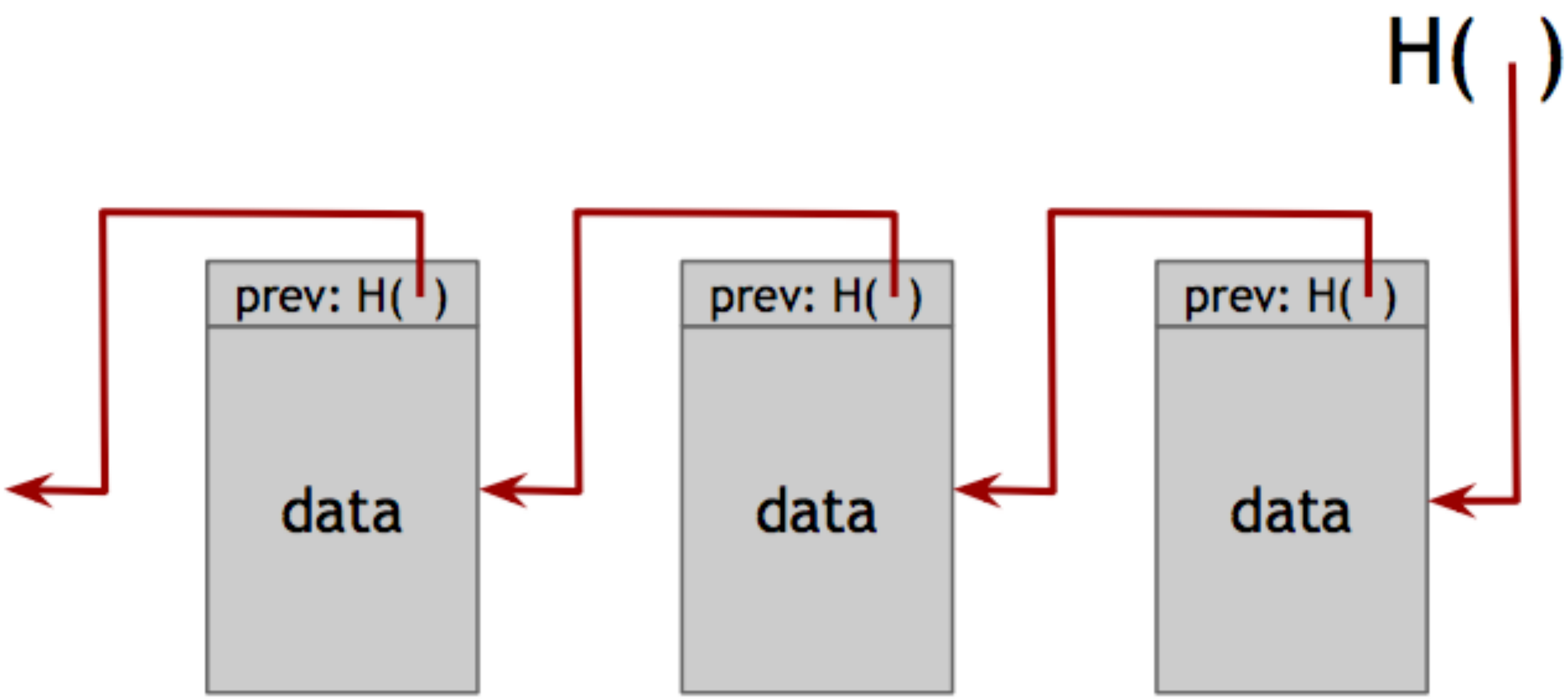
Hash Pointer

- Is a reference to the beginning address that stores the data together with its hash value
- With a hash pointer, we can
  - Access the data
  - Verify the integrity of the data
- Can combine hash pointers with other dynamic data structures as linked-list or tree for some sophisticated integrity checking

Hash pointer symbol



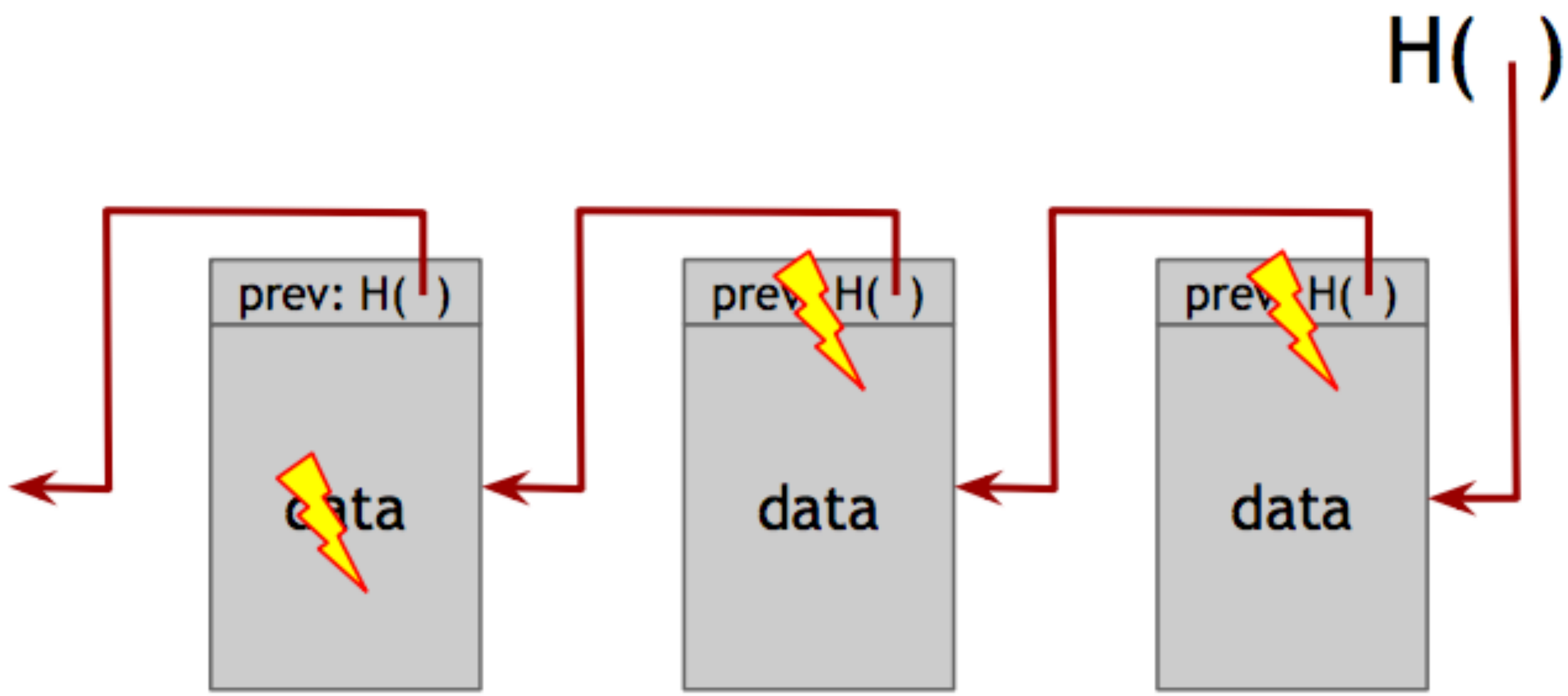
linked list with hash pointers = "block chain"



use case: tamper-evident log

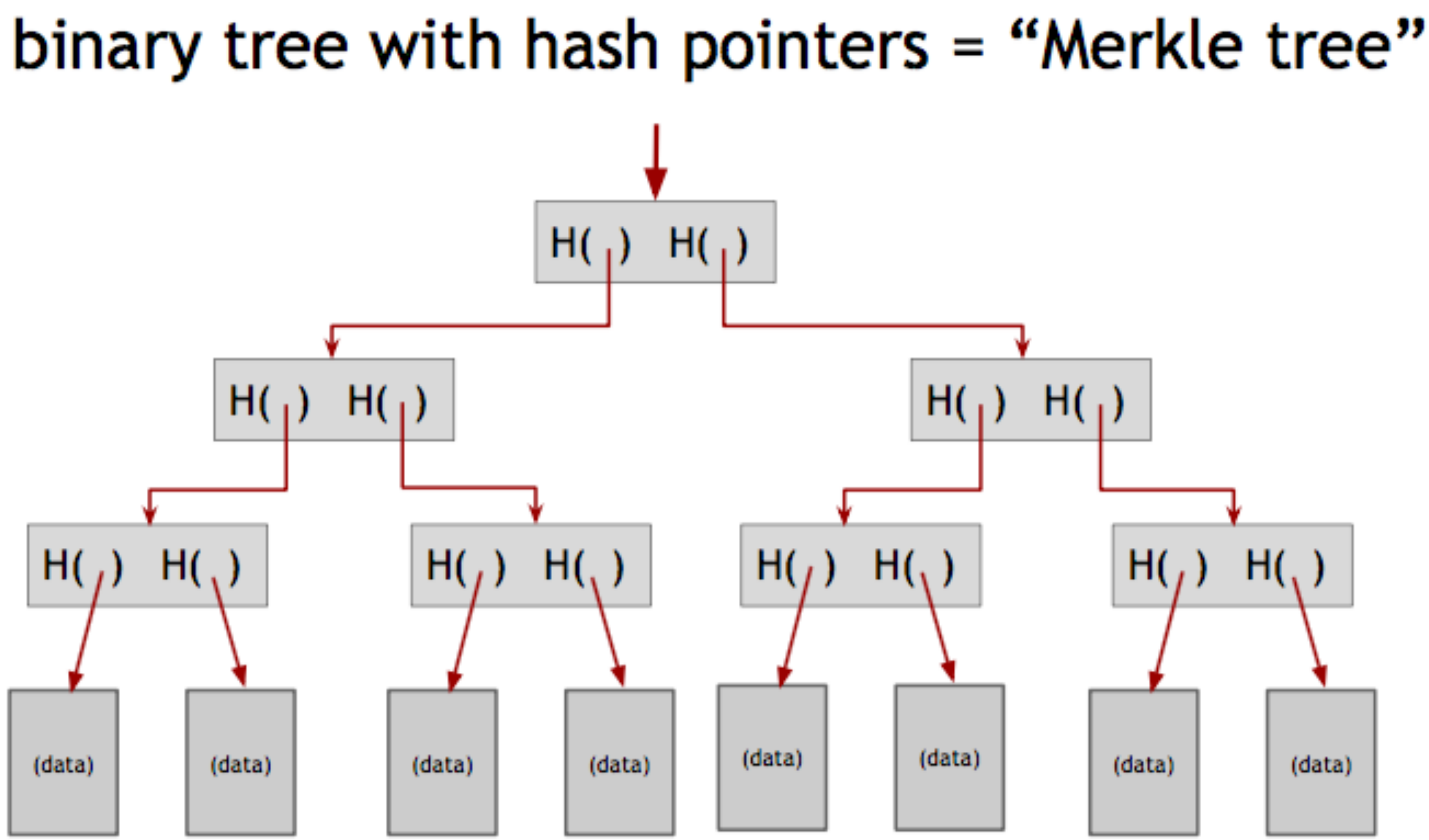
Hash of the head (or root) is kept separate from the rest of the database

detecting tampering



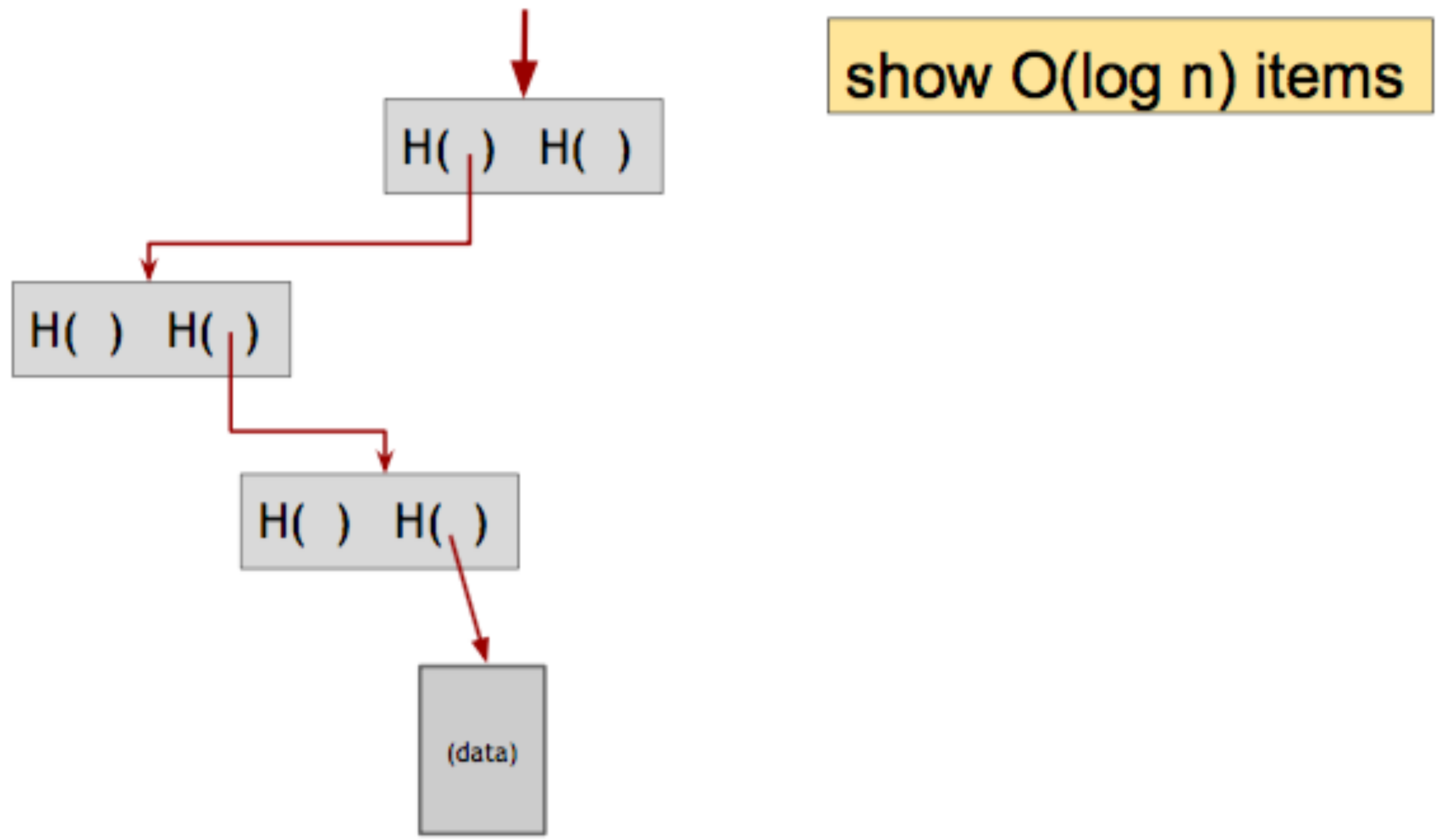
use case: tamper-evident log

An attacker needs to modify all data and hashes in the log, but will hit a roadblock as he/she cannot get to the root hash



Merkle tree style root hash calculation

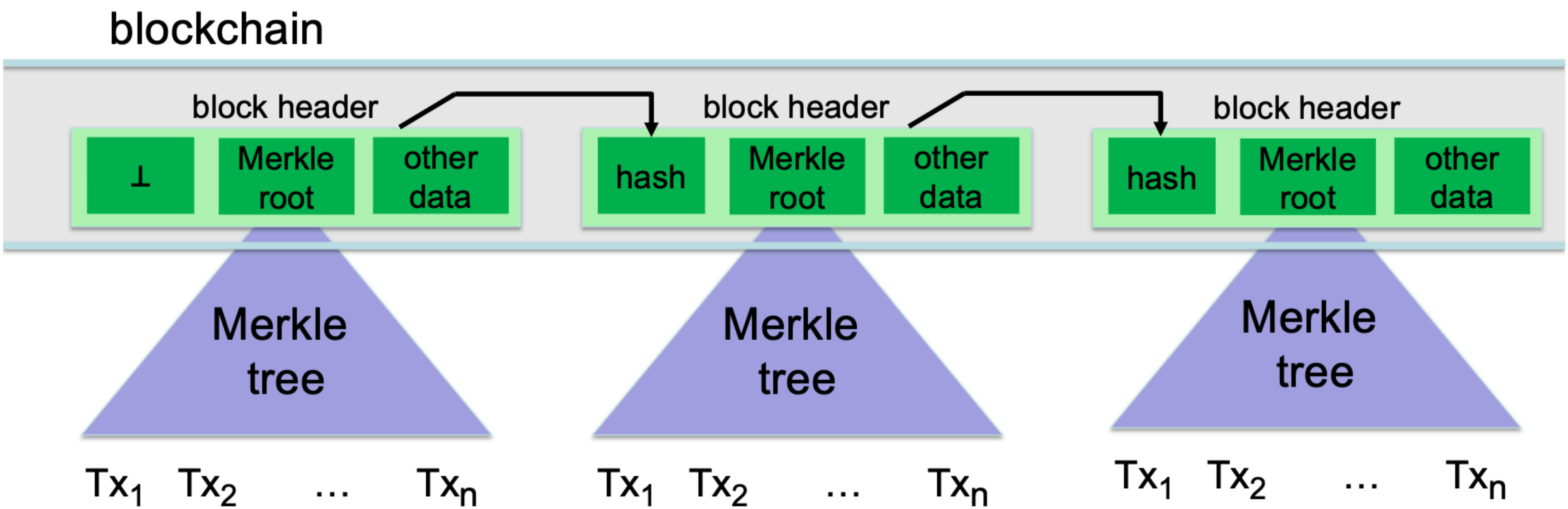
proving membership in a Merkle tree



Membership of a data block in the set of n data blocks can be verified using only O(log n) size of evidence

If you do it in a linear fashion, the size of evidence needed for such verification is O(n)

Abstract block chain

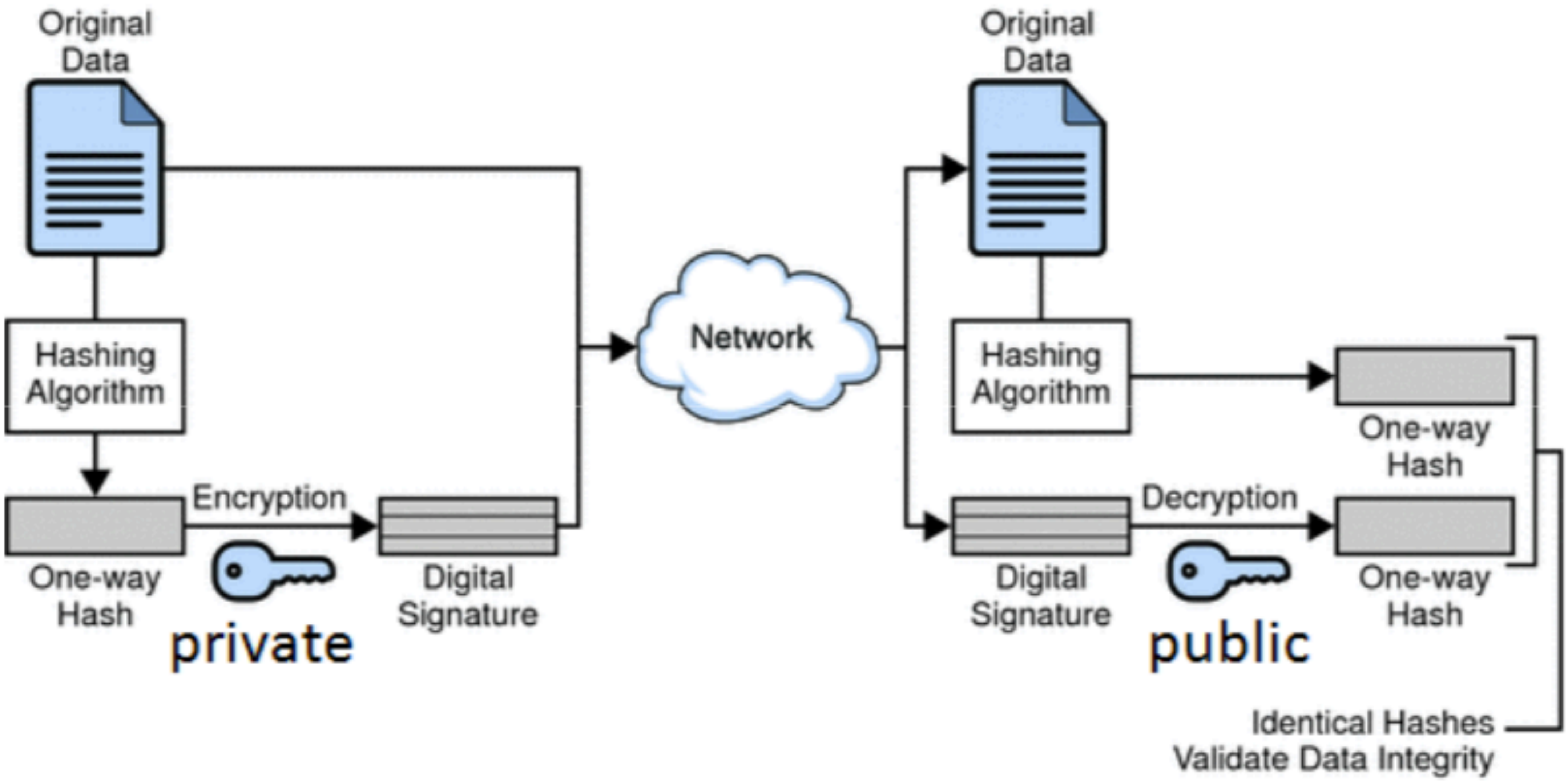


Merkle tree in blockchain; Tx can be verified to be included in a given block in Merkle tree style

Digital Signature

- Use the key X_{public} to represent a person (X has X_{public} as its identity)
  - X must be the only person who knows the corresponding private key, $X_{private}$
- If the signature of X is required on a document, M
  - Construct the digital signature using the private key and the encryption of the digest of M:
    - Digital signature = $(H(M))^{Xprivate}$
  - Send the signed message, M and its digital signature, to the receiver:
    - Signed message = $[M, (H(M))^{Xprivate}]$
  - The receiver can verify 2 things: 1) if the signed message contains the true digital signature 2) the integrity of M
    - Three steps:
      - Decrypt the digest of M with X_{public}
      - Verify that this digest is, indeed, H(M)
      - Hash M to see if you get H(M)

Digital Signature Illustrative Diagram



Non-Repudiation

- If the private key is never revealed to anyone beside X, the digital signature of X has non-repudiation property:
  - X cannot deny his/her associations/actions with $X_{private}$
- This non-repudiation property can not be had in a symmetric-key crypto system
  - If Alice sends a message to Bob, Alice can refuse that she ever sent that message and claim that Bob was sending the message to himself

Digital signature used blockchain

1. RSA signatures (not used in blockchains):
   - long sigs and public keys (≥256 bytes), fast to verify
2. Discrete-log signatures: Schnorr and ECDSA (Bitcoin, Ethereum)
   - short sigs (48 or 64 bytes) and public keys (32 bytes)
3. BLS signatures: 48 bytes, aggregatable, easy threshold (Ethereum 2.0, Chia, Dfinity)

---

4. Post-quantum signatures: long (≥768 bytes)

Let’s revisit the man-in-the-middle inherent in a public-key crypto system

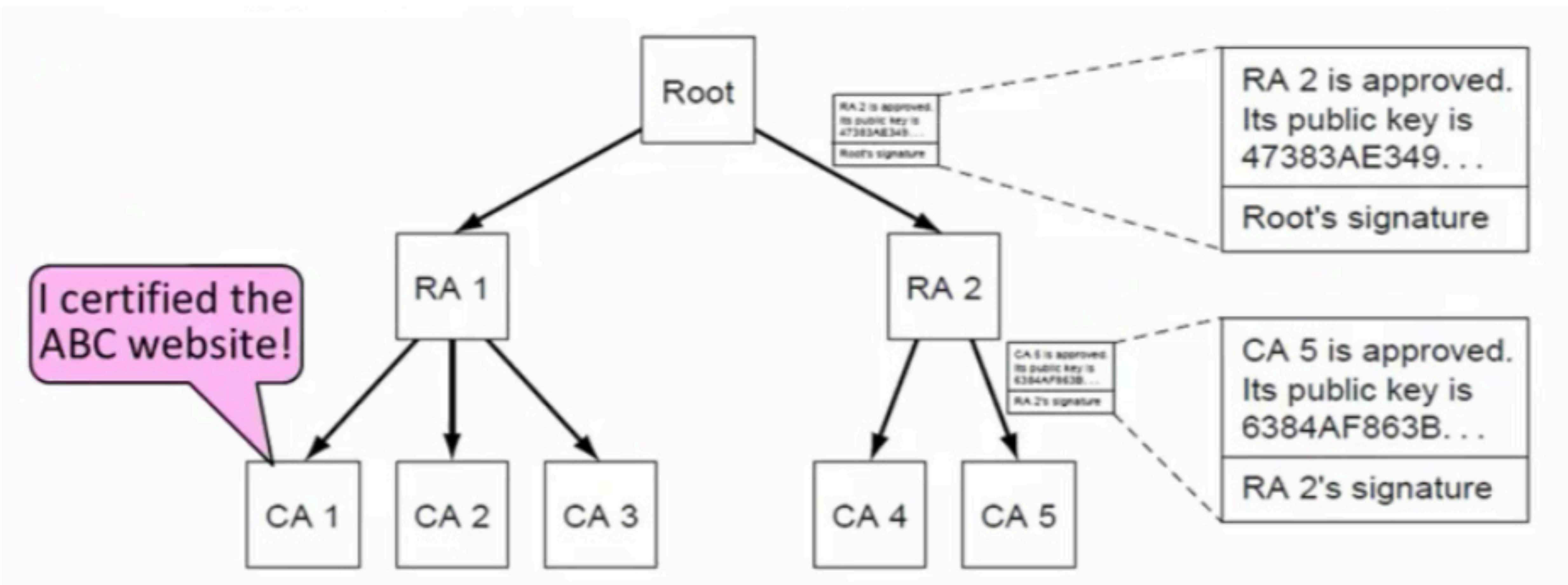
Certificate authorities

- How do we know that X_{public} really belongs to X and we are not facing a man-in-the-middle attack
  - The answer: trust a certificate authority (CA) such as Verisign or Entrust
  - X goes to a CA to presents the necessary evidence to authenticate himself
    - CA signs X_{public} with the CA's digital signature to ascertain the X_{public} does belong to X: **[X_{public}, $H(X_{public})^{CAprivate}$] - this is a certificate**
  - Before we use X_{public} in a communication session, ask X for the certificate of X_{public}
  - Then, verify the authenticity of the certificate with **CA_{public}**
  - Where do we obtain all these **CA_{public}** ?
    - Browser vendors compile these public keys of various CAs into browser programs

Certificate format

PKI (Public Key Infrastructure)

- Hierarchical management for public key verification



PKI (Public Key Infrastructure)

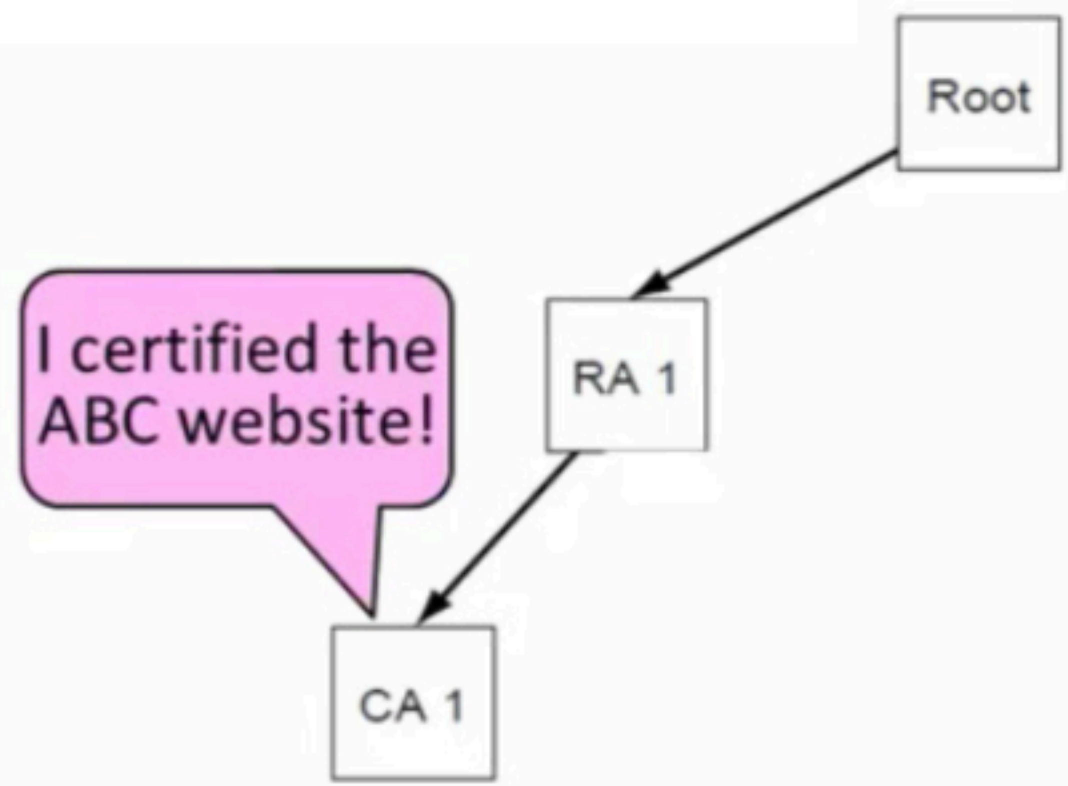
- The server must house its own certificate together with certificates of other CAs at higher levels, all the way up to the root CA

Browser has Root's public key

{RA1's key is X} signed Root

{CA1's key is Y} signed RA1

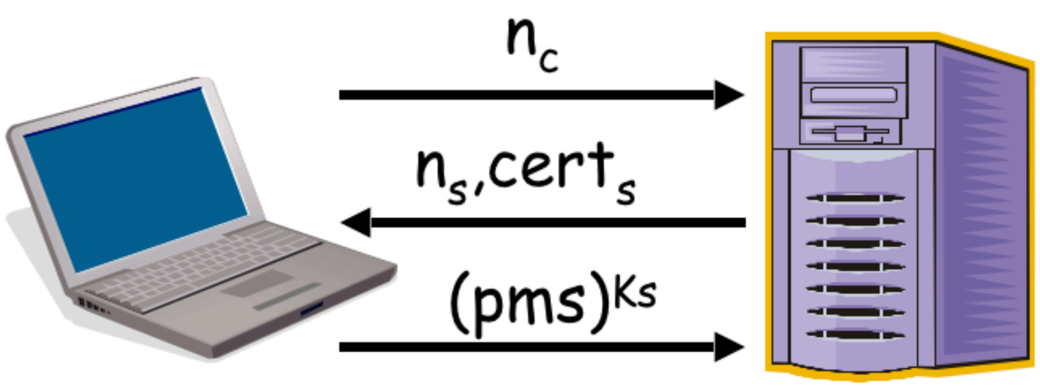
{ABC's key Z} signed CA1



Asymmetric vs symmetric key crypto system

- Symmetric-key crypto system is about 1000 times faster that its asymmetric-key counterpart
- Therefore, in a communication session, the initial stage is to use asymmetric key system to exchange a shared-key
- After the initial stage, this shared-key is used to encrypt/ decrypt all communicating data thereafter

SSL/TLS HTTPS



- SSL Web Protocol
  – Port 443: secure http (https)
  – Use public-key encryption to distribute a shared-key
- Server has a certificate containing digital signature of the certificate authority
- Want to construct a 48-byte master secret to represent a shared symmetric key
  - Client sends a random data of 28 bytes nc to the server
  - Server sends a random data of 28 bytes ns and certificate certs to the client
  - Client verifies the certificate, making sure it is talking the right server
  - Client constructs a pre-master secret (pms) of 46 bytes encrypted using the server's public key
  - At this point, the server and the client share nc , ns ,and pms, both use this shared information to construct a master secret of 48 bytes
  - A shared symmetric key to be used in subsequent communication session after this initial setup stage can be generated from this master secret

What We Have Learned

- Asymmetric key crypto system
- Man-in-the-middle problem
- Hash function: its usage and security properties
    - Hash pointer
    - Merkle tree
    - Digital signature
- Certificate authorities to solve the man-in-the-middle problem
- SSL/TLS