Solution Space and Dispatched Work

PLANTS BoF[1] IETF 124

[1] Bowl of Fruit
[1] Bouquet of Flowers
[1] Bunch of Ferns

OK, post-quantum is too big. Now what?

*"Does the community think that the problem statement is clear, **well-scoped, solvable**, and useful to solve?"*

Key ideas from dispatched work

Why does this help?

Comparing to today's systems

PKI and transparency at large

Key ideas from dispatched work

draft-davidben-tls-merkle-tree-certs

Don’t log what you issue,

Issue by logging

Don’t log what you issue, issue by logging

A traditional CA first **signs**, *then* **logs** the result and collects **SCTs**

- CT was layered on top of existing PKI
- Complications stem from the gap between issuance and logging

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

- Each CA runs a separate *issuance log*
- Independent *mirrors* and *witnesses* co-sign log state

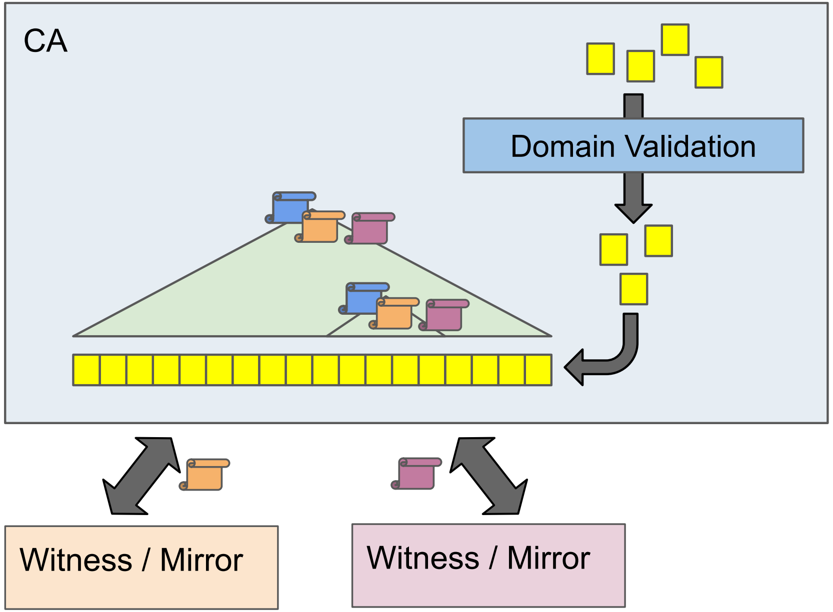
How does this work?

CAs run *issuance logs* describing what they've issued

Log entries describe certified claims

CAs sign checkpoints and subtrees to attest state of tree

CAs collect cosignatures from witnesses and mirrors to ensure correct operation



Certificates just prove entries are in the log

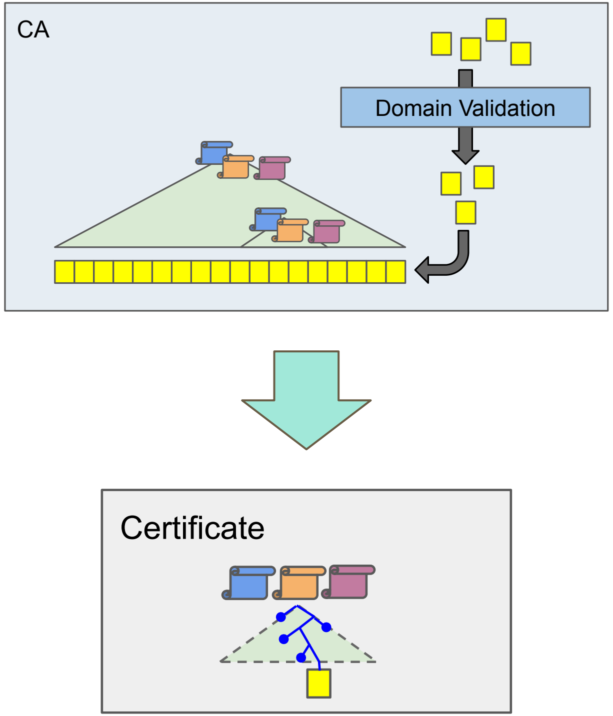
Log entry becomes the TBSCertificate

Log index becomes the serial number

Inclusion proof and cosignatures become the X.509 signature value

Clients check inclusion proof and cosignatures

- One cosignature from CA (validation)
- Enough witness/mirror cosignatures (akin to CT)



Why does this help?

Pruning via tree signatures

Smaller log entries

Fewer log entries

Optimized certificates

Smaller log entries

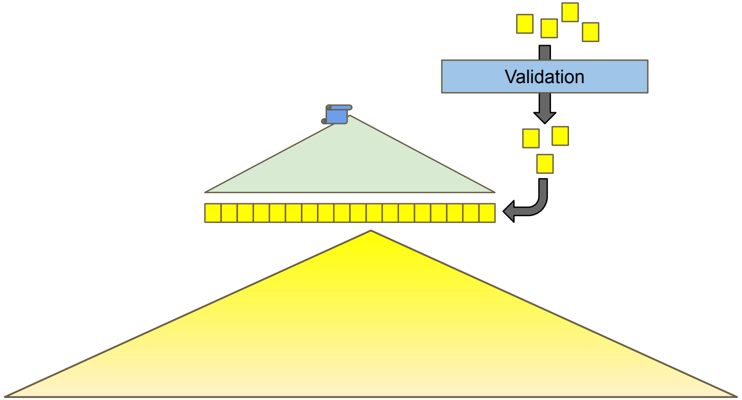
Log entries **do not pay post-quantum tax**

No per-entry signatures

Store hash of public key in log

Possible because logging *is* issuance

Latest checkpoint signature gives non-repudiation




```
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
}
```

Naive move to ML-DSA-44: **8.3x larger** entries

Compared to *CT today*, MTC log entries are **12x smaller**

Compared to naive ML-DSA-44 transition, MTC log entries are **97x smaller**

Fewer log entries

Both precertificates and final certificates are logged, in *every* log

- Only two logs needed, but cross-logged after-the-fact
- Monitors download each certificate **up to 2N times**
- More logs ⇒ more copies of the certificate ⇒ **higher monitor overhead**

Now, each issuance creates **only one entry in one tree**

- Mirrors provably identical ⇒ monitors **only download each entry once**
- More mirrors ⇒ same entry is more available ⇒ **lower serving overhead**
- No DoS problems from issuance/logging gap and external logging endpoint

2x entry count reduction per certificate per log

Up to **20x** entry total count reduction (counted 10 logs for 2025h2)

Optimized certificates

Any future checkpoint can construct a certificate for a log entry

- Designate periodic (e.g. hourly) checkpoints as *landmarks*
- Clients receive validated landmark hashes out-of-band
- Servers only need to prove inclusion to pre-distributed landmarks

Caveats

- Next landmark may not be allocated for a while
- Clients may be out-of-date

Make *two* certificates from one log entry: full and *signatureless*

Negotiate signatureless certificate when possible, full certificate otherwise

3 ML-DSA signatures: **7,260 bytes** (1 CA sig, 2 SCTs)
1 inclusion proof: **736 bytes** (estimated from Web PKI volumes)

Comparing to today’s systems

Rearranging the garden

Analogous PKI roles as today

Builds on existing CT work

“Just” a funny signature

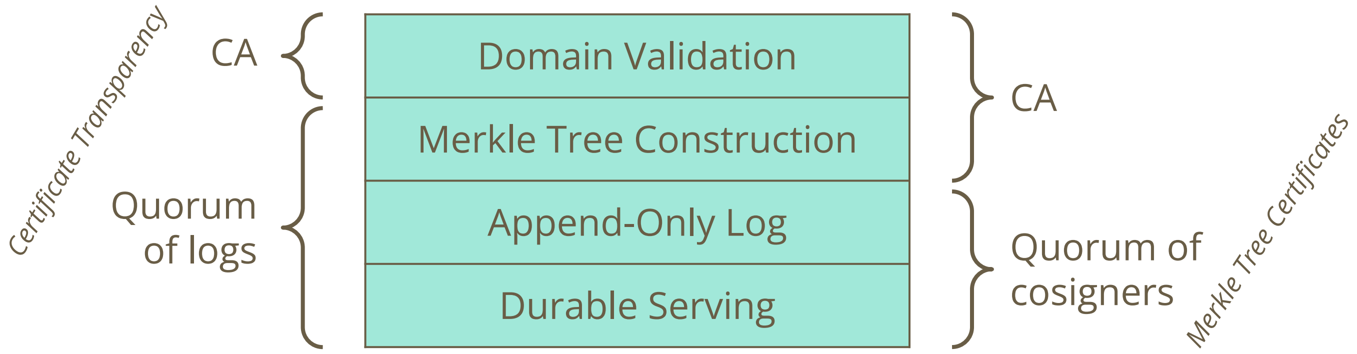
Certificate negotiation in TLS

View from certificate ecosystem

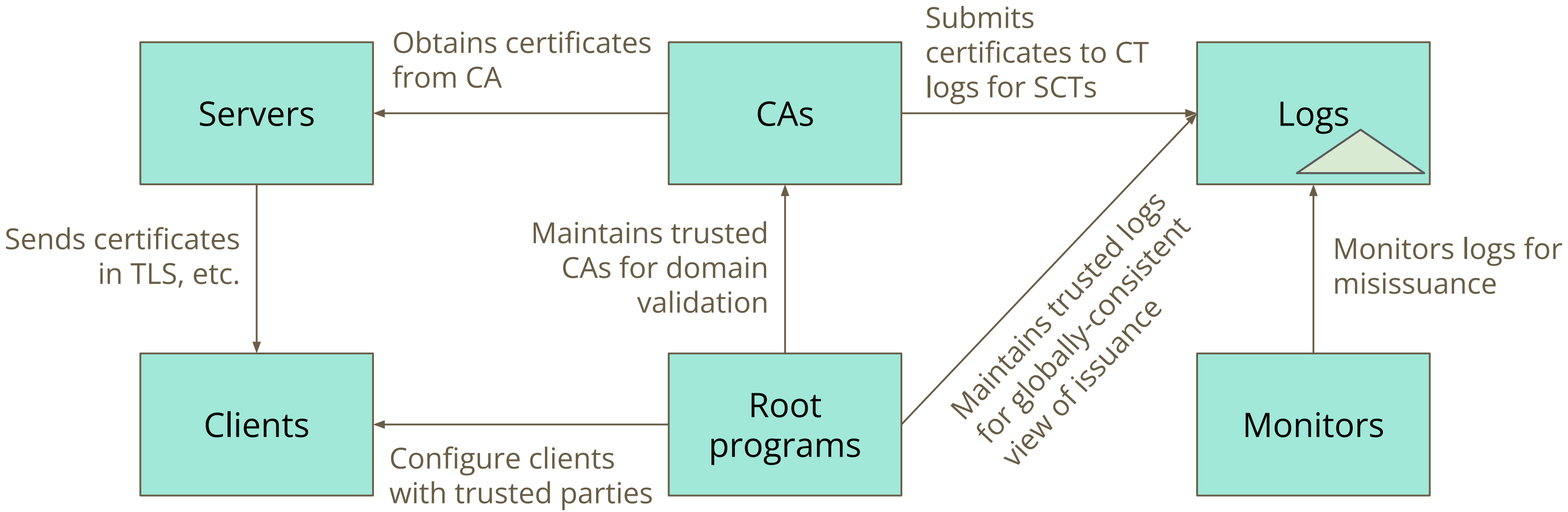
Traditional CAs ⇒ MTC CAs

CT Logs ⇒ Cosigners for issuance log

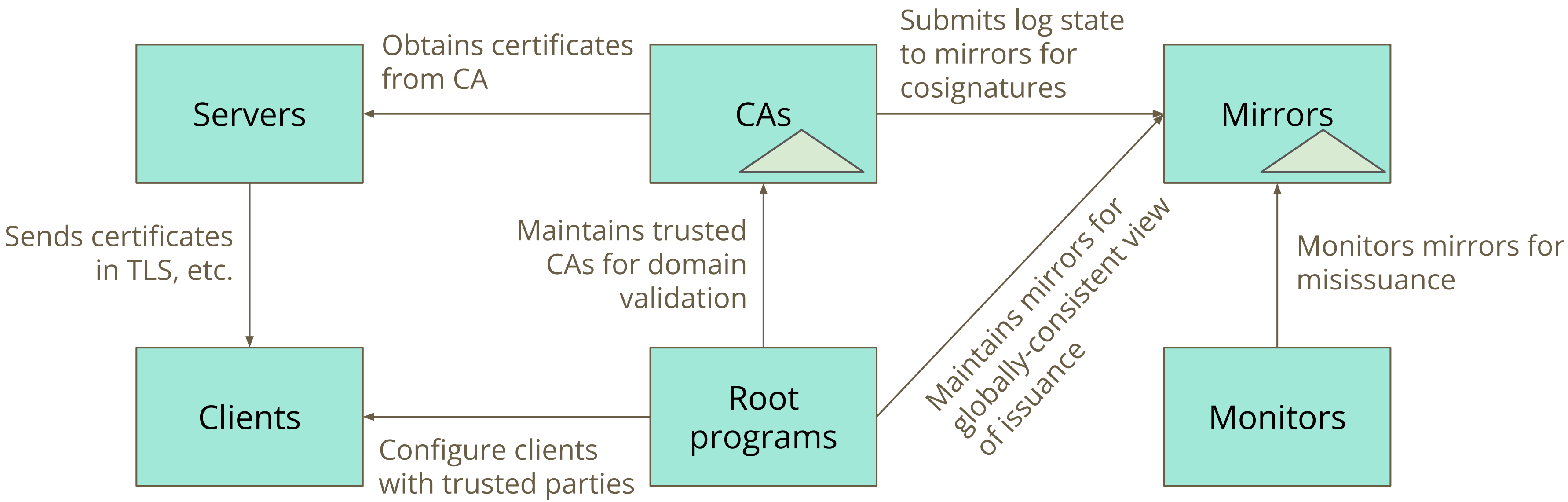
Similar responsibilities, security assumptions, communication flows



PKI roles today



PKI roles in this design



Building on existing CT work: issuance logs

Same tree structure from RFC 6962 / RFC 9162

Apply operational experience from CT community

Can reuse improved current generation of logs (Tiled Transparency Logs)

- Collection of immutable, cacheable resources
- Multiple implementations
- https://c2sp.org/tlog-tiles

Changes in MTC address many remaining operational challenges

tlog-based work (e.g. vindex) can apply

Building on existing CT work: witnesses and mirrors

Witnesses and mirrors come from existing CT community work

A *witness* is lightweight service that enforces append-only view of logs

- Maintains the latest checkpoint of a log
- Given consistency proof to new checkpoint, updates state and cosigns checkpoint
- Quorum of witnesses defends against split view attacks

A *mirror* is a witness that also mirrors the log

- Will not cosign checkpoint until tree is mirrored
- Quorum of mirrors ensure durable serving

“Just” a funny signature

X.509 certificate contains

- TBSCertificate: information about *subject*, certified by *issuer*
- Signature: proves issuer certified TBSCertificate

MTC certificates prove TBSCertificate with inclusion proofs and cosignatures

Encode these as the “signature”

Serving software largely ignores X.509 signatures

Can think of MTC as a funny signature algorithm that logs everything it signs

Certificate negotiation in TLS

Address handshake size with optimized and full certificates

- Not limited to lowest common denominator certificate

TLS is an online protocol, with a rich history of negotiation

Server and client certificate selection

- Sections 4.4.2.2 and 4.4.2.3 of RFC 8446
- draft-ietf-tls-trust-anchor-ids

Draft charter: "… when used in interactive protocols like TLS"

PKI and transparency at large

Navigating the vast forest

Put down roots before branching out

Scope of work

Many exciting things going on around transparent PKIs

- Subtree optimization for inclusion proofs
- Details on certificate construction
- Putting other structures in the logs, e.g. revocation
- Log serving, witnessing, mirroring protocols
- More efficient monitor lookups
- <Your cool ideas here!>

Draft charter focuses on putting down roots for the future

- “...mechanisms needed to interoperably construct and consume certificates...”
- “An extensible ... transparency log structure...”

After core work, we can recharter and branch out as IETF work is identified

Putting down roots

Relying Party Policies

Put it all together: which trusted cosigners, policies, etc.

Transparency Ecosystem

Log serving protocols

Cosigner roles and protocols

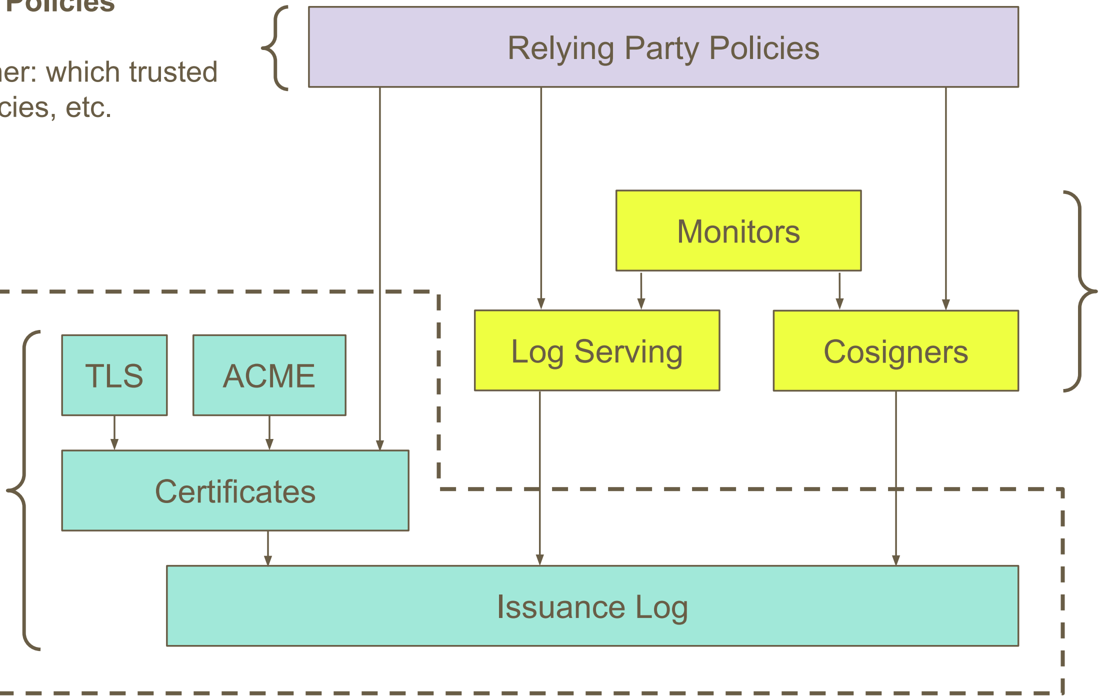
Running code: Static CT and Tiled Transparency Logs

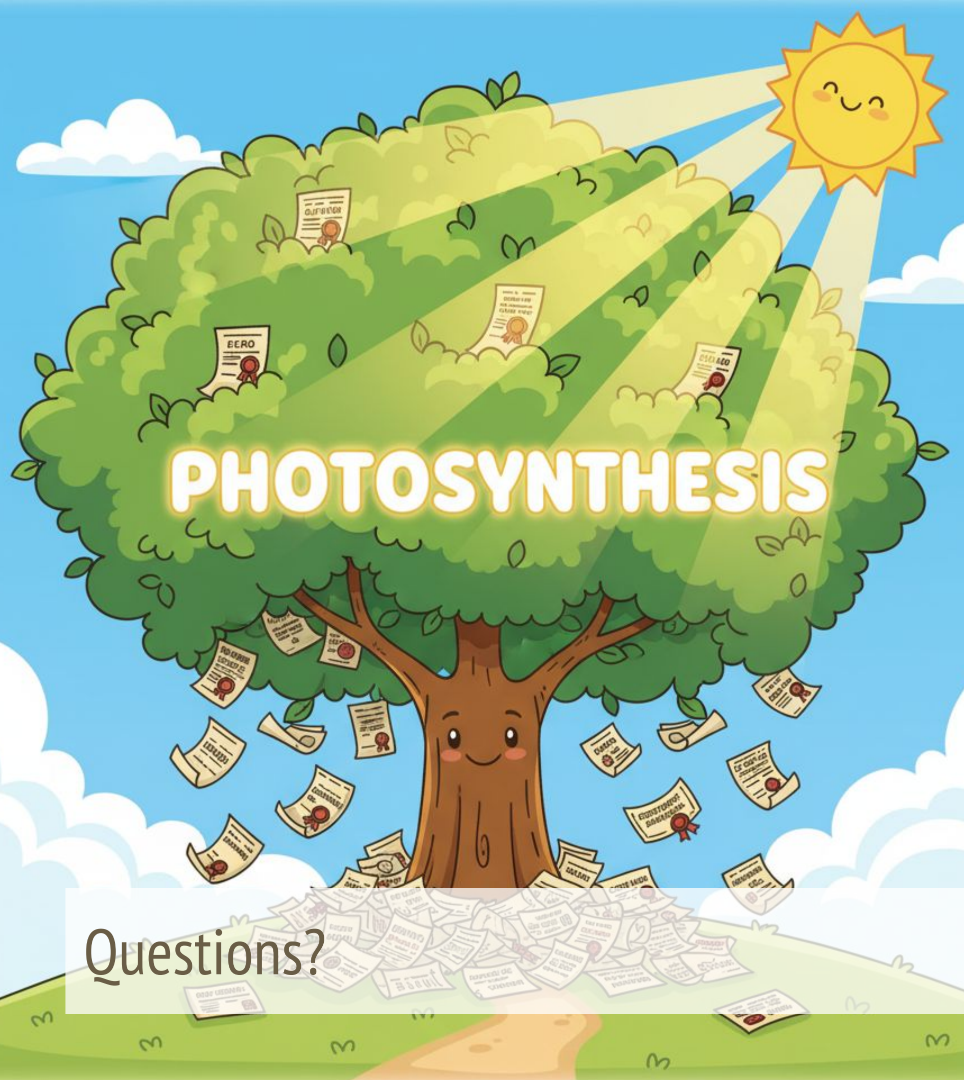
Certificate Construction

Certificate profile

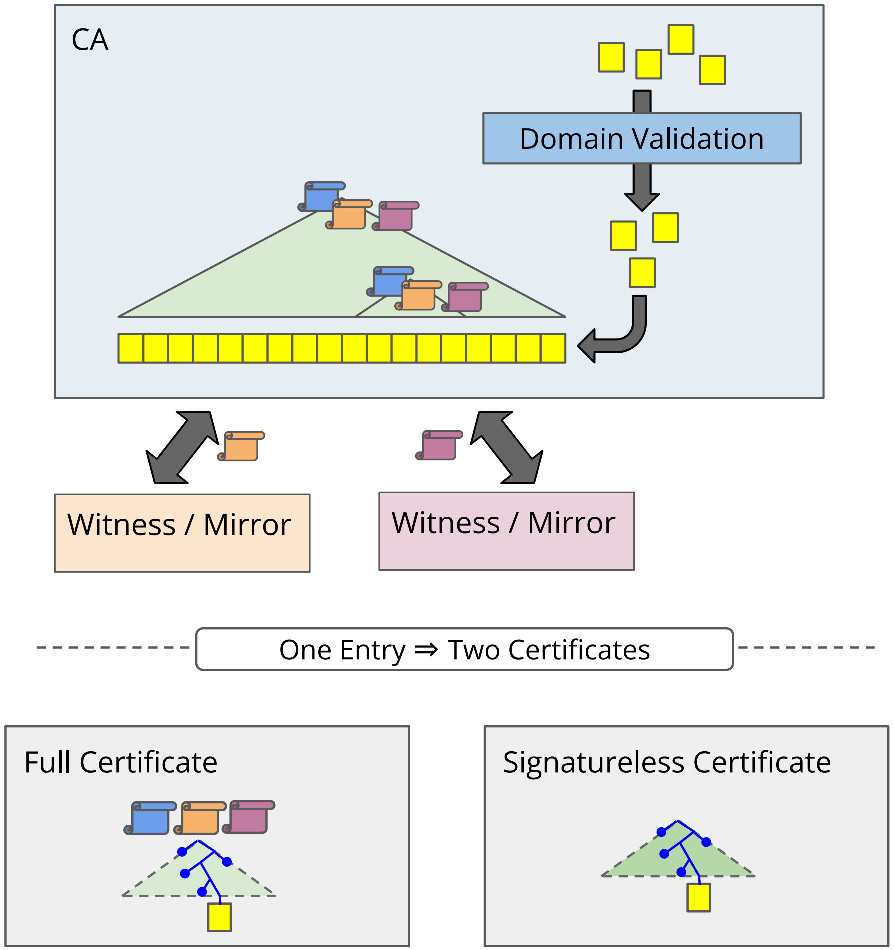
Issuance log structure

TLS, ACME integration




PHOTOSYNTHESIS

Questions?


CA
Domain Validation
Witness / Mirror
Witness / Mirror
One Entry ⇒ Two Certificates
Full Certificate
Signatureless Certificate