Overview of changes between RFC6962 and RFC6962-bis

Author: Eran Messeri, eranm@google.com
Last updated: 2016-7-21

Overview

RFC6962-bis is a standards-track adaptation of RFC6962. It is the result of industry feedback, experience from implementing RFC6962 and discussions on the trans IETF mailing list.

Unchanged

The Merkle tree structure and its use have remained the same (although the data structures contained have changed). In particular:

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

Removed

- The concept of Precertificate Signing Certificate is gone: Its purpose was to allow a different issuer to sign a precertificate, thus avoiding the same issuer signing two X.509 certificates with the same serial number (even though one of them is unusable). As precertificates are no longer represented as X.509 certificates, that reasoning is gone.
- Precertificate Poison Extension is no longer needed for the same reason.
- One layer of abstraction from the MerkleTreeLeaf data structure: Used to be a struct that only had one type, "timestamped_entry". Entries are now simplified for certs / precerts.

Changed

- Precertificates are CMS objects containing the TBSCertificate designated for the final certificate. Crucially, the certificate's serial number is still included.
- Signatures in SCTs for X.509 certs no longer cover the entire certificate as-is.
- 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).
- SCT Extensions are now typed.
- STHs can now contain extensions (which are also typed).

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;
```

Each leaf is the leaf hash of a `TransItem` structure of type `x509_entry_v2` or `precert_entry_v2` , which encapsulates a `TimestampedCertificateEntryDataV2` (only those two `TransItem` types are allowed in the log).

The leaf data is:

```
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`.
Additionally, the log logs the 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.

Signed Certificate Timestamps are now defined uniformly for X.509 certs and precertificates:

```
enum {
    reserved(65535)
} SctExtensionType;

struct {
    SctExtensionType sct_extension_type;
    opaque sct_extension_data<0..2^16-1>;
} SctExtension;

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

Notes:

- The timestamped_entry is a TransItem of type x509_entry_v2 or precert_entry_v2 only.
- The LogID is now a part of the SCT.
- Extensions are now typed (no extensions are currently defined).

Name redaction

Some labels of domain names can be redacted when a precertificate is logged. This is done by submitting precertificates without the subjectAltName extension, but with a similarly-formatted extension where some labels in dNSName fields are replaced with a hash of the label (and SKI, used as a salt). Redaction of wildcards (‘*’) is not allowed.

Client API changes

All HTTP methods used to access logs have changed and are now under the `ct/v2` namespace.
Notable changes:

- Wherever there’s a `TransItem` type defined for a datum, that datum is returned from the log in TLS encoding (compared to JSON in RFC6962), intended to simplify client implementation.
- Error codes (fixed strings) and error messages (human readable) are defined for all methods.
- HTTP codes 500, 503 are explicitly defined as transient errors.
- Provisions have been made to deal with skew on distributed log implementations. In particular, to cope with a situation where a log’s front-ends know about (slightly) different STHs, 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 (see `get-all-by-hash` and the changes to `get-proof-by-hash`, `get-entries`).

Method-specific 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.

- `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.

Renamed:

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

Removed:

- `get-entry-and-proof`

New, **optional**:
These methods can optionally be implemented by logs but are not an essential part of the protocol. Crucially, they can be implemented by log mirrors even if the logs mirrored do not support them.

- `get-entry-for-sct` : Returns the entry index given an SCT, if incorporated into the log.
- `get-entry-for-certificate` : Returns the entry index for a TBSCertificate hash, if exists in this log.

Backwards compatibility

Logs can either conform to RFC6962 (“v1”) or RFC6962-bis (“v2”), not both, as the format of the data logged is different.

As v1 and v2 SCTs are delivered using different X.509, TLS and OCSP extensions, they can mostly co-exist and TLS clients can support both simultaneously.

One provision has been made for embedded v1 and v2 SCTs by requiring v2 clients to remove v1 SCTs from the TBSCertificate portion of the certificate before validating v2 SCTs over it (to allow v2 SCTs to be embedded before v1 SCTs are issued).