chrome

Merkle Tree Certificates

transparency as the root of trust for the PQ age

Joe DeBlasio
jdeblasio@chromium.org

Transparency.dev
Summit 2025

Oct 2025

And you are...?

Engineering Manager on **Chrome's Network Security team**

Think CT, HSTS, HTTPS-by-default, revocation, other WebPKI.

I'm presenting the work of many others, some of whom are here!

David Benjamin
Devon O'Brien
Bas Westerbaan
Luke Valenta
Filippo Valsorda
Carlos Joan Rafael Ibarra Lopez
Christopher Patton
Matt Mueller
Mustafa Emre Acer
Nick Harper

(With more to come!)

oh look it's me

1. **What, and why, are Merkle Tree Certificates**
2. **Standardization**
3. **Our upcoming experiment**
4. **Future policy possibilities**

Post-quantum algorithms are too large

RSA-2048 yesterday	256-byte public keys	256-byte signatures
P-256 today	64-byte public keys	64-byte signatures
ML-DSA-44 tomorrow	1,312-byte public keys	2,420-byte signature

Major increase in sizes!

TLS handshakes use 3+ signatures and 1+ public key.

CT log entries get a *lot* bigger

Precertificate

X.509 overhead

EE public key

CA signature

Cert with embedded SCTs

X.509 overhead

EE public key

CA signature

2 log signatures in embedded SCTs

Naive move to ML-DSA-44 yields **8.3x** increase

MTCs: inclusion in a CA's log *is* the proof of CA issuance

Today's CAs first **signs**, *then* **logs** the result, collects **SCTs**

- Result of CT layered on top of existing PKI

A Merkle Tree CA first **logs**, *then* **signs** a log checkpoint and collects **cosignatures**

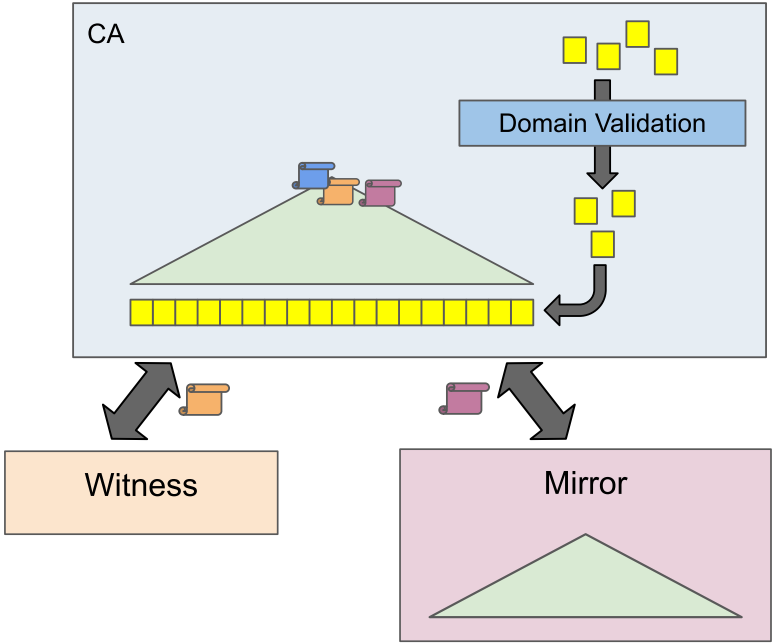
- CAs maintain log of issued certificates
- Log **mirrors** (i.e. copies) replace CT logs with distinct contents

Log first, *then* sign

CAs run *issuance logs*, containing log entries describing issued certificates

CA signs checkpoint of tree

CA pushes entries to get additional cosignatures from mirrors+witnesses



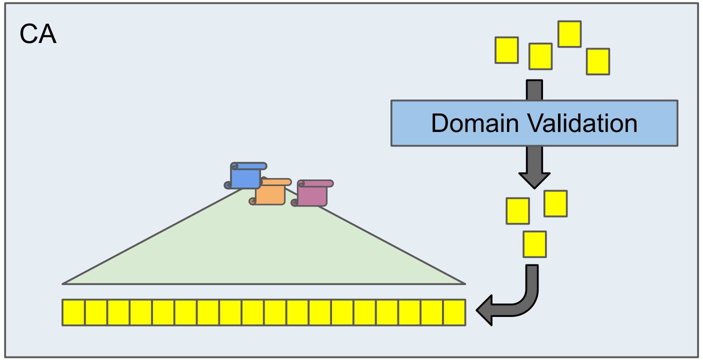
Certificates just prove entries are in the log

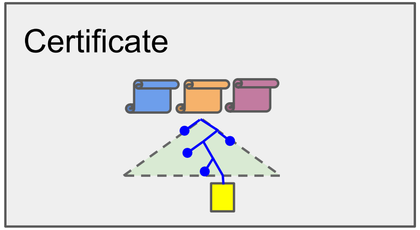
Log entry becomes (most of) the `TBSCertificate`

Log index becomes the serial number

X.509 signature is replaced with

Inclusion proof + cosignatures





Clients check inclusion proof and cosignatures against policy

- Cosignature from CA (trusted for validation)
- Cosignatures from witnesses/mirrors (akin to CT)

Why this design?

Smaller log entries

Fewer log entries

Optimized certificates

Smaller log entries

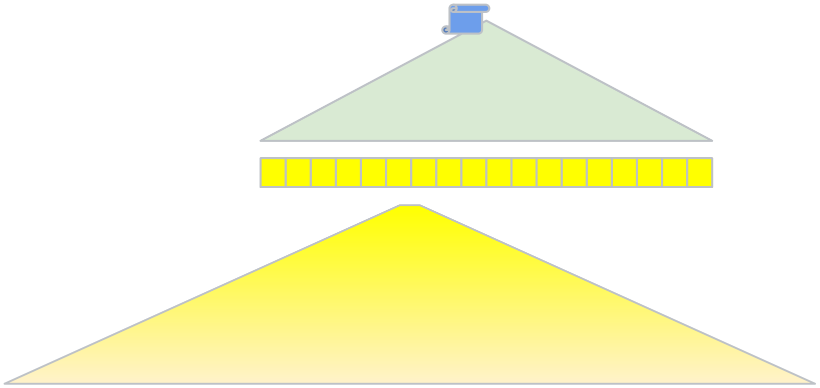
MTC log entries **remove all the expensive stuff**

- No per-entry signatures
- Replaces public keys with hashes

Possible because logging ***is*** issuance

- Signed checkpoint gives non-repudiation

11x entry size reduction today
100x PQ entry size reduction



```
TBSCertificateLogEntry  ::=  SEQUENCE  {
  version         [0]  EXPLICIT Version DEFAULT v1,
  issuer               Name,
  validity             Validity,
  subject              Name,
  subjectPublicKeyInfoHash OCTET STRING,
  issuerUniqueID  [1] IMPLICIT ... OPTIONAL,
  subjectUniqueID [2] IMPLICIT ... OPTIONAL,
  extensions      [3] EXPLICIT Extensions OPTIONAL
  -- no signature
}
```

Fewer log entries

Today, both precerts and final certificates are typically logged

- **In every log!** Only need 2, but usually cross-logged
- monitors download *each* certificate **up to 2N times**
- More logs ⇒ more copies ⇒ **higher monitor overhead**

With MTCs, each issuance creates **only one entry in one tree**

- Mirrors are provably identical ⇒ monitors **only download each entry once**
- More mirrors ⇒ same entry is more available ⇒ **lower serving overhead**

2x entry reduction per log
Up to 28x reduction across ecosystem

Optimized certificates

MTCs allow you to issue **two** types of certificates

Traditional certificates

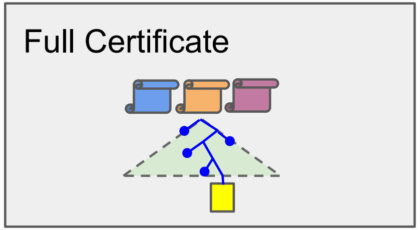
Require N signatures (from CA + mirrors)

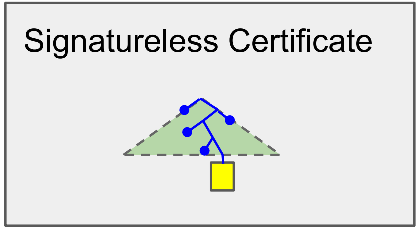
Status quo; large

Signatureless certificates

No signatures means much smaller TLS handshake

But requires up-to-date clients





Signatureless certificates

Key insight: inclusion can be proved to *any* future log checkpoint

1. Designate some periodic (e.g. hourly) checkpoints as "landmarks"
2. Browsers then receive validated hashes of landmarks out-of-band.
3. Server then just needs prove inclusion to these already-trusted hashes

BUT

- Can't make signatureless certificate immediately -- must wait until next landmark.
- Requires up-to-date clients.
- Servers and clients must agree on what landmarks to use.

Other neat properties

Index-based revocation
- Can revoke without seeing certificate details

Issuance log is *just a tiled transparency log*
- Reuse community’s operational expertise

X.509 construction is “just” a funny signature algorithm
- TLS server software generally ignores algorithm and signature

CA signatures are batched
- No big deal CA’s HSM is really slow

Lots more we're not talking about

Subtrees

Enable smaller inclusion proofs

Pruning

Mitigate log growth (and obviate log sharding) by dropping log entries once they're no longer relevant, while still maintaining Merkle tree properties.

And lots more. See the draft!

Standardization: PKI, Logs, And Tree Signatures (PLANTS)

IETF	plants@ietf.org https://mailman3.ietf.org/mailman3/lists/plants.ietf.org/ BoF at IETF 124 Montreal on November 4
GitHub	https://github.com/davidben/merkle-tree-certs

Experiment

Partnership between Cloudflare and Chrome

Conventional crypto (ECDSA/RSA certs; ECDSA cosigners)

Signatureless certificates only -- thus no 3p witnessing or mirroring.

Experiment

First wrinkle: Cloudflare isn't a public CA

- All MTCs must match existing conventional X.509 "bootstrap" certificate.
- Cloudflare will publish these certificates in parallel.

Chrome servers will...

- Fetch the log's landmarks.
- Validate consistency.
- Verify that...
  - each MTC is included in those landmarks
  - each MTC matches the corresponding bootstrap certificate
  - each bootstrap certificate would validate in Chrome

Experiment

Chrome servers will then...

Deliver validated landmark hashes to Chrome clients

Chrome clients will...

Indicate support for MTCs to servers by sending `trust_anchors` TLS extension.

If provided with an MTC, verify the entry's inclusion proof to the known landmark.

Work is happening *now*. Goal is to be authenticating real TLS connections in **December**.

Policy

Expecting MTCs to be PQ-only in Chrome

Hoping MTCs can be Chrome's *only* PQ root store

Lots of cosigner policy options

Mirror CT policy
Status quo
Require cosigs from two 3p mirrors

Let CAs count as a mirror
Require cosigs from one 3p mirror
Require CAs to publicly host their log

Require cosignatures from UA mirror(s)
Require one or two cosigs
No 3p mirror ecosystem

Require *Chrome's* cosignature
No 3p non-CA mirror ecosystem
Smallest certificates

Questions?

jdeblasio@chromium.org
https://github.com/davidben/merkle-tree-certs/
chrome-certificate-transparency@google.com