RFC6962 vs RFC6962-bis: What's changed?

Eran Messeri, Google

Unchanged

The Merkle tree structure and its use have remained the same. In particular:

- Nodes and leaves are hashed the same way.
- Inclusion, consistency proofs are conceptually the same.
- Concept of precertificates (the implementation **has** changed).
- SCT distribution mechanisms.

Removed

- The concept of Precertificate Signing Certificate is gone, as precertificates are no longer X.509 certificates.
- Precertificate Poison Extension is no longer needed.
- One layer of abstraction from the MerkleTreeLeaf data structure. Entries are now simplified for certs / precerts.
- SCTs for X.509 certs are no longer over the entire cert as-is.

Changed (overview)

- Precertificates are CMS objects containing the TBSCertificate designated for the final certificate.
  - The certificate’s serial number is still included.
- Dealing with private DNS labels:
  - Labels can be redacted from domain names in precerts.
  - Name-constrained intermediate CA certs can be logged instead of the leaf cert.
- OIDs are now used as Log IDs instead of a public key hash, taking up less bytes (but requiring more administrative work).

Log entries in RFC6962-bis

New data structure defined: `TransItem`

```
struct {
    VersionedTransType versioned_type;
    select (versioned_type) {
        case x509_entry_v2: TimestampedCertificateEntryDataV2;
        case precert_entry_v2: TimestampedCertificateEntryDataV2;
        case x509_sct_v2: SignedCertificateTimestampDataV2;
        case precert_sct_v2: SignedCertificateTimestampDataV2;
        case tree_head_v2: TreeHeadDataV2;
        case signed_tree_head_v2: SignedTreeHeadDataV2;
        case consistency_proof_v2: ConsistencyProofDataV2;
        case inclusion_proof_v2: InclusionProofDataV2;
       case x509_sct_with_proof_v2: SCTWithProofDataV2;
       case precert_sct_with_proof_v2: SCTWithProofDataV2;
    } data;
} TransItem;
```

Log Entries (cont'd)

Each leaf is the leaf hash of a `TransItem` structure of types:

- `x509_entry_v2`
- `precert_entry_v2`

Both structures encapsulates a `TimestampedCertificateEntryDataV2` (only those two `TransItem` types are allowed in the log).

Log Entries: leaf data

```
opaque TBSCertificate<1..2^24-1>;

struct {
    uint64 timestamp;
    opaque issuer_key_hash[HASH_SIZE];
    TBSCertificate tbs_certificate;
    SctExtension sct_extensions<0..2^16-1>;
} TimestampedCertificateEntryDataV2;
```

The `issuer_key_hash` binds the issuer to the `tbs_certificate`.

Log entries: original submission

```
opaque ASN.1Cert<1..2^24-1>;
struct {
    ASN.1Cert leaf_certificate;
    ASN.1Cert certificate_chain<0..2^24-1>;
} X509ChainEntry;
opaque CMSPrecert<1..2^24-1>;
struct {
    CMSPrecert pre_certificate;
    ASN.1Cert precertificate_chain<1..2^24-1>;
} PrecertChainEntryV2;
```

The `X509ChainEntry`/`PrecertChainEntryV2` are returned by the log together with the leaf data in reply to a get-entries call.

Data structure-related changes

`TransItem` can be used everywhere RFC6962 SCTs can be used (TLS extension, embedded in certificates, etc).

- Allows embedding/attaching inclusion proofs alongside certificates (and SCTs).
- Can be used to provide clients with new STHs, etc.

(Where `SCTList` was used, a `TransItemList` is now used)

Signed Certificate Timestamps

```
struct {
    LogID log_id;
    uint64 timestamp;
    SctExtension sct_extensions<0..2^16-1>;
    digitally-signed struct {
        TransItem timestamped_entry;
    } signature;
} SignedCertificateTimestampDataV2;
```

Note:

- The `timestamped_entry` is a `TransItem` of type `x509_entry_v2` or `precert_entry_v2` only.
- Extensions (`SctExtension`) are typed.
- The LogID is a part of the SCT.

Client API changes

- TransItems are returned from the log TLS encoded, not in JSON.
- Error codes (fixed strings) and error messages (human readable) are defined for all methods.
- HTTP codes 500, 503 are explicitly defined as transient errors.
- Dealing with skew on distributed log implementations:
  - a front-end receiving a request for an inclusion or consistency proof it is not aware of can now reply with a proof chaining to the STH it is aware of, including the STH itself.

Method-specific client API changes

Return value changes for `add-chain, add-pre-chain, get-sth, get-sth-consistency, get-proof-by-hash`:

- SCTs, STHs are returned in their binary representation, base64-encoded.
- Instead of JSON which then had to be serialized to the right form.
- Same goes for inclusion, consistency proofs.

Client API changes (cont'd)

`add-pre-chain`: Different precertificate encoding (using CMS).

`get-entries`: `extra_data` renamed to `log_entry`, SCT also returned.

`get-sth-consistency`: Optionally returns an STH if the second tree size provided is unknown (and a consistency proof to that STH).

`get-proof-by-hash`, `get-entries`: Optionally returns a base64-encoded STH if the tree size specified is unknown.

`get-all-by-hash`: New method for providing STH + consistency + inclusion proofs if the tree size specified is unknown.

Client API changes (cont’d)

Renamed:

- `get-roots` was renamed to `get-anchors`

Removed:

- `get-entry-and-proof`

Backwards compatibility

- Logs can either conform to RFC6962 (“v1”) or RFC6962-bis (“v2”), not both.
- v1 and v2 SCTs are delivered using different X.509, TLS and OCSP extensions.
  - They can mostly co-exist.
  - TLS clients can support both simultaneously.
- Except for embedded SCTs: v2 clients are requried to remove v1 SCTs from the TBSCertificate portion of the certificate before validating v2 SCTs over it.