

Lettuce Get To Work!



David Benjamin

植物
IETF ~~一百二十五~~

We have a draft!

Workgroup: PKI, Logs, And Tree Signatures
Internet-Draft: draft-ietf-plants-merkle-tree-certs-latest
Published: 15 March 2026
Intended Status: Standards Track
Expires: 16 September 2026
Authors: D. Benjamin D. O'Brien B. E. Westerbaan L. Valenta
 Google LLC Apple Inc. Cloudflare Cloudflare
 F. Valsorda
 Geomys

Merkle Tree Certificates

Now what?

Draft-02 updates

Architecture document

ACME and TLS connections

Log generations

Representing a CA

Looking further ahead

Draft-02 updates

Renamed terms

“full certificate” ⇒ “standalone certificate”

“signatureless certificate” ⇒ “landmark certificate”

<https://github.com/davidben/merkle-tree-certs/pull/187>

Public key types included in TBSCertificateLogEntry

<https://github.com/davidben/merkle-tree-certs/pull/186>

Architecture

Architecture document

July 2026: “Submit an informational architecture document describing taxonomy, information flows, and use cases.”

Opportunity to clarify:

- Kinds of applications where MTC is (and isn't) applicable
 - See Filippo's talk
- How MTC fits into an overall transparency story

History from RFC 6962...

Merkle Trees	
6962 Log Entries 6962 SCTs	6962 Log API

RFC 6962 was really two suites of protocols:

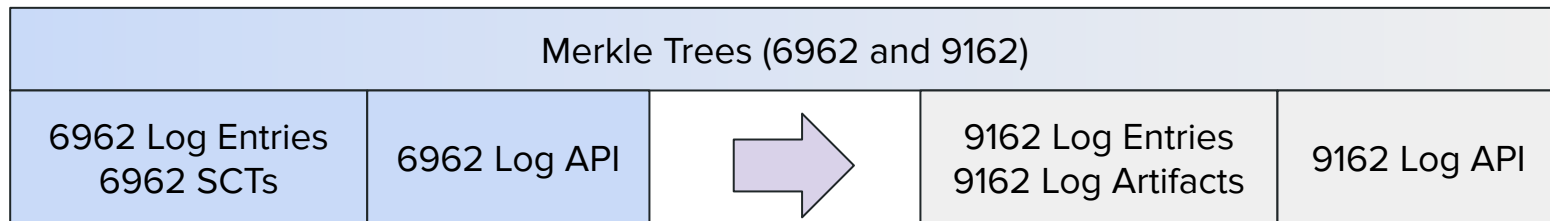
- Log entry format and log artifacts used in TLS (etc) and X.509
- Log serving APIs, monitoring, auditing, etc.

Two suites are fairly independent:

- Log artifacts (SCTs, etc.) don't depend on serving API
- Serving APIs don't depend on entry format or how log artifacts are used

CT evolution reflects this split...

...RFC 9162...



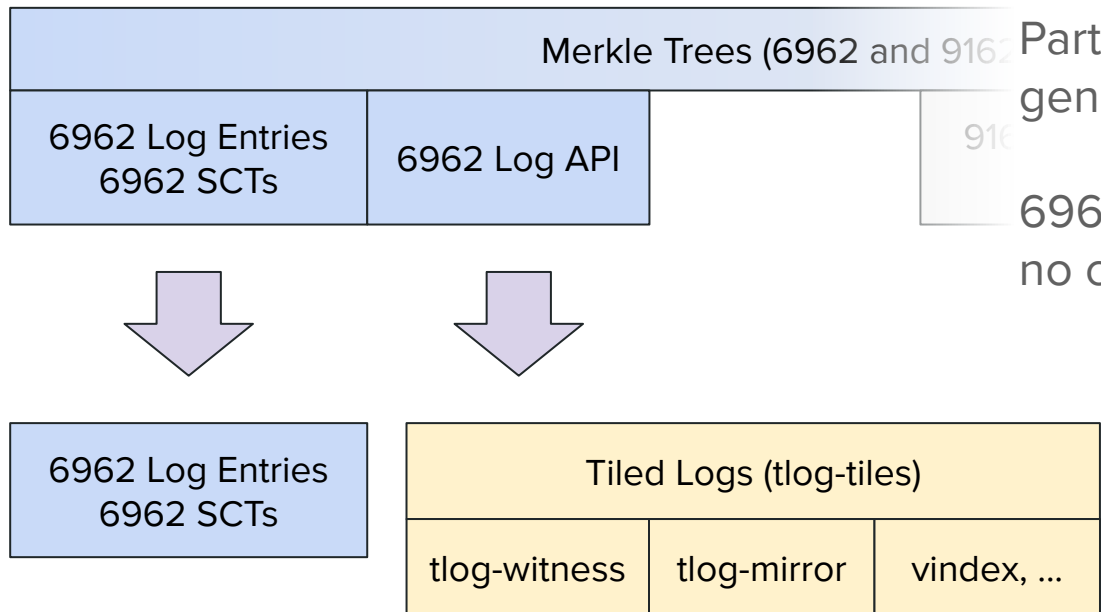
RFC 9162 updates both halves of RFC 6962

Not deployed in practice

Deployments ended up going in a different direction...

...Static CT API...

More scalable, low-cost log API than 6962 or 9162

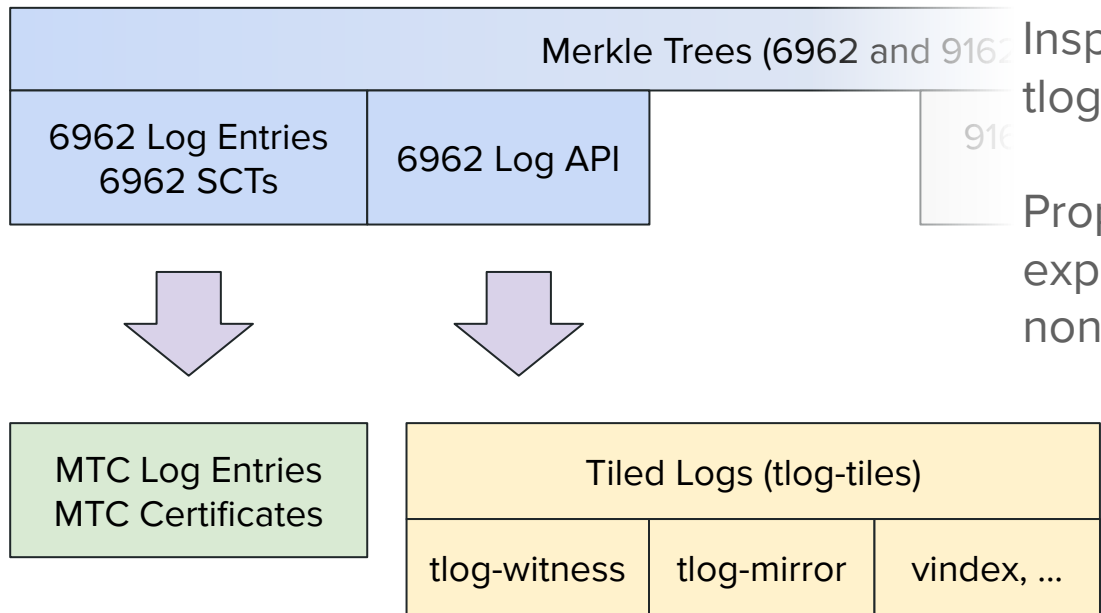


Part of “tlog” ecosystem of generic transparency protocols

6962 log entries and SCTs as-is: no change to X.509 or TLS

...Merkle Tree Certificates

MTC improves log entries and use in certs



Inspired by and compatible with tlog, but not required to use

Proposal: Architecture draft explains how MTC fits in all this, non-normative overview of tlog

What else?

MTC draft currently discusses both “architecture” and “protocol”

What goes in which draft?

Plan: Discuss, then put together concrete text for WG to consider

Current discussion here:

<https://mailarchive.ietf.org/arch/msg/plants/7m9bD2RpFTEnkuYXtmy3X7vtJq0/>

<https://github.com/davidben/merkle-tree-certs/issues/192>

ACME and TLS

ACME

MTC CA produces two certificates per issuance, not one:

- Standalone certificate
- Landmark certificate — optional, issued on a delay, not usable by all RPs

How to fetch via ACME?

- Delay landmark certificate without delaying standalone
- Attach metadata needed for certificate selection

Draft has initial proposal — repurpose ACME alternate URLs mechanism

Discussion at ACME list to refine this

TLS

Landmark certificates not usable by all RPs

Somehow communicate RP state to select certificate

- RP may have different state per CA — one CA may be temporarily inaccessible
- One slow CA should not break optimization for all CAs

Fits draft-ietf-tls-trust-anchor-ids

- Model landmark as a trust anchor
- Append landmark # to CA's base ID, e.g. **32473.1.400**
CA landmark

Wire-compatible extension to improve this — 32473.1.**400** implies 32473.1.**398**, **.397**, ...

Discussion at TLS list

Log Generations

Log generations

Current design assumption:

1 CA instance \Leftrightarrow 1* cosigner key \Leftrightarrow 1 issuance log

Log generations:

1 CA instance \Leftrightarrow 1* cosigner key \Leftrightarrow N issuance logs

* Unless we use multiple cosigners for key rotation

Cosigned MTC CAs have new failure modes

MTC CA signatures commit to log state

What happens if CA accidentally makes a bad commitment?

- CA loses state and cosigners are further ahead
- CA commits to two versions of checkpoint N and partitions cosigners
- CA commits to impossible checkpoint with no known preimage

Breaks transparency — cosigners prevent CA from continuing by design

How does this impact broader PKI?

PKI incident management

CA instance broken? Stop trusting that CA instance and move to another

Maybe same CA operator if still trustworthy, different operator if not

Updated RPs get reconfigured, out-of-date RPs stay the same

Services often need to serve *both* old and new RPs

Old RPs may *only* trust old CA instance

Existing CA failures: old CA is still *functional*, just may not be *trusted*

New MTC CA failures: old CA may be stuck and unable to issue

Incident recovery goals

Up-to-date RPs can be served *and* are secure

Out-of-date RPs *at least* can be served

Log generations: CA is a *sequence* of numbered logs with a shared signer

Before

Issuer identifies CA (single log)

Serial # identifies entry (entry index)

After

Issuer identifies CA (log sequence)

Serial # identifies entry

(generation $\times 2^{64}$ + entry index)

Transparency

Must ensure monitors are watching generations that RPs accept

Range of options:

- RPs accept all generations, trust cosigners and monitors to coordinate on which generations exist
- RPs enforce updatable max generation, relax to all generations if out-of-date
- RPs enforce static max generation, pre-allocate, say, 64 generations and assume CA will not run out

None of these impact certificate wire format

Proposal: spec certificate format, leave room for deployments to iterate on strategy

Representing a CA

CAs as X.509 certificates

Many existing applications use X.509 for trust anchors

Some RPs can design better format, but drop-in format easier for long tail

TODO in current draft

Minimum scope: non-transparency-enforcing, standalone-certificate-only verification

Should encode:

- This is an MTC CA
- Cosigner ID
- Cosigner key
- Cosigner signature algorithm
- Bounds on generation number and entry index?

What's in a key?

Normal SPKI, extra info in critical extension

```
id-pe-mtcCertificateAuthority  
  OBJECT IDENTIFIER ::= { TBD }
```

```
MTCertificateAuthority ::= SEQUENCE {  
  cosignerID TrustAnchorID,  
  signatureAlgorithm AlgorithmIdentifier,  
  minSerial INTEGER,  
}
```

Extension implies MTC CA

Triggers MTC signature algorithm for normal SPKI

Assumes RPs use trust anchor extensions

New kind of SPKI for cosigner

```
id-alg-mtcCosigner  
  OBJECT IDENTIFIER ::= { TBD }
```

```
MTCCosignerPublicKey ::= SEQUENCE {  
  cosignerID TrustAnchorID,  
  signatureAlgorithm AlgorithmIdentifier,  
  publicKey SubjectPublicKeyInfo,  
  minSerial INTEGER,  
}
```

New kind of “public key” paired with MTC signature algorithm

Two-level SPKI

Looking Further Ahead

Looking further ahead

More opinionated and/or recommended ID allocation

<https://github.com/davidben/merkle-tree-certs/issues/152>

Designing a PQ transition

<https://www.chromium.org/Home/chromium-security/post-quantum-auth-roadmap/>

Cosigner negotiation

Don't send cosignatures that RP doesn't need

<https://github.com/tlswg/tls-trust-anchor-ids/issues/54>

加油！

油多菜更香

Questions?