Protocol Document

Brendan McMillion
IETF 126 / July 24, 2026

Monitoring Wire Formats

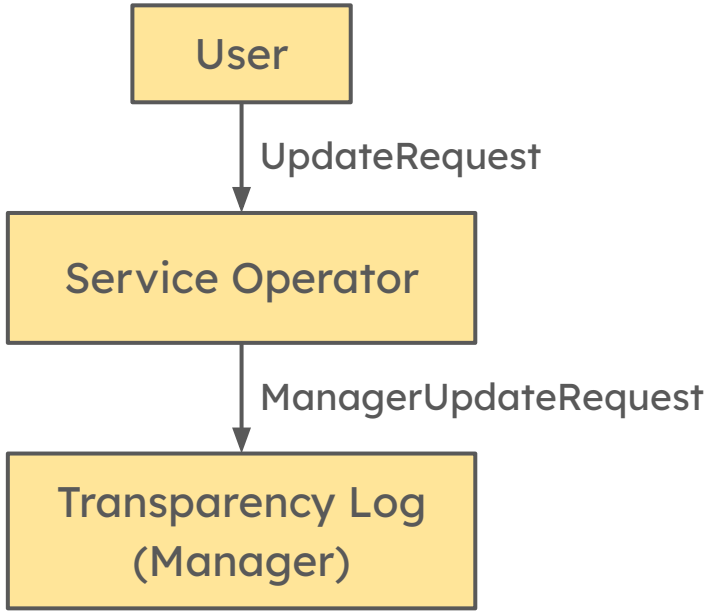
- The algorithms for performing operations are (essentially) unchanged since last IETF.
- However, at last IETF, we didn’t have concrete request / response wire formats for monitoring-related operations:
  - Contact Monitoring, Owner Initialization, Owner Monitoring
- Now we do!

Third-Party Management

Recap: TPM is a Deployment Mode where a third-party operates the Transparency Log and the Service Operator just signs new label versions.

Previous wire format:

```
struct {
  UpdateRequest request;
  opaque signature<0..2^16-1>;
} ManagerUpdateRequest;
```



Third-Party Management

Implementing this is impossible:

- Service Operator's signature needs to cover the version number, **BUT…**
- The version number is being assigned by the Transparency Log

Also doesn't support multiple versions in one UpdateRequest.

```
struct {
  UpdateRequest request;
  opaque signature<0..2^16-1>;
} ManagerUpdateRequest;
```

Third-Party Management

New wire format:

```
struct {
  optional<uint64> last;

  opaque label<0..2^8-1>;
  optional<uint32> greatest_version;
  UpdateValue values<0..2^8-1>;

  uint32 signed_version;
} ManagerUpdateRequest;
```

= User's UpdateRequest

Vector of new values + sig over each

First version counter assigned
(Means Service Operator is now authoritative for assigning version numbers)

Credentials

Recap: Credentials are KT proofs meant to be sent peer-to-peer and verified with no network access. Useful for anonymity.

- Now fully specified. Two types: *standard* and *provisional*
- Both correspond (roughly) to a Greatest-Version Search for a label
- All credentials are constructed wrt to a “recent” distinguished log entry
  - Applications provide their own definition of “recent”, just matters that all users agree (modulo reasonable clock skew)
- Validity period: Starts when DLE is issued, ends when DLE is no longer “recent”. Reasonable clock skew applies to both of these.

Credentials (Standard)

The version of the label we want exists in the distinguished log entry.

```
struct {
  opaque label<0..2^8-1>;

  uint32 version;
  opaque opening[Nc];
  UpdateValue value;
  BinaryLadderStep binary_ladder<0..2^8-1>;

  uint64 position;
  CredentialType credential_type = standard;
  select (Credential.credential_type) {
    case standard:
      PrefixProof distinguished;
  };
} Credential;
```

Label for credential

= SearchResponse for `label`

Position of the distinguished log entry that credential is based on

Lookup for `version` of `label` in log entry `position`

Credentials (Standard)

The version of the label we want exists in the distinguished log entry.

```
struct {
  opaque label<0..2^8-1>;

  uint32 version;
  opaque opening[Nc];
  UpdateValue value;
  BinaryLadderStep binary_ladder<0..2^8-1>;

  uint64 position;
  CredentialType credential_type = standard;
  select (Credential.credential_type) {
    case standard:
      PrefixProof distinguished;
  };
} Credential;
```

- Relatively small, fast to verify
- Never triggers Contact Monitoring

Credentials (Provisional)

The version of the label we want **doesn't** exist in the DLE.

- May exist in another DLE, just not the one the prover chose.

```
struct {
  opaque label<0..2^8-1>;

  uint32 version;
  opaque opening[Nc];
  UpdateValue value;
  BinaryLadderStep binary_ladder<0..2^8-1>;

  uint64 position;
  CredentialType credential_type = provisional;
  select (Credential.credential_type) {
    case provisional:
      TreeHead tree_head;
      CombinedTreeProof search;
  };
} Credential;
```

The same as before

Tree head
+ Greatest-Version Search

Credentials (Provisional)

The version of the label we want **doesn't** exist in the DLE.

- May exist in another DLE, just not the one the prover chose.

```
struct {
  opaque label<0..2^8-1>;

  uint32 version;
  opaque opening[Nc];
  UpdateValue value;
  BinaryLadderStep binary_ladder<0..2^8-1>;

  uint64 position;
  CredentialType credential_type = provisional;
  select (Credential.credential_type) {
    case provisional:
      TreeHead tree_head;
      CombinedTreeProof search;
  };
} Credential;
```

- Somewhat larger, slower to verify
- Usually triggers Contact Monitoring

Credentials (Provisional)

- Any user that sends a provisional credential needs to send a CredentialUpdate structure within a bounded amount of time
- If they don’t, users that received the credential need to request it from the Transparency Log

```
struct {
  uint64 position;
  CombinedTreeProof monitor;
} CredentialUpdate;
```

Next distinguished log entry

Contact Monitoring proof

New Operation: Walking Distinguished Heads

Gap in protocol functionality:

- Users that verify credentials are expected to know all recent DLEs
- Users that want to gossip need some common reference point
- To make an OwnerInitRequest, users need to specify a DLE

You can kind of get this information from Owner Monitoring, but it’s messy

New Operation: Walking Distinguished Heads

Operation implementation:

- Right-to-left depth-first search for all “recent” distinguished log entries, given some application definition of “recent”.
- Users can optionally provide a **stopping point**, representing a DLE they already know about, where the search can stop early if reached.

Suggested definitions of “recent” for distinguished log entries:

- One of the `n` rightmost
- Log entry timestamp is less than `x` seconds in past

New Operation: Walking Distinguished Heads

Wire format:

```
struct {
  optional<uint64> last;
  optional<uint64> stop;
} DistinguishedRequest;
```

```
struct {
  FullTreeHead full_tree_head;
  CombinedTreeProof distinguished;
} DistinguishedResponse;
```

Appendix: Maximum Lifetime vs Reasonable Monitoring Window

Document contains two similar concepts the interact in subtle ways:

- **Maximum Lifetime:** Amount of time after which log entries are deleted. Users assume these log entries are expired and off-limits.
- **Reasonable Monitoring Window:** The general frequency with which users are expected to perform monitoring. This determines which log entries are distinguished.

Appendix: Maximum Lifetime vs Reasonable Monitoring Window

New text explains why, among other things:

- There's always at least one unexpired distinguished log entry for users to inspect
- Contact Monitoring is guaranteed to terminate within one interval of the Reasonable Monitoring Window
- Users that perform monitoring at least once per every interval of the Reasonable Monitoring Window will never need to monitor an expired log entry.

Next Steps

Asking people to review before WGLC so that WGLC can go smoothly :)

Implementation Report

Bren2010/katie (Go)

- Essentially complete
- Could use improved documentation, more thorough tests

Cleanroom Implementation: claude-kt

Prompt: *“There's documentation describing a protocol in the docs folder. Please implement a client and server for this protocol in Golang.”*

Cleanroom Implementation: claude-kt

Prompt: *“There's documentation describing a protocol in the docs folder. Please implement a client and server for this protocol in Golang.”*

- Thought for ~1 hour, consumed $39 in tokens (= hit $40 limit)
- Wrote 30 files, 7100 lines of code. Some functionality omitted.
- Found 1 substantive mistake in spec (and katie), ~10 drafting errors

Cleanroom Implementation: claude-kt

Prompt: *“There's documentation describing a protocol in the docs folder. Please implement a client and server for this protocol in Golang.”*

- Thought for ~1 hour, consumed $39 in tokens (= hit $40 limit)
- Wrote 30 files, 7100 lines of code. Some functionality omitted.
- Found 1 substantive mistake in spec (and katie), ~10 drafting errors
- Wire compatible with Bren2010/katie as far as I’ve seen

Experiment successful??

That’s all

Thanks :)