Certificate Transparency

The joys of discoverability

mhutchinson@google.com
al@google.com

Rewind: what’s the problem?

CAs are trusted to issue certs

The nature of cert usage means that bad certs can stay hidden

Not just a theoretical problem; misissued certs were found

How did CT solve it?

Public Log of all certs issued:

- CAs must submit all certs to logs
- Browsers won’t trust a cert unless it is in a Trusted Log(s)
- Verifiers inspect log contents to discover bad certs

How are certs issued?

When a CA receives a Certificate Signing Request:

1. Confirm identity of the domain owner
2. Create precert containing provided Public Key
3. CA performs dance with Transparency Logs:
    a. Precert is sent to the log
    b. Artifact that this certificate is logged is collected
4. Final certificate containing log artifacts is created
5. Final cert issued to the domain owner

What are verifiable logs?

Logs support *efficient* cryptographic proof of:

- Leaf inclusion
- Consistency
- Single-view

Accomplished using a *Merkle Tree*: hashing is the primary operation.

Proofs can be acquired online, or bundled for offline usage.

What is log transparency?

CT is a mechanism to ensure that all certificates are *discoverable*.

Discoverability means all stakeholders see the same list of issued certs:

- Browsers
- Domain owners
- Security researchers
- CAs

Another way of looking at it...

Certificates are effectively a claim:

"This public key was legitimately requested for the specified domain"

- Browsers - **Believer**
- Domain owners - **Verifier**
- Security researchers - **Verifier**
- CAs - **Claimant**, **Verifier**

https://github.com/google/trillian/tree/master/docs/claimantmodel

Does this apply outside of CT?

Discoverability:

- Any Claim a Believer trusts must eventually be verified by a Verifier

Other adopters of log transparency:

- Pixel 6 Binary Transparency
- Firmware Transparency
- sum.golang.org