Lecture 6:
Watching the Gatekeepers:
Certificate Transparency

CPSC 444/544, 2026 Spring

Instructor: Fan Zhang

Yale

Computer Science

BBC

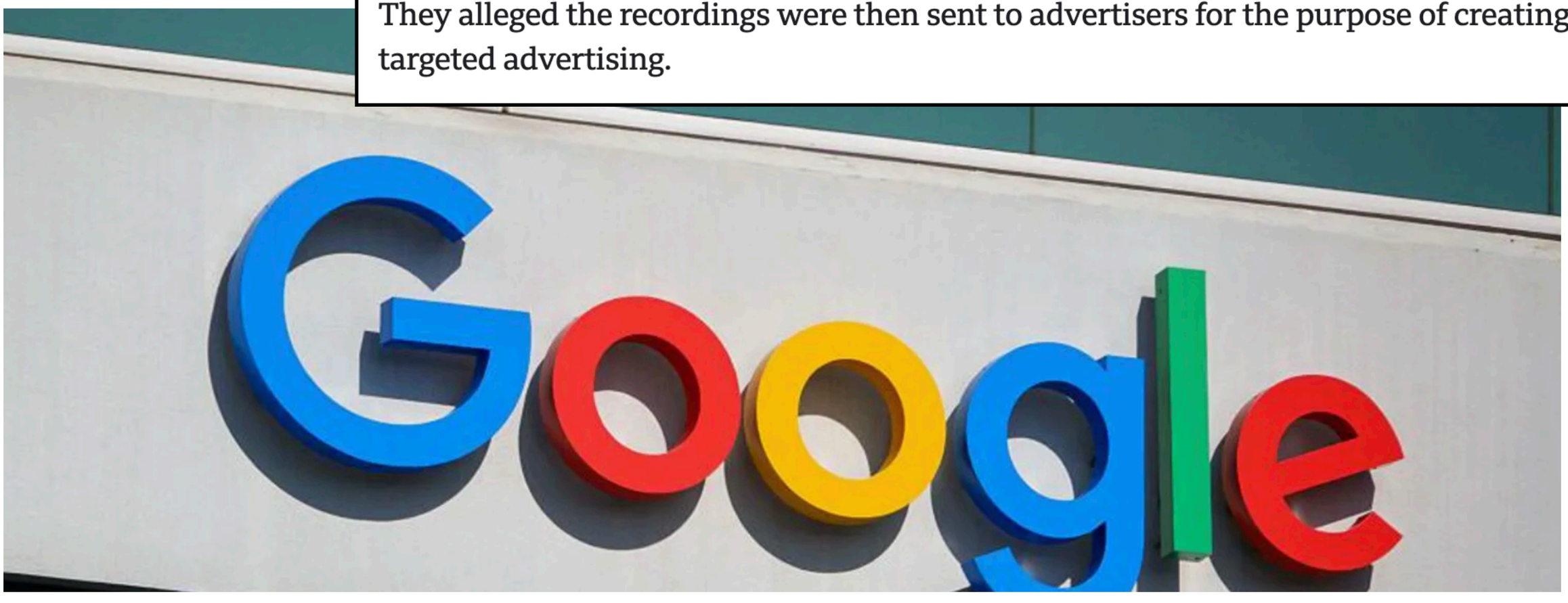
Google to pay $68m to settle lawsuit claiming it recorded private conversations

2 days ago

Laura Cress
Technology reporter

But the lawsuit claimed Google Assistant would sometimes turn on by mistake - the phone thinking someone had said its activation phrase when they had not - and recorded conversations intended to be private.

They alleged the recordings were then sent to advertisers for the purpose of creating targeted advertising.

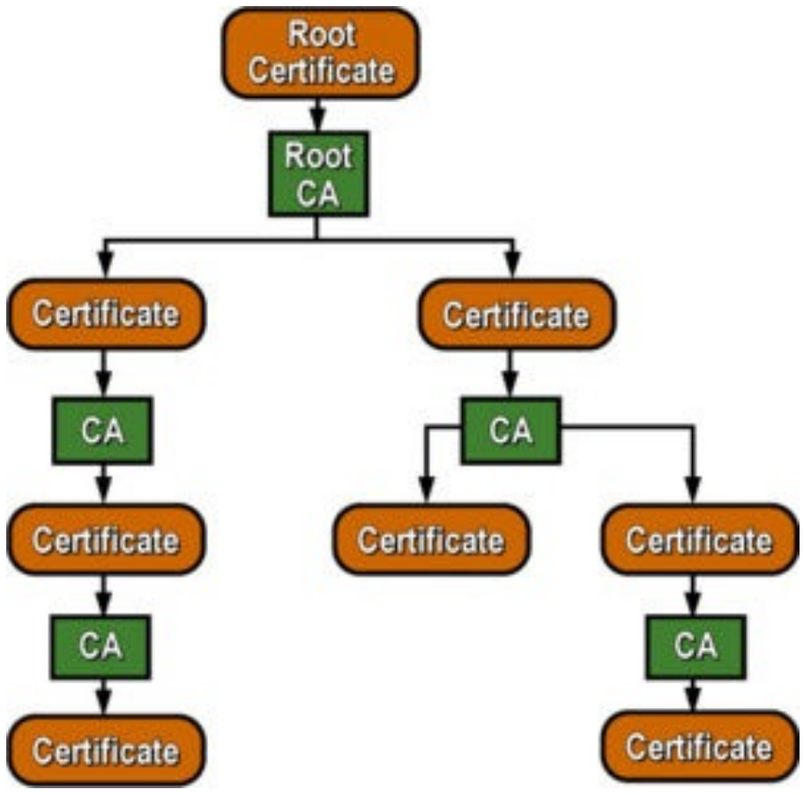


Certificate Authority (CA)

- Have you used CAs?
- CAs are the gatekeepr of secure web, deciding which *PKs* can represent which *identities*
- Around 100+ CAs trusted by major browsers
- Who watches the gatekeepers?

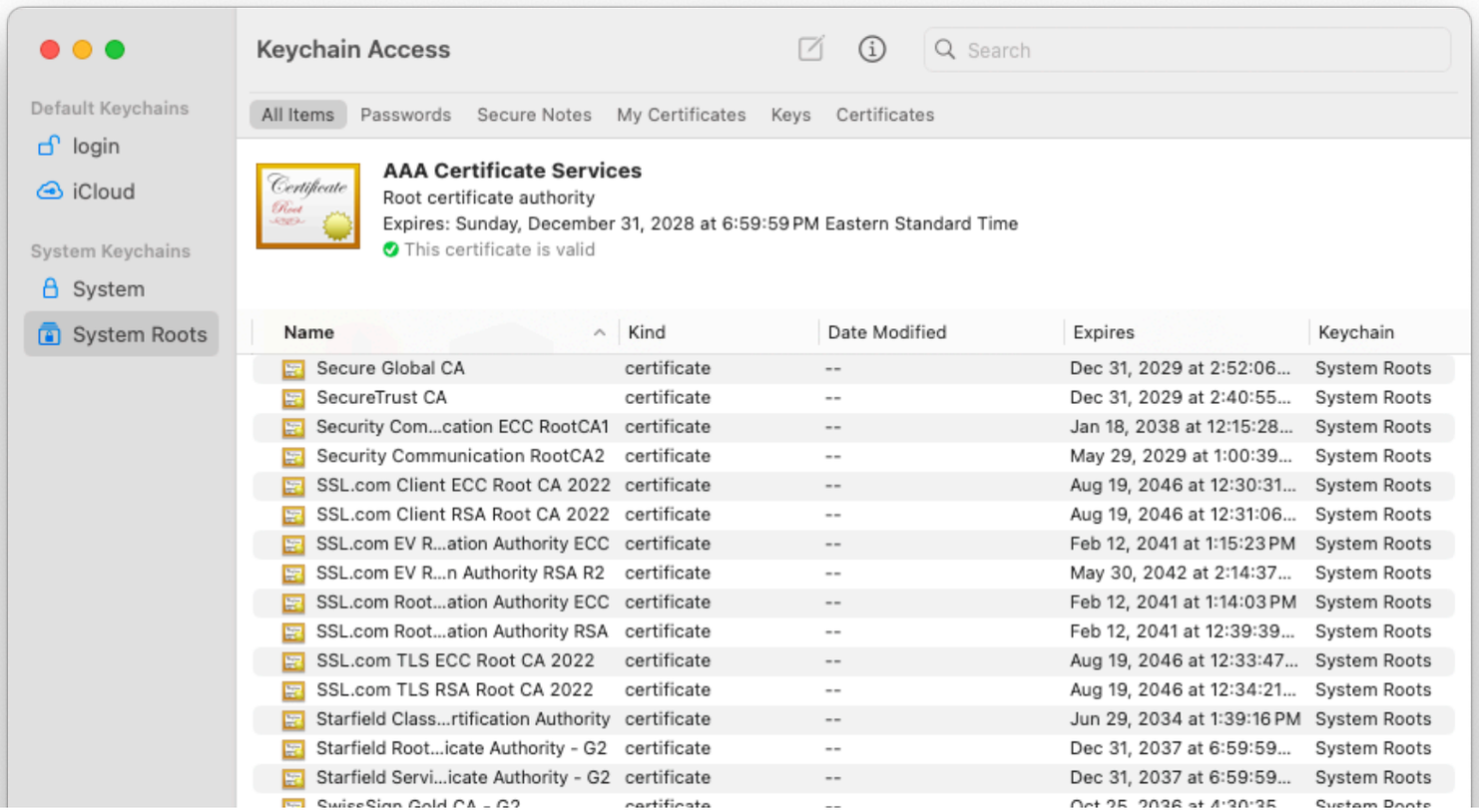
Background on the operation of CAs

Public Key Infrastructure (PKI) for Web



- CA Hierarchy
  - Root CA: endorses Intermediate CAs
  - Intermediate CAs: appoints lower-level CAs or handle day-to-day issuance
  - Benefits: “hot keys” v.s. “cold key” separation
- Browsers typically trust 100+ root CAs
  - Browsers have different “root programs” to vet CAs
- TLS servers present *certificate chains* to browsers

Root CAs are **baked in** by system vendors

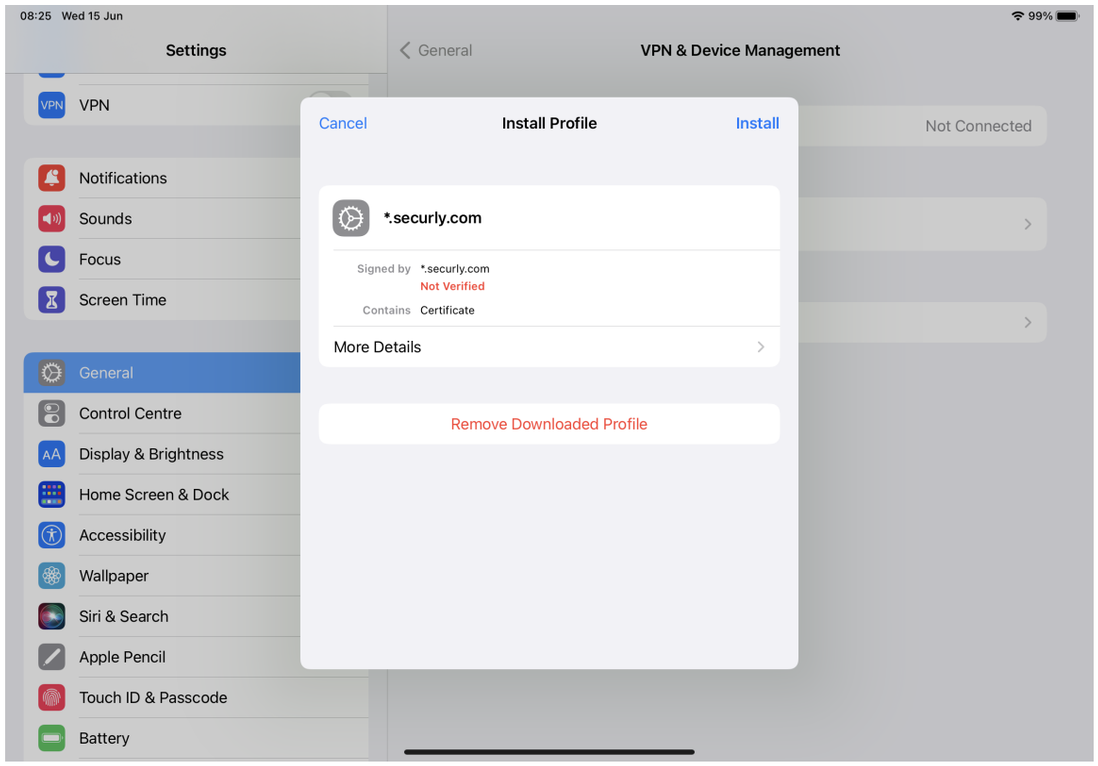


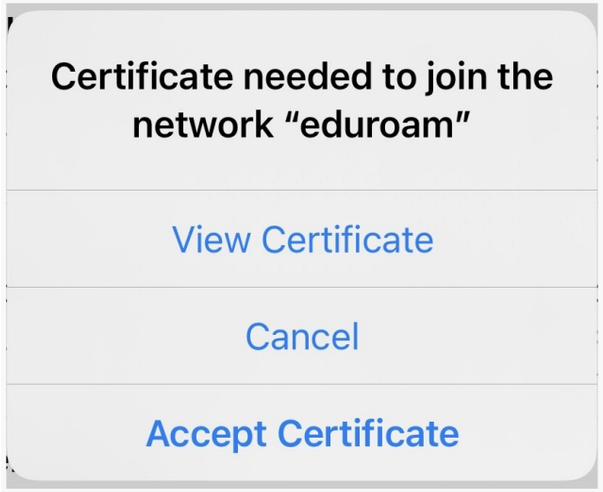
"System Roots" Keychain on macOS

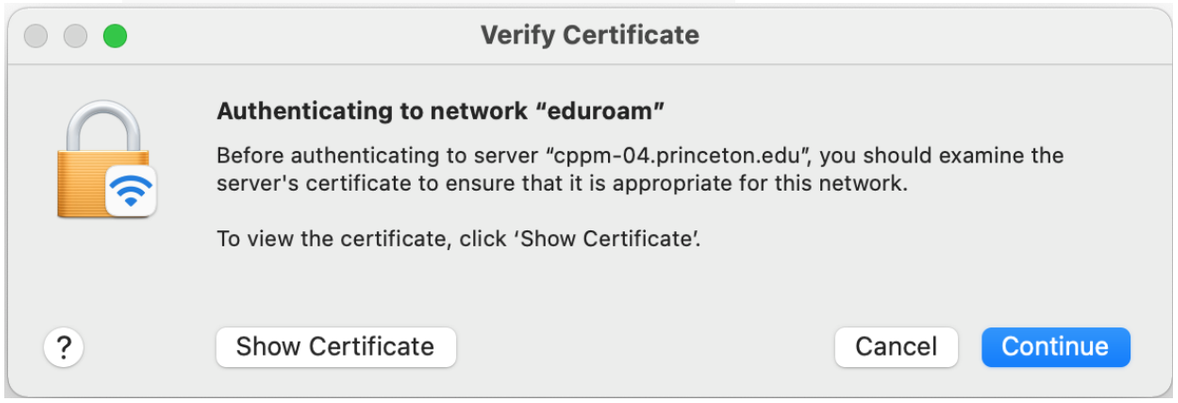
```
/etc/ssl/certs/ca-certificates.crt,  // Debian/Ubuntu/Gentoo etc.
/etc/pki/tls/certs/ca-bundle.crt",   // Fedora/RHEL 6
/etc/ssl/ca-bundle.pem,              // OpenSUSE
```

Be careful when installing certs

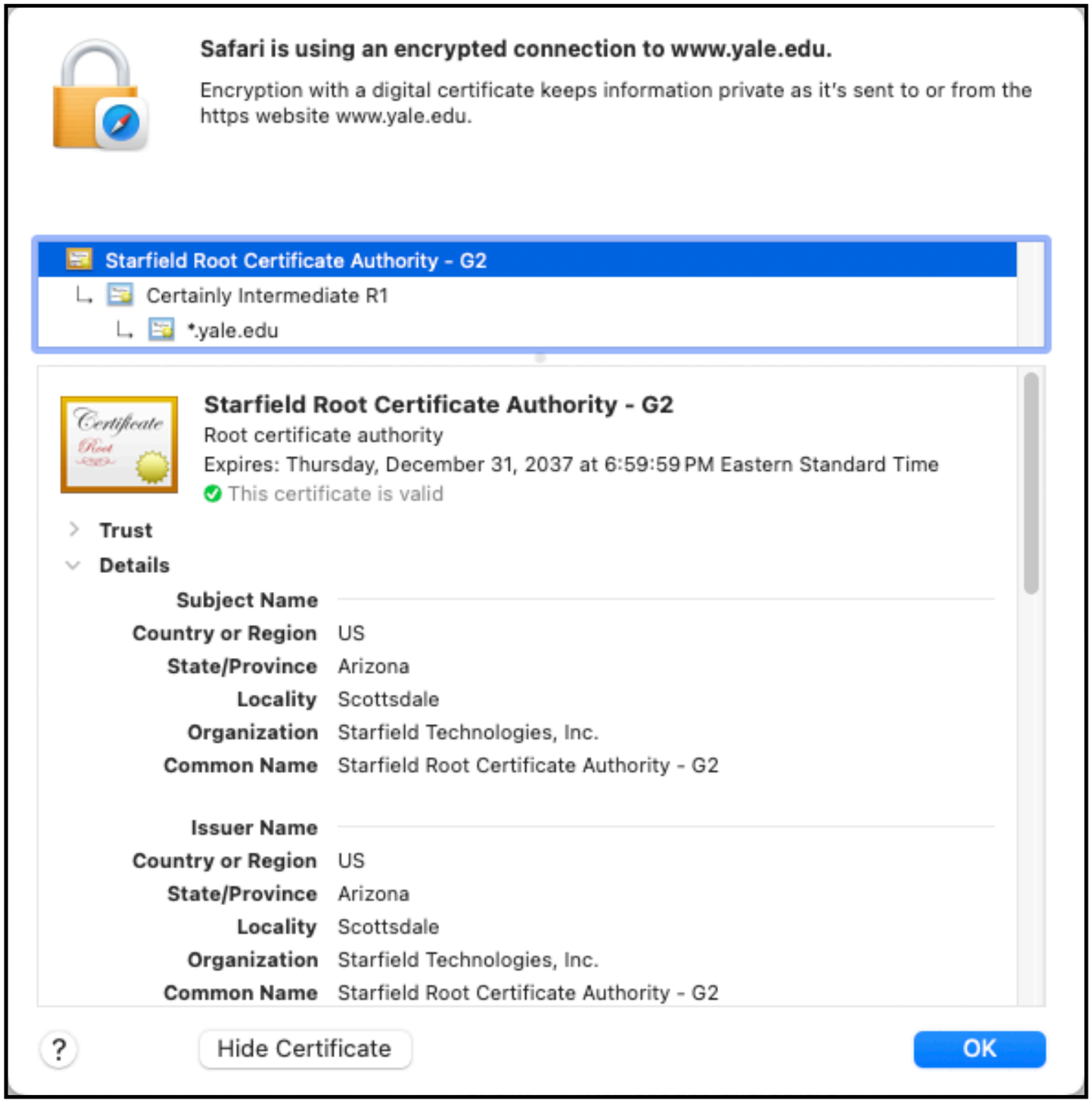
- A decent OS should warn their users when some applications try to install new certificates
- Examine the cert to make sure it's not doing more than what's needed.







Cert chains for*.yale.edu



- **Common Name**: *.yale.edu
- **Issuer by**: Certainly Intermediate R1
- Which itself is certified by Starfield Root CA - G2

starfieldtech.com

STARFIELD
TECHNOLOGIES

SSL Certificates protect customers' data when they browse or shop online.

Our SSL Certificates protect a single domain or multiple domains websites

SSL CERTIFICATES DOCUMENTATION

SHOP SSL CERTIFICATES

Starfield Technologies

Owner	GoDaddy
Country	United States
Markets	World
Website	starfieldtech.com

whitehouse.gov

Certificate Viewer: whitehouse.gov

General | Details

Issued To

Common Name (CN)	whitehouse.gov
Organization (O)	<Not Part Of Certificate>
Organizational Unit (OU)	<Not Part Of Certificate>

Issued By

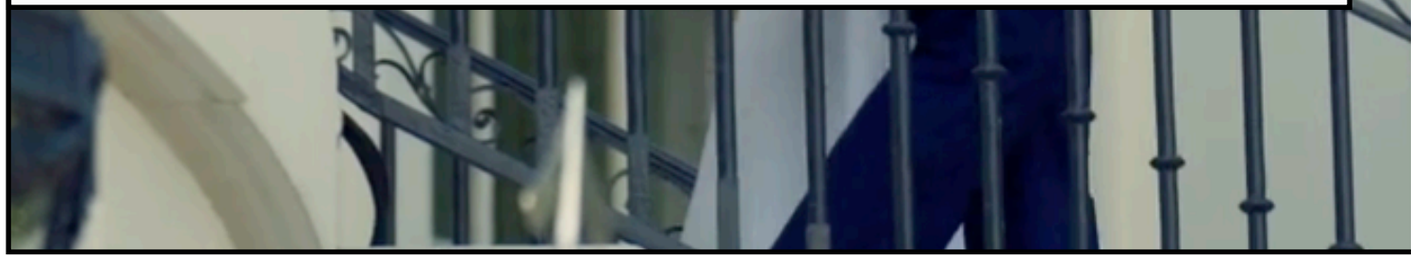
Common Name (CN)	E7
Organization (O)	Let's Encrypt
Organizational Unit (OU)	<Not Part Of Certificate>

Validity Period

Issued On	Tuesday, January 27, 2026 at 5:14:00 AM
Expires On	Monday, April 27, 2026 at 6:13:59 AM

SHA-256 Fingerprints

Certificate	5f2f22fc85cb1309f6410e8d2afed07fbc8cfcb6f4e2c79ece663c3a71082054
Public Key	055d112909c4c2fd82461c9f5e89fd2b063fb341d03e7f0c3f37f271c6fa2f51



Let's Encrypt

Let's Encrypt

Formation	November 18, 2014; 11 years ago
Founder	Electronic Frontier Foundation Mozilla Foundation University of Michigan Akamai Technologies Cisco Systems
Headquarters	San Francisco, California, U.S.
Coordinates	37.800322°N 122.449951°W
Services	X.509 certificate authority
Parent organization	Internet Security Research Group
Budget	US$3.6 million[1] (2019)
Staff	27[2] (2023)
Website	letsencrypt.org

How do (leaf) CAs verify domain ownership?

- **Email verification**: send a one-time code to (say) admin@domain.com.
- **DNS record verification**: challenge the domain owner to add strings to a DNS record (TXT or CNAME).
- **HTTP verification**: challenge the domain owner to add strings to http://domain.com/acme-challenge/
- DNS and HTTP verification can be done automatically
  - Let's Encrypt invented ACME (Automated Certificate Management Environment)
- **Fundamentally trusted: It may not do nothing (no way to verify that they did)**

What if certs are mis-issued?

- Suppose Alice trusts a CA that associates PK_{Attacker} with google.com (called **mis-issuance**)
- Attacker can now 1) impersonate google.com and/or 2) decrypt all traffic from user to Google

Attacker with a certificate for google.com from a trusted CA

Google

Real-world MITM attacks involving bogus certs

- Typical incentives: advertisement, surveillance
- **Superfish** adware shipped with some Lenovo PCs install a root CA certificate to decrypt HTTPS traffic for advertisement.
- **Xfinity** MITM attacks customers to inject ads.
  - “Xfinity is Man-in-the-Middle (MITM) Attacking my Internet” — Alex Piechowski 10/29/2019
- **Kazakhstan government** mandated installation of “national security certificate” to perform nation-wide MITM attacks.
  - “Investigating Large Scale HTTPS Interception in Kazakhstan” Raman et al. IMC’20
- **US and EU** have legal framework for “lawful interception”

Mis-issuance can happen for many other reasons

- **2001, VeriSign**: Issued two Class 3 code-signing certificates to someone claiming to be from Microsoft, which could have been used to sign malicious code as if it were from Microsoft.
- **2008, Trustico (via Comodo)**: Issued certificates **without proper validation**, including one for a domain that did not belong to the requester.
- **2011, Comodo**: A reseller in Iran was **compromised**, and attackers issued fraudulent certificates for high-profile domains like Google, Yahoo, and Microsoft.
- **2011, DigiNotar**: **Hacked**, leading to the issuance of hundreds of fraudulent certificates, including one for Google.com. The breach resulted in DigiNotar's bankruptcy.
- **2012, TURKTRUST**: **Mistakenly** issued intermediate CA certificates instead of regular SSL certificates, which were used to issue unauthorized certificates for Google domains.
- **2015, CNNIC**: Issued an **intermedia** mpany, which was then used to issue unauthorized certificates for Google

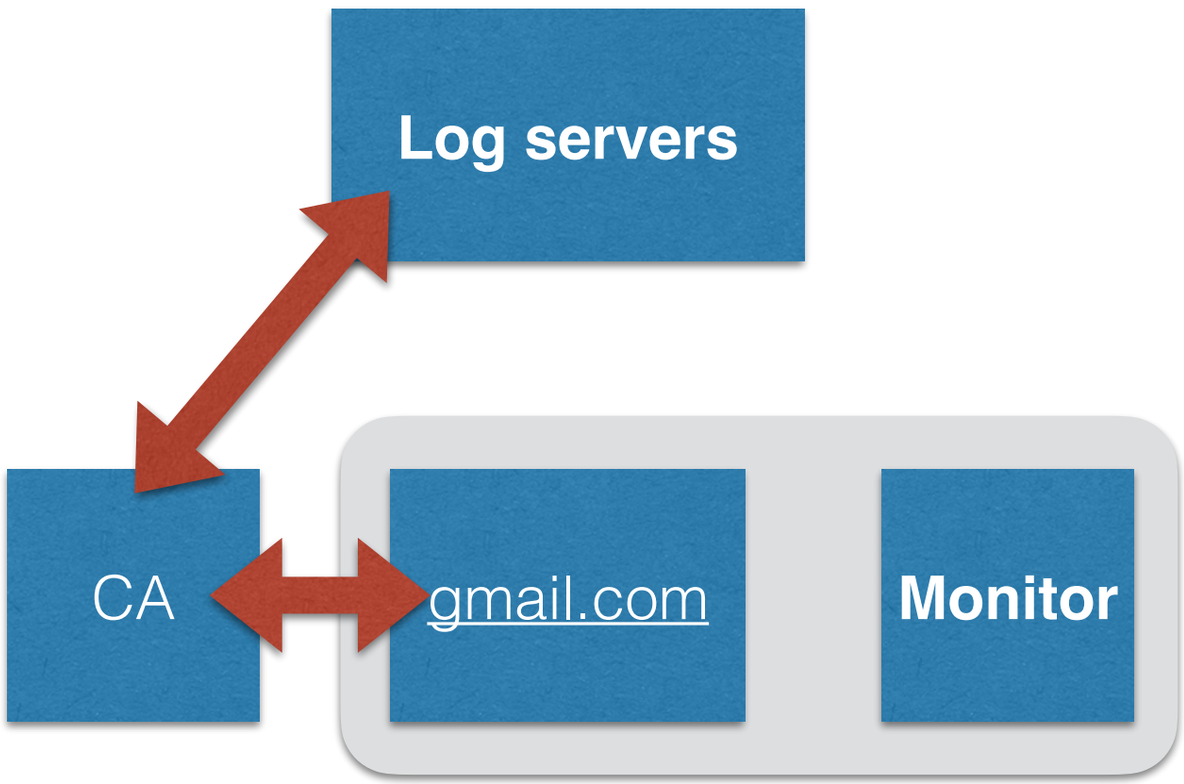
Wat do?

- **2016, WoSign and StartCom**: WoSign **backdated** SHA-1 certificates to bypass browser restrictions and acquired StartCom, which was also implicated in misissuance. Major browsers distrusted both CAs.
- **2017, Symantec**: Issued numerous **improperly validated certificates** over several years, leading to Google and Mozilla distrusting Symantec's certificates.
- **2019, Trustico**: Requested the mass revocation of 23,000 certificates **without proper justification**, raising concerns about their practices.
- **2020, Let's Encrypt**: Revoked over 3 million certificates due to a **bug** in their validation software that affected certificate issuance.
- **2021, Sectigo**: Revoked over 35,000 certificates after discovering a **bug** in their validation process that allowed improper issuance.
- **2022, Google Trust Services**: Briefly misissued certificates due to a **configuration error**, though the impact was limited and quickly resolved.

CA Accountability

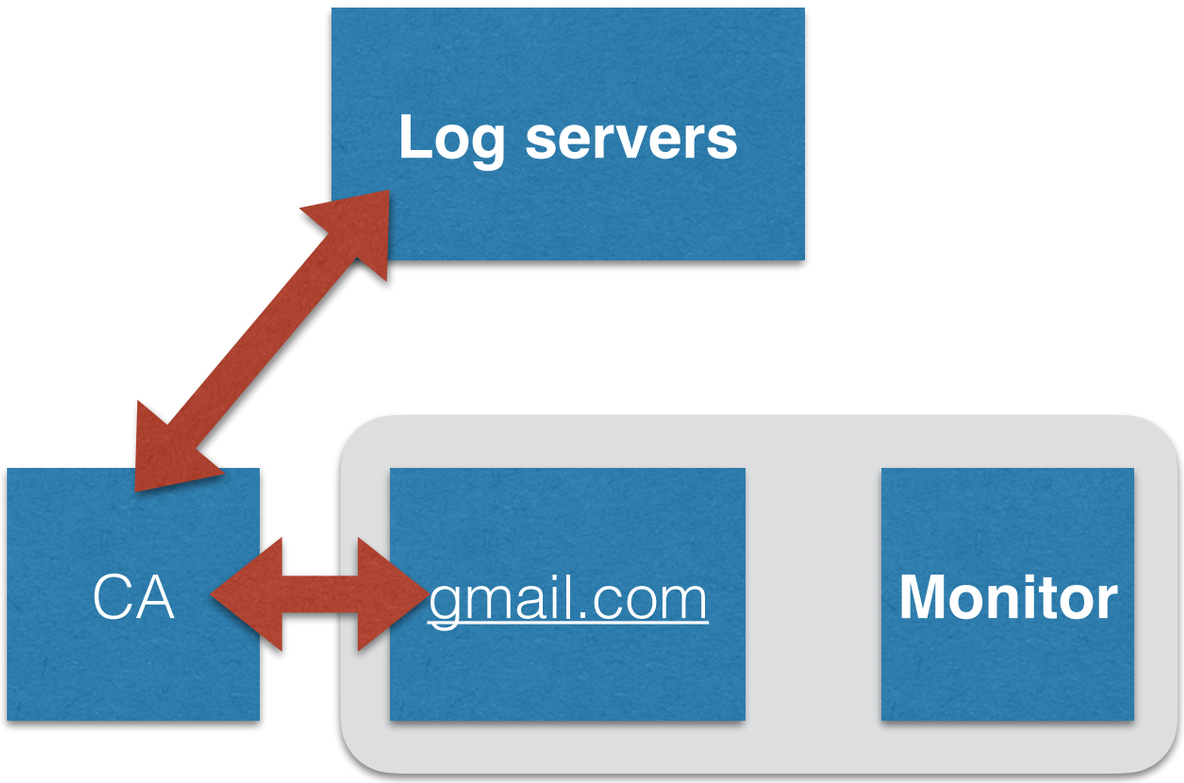
- When prevention is impossible, the next best thing is accountability after that fact
- More precisely: detect mis-issuance and remove offending CAs from trusted roots.
- Some challenges:
  - Only domain owners can detect mis-issuance
  - Attackers may target a small set of users.
- Solution: Users should only accept a cert if it will be seen by **all domain owners**.

Certificate Transparency Logs



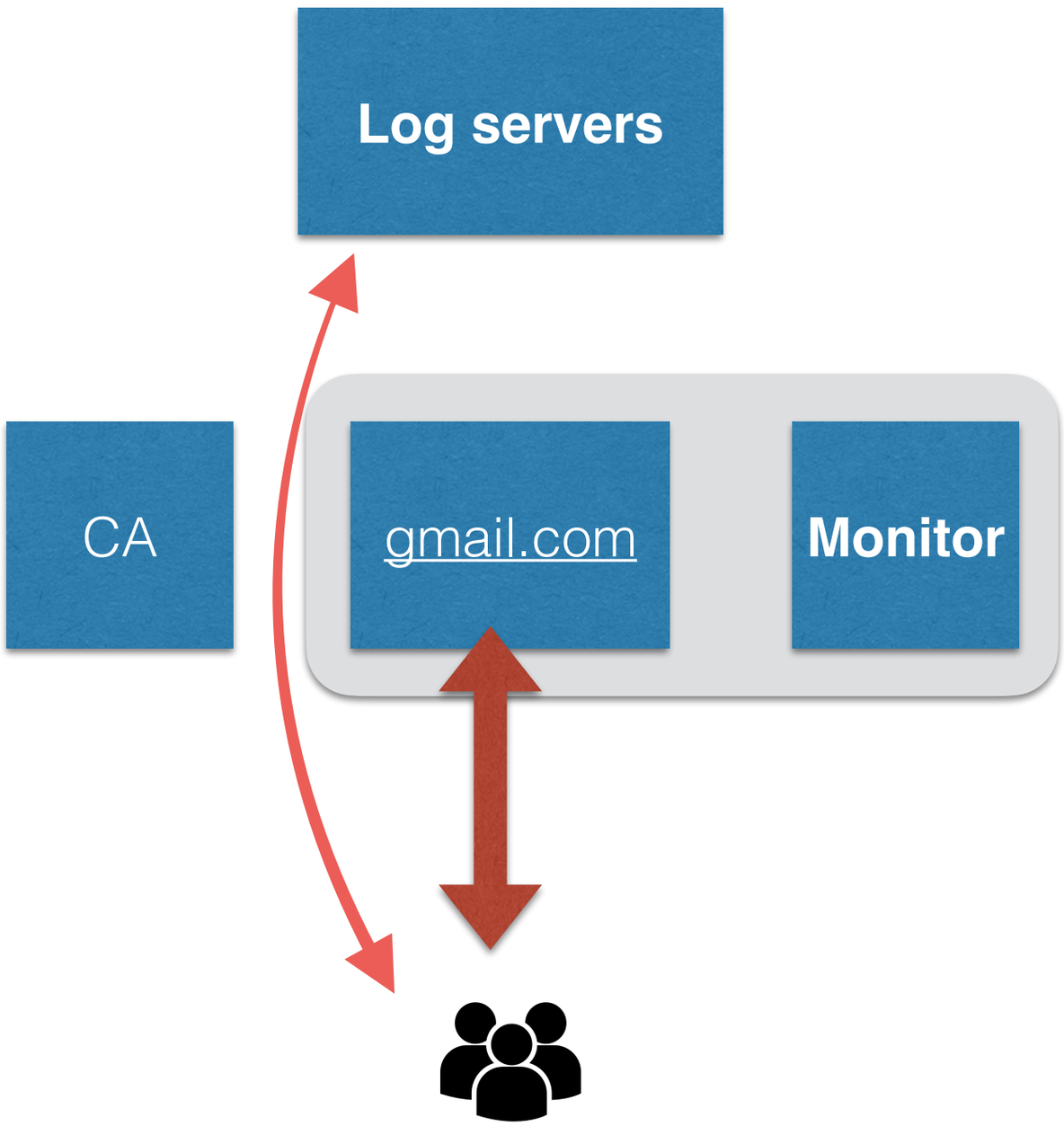
- Idea: build a **public log** to store the world's certificates.
- Running example throughout: gmail.com, CA, **CT log server**, browser, **gmail's monitor**
  - new players in bold
- Adversary model: Don't want to trust Log servers

Workflow: Issuance



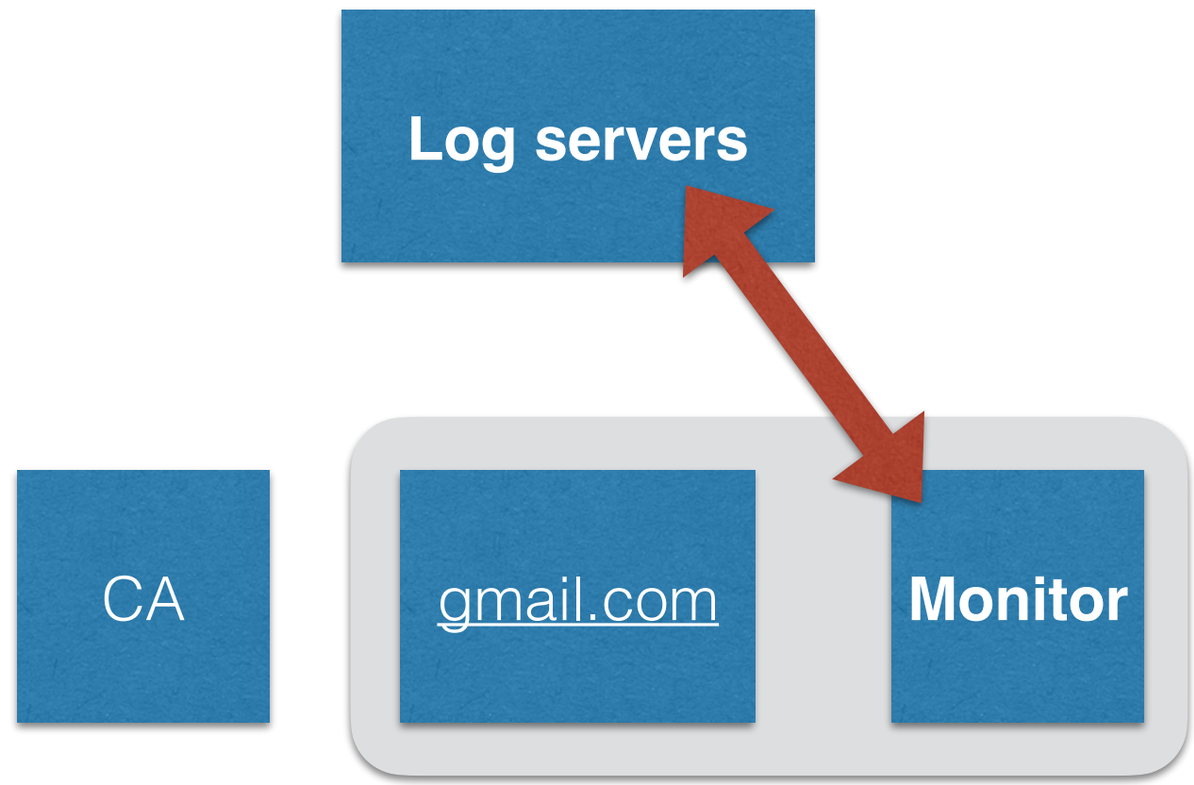
- gmail.com asks CA for a certificate
- CA issues cert & registers it with CT log servers (typically more than one)

Workflow: User/Browser



- Browser connects to gmail.com
- gmail.com provides cert during TLS handshake
- Browser checks* if cert is in the log. If not, abort.
  - Better not leave browsing history to Log server.

Workflow: Monitoring



- A monitor periodically scans the log for all certs issued for“gmail.com”, complains if there are bogus ones.
- Monitor is ideally run by gmail.com but gmail.com's CA may run it too.

Security goal: all bogus certs will be detected eventually.

Concerns

- Security concerns: Malicious log servers may collude with CAs to attack targeted users.
  - Deletion: log a bogus cert, fool the user, then delete it before the monitor finds out
  - Equivocation: log a bogus cert, but present conflicting views to users and monitors.
- Privacy concerns: user querying logs for every cert leaks browsing history…

The primitive needed

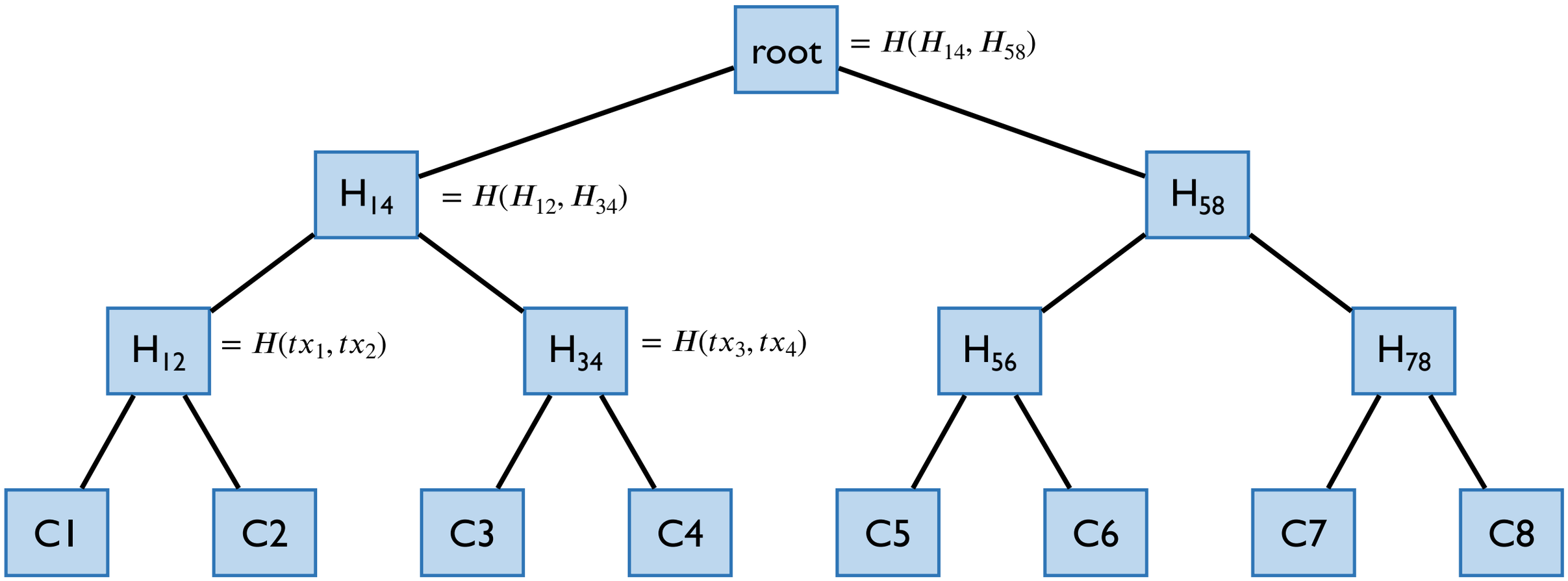
- Append-only and fork-resistant dictionary that
- **Succinct proofs of append-only**: malicious log server can't delete an item from the log
- **Fork-resistant**: if two users compare notes, they can identify forking.
- **Dictionary**: Support efficient (sub-linear) lookup for *all values* belong to a key.

Transparency Logs via Append-Only Authenticated Dictionaries. Tomescu et al. ACM 2019.

A suboptimal, but widely used solution: Append-only Merkle Trees

Merkle Tree

(Due to Ralpha Merkle in 1979)

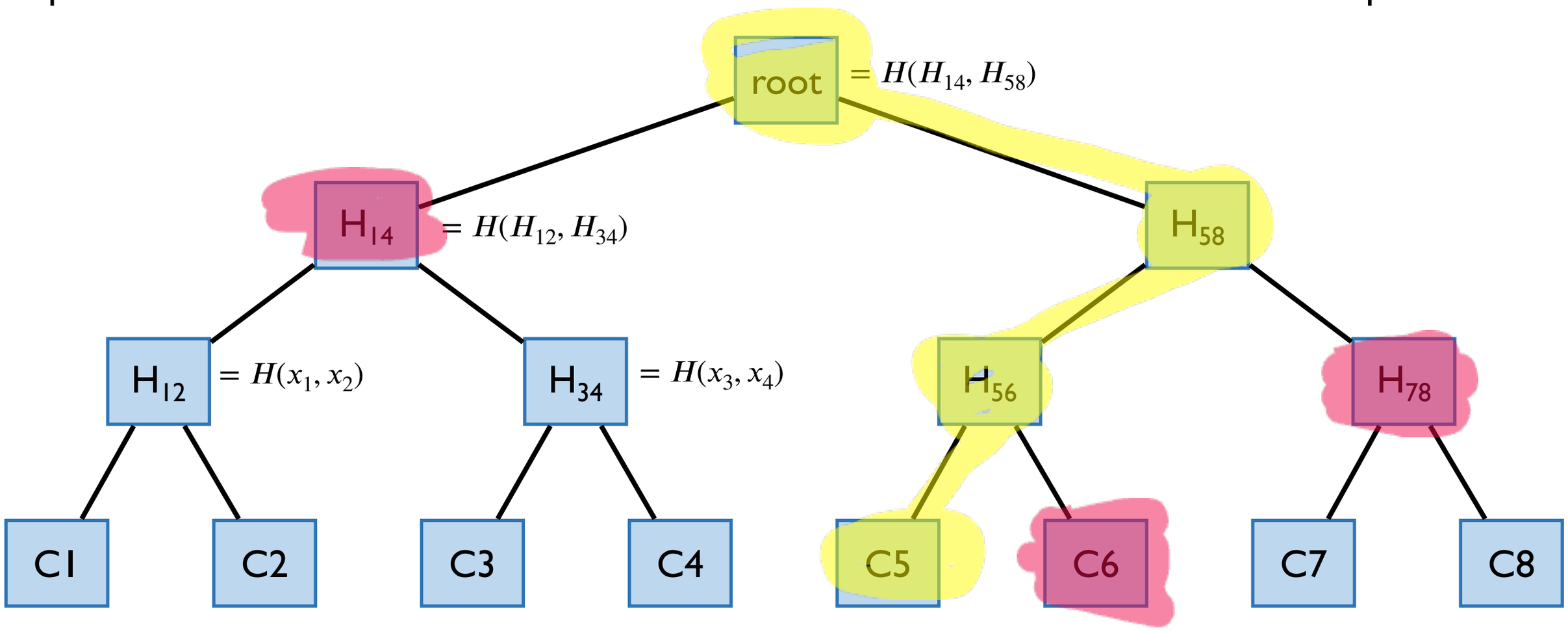


The **root** is a digest of all **leaves**.

Membership proof

Definition (Merkle Path of a leaf): the path from the leaf to the root.

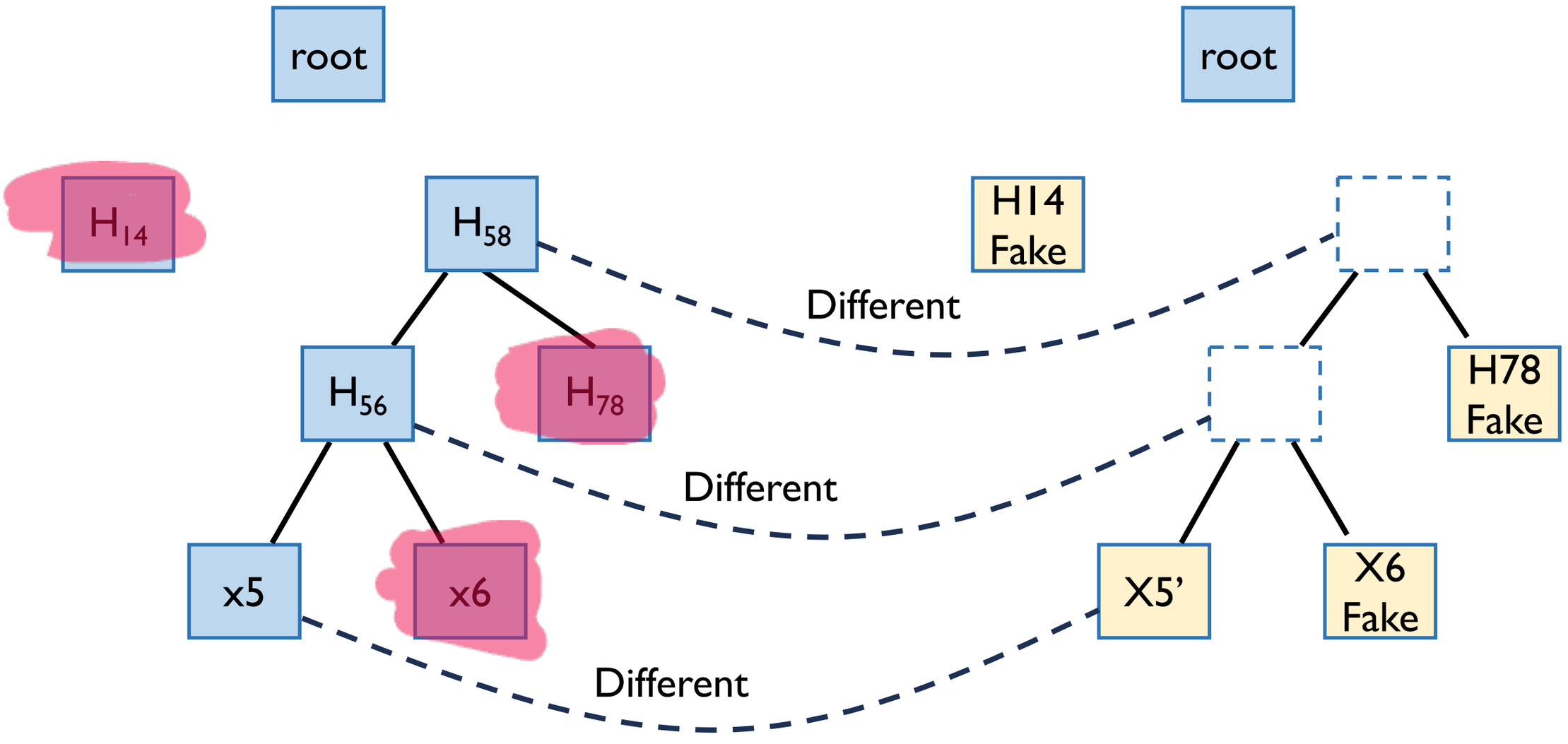
Membership proof of a leaf: siblings of the nodes on its Merkle path.



Proof that 5th leaf is C_5: $\pi = (C_6, H_{78}, H_{14})$

Verification algorithm: $H(\pi_3, H(H(C_5, \pi_1), \pi_2)) \stackrel{?}{=} \text{root}$

Security of membership proofs: faking one requires finding *collisions* somewhere along the path

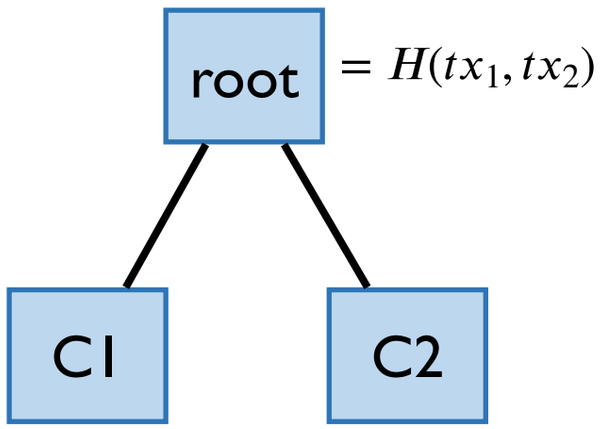


One way to think about it: a root uniquely defines a tree.

MT based CT logs

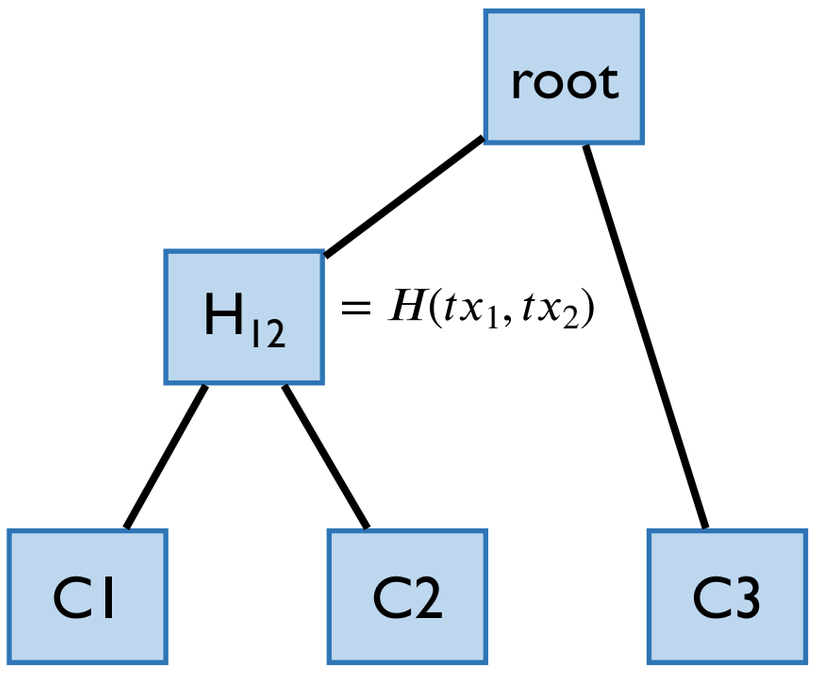
- Specified in IETF RFC 6962 and RFC 9162
- Most deployed logs and clients still use the RFC 6962 design
- Current usable logs

MT based CT logs



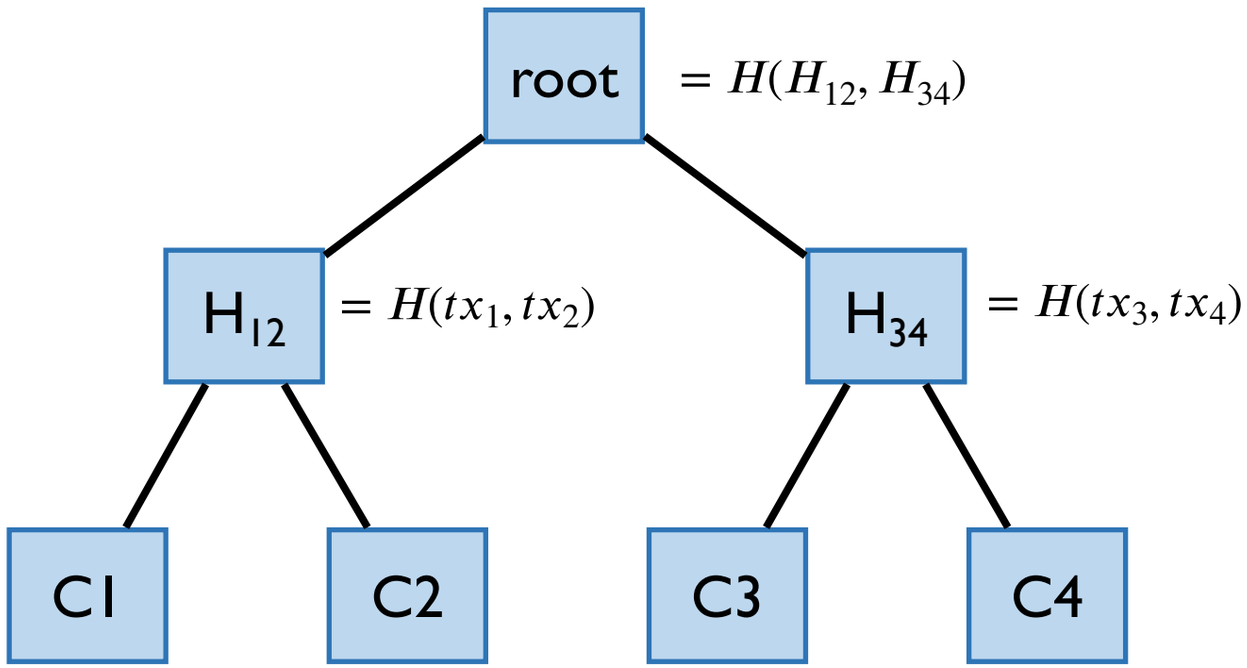
Leaves are (hashes of) **chronologically** ordered certs.

MT based CT logs



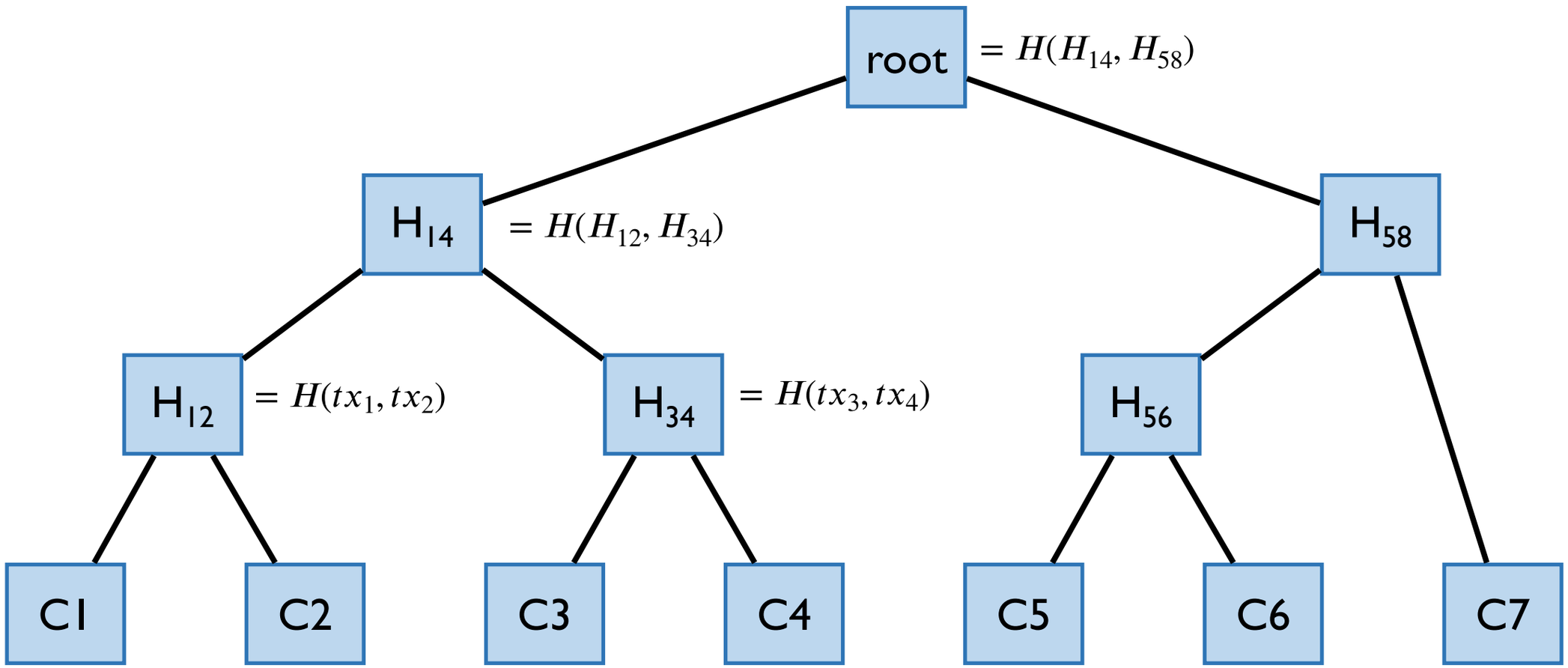
Leaves are (hashes of) **chronologically** ordered certs.

MT based CT logs



Leaves are (hashes of) **chronologically** ordered certs.

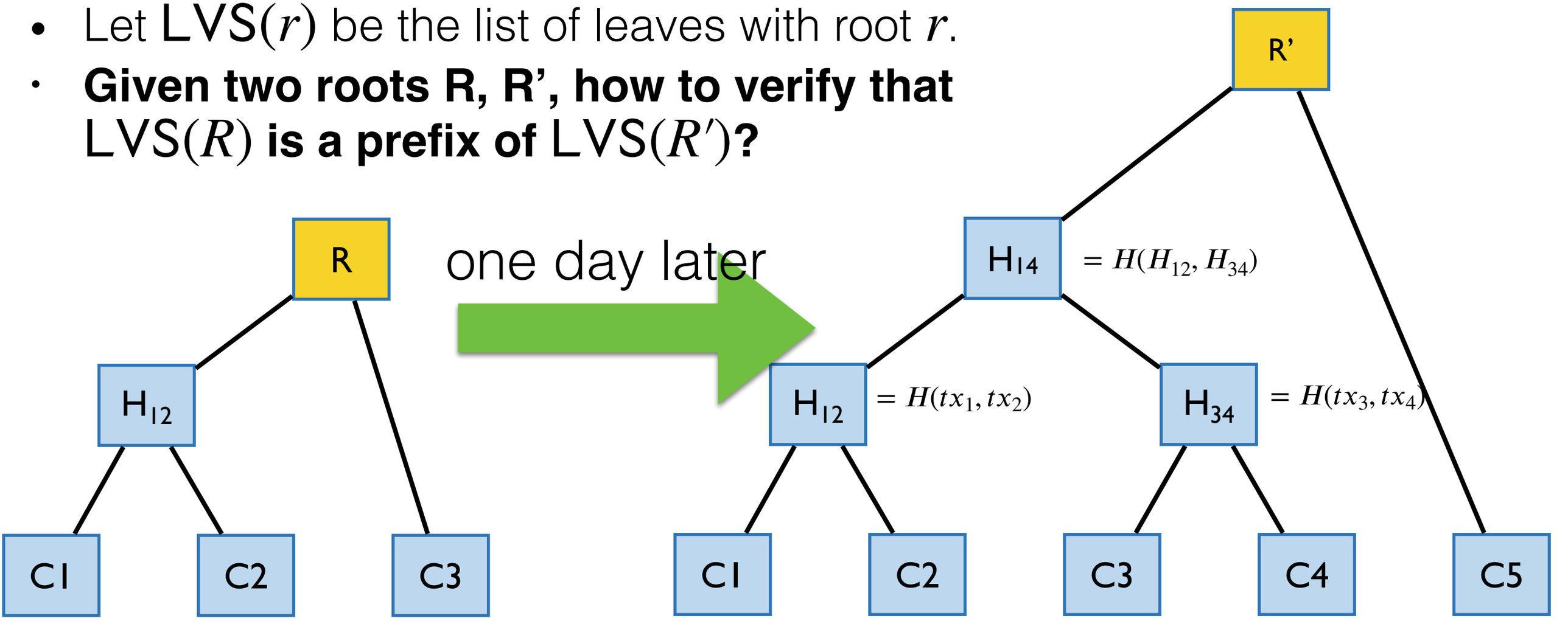
Merkle Tree



Leaves are (hashes of) **chronologically** ordered certs.

Append-only proof

- Let $\mathsf{LVS}(r)$ be the list of leaves with root r.
- **Given two roots R, R', how to verify that** $\mathsf{LVS}(R)$ **is a prefix of** $\mathsf{LVS}(R')$**?**

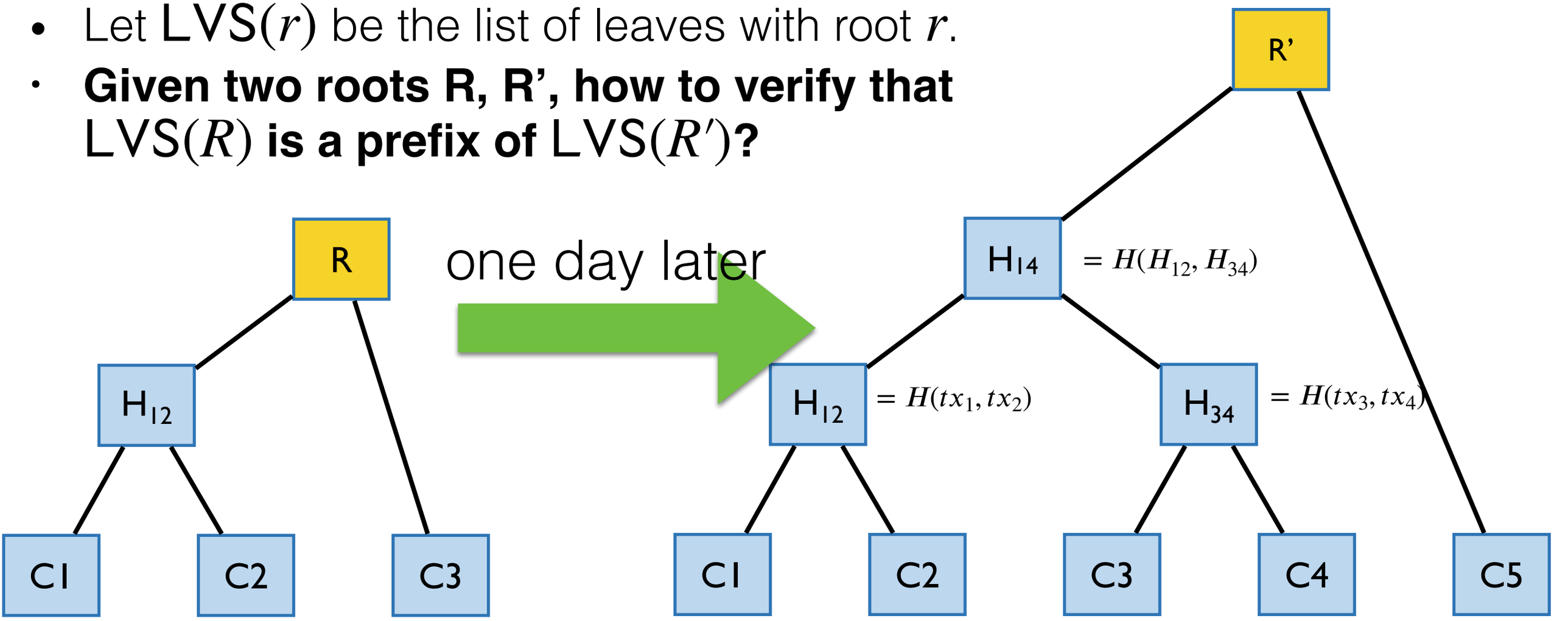


Application in CT: incremental monitor.

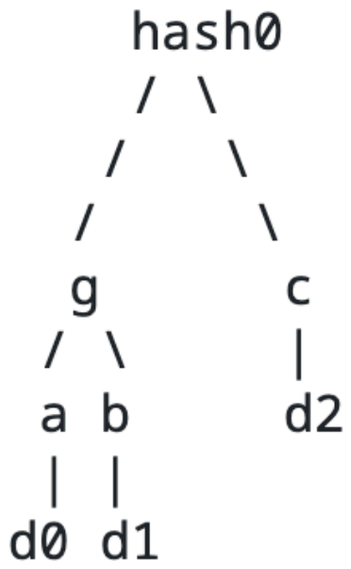
~6-8 million certs per day [Li *et al.*, CCS'19]

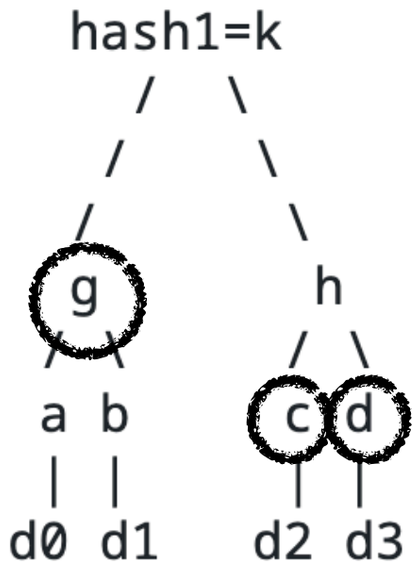
Append-only proof

- Let $\mathsf{LVS}(r)$ be the list of leaves with root r.
- **Given two roots R, R', how to verify that** $\mathsf{LVS}(R)$ **is a prefix of** $\mathsf{LVS}(R')$**?**



Append-only proof = **membership proof for** $\mathsf{LVS}(R)$ **in both trees**

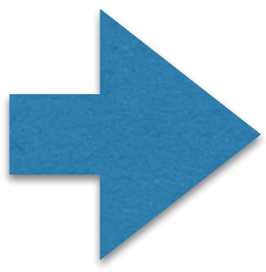




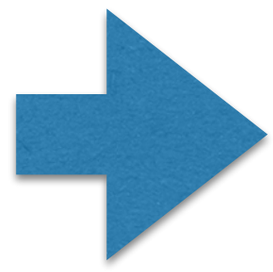
Slight notation change: leaves d0..dn are explicitly shown.

Append-only proof for (hash0, hash1, 3, 4)?

d0: a, c, h
d1: b, c, h
d2: g, d

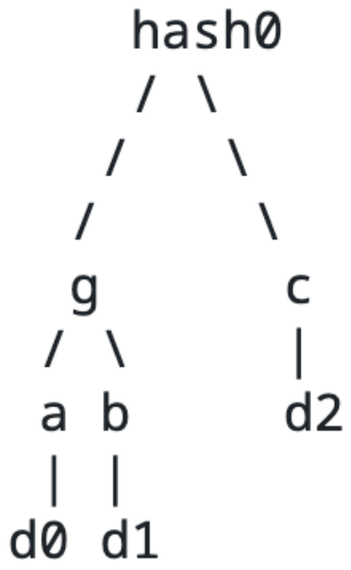


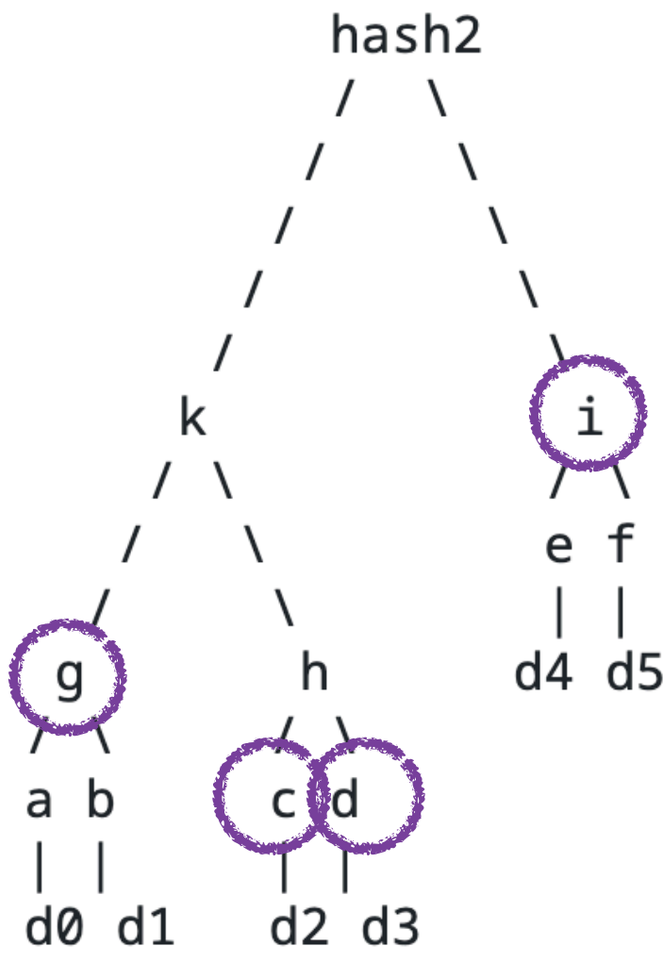
a, b, g, c, d, h



g, c, d

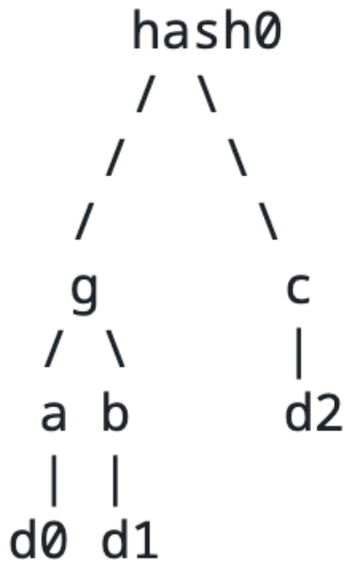
- Verify g, c is in hash0
  - That means g, c commits to $(d_0, d_1), d_2$
- Compute root' from g, c, d
  - Combined membership proofs for g, c
  - That means $(d_0, d_1), d_2$ are still in hash1



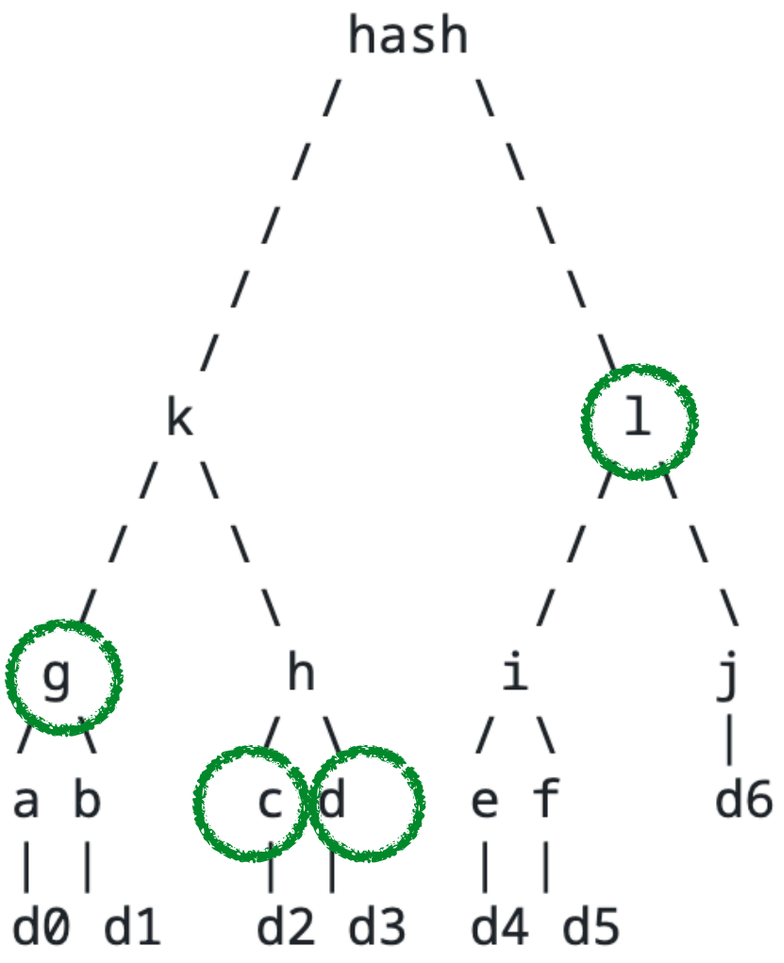


Append-only proof for (hash0, hash2, 3, 6)?

- Verify g, c in hash0
  - That means g and c commit to $(d_0, d_1), d_2$
- Compute hash2 from g, c, d, i
  - That means d0, d1, d2 are in leaves 1-3



Append-only proof
(hash0, hash, 3, 7)?



MT based CT logs

Log servers

CA

gmail.com

Monitor

- Log server maintains a Merkle Tree
- Return **signed root (Signed Tree Head, STH)** upon request
- Monitor periodically queries for new root (STH), new certs, and *append-only proofs since* previous root.

MT based CT logs

Log servers

CA

gmail.com

Monitor

- How do browser uses the log?
- Simply querying for membership proofs doesn't work for privacy reasons.
- Come back to this.

Signed Certificate Timestamp (SCT)

- Practical challenge: Adding certs to log takes time (e.g., it's batch done every 24 hours)
- log server returns a *promise* for inclusion called Signed Certificate Timestamp (SCT)
  - Typical maximum merge delay (MMD) is typically 24 hours
- Cert issuance with SCT:
  - gmail.com asks CA for a certificate
  - CA creates cert for gmail.com
  - CA registers cert with CT log server (typically more than one)
  - **CT log servers return SCTs**
  - **CA adds SCTs in the cert**
- Browser checks SCTs in lieu of inclusion proofs (improving privacy)

*.google.com

Subject Name
- Common Name *.google.com

Issuer Name
- Country or Region US
- Organization Google Trust Services
- Common Name WR2

Serial Number 43 63 E8 60 14 34 15 69 12 FA 36 D1 41 37 F2 67
Version 3
Signature Algorithm SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
- Parameters None

Not Valid Before Monday, January 20, 2025 at 3:36:04AM Eastern Standard Time
Not Valid After Monday, April 14, 2025 at 4:36:03AM Eastern Daylight Time

Public Key Info
- Algorithm Elliptic Curve Public Key (1.2.840.10045.2.1)
- Parameters Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
- Public Key 65 bytes : 04 6E 57 6B 72 79 7E 23 ...
- Key Size 256 bits
- Key Usage Encrypt, Verify, Derive

Signature 256 bytes : 8E CD FF AA A9 7F BB F6 ...

Extension Key Usage (2.5.29.15)
- Critical YES
- Usage Digital Signature

Extension Basic Constraints (2.5.29.19)
- Critical YES
- Certificate Authority NO

Extension Extended Key Usage (2.5.29.37)
- Critical NO

MIIJQiBxr2Qm2XGG2i2mG22HHWmWW2WG2mm3GW

- DNS Name *.gvt1.com
- DNS Name *.gcpcdn.gvt1.com
- DNS Name *.gvt2.com
- DNS Name *.gcp.gvt2.com
- DNS Name *.url.google.com
- ...

Extension Certificate Policies (2.5.29.32)
- Critical NO
- Policy ID #1 (2.23.140.1.2.1)

Extension CRL Distribution Points (2.5.29.31)
- Critical NO
- URI http://c.pki.goog/wr2/oBFYYahzgVI.crl

Extension Embedded Signed Certificate Timestamp List (1.3.6.1.4.1.11129.2.4.2)
- Critical NO
- SCT Version 1
- Log Operator Google
- Log Key ID 4E 75 A3 27 5C 9A 10 C3 38 5B 6C D4 DF 3F 52 EB 1D F0 E0 8E 1B 8D 69 C0 B1 FA 64 B1 62 9A 39 DF
- Timestamp Monday, January 20, 2025 at 4:36:07AM Eastern Standard Time
- Signature Algorithm SHA-256 ECDSA
- Signature 71 bytes : 30 45 02 20 23 89 22 91 ...
- SCT Version 1
- Log Operator DigiCert
- Log Key ID 73 20 22 0F 08 16 8A F9 F3 C4 A6 8B 0A B2 6A 9A 4A 00 EE F5 77 85 8A 08 4D 05 00 D4 A5 42 44 59
- Timestamp Monday, January 20, 2025 at 4:36:07AM Eastern Standard Time
- Signature Algorithm SHA-256 ECDSA
- Signature 71 bytes : 30 45 02 21 00 B0 4E 62 ...

Extension Certificate Authority Information Access (1.3.6.1.5.5.7.1.1)
- Critical NO
- Method #1 Online Certificate Status Protocol (1.3.6.1.5.5.7.48.1)
  - URI http://o.pki.goog/wr2
- Method #2 CA Issuers (1.3.6.1.5.5.7.48.2)
  - URI http://i.pki.goog/wr2.crt

Fingerprints
- SHA-256 95 6A BF E9 3A B7 AB 83 26 FB CA 16 97 99 EB C9 1A 5B 39 9B F4 28

New privacy problems

- It's crucial to audit that promises in SCT are upheld by log servers.
- Who can audit? Only **browsers** see all SCTs.
- Several options, none perfect
  - Directly query the log: leak browsing history
  - Route query through proxy/mixnet
  - Route query through DNS (who already knows client's browsing history)
  - Use of Private Information Retrieval

State of the Art : Auditing by Browsers

(some)
proxying ++

Safe Browsing API
Proxying

No Auditing

- Still an open problem (See “SoK: SCT Auditing in CT” by Meiklejohn *et al.*)

Slide taken from “SCT Auditing: Revisited” (https://transparency.dev/pdfs/summit2024/heimberger-sct.pdf)

Auditing other aspects

- Third-party auditors can periodically query for STHs and verify append-onlyness.
- Large organizations such as Google or independent watchdogs act as auditors.

Append-only proofs *can* prevent forks

- Locally verifying append-only proofs does not prevents *targeted attacks*
  - e.g., log server presents one fork for user and another for monitor [draw on whiteboard]
- **Equivocation detection**: Browsers and monitors gossip to compare STHs
  - Any pair of STHs from a given log server should be **append-only**.

Real-World Adoption of CT

- A list of logs https://ct.cloudflare.com/logs

Browser	Current SCT requirements	Current OCSP/TLS extension requirements
Chrome/Chromium	• One SCT from a currently approved log • Duration ≤ 180 days: 2 SCTs from once-approved logs • Duration > 180 days: 3 SCTs from once-approved logs[9][10]	• 1 SCT from a current Google log • 1 SCT from a current non-Google log
Firefox	• desktop: 2 SCTs from once-approved logs, since v135[11][12] (released 2025-02-04) • Firefox for Android: 2 SCTs from once-approved logs, since v145[11] (released 2025-11-11)	Two SCTs from currently approved logs
Safari	• One SCT from a currently approved log • Duration ≤ 180 days: 2 SCTs from once-approved logs • Duration > 180 days: 3 SCTs from once-approved logs[13]	Two SCTs from currently approved logs

https://en.wikipedia.org/wiki/Certificate_Transparency

Look up certs

crt.sh Identity Search

Group by Issuer

Criteria	Type: Identity Match: ILIKE Search: 'fanzhang.me'

Certificates

crt.sh ID	Logged At	Not Before	Not After	Common Name	Matching Identities	Issuer Name
16559503878	2025-02-06	2025-02-06	2025-05-07	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E5
16003305301	2024-12-08	2024-12-08	2025-03-08	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E6
15660611998	2024-12-08	2024-12-08	2025-03-08	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E6
14856369908	2024-10-09	2024-10-09	2025-01-07	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E5
14856356953	2024-10-09	2024-10-09	2025-01-07	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E5
14140384323	2024-08-10	2024-08-10	2024-11-08	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E5
14071183639	2024-08-10	2024-08-10	2024-11-08	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E5
13349806863	2024-06-11	2024-06-11	2024-09-09	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E6
13349801200	2024-06-11	2024-06-11	2024-09-09	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=E6
12695748442	2024-04-12	2024-04-12	2024-07-11	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=R3
12695740250	2024-04-12	2024-04-12	2024-07-11	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=R3
12052441841	2024-02-12	2024-02-12	2024-05-12	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=R3
12052444873	2024-02-12	2024-02-12	2024-05-12	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=R3
11444115896	2023-12-14	2023-12-14	2024-03-13	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=R3
11426526208	2023-12-14	2023-12-14	2024-03-13	*.fanzhang.me	*.fanzhang.me fanzhang.me	C=US, O=Let's Encrypt, CN=R3

- A convenient tool https://crt.sh

Research Directions

- **Improving log data structure**: Merkle Tree can prove set membership, but doesn't support range queries or non-membership proofs:
  - CONIKS (Melara et al, USENIX Security '15)
  - Append-Only Authenticated Dictionaries (CCS'19)
  - Merkle2: A Low-Latency Transparency Log System (S&P'21)
  - SEEMless: Secure End-to-End Encrypted Messaging with less Trust (CCS'19)
  - Parakeet: Practical Key Transparency for End-to-End Encrypted Messaging (NDSS'23)
  - ELEKTRA: Efficient Lightweight multi-dEvice Key TRAnsparency (CCS'23)
- **Better SCT audibility**

Summary of CT

- Motivation: CA misbehaviors are hard to eliminate
- Goal: When prevention is impossible, accountability is usually the next option
- Idea: put all certs in multiple publicly accessible, append-only, equivocation-free logs. Browser only trusts certs in the logs.
- Technical tools: Merkle tree, inclusion proofs, append-only proofs
- Widely used in practice, though there are still open challenges (e.g., privacy-preserving SCT auditing)