Berkeley CS276 & MIT 6.875

Merkle Trees and Transparency Logs

Lecturer: Raluca Ada Popa

Oct 12, 2020

Announcements

- Starting to record
- This lecture:
  - Applied: practice digital signatures and CRH in a real cryptographic system
  - Focus is on systems building with crypto, so less time for formalism
  - Will post lecture after class due to Q&A

Recall: Collision Resistant Hash Function (CRH)

Let $H: \{0,1\}^* \to \{0,1\}^m$ is a collision resistant hash function if for all PPT algorithms A, for all k sufficiently large:

$$\Pr\left[(x, y) \leftarrow A\left(1^k\right) s.t. H(x) = H(y) \wedge x \neq y\right] \leq negl(k)$$

Merkle trees

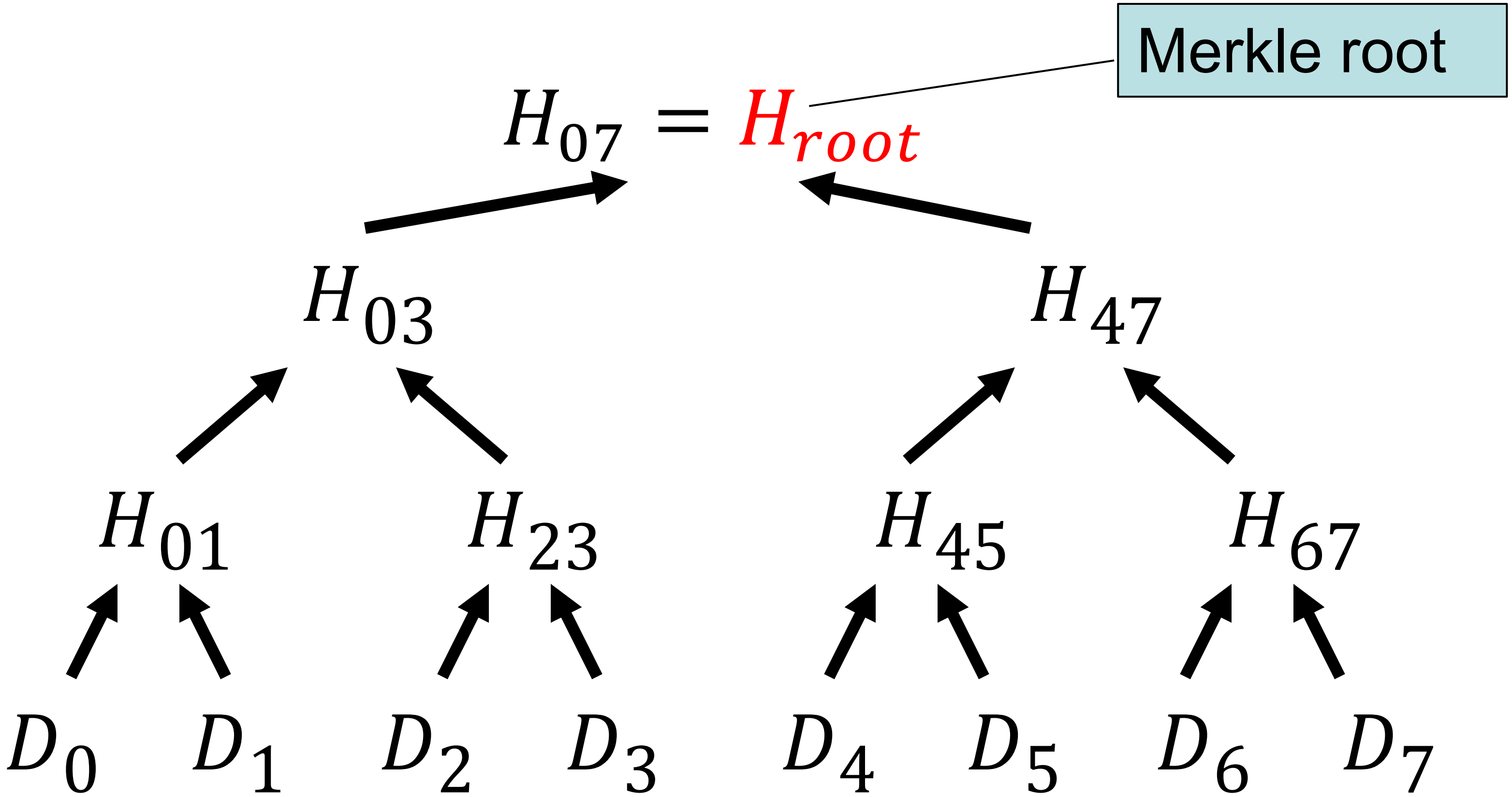
- A very useful tool invented by Ralph Merkle in 1979
- Used in many theoretical constructions and practical crypto systems
  – Bitcoin
  – Certificate & Key Transparency
  – secure storage

Merkle Hash Tree

A hash tree over a set of data values $D_0, D_1, \ldots, D_N$

Each node is the hash of its two children:

$H_{03} = hash(H_{01}, H_{23})$, where $hash$ is CRH

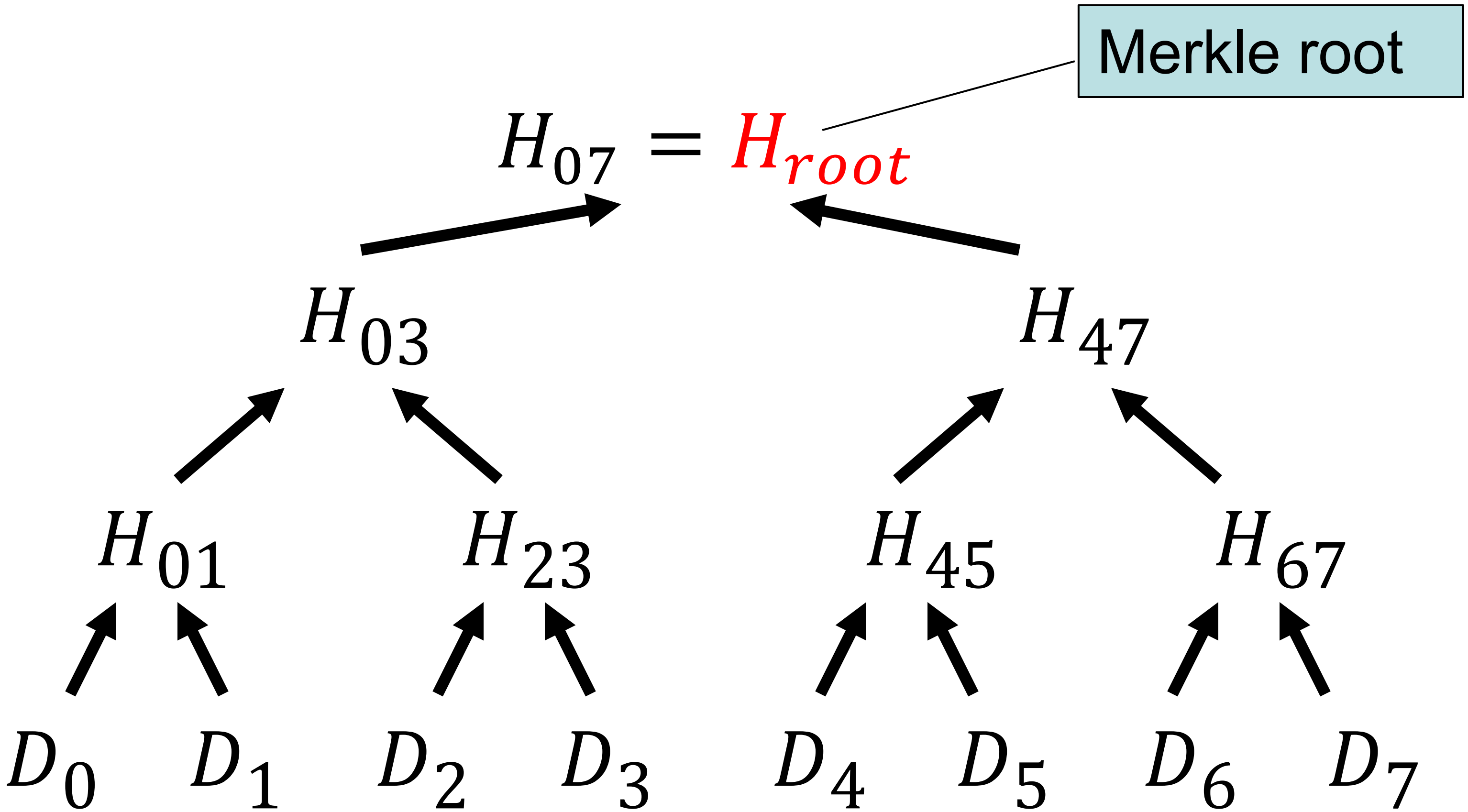


(Assume each D_i has a data tag and padded to a fixed length)

Merkle Hash Trees

Claim: If $hash$ is a CRH then H_{root} is a CRH.

Proof: ?

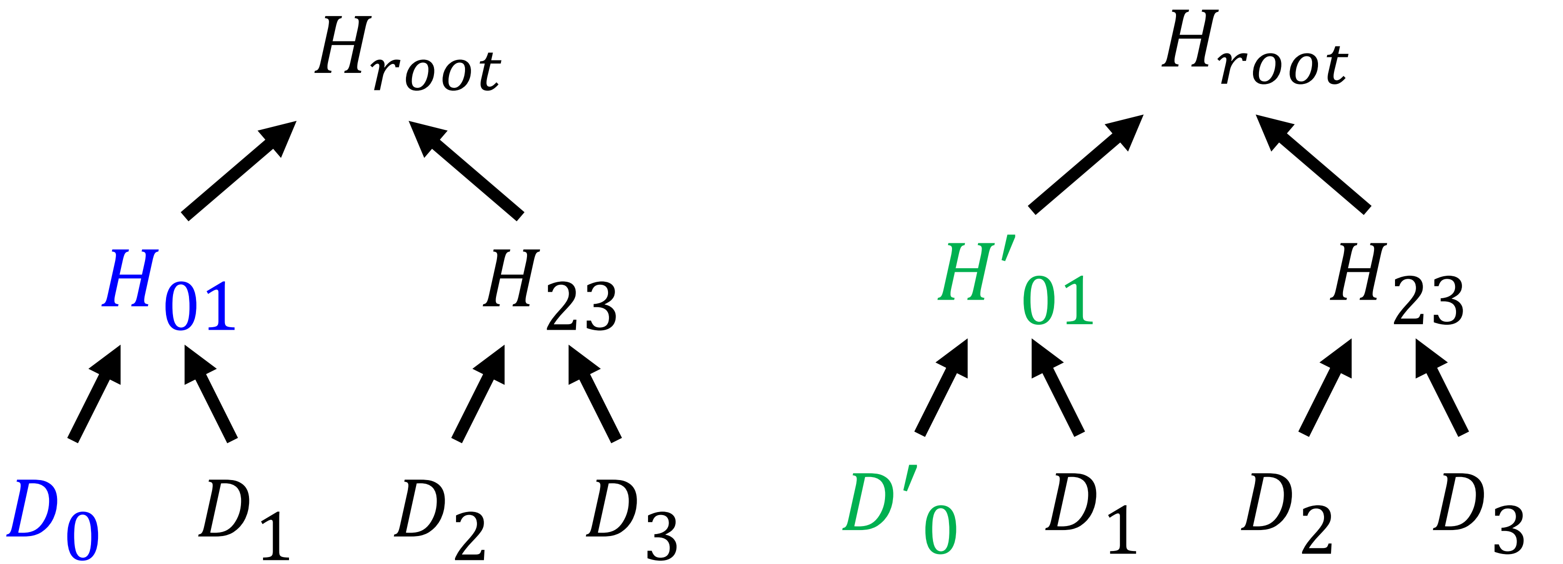


Merkle Hash Trees

Claim: If $hash$ is a CRH then H_{root} is a CRH.

Proof: Assume H_{root} is not a CRH. Let's show that $hash$ is not a CRH (i.e., we produce a collision in poly time) to achieve contradiction.

$\exists$ $PPT\ A$ that can find a collision $(D_0, \ldots, D_m)$ and $(D'_0, \ldots, D'_n)$

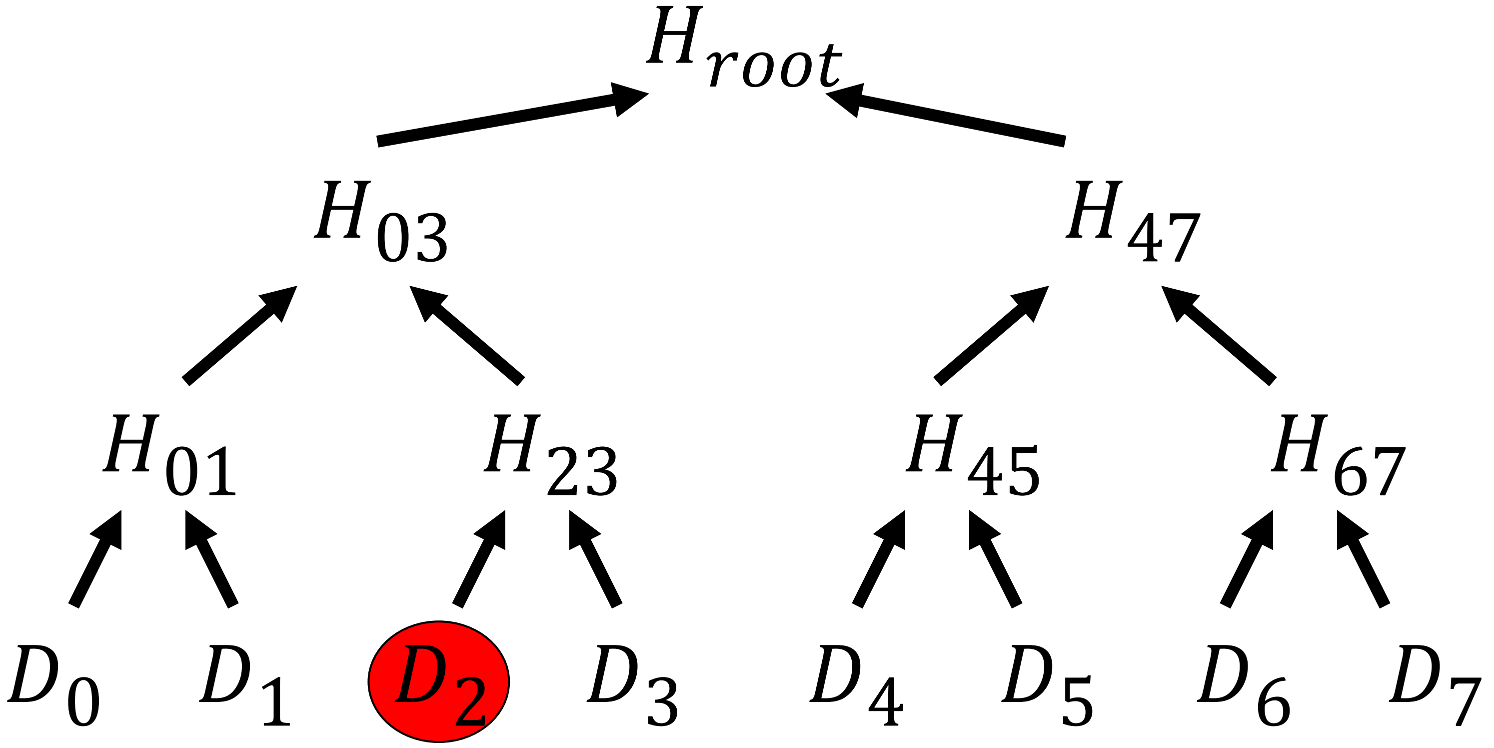


(H_{01}, H_{23}) and (H'_{01}, H_{23}) are a collision

Authentication path

Assume a verifier knows H_{root}.

How can Alice prove to the verifier that D_2 was among the data items that produces H_{root}?

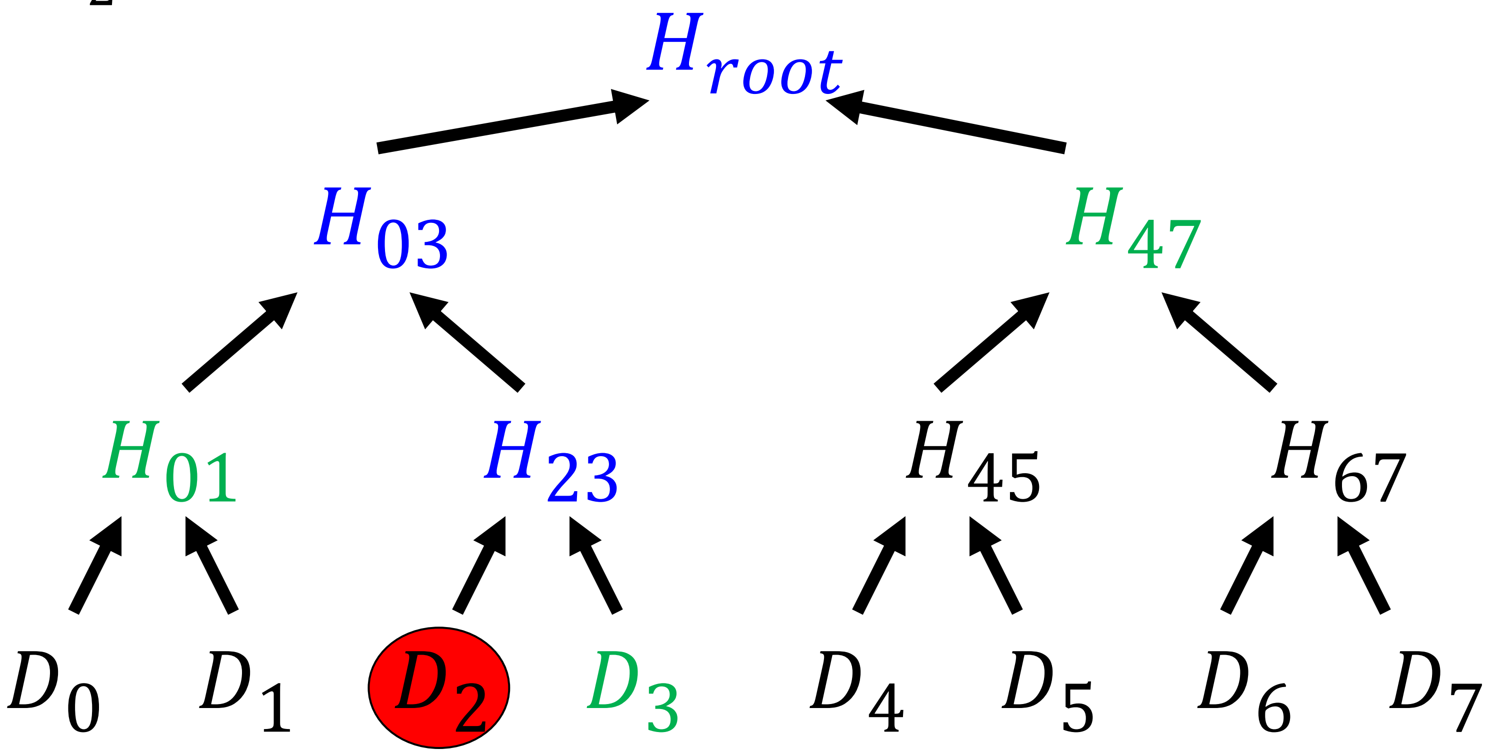


Authentication path

Assume a verifier knows H_{root}.

How can it authenticate D_2?

Alice provides authentication path: siblings of nodes from D_2 to root



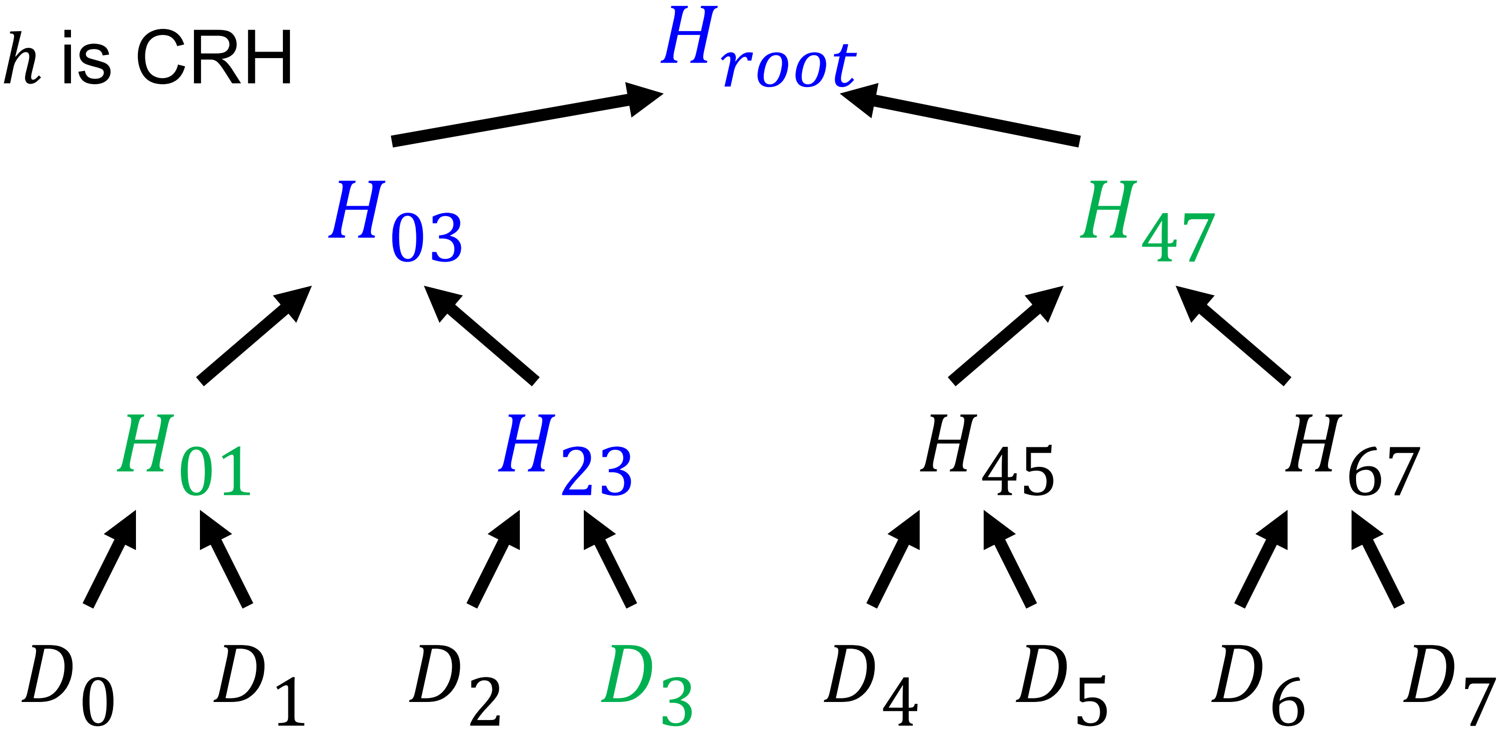
Authentication path

Assume a verifier knows H_{root}.

Alice provides authentication path: siblings of nodes from D_2 to root.

Why can't Alice lie?

$hash$ is CRH



Asymptotics

n # of data items m hash size

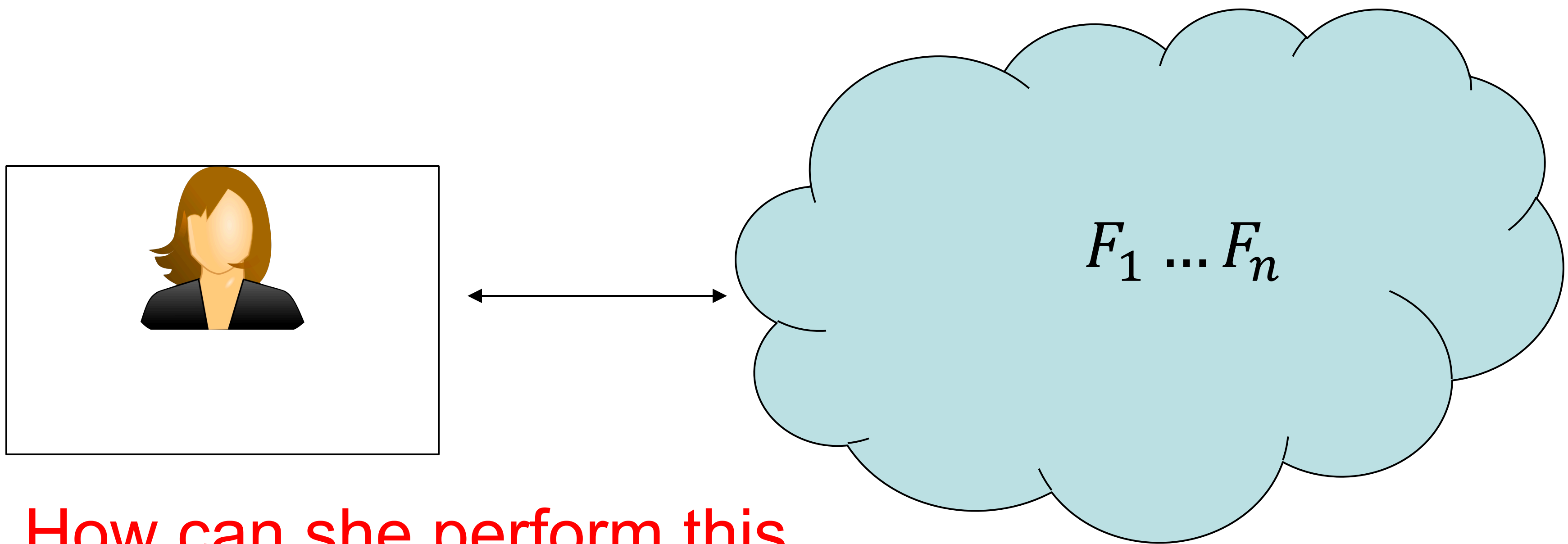
Size of Merkle tree: $O(n)$

Size of Merkle root: $O(m)$

Size of authentication path: $O(m \log n)$

Warmup app: Secure storage

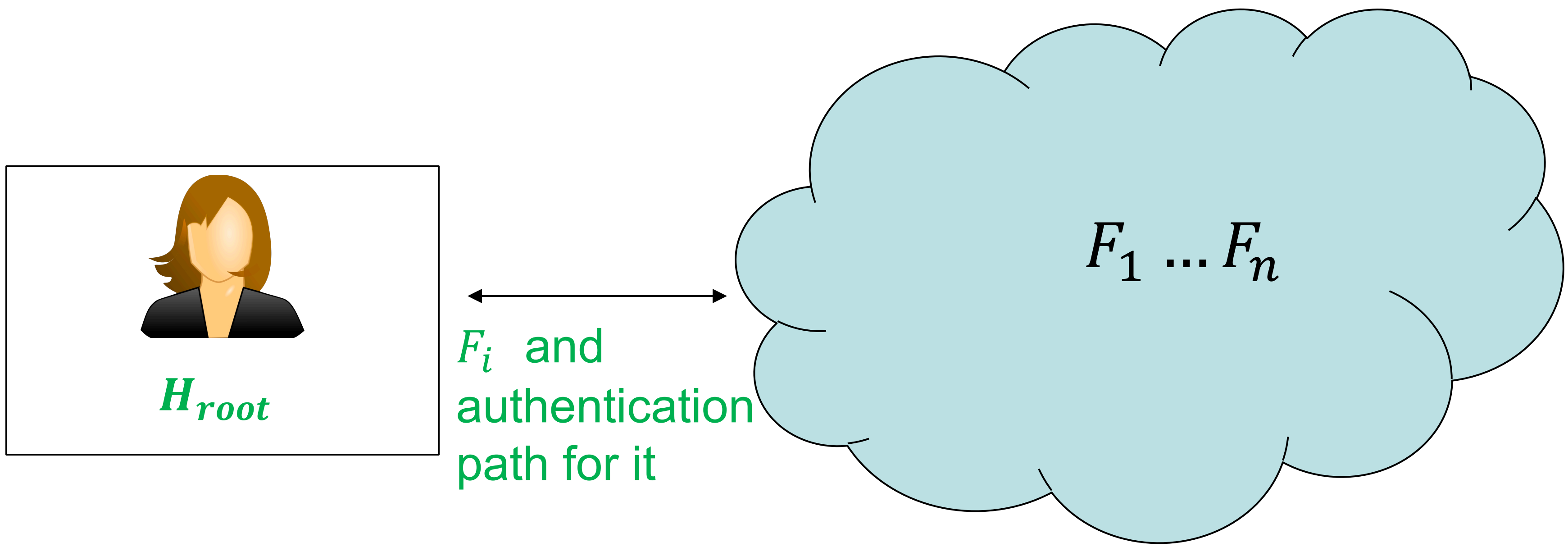
Alice has files $F_1 \ldots F_n$, stores them on the cloud. When she retrieves file i, she wants to verify that an untrusted cloud did not modify it.



How can she perform this check sublinear in n?

Secure storage

Alice has files $F_1 \dots F_n$, stores them on the cloud. When she retrieves file i she wants to verify that an untrusted cloud did not modify it.



Transparency logs

Web certificates

A website like Google obtains a certificate of the form

$$sign_{CA}(PK_{bank}, \text{“}bank.com\text{”}, expiry)$$

where CA is a certificate authority trusted by user browsers

CAs have often been compromised

Today, Microsoft issued a Security Advisory warning that fraudulent digital certificates were issued by the Comodo Certificate Authority. This could allow malicious spoofing of high profile websites, including Google, Yahoo! and Windows Live.

The advisory states how 9 certificates were fraudulently issued by Comodo for the following names:

- login.live.com
- mail.google.com
- www.google.com
- login.yahoo.com (3 certificates)
- login.skype.com
- addons.mozilla.org
- “Global Trustee”

Why is CA compromise bad?

User encrypts https traffic with attacker key

The attacker who penetrated the Dutch CA DigiNotar last year had complete control of all eight of the company's certificate-issuing servers during the operation and he may also have issued some rogue certificates that have not yet been identified. The final report from a security company commissioned to investigate the DigiNotar attack shows that the compromise of the now-bankrupt certificate authority was much deeper than previously thought.

DigiNotar®
A VASCO COMPANY

A Dutch certificate authority that suffered a major hack attack this summer has been unable to recover from the blow and filed for bankruptcy this week.

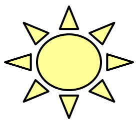
Core problem

When seeing a certificate for google.com, we fundamentally cannot tell if a certificate is corrupted or not

A huge problem since https’s creation, many attempts at solutions have been unsatisfactory

Only in recent years a satisfactory solution emerged

Certificate Transparency (CT)


"Sunlight is the best disinfectant."
— Supreme Court Justice Louis Brandeis

- Ensure transparency: everyone sees the same certificates
  - Both the user and the cert owner
- Ben Laurie, Adam Langley and Emilia Kasper proposed CT in IETF Internet Draft in 2012 under the code-name "Sunlight".

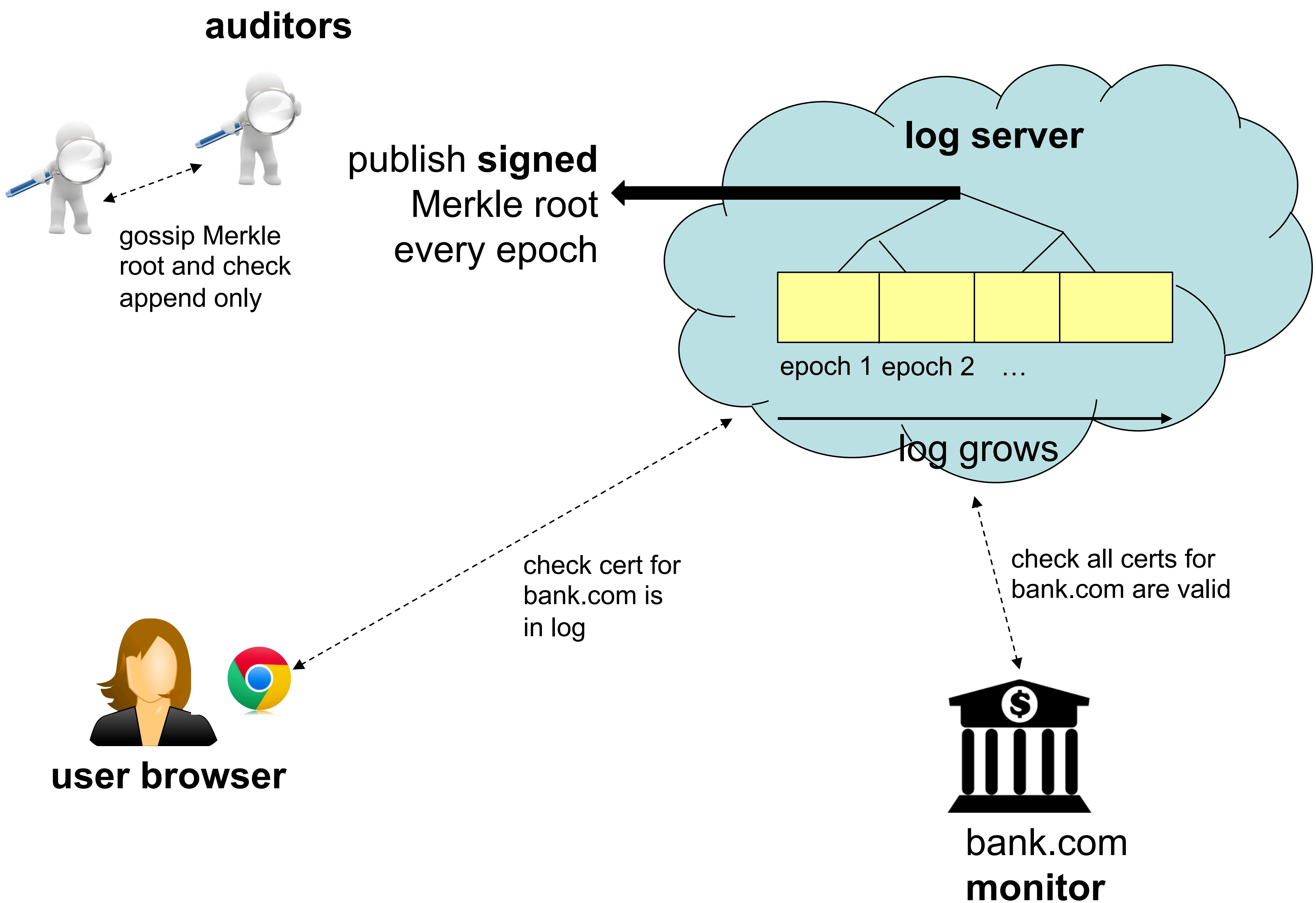
Adoption

- As of May 2020, CT has publicly logged over 9.2 billion certificates.
- Google Chrome requires web certificates issued after April 30,2018 to appear in a CT log.

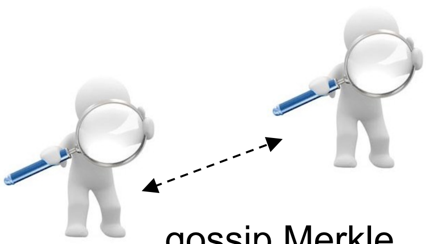
Parties

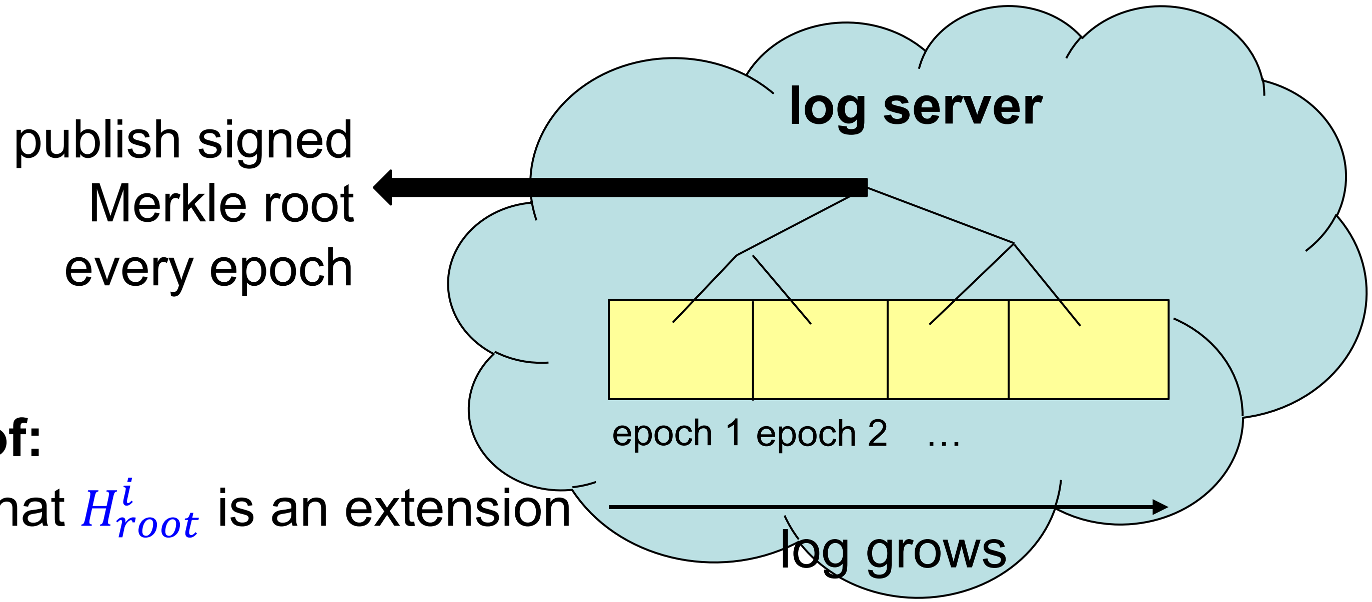
- Log server: stores certs in a log
  - Untrusted (except for DoS)
- Monitors: owners of certificates
  - Trusted to monitor its cert
- Auditors: audit the log is append-only
  - Anyone can be an auditor
  - Untrusted except that at least one auditor should be honest and reachable
- User browsers: check that certs appear in the log
  - Trusted to check each cert it receives

No central point of attack for all certificates


auditors
log server
publish **signed** Merkle root every epoch
gossip Merkle root and check append only
epoch 1 epoch 2 …
log grows
check cert for bank.com is in log
check all certs for bank.com are valid
user browser
bank.com **monitor**

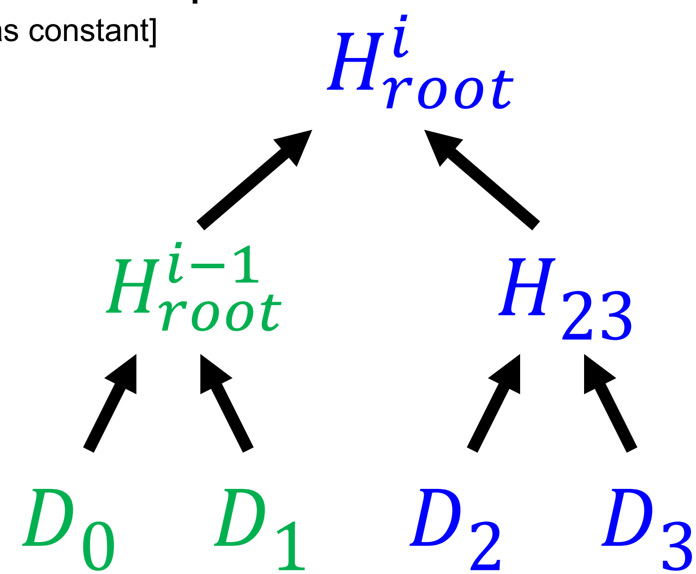
auditors



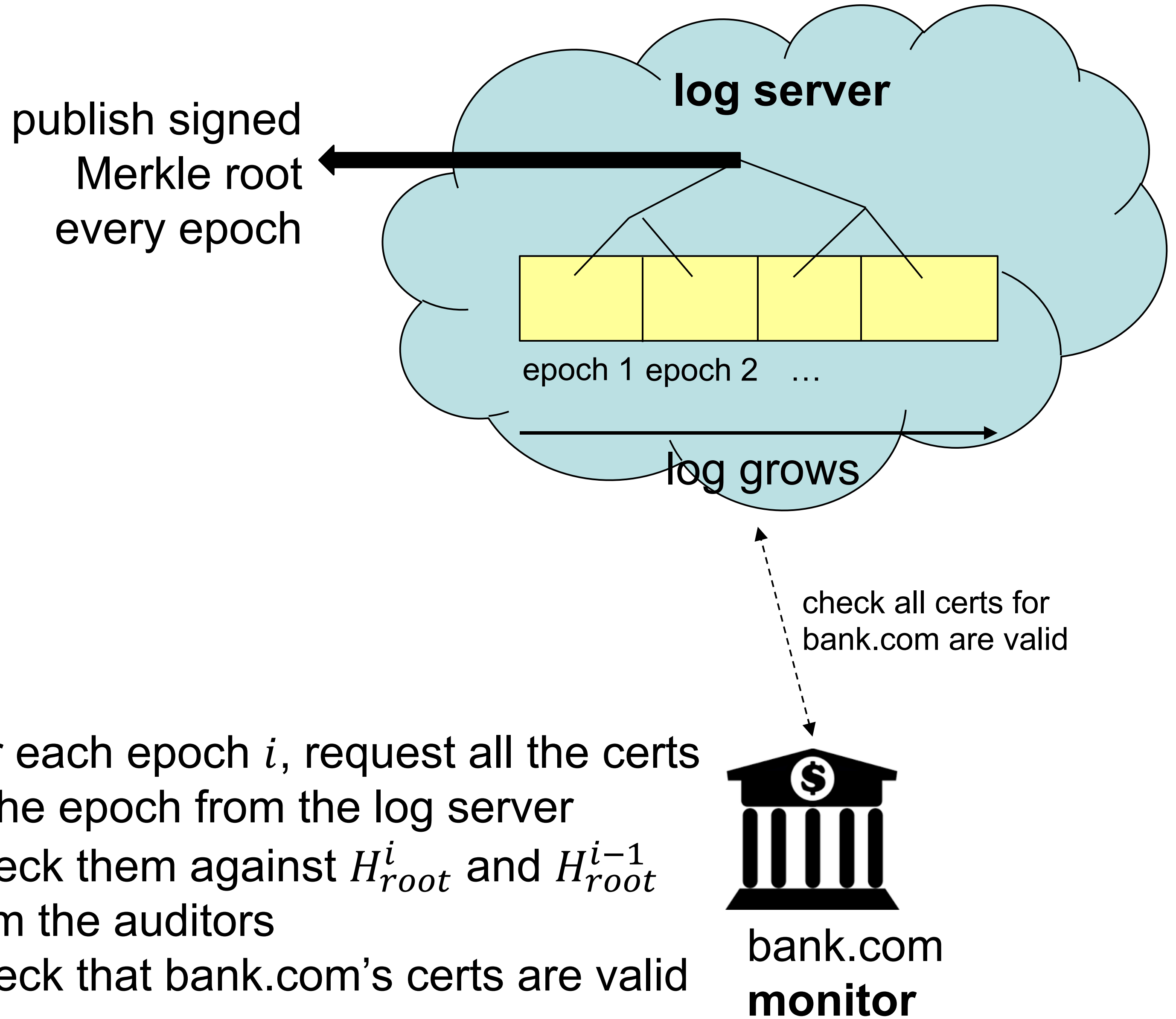


Consistency proof:

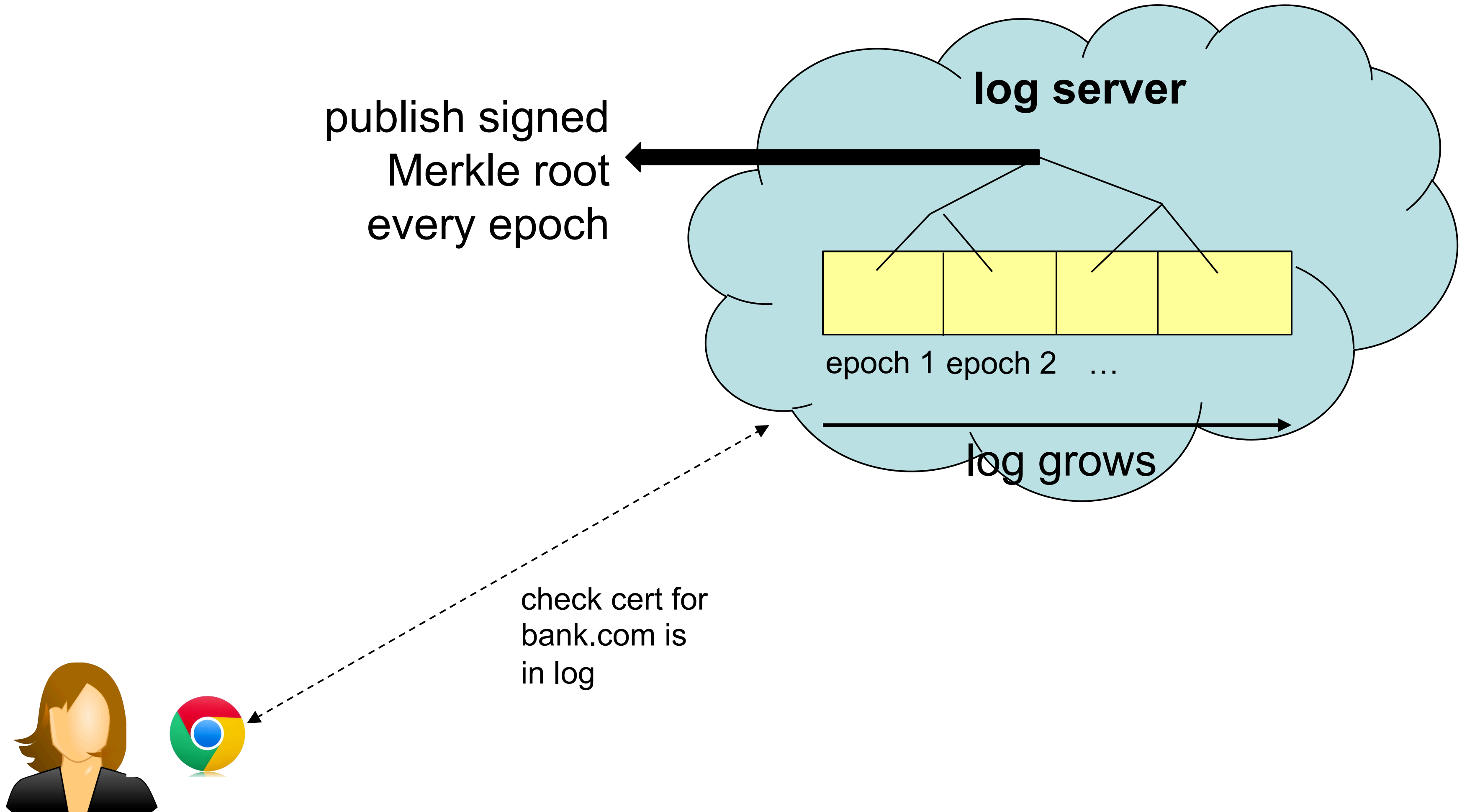
- Server proves that H^{i}_{root} is an extension of H^{i-1}_{root}
- $O(\log n)$, for n #epochs
  [treating hash size as constant]



(In practice, the tree growing to the right will not be full so there are some extra technicalities)



- For each epoch i, request all the certs in the epoch from the log server
- Check them against H_{root}^{i} and H_{root}^{i-1} from the auditors
- Check that bank.com's certs are valid



Inclusion proof:

- Obtain H^i_{root} from auditors
- Server proves that cert is in H^i_{root} by supplying the authentication path
- $O(\log n)$, for n #epochs

Guarantee: transparency

Assuming

- $hash$ is a CRH,
- signature scheme is existentially unforgeable,
- at least one auditor is honest and reachable,
- a monitor monitors its certs,

If a user receives a compromised cert,

and the user checks inclusion for the cert,

then

- either the monitor detects the compromised cert or some party detects log server misbehavior.

Any questions on CT?