Public Notary Transparency
Internet Draft
Intended status: Standards Track
Expires: April 2016

S. Kent
D. Mandelberg
K. Seo
BBN Technologies

October 2, 2015

Certificate Transparency (CT) System Architecture
draft-kent-trans-architecture-00.txt

Status of this Memo

Copyright Notice

Abstract

This document describes the architecture for Certificate Transparency (CT) focusing on the Web PKI context. It defines the goals of CT and the elements that comprise CT. It also describes the critical features of these elements. Other documents describe in detail the operation of these elements.

Table of Contents

1. Introduction

Certificate transparency (CT) is a set of mechanisms designed to deter, detect, and facilitate remediation of certificate mis-issuance. CT deters mis-issuance by encouraging CAs to publish the certificates that they issue in a publically-accessible log. The log uses a Merkle tree design to ensure that it is an append-only database, and the log entries are digitally signed by the log operator. Monitoring of logs detects mis-issuance. Remediation of mis-issuance is effected via certificate revocation.

The term mis-issuance refers to violations of either semantic or syntactic constraints associated with certificates. The fundamental semantic constraint for a (Web PKI) certificate is that it was issued to an entity that is authorized to represent the Subject name in the certificate, in addition to all Subject Alternative names (SANs), if any are present. (It is also assumed that the entity requested the certificate from the CA that issued it.) Throughout the remainder of this document we refer to a semantically mis-issued certificate as "bogus."

A certificate is characterized as syntactically mis-issued if it violates syntax constraints associated with the class of certificates that it purports to represent. Syntax constraints for certificates are established by certificate profiles, and typically are application-specific. For example, certificates used in the Web PKI environment might be characterized as domain validation (DV) or extended validation (EV) certificates. Certificates issued for use by applications such as IPsec or S/MIME have different syntactic constraints from those issued in the Web PKI context. Throughout the remainder of this document we refer to a syntactically mis-issued certificate as "erroneous."

As noted above, CT deters mis-issuance by encouraging CAs to log the certificates that they issue. A CT log is a publicly auditable, append-only, database of issued certificates [cite 6962-bis] based on a binary Merkle hash tree [Merkle]. Each CT log operates in a fashion that enables anyone to detect inconsistent behavior, thus logs need not be operated by trusted (third) parties. (Detection of inconsistent behavior by a log is the function of a CT Auditor. Some forms of log misbehavior require comparing information gleaned from multiple sources, e.g., using mechanisms such as the ones described in [Gossip]. If an Auditor detects misbehavior by the log, it will notify Monitors and Subjects that have registered with it.) A (semantically) mis-issued certificate that has been logged can be detected by any entity that monitors the log and that has knowledge of all legitimate certificates issued to the named certificate Subject. Thus CAs are deterred from logging mis-issued certificates, because of the implied reputational consequences. (The assumption is that a CA that is detected repeatedly mis-issuing certificates may be shunned by the community.)

CT enables detection of mis-issuance via the Monitor function [cite Monitor]. A CT Monitor examines all entries from a set of logs and compares these entries to reference data for a set of one or more Subjects. The reference data consists, at a minimum, of a list of Subject and Subject Alternative Names and the pubic key information associated with each, supplied by the Subject. If a Monitor detects a

log entry for a certificate that is inconsistent with the reference data for a Subject, the Monitor notifies the Subject. A Subject may perform self-monitoring. In the Web PKI context, a Subject is a web site. Monitors implement the mis-issuance detection aspect of CT.

Revocation of a bogus/erroneous certificate is the primary means of remedying mis-issuance. A browser vendor may distribute a "blacklist" of mis-issued certificates or a bad-CA-list of certificates of CAs that have mis-issued certificates. Browsers may then use such lists to reject certificates on the blacklist, or certificates for which the issuing CA is on the bad-CA-list. This form of revocation, although not codified in IETF standards, is also a means of remediation for mis-issuance. Throughout the remainder of this document, references to certificate revocation as a remedy encompass these and analogous forms of revocation.

Figure 1 provides a top-level view of these elements of CT.

```
   +-----+     +----+
   | Log |<--->| CA |<*********************
   |     |     +----+                     *
   |     |        ^                       *
   |     |        *                       *
   |     |        v                       *
   |     |     +---------+                *
   |     |<--->| Subject |<*************  *
   |     |     +---------+             *  *
   |     |        ^   ^                *  *
   |     |        *   *******          *  *
   |     |        v         *          *  *
   |     |     +---------+  *          *  *
   |     |<--->| Browser |  *          *  *
   |     |     +---------+  *          *  *
   |     |        ^         *          *  *
   |     |        *         *          *  *
   |     |        v         v          *  *
   |     |     +----------------+      *  *
   |     |<***>| Browser Vendor |<***  *  *
   |     |     +----------------+   *  *  *
   |     |                          v  v  v
   |     |                         +---------+
   |     |<----------------------->| Monitor |
   |     |                         +---------+
   |     |                              ^
   |     |                              *
   |     |                              v
   |     |                         +---------+
   |     |<----------------------->| Auditor |
   +-----+                         +---------+

   Legend:
   <---> Interface defined by CT
   <***> Interface out of scope for CT
```

Figure 1 Elements of the CT Architecture

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Beneficiaries of CT

There are three classes of beneficiaries of CT: certificate Subjects, relying parties (RPs), and Certification Authorities (CAs). In the initial context of CT, the Web PKI, Subjects are web sites and RPs are browsers employing HTTPS to access these web sites. CAs are issuers of certificates used in the Web PKI context.

A certificate Subject benefits from CT because CT helps (Monitors) detect certificates that have been mis-issued in the name of that Subject. A Subject learns of a bogus/erroneous certificate (issued in its name), via a CT Monitor, as noted above. (The Monitor function may be provided by the Subject itself, i.e., self-monitoring, or by a third party trusted by the Subject.) When a Subject is informed of certificate mis-issuance by a Monitor, the Subject is expected to request/demand revocation of the bogus/erroneous certificate by the issuing CA and/or by the browser vendors.

A Subject also may benefit from the Monitor function of CT even if the Subject's legitimate certificate(s) has(have) not been logged. Monitoring of logs for certificates issued in the Subject's name suffices to detect an instance of mis-issuance targeting the Subject, if the bogus/erroneous certificate is logged.

A TLS client (e.g., a browser) benefits from CT if the TLS client rejects a mis-issued certificate, i.e., treats the certificate as invalid. A TLS client is protected from accepting a mis-issued certificate if that certificate is revoked, and if the TLS client checks the revocation status of the certificate. (A TLS client also is protected if a browser vendor "blacklists" a certificate or a CA as noted above.) A TLS client also may benefit from CT if the client validates a Signed Certificate Timestamp (SCT) [6962-bis] associated with a certificate, and rejects the certificate if the SCT is invalid.

CAs are also CT beneficiaries. If one CA issues a legitimate certificate to a Subject, and another CA issues a bogus certificate, the second certificate can be detected by CT Monitoring (if the bogus certificate has been logged). In this fashion the CA that issued the legitimate certificate benefits, since the bogus certificate is detected and, presumably revoked. Even the CA that issued the bogus certificate is a potential beneficiary. If the bogus certificate was issued as a result of an error or an (undetected) attack, CT can help the CA become aware of the error or attack and act accordingly. This is presumed to be beneficial to the reputation of this CA.

3. The Elements of the CT Architecture

There are six elements of the CT architecture: logs, CAs, Monitors, Subjects, TLS clients (and vendors of the client's software), and Auditors. (CAs, Subjects, and TLS clients are pre-existing elements affected by CT; logs, Monitors, and Auditors are new elements introduced by CT.) Figure 2 shows how all of these elements interact with the central element, the log. Figure 3 shows how the pre-existing elements interact with one another under CT. Figure 4 shows the interactions of monitors and auditors that are not covered by Figure 2.

```
   +-----+                                     +---------------+
   | Log |<-  add-chain or add-pre-chain  -----| CA or Subject  |
   |     |--  SCT for the new entry  --------->|                |
   |     |<-  get-proof-by-hash  --------------|                |
   |     |--  inclusion proof for the entry  ->|                |
   |     |                                     +---------------+
   |     |                       +---------+
   |     |<-  get-sth [1]  ------| Monitor |
   |     |--  current STH  ----->|         |
   |     |<-  get-entries [1]  --|         |
   |     |--  log entries  ----->|         |
   |     |                       +---------+
   |     |                             +---------+
   |     |<-  get-proof-by-hash [2]  --| Browser |
   |     |--  inclusion proof [2]  --->|         |
   |     |                             +---------+
   |     |                        +----------------+
   |     |<-  get log metadata  --| Browser Vendor |
   |     |--  log metadata  ----->|                |
   |     |                        +----------------+
   |     |                               +-----------------+
   |     |                               |     Auditor     |
   |     |                               |+---------------+|
   |     |<-  get-sth [1]  --------------|| MMD checking  ||
   |     |--  current STH  ------------->||               ||
   |     |<-  get-entries [1]  ----------||               ||
   |     |--  log entries  ------------->||               ||
   |     |                               |+---------------+|
   |     |<-  get-sth  ------------------|| STH frequency ||
   |     |--  current STH  ------------->|| checking      ||
   |     |                               |+---------------+|
   |     |<-  get-sth [1]  --------------|| Append-only   ||
   |     |--  current STH  ------------->|| checking      ||
   |     |<-  get-entries [1]  ----------||               ||
   |     |--  log entries  ------------->||               ||
   |     |<-  get-sth-consistency [3]  --||               ||
   |     |--  consistency proof  ------->||               ||
   +-----+                               |+---------------+|
                                         +-----------------+
```

[1] The get-sth operation is performed periodically, and get-entries is performed each time a new STH is available.
[2] See Section 3.5 for privacy and performance caveats.
[3] If the Auditor stores copies of all Log entries, then this operation is not needed.

Figure 2 Interactions with a Log

```
   +----------------+                                        +---------+
   | Browser Vendor |--  log metadata[1]  ------------------>| Browser |
   |                |--  revocation information[1]  -------->|         |
   +----------------+                                        |         |
                                                             |         |
   +----+                      +---------+                   |         |
   | CA |  / certificate \-----| Subject |                   |         |
   |    |<-\ request     /     |         |                   |         |
   |    |--  certificate[2]  ->|         |                   |         |
   +----+                      |         |                   |         |
                               |         |                   |         |
                               |         |  / TLS        \---|         |
                               |         |<-\ connection /   |         |
                               |         |--  certificate  ->|         |
                               |         |--  SCT[3]  ------>|         |
                               |         |<-  HTTPS  ------->|         |
                               +---------+                   +---------+
```

[1] Not subject to standardization.
[2] Optionally including SCTs in an extension.
[3] Optional, via an OCSP response or in a TLS extension.

Figure 3 Interfaces of Pre-existing Elements

```
   +---------+                                                +---------+
   | Monitor |<-  establish a business relationship [1]  ->| Subject |
   |         |<-  list of protected subject names  --------|         |
   |         |  / per protected subject name, a  \---------|         |
   |         |<-\ list of acceptable public keys /         |         |
   |         |                                             +---------+
   |         |
   |         |                                        +----+
   |         |--  notification of mis-issuance  --+-->| CA |
   |         |                                    |   +----+
   |         |                                    |
   |         |                                    |   +----------------+
   |         |                                    +-->| Browser Vendor |
   |         |                                        +----------------+
   |         |
   |         |                                            +---------+
   |         |<-  notification of log mis-behavior [2]  --| Auditor |
   +---------+                                            +---------+
```

[1] In the case of a self-monitor, the business relationship is trivial - the Subject and Monitor are the same organization.
[2] An entity performing the Monitor function MAY also choose to implement some of the Auditor functions. In that case the Monitor/Auditor interface is trivial. If the Auditor is separate, we note that there is no interface defined at the time of this writing.

Figure 4 Monitor and Auditor Interfaces

3.1. Logs

Logs are the central elements of the CT architecture. Logging of certificates enables Monitors to detect mis-issuance and, subsequently, to trigger revocation requests to CAs and/or browser vendors. Logging also deters mis-issuance, as noted above. The interfaces to a log are defined in [6962-bis], as are the details of how a log operates.

Briefly, a certificate chain (that must be verifiable under a trust anchor acceptable to the log) is submitted to a log by a CA, Subject or other interested party. The log creates an entry for the chain, hashing it with information from other log submissions. The log returns a Signed Certificate Timestamp (SCT) to the submitter. The SCT can be conveyed to RPs in one of three ways: it can be incorporated into a certificate by the CA that issues it; it can be conveyed via the TLS handshake between an RP and a web site; or it can be embedded into an OCSP response sent to an RP. (Only the issuer of a certificate can submit a so-called "pre-certificate" to a log,

to acquire an SCT for inclusion into the certificate, prior to signing the certificate.) The SCT is a token that can be verified by RPs (and Monitors) to establish, to first order, that a certificate has been logged. See [6962-bis] for additional details.

All clients that interact with a log require access to metadata associated with each log upon which they rely. This metadata includes the URL and public key for the log, the list of trust anchors accepted by the log, the hash and signature algorithms employed, etc. Log metadata is made available to RPs via out of band means that are outside the scope of the CT specifications. In the Web PKI context, CT assumes that browser vendors will make log metadata available to browsers via the same mechanisms used to convey trust anchor (and vendor-managed revocation data). Thus log metadata is not mutable by log operators (since it is part of browser configuration data), with one exception. When a log ceases operation it publishes its final STH, enabling clients to verify previous log entries and to detect any (unauthorized) additions to the log. See [6962-bis] for additional details.

3.2. Certification Authorities (CAs)

A CA interacts with a log to submit a certificate (or a pre-certificate) to create a log entry. (Most logged certificates are expected to be end-entity certificates, each associated with the web site that it represents. However, it also is possible to log a CA certificate under certain circumstances. See Section 3.2.3 of [6962-bis].) The pre-certificate capability is offered to facilitate rapid deployment of CT. It has the advantage that web sites need not make any software changes to acquire one or more SCTs, because the SCTs are embedded in the certificate itself. There is, however, a downside of embedding SCTs in certificates. If a log that provided an SCT is compromised or otherwise becomes not acceptable to RPs and Monitors, the certificate associated with that SCT may have to be re-issued with a replacement SCT. Thus, in the long term, the options of conveying an SCT via the TLS handshake or in an OCSP response (perhaps "stapled" into the handshake [RFC6961], are preferred. However, transmission of an SCT via the TLS handshake requires changes to web site software to acquire and insert SCTs. Transmission via an OCSP response requires that either RPs fetch such responses (which appears not to be the norm), or that a web site passes the OCSP data via the TLS handshake (and that the OCSP signer be prepared to generate this modified form of response).

A CA may submit a "name-redacted" pre-certificate to a log. A name-redacted pre-certificate includes one or more "?" labels in lieu of DNS name components. Name-redaction is a feature of CT designed to

enable an organization to log certificates without revealing all of the DNS name components in the certificate that will be matched to the log entry. This is an attractive feature for organizations that want to benefit from CT without revealing internal server names as a side effect of logging. An end-entity certificate that is to be treated as logged via this mechanism MUST contain a critical (X.509v3) extension that indicates which labels have been redacted in the log entry. This extension is needed to enable TLS clients and Monitors to match a received certificate against the corresponding log entry in an unambiguous fashion. See Section 3.2.2 of [6962-bis] for more details.

The CT architecture does not mandate a specific number of SCTs that should be associated with a certificate. TLS clients and Monitors might establish requirements for the minimum number of associated SCTs in different contexts, but such requirements are outside the scope of the CT architecture.

After an SCT has been returned, it is RECOMMENDED that a CA verify that a certificate (or pre-certificate) that it has submitted has in fact been logged. To perform this verification, the CA waits for an interval dictated by the Maximum Merge Delay (MMD) associated with the log, and then requests both a Signed Tree Head (STH) and an inclusion proof. The CA SHOULD then verify the data returned by the log, as described in Sections 3.6, 4.3 and 4.5 of [6962-bis].

<we plan to insert much of Rob's text on redacted certificates here, since that text specifies CA behavior for CT.>

3.3. Monitors

The primary role of a Monitor is to watch a set of logs, looking for log entries of interest. A Subject may act as a self-monitor, or may make use of the services of a third-party Monitor.

In the self-monitoring context, log entries of interest are ones that contain a Subject or Subject Alternative Name (SAN) associated with the Subject's web site(s). (Name-constrained CA certificates and wildcard certificates also have to be examined to detect certificates that would match the end-entity certificates associated with a Subject's web sites.) Whenever a certificate of interest is detected, the Subject compares it with the public key information associated with the Subject's certificate(s). If there is a mismatch, this indicates that this logged certificate was mis-issued. The Subject contacts the CA that issued the certificate (using the Issuer name in the certificate), and requests revocation of the mis-issued certificate, to resolve the problem. (The means by which a Subject

determines how to contact a CA based on the issuer name is outside the scope of this specification.) The means by which a Subject determines which set of logs to watch is outside the scope of the CT specifications. It is anticipated that there will be a small number of logs that are widely used, and that the metadata for these logs will be available from browser vendors (see Section 3.5 below).

A third-party Monitor watches for certificates of interest to its clients. Each client of a third party Monitor supplies the Monitor with a list of Subject names and SANs associated with the client's web site(s), and public key information associated with each name. The Monitor watches a set of logs looking for entries that match the client certificates of interest. If it detects an apparent mis-issued certificate, the Monitor contacts the client and forwards the log entry, along with log metadata. The client (Subject) then follows the procedure noted above to request revocation of the mis-issued certificate. It is RECOMMENDED that third-party Monitors make public the set of logs that they watch, and the set of third-party Auditors they rely upon, to help clients decide when choosing a third-party Monitor.

A Monitor (self or third-party) that is "watching" a log periodically queries the log to determine if there is a new STH, using the get-sth interface (see Section 4.3 of [6962-bis]). When a new STH is detected, the Monitor then uses the get-entries interface to the log (see Section 4.7 of [6962-bis] to retrieve all new log entries (relative to the previous STH acquired by the Monitor). (This command requires the Monitor to indicate the start and end entries, by index, data that is provided by get-STH.) The Monitor examines each log entry to determine if it is of interest, as per the definition above. (This procedure applies to wildcard certificate log entries as well as to certificates with fully-specified DNS names.)

If a Monitor encounters a log entry for a name-redacted certificate (Section 3.2.2 in [6962-bis]) it MUST evaluate whether that certificate is of interest. To do so, the Monitor compares the non-redacted part of the name in the log entry against the list of names of interest to this Monitor. The redacted name, is transformed into a wildcard name by substituting "*" for "?" name components. The resulting name is then compared to the list of names of interest to the Monitor. If a match is found, the Monitor then compares the list of public keys for the name. If the public key in the log entry does not match any in this list, the Subject associated with the specified name is notified.

A Monitor MAY retain its own copies of log entries, but it is not required to do so. Local caching of log entries would be useful for a

third party log that acquires a new client, since the Monitor could examine the older entries for certificates that are now of interest. For a self-Monitor, maintaining a cache of old log entries may not be useful and may represent a storage burden.

Note that the Monitor function, as described above, does not try to detect mis-behavior by a log. That is an Auditor function, which is described below. A Monitor MAY incorporate some or all of the Auditor functions; it MAY make use of third-party Auditors, or it may eschew responsibility for auditing. A third-party Monitor SHOULD make known to its clients which, if any, Auditor functions it offers to its clients. The means by which Subjects determine the set of functions provided by a third-party Monitor is not defined by this document; it will be described in a Monitor API specification [cite Monitor].

CT does not include any mechanisms designed to detect misbehavior by a Monitor. A self-Monitor does not require such mechanisms; Subjects who elect to rely upon third-party Monitors would benefit from such mechanisms.

3.4. Subjects (TLS web servers)

A Subject (e.g., a web site operator) MAY submit its certificate(s) to a log, and acquire an SCT for each certificate it submits, using the add-chain log interface (see Section 4.1 of [6962-bis]). There are three reasons for a Subject to log its own certificate(s): (1) its CA did not embed an SCT in the certificate(s) it issued to the Subject, (2) the Subject wants to acquire SCTs from additional logs, or (3) the Subject wants the flexibility offered by conveying SCTs (from logs of its choosing) in the TLS handshake (including via OCSP). Appendix B describes the requirements imposed on Subjects for delivery of SCTs to CT-enabled TLS clients.

When a Subject has acquired an SCT, it SHOULD perform the same checks described for a CA (see Section 3.2 above), to verify that the log has created an entry for each submitted certificate.

It is RECOMMENDED that every Subject either perform self-monitoring, or become a client of a third-party Monitor (see Section 3.3 above). When a Subject becomes aware of a mis-issued certificate (based on the Monitor function), the Subject confirms that the log entry conflicts with one of its certificates. (In this context, a conflict arises if the name in a Subject's certificate matches or is encompassed by the name in the log entry, and the certificate was not issued to the Subject.) If a conflict is detected, the Subject contacts the CA that issued the certificate and requests that it be revoked, using whatever mechanisms the CA provides for such requests.

The Subject may also contact browser vendors and ask that they put the certificate on a blacklist of mis-issued certificates or put the CA's certificate on a bad-CA-list.

3.5. TLS clients (web browsers)

As noted in Section 2, a TLS client can benefit from CT even without actively participating. A Monitor will detect a mis-issued, logged certificate and notify the affected Subject. The Subject will, in turn attempt to trigger revocation by the CA that mis-issued the certificate in question. If the CA refuses to revoke the certificate, and it is acting "improperly", then the Subject could notify browser vendors who could blacklist the CA or the certificate in question, effecting revocation via other means. Thus a TLS client that processes certificate revocation status data, e.g., CRLs, OCSP responses, can be protected from bogus certificates that have been logged, detected, and revoked. Appendix B describes the requirements imposed on a CT-enabled browser to signal its capability and to accept SCTs conveyed via any of the three methods defined there. Appendix C describes the process a CT-enabled browser uses to match an SCT to a certificate if the SCT is not embedded in the certificate.

If a TLS client required that a certificate it accepted was accompanied by an SCT, the client could have some confidence that the certificate had been logged. This would increase confidence that the certificate, if it were mis-issued, will have been revoked. However, there are two problems with mandating that every TLS client reject (treat as invalid) any certificate that is not accompanied by an SCT. First, such behavior does not accommodate incremental deployment of CT. Second, the mere presence of an SCT is not a guarantee that the certificate has been logged.

To have high confidence that a certificate has been logged, a TLS client would have to verify that a log entry exists for the certificate. (A typical TLS client, i.e., a browser, would use the log metadata provided by the browser vendor to contact one of more logs, and to verify signed data from each log.) This requires acquisition of additional data from each log, i.e., an inclusion proof (see Section 4.5 of [6962-bis]). Requesting an inclusion proof for a certificate discloses to a log that the RP is interested in the certificate in question. For a browser, this would disclose which web sites a user was visiting (if the web sites provided SCTs), a potential privacy concern for many users. Also, the data acquisition and processing may pose an unacceptable burden for some TLS clients, (e.g., browsers), and thus may not be performed in realtime anyway. Thus a TLS client is NOT REQUIRED to reject a certificate when no

associated SCT is available. Nonetheless, if an SCT is provided with a certificate, its signature SHOULD be verified and the SCT data compared to the certificate in question, if doing so would not impose an undue burden on the TLS client. (Such checks MAY be performed in realtime, or may be deferred. If the checks are deferred and they fail, the client will know that the supplied SCT was bogus. The client SHOULD retain this knowledge and reject a certificate associated with a bogus SCT.) If the signature check fails or the SCT does not correspond to the certificate in question, the certificate is suspect and SHOULD be treated as invalid by the TLS client.

A TLS client that is a browser MAY discriminate against a certificate presented for a web site if the certificate is not accompanied by an SCT, e.g., providing an indication of this via the user interface. The details of such discrimination are outside the scope of this specification. However, such discrimination MUST NOT cause the certificate to be treated as revoked/invalid, until such time as an incremental deployment strategy (that is backwards compatible) is defined and approved by the IETF.

3.6. Auditors

Auditors perform checks intended to detect mis-behavior by logs. There are four log behavior properties that Auditors check:

1. The Maximum Merge delay (MMD)

2. The STH Frequency Count

3. The append-only property

4. The consistency of the log view presented to all query sources

The first three of these checks are easily performed using existing log interfaces and log metadata. The last check is more difficult to perform because it requires a way to share log responses among a set of CT elements, perhaps including browsers, web sites, Monitors, and Auditors, e.g., so-called gossiping [Gossip]. There is as yet no standard for gossiping and thus the last check is NOT required of Auditors at this time.

3.6.1. Checking MMD, STH Frequency Count and Append-only property

Checking that a log is behaving correctly with regard to MMD, STH Frequency Count and Append-only property SHOULD be performed using the algorithm described in Appendix A:

1. The MMD for a log is the maximum time that may elapse between the time that an SCT is issued and a log entry is created. When an Auditor executes the algorithm in Appendix A, Step 7 enables it to detect when the MMD has been exceeded for the certificate append that triggered the new STH. The Auditor's polling period SHOULD be chosen to be small relative to the MMD in order to maximize the chance of successful detection of an MMD violation.

2. To prevent the use of an STH to identify an individual log client, a log MUST NOT generate an STH more frequently than is declared in the log metadata. To verify that a log is not violating this guarantee, when an Auditor executes the algorithm in Appendix A, Step 5 enables it to determine how long it has been since the STH changed and to detect if this period is shorter than the minimum required. The Auditor's polling period SHOULD be chosen to be more frequent than the minimum frequency in order to maximize the chance of successful detection of too frequent generation of STHs.

3. In order to verify the append-only property, an Auditor executes the algorithm as described in Appendix A.1.

3.6.2. Checking for Consistency of Log Views

In order for an Auditor to verify that a log provides a consistent view to all query sources, the Auditor needs to see the results of queries to the log from a broad range of requesters. In principle this could be accomplished using a gossip protocol that has the following constraints:

1. TLS clients are not expected to interact directly with a Log for performance and privacy reasons (see Section 3.5).

2. TLS clients generally do not communicate directly with one another (with a few exceptions). As such, a gossip protocol would be easier to deploy if it does not rely on direct communication among TLS clients.

3. If TLS clients have to acquire and distribute CT information about the web sites they visit, this should not overburden the browsers, Subject web sites, or Logs.

4. There needs to be a mechanism for Auditor(s) to inform Monitors (and maybe browser vendors) about mis-behaving logs. The Auditors could be standalone entities selected by Monitors and browsers, (more properly, browser vendors), as a way to obtain information about misbehaving logs. Alternatively, these parties could operate their own Auditors.

5. Browser vendors need to be able to update the blacklists of mis-issued certificates and the bad-log-lists used by their browsers.

4. Security Considerations

CT is a system created to improve security for X.509 public key certificates, especially in the Web PKI context. An attack analysis [draft-trans-threat-analysis] examines the types of attacks that can be mounted against CT, to effect mis-issuance, and how CT addresses (or fails to address) each type of attack. That analysis is based on the architecture described in this document, and thus readers of this document are referred to that one for a thorough discussion of the security aspects of CT. Briefly, CT logs represent a viable means of deterring semantic mis-issuance of certificates. Monitors are an effective way to detect semantic mis-issuance of logged certificates. The CT architecture enables certificate Subjects to request revocation of mis-issued certificates, thus remedying such mis-issuance. Residual vulnerabilities exist with regard to some forms of log and Monitor misbehavior, because the architecture does not include normative means of detecting such behavior. The current design also does not ensure the ability of Monitors to detect syntactic mis-issuance of certificates. This is because provisions for asserting the type of certificate being issued, for inclusion in an SCT, have not been standardized.

5. IANA Considerations

<TBD>

6. References

6.1. Normative References

[Merkle] Merkle, R. C. (1988). "A Digital Signature Based on a Conventional Encryption Function." Advances in Cryptology - CRYPTO '87. Lecture Notes in Computer Science 293. p. 369

[6962-bis] Laurie, B., Langley, A., Kasper, E., Messeri, E., and R. Stradling, "Certificate Transparency," draft-ietf-trans-rfc6962-bis-08 (work in progress), July 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, June 2013.

[RFC6961] Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension," RFC 6961, June 2013.

6.2. Informative References

[draft-trans-threat-analysis] Kent, S., "Attack Model for Certificate Transparency," draft-ietf-trans-threat-analysis-01 (work in progress), June 2015.

[Gossip] Nordberg, L., Gillmore, and Ritter, T., "Gossiping in CT," draft-ietf-trans-gossip-00 (work in progress), August 2015.

[Auditor] ?? work in progress.

[Monitor] ?? work in progress.

7. Acknowledgments

Some of the text included in this document (including the algorithms described in Appendices A and B), was produced by B. Laurie, A. Langley, E. Messeri, and R. Stradling in earlier versions of [6962-bis]. It has been extracted and edited for use here.

Appendix A. Log Checking Algorithms (Normative)

This appendix specifies nominal algorithms for use in performing various checks based on log data. An Auditor, Monitor, or TLS client, performing a specified check MUST implement an algorithm equivalent to the one described here, i.e., an algorithm that yields the same results when supplied with the same inputs. These algorithms were developed by Ben Laurie, et al., and initially included in the document that has now become the log specification [6962-bis].

A.1. Append-only Check

This is a check performed by an Auditor to verify that a log is operating in a fashion consistent with the "append-only" requirement (see Section 3.6 above).

1. Fetch the current STH (see Section 4.3 of [6962-bis]).

2. Verify the STH signature.

3. Fetch all the entries in the tree corresponding to the STH (see Section 4.7 of [6962-bis]).

4. Confirm that the tree constructed from the fetched entries produces the same hash as that in the STH.

5. Fetch the current STH again; repeat until the STH changes.

6. Verify the STH signature.

7. Fetch all the new entries in the tree corresponding to the STH. If they remain unavailable for a period beyond the MMD for this log then this should be viewed as misbehavior on the part of the log.

8. Either:

    1. Verify that the updated list of all entries generates a tree with the same hash as the new STH.

    Or, if the Auditor is not keeping a local cache of all entries from this log:

    1. Fetch a consistency proof for the new STH with the previous STH (see Section 4.4 of [6962-bis]).

    2. Verify the consistency proof.

    3. Verify that the new entries generate the corresponding elements in the consistency proof.

9. Go to Step 5.

A.2. Inclusion Proof Verification

This algorithm is performed by a log client that has received an "audit_path" and "leaf_index" and wishes to verify inclusion of an input "hash" for an STH with a given "tree_size" and "root_hash". It demonstrates that the "hash" was included in the "root_hash".

1. Set "fn" to "leaf_index" and "sn" to "tree_size - 1".

2. Set "r" to "hash".

3. For each value "p" in the "audit_path" array:

   If "LSB(fn)" is set, or if "fn" is equal to "sn", then:

    1. Set "r" to "HASH(0x01 || p || r)"

    2. If "LSB(fn)" is not set, then right-shift both "fn" and "sn" equally until either "LSB(fn)" is set or "fn" is "0".

   Otherwise:

      Set "r" to "HASH(0x01 || r || p)"

   Finally, right-shift both "fn" and "sn" one time.

4. Compare "r" against the "root_hash". If they are equal, then the log has proven the inclusion of "hash".

A.3. Verifying consistency between two STHs

This algorithm is used by an Auditor to establish that two STHs represent valid states for a log, consistent with the tree sizes indicated. The algorithm assumes that the Auditor has acquired an STH "first_hash" for tree size "first", an STH "second_hash" for tree size "second" where "0 < first < second", and has received a "consistency" array that they wish to use to verify both hashes.

1. If "first" is an exact power of 2, then prepend "first_hash" to the "consistency" array.

2. Set "fn" to "first - 1" and "sn" to "second - 1".

3. If "LSB(fn)" is set, then right-shift both "fn" and "sn" equally until "LSB(fn)" is not set.

4. Set both "fr" and "sr" to the first value in the "consistency" array.

5. For each subsequent value "c" in the "consistency" array:

   If "LSB(fn)" is set, or if "fn" is equal to "sn", then:

    1. Set "fr" to "HASH(0x01 || c || fr)"

       Set "sr" to "HASH(0x01 || c || sr)"

    2. If "LSB(fn)" is not set, then right-shift both "fn" and "sn" equally until either "LSB(fn)" is set or "fn" is "0".

   Otherwise:

      Set "sr" to "HASH(0x01 || sr || c)"

   Finally, right-shift both "fn" and "sn" one time.

6. After completing iterating through the "consistency" array as described above, verify that the "fr" calculated is equal to the "first_hash" supplied and that the "sr" calculated is equal to the "second_hash" supplied.

A.4. Verifying log root hash using log entries

This algorithm is used by any log client to verify that an STH (of "tree_size") for a log is consistent with a complete list of leaf input "entries" from "0" up to "tree_size - 1".

1. Set "stack" to an empty stack.

2. For each "i" from "0" up to "tree_size - 1":

    1. Push "HASH(0x00 || entries[i])" to "stack".

    2. Set "merge_count" to the lowest value ("0" included) such "LSB(i >> merge_count)" is not set. In other words, set "merge_count" to the number of consecutive "1"s found starting at the least significant bit of "i".

    3. Repeat "merge_count" times:

1. Pop "right" from "stack".

2. Pop "left" from "stack".

3. Push "HASH(0x01 || left || right)" to "stack".

3. If there is more than one element in the "stack", repeat the same merge procedure (Step 2.3 above) until only a single element remains.

4. The remaining element in "stack" is the Merkle Tree hash for the given "tree_size" and should be compared by equality against the supplied "root_hash".

Appendix B. SCT Transmission (Normative)

A TLS-enabled web server that supports CT MUST convey SCT data corresponding to at least one certificate in the chain via the TLS handshake. Three mechanisms are defined for conveying the required SCT data and Compliant TLS clients MUST implement all three mechanisms.

1. A TLS extension (Section 7.4.1.4 of [RFC5246]) with type "signed_certificate_timestamp" may be used. This mechanism allows TLS servers to participate in CT without the cooperation of CAs, unlike the other two mechanisms. It also allows SCTs to be updated by the server as needed.

2. An Online Certificate Status Protocol (OCSP) [RFC6960] response extension may be employed, where the OCSP response is provided in the "certificate_status" TLS extension (Section 8 of [RFC6066]), also known as OCSP stapling. This mechanism is already widely (but not universally) implemented. It also allows SCTs to be updated by the sever as needed.

3. An X509v3 certificate extension may be employed. This mechanism allows the use of unmodified TLS servers. However, the included SCTs cannot be changed without re-issuing the certificate. Thus a web cannot readily update the SCT data, e.g., to add SCTs from additional logs. If the SCT embedded in the certificate was issued by a log that is no longer trusted by TLS clients, the server will have to acquire a new certificate.

It is RECOMMENDED that TLS servers send SCTs from multiple logs, in case one or more logs are not acceptable to the TLS clients that visit the server. (A log might become unacceptable if, for example, it has been identified as misbehaving by Auditors, or as the result of a compromise of the log.) Multiple SCTs are represented in an SCT list as follows:

```
opaque SerializedSCT<1..2^16-1>;

struct {
    SerializedSCT sct_list <1..2^16-1>;
} SignedCertificateTimestampList;
```

Here, "SerializedSCT" is an opaque byte string that contains the serialized SCT structure. This encoding ensures that TLS clients can decode each SCT individually (i.e., if there is a version upgrade, out-of-date clients can still parse old SCTs while skipping over new SCTs whose versions they don't understand).

As noted in (1) above, one or more SCTs can be sent during the TLS handshake using a TLS extension with type "signed_certificate_timestamp".

TLS clients that support this extension SHOULD send a ClientHello extension with the appropriate type and empty "extension_data".

TLS servers MUST send SCTs in this TLS extension only to a TLS client that has indicated support for the extension in the ClientHello. The SCTs are sent by setting the "extension_data" to a "SignedCertificateTimestampList".

Session resumption uses the original session information: TLS clients SHOULD include the extension type in the ClientHello, but if the session is resumed, the TLS server is not required to process it or include the extension in the ServerHello.

Appendix C. Matching an SCT to a Certificate

When a TLS client receives an SCT via one of the mechanisms described in Appendix B above, the client needs to match the SCT to a certificate in the certificate chain. For an SCT embedded in a certificate, the matching is trivial: the SCT belongs to the certificate in which it is embedded. In either of the other cases, the client uses the following algorithm (or an equivalent algorithm that produces the same results in all cases).

For each certificate in the certificate chain, starting with the trust anchor and proceeding down to the TLS server's end entity certificate:

1. Copy the certificate's tbsCertificate field.

2. If the tbsCertificate copy contains a redacted labels extension:

    1. For each DNS-ID in the tbsCertificate copy:

        1. Determine the number of labels to redact, X. For the Nth DNS-ID, the number of labels to redact is either the Nth INTEGER in the redacted labels extension (if that extension has N or more INTEGERS) or the last INTEGER in the extension (if the extension has fewer than N INTEGERS).

        2. For each of the leftmost X labels in the DNS-ID: if the label is not the wildcard label ("*"), replace the label with "?".

    2. If the tbsCertificate copy contains a CN-ID (which MUST match the first DNS-ID), change this CN-ID to be equal to the first DNS-ID as (potentially) modified above.

3. In the tbsCertificate copy, remove the SCT list extension (if it is present).

4. Compare the (potentially) modified tbsCertificate copy against the tbs_certificate field in the SCT's signed_entry. If they are bytewise equal, then this is the certificate that the SCT matches, and this algorithm is finished.

Authors' Addresses

Stephen Kent
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: kent@bbn.com

David Mandelberg
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: david@mandelberg.org

Karen Seo
BBN Technologies
10 Moulton St.
Cambridge, MA 02138
US

Email: kseo@bbn.com