CTng: Secure Certificate and Revocation Transparency

In God we Trust; Loggers we Monitor

Hemi Leibowitz
Bar-Ilan University, Ramat Gan, Israel

Haitham Ghalwash
University of Connecticut, Storrs, CT

Ewa Syta
Trinity College, Hartford, CT

Amir Herzberg
University of Connecticut, Storrs, CT

Abstract

In this work, we study *Certificate Transparency (CT)*, an important standardized extension of classical Web-PKI, deployed and integrated into major browsers. We evaluate the properties of the published design of CT-v1 (RFC 6962), and identify five major concerns, which persist in drafts for CT-v2. Most significantly, CT-v1 fails to achieve the main goal of the original CT publications, namely security with *No Trusted Third Party* (*NTTP*) and it does not ensure *transparency for revocation status*. Several recent works [1,4,6,10,19,21,25] address some of these issues but at the cost of significant, non-evolutionary deviation from the existing standards and ecosystem.

In response, we present CTng, a redesign of CT. CTng achieves security, including transparency of certificate and of revocation status, with *No Trusted Third Party*, while preserving client's privacy, allowing offline client validation of certificates, and facilitating resiliency to DoS. CTng is efficient and practical, and provides a possible next step in the evolution of PKI standards. We present a security analysis and an evaluation of our experimental open source prototype shows that CTng imposes acceptable communication and storage overhead.

1 Introduction

Public Key Infrastructure (PKI) facilitates the secure use of public keys, which is critical for the security of open distributed systems such as the Internet. Typically, a relying party obtains a public key and validates it using a *certificate* signed by a trusted Certificate Authority (CA). The PKI defines how certificates are issued and revoked (by the CA), and how they are validated (by relying parties). Deployed PKIs mostly follow the X.509 standard [2,9], and the X.509-based PKI is critical to the TLS, SSH, S/MIME and IPsec standards, to name a few. However, there have been numerous *PKI failures* over the years, causing relying parties to rely on public keys violating their policies and needs, and, often, with the corresponding private keys controlled and exploited by attackers. These failures are mostly for Web-PKI, the most common application of PKI. Web-PKI is especially vulnerable since the relying parties (browsers) trust a set of *root CAs*, where each CA is allowed to issue certificates for end-entities and other CAs without any restriction, and in particular, without any domain name restrictions. As a result, each of the root CAs is effectively a *Trusted Third Party*, without any restrictions on its abilities to certify and without mechanisms to detect failures, typically due to a rogue or negligent CA.

There have been multiple proposals and efforts to improve the security of PKI schemes and prevent or reduce such failures, including [1,4–7,10–12,16–21,25,28–33]. We discuss the evolution of PKI schemes in §2 and Table 1.

In this work, we focus on *Certificate Transparency (CT) [16,17,28]*, which is the only 'post-X.509' PKI scheme deployed and used in practice (Web-PKI). CT is an IETF effort spearheaded by Google, whose goal is to provide a *public log* of issued certificates, maintained by entities called *loggers*. This log can allow early detection of rogue certificates, e.g., by the legitimate owner of a certified domain name, and, consequently, accountability of the CAs. CT brings greater scrutiny and oversight to the certificate issuing process and significantly reduces the risks from rogue certificates and rogue or negligent CAs; without CT, detection and accountability have been a serious challenge. CT has been adopted by the popular Google Chrome and Apple Safari browsers, which require that all digital certificates are CT-compliant. There are several deployed CT loggers, run by Google, Cloudflare, Let's Encrypt and others, and many CAs issue CT-compliant certificates.

A major goal of CT, already highlighted in [14,17], is to avoid the assumption of a *Trusted Third Party*. Indeed, pre-CT X.509, especially as deployed in Web-PKI, requires a complete trust in the root CAs. Avoiding the assumption of a trusted CA, or any trusted third party, was eloquently named as the *No Trusted Third Party (NTTP) goal* in [14,17].

CT is being standardized by the IETF, in CT-v1 [16] and CT-v2 [28]. Both CT-v1 and CT-v2 ensure certificate transparency

	X.509 w/CRL	X.509 w/OCSP	CONIKS [21]	CTv1 [16]	CTv2 [28]	ARPKI [25]	CIRT [1]	CTwP [6]	CT-PIR [10, 19]	CTor [4]	**CTng**
Accountability	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Cert. Transparency	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Revocation Trans.	✗	✗	N/A	✗	✗	✗	✓	✗	✗	✗	✓
Non-equivocation	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✗
Client privacy	✓	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓
NTTP	✗	✗	✓	✗	✗	✓	✗	✗	✗	✗	✓
NTTP-revocation	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Offline validation	✓	✗	✗	✗	✗	✓	✗	✗	✗	✗	✓
DoS-Resiliency	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
Evolutionary	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓
Other concerns	Trusted CA		No certs	Trusted Logger			Overhead, complex			Tor	

Table 1: Comparison of relevant PKI schemes; see text for other schemes and for details. Properties include: **Certificate Accountability** (certificate can be traced back to the issuing CA), **Certificate Transparency** (all certificates publicly available), **Revocation-status Transparency** (the revocation-status is publicly available), **Non-equivocation** (no conflicting certificates for the same domain), **Offline validation** (certificate transparency and non-reocation are validated locally, using prefetched data, without online communication), **NTTP** (No Trusted Third Party; separately for revocation), **Client (relying party) privacy** (no exposure of the certificates checked by relying parties), **DoS-Resiliency** (entities can bound the rate of requests), **Evolutionary** (same entities, interactions and ecosystem as of current standards), and other concerns.

but only if the *logger is honest* and for this reason fail to achieve the NTTP goal. Currently, browsers that support CT, rely on redundancy of loggers, i.e., require the logging of each certificate with multiple loggers. However, since the loggers are chosen by the (possibly corrupt) CAs, the security benefits are limited. Furthermore, Google's Chrome implementation requires one of the two loggers to be the one operated by Google itself, making the requirement amount to, basically, requiring clients to trust Google. Many customers probably trust Google, but this is a far cry from the NTTP goal.

CT-v1 and CT-v2 do not ensure *revocation-status transparency (RT)*, enabling the *Zombie certificate attack* which we present in §2.4. Of course, there exist established mechanisms to handle certificate revocation, including both 'online' mechanisms such as OCSP [23], and 'offline' (periodical) mechanisms such as CRLs [9], CRLite [15], Let's Revoke [27], CRLSets [13] and OneCRL [8]. However, *none* of them ensure *transparency of revocation status*; as a result, these revocation mechanisms, and hence also CT-v1 and CT-v2, are vulnerable to attacks by a rogue CA (§2.4). Extensions to CT that handle revocation transparency were proposed [18, 25], but not deployed.

We present the *first systematic study* of the security guarantees of CT (both CT-v1 [16] and CT-v2 [28]). We find that *CT ensures many PKI security properties*. However, two important properties, *certificate transparency* and *revocation status transparency*, hold *only assuming a Trusted Third Party*: the logger for certificate transparency, and the CA for revocation status transparency. Other drawbacks of CT-v1 [16] and CT-v2 [28] include loss of relying party (client) privacy, requiring clients to perform on-line validation checks, and vulnerability to DoS attacks; see Table 1 and §2.

We also present CTng, a further evolution of CT. CTng modifies the current CT design in limited ways, sufficient to ensure the security properties, including certificate and revocation status transparency, *without requiring any trusted third party*, i.e., satisfying the *NTTP* goal. CTng also addresses other drawbacks of current CT designs: it ensures *relying party privacy*, allows clients to *validate certificates locally (offline)*, and facilitates *robustness against DoS attacks*. Furthermore, *CTng is practical*; in particular, it requires only limited, evolutionary changes in the CT and Web-PKI ecosystem and interactions, allowing incremental adoption and it imposes only modest overhead on relying parties (browsers). The evolutionary nature of CTng facilitates reuse of existing modules; in fact, our (open-source) implementation logger reuses Google's logger's code as is, and the total implementation required only 4056 lines of code; see Table 6. To support revocation, CTng uses the efficient Certificate Revocation Vector (CRV) design of [27], inspired by CRLite [15]. CTng extends the CRV design to further provide efficient revocation status transparency. The revocation status transparency mechanisms are independent from the certificate transparency mechanism, allowing for incremental adoption. Our experiments confirm the efficiency of CTng, especially for relying parties. For a realistic deployment of 200 CAs logging with 5 loggers each, and a total of 200 million certificates (Table 5), the size of a daily certificate transparency update is 10-50 kB and the required storage is 3.65-18.25MB, depending on the number of logger updates; the size of the revocation transparency update is 314 kB (1% revoked certificates) and 618 kB (10%, massive revocation event [34]) while the storage is 2.79 MB (1%) and 12.59 MB (10 %), when compressed. See §6 for further details.

Contributions:

- We review the evolution of PKI schemes, focusing on Certificate Transparency, and identify security goals and drawbacks of different schemes (Table 1 and §2).
- We present CTng, an evolutionary extension of CT as defined in [16, 28], with comparable efficiency and complexity. CTng addresses the major issues with current CT: it ensures revocation status transparency and NTTP, preserves privacy, allows offline certificate validation, and facilitates resilience against DoS.
- We present a (simple) *Zombie certificate attack* and explain why current CT is vulnerable to it, motivating the need for revocation status transparency (and CTng).
- We present a performance evaluation of our open-source implementation of CTng, showing its efficiency and scalability.

2 The Evolution of PKI Schemes

This section reviews the evolution and properties of (applied) PKI schemes, focusing on Certificate Transparency (CT) [16, 18], as summarized in Table 1.

2.1 Failures of Classical PKI

The basic construct of PKI is the *public-key certificate* which follows the X.509 standard. A certificate is a statement, digitally signed by a *certificate authority (CA, aka the issuer)*, which contains the *public key pk*, an *identifier* and/or *properties* of the *subject*. The subject is the owner of the corresponding private (secret) key *sk*. The certificate is issued by the CA at the request of the subject, for example, *bank.com*. Before issuing, the CA should validate that the request is *authentic*, i.e., was really sent by *bank.com*. The certificate subject can request to have its certificate revoked (invalidated prior to its expiration date), e.g., if its private key is compromised or if the certificate was improperly or fraudulently issued.

X.509-based certificates rely on *Trusted Third Parties (TTP)*; a relying party should use a certificate only if the certificate was issued and properly signed by a certificate authority (CA) which the relying party trusts. Trust may be direct (such as a root CA) or through a chain of trust (such as using intermediate CAs). Unfortunately, in multiple cases, CAs failed to protect their private keys, failed to validate that the request for certificate was by the legitimate domain owner, or failed in other ways. Prior to CT, there was no efficient and *deployed* mechanism to detect such failures or the resulting fraudulent certificates. Fraudulent certificates could be 'stealthily' abused for a long time, and would only be discovered, rarely, when the targeted relying party suspected the certificates. This problem is aggravated by the fact that in Web-PKI, *any* CA can issue a certificate for *any* domain.

2.2 CT: Vision, Entities and Processes

Certificate Transparency was proposed to ensure *transparency* of certificates, i.e., provide public logs containing all certificates, The logs are maintained as Merkle trees [22] of certificates by entities called *loggers*. Loggers provide a signed attestation for logging a certificate. Relying parties only use certificates which include[1] an attestation signed by a trusted logger. In CT, this attestation is called an *SCT (Signed Certificate Timestamp)* and it is simply a signature of the logger on the certificate and current time, a *promise* to add the certificate to the log. Only later, but at least once every *MMD (Maximal Merge Delay)*, the logger computes the head (digest) of the log, and signs it and the current time producing an STH. In contrast, in CTng, the logger first adds certificates to the tree and computes the STH, and only then sends to the CA the STH, together with a *Proof of Inclusion (PoI)*, attesting that the certificate was included in the log.

CT introduces another type of entity, the *monitors*, whose role is to monitor public logs and detect rogue certificates, and the rogue or faulty CAs that issued them. Ideally, monitors would allow to detect rogue certificates prior to their use in an attack against some relying party, and to quickly address a faulty or rogue CA - if necessary, by removing them from the list of trusted CAs. Clearly, CT defends against rogue CAs, assuming a benign logger. However, what about rogue loggers (and monitors)? Does CT simply move the Trusted Third Party (TTP) assumption from the CAs to loggers and/or monitors? The early CT publications defined a more ambitious goal, called *No TTP (NTTP)*: ensuring security without assuming *any TTP* [14].

To achieve NTTP, CT prescribes, in addition to *logging* and *monitoring*, also *auditing* to validate attestations issued by loggers and *gossiping* to detect inconsistent $STHs$. However, as we explain next, this property is *not* achieved by the currently published variants of CT; in particular, *none* of them include a well-defined gossip mechanism, or identify precise assumptions which are required to achieve security. CTng, on the other hand, clearly defines all CT processes, including auditing and gossip, and our analysis ensures that security is guaranteed under well-defined and reasonable assumptions, finally satisfying the NTTP goal.

2.3 Existing CT Versions and their Limitations

CT is widely adopted in practice, including by two popular browsers (Chrome and Safari) and it is the subject of the IETF

[1]By sending the attestation included *inside* the certificate using the X.509v3 extensions mechanism [9]. CT certificates are compatible with 'legacy' (pre-CT) subjects and relying parties.

Public Notary Transparency (trans) working group, which produced an experimental RFC6962 [16] and is working on a revised version [28]; we refer to these two versions as CT-v1 and CT-v2, respectively. The most significant difference between CT-v1 and CT-v2 is that CT-v1 claims the goal of security against rogue logger (although it does not fully achieve this goal), while CT-v2 explicitly assumes loggers are benign. Neither version, however, achieves security allowing for rogue loggers, i.e., they do not achieve the *NTTP* goal.

CT-v1 attempts to detect rogue loggers by combining three mechanisms: *monitoring, auditing* and *gossiping*. In *monitoring*, the monitors, periodically, request the $STHs$ as well as new certificates from the loggers. *Auditing* is performed by relying parties, when validating a certificate and the associated SCT To audit, the relying party should request the corresponding STH together with a proof of inclusion (*PoI*) from the logger, and then use the *PoI* to validate that the certificate appeared in the Merkle tree whose head was signed in the STH. Finally, relying parties and monitors should *gossip* the $STHs$ among them, detecting if the logger sends conflicting $STHs$.

However, implementations of CT rarely implement these mechanisms; and even when implemented, they do not suffice to ensure NTTP - which are probably the reasons that CT-v2 simply assumes a trusted logger. We first explain why these mechanisms would not suffice to ensure NTTP, then explain why they are often not implemented, and then what is the current approach to deal with the concern of rogue loggers.

Why CT-v1 is not fully implemented? Two important mechanisms of CT-v1 are rarely implemented: gossip and audit. The lack of gossip is due to the fact it was never fully specified, possibly due to performance and implementation challenges[2]. The lack of audit is due to serious concerns regarding *privacy, performance/usability* and *Denial-of-Service (DoS)*. *Privacy* is exposed since, by sending the SCT during audit, the client exposes which website it is visiting; the *performance/usability* concern is due to the fact that the client is supposed to wait for a response from the logger, which can cause significant delay; and this design opens the logger to *DoS* and performance challenges, since it has to respond to requests from an unknown number of (unknown) clients. Note also that audit requires the client to be connected (online), which is not always true.

Why does CT-v1 fail to ensure NTTP? Even if CT-v1 was fully implemented, there are three reasons that it would still fail to ensure NTTP. First, CT-v1 does not ensure revocation-status transparency; hence, a rogue CA may provide misleading certificate-status to allow attack on a relying party (§ 2.4). Second, a rogue logger may provide an SCT to a certificate it never sends to the monitors yet avoid detection: if a relying party audits this SCT, the rogue logger sends a corresponding STH and *PoI*. Even if the relying party gossips this STH with the monitors, the attack will not be detected since the logger uses a different timestamp on this STH from the timestamps of $STHs$ it sends to the monitors. Finally, a logger can simply fail to respond to auditing requests, and/or to requests for a proof of consistency (*PoC*) between two $STHs$.

How does current CT defend against rogue loggers? The Chrome browser provides a limited defense against rogue loggers, using *logger redundancy*[3]. Namely, Chrome requires certificates to come with at least two $SCTs$, one of them from a Google log and one from a non-Google log. The support for multiple $SCTs$ is standardized in CT-v2 [28], although, the standard does not *require* logger redundancy. Safari's implementation requires only *log redundancy*, i.e., multiple $SCTs$ from different logs - but possibly from the same logger. It suffices for one of the logs (or loggers) to operate correctly, for certificate transparency to hold.

However, log/logger redundancy provides limited security. First, it does not provide revocation-status transparency, so relying parties are still vulnerable to a rogue CA (see § 2.4). Second, the logs or loggers are chosen by the CA; log redundancy is even less meaningful, as the different logs may be from the same log operator. This is not very secure against collaborating rogue CA and rogue logger. Note also that some of the loggers are also CAs, and monitors do not necessarily monitor all loggers. Therefore, this assumption may not be much better than the assumption of a trusted CA - except, possibly, that the 'loggers club' may, at least in the present, be more selective than the 'CA club'.

CTng provides an alternative design, which achieves the NTTP goal [14], under what we consider as reasonable assumptions, and also avoids privacy exposure during auditing. CTng is also practical: it does not introduce significant overhead, complexity or compatibility issues and its design was guided by these goals. It also addresses another major shortcoming of current CT: its lack of *revocation status transparency*. Let us now explain the potential risks due to using CT without revocation status transparency.

2.4 Revocation is Broken: the Zombie Certificate Attack

Certificates should be sometimes revoked, i.e., invalidated before their expiration date. Revocation can be for different reasons: the secret key corresponding to the endorsed public key is lost, stolen or otherwise compromised or, perhaps more importantly, the certificate is found to be issued in error or fraudulently. The latter is particularly important as Certificate Transparency provides *retroactive security* only - CT does not prevent fraudulent certificates from being issued, it only allows them to be discovered and the only meaningful response is to have the certificate revoked.

[2]RFC6962, Section 5 states "All clients should gossip with each other, exchanging $STHs$ at least"; this seems to require client-to-client communication, which is hard to implement and may require high overhead.

[3]Chrome policy is at `https://github.com/chromium/ct-policy/blob/master/ct_policy.md`, and Safari's policy is at `https://support.apple.com/en-us/HT205280`.

Revocation is supported already in early versions of X.509, using CRLs and then also using OCSP. However, in spite of some optimizations, the overhead of CRLs is excessive, and most browsers stopped supporting them. Some browsers support OCSP, but due to delay and availability concerns, often allow ‘soft-fail’, i.e., accepting certificates when OCSP response is not received ‘in time’. An alternative is ‘Stapled OCSP’ where the OCSP response is sent by the server, with the Must-Staple extension to ensure ‘hard-fail’; however, this has deployment challenges. As a result, major browsers changed to *prefetch* revocation updates, typically directly from the browser-vendors. See details in see [3, 15, 27]; note that the current prefetch designs imply reliance on a Trusted-Third-Party, i.e., are vulnerable to a rogue CA and/or rogue vendor.

This problem is not addressed by the current design and deployments of CT. Indeed, neither CT-v1 nor CT-v2 ensure *revocation status transparency*. Hence, we must rely on certificate authorities to honestly attest to the certificates revocation status. This is a problematic assumption as the motivation behind CT is based on the fact that we *cannot* trust CAs to honestly issue certificates. If that is the case, why would we trust them to honestly revoke certificates, i.e., report correct revocation status?

The lack of revocation status transparency enables a rogue CA to launch the following simple yet quite powerful attack, which we call the *Zombie certificate attack*, where the certificate’s revocation status might be inconsistently presented to relaying parties in two ways. A rogue CA can mislead some victim relying party into trusting and using a revoked certificate, whose private key is known to the attacker, by providing a rogue attestation that the certificate is not revoked. In a different attack, consider a certificate that was *not* revoked A rogue CA can provide a misleading attestation to a particular client (e.g., browser), as if that certificate was revoked. This may cause the browser to reject the (valid) certificate, preventing the use of the corresponding website or service (a form of censorship), or forcing the client to connect to another service.

In [18], Laurie describes a high-level design for *Revocation transparency*, however, it lacks details, and suffers from severe performance concerns. These concerns, and a more complete and efficient design, are presented in CIRT [25]. However, CIRT does not provide security against rogue loggers, i.e., does not ensure NTTP, and does not address other concerns such as client-privacy and resiliency to DoS. Both designs also fail to address the efficiency and availability concerns that motivate browsers to avoid relying on CRLs and online OCSP queries.

CTng provides an efficient and practical solution to revocation, ensuring transparency; the main innovation is that relying parties (e.g., browsers) prefetch revocation status information from a *monitor*, using the efficient CRV encoding of [27]. To ensure NTTP without requiring relying-parties to fetch or verify multiple signatures, CTng relies of thereshold signatures.

3 CTng: Goals and Model

This section describes our goals, system model and adversary model.

3.1 CTng Goals

Evolutionary extension to Web-PKI and CT. A principle we adopted is to augment, rather than to replace, the existing Certificate Transparency and Web-PKI, minimizing changes and proceeding with those which appear unavoidable to achieve our security goals. New or significantly revised security mechanisms face unavoidable adoption challenges and therefore, minimizing the potential disruption is critical. Indeed, CT itself was carefully designed to limit changes to the Web-PKI ecosystem, such as the introduction of loggers and monitors. CTng does not introduce any additional entities, and requires only modest changes to the roles and processes of the existing CT entities.

In particular, CTng does allow the use of the existing CT logger, as already deployed, and only requires to *add* a separate *revocation status logger* module. Because the certification log does not change, CTng is also compatible with existing monitors.

Preserve Web-PKI and CT security properties. As a part of the evolutionary approach, CTng should provide all existing security properties of Web-PKI and CT, such as accountability and transparency of certificates.

Ensure revocation status transparency. CTng should provide revocation status transparency to ensure integrity of the revocation mechanism, even if a rogue CA sends incorrect revocation status information. In particular, CTng needs to protect against the Zombie certificate attack.

Relying-party Privacy. CTng should not compromise the privacy of the relying parties. In particular, CTng will not expose which certificates are used and/or audited by relying parties, even to an attacker who controls the corresponding CA, logger and monitors.

No Trusted Third Party (NTTP). A major goal for CTng is to achieve NTTP security, i.e., achieve its security properties, including certificate transparency and revocation status transparency, with no trusted third parties. More precisely, we only assume that there are at most f rogue entities, where CTng should be efficient for significant values of f.

Offline client. CTng allows clients to validate certificates *offline*, i.e., without any communication (except for receiving the certificate). This has few advantages, but let us focus on one: avoiding a delay or failure due to waiting for responses; in reality, this issue can be crucial as delays can be significant and failures may occur for different reasons. The importance of supporting offline clients is amply illustrated by the adoption, and later abandonment, of the OCSP protocol for checking certificate (revocation) status [13,23].

DoS-resiliency. Most PKI designs require servers to provide online services to arbitrary, unknown clients. For example, OCSP requires CAs to send (signed) responses to clients, and both CT-v1 and CT-v2 require loggers to send audit responses, containing the STH and PoI. Furthermore, to reduce (not eliminate) the privacy exposure, these requests are sent over a secure connection (TLS). All of these allow rogue clients to perform DoS attacks by overloading the servers with requests. This issues can also translate to DoS on the website, if the client would 'hard-fail' upon not receiving the response, which can also be due to a DoS on the client's communication. (If the client 'soft-fails', this issue allows circumvention of security.)

Efficiency. CTng should be as efficient for subjects, relying parties, CAs and loggers as CT-v1 and CT-v2. Monitors in CTng will perform additional functions compared to monitors in CT-v1 and CT-v2, however, these services involve minimal overhead, typically negligible compared to the 'classical' monitoring service.

Efficiency, here, refers to the computational resources, communication overhead, and to the total delay per operation. In particular, CTng should ensure **liveness,** i.e., every service should be completed within bounded (and minimal) time.

3.2 System Model and Assumptions

CTng considers five types of entities: *CAs, loggers, monitors, subjects* and *relying parties*.

We assume the standard Web-PKI model, with a set of *root (anchor) CAs* trusted by relying parties. We assume that relying parties are initialized with the public keys a set of loggers and monitors, and monitors are initialized with the public keys and of other monitors. For convenience, we consider only a single log for each logger, although CTng can be trivially adjusted to allow multiple logs by the same logger. As discussed earlier, we present CTng using and extending the CRV revocation mechanism of [27]; for simplicity, we also consider only a single CRV per CA.

Loggers periodically (at least every MMD) publish a timestamped digest of the certificate log (STH) and (at least every MRD) publish a timestamped digest of the revocation status log (SRH_L). We assume that monitors know the value of the MMD and MRD of each logger.

A challenge for CTng- as for many distributed security services, which require timely services and distribution of information - is that such services require reliable, timely communication and clock synchronization. CTng requires only *loosely-synchronized clocks and bounded-delay communication*. Specifically, we assume that entities have loosely synchronized clocks, i.e., the clock drift between any two clocks is bounded by Δ_{clk} at any time. Further, the communication delay between any pair of entities is bounded by Δ_{com}. We assume that $2\Delta_{clk} + \Delta_{com} \leq MMD$. For simplicity, we adopt the standard simplifying assumption of ignoring computation time and network bandwidth limitations.

We assume that all benign monitors form a connected graph, whose diameter is at most a known bound denoted by d_M. We also assume that each logger is monitored by at least $f+1$ honest monitors, where f is the known bound on the number of malicious monitors. This requirement can be easily ensured by having at least $2f+1$ monitors watching over each logger. These assumptions are summarized by the following definition.

Definition 1 (Monitors initialization and connectivity)**.** *Logger L is* $(d_M, f, t_{IM})-$*monitored in an execution, if:*

1. *L is monitored by* $f+1$ *or more benign monitors, since time* t_{IM} *or earlier.*
2. *Let* V_B *be the set of benign monitors and* E_B *be the set of edges connecting gossiping pairs of benign monitors. Then* (V,E) *is a connected graph whose diameter is at most* d_M.

3.3 Adversary Model and Assumptions

In order to ensure NTTP-security, we allow the attacker to completely control an arbitrary set of up to f monitors as well as an arbitrary set of CAs and loggers, and, of course, subjects and relying parties. Further, we assume that the attacker cannot interfere with communication between honest entities. This assumption is important since transparency requires loggers to respond to requests and provide logs in a timely fashion. If we allow the adversary to interfere with communication, we cannot identify rogue loggers who fail to keep up to their commitments (responding to requests).

We assume that the adversary is computationally bounded, and the standard security assumption regarding cryptographic mechanisms used by CTng. Specifically, we assume that we use an existentially-unforgeable digital signature scheme, Merkle trees with the standard collision-resistance and proof of inclusion properties, and a collision-resistant hash function.

4 CTng: Design

CTng consists of four main interactions, each marked by a different color in Figure 1: issuing certificates (§4.2), revoking certificates (§4.3), monitoring and gossip (§4.4), and, finally, periodic updates ('prefetch'), allowing relying parties to locally confirm that a received certificate is transparent and non-revoked (§4.5). Before we present each of these interactions, we begin with a high-level overview (§4.1). Table 2 presents notations used in figures and text.

Simplification: For clarity of exposition, in the figures and discussion in this section, we simplify and assume perfectly synchronized clocks ($\Delta_{clk} = 0$). However, our analysis allows bounded drift, i.e., $\Delta_{clk} > 0$; see §5. We also assume one CA

	Notation	Meaning	Usage
CT	STH	Signed Tree Head	Signature by logger over the 'head' (root/digest) of the certificate transparency log. See Equation 1.
	PoI	Proof of Inclusion	Verifiable proof that a given certificate was included in a log.
	MMD	Maximum Merge Delay	Maximum time interval between the issuing of consecutive $STHs$ by the same logger.
	SCT	Signed Certificate Timestamp	Logger's signature on a certificate and current time. Serves as the logger's commitment to add the certificate to the log within MMD. Essential in CT-v1, optional in CTng.
	STH_{M_i}	Partial signature on STH by monitor i	Endorsement of STH by an individual monitor i.
	STH_M	Complete signature on STH	Endorsement of STH by at least f+1 monitors.
RT	CRV	Certificate Revocation Vector	Bit vector maintained by each CA, where each bit represents the revocation status of a certificate issued by that CA. Each certificate has a unique identifier which maps it to a specific bit in a CRV.
	ΔCRV	Difference from previous CRV	Bit vector indicating certificates revoked since the last CRV.
	MRD	Maximum Revocation Delay	Maximum time interval between the issuing of consecutive $CRVs$ by the same CA; typically a day.
	SRH_{CA}	Signed Revocation Hash (by CA)	Signature of a CA over the revocation status of certificates issued by the CA. See Equation 3.
	SRH_L	Signed Revocation Hash (by logger)	Signature of logger L over revocation status of certificates issued by a specific CA. See Equation 4.
	SRH_{M_i}	Partial signature on SRH_L by monitor i	Endorsement of SRH_L by an individual monitor i.
	SRH_M	Complete signature on SRH_L	Endorsement of SRH_L by of at least f+1 monitors.
Time	Δ_{clk}	Clock synchronization bound	Maximum clock drift assumed between any two clocks at any time.
	Δ_{com}	Communication delay bound	Maximum communication delay between any pair of entities.
	t^I_ϕ	Initialization time of entity ϕ	Defines when entity ϕ was initialized. Times are given on the entity's own clock.
Security	PoM	Proof of Misbehavior	Verifiable proof showing that an entity is corrupted, e.g., two conflicting $STHs$ issued by the same logger.
	NTTP, f	No Trusted Third Party	Security goal: security properties are achieved even if up to f entities are corrupt.
	d_M	Diameter of benign monitors	Maximal diameter of the graph of benign monitors, with edges for gossiping-pairs.

Table 2: A list of notations used throughout this work.

and one logger; the design trivially extends to many CAs and loggers.

4.1 Overview of CTng

As can be seen in Table 1, CTng is the first PKI design to ensure security without assuming Trusted Third Parties (NTTP). CTng is also the first to achieve many other combinations of important features, e.g., certificate transparency with DoS-resiliency and revocation status transparency with *both* client privacy and offline validation.

Importantly, these significant security advantages of CTng require only modest, *evolutionary* changes from the existing, deployed PKI designs. The most significant change, especially compared to the previous versions of CT, is that CTng allows relying parties to validate both transparency and non-revocation *offline*, i.e., without any PKI specific communication at the time the certificate is evaluated (e.g., when establishing a TLS connection) and only requiring *periodical, offline updates from the monitors* (steps P.1 and P.2). The offline validation preserves client's privacy, avoids the delay and unavailability concerns due to online connectivity, and has modest communication and storage requirements.

Specifically, CTng ensures transparency by having relying parties accept certificates *only* with a PoI that can be validated using a verified STH. A verified STH is one received in periodical updates from a monitor - in the past, or, optionally, in the near future. Similarly, relying parties maintain fresh *Certificate Revocation Vectors ($CRVs$)* for each CA, by periodically downloading from a monitor updated $\Delta CRVs$ and $SRHs$ allowing validation of the $\Delta CRVs$. Proposed in [27], a CRV is a vector containing one bit for each certificate issued by the CA; the bits of revoked certificates are turned on. A ΔCRV is a compressed vector, with only bit for newly revoked certificates turned on.

Both $STHs$ and $SRHs$ are signed by a set of $f+1$ monitors, using a threshold signature ([26]); this suffices to ensure NTTP for transparency of both certificate ($STHs$ with $PoIs$) and revocations ($SRHs$ and $\Delta CRVs$). Overhead is minimal: communication and validation of only one signature per validation, received from any monitor (ensuring availability and robustness to DoS).

Issuing and logging certificates. A subject (step I.1) requests a CA to issue a new certificate. If the request is valid, the CA generates a certificate and submits it to a logger (step I.2). The logger, within MMD, responds (step I.3) with the periodical Signed Tree Head (STH) and the $PoIs$ for the certificates issued (since last STH). The STH is a signed digest of a Merkle tree containing *only certificates logged since previous* STH; this suffices to ensure transparency, since monitors and relying parties maintain $STHs$ (until corresponding certificates expire), and $STHs$ are numbered sequentially, ensuring consistency. The use of these much-smaller trees improves efficiency[4] (esp. of $PoIs$ and their validation), e.g., compared to CT-v1.

The CA returns the complete certificate, with STH and PoI, to the subject (step I.4). The subject can optionally request a monitor to monitor logs for its id, notifying the subject if any relevant, suspicious certificates are found.

Revoking certificates and logging certificate status. If necessary, the subject requests to have its certificate revoked (step R.1). The CA locally marks the certificate as revoked. Once

[4]CTng supports use of legacy CT-v1 loggers, but then these savings are lost.

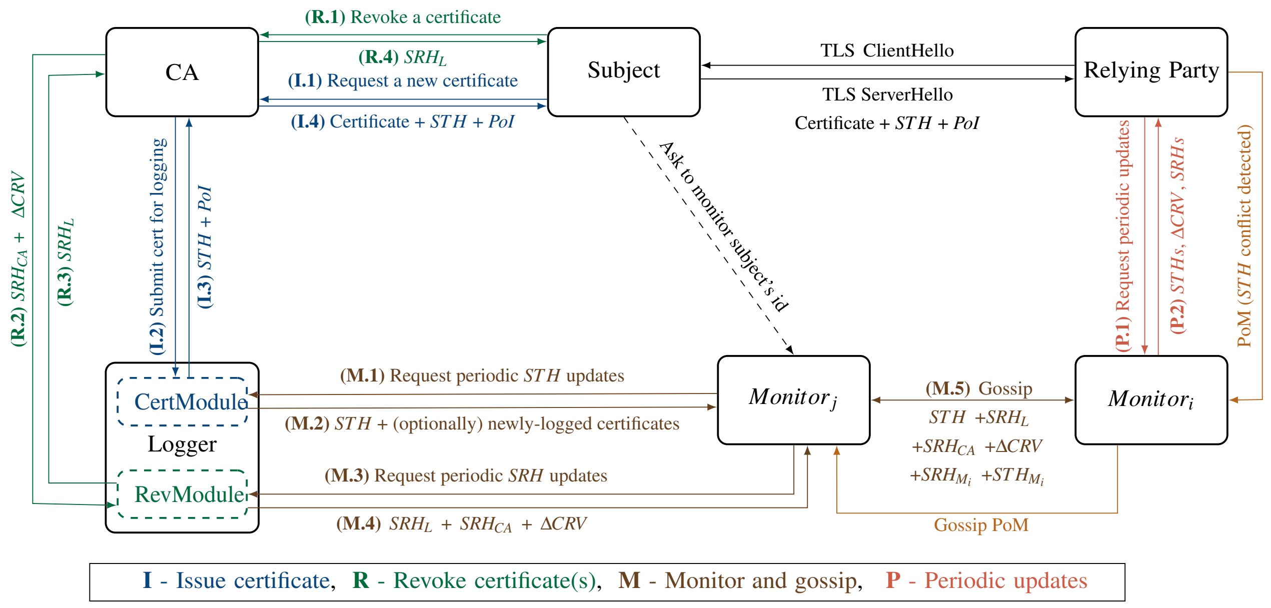

(R.1) Revoke a certificate
(R.4) SRH_L
(I.1) Request a new certificate
(I.4) Certificate + STH + PoI
CA
Subject
Relying Party
TLS ClientHello
TLS ServerHello
Certificate + STH + PoI
Ask to monitor subject's id
(R.2) SRH_{CA} + ΔCRV
(R.3) SRH_L
(I.2) Submit cert for logging
(I.3) STH + PoI
(P.1) Request periodic updates
(P.2) $STHs$, ΔCRV, $SRHs$
PoM (STH conflict detected)
CertModule
Logger
RevModule
(M.1) Request periodic STH updates
(M.2) STH + (optionally) newly-logged certificates
(M.3) Request periodic SRH updates
(M.4) SRH_L + SRH_{CA} + ΔCRV
$Monitor_j$
$Monitor_i$
(M.5) Gossip
STH +SRH_L
+SRH_{CA} +ΔCRV
+SRH_{M_i} +STH_{M_i}
Gossip PoM
I - Issue certificate, R - Revoke certificate(s), M - Monitor and gossip, P - Periodic updates

Figure 1: CTng: entities and interactions. See §4.1 for an overview and Table 2 for notation.

per *MRD*, each CA sends a *revocation update* to the logger (step R.2); CTng uses the efficient *Certificate Revocation Vector (CRV)* design from [27], and therefore sends the compact (compressed) ΔCRV, and the SRH_{CA} attesting for all revocations done so far. The logger may respond with SRH_L, its attestation of revocation transparency; and the CA may return the SRH_{CA} and/or SRH_L to the subject (steps R.3, R.4).

Monitoring and gossip. Monitors, in CTng, may provide oversight of certificates in logs, typically as requested by clients; and also provide transparency and revocation updates to relying parties, allowing them to validate certificates. To ensure NTTP, each logger is *monitored* by at least $f+1$ *benign monitors* (or, by $2f+1$ monitors, out of whom f can be faulty). Every *MMD*, each monitor requests *certificate log updates ($STHs$)* from each logger it monitors (step M.1); often, but not always, monitors will also ask for the certificates issued (from last update). The logger responds with all $STHs$ issued since the last update, and, if requested, also with the logged certificates. Similarly, every *MRD*, each monitor requests *revocation status updates* ($\Delta CRVs$ and $SRHs$) from each logger (step M.3). Each logger responds with the revocation updates ($SRHs$ and $\Delta CRVs$) from all of its CAs (step M.4).

Upon receiving an update, the monitor (step M.5) validates it and gossips it with other monitors. If a logger equivocates (by sending conflicting $STHs$ or $SRHs$), the monitors quickly detect it. If conflicts are found, the monitors gossip a *Proof of Misbehavior (PoM)*; otherwise, each monitor gossips its share of an $f+1$ threshold signature, ensuring NTTP, and upon receiving the necessary shares, combines them to produce the corresponding monitor threshold signatures: STH_M and SRH_M, used in periodic updates of relying parties.

We next discuss each of these processes in detail.

4.2 Issuing Certificates

We now discuss the process of issuing certificates, including the logging of the issued certificates; this process is illustrated in Figure 2.

The process of issuing a certificate in CTng starts just like in CT. First, a subject sends a request to a CA for a new certificate and the CA validates that the subject has the right to receive the certificate[5], e.g., for TLS certificates, the CA verifies that the subject owns the corresponding domain name.

Let us first describe how the process continues in (current) CT. The CA generates a pre-certificate[6] with a non-critical X.509v3 extension, which we refer to as the *CT extension*, containing one or more $SCTs$, i.e., commitments made by loggers to add the certificate to their logs. Logs are publicly available to monitors and they can be audited using those $SCTs$. When requested by relying parties or any other entity, the loggers *are supposed to* provide the corresponding $STHs$ with $PoIs$, proving that the certificates were indeed (eventually) added to the logs. This design has drawbacks, however. In particular, the above audit process requires that relying parties send their $SCTs$ to the logger, which exposes to the logger the websites visited by the relying party (the browser), and requires browsers to wait for a response. For this reason, the implementations of CT, both in Chrome and Safari do not perform the auditing process; see §2.3 for more details.

[5] As in CT-v1 and other PKI schemes, CTng does not specify the validation process used by the CA.

[6] The CT-extension should contain input from the logger - SCT for CT-v1, and STH and PoI for CTng; the certificate, signed by the CA but with an empty CT-extension, is called a pre-certificate.

CTng also uses the CT extension, similarly kept empty by the CA in the pre-certificate sent to the logger. However, in CTng, the logger 'fills' the CT extension with an STH together with a *PoI*, rather than one or several $SCTs$. Namely, in CTng, certificates include a *proof* that they were logged, rather than just a *promise* that they would be.

We now describe the *logging* process for newly issued certificates. Each logger maintains a log of certificates as a Merkle tree, which is periodically updated by adding certificates submitted by the CAs and generating an STH. To ensure security, logger should *always* produce an STH upon completion of each MMD, even if no new certificates were issued. If STH are *only* issued at end of MMD, then issuing may require up to an MMD. Waiting an MMD for a new certificate should usually be fine, especially that MMD may be quite short (and logger specified). However, loggers may want to issue $STHs$ more frequently, e.g., to allow quick recovery from key exposure or just urgently-issue a new domain. To do so, loggers can issue 'within-MMD-$STHs$'; these are treated just like the 'end-of-MMD' $STHs$ by relying parties (the difference is only that monitors may pick them only at the end of the MMD period). To ensure that all $STHs$ are accounted for, each STH includes a sequential serial number, so monitors and relying parties can confirm they have all $STHs$ issued by a logger until the latest MMD; any other STH from the logger until this time, is a *conflict* and proves that the logger is misbehaving. If a malicious logger produces an STH, fails to disclose it to the monitors but provides it to a subject (through a malicious CA), the relying party will discover this misbehavior, by comparing the STH against the set of published $STHs$ of this logger, which the relying party receives in *periodical updates* from monitors, as described in § 4.5. The discovery will usually be immediate, *before usage*, or, optionally, occur at the next update, and always result in a third-party verifiable *Proof of Misbehavior (PoM)* of the logger.

To ensure efficient, bounded storage and communication resources, CTng limits the maximal number of $STHs$ a logger may issue in a single MMD. This is consistent with CT-v2, which defines a maximum number of $STHs$ per MMD, as one of the required log parameters.

An STH computed by logger L is a tuple:

$$STH_L^{i_P,\, i_S} \equiv (t_L,\, i_P,\, i_S,\, i_C,\, h,\, \sigma) \tag{1}$$

where:

1. t_L is the *timestamp*, i.e., the time on logger L's clock, when the STH was computed; L produces a periodic STH whenever its clock shows $t_L = t_L^I + i_P \cdot MMD$, for every integer i_P, and 'within-MMD-$STHs$' as needed (up to the maximal allowed number). Each STH will have a unique t_L value.
2. $i_P \in \mathbb{N}$ is the number of MMD periods (since logger L started, at t_L^I). For the 'usual' case of STH issued at the end of of the MMD period, we have $t_L = t_L^I + i_P \cdot MMD$; for 'within- MMD-$STHs$', i_P is $\lceil (t_L - t_L^I)/MMD \rceil$.
3. i_S is the sequence number of this STH since the beginning of this MMD period, i.e., since $t_L^I + (i_P - 1) \cdot MMD$; typically, $i_S = 1$.
4. i_C is the total number of entries (certificates) in the log Log, i.e., $i_C = |Log|$, when this STH is issued.
5. h is the 'head' (digest) of the Merkle tree, whose leaves are the pre-certificates in the log Log, *issued since the previous* STH. (In CT-v1, the tree is over all certificates.)
6. $\sigma = Sign_L(t_L \,||\, i_P \,||\, i_S \,||\, i_C \,||\, h)$ is the logger's signature on the STH using the logger's secret signing key.

We assume that all parties are aware of loggers' public parameters (e.g., their urls, public keys, logging policies, etc.), and specifically the time logger L was first initialized, denoted as t_L^I (on L's clock). This value could also be included in the $STHs$ signed by this logger.

4.3 Revoking Certificates

We now discuss the CTng certificate revocation process, as illustrated and summarized in Figure 3. This process involves the subject (initiating the revocation), the CA (performing the revocation) and the logger (logging the revocation).

As mentioned before, CTng uses the efficient *Certificate Revocation Vector (CRV)* structure of [27]. A CRV is a bit vector which uses a single bit to represent the revocation status of a certificate. Each certificate has a unique identifier which maps it to a specific bit in a CRV.

Revoking a certificate. As shown in Figure 3, CTng revocation interaction between subject (the owner of a certificate) and a CA is simple, mostly just as in X.509. Namely, the subject requests to have their certificate revoked and the CA, after authenticating the request using some CA-specific process, marks the corresponding bit in the appropriate CRV it maintains, and then acknowledges processing the revocation.

Let us describe the process as performed by CA $i \in \mathsf{N_{CA}}$, referring to the MRD period as a 'day' (a typical value used by browsers). During day $d-1$, the CA *locally* updates new revocations into $CRV_i(d)$, i.e., the CRV that will be used by relying parties on day d. A subject can confirm that its certificate was indeed revoked by auditing the certificate's revocation status by contacting one of the monitors; the monitors should have the updated value $CRV_i(d)$ 'soon' after the beginning of day d.

Logging revocation status. Recall that we use t_{CA}^I to denote the time when CA i begins running. On $t_{CA}^I + d \cdot MRD$, i.e., beginning of 'day' d of CA, the CA computes $\Delta CRV(d)$, the set of the certificates that were revoked since 'yesterday', i.e., since $CRV(d-1)$:

$$\Delta CRV(d) \quad = \quad CRV(d-1) \oplus CRV(d) \tag{2}$$

From this point, the CA 'freezes' $CRV(d)$, i.e., new revocations received and processed will be reflected in $CRV(d+1)$. Then, the CA generates $SRH_{CA}(d)$, which is the CA's signature over the revocation status information for day d, as:

$$SRH_{CA}(d) = Sign_{CA}(d \,||\, h(CRV(d)) \,||\, h(\Delta CRV(d))) \qquad (3)$$

The CA then sends $\Delta CRV(d)$ and $SRH_{CA}(d)$ to the logger. The logger validates $SRH_{CA}(d)$ by using Equation 2 to compute $CRV(d)$; if valid, the logger generates $SRH_L(d)$, which is the logger's signature over the revocation status of certificates provided by the submitting CA for day d and the CA's signature. Namely:

$$SRH_L(d) \quad = \quad Sign_L(d \,||\, h(CRV(d)) \,||\, h(\Delta CRV(d)) \,||\, h(SRH_{CA} \qquad (4)$$

The logger then sends $SRH_L(d)$ back to the CA and makes the revocation information available to the monitors. If the logger does not receive a (valid) periodic SRH_{CA} from the CA, it produces the following error message instead:

$$SRH_L(d) \quad = \quad Sign_L(d \,||\, \text{no } SRH_{CA}(d) \text{ received}) \qquad (5)$$

4.4 Monitoring and Gossip

CTng significantly expands the role of monitors compare to current CT, e.g., CT-v1. In both, monitors can provide service of monitoring the certificates issued by specified loggers, typically, to detect issuing of certificates of interest, typically, illegitimate or suspect use of a domain name, often as requested by a legitimate owner of a domain. To support this functionality, loggers are expected to provide monitors with the logged certificates and the information necessary to validate the logs for consistency.

If loggers are benign, this suffices to ensure transparency, assuming each logger is monitored by at least one honest monitor; indeed, CT-v2 explicitly assumes benign loggers. However, a rogue logger may misbehave in different ways: it may not cooperate with the monitors, send inconsistent $STHs$ to different monitors, or send $STHs$ with different timestamps to prevent detection of inconsistency. CT-v1 and CT-v2 do not address these issues, i.e., ensure transparency only assuming benign loggers, although there is mention of (incomplete) mechanisms to detect rogue loggers, such as gossip and audit.

In contrast, CTng ensures transparency without assuming benign loggers or any other Trusted Third Party, i.e., achieves the NTTP goal as in [14]. Basically, CTng monitors perform the following tasks: (1) *monitor logs* for newly logged certificates and newly issued revocation status information and *monitor loggers* to ensure that they behave honestly, (2) *gossip* the information learned from loggers with other monitors, and lastly (3) provide periodic *transparency and revocation updates* to relying parties (§ 4.5).

These operations allow to hold loggers accountable. If a rogue logger sends two conflicting $STHs$ to different monitors, monitors will quickly detect it through gossip. If a rogue logger sends two conflicting $STHs$, one to all monitors and one to some victim relying party, then this misbehavior will be detected by the relying party, since relying parties also receive the $STHs$ in periodic updates from the monitors. Furthermore, a pair of conflicting $STHs$ provides a Proof of Misbehavior (PoM), which can be verified by any party, showing that the logger is corrupted, which allows CTng to ensure NTTP-security.

4.4.1 Monitoring

We now focus on the *monitoring* functionality (Figure 4), which involves periodic communication between monitors and loggers. Each monitor periodically contacts each logger it watches, to request the certificate transparency (CT) updates ($STHs$), at least every MMD, and the revocation transparency (RT) updates ($SRHs$ and $CRVs$), at least every MRD. In practice, MMD and MRD may coincide; monitors can also contact loggers more frequently. Monitors may elect to request all newly logged certificates, which is necessary to monitor certificates, e.g., to detect suspect certificates. While only $f+1$ benign monitors are *required* to monitor each logger (and as such, directly contact the logger), all monitors receive each logger's periodic updates, not including certificates, through gossip among the monitors. Each monitor validates the update it receives from the logger.

Validating periodic certificate transparency (CT) updates. Upon request from a monitor for a CT update for a specific MMD period i_P, the logger returns $STH_L^{i_P,*}$, the set of all $STHs$ issued during i_P. Each STH in $STH_L^{i_P,*}$ should be as as described in Eq. (1). The monitor stores the response locally, checks that it is valid: $STHs$ include valid signature (σ), all contain the correct period (i_p) and continuous, sequential serial numbers (i_s), with the lowest i_s being exactly one more than that of the last STH from the previous period ($i_p - 1$), and the total number of $STHs$ in the set not exceeding the maximal allowed number.

If the monitor requested $\{Cert\}_{i_P}$, the certificates issued during i_P, then the monitor stores them locally and verifies that $|\{Cert\}_{i_P}| = i_C$ and that h matches the head of the Merkle tree, whose leaves are certificates included in $\{Cert\}_{i_P}$.

Validating periodic revocation (RT) updates. The periodic RT update consists of $\Delta CRV_{CA}(d)$, which specifies the newly revoked certificates, $SRH_{CA}(d)$, which is the CA's commitment to the revocation information $CRV(d)$ for day d, and $SRH(d)$, which is the logger's endorsement of $SRH_{CA}(d)$. The monitor stores the update locally and validates that the updates are consistent with Eqs. 2-4. Specifically, the monitor computes:

$$CRV_{CA}(d) = CRV_{CA}(d-1) + \Delta CRV_{CA}(d) \qquad (6)$$

and validates that $SRH_{CA}(d)$ and $SRH(d)$ are properly signed by the CA and logger (Eqs. 3 and 4).

4.4.2 Gossip

Monitors gossip with other monitors, by forwarding all $STHs$, $\Delta CRVs$ and $SRHs$ (from loggers or other monitors), as well as all PoMs from relying parties and other monitors, and finally also *accusations of misbehavior* signed by monitors. Gossiping is performed via interactions between monitors, as illustrated in Figure 6.

At least $f+1$ benign monitors, or $2f+1$ arbitrary monitors, should request the 'daily' revocation information $(\Delta CRV(d), SRH_{CA}(d), SRH_L(d))$ directly from the logger. The redundancy ensures that if a logger fails to send periodic updates, then a sufficient number $(f+1)$ of benign monitors can *accuse* the logger of misbehavior; the $f+1$ accusations, are considered a Proof of Misbehavior (PoM). The redundancy also addresses possible communication problems between few monitors and the logger.

Monitors verify that $STHs$ and $SRHs$ that they receive from other monitors via gossip (and directly from relying parties) are consistent with the $STHs$ and $SRHs$ which they received from loggers. The receipt of two *different* $STHs$ with the same sequence numbers or timestamps, or an STH whose combination of period-number (i_P) and timestamps conflict (not as per item 2 below Eq. (1)), or of two different $SRHs$ for the same or overlapping periods, results in a Proof of Misbehavior (PoM) against the issuing logger. Notice we allow multiple different $STHs$ for the same MMD period, but they must have distinct and continuous sequence numbers (see Equation 1). A monitor detecting such conflict also gossips the PoM to all monitors, as well as shares it with relying parties upon their periodic updates.

A monitor M that receives a transparency or revocation update, gossips it immediately. However, then, it *waits* $2 \cdot \Delta_{com} \cdot d_M$, where d_M is the diameter of the graph of benign monitors and connections among them, i.e., enough time for the update to reach all monitors and for any response PoM, or conflicting STH or SRH_L, to reach M. If no conflict or PoM is received, the monitor *endorses* the update by signing it using its share of a threshold signing key using a threshold signature algorithm, e.g. [26]; we denote the resulting signature shares by SRH_{M_i} and STH_{M_i}. Once a monitor receives $f+1$ of these shares, it can combine them to produce the corresponding monitor threshold signatures: STH_M and SRH_M. This can be trusted by the relying parties, since they are confident that benign monitors have been exposed to the same STH or SRH_L.

Let us elaborate, focusing, for example, on the revocations ($SRHs$ and $\Delta CRVs$); the CT updates ($STHs$) are handled similarly. Let (s, v) denote a threshold signature (private, public) key-pair, and let s_i denote the *fragment/share* of s which is known to monitor M_i. Suppose monitor M_i receives the daily revocation information of $CA \in \mathsf{N_{CA}}$ which was logged by logger $L \in \mathsf{N_L}$, i.e., M_i receives $\Delta CRV_{CA}(d), SRH_{CA}(d), SRH_L(d)$. After M_i waited $2 \cdot d_M \cdot \Delta_{com}$ and received no conflicting SRH_L or PoM, then it uses s_i to compute, and then gossip, its *signature share* $SRH_{M_i}(d)$:

$$SRH_{M_i}(d) = Sign_{s_i}(\Delta CRV_{CA}(d) \mid\mid SRH_{CA}(d) \mid\mid SRH_L(d)) \tag{7}$$

When a monitor receives $f+1$ different signature shares, $\{SRH_{M_i}(d)\}_{i=1}^{f+1}$, signed by different monitors, it uses the threshold-signature *combine*() operation and receives the joint threshold signature $SRH_M^{CA}(d)$, which could be viewed as a 'regular' signature using the threshold monitor's signing key s:

$$SRH_M(d) = Sign_s(\Delta CRV_{CA}(d) \mid\mid SRH_{CA}(d) \mid\mid SRH_L(d)) \tag{8}$$

PoM and alerts. When a PoM is detected (or received), the monitor should gossip it with all of its peers. Misbehaving loggers might not cooperate with monitors, and fail to respond properly to all or some specific monitoring requests, hoping to avoid detection. However, if a logger does not reply within $2\Delta_{com}$ to a monitoring request, the monitor detects that the logger is non-responsive. The monitor cannot send a PoM, but it generates and gossips a signed *accusation* message stating that the specific logger is non-responsive. The collection of $f+1$ *accusation* messages, signed by different monitors, is considered equivalent to a PoM. Since we assumed that each logger is monitored by at least $f+1$ benign monitors, and the graph of benign monitors is connected, it follows that a logger that fails to cooperate with the monitors is quickly detected and ignored.

4.5 Periodic Relying Party Update Process

In CTng monitors not only do they *monitor* certificates and revocations, but also provide relying parties with information necessary to validate certificates *offline*, i.e., without depending on any real-time communication. Specifically, every relying party, periodically, requests *transparency updates* from one of the monitors. The updates include newly issued $STHs$, by *every* logger, which ensures certificate transparency, and $\Delta CRVs$ and $SRHs$, which ensure revocation transparency. To ensure resiliency to rogue monitors, i.e., NTTP, while allowing the relying party to connect to request updates from only one monitor (for efficiency), the updates are also signed by the threshold monitors key, i.e., must include the corresponding STH_M and SRH_M. This process as well as how TLS connections are established are illustrated in Figure 5.

CTng design does *not* assume trusted monitors since responses are signed by a threshold of $f+1$ monitors; we only assume that there are no more than f rogue monitors, and that every relying party can query (at least) *one* benign monitor to ensure liveness. For efficiency, relying parties first query one ('favorite') monitor; if it fails, they query f other monitors.

Transparency updates. In CTng, certificates always include, in their CT-extension, an STH and a PoI. The relying party uses the PoI to validate that the certificate is included in the STH; this ensures transparency, as long as this STH was also reported to the monitors. To validate this, efficiently and without requiring online communication, the relying party retains all $STHs$, from all loggers, periodically updated from the monitors. The total storage and communication requirements are modest, as each STH is very compact (e.g., 666 bytes; see Table 4). In practice, relying parties would only need to retain $STHs$ for a period of time corresponding to the maximum lifetime of certificates, which is now capped at 398 days by major browsers [24].

In rare cases, a relying party might need to verify a recently issued certificate whose STH has not yet been included in the last update from the monitors. This situation should be rare, since benign websites, usually, request certificates in advance. Therefore, relying parties can simply refuse such connections. Alternatively, they may accept the certificate, or immediately ask monitors for a 'real-time' update. Note that even such 'real-time' update' may not yet include the period covered by this very-new STH, 'forcing' the relying party to decide whether to 'hard-fail' (refuse the use of the certificate) or to 'soft-fail' (use the certificate anyway).

Whenever the relying party detects conflicting $STHs$- immediately or in an update after receiving a 'fresh' certificate - this 'conflicting' pair of $STHs$, with same sequence number and/or timestamp, 'prove' that the logger has misbehaved, i.e., result in a PoM which the relying party forwards to $f + 1$ monitors. Since this misbehavior is guaranteed to be detected in a relatively short amount of time, it seems unlikely that rogue loggers would perform such attacks.

Revocation updates: $\Delta CRVs$ and SRHs. CTng relying parties also receive *all* of the revocation data for all CAs. For each CA, relying parties download, once per *MRD*, the ΔCRV, which is a (compressed) bitmap indicating certificates revoked since last update, as well as the signatures authenticating the $\Delta CRVs$, i.e., the corresponding *SRHs*. Specifically, this includes: $\Delta CRV_{CA}(d)$, $SRH_{CA}(d)$, $SRH_L(d)$ and $SRH_M(d)$.

The relying party will validate and then update its copy of the *CRV* (Equation 6), which provides most recent revocation status information. The communication overhead of this process is small, since the $\Delta CRVs$ are usually sparse (few new revocations in a single *MRD*), and therefore compress very well. The storage required for the $CRVs$ is also modest: at most, one bit per certificate, even under the unlikely scenario of a very large number of revocations which makes compression inefficient (see Table 5 for details).

Comparison to existing prefetch of revocation information. Our periodic update approach is consistent with the (proprietary) prefetch mechanisms of major browsers [8, 13]. Current browsers download revocations from their vendors; this is also possible in CTng, as each vendor could run its own monitor. By using monitors for both auditing loggers (for transparency) and for receiving revocation information (CRVs), the monitors become service providers to relying parties such as browsers.

Furthermore, the periodic information is signed and therefore can be *cached by untrusted parties* such as ISPs or CDNs, or even distributed by appropriate encoding in DNS records, allowing easy deployment using the existing efficient DNS caching infrastructure.

5 Security Analysis

The basic properties of CTng, such as accountability of a CA for issued certificates, follow immediately from the basic X.509 design, e.g., the fact that certificates are signed. In this section, we focus on properties related to *transparency*, which depend on mechanisms beyond the basic X.509 design.

The analysis includes some 'hairy' aspects related to delays (bounded by Δ_{com}) and to clock synchronization (where the drift is bounded by Δ_{clk}); some of these aspects were glossed over in the design section, for simplicity. In particular, let us introduce assumption and notation.

The MMD and MRD assumptions: the *MMD* and *MRD* periods are much larger than then the maximal delay Δ_{com} and clock-drift Δ_{clk}. Specifically, $MMD > 2 \cdot \Delta_{clk} + (5+4d_M) \cdot d_M \cdot \Delta_{com}$ and $MRD > 2 \cdot \Delta_{clk} + (5+4d_M) \cdot d_M \cdot \Delta_{com}$.

The logger initialization assumption: for simplicity, we assume that loggers are always initialized before any monitor begins monitoring them.

Local time (t^{ϕ}) notation: when referring to times measured on the local clock of an entity ϕ, we identify the entity by superscript, e.g., t^{ϕ}. We do not include superscript for real-time values.

In CTng, once a monitor begins monitoring a logger, it would send requests periodically, again and again: every *MMD* a request for $STHs$ (and optionally certificates), and every *MRD* a request for *SRHs* and $\Delta CRVs$. The following Lemma bounds the maximal time until the next request (of each type), from any given time t_1; this mainly deals with the impact of potentially drifting clocks, taking advantage of Δ_{clk}, the maximal clock bias. The lemma allows for the initialization time of a logger to be specified by the logger as a response to the first query from the monitor (rather than just assuming it's known - a simplification which may be inconvenient in practice).

Lemma 1. *Consider a run of CTng, with a benign monitor M, who begins monitoring logger L (for some domain) at time t_0. Then, for every $t_1 \geq t_0$:*

1. *During $[t_1, t_1 + 2 \cdot \Delta_{clk} + MMD]$, either M sends to L at least one requests for periodic STH (and certificates, if desired) update, or M detects that L is corrupt.*
2. *During $[t_1, t_1 + 2 \cdot \Delta_{clk} + MRD]$, either M sends to L at least one request for periodic CRV and SRH_L update, or M detects that L is corrupt.*

Proof: The periodicity mechanism is identical for the STH requests (every *MMD*) and for the SRH_L and ΔCRV requests (every *MRD*), so it suffice to prove for STH requests.

If $t_1 = t_0$, then M would *immediately* send a monitor request to L. Assume, therefore, that $t_1 > t_0$. If M does not receive the

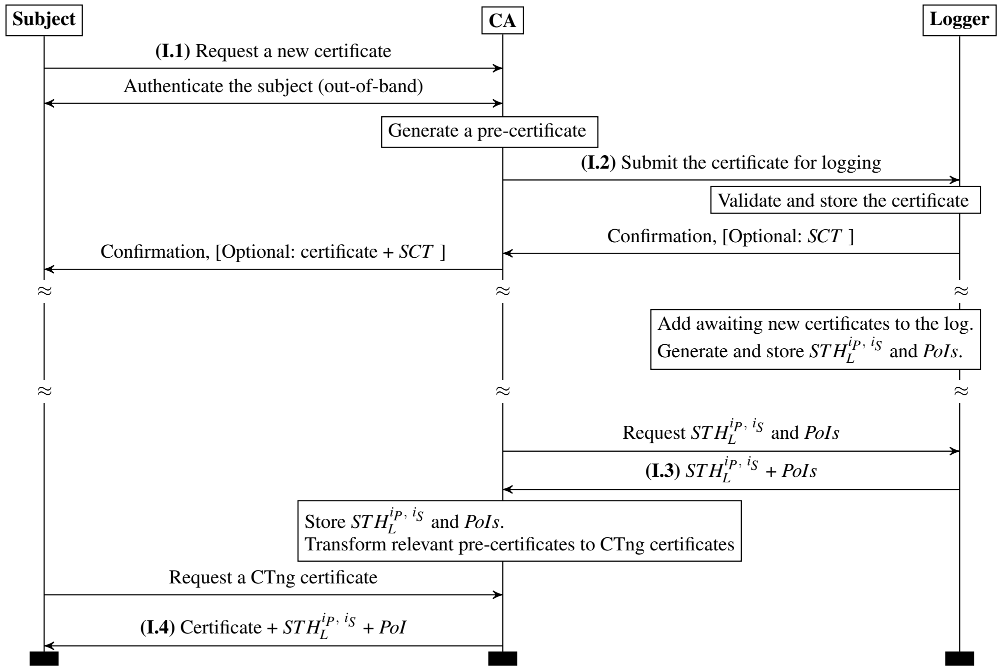

Subject
CA
Logger
(I.1) Request a new certificate
Authenticate the subject (out-of-band)
Generate a pre-certificate
(I.2) Submit the certificate for logging
Validate and store the certificate
Confirmation, [Optional: SCT]
Confirmation, [Optional: certificate + SCT]
Add awaiting new certificates to the log.
Generate and store $STH_L^{i_P, i_S}$ and $PoIs$.
Request $STH_L^{i_P, i_S}$ and $PoIs$
(I.3) $STH_L^{i_P, i_S}$ + $PoIs$
Store $STH_L^{i_P, i_S}$ and $PoIs$.
Transform relevant pre-certificates to CTng certificates
Request a CTng certificate
(I.4) Certificate + $STH_L^{i_P, i_S}$ + PoI

Figure 2: Certificate issuing and logging. **(I.1)** The subject requests a certificate from a CA who verifies the request, generates the certificate if warranted and **(I.2)** submits it for logging. Optionally, the logger can immediately issue an SCT, a promise to log the certificate within a specific timeframe. Periodically, the logger adds newly submitted certificated to its log and generates a new STH. Upon request from a CA, **(I.3)** the logger provides $PoIs$ and STH for the recently submitted certificates. Lastly, **(I.4)** the CA return certificates with the proof of logging to the subjects.

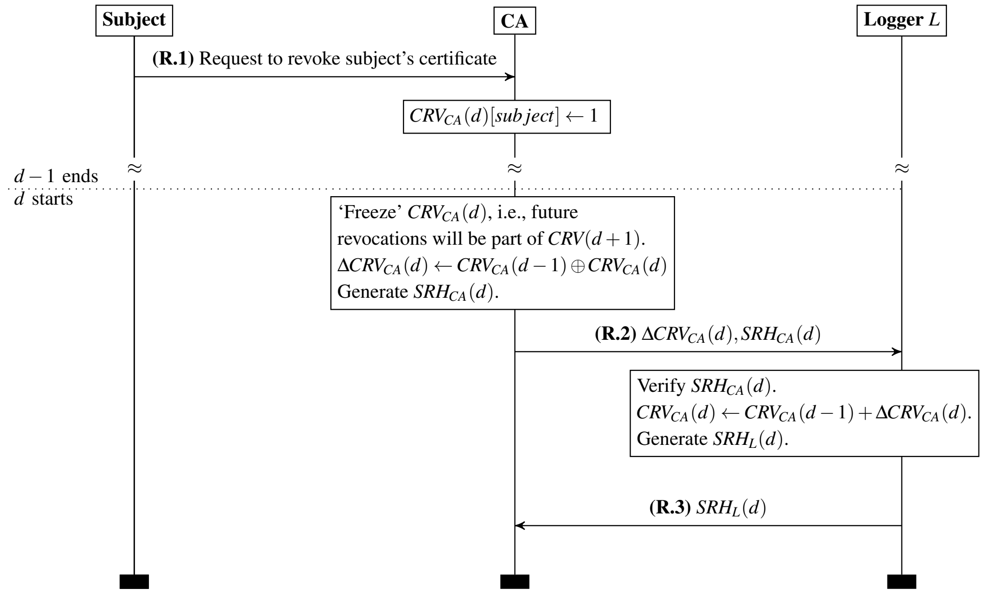

Subject
CA
Logger L
(R.1) Request to revoke subject's certificate
$CRV_{CA}(d)[subject] \leftarrow 1$
$d-1$ ends
d starts
'Freeze' $CRV_{CA}(d)$, i.e., future revocations will be part of $CRV(d+1)$.
$\Delta CRV_{CA}(d) \leftarrow CRV_{CA}(d-1) \oplus CRV_{CA}(d)$
Generate $SRH_{CA}(d)$.
(R.2) $\Delta CRV_{CA}(d), SRH_{CA}(d)$
Verify $SRH_{CA}(d)$.
$CRV_{CA}(d) \leftarrow CRV_{CA}(d-1) + \Delta CRV_{CA}(d)$.
Generate $SRH_L(d)$.
(R.3) $SRH_L(d)$

Figure 3: Revoking certificates in CTng. **(R.1)** The subject requests to revoke its certificate from the CA that issued it, and the CA updates the relevant CRV locally to reflect the revocation. Periodically, the logger generates the ΔCRV and SRH_{CA}, and **(R.2)** submits them to the logger, which verify them and then generates appropriate SRH_L and **(R.3)** sends the STH back to the CA.

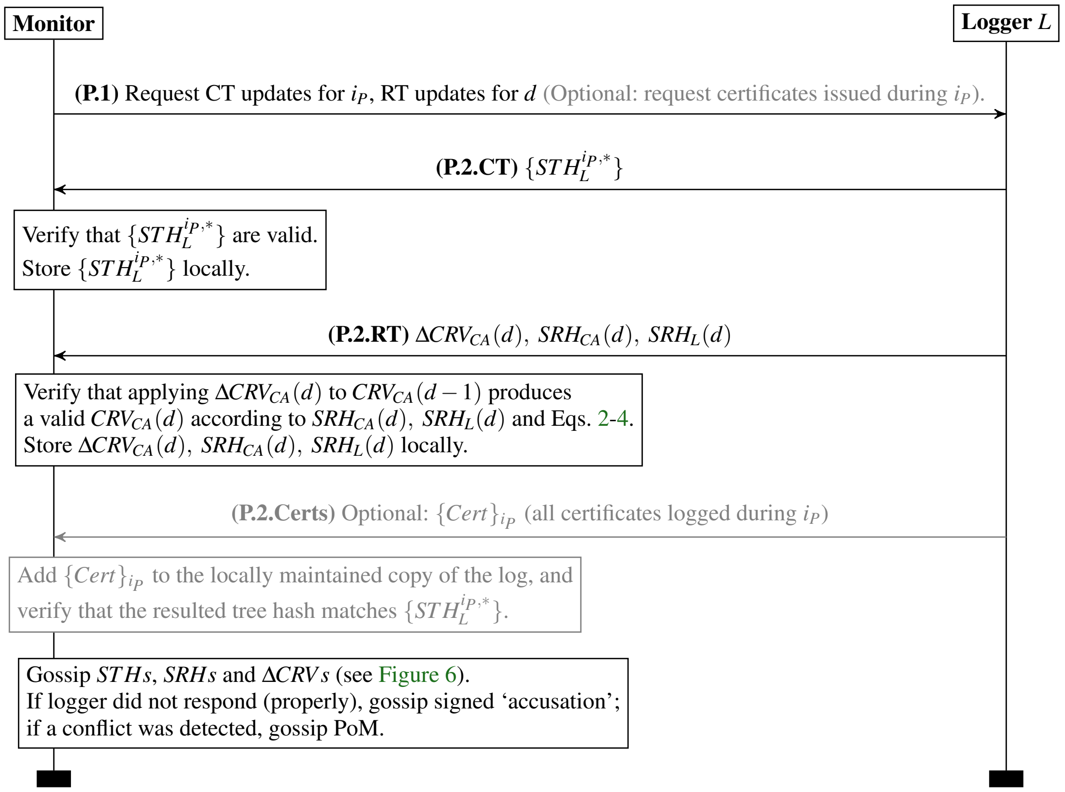

Monitor
Logger L
(P.1) Request CT updates for i_P, RT updates for d (Optional: request certificates issued during i_P).
(P.2.CT) $\{STH_L^{i_P,*}\}$
Verify that $\{STH_L^{i_P,*}\}$ are valid.
Store $\{STH_L^{i_P,*}\}$ locally.
(P.2.RT) $\Delta CRV_{CA}(d)$, $SRH_{CA}(d)$, $SRH_L(d)$
Verify that applying $\Delta CRV_{CA}(d)$ to $CRV_{CA}(d-1)$ produces
a valid $CRV_{CA}(d)$ according to $SRH_{CA}(d)$, $SRH_L(d)$ and Eqs. 2-4.
Store $\Delta CRV_{CA}(d)$, $SRH_{CA}(d)$, $SRH_L(d)$ locally.
(P.2.Certs) Optional: $\{Cert\}_{i_P}$ (all certificates logged during i_P)
Add $\{Cert\}_{i_P}$ to the locally maintained copy of the log, and
verify that the resulted tree hash matches $\{STH_L^{i_P,*}\}$.
Gossip $STHs$, $SRHs$ and $\Delta CRVs$ (see Figure 6).
If logger did not respond (properly), gossip signed 'accusation';
if a conflict was detected, gossip PoM.

Figure 4: Periodic monitoring between monitors and loggers assuming $MMD = MRD$.

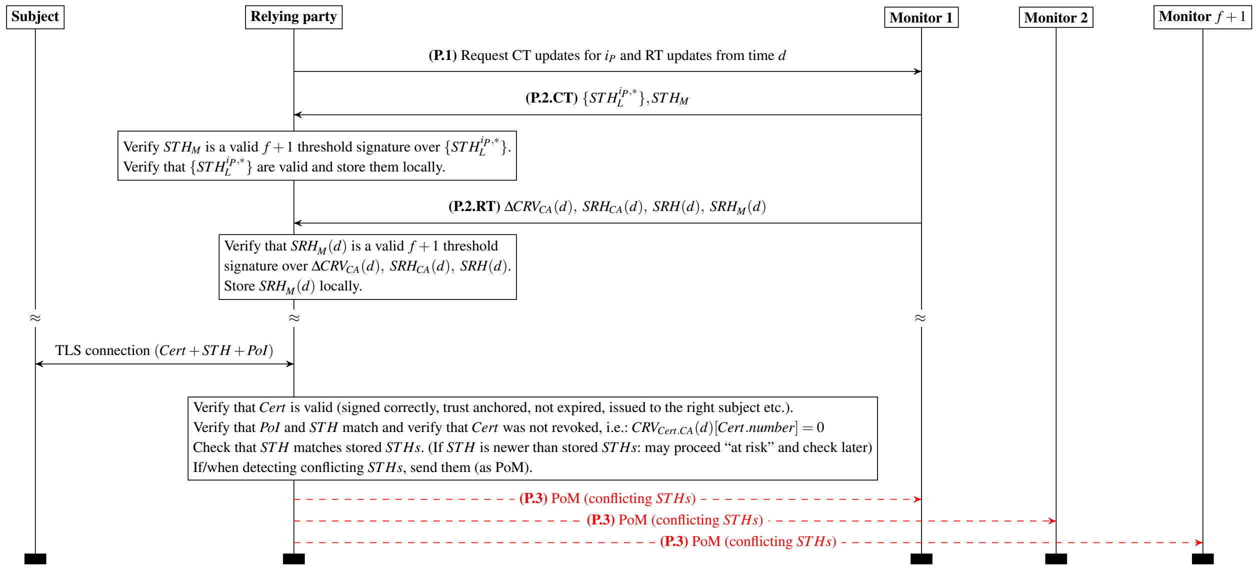

Subject
Relying party
Monitor 1
Monitor 2
Monitor $f+1$
(P.1) Request CT updates for i_P and RT updates from time d
(P.2.CT) $\{STH_L^{i_P,*}\}, STH_M$
Verify STH_M is a valid $f+1$ threshold signature over $\{STH_L^{i_P,*}\}$.
Verify that $\{STH_L^{i_P,*}\}$ are valid and store them locally.
(P.2.RT) $\Delta CRV_{CA}(d)$, $SRH_{CA}(d)$, $SRH(d)$, $SRH_M(d)$
Verify that $SRH_M(d)$ is a valid $f+1$ threshold
signature over $\Delta CRV_{CA}(d)$, $SRH_{CA}(d)$, $SRH(d)$.
Store $SRH_M(d)$ locally.
$\approx$
$\approx$
$\approx$
TLS connection $(Cert + STH + PoI)$
Verify that $Cert$ is valid (signed correctly, trust anchored, not expired, issued to the right subject etc.).
Verify that PoI and STH match and verify that $Cert$ was not revoked, i.e.: $CRV_{Cert.CA}(d)[Cert.number] = 0$
Check that STH matches stored $STHs$. (If STH is newer than stored $STHs$: may proceed "at risk" and check later)
If/when detecting conflicting $STHs$, send them (as PoM).
(P.3) PoM (conflicting $STHs$)
(P.3) PoM (conflicting $STHs$)
(P.3) PoM (conflicting $STHs$)

Figure 5: Periodic transparency update process between a relying party and a monitor as well as a TLS connection between a relying party and a subject). **(P.1)** The subject requests certificate transparency (CT) and revocation transparency (RT) updates from some monitor; in response, **(P.2.CT)** receives the newly issued $STHs$ and **(P.2.RT)** updated revocation status information. **(P.3)** If the relying party finds conflicting information (PoM), it submits it to $f + 1$ monitors.

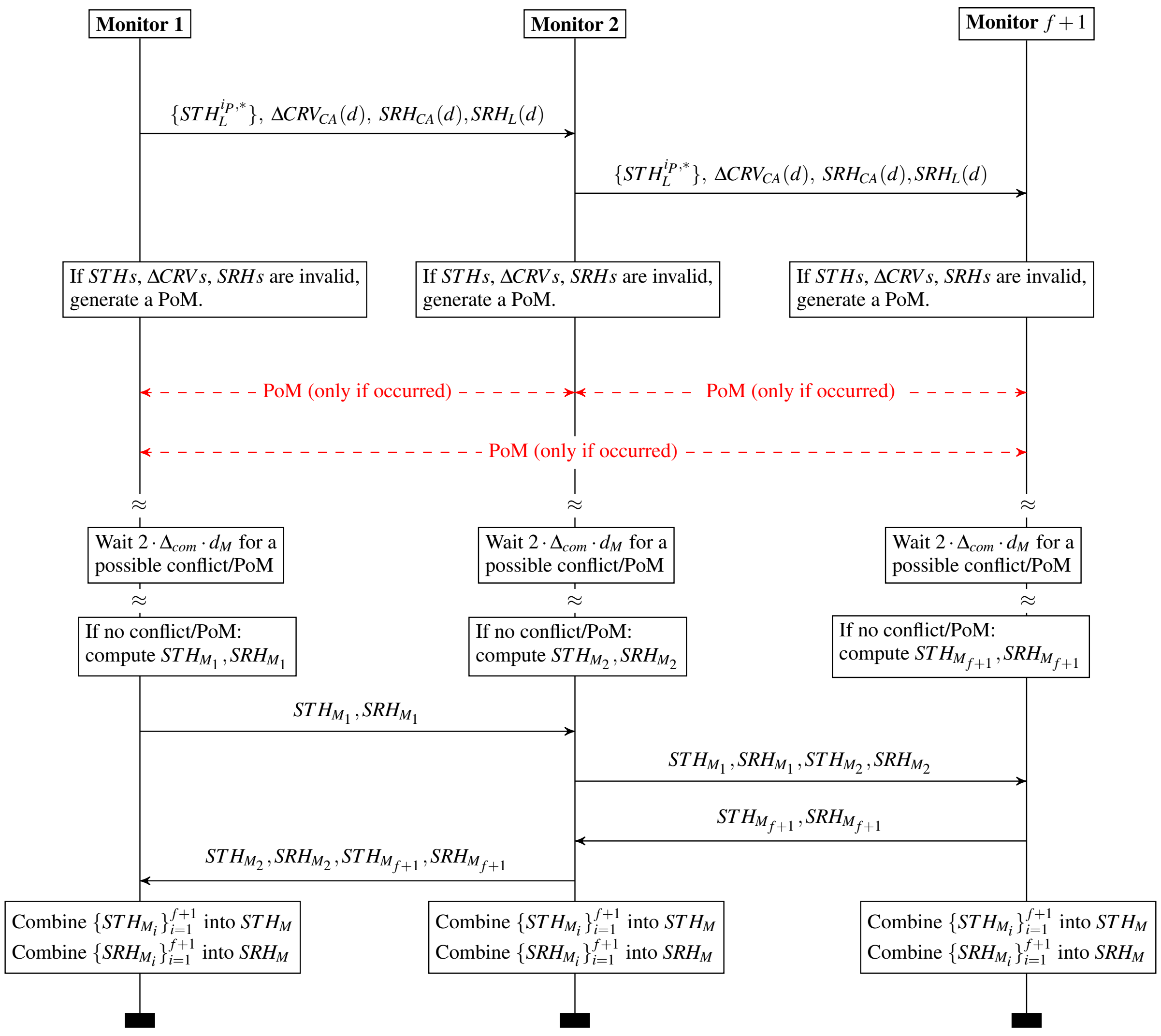

Monitor 1
Monitor 2
Monitor $f+1$
$\{STH_L^{i_P,*}\}$, $\Delta CRV_{CA}(d)$, $SRH_{CA}(d)$, $SRH_L(d)$
$\{STH_L^{i_P,*}\}$, $\Delta CRV_{CA}(d)$, $SRH_{CA}(d)$, $SRH_L(d)$
If $STHs$, $\Delta CRVs$, $SRHs$ are invalid, generate a PoM.
If $STHs$, $\Delta CRVs$, $SRHs$ are invalid, generate a PoM.
If $STHs$, $\Delta CRVs$, $SRHs$ are invalid, generate a PoM.
PoM (only if occurred)
PoM (only if occurred)
PoM (only if occurred)
Wait $2 \cdot \Delta_{com} \cdot d_M$ for a possible conflict/PoM
Wait $2 \cdot \Delta_{com} \cdot d_M$ for a possible conflict/PoM
Wait $2 \cdot \Delta_{com} \cdot d_M$ for a possible conflict/PoM
If no conflict/PoM: compute STH_{M_1}, SRH_{M_1}
If no conflict/PoM: compute STH_{M_2}, SRH_{M_2}
If no conflict/PoM: compute $STH_{M_{f+1}}, SRH_{M_{f+1}}$
STH_{M_1}, SRH_{M_1}
$STH_{M_1}, SRH_{M_1}, STH_{M_2}, SRH_{M_2}$
$STH_{M_{f+1}}, SRH_{M_{f+1}}$
$STH_{M_2}, SRH_{M_2}, STH_{M_{f+1}}, SRH_{M_{f+1}}$
Combine $\{STH_{M_i}\}_{i=1}^{f+1}$ into STH_M Combine $\{SRH_{M_i}\}_{i=1}^{f+1}$ into SRH_M
Combine $\{STH_{M_i}\}_{i=1}^{f+1}$ into STH_M Combine $\{SRH_{M_i}\}_{i=1}^{f+1}$ into SRH_M
Combine $\{STH_{M_i}\}_{i=1}^{f+1}$ into STH_M Combine $\{SRH_{M_i}\}_{i=1}^{f+1}$ into SRH_M

Figure 6: Gossip between monitors, where $f = 2$.

monitoring response by $t_0 + 2\Delta_{com}$, it detects that L is corrupt. Hence, assume that M receives this response by time $\tilde{t}_0 \leq t_0 + 2\Delta_{com}$. From the response, M learns t_I^L, the initialization time of L (on L's clock).

Beginning at $\tilde{t}_0$, the monitor M sends monitoring requests periodically, once per MMD, whenever its clock shows time $t_I^L + \Delta_{clk} + i \cdot MMD$ for any integer i, unless it earlier detected that L is corrupt. In particular, assuming that $t_1 \geq \tilde{t}_0$, then M sends a monitoring request when its clock shows time $\tilde{t}_1^M = t_I^L + \Delta_{clk} + i_1 \cdot MMD$, where $i_1 = \left\lceil \frac{t_1 - t_I^L}{MMD} \right\rceil$. By definition of i_1, we have: $i_1 + 1 < \frac{t_1 - t_I^L}{MMD} \leq i_1$, hence:

$$t_I^L + \Delta_{clk} + \frac{t_1 - t_I^L}{MMD} \cdot MMD \leq \tilde{t}_1^M \leq t_I^L + \Delta_{clk} + \left(\frac{t_1 - t_I^L}{MMD} + 1\right) \cdot MMD$$

Namely:

$$\Delta_{clk} + t_1 \leq \tilde{t}_1^M \leq \Delta_{clk} + t_1 + MMD$$

Since M's clock is within Δ_{clk} from real time, we have that M, unless it earlier detected that L is corrupt, sends a monitoring request at (real) time $\tilde{t}_1$ such that:

$$t_1 \leq \tilde{t}_1 \leq t_1 + 2 \cdot \Delta_{clk} + MMD$$

It remains to consider the special case where $t_1 > \tilde{t}_0$, i.e., t_1 occurs after M sent the first monitoring request to L, at t_0, but before M received the response, at $\tilde{t}_0$. Let $\tilde{t}_0^M$ denote the value of M's clock at $\tilde{t}_0$. Define again $i_1 = \left\lceil \frac{t_1 - t_I^L}{MMD} \right\rceil$ as before; then, M sends the next monitoring request when its clock shows $\tilde{t}_1^M = t_I^L + \Delta_{clk} + i_1 \cdot MMD$, just like before, *unless* $\tilde{t}_1^M < \tilde{t}_0^M$, i.e., M was 'supposed' to send a monitoring request exactly while it was waiting for the response to the first monitoring request it sent. In this case, M would *immediately* send a monitoring request, i.e., at $\tilde{t}_0 \leq t_1 + 2\Delta_{com}$. The claim follows since $2\Delta_{com} < MMD < 2 \cdot \Delta_{clk} + MMD$. □

Certificate Transparency (CT) was developed as a response to several failures of Certificate Authorities, which resulted in issuing of rogue certificates. The basic goal of CT is to detect rogue certificates, in order to deal with them, and possibly with the CA failure that allowed their issuing.

In the following Theorem, we consider the simpler case where both monitor and logger are benign, and prove that the (benign) monitor will detect issuing of certificates for a monitored domain, if logged by the (benign) logger. We state and prove the property for CTng, although it is really a special case of the theorems we prove later, allowing rogue loggers and monitors. We present this theorem both as a 'warm up exercise' and since the theorem and proof can be easily adapted for CT-v1 and CT-v2, with only minor changes; to our best knowledge, such analysis has not been presented.

Theorem 5.1. *Consider a run of CTng with a benign logger L and a benign monitor M. Assume that at time t_0, monitor M is asked to monitor certificates for domain D at logger L. Let c be a certificate for domain D logged by L at time t_1. Then, monitor M outputs certificate c at or before* $\max(t, t_0) + MMD + 2 \cdot (\Delta_{clk} + \Delta_{com})$.

Proof: From Lemma 1, M requests L to send certificates and STH at least once during $[t_1, t_1 + MMD + 2 \cdot \Delta_{clk}]$; let $t_2 \in [t_1, t_1 + MMD + 2 \cdot \Delta_{clk}]$ be the first request in this interval. Since L is benign and logged c at t_1, it would immediately respond to the request, and M would receive the response within $2 \cdot \Delta_{com}$. Hence, M receives c before $t_2 + 2\Delta_{com} < t_1 + MMD + 2 \cdot (\Delta_{clk} + \Delta_{com})$. $\square$

Theorem 5.1 focuses on the simple case, where the logger and monitor are benign. However, the vision of CT is more ambitious: *No Trusted Third Party (NTTP)*, i.e., ensure security without assuming that any specific entities are benign. Namely, not only we should allow for a possibly corrupt CA, we should allow for up to f loggers and monitors to be corrupt. We proceed to show that CTng handles such corruptions, first focusing on the certificate transparency property (and later extending to revocations, too). The proof is built gradually using few lemmas, hopefully making it easier to understand.

Simplification: For simplicity, let us assume, in the following proofs, that $MMD = MRD$. This avoids some annoying duplicity and few details.

Lemma 2. *Let M be a benign monitor that monitors logger L from time t_0. At any time $t_1 > t_0 + 4\Delta_{com}$, monitor M either has detected that L is faulty, or has a sent of valid STH, SRH_L and ΔCRV, whose timestamp t_1^L satisfies: $t_1^L \geq t_1 - MMD$ and $t_1^L \geq t_1 - MRD$.*

Proof: As shown in Lemma 1, from $t_0 + 2\Delta_{com}$, monitor M requests updates (of $STHs$, $SRHs$ and ΔCRV) from L whenever M's clock shows $t_I^L + i \cdot MMD + \Delta_{clk}$, for some $i > 0$. The last STH request before $t_1 - 2\Delta_{com}$ is sent when M's clock shows $t_I^L + i_1 \cdot MMD + \Delta_{clk}$, where $i_1 = \left\lfloor \frac{t_1 - t_I^L}{MMD} \right\rfloor$ (unless a fault was detected earlier). The response should arrive within $2 \cdot \Delta_{com}$, i.e., definitely before t_1, and contain valid STH, SRH_L and ΔCRV, whose timestamp is $t_I^L + i_1 \cdot MMD$; otherwise, a fault is detected. Hence, by t_1, either M detected L is faulty, or M has valid STH, SRH_L and ΔCRV whose timestamp t_1^L satisfies:

$$\begin{aligned} t_1^L &= t_I^L + i_1 \cdot MMD \\ &= t_I^L + \left\lfloor \frac{t_1 - t_I^L}{MMD} \right\rfloor \cdot MMD \\ &\geq t_1 - MMD \end{aligned}$$

$\square$

Lemma 3. *Let M be a benign monitor that receives at time t_1 an STH and/or a (SRH_L, ΔCRV) pair from logger L with timestamp t_1^L. Then by time $t_1 + 3 \cdot d_M \cdot \Delta_{com}$, all benign monitors will receive the corresponding STH_M and/or SRH_M, as well as ΔCRV, and/or a PoM against L.*

Proof: Benign monitors gossip $STHs$ and (SRH_L, ΔCRV) pairs from logger L as soon as they receive them, unless they have a PoM. Hence, by $d_M \cdot \Delta_{com}$, all benign monitors will receive the STH and/or (SRH_L, ΔCRV) pair. Upon the first time it receives a valid STH and/or SRH_L, a benign monitor waits for $2 \cdot d_M \Delta_{com}$, to see if a PoM (or a conflicting STH/ SRH_L) is received; if not, when the monitor gossips its share of the threshold signature, STH_{M_i} (or SRH_{M_i}). Within at most additional $d_M \cdot \Delta_{com}$, all benign monitors receive all these shares, and combine them into the complete threshold signatures, STH_M and/or SRH_M. $\square$

Lemma 4. *Let L be a logger that is monitored by $f+1$ benign monitors $\{M_1, \ldots, M_{f+1}\}$, from time t_0 (or earlier). At any time $t_2 > t_0 + 3 \cdot d_M \cdot \Delta_{com} + 4 \cdot \Delta_{com}$, every monitor M has either a PoM showing that L is faulty, or valid STH_M, STH, SRH_M, SRH_L and ΔCRV from L, whose timestamp t_2^L satisfies: $t_2^L \geq t_2 - MMD - 3 \cdot d_M \cdot \Delta_{com}$.*

Proof: Let $t_1 = t_2 - 3 \cdot d_M \cdot \Delta_{com} > t_0 + 4 \cdot \Delta_{com}$. From Lemma 2, at t_1, each monitor M_i either has detected that L is faulty, or has a valid STH and pair of SRH_L and ΔCRV, whose timestamp t_1^L satisfies: $t_1^L \geq t_1 - MMD$. If any of these $f+1$ monitors, say M_i, has such STH and/or SRH_L and ΔCRV at t_1, then from Lemma 3, by $t_1 + 3 \cdot d_M \cdot \Delta_{com}$, every benign monitor receives the corresponding STH_M and/or SRH_M, or a PoM against L.

It remains to consider the case that none of the $f+1$ monitors $\{M_1, \ldots, M_{f+1}\}$ has a valid STH, or pair of SRH_L and ΔCRV, at t_1; hence, all of them have detected that L is faulty. In this case, before t_1, each of them gossips a signed *accusation* message, stating that L is faulty. Before $t_1 + d_M \cdot \Delta_{com}$, all benign monitors receive these $f+1$ signed gossiped accusation, and therefore all benign monitors are aware that L is indeed faulty. As a result, each of them uses the threshold signature to sign a share of a PoM message against L, and they gossip these shares of the signed PoM. Before $t_1 + 2 \cdot d_M \cdot \Delta_{com}$, all benign monitors receive these shares and then use the threshold signature 'combine' function, to produce a PoM against L, signed by the threshold monitors' key. □

Theorem 5.2. *Consider logger L which is monitored by $f+1$ benign monitors from time t_0 (or earlier), including at least one monitor, say M, that monitors* certificates *issued by L (not just STHs). Let R be a benign relying party that receives, at time t_R, certificate c which has an STH signed by L with timestamp $\hat{t}_c^L > t_0 + 4\Delta_{com}$. Then (at least) one of the following holds:*

1. *The relying party detects, immediately at t_R, that c is invalid.*
2. *Monitor M outputs c or detects that L is rogue, before $\hat{t}_c^L + MMD$.*
3. *The relying party and all monitors have a PoM proving that L is misbehaving, before $t_R + 2MMD$.*

Proof: Since R is benign, it validates certificate c upon receiving it (at t_R). If c is found invalid, case 1 holds. Therefore, assume c is found valid, and let STH_c, PoI_c denote the (properly signed) STH and corresponding PoI, included in the CT-extension of c; in particular, since c is valid, it follows that R used the PoC to successfully confirm that c was included in the tree whose digest (hash) was signed in STH_c.

Relying party R also confirms that the timestamp t_c^L isn't too much into the future, namely, that $t_c^L \leq t_R + \Delta_{clk}$.

Let $t_2 = t_R + MMD + 3 \cdot d_M \cdot \Delta_{com}$. Before t_2, the relying party R makes the periodic request for transparency updates, i.e., $STHs$, to a monitor. The response should be either a PoM for L, or an STH_M for L, signed using the threshold signature key requiring $f+1$ monitors, whose timestamp t_R^L satisfies: $t_R^L \geq t_2 - MMD - 3 \cdot d_M \cdot \Delta_{com} = t_R$ (from Lemma 3 and Lemma 2).

However, it is possible that R attempts the update with a rogue monitor, who does not respond, in time, with either the PoM or the (valid) STH_M. In this case, after timeout $(2 \cdot \Delta_{com})$, R marks this monitor as corrupt and re-sends the request, this time to at least f (other) monitors. At least one monitor is benign, so a response (PoM or STH_M) is received before $t_2 + 4\Delta_{com}$.

From the response, R can confirm that STH_c has been previously shared with the monitors; if this is not the case, the pair of STH_c and the conflicting STH_M from the monitors, are a PoM against the logger. The relying party sends this PoM to at least $f+1$ monitors, so it must be received by at least one benign monitor (within additional Δ_{com}). This monitor gossips the PoM, so it reaches all monitors within an additional $d_M \cdot \Delta_{com}$, namely, before $t_2 + (5 + d_M) \cdot \Delta_{com} = t_R + MMD + 3 \cdot d_M \cdot \Delta_{com} + (5 + d_M) \cdot \Delta_{com} = t_R + MMD + (5 + 4d_M) \cdot \Delta_{com}$.

Using the MMD assumption $(5 + 4d_M) \cdot \Delta_{com} \leq MMD$, we have that all monitors, as well as R, has the PoM within $2MMD$ after R receives the rogue certificate. □

Theorem 5.2 focuses on transparency. We now present a similar theorem for revocation status transparency.

revocation transparency * *Proof:* Since R is benign, it validates certificate c upon receiving it (at t_R). If c is found invalid, case 1 holds. Therefore, assume c is found valid.

Let $t_2 = t_R + MRD + 3 \cdot d_M \cdot \Delta_{com}$. Before t_2, the relying party R makes the periodic request for revocation updates, i.e., $SRHs$ and $\Delta CRVs$, to a monitor. If the response is a PoM for L or an indication that L has stopped logging revocations for this CA, then case 4 holds. Hence, assume R receives an updated, valid SRH_M and ΔCRV for the CA (that issued c). If the entry for c in the resulting CRV indicates that it was revoked, then this must have been known to all benign monitors by t_R, i.e., case 3 holds. Otherwise, i.e., if the entry indicates that c was not revoked, then by $t_R - 2MRD$, there cannot have been any benign monitor which received a SRH_L and ΔCRV indicating that c is revoked, or there would have been a detected conflict (and PoM against L). □

6 Implementation and Evaluation

We next present the performance evaluation of our CTng prototype.

6.1 System Implementation and Experimental Setup

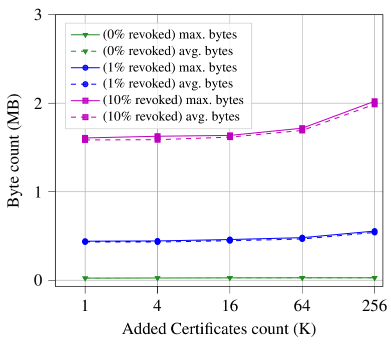
We implemented CTng in Go, building upon the existing Google implementation of CT[7]. We ran our experiments on Deterlab[8] with machines equipped with Dual Xenon processors, 2 GB or 4 GB of RAM, and network bandwidth of 100 Mbps. Connections were configured with a randomized communication latency of 40 ms (20 ms standard deviation). For all experiments, we used 4 loggers and varied the number of monitors from 4 to 32.

We relied on three settings. First, CTng without any revocation, referred to as **CTng (0% revoked)**. Second, CTng with revocation assuming an average rate of revocations (1% of certificates), referred to as **CTng (1% revoked)**. Third, CTng with revocation assuming a massive revocation event [34] (10% of certificates), referred to as **CTng (10% revoked)**.

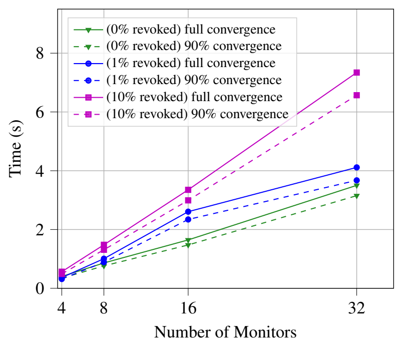
6.2 Experimental Results

First, we measured the impact of adding new certificates to logs. We started with a base tree of 1M certificates and progressively added a variable amount of new certificates, ranging from 1K up to 256K with 16 monitors. As illustrated in Figure 7a, we observed small, slowly-growing communication overhead. Without revocation, adding 256K certificates rather than 1K certificates results in insignificant additional overhead: $STHs$ are not affected by the number of certificates

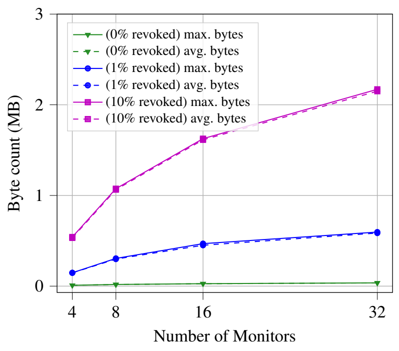
[7]https://github.com/google/certificate-transparency-go/
[8]https://www.deterlab.net



(a) Communication overhead for 16 monitors after adding variable amount of certificates to a base tree of 1M certificates.



(b) Convergence time based on different number of monitors after adding 16K certificates to a base tree of 1M certificates.



(c) Communication overhead based on different number of monitors after adding 16K certificates to a base tree of 1M certificates.

Figure 7: Communication overhead and convergence time for CTng considering different amount of monitors and certificates.

	Average (s)	**Minimum (s)**	**Maximum (s)**
CTng (0% revoked)	1.59	1.38	1.72
CTng (1% revoked)	2.15	1.87	2.33
CTng (10% revoked)	3.27	3.03	3.47

Table 3: Convergence time for 16 monitors after adding variable amount of certificates to a base tree of 1M certificates.

Gossiped data	**Average size**
STH	666 B
SRH_L	659 B
$SRH_L + \Delta CRV + \text{h}(CRV)$ (1% revoked cert.)	34 kB
$SRH_L + \Delta CRV + \text{h}(CRV)$ (10% revoked cert.)	128 kB

Table 4: Size of gossiped objects in CTng

Overhead	STH		$SRH_L + \Delta CRV + \text{h}(CRV)$	
	1 update per MMD	**5 updates per** MMD	**1% revoked certificates**	**10% revoked certificates**
Communication (1 day)	10 kB	50 kB	314 kB	618 kB
Storage (1 year)	3.65 MB	18.25 MB	2.79 MB (25.19 MB)	12.59 MB (25.19 MB)

Table 5: Communication and storage overhead for relying parties in CTng, for 200 CAs logging with 5 loggers each, and a total of 200M certificates. For revocation, we list compressed and (uncompressed) storage.

in the log. CTng eliminates proofs of consistency, used in CT-v1, which grow with the log.

When revocation are used, the ΔCRV and $SRHs$ do not introduce significant overhead. For example, revoking 10% of certificates after adding 1K certificates and after adding 256K certificates, incurs an overhead of ~ 1.61MB and ~ 2.03MB, respectively. This is mainly due to the efficiency of $\Delta CRVs$.

The time it takes for monitors to converge on the transparency updates is also small, as shown in Table 3. Specifically, it took 1.6 seconds on average when no revocations were performed, and 2.2 seconds and 3.3 seconds on average for 1% revocations and 10% revocations, respectively, for all added certificate sizes.

We also evaluated the impact of scaling the number of monitors from 4 up to 32. Figure 7b illustrates the average convergence time it takes for transparency updates to reach all monitors, and demonstrates a (relatively) steady linear increase of a reasonable average convergence time. Specifically, we observed that even for 32 monitors, the average amount of time it took to converge after 10% percent revocation, was ~ 7.35 seconds and only ~ 3.36 seconds after 1% revocation. Similarly, Figure 7c illustrates the communication overhead, and demonstrates an acceptable growth rate, peaking at ~ 0.6MB for 1% revocation and ~ 2.18MB for 10% revocation, both for 32 monitors.

Relying parties' communication and storage requirements. Perhaps most importantly, as we expect monitors to be well provisioned servers, CTng has a reasonable communication and storage overhead for relying parties (browsers) (Table 5).

Each relying party should periodically, typically daily, fetch updates ($STHs$, $SRHs$ and $\Delta CRVs$) from a monitor. For a realistic deployment of 200 CAs logging with 5 loggers each, and a total of 200 million certificates, the size of the $STHs$ update is 10kB assuming that each logger produces a single STH per MMD, or 50kB for 5 updates a day, scaling linearly. The storage overhead is 3.65 MB and 18.25 MB, respectively, for one year, which is the lifetime of most certificates [24]. The revocation transparency update is needed to verify that the certificate in question is non-revoked. The daily revocation update is 314 kB (1% revoked certificates) and 618 kB (10%, massive revocation event [34]) while the storage is 2.79 MB (1%) and 12.59 MB (10 %), when compressed, and 25.19 MB for both if not compressed. This revocation information would replace revocation data the browsers currently need to store.

Given the high efficiency of CTng, designers may opt for even shorter periods (MMD, MRD and/or periodic updates).

7 Conclusions

Security of systems is often based on *Trusted Third Parties (TTPs)*, with Certificate Authorities (CAs) being the classical

Entity	Number of files	Lines of Code (Go)
Gossiper	4	446
Monitor	18	1882
Certificate Authority	9	1034
Logger	3	452
Relying Part	3	242
Total	37	4056

Table 6: Components and Lines of Code in CTng.

example. However, a series of failures of CAs motivated the development of Certificate Transparency (CT) with the ambitious goal of a *No Trusted Third Party (NTTP)* design. The concept of NTTP is exciting and important - but non-trivial to achieve, esp. efficiently. Indeed, existing designs, implementation and even standards for CT, failed to achieve NTTP, and they ensure security only for trusted loggers - although, this was recognized only quite late [28], and have other drawbacks (Table 1).

We present CTng, an evolutionary update to CT, which implements the CT vision, and in particular, *ensures NTTP*. CTng is a practical design, which is backward-compatible with the current CT (and even pre-CT) PKI, reusing most of the CT (and X.509) concepts, architecture, protocols and even software. CTng also address an important privacy concern present in the previous CT designs.

CTng also includes a critical missing component from the current CT design, required to ensure NTTP for PKI schemes: *revocation status transparency*. Without it, a rogue CA can launch a simple yet effective attack which we call the *Zombie certificate attack*.

Acknowledgments

We thank Federico Cedolini, Abhinna Adhikari, and Eli Shattuck, who worked on the implementation of CTng, and DeterLab for the testbed resources. This work was partially supported by the Comcast Corporation. The opinions expressed are of the authors and not of their university or funding sources.

References

[1] David Basin, Cas Cremers, Tiffany Hyun-Jin Kim, Adrian Perrig, Ralf Sasse, and Pawel Szalachowski. ARPKI: Attack Resilient Public-Key Infrastructure. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 382–393. ACM, 2014.

[2] BLUE BOOK CCITT. Recommendations X. 509 and ISO 9594-8. *Information Processing Systems-OSI-The Directory Authentication Framework (Geneva: CCITT)*, 1988.

[3] Taejoong Chung, Jay Lok, Balakrishnan Chandrasekaran, David Choffnes, Dave Levin, Bruce M Maggs, Alan Mislove, John Rula, Nick Sullivan, and Christo Wilson. Is the web ready for ocsp must-staple? In *Proceedings of the Internet Measurement Conference 2018*, pages 105–118, 2018.

[4] Rasmus Dahlberg, Tobias Pulls, Tom Ritter, and Paul Syverson. Privacy-preserving & incrementally-deployable support for certificate transparency in tor. *Proceedings on Privacy Enhancing Technologies*, 2021(2):194–213, 2021.

[5] Peter Eckersley. Sovereign Key Cryptography for Internet Domains. https://git.eff.org/?p=sovereign-keys.git;a=blob;f=sovereign-key-design.txt;hb=HEAD, 2012.

[6] Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. Certificate transparency with privacy. *Proc. Priv. Enhancing Technol*, 2017(4), 2017.

[7] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. A Decentralized Public Key Infrastructure with Identity Retention. *IACR Cryptology ePrint Archive*, 2014:803, 2014.

[8] Mark Goodwin. Revoking intermediate certificates: Introducing OneCRL. Mozilla Security Blog, https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl March 2015.

[9] International Telecommunication Union. Recommendation ITU-T X.509, OSI – The Directory: Public-Key and Attribute Certificate Frameworks, 2016.

[10] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. Revisiting user privacy for certificate transparency. In *EuroS&P*, pages 432–447. IEEE, 2019.

[11] Tiffany Hyun-Jin Kim, Lin-Shung Huang, Adrian Perrig, Collin Jackson, and Virgil Gligor. Accountable Key Infrastructure (AKI): A Proposal for a Public-Key Validation Infrastructure. In *Proceedings of the 22nd international conference on World Wide Web*, pages 679–690. ACM, 2013.

[12] Murat Yasin Kubilay, Mehmet Sabir Kiraz, and Haci Ali Mantar. CertLedger: A new PKI model with Certificate Transparency based on blockchain. *arXiv preprint arXiv:1806.03914*, 2018.

[13] Adam Langley. Revocation checking and Chrome's CRL.

http://www.imperialviolet.org/2012/02/05/crlsets.html, Feb 2012.

[14] Adam Langley. Certificate transparency: Comparison with other technologies. In http://http://www.certificate-transparency.org/comparison, July 2014.

[15] James Larisch, David R. Choffnes, Dave Levin, Bruce M. Maggs, Alan Mislove, and Christo Wilson. Crlite: A scalable system for pushing all tls revocations to all browsers. In *IEEE Symposium on Security and Privacy*, pages 539–556. IEEE Computer Society, 2017.

[16] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962 (Experimental), June 2013.

[17] Ben Laurie. Certificate transparency. *Communications of the ACM*, 57(10):40–46, 2014.

[18] Ben Laurie and Emilia Kasper. Revocation Transparency. *Google Research, September*, 2012.

[19] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In Rainer Böhme and Tatsuaki Okamoto, editors, *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, volume 8975 of *Lecture Notes in Computer Science*, pages 168–186. Springer, 2015.

[20] Stephanos Matsumoto and Raphael M Reischuk. IKP: Turning a PKI Around with Decentralized Automated Incentives. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 410–426. IEEE, 2017.

[21] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. CONIKS: Bringing Key Transparency to End Users. In *USENIX Security Symposium*, pages 383–398, 2015.

[22] Ralph C Merkle. A Digital Signature Based on a Conventional Encryption Function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[23] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. X.509 internet public-key infrastructure — online certificate status protocol (OCSP). Internet RFC 2560, June 1999.

[24] Patrick Nohe. Maximum ssl/tls certificate validity is now one year. GlobalSign Blog, June 2020.

[25] Mark Dermot Ryan. Enhanced certificate transparency and end-to-end encrypted mail. In *NDSS*, 2014.

[26] Victor Shoup. Practical Threshold Signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.

[27] Trevor Smith, Luke Dickenson, and Kent E. Seamons. Let's revoke: Scalable global certificate revocation. In *NDSS*. The Internet Society, 2020.

[28] R Stradling, B Laurie, A Langley, E Kasper, and E Messeri. Certificate transparency version 2.0 draft-ietf-trans-rfc6962-bis-35. 2021.

[29] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, and Bryan Ford. Certificate Cothority: Towards Trustworthy Collective CAs. *Hot Topics in Privacy Enhancing Technologies (HotPETs)*, 7, 2015.

[30] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping Authorities "Honest or Bust" with Decentralized Witness Cosigning. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 526–545. Ieee, 2016.

[31] Pawel Szalachowski, Stephanos Matsumoto, and Adrian Perrig. PoliCert: Secure and Flexible TLS Certificate Management. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 406–417. ACM, 2014.

[32] Alin Tomescu and Srinivas Devadas. Catena: Efficient Non-equivocation via Bitcoin. In *2017 38th IEEE Symposium on Security and Privacy (SP)*, pages 393–409. IEEE, 2017.

[33] Jiangshan Yu, Vincent Cheval, and Mark Ryan. DTKI: A New Formalized PKI with Verifiable Trusted Parties. *The Computer Journal*, 59(11):1695–1713, 2016.

[34] Liang Zhang, David Choffnes, Dave Levin, Tudor Dumitraş, Alan Mislove, Aaron Schulman, and Christo Wilson. Analysis of ssl certificate reissues and revocations in the wake of heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, 2014.