Privacy-Preserving & Incrementally-Deployable Support for Certificate Transparency in Tor

Abstract: The security of the web improved greatly throughout the last couple of years. HTTPS is the new default, with web browsers moving from positive to negative security indicators that instead warn the user if a connection to a website is unencrypted. The ecosystem around Certificate Authorities (CAs) also improved: the advent of mandatory Certificate Transparency (CT) logging—as enforced by commonly used web browsers—make the long-known problem of weakest-link security in CAs more sustainable. However, Tor and the Firefox-based Tor Browser (TB) lacks support for CT.

In this paper, we present privacy-preserving and incrementally-deployable designs that add support for CT in Tor. Our designs go beyond the support found in web browsers like Chrome and Safari: we aim to detect and disclose cryptographic evidence of CA and/or CT log misbehavior to public scrutiny when done to man-in-the-middle or impersonate HTTPS connections that originated from TB. As a starting point our base design targets CA misbehavior by sharing observed certificates in TB with the wider CT landscape. Associated privacy leaks, such as browsing behaviors, are minimized using distributed reporting and caching of encountered certificates at relays in the Tor network.

We also present two possible extensions to our base design that detects CT log misbehavior. One extension makes a slight change to the API and operation of CT logs, whereas the other adds more complexity to Tor but in turn completely removes all trust placed in CT log operators. Exposing misbehaving CT logs is important for the web in general and not just Tor, since CT as deployed today lacks the strong mechanisms necessary to reduce trust in log operators. We turn Tor into a system for maintaining a probabilistically-verified view of the CT-landscape available from Tor's consensus.

Keywords: Certificate Transparency, Tor

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

1 Introduction

Metrics reported by Google and Mozilla reveal that encryption on the web skyrocketed the past couple of years: at least 85% of all web pages load using Transport Layer Security (TLS) as part of HTTPS [41, 43]. An HTTPS connection is initiated by a TLS handshake where the client web browser requires that the web server presents a valid certificate to authenticate the identity of the server (e.g., to make sure that the client who wants to visit mozilla.org really is connecting to Mozilla, and not, say, Google). A certificate is considered valid if it is digitally signed by a Certificate Authority (CA) that the browser trusts. Each CA is trusted to certify (sign) that specific cryptographic key-material as part of a certificate belongs to a particular domain name.

The CA trust model suffers from *weakest-link* security: web browsers trust hundreds of CAs, and it is enough to compromise a single CA to get a certificate mis-issued in the name of a target domain [7, 15]. Motivated in part by prominent CA compromises—such as the issuance of a fraudulent certificate for *.google.com and *.torproject.org by DigiNotar in 2011 [36]—several major browser vendors have mandated that certificates issued by CAs must be included into trusted Certificate Transparency (CT) logs for browsers to trust them [24, 26, 27]. The idea behind CT is that, by making all issued certificates transparent, mis-issued certificates can be detected *after* issuance and appropriate actions taken to keep the wider web safe (e.g., revocation of certificates, suspected compromises investigated, or trust in misbehaving CAs removed from browsers). Browsers that wish to benefit from CT augment the TLS handshake to also require cryptographic proof from the server that the presented certificate will be included into CT logs trusted by the browser. Notable browsers with mandatory CT support are Google Chrome and Apple's Safari [34, 37]. Unfortunately, Mozilla's Firefox lacks support but has a partial implementation [38].

Beyond encryption on the web, anonymous access to the web has also matured. In particular, Tor with its Tor Browser (TB) today has millions of daily users [12, 30], and effort is ongoing into maturing the technology for wider use [42]. The basis of TB is Firefox, where TB then enables anyone to browse the web anonymously by relaying traffic between the user's browser and the web server in question through the Tor network, consisting of thousands of voluntary-run relays across the globe [48].

Just like attackers may be interested in breaking HTTPS, attackers may also be interested in breaking the anonymity provided by Tor to deanonymize users. One common deanonymization technique, known to be used in practice is to compromise TB instead of directly circumventing the anonymity provided by Tor [8, 39, 40, 44]. Modern web browsers like TB (i.e., Firefox) are one of the most complex types of software in wide use today. This complexity and wide use leads to security vulnerabilities and incentives for exploitation. For example, the exploit acquisition platform Zerodium offers up to $100,000 for a zero-day exploit against Firefox that leads to remote code execution and local privilege escalation [45] (i.e., full control of the browser).

An attacker that wishes to use such an exploit to compromise and then ultimately deanonymize a TB user has to deliver the exploit to TB. Since the web is mostly encrypted today, this primarily has to happen over an HTTPS connection, where the attacker controls the content returned by the web server. While there are a number of possible ways for an attacker to accomplish this (e.g., by compromising a web server that the target TB user connects to), one option is to *impersonate* a web server by acquiring a fraudulent certificate. Due to the Tor network being run by volunteers, getting into a position for performing such an attack is relatively straightforward (the attacker can volunteer to run malicious exit relays [58]). The same is true for an attacker that wishes to *man-in-the-middle* connections made by TB users. In such a case, a TB exploit may not even be needed to deanonymize the user; e.g., if the user logs into an account at the service associated with its identity and that the attacker can now observe.

1.1 Introducing CTor

In this paper, we propose an incrementally deployable and privacy-preserving design which we refer to as CTor. CTor brings CT to Tor by modifying how both TB and relays in the Tor network work. The goal is to make impersonation and man-in-the-middle attacks on HTTPS connections *detectable after the fact* when they are carried out by a powerful attacker that:

1. can acquire a forged certificate from a trusted CA,
2. with the necessary forged cryptographic proofs from CT logs so that TB accepts the certificate as valid (with no intent of making the forged certificate publicly available in a CT log), and
3. has the capability to fully gain control of TB shortly after the establishment of the HTTPS connection.

Our base CTor design uses the CT log ecosystem against the attacker to ensure a non-zero (tweakable) probability of public disclosure of the forged certificate *per use* against a TB user. This is done by probabilistically adding the certificate to *other CT logs* than those directly inferable as under attacker control (due to the accompanying proofs). TB uses modified relays to cache and interact with CT logs for the sake of privacy, ensuring that CT log operators (e.g., Google and Cloudflare) do not get a real-time feed of all browsing activities from TB (in addition to what their privileged positions on the Internet already provides them [16, 19]).

In addition to the base CTor design we present two extensions. Both extensions have in common that their goal is to make transparent the forged cryptographic proofs from attacker-controlled CT logs used to get TB to accept malicious HTTPS connections. Such a proof would likely immediately disqualify each offending CT log and thus come at great cost to the attacker in terms of lost capability. Even a small probability of detection may entail unacceptable risk at such a high cost. Note that we have already observed several instances of CT logs that, intentionally or not, misbehaved [31, 52, 53]. Most recently, a compromised CT log signing key was reported for the first time [50].

The key change in the first extension is that relays now challenge CT logs to cryptographically prove the correctness of their prior proofs presented to TB instead of simply sharing the certificates (the subject of the proofs) with other CT logs. In case an offending CT log fails to provide such a proof in a timely manner, the querying relay ensures that the proof arrives at a trusted auditor. While this adds both more complexity and operational challenges to Tor (mainly auditors and storage at relays), it removes all trust in the CT ecosystem. Another benefit is that, as a consequence of CTor's design, Tor would provide a probabilistically-verified view of the CT log ecosystem from Tor's consensus.

The second extension involves only a minor tweak to the base CTor design. Unfortunately, this tweak is primarily not to Tor but to the API of CT logs. While similar changes have been discussed earlier in the context of gossiping in CT [14], it is still a significant change. The small tweak is that CT logs provide an API for receiving cryptographic proofs made by other CT logs wrt. certificate inclusion. This would then mean that relays would add to other CT logs proofs observed by TB clients in addition to the certificate. This would also significantly reduce the trust assumptions the wider web has to place on CT log operators, from close to full trust as-is today to only trusting that some operators are honest.

1.2 Contribution and Structure

Section 2 provides necessary background on CT and Tor. Our threat model is described in detail in Section 3, taking into account the threat models of Tor and CT.

Section 4 presents the base design of CTor that enables *TB to probabilistically provide evidence to CT logs that it has been presented with a fraudulent certificate while preserving privacy*. This greatly impacts risk-averse attackers, since part of its fraudulent behavior has been made transparent through the CT ecosystem: the issuing CA may get revoked from the trust store of browsers, the domain name in question (as part of the certificate) is made public, and awareness of the event may draw unwanted attention. Our security analysis, in Section 5, shows that one of the best bets for an attacker would be to attack the entire Tor network or all CT logs: an act that would also likely draw unwanted attention.

Two extensions to the base CTor design are presented in Sections 6 and 7. Both extensions aim to make transparent the accompanying fraudulent proofs issued by CT logs to convince TB to accept a fraudulent certificate. The deployment of either extension *would greatly contribute to the open question of how to reduce trust in CT log operators* currently caused by the lack of an appropriate gossiping mechanism and privacy issues while interacting with CT logs [14, 17, 33]. In particular, the auditor-based extension (Section 6) would result in a *probabilistically-verified view of the entire CT log ecosystem* available from Tor's consensus. This view could be used as the basis for trust by other browsers, *greatly improving the security posture of the entire web.*

By splitting our designs into a base design and two possible extensions we argue that CTor is *incrementally deployable*: start with completing Firefox support for CT, add the base CTor design, then pick one of the two extensions based on how CT and Tor evolves.

Section 8 estimates performance aspects of our designs, showing that the estimated (based on Mani et al.'s measurements [30]) circuit-, bandwidth- and memory-*overheads are modest even without caching.* Privacy aspects of our design choices are covered in Section 9, with a focus on the essential role of the distributed nature of the Tor network in both preserving privacy of Tor users as well as the overall security of our proposed designs. In gist, *a similar approach would be privacy-invasive without Tor*, e.g., if adopted by Google Chrome. Section 10 presents related work and Section 11 conclusions.

2 Background

2.1 Certificate Transparency

The idea to transparently log TLS certificates emerged at Google in response to a lack of proposals that could be deployed without drastic ecosystem changes and/or significant downsides [24]. In reality, CT is about logging certificate *chains*: a domain owner's certificate is signed by an intermediate CA, whose certificate is in turned signed by a root CA that acts as a trust anchor [15]. The resulting certificate chain is composed of the domain owner's leaf certificate, the intermediate CA certificate, and the root CA certificate. Trust anchors are shipped in software, such as web browsers and operating systems.

2.1.1 Cryptographic Foundation

The operator of a CT log maintains a tamper-evident append-only Merkle tree [26, 27]. At any time, a Signed Tree Head (STH) can be produced which fixes the log's structure and content. Important attributes of an STH include the tree head (a cryptographic hash), the tree size (a number of entries), and the current time. Given two tree sizes, a log can produce a *consistency proof* that proves the newer tree head entails everything that the older tree head does. As such, anyone can verify that the log is append-only without downloading all entries and recomputing the tree head. Membership of an entry can also be proven by producing an *inclusion proof* for an STH. These proof techniques are formally verified [13].

Upon a valid request, a log must add an entry and produce a new STH that covers it within a time known as the Maximum Merge Delay (MMD), e.g., 24 hours. This policy aspect can be verified because in response, a Signed Certificate Timestamp (SCT) is returned. An SCT is a signed promise that an entry will appear in the log within an MMD. A log that violates its MMD is said to perform an *omission attack*. It can be detected by challenging the log to prove inclusion. A log that forks, presenting one append-only version to some entities and another to others, is said to perform a *split-view attack*. Split-views can be detected by STH gossip [6, 10, 33, 55].

2.1.2 Standardization

The standardized CT protocol defines public HTTP(S) endpoints that allow anyone to check the log's accepted trust anchors and added certificates, as well as obtaining the most recent STH and fetching proofs [26, 27]. For example, the `add-chain` endpoint returns an SCT if the added certificate chain ends in a trust anchor returned by the `get-roots` endpoint. We use the `add-chain` endpoint in Section 4, and the `get-proof-by-hash` as well as the `get-sth` endpoints in Section 6. The `get-proof-by-hash` endpoint takes as input a leaf certificate hash and an STH's tree size, which indicates the tree head that the log should base its inclusion proof on. The proof is valid if it can be used in combination with the certificate to reconstruct the tree head of some STH.

2.1.3 Verification

The CT landscape provides a limited value unless it is verified that the logs play by the rules. The rules are largely influenced by major browser vendors that define CT policies. For example, how is CT enforced by their user agents, and what is required to become a recognized CT log in terms of uptime requirements and accepted trusted anchors. Google Chrome and Apple's Safari require that a certificate chain must be accompanied by two independent SCTs [34, 37]. Independence refers to the associated log operators. While there are several ways that a log can misbehave with regards to these policy aspects, the most fundamental forms of cheating are omission and split-view attacks: it would undermine detection of mis-issued TLS certificates. A party that follows-up on inclusion and consistency proofs is said to *audit* the logs. Wide-spread client-side auditing is a premise for CT logs to be untrusted, but none of the web browsers that enforce CT engage in such activities yet. For example, it is difficult due privacy concerns [17]. CT also assumes that domain owners *monitor* the logs for mis-issuance by inspecting the added certificates [9, 28].

2.2 Tor

Most of the activity of Tor's millions of daily users starts with TB and connects to some ordinary website via a circuit comprised of three randomly-selected Tor relays. In this way no identifying information from Internet protocols (such as IP address) are automatically provided to the destination, and no single entity can observe both the source and destination of a connection. TB is also configured and performs some filtering to resist browser fingerprinting, and first party isolation to resist sharing state or linking of identifiers across origins. More generally it avoids storing identifying configuration and behavioral information to disk.

Tor relays in a circuit are selected at random, but not uniformly. A typical circuit is comprised of a *guard*, a *middle*, and an *exit*. A guard is selected by a client and used for several months as the entrance to all Tor circuits. If the guard is not controlled by an adversary, that adversary will not find itself selected to be on a Tor circuit adjacent to (thus identifying) the client. And because some relay operators do not wish to act as the apparent Internet source for connections to arbitrary websites, relay operators can configure the ports (if any) on which they will permit connections besides to other Tor relays. Finally, to facilitate load balancing, relays are assigned a weight based on their apparent capacity to carry traffic. In keeping with avoiding storing of linkable state, even circuits that share an origin will only permit new connections over that circuit for ten minutes. After that, if all connections are closed, all state associated with the circuit is cleared. Similarly, at each relay state associated with a given circuit is cleared shortly after a circuit is closed.

Tor clients use this information when choosing relays with which to build a circuit. They receive the information via an hourly updated *consensus*. The consensus assigns weight as well as flags such as `guard` or `exit` along with auxiliary flags such as `stable`, which, e.g., is necessary to obtain the `guard` flag since guards must have good availability. Self-reported information by relays in their *extra-info document*, such as statistics on their read and written bytes, are also part of the consensus and uploaded to *directory authorities*. Directory authorities determine the consensus by voting on various components making up the shared view of the state of the Tor network. Making sure that all clients have a consistent view of the network prevents epistemic attacks wherein clients can be separated based on the routes that are consistent with their understanding [11]. This is only a very rough sketch of Tor's design and operation. More details can be found by following links at Tor's documentation site [57].

Tor does not aim to prevent end-to-end correlation attacks. An adversary controlling the guard and exit, or controlling the destination and observing the client ISP, etc. is assumed able to confirm who is connected to whom on that particular circuit. The Tor threat model assumes an adversary able to control and/or observe a small to moderate fraction of Tor relays measured by both number of relays and by consensus weight, assumes a large number of Tor clients able to, for example, flood individual relays to detect traffic signatures of honest traffic on a given circuit [18]. Also, the adversary can knock any small number of relays offline via either attacks from clients or direct Internet DDoS.

3 Threat Model

We consider an attacker that controls a CA, enough CT logs to pass TB's SCT-centric CT policy, some Tor clients, and a fraction of Tor relays. For example, it is possible to issues certificates and SCTs, dishonor promises of public logging, present split-views at will, intercept and delay traffic from controlled exit relays as well as CT logs, and be partially present in the network. This includes a weaker attacker that does not *control* CAs and CT logs, but, who *gained access* to the relevant signing keys [23, 31]. A modest fraction of CTor entities can be subject to DoS, but not everyone at once and all the time. In other words, we consider the threat model of Tor and TB as a starting point [12, 35]. Any attacker that can reliably disrupt CT and/or Tor to such a large extent is therefore not within our threat model.

Given that we are in the business of enforcing CT, the attacker needs to hide mis-issued TLS certificates and SCTs from entities that audit the CT landscape. As described in Section 2.1, this can either be achieved by omission or split-view attacks. Our intended attacker is clearly powerful and may successfully issue a certificate chain and associated SCTs without detection some of the time, but, a CA caught in mis-issuance or a CT log that violated an MMD promise will no longer be regarded as trusted. Therefore, we assume a *risk-averse* attacker that above a relatively low probability of detection would be deterred from engaging in such activities.

We want to minimize the existence of undetectable man-in-the-middle attacks against TB. The goal of *detection* is inherited from CT's threat model, which aims to remedy certificate mis-issuance *after the fact*; not prevent it [24]. We focus on HTTPS traffic that TB generated because Tor is used to browse the normal (encrypted) web [30]. Note that it is generally *difficult* to target a specific TB user due to Tor's anonymity, circuit isolation, and use of HTTPS Everywhere. However, targeting some or all users that visit a website is an eminent threat: the attacker could intercept traffic from an exit relay or upstream of the website. Once network traffic is intercepted (with associated forged certificate and SCTs), it is possible for the attacker to attempt to to serve an exploit with the goal of *user deanomymization*. Without the ability to intercept traffic (i.e., forge certificates and SCTs without detection), user deanomymization becomes harder.

We identify additional threats that follow from our threat model and design in the security analysis. Namely, attack vectors related to timing (Section 5.2.1) as well as Tor relay tagging and flooding (Section 6.2).

4 Design

Unfortunately, this design is not as straightforward as completing Mozilla's partial implementation in Firefox, enabling it in TB and having TB directly query CT logs. For one, real-time interaction with CT logs by TB with little to no caching leads to CT log operators (e.g., Google and Cloudflare that are major operators) getting an unacceptably centralized degree of insight into TB usage across the entire network. Adding caching or delays to TB is non-trivial due to TB's security and privacy design goals (see Section 2). In particular, the *state separation*, *cross-origin identifier unlinkability* and *long-term unlinkability* goals rule out a large number of designs in which TB caches or delays interacting with CT logs on its own. Instead, we use relays in the Tor network to cache and interact with CT logs. We refer to these relays as Certificate Transparency Relays (CTRs). Further, we refer to the certificate chains and associated SCTs—regardless of how they are delivered to TB (e.g., included in the certificate, over OCSP, or stapled)—as SCT Feedback Objects, or SFOs for short [33].

First we introduce a base design that is extended later on. Our aim is to hold CAs accountable in a setting where some, but not all, CT logs are dishonest. In other words, the starting-point is to catch misbehaving CAs by adding resilience against misbehaving CT logs. The basic idea is shown in Figure 1 as three distinct phases: a *submission phase* where SFOs are presented to TB and submitted probabilistically to CTRs, a *storage phase* where the submitted SFOs are stored for some time to obfuscate real-time browsing patterns, and an *auditing phase* where the stored SFOs are audited further by adding the underlying certificate chains to independent CT logs. The exact behavior of these phases are governed by parameters set in the Tor consensus.

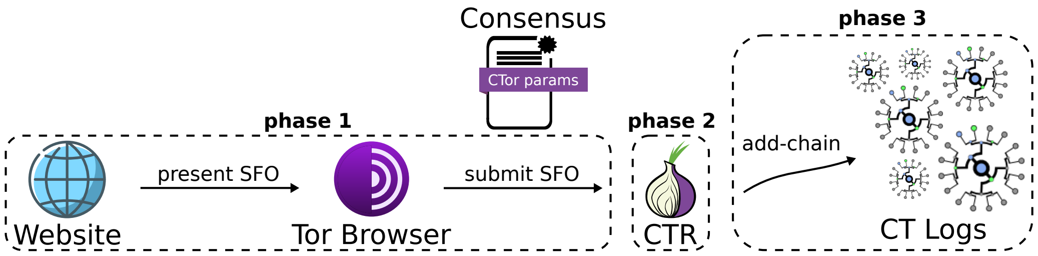

Fig. 1. The base design of CTor divided into three phases. A website presents an SFO to TB, and TB in turn with some probability submits the SFO to a random CTR (phase 1). A CTR stores the SFO a period of time (phase 2). The CTR adds the certificate in the SFO to a random independent CT log (phase 3).

4.1 Tor Consensus

4.1.1 CTR Flag

The existing `known-flags` item determines the different flags that the consensus might contain. We add another flag named `CTR`, which indicates that a Tor relay should support CT-auditing as described later. For now, assume that a relay qualifies as a CTR if it is flagged as `stable` and not `exit`, to spare the relatively sparse exit bandwidth and only use relays that can be expected to stay online. Section 9 discusses trade-offs in the assignment of the `CTR` flag.

4.1.2 Recognized CT Log

CTRs interact with logs in the CT landscape that Tor's consensus recognizes, which means that a majority of directory authorities voted for inclusion by proposing a `ct-log-info` entry containing a log ID, a public key, and a base URL [26, 27]. While it is a natural starting point to recognize the same logs as TB, such overlap is not required because the base design only adds *certificate chains* in other independent logs that accept them. This simplifies the addition of community-driven logs that, for example, could serve the primary purpose of facilitating auditing and increasing diversity to distribute mis-issued certificates widely. The premise of trust anchor policies that cover the commonly accepted root certificates is aligned with standardized suggestions [26, 27] as well as Apple's requirements [46]. We vet this assumption further based on related work in Appendix A [22].

4.1.3 Other Parameters

Directory authorities influence the way in which TB and CTRs behave by voting on other necessary parameters. Below, the value of an item is computed as the median of all votes.

ct-submit-pr: A floating-point in $[0, 1]$ that determines Tor Browser's submission probability. For example, 0 disables submissions while 0.10 means that every 10^{th} SFO is sent to a random CTR on average.

ct-sfo-max-bytes: A natural number that determines how many wire-bytes a normal SFO should not exceed. As outlined in Section 4.2, excessively large SFOs are subject to stricter verification criteria.

ct-log-timeout: A natural number that determines how long a CTR waits before concluding that a CT log is unresponsive, e.g., 10 seconds. As outlined in Section 4.4, timeouts trigger implicit resubmissions.

ct-delay-dist: A distribution that determines how long a CTR should wait at minimum before auditing a submitted SFO. As outlined in Section 4.3, random noise is added, e.g., on the order of minutes to an hour.

ct-backoff-dist: A distribution that determines how long a CTR should wait between two auditing instances, e.g., a few minutes on average. As outlined in Section 4.4, CTRs audit pending SFOs in batches at random time intervals to spread out log overhead.

Besides improving load-balancing and performance, sampling the parameter `ct-backoff-dist` helps protect privacy by effectively creating a timed mix [51], though one with a randomized firing interval. The randomness in whether a client submits an SFO at all combined with the timed mixing potentially complicates leaking client behavior to a CT log. But we cannot reliably assume that other honest clients submit to the CTR between mix firings (submissions to CT logs). Adding a per-SFO value sampled from `ct-delay-dist` effectively adds stop-and-go mixing [21] to the privacy protection, but where there is only one mix (CTR) between sender (client) and receiver (CT log). So there is no point in a client-specified interval-start-time such that the mix drops messages arriving before then, and there is no additional risk in having the interval end time set by the mix rather than the sender. This means both that some SFOs a client sends to a CTR at roughly the same time might be in different batches of SFOs sent to a CT log and that SFOs submitted to that CTR by other honest clients are more likely to be mixed with these.

4.2 Phase 1—Tor Browser

The first phase covers TB and how an SFO is validated in the TLS handshake. We rely on TB's trust store to check certificate chains, an assumed SCT-centric CT policy similar to Chrome [34] and Safari [37] to avoid breakage, and `ct-max-sfo-bytes` as well as `ct-submit-pr` to determine whether and how there should be any follow-up auditing that goes beyond SCT signature verification. Given an incoming SFO s:

1. Raise a certificate error and stop if the certificate chain of s is not rooted in TB's trust store.
2. Raise a certificate transparency error and stop if the SCTs of s fail TB's SCT-centric CT policy.
3. Accept s and conduct the remaining steps in the background if $\mathsf{len}(s) \leq$ `ct-max-sfo-bytes`, otherwise complete all steps and then accept s as valid.
4. Flip a biased coin based on `ct-submit-pr` and stop if the outcome indicates no further auditing.
5. Submit s to a sampled CTR's SFO-endpoint on a pre-built CT-circuit. The circuit used for submission is closed immediately after use without waiting for any acknowledgment.

In other words, an SFO is accepted as valid if the end-entity certificate is rooted in a trust anchor *and* it has enough SCTs as dictated by an SCT-centric CT policy. The decision and possible submission of an SFO to a random CTR should never block the TLS handshake under normal circumstances, whereas *excessively large* SFOs do due to high security risks. See Section 5.2.1, which also motivates why it is paramount that submission circuits are pre-built and closed as soon as possible.

4.3 Phase 2—Storage

We suggest that CTRs accept SFO submissions on an HTTP endpoint.[1] For example, Nordberg *et al.* defined an SCT feedback interface that can be reused if an array-length of one is enforced by the CTR [33]. With regards to some CT circuit, process an incoming SFO s as follows:

1. Close the current circuit to enforce one-time usage.
2. Stop if no CT log in the Tor consensus accepts the trust anchor of the underlying certificate chain in s.
3. Stop if s is cached (see Section 4.4) or already pending to be audited.
4. Sample an independent CT log l that issued no SCT in s. If there are no independent CT logs listed in the Tor consensus, sample a dependent CT log instead.
5. Compute an `audit_after` timestamp $\mathtt{t} \leftarrow \mathsf{now}() + \mathsf{random}(\texttt{ct-delay-dist})$. The former returns the current time and the latter a random delay.
6. Add (l, t, s) to a buffer of pending SFOs.

An SFO that (i) cannot be audited with regards to a CT log that the Tor consensus recognizes, (ii) is already audited as indicated by a *cache*, or (iii) is pending to be audited in a *buffer* of pending SFOs, is discarded. In contrast, a new SFO is stored in the CTR's buffer alongside an `audit_after` timestamp and a sampled CT log. The `audit_after` timestamp specifies the earliest point in time that an SFO will be audited in phase 3, which adds random noise that obfuscate real-time browsing patterns in the Tor network. Auditing is also fixed at this stage with regards to some CT log. If memory becomes a scarce resource, delete SFOs at random [33].

4.4 Phase 3—Auditing

Each CTR maintains a single shared circuit that is used to interact with all CT logs that have `ct-log-info` items. For *each* such CT log l, the CTR runs the following steps indefinitely:

1. Sample a delay $d \leftarrow \mathsf{random}(\texttt{ct-backoff-dist})$ and wait until d time units elapsed.
2. For each pending buffer entry (l', s, t), where $l' = l$ and $t <= \mathsf{now}()$:
   (a) Using `ct-log-timeout` as the timeout, attempt to add the certificate chain in s to l with the `add-chain` [26] or `submit-entry` [27] endpoints.
      – On valid SCT: cache the SFO, then discard it from the buffer of pending SFOs.
      – On any other outcome: go to step 1.

As shown above we take an unusual approach towards auditing SFOs. Rather than following up on an SFO's inclusion status, the underlying certificate chain is *added*

1 Tor's HTTP DirServer codebase can be reused as extension point to interact with the tor daemon, i.e., add another listener.

to a random CT log (see Section 4.3, step 4). As such, the end-entity certificate is disclosed to public scrutiny and the issuing CA will be held accountable if the log in question is honest. A request to add the certificate chain is considered successful if a valid SCT is returned within a timely manner. On any other outcome, such as invalid SCT signature or timeout, the SFO remains in the buffer as the loop breaks and the CTR backs-off in hope that the log becomes available again after wake-up.

Each CTR keeps a least recently used cache of the SFOs it validates, e.g., by hashing the entire SFO or selected SCTs in it. A few megabytes should significantly reduce load on CT logs. Also note that the steps above are done in parallel for each CT log with a `ct-log-info` entry in the consensus. This assures that the unavailability of one CT log does not halt auditing of others.

5 Security Analysis

5.1 Impact of Being Detected

Given a man-in-the-middle attack against TB we consider four different types of *impact*, namely:

None the attack was undetected or detected without knowing any details as to how it was carried out.

Minor the attack was detected due to some cover-up that involved network-wide actions against CTor. This is likely hard to attribute to the actual attacker, but nevertheless it draws much unwanted attention.

Significant the attack generated public cryptographic evidence that proves CA misbehavior.

Catastrophic the attack generated public cryptographic evidence that proves CT log misbehavior.

Our base design can lead to significant impact events. The extended designs in Sections 6–7 aim higher, making it possible to cause events of catastrophic impact.

5.2 Probability of Being Detected

Suppose the attacker mis-issued a certificate chain that is rooted in TB's trust store, and that it is accompanied by enough SCTs from attacker-controlled CT logs to pass TB's CT policy. The resulting SFO is further used to man-in-the-middle a single TB user. Clearly, the attacker-controlled CT logs have no intention to keep any promise of public logging as that would trivially imply significant impact. The attacker's risk of significant exposure is instead bound by the probability that *any* of the three phases in our design fails to propagate the mis-issued SFO to a benign independent CT log. We analyze each phase separately.

In sum the risk of exposure can never be higher than the probability that TB submits an SFO to a genuine CTR successfully, and that risk decreases proportionally with the attacker's increased capability to operate or DoS CTRs as well as independent CT logs.

5.2.1 Phase 1: Submission

The probability of detection cannot exceed the value of `ct-submit-pr`. We focus our analysis on the scenario where the attacker's mis-issued SFO is audited further. There are two cases to consider, namely, the mis-issued SFO is either larger than `ct-max-sfo-bytes` or it is not.

If the SFO is larger than `ct-max-sfo-bytes`, TB blocks until the SFO is submitted and its CT circuit is closed. As such, it is impossible to serve a TB exploit reactively over the man-in-the-middled connection that shuts-down the submission procedure before it occurs. Assuming that forensic traces in tor and TB are unreliable,[2] the sampled CTR identity also cannot be revealed with high certainty afterwards by compromising TB. The attacker may know that the SFO is stored by *some CTR* based on timing, i.e., blocking-behavior is likely measurable and distinct. The important part is not to reveal *which CTR* received a submission: a single Tor relay may be subject to DoS.

If the SFO is smaller or equal to `ct-max-sfo-bytes`, then there is a race between (i) the time it takes for TB to submit the SFO and close its CT circuit against (ii) the time it takes for the attacker to compromise Tor Browser and identify the CTR in question. It is more advantageous to try and win this race rather than being in the unfruitful scenario above. Therefore, the attacker would maximize the time it takes to perform (i) by sending an SFO that is `ct-max-sfo-bytes`. Our design reduced the threat of an attacker that wins this race by using pre-built CT circuits that are closed immediately after use. This makes the attack surface *narrow*, limiting the number of reliable exploits (if any).

2 "tor" (aka "little-t tor") is the tor process TB uses to interact with the Tor network. On marking a circuit as closed in tor, tor immediately schedules the associated data structures to be freed as soon as possible.

Note that the attack surface could, in theory, be eliminated by setting `ct-max-sfo-bytes` to zero. It is too costly in terms of TLS handshake latency, however.

5.2.2 Phase 2: Storage

The probability of detection cannot exceed $1 - (f_{\mathsf{ctr}} + f_{\mathsf{dos}})$, where f_{ctr} is the fraction of malicious CTRs and f_{dos} the fraction of CTRs that suffer from DoS. We analyze the case of successful submission to a genuine CTR.

The time that an SFO is stored is governed by randomness in the `audit_after` timestamp t as well as the back-off delay d, which combined is in the order of minutes (Section 4.1.3). The attacker might try to increase the storage time by making the CT landscape unavailable, forcing additional back-off times and resubmissions. The attacker does not know which independent log to target, however. Since it is not within our threat model to DoS a significant fraction of CT logs simultaneously, any attempt to increase the storage time will be unreliable and thus of little use for our attacker.

The attacker might still try to intervene within the time window at hand, which involves some method that deletes the mis-issued SFO that is stored somewhere. Fortunately, it is not known which CTR to target. This means that any attempt to intervene must target all CTRs. It is clearly not within Tor's threat model to DoS each and every CTR. A less intrusive approach towards intervention would be to simply submit SFOs, posing as genuine TB clients. If enough SFOs are submitted, CTRs could run out of memory and therefore be forced to delete at random (Section 4.3). The threat of such a flood does not apply here: it is impractical to conduct with high certainty within minutes (Section 6.2.1).

5.2.3 Phase 3: Auditing

The probability of detection cannot exceed $1 - f_{\mathsf{log}}$, where f_{log} is the fraction of seemingly independent but attacker-controlled CT logs. We analyze the case of a CTR that sampled a genuine CT log in phase 2, which remains fixed in the event of transient errors. Adding a certificate chain to the log as part of the auditing phase does not leak which CTR holds an SFO *to the attacker* unless all log connections that leave the Tor network are inspected.[3] The attacker in our threat model is only partially present in the network, however. An identifiable CTR can therefore not be targeted reliably before any resubmission takes place. Without resubmission or once logging finally succeeds, the mis-issued certificate is exposed to the public and *significant impact* is achieved.

6 Auditor Extension

Until now we have not *verified* whether the certificate chain of an SFO is publicly logged as promised by the issued SCTs. As such, it is unlikely that a misbehaving CT log will be detected. Our base design can be extended to follow-up on an SFO's inclusion status rather than adding the underlying certificate chain to another CT log. This has the benefit of relaxing our initial trust assumption, namely that some CT logs are honest, as well as making it possible to disclose those CT logs that are dishonest. The downside is added complexity, which introduces additional threats that must be considered.

6.1 Design Sketch

Figure 2 provides an overview of the extended design. TB submits presented SFOs probabilistically to CTRs that are selected at random, and CTRs store the submitted SFOs before any auditing takes place. Here, auditing refers to inclusion verification rather than adding certificate chains. The moment before auditing, the SFO in question is shared with a CTR that acts as a *watchdog*. Unless the auditing CTR receives a timely inclusion proof and acknowledges it to its watchdog, the (now suspicious) SFO is reported to a CT auditor. Phase 1 remains unchanged, some changes are needed in phase 2, and major changes are required in phase 3 as well as the Tor consensus. The extra-info document also includes two new metrics that are related to flooding.

3 We have not specified any further verification of the log's TLS certificate, which means that the attacker can forge it without detection. It is possible to eliminate this attack vector by pinning additional public key information as part of a log's `ct-log-info` item. Assuming the ability to inspect the audit loop, the attacker can trivially identify the CTR by tagging. See Section 6.2.2.

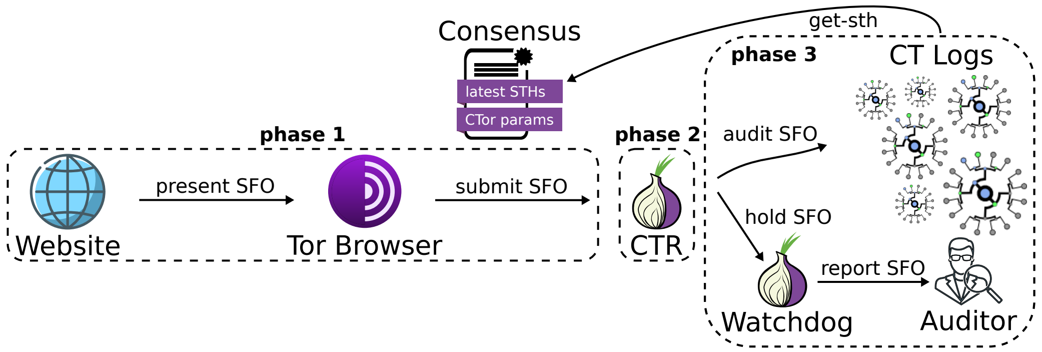

Fig. 2. The auditor extension, where cryptographic evidence of log omission can be collected. The extension changes the consensus to include the latest STHs from CT logs. The CTRs in phase 3 now challenge logs to prove inclusion of certificates from SFOs, using other CTRs as "watchdogs", ensuring that SFOs that are not provably correct are reported to trusted auditors.

6.1.1 Tor Consensus

Tor's consensus should capture a fixed view of the CT landscape by publishing STHs from all recognized logs. A CT log is recognized if a majority of directory authorities proposed a `ct-log-info` item, which contains a log's ID, public key, base URL, MMD, and most recent STH. Each directory authority proposes its own STH, and agrees to use the most recent STH as determined by timestamp and lexicographical order. Since CTRs verify inclusion with regards to SCTs that TB accepts, the CT logs recognized by TB must be in Tor's consensus.

Tor's directory authorities also majority-vote on `ct-auditor` items, which pin base URLs and public keys of CT auditors that watchdogs contact in case that any log misbehavior is suspected. A watchdog triggers if the time specified by `ct-watchdog-timeout` elapses without receiving any acknowledgment. The following auditor submission is governed by a `ct-auditor-timeout`, which, if triggered, results in a resubmission later on.

```
1 :  t ← now() + MMD + random(ct-delay-dist)
2 :  if SCT.timestamp + MMD < now() :
3 :      t ← now() + random(ct-delay-dist)
```

Fig. 3. Algorithm that computes an `audit_after` timestamp t.

Recall from Section 2.1 that an inclusion proof is fetched with regards to an STH. As such, we discard SCTs that cannot be verified due to lack of `ct-log-info` items in the Tor consensus. The sampled CT log l now refers to an entity that issued an SCT in the submitted SFO, and it will be challenged to prove inclusion in phase 3 sometime after the `audit_after` timestamp t elapsed. Figure 3 shows that t takes the log's MMD into account. This is one of two parts that prevent *early signals* to the issuing CT logs that an SFO is being audited. For example, if an SFO is audited before the MMD elapsed, the issuing CT log could simply merge the underlying certificate chain to avoid an MMD violation. This would not yield any improvement with regards to the base design.

6.1.2 Phase 2—Storage

Other than updating the criteria of what it means that an SFO can be audited in phase 3, the computation of l and t changes for a buffer's entry. With regards to some CT circuit, process an incoming SFO s as follows:

1. Close the current circuit to enforce one-time usage.
2. Discard unrecognized SCTs in s whose logs have no corresponding `ct-log-info` items listed in the Tor consensus. Stop if there are no remaining SCTs in s.
3. Stop if s is cached or pending to be audited already.
4. Sample a CT log l that issued a remaining SCT in s.
5. Compute an `audit_after` time t, see Figure 3.
6. Add (l, t, s) to a buffer of pending SFOs.

6.1.3 Phase 3—Auditing

In addition to maintaining a single Tor circuit that is used to interact with CT logs that have `ct-log-items`, a distinct Tor circuit is maintained and rotated that ends at a random watchdog CTR. Given a known CT log l:

1. Sample a delay $d \leftarrow$ random(`ct-backoff-dist`) and wait until d time units elapsed.
2. Connect to a random watchdog CTR.
3. For each pending buffer entry (l', s, t), where $l' = l$, $t <=$ now() and $t <=$ STH.timestamp:
   (a) Share s with the current watchdog.
   (b) Use `ct-log-timeout` and STH.treesize to set a timer and challenge the log to prove inclusion.

- On valid proof: send an acknowledgment to the watchdog, then cache s and discard it.
- On any other outcome: discard s, close circuit to the watchdog CTR, and go to step 1.

An SFO is not audited for inclusion until the log's MMD elapsed *and* there is an STH in the Tor consensus that captures it. As such, a CT log that intends to omit a certificate chain despite promising to merge it within its MMD will not get an early signal that a CTR will audit it. Before auditing, the SFO in question is shared with a watchdog that takes on the responsibility of reporting suspicious SFOs to the pinned CT auditors. An SFO is considered suspicious if it is not acknowledged by the log-challenging CTR as verified within the time specified by `ct-watchdog-timeout`. As argued in Section 6.2.2, it is motivated to use a watchdog: the attacker learns which CTR holds the problematic SFO at the time of auditing, and during the `ct-log-timeout` actions could then be taken to ensure that the SFO does not reach a CT auditor. A watchdog that receives a suspicious SFO reports it to a random CT auditor, and resubmits it later on if the `ct-auditor-timeout` happens to trigger. An important detail is that at most one SFO should await per log in each watchdog circuit, and in the case of resubmissions these are stored as part of the overall SFO buffer to follow the delete-at-random strategy.

6.1.4 Extra-Info Document

Following from the `audit_after` timestamp algorithm in Figure 3, an SFO may be stored for at least an MMD. This results in a relatively large time window during which the attacker can attempt to flood all CTRs in hope that a target SFO is *flushed* from the relevant CTR buffer due to finite memory constraints (Section 4.3). The threat of flooding is discussed in Section 6.2.1. It can be detected if CTRs publish `ct-receive-bytes` and `ct-delete-bytes` in the extra-info document. These metrics indicate how many SFO bytes were received and deleted throughout different time intervals, which is similar to other extra-info metrics such as `read-history`.

6.2 Discussion

We analyze *differences* with regards to the base design in Section 4. The catastrophic impact level is possible because SFOs that cannot be timely validated against Tor's view of the CT landscape are reported. While trust in CT logs is eliminated, it is shifted towards trusted CT auditors that endeavour to follow-up on suspected log misbehavior. The premise of third-party auditors is fundamental in our setting: manually interacting with the wider CT ecosystem to expose a misbehaving log exceeds what could reasonably be expected by a relay operator. Another observation is that these auditors are subject to different operational requirements than a browser-included CT log, which is somewhat analogous to the base design that instead permits additional community logs for the sake of diversity of actors in the ecosystem. Appendix B provides further details on what would be reasonable expectations on auditors.

6.2.1 Phase 2—Storage

The main difference with regards to the base design is that an SFO can be stored much longer. The attacker can maximize the `audit_after` timestamp by using a newly issued SFO, resulting in a storage phase of *at least* an MMD. This time can be extended further by delaying issuance of an STH that would capture the omission: a log must produce an STH every MMD [26, 27]. This means that the maximized storage time is in the order of two MMDs; not considering that it also takes time for an STH to be propagated into the Tor consensus. Most logs use an MMD of 24 hours, resulting in an attack window that ranges from one to two days. A risk-averse attacker should prefer to use the lower bound: suddenly not issuing any new STH is visible, attracting attention.

Following from Tor's threat model, the mis-issued SFO must be stored in volatile memory and not to disk. Two risks emerge as a result of the increased storage time: the CTR in question might be restarted by the operator, and the attacker could flood it by submitting a large volume of SFOs. A risk-averse attacker cannot rely on the former to avoid detection, but the latter might flush a target SFO from a CTR's memory *if deleted at random*. Appendix C shows that the number of SFO submissions k that the attacker needs to flush a buffer of $n > 1$ entries with some probability $p < 1$ is given by:

$$k = \frac{\log(1-p)}{\log(1-\frac{1}{n})} \tag{1}$$

It is recommended that a non-exit relay should have at least 512 MB of memory. If the available bandwidth exceeds 40 Mbps, it should have at least 1 GB [49]. Given that these recommendations are lower bounds, suppose the average memory available to store SFOs is 1 GiB (a conservative assumption). Further, our dataset in Section 8 shows that the average SFO is in the order of 6 KiB. This means that the buffer capacity is $n \leftarrow 174763$ SFOs. Plugging it into Equation 1 for $p \leftarrow \frac{9}{10}$, the attacker's flood must involve $k \leftarrow 402406$ submissions. In other words, 2.3 GiB must be transmitted to flush a single CTR,[4] which takes 7.9–39.3 minutes if its bandwidth is between 8 and 40 Mbps. Thus, it is impractical to flush all CTRs within minutes.

Metrics reported by the Tor project show that there are over 4000 relays that match our CTR criteria set in Section 4.1.1 [48]. As such, a network-wide flush involves the transmission of at least 8.99 TiB. It might sound daunting at first, but distributed over a day it only requires 0.91 Gbps. While we cannot avoid early signals to the logs *and* at the same time prevent flushing without writing anything to disk, it is detectable based on the extra-info document. On the flip-side, *not observing any flushing* adds a large degree of confidence that there are no significant numbers of mis-issued SFOs.

6.2.2 Phase 3—Auditing

The main difference with regards to the base design is that we query the attacker for inclusion proofs; not an independent CT log. As such, there is a time window to act between the time that the attacker learns some CTR audits a mis-issued SFO and until the `ct-log-timeout` elapses. Suppose that the attacker could identify the CTR in question at the time of auditing, i.e., despite our design (re)using a Tor circuit for all inclusion proofs. Clearly, the query timeout must be a few seconds to avoid premature auditor reporting. This leaves a large enough window to simply DoS the CTR in question. Our design makes no attempt to hide the CTR's identity while querying the log. For example, the attacker can trivially *tag* each CTR by submitting many distinct SFOs, and upon seeing them in the audit loop the exact identity is revealed. We mitigated this threat using a watchdog, sending the SFO upfront *before* auditing. The attacker does not know the watchdog identity based on the same premises that the attacker does not know which CTR stores an SFO during the storage phase. It should be noted that the use of watchdog CTRs give the attacker a *second shot* at being selected at random.

Another difference is that inclusion proofs are based on STHs that the directory authorities fetch from attacker-controlled CT logs, agreeing on which ones to use via deterministic rules. This means that the attacker controls which STHs go into the Tor consensus. Once an STH is announced, it follows from Tor's threat model that it is fixed because a threshold of directory authorities are benign. As such, CTRs have access to the same (in)consistent view of the CT landscape. Fortunately, any inconsistent view that makes it into the Tor consensus is trivially detected: the announced STHs are public and auditable by anyone. Therefore, the attacker should be deterred from creating split-views in Tor. Other user agents could benefit from Tor's audited view of CT logs.

7 Log Extension

If we allow small—yet significant—changes to the CT landscape, it is possible to avoid inclusion verification and the involved complexities as described in Section 6. The extension is based on returning back to the premise of some CT logs being honest, but still extend the base design (Section 4) to *detect* misbehaving CT logs. The extension is nearly identical to the base design: TB submits presented SFOs probabilistically to randomly selected CTRs in phase 1, which store the submitted SFOs in phase 2 before any auditing takes place in phase 3. Here, *auditing* refers to adding a *full SFO* to an independent CT log; not just the underlying certificate chain as in the base design. This is the main difference in the log extension. By also adding the SCTs to other CT logs, the issuing CT logs can be held accountable.

The prerequisite for such an extension to work is that CT logs support an additional API endpoint: `add-sfo`. First we describe how this endpoint could be operated so that *the only change* is that certificate chains and SCTs are added. Next, we explain how some complexity could be moved into Tor but at the cost of bringing back the threat of flooding (see Section 6.2.1).

Approach 1. Recall from Section 6.1.2 that there must not be any early signals that allow misbehaving CT logs to reactively merge certificate chains before any MMD promise is violated. To ensure that this is the case, the `add-sfo` endpoint could require that the added SFO will be merged first after the MMD (of the issuing log) has elapsed, such that the log (should have) produced a STH that captured the omission by then. For example, the SCT that is returned by the `add-sfo` endpoint could have a timestamp that is future-dated

4 As a corner case and implementation detail, it is important that TB and CTRs *reject* SFOs that are bogus in terms of size: it is a trivial DoS vector to load data indefinitely. Analysis based on, e.g., 1 MiB SFOs also requires 2.3 GiB of data to flush.

by at least two MMDs, and then it is not considered for merging until that timestamp is in the present. The appeal of *delayed merges* is that SFOs need not be stored any longer than what is necessary for privacy at CTRs, completely avoiding the threat of network-wide flushes within our threat model. If so, the analysis in Section 4 applies without modifications, except that misbehaving CAs *and* CT logs are exposed.

Approach 2. The other option is to make minimal changes to the CT landscape by operating the `add-sfo` endpoint without any other expectation than that it must allow SCTs in addition to a certificate chain. To avoid early signals, CTRs should employ the same delay strategy as suggested for CT logs above: wait at least two MMDs before adding a (newly issued) SFO. This is essentially the same as an attacker that maximized the storage phase by waiting until the last second to produce an STH as discussed in Section 6.2.1, except that such behavior is assumed rather than confirmed. Clearly this brings flooding back to the table, and CTRs need the extra-info metrics from Section 6.1.4 to allow detection.

8 Performance

Our performance evaluation estimates the expected overhead as the base design of CTor is instantiated. Mani *et al.* showed that up to 140 million websites are visited over Tor on a daily basis (upper bound of 95% confidence interval) [30]. Our analysis also needs to know the size of a typical SFO, as well as the distribution of presented SFOs per website. To this end we collected a dataset based on the most popular webpages submitted to Reddit (r/frontpage, all time) as of December 4, 2019. An average certificate chain is 5440 bytes, and is seldom accompanied by more than a few SCTs. As such, we assume that a typical SFO is 6 KiB. No certificate chain exceeded 20 KiB, and it is likely a conservative value for `ct-max-sfo-bytes` that avoids blocking in the TLS handshake. The average number of SFOs per website was seven, which might be an overestimate due to collecting the dataset with fresh Chromium instances and NetLog *without* filtering SFOs related to the initial call-home on start-up behavior.

Now we take a closer look at circuit, bandwidth, and memory overhead using a modest 10% submission probability and an average 10 minute auditing delay at CTRs. The positive effect of CTR caching is *disregarded* for simplicity, or, viewed from a different perspective: we can already show that the incurred CTor overhead is acceptable and/or insignificant from basic sketching.

Circuit overhead. Equation 2 shows the average circuit overhead for TB over time, where p is the submit probability and $\mathcal{D}$ a distribution describing how many SFOs are presented per website visit.

$$\frac{p}{n}\sum_{i=1}^{n} c_i, \text{where } c_i \leftarrow_{\$} \mathcal{D} \qquad (2)$$

Using our submission probability and approximated $\mathcal{D}$ with $n \leftarrow 8858$ data points, the circuit overhead is 0.70 on average, where the circuits are short-lived and transport on average 6 KiB. Further, each CTR also maintains a single long-lived circuit at all times to interact with logs in the CT landscape.

Bandwidth overhead. An SFO that is audited further gets submitted to a CTR over a Tor circuit. Later on, the underlying certificate chain is added to an independent CT log using another Tor circuit. This means that the bandwidth of six relays are involved, one of which must be an exit. Given the daily website visits of Mani *et al.* as well as our submission probability and SFO distribution, this amounts to 98 million SFO submissions and thus 560.8 GiB per day. Converted into bits per second and taking six Tor relays into account yields 334.5 Mbps in total. Such order of overhead is small when compared to Tor's capacity: 450 Gbps [47].

Memory overhead. During the average storage time of 10 minutes, there are 680.6 k SFO submissions that are distributed randomly across more than 4000 CTRs. This results in an average buffer size of 170 SFOs, which corresponds to 1.0 MiB memory. Such order of overhead is small when compared to the recommended relay configuration: at least 512 MiB [49]. Note that the expected memory overhead does not increase much even if the storage phase is extended for *newly issued* SFOs as in Section 6. For example, suppose that all SFOs had 90 day lifetimes due to being issued by Let's Encrypt [1]. On average, 1.1% of the submitted SFOs must then be stored for the order of an MMD rather than 10 minutes.

9 Privacy

There is an inherent privacy problem in the setting due to how CT is designed and deployed. A browser, like TB, that wishes to validate that SFOs presented to it are *consistent* and *included* in CT logs must directly or indirectly interact with CT logs wrt. its observed SFOs. Without protections like Private Information Retrieval [5] that require server-side support or introduction of additional parties and trust assumptions [20, 29], exposing SFOs to any party risks leaking (partial) information about the browsing activities of the user.

Given the constraints of the existing CT ecosystem, CTor is made privacy-preserving thanks to the distributed nature of Tor with its anonymity properties and high-uptime relays that make up the Tor network. First, all communication between TB, CTRs, CT logs, and auditors are made over full Tor-circuits. This is a significant privacy-gain, not available, e.g., to browsers like Chrome that in their communications would reveal their public IPv4-address (among a number of other potentially identifying metadata). Secondly, the use of CTRs as intermediaries probabilistically delays the interaction with the CT logs—making correlating TB user browsing with CT log interaction harder for attackers—and safely maintain a dynamic cache of the most commonly already verified SFOs[5]. While regular browsers like Chrome could maintain a cache, TB and its security and privacy goals (see Section 2.2) prohibit such shared (persisted) dynamic state.

CTRs are also essential for security in the setting of the auditor extension. A rational attacker will produce SFOs with SCT timestamps that are too new to have to be part of CT logs at the time of the attack. TB cannot store SFOs for any extended time period without violating its security and privacy goals, not to mention that presumably most TB sessions are not maintained for days (as required, see analysis in Section 6.2) but rather minutes [3]. Further, buffering at TB is not an option, since an attacker could compromise TB shortly after presenting the forged SFO.

The main limitation of CTor in terms of privacy is that CTor continuously leaks to CT logs—and to a *lesser extent* auditors in one of our extensions due to break-and-backoff behavior if the logs are unresponsive—a fraction of certificates of websites visited using TB to those that operate CT logs. This provides to a CT log a partial list of websites visited via the Tor network over a period of time (determined by `ct-delay-dist`), together with some indication of distribution based on the number of active CTRs. It does not, however, provide even pseudonymously any information about which sites individual users visit, much less with which patterns or timing. As such it leaks significantly less information than does OCSP validation by TB or DNS resolution at exit-relays [19], both of which indicate visit activity in real time to a comparably small number of entities.

Another significant privacy limitation is that relays with the CTR flag learn real-time browser behavior of Tor users. Relays without the Exit flag primarily only transport encrypted Tor-traffic between clients and other relays, never to destinations. If such relays are given the CTR flag—as we stated in the base design, see Section 4.1.1—then this might discourage some from running Tor relays unless it is possible to opt out. Another option is to give the CTR flag only to exit relays, but this *might be* undesirable for overall network performance despite the modest overhead of CTor (Section 8). Depending on the health of the network and the exact incremental deployment of CTor, there are different trade-offs. For example, fewer CTRs with higher bandwidth and less memory combined makes it easier to perform flooding attacks that flush the entire network.

10 Related Work

Google Chrome and Apple's Safari enforce CT by mandating that every TLS certificate must be accompanied by two independent SCTs [34, 37]. We proposed that TB should follow suit, but, unlike any other web browser that enforces CT, CTor provides *concrete next steps* that relax the centralized trust which is otherwise and evidently misplaced in CT logs [31, 50, 52, 53]. Several proposals surfaced that aim to do better than today's CT policies, targeting omissions and split-views.

Laurie proposed that inclusion proofs could be fetched over DNS to avoid additional privacy leaks, i.e., a user's browsing patterns are already exposed to the DNS resolver but not the logs in the CT landscape [25]. CT/bis provides the option of serving stapled inclusion proofs as part of the TLS handshake in an extension, an OCSP response, or the certificate itself [27]. Lueks and Goldberg proposed that a separate database of inclusion proofs could be maintained that supports information-theoretic PIR [29]. Kales *et al.* improved scalability by reducing the size of each entry in the PIR database at the cost of transforming logs into multi-tier Merkle trees, and additionally, showed how the upper tier could be expressed as a two-server computational PIR database to ensure that any inclusion proof can be computed privately on-the-fly [20]. Nordberg *et al.* avoid inclusion

5 Note that the long-tail nature of website popularity—over the Tor network as well as the regular web [30]—means that even a relatively small cache of the most frequently observed SFOs will lead to a significant reduction of queries to the CT logs.

proof fetching by hanging on to presented SFOs, handing them back to the same origin at a later time [33]. In contrast, CTor protects the user's privacy without any persited browser state by submitting SFOs on independent Tor circuits to CTRs, which in turn add random noise before there is any log interaction to speak of. The use of CTRs enable caching similar to CT-over-DNS, but it does not put the logs in the dark like PIR could.

Inclusion proofs are only meaningful if everyone observes the same consistent STHs. One option is to configure client software with a list of entities that they should gossip with, e.g., CT monitors [4], or, browser vendors could push a verified view [54]. Such trusted auditor relationships may work for some but not others [33]. Chuat *et al.* proposed that HTTPS clients and HTTPS servers could pool STHs and consistency proofs which are gossiped on website visits [6]. Nordberg *et al.* suggested a similar variant, reducing the risk of user tracking by pooling fewer and recent STHs [33]. Dahlberg *et al.* noted that such privacy-insensitive STHs need not be encrypted, which could enable network operators to use programmable data planes to provide gossip as-a-service [10]. Syta *et al.* proposed an alternative to reactive gossip mechanisms by showing how an STH can be cosigned efficiently by many independent witnesses [55]. A scaled-down version of witness cosigning could be instantiated by cross-logging STHs in other CT logs [14], or, in other append-only ledgers [56]. CTor's extended design in Section 6 ensures that anyone connected to the Tor network is on the same view by making STHs public in the Tor consensus. In contrast, the base design is not about catching log misbehavior, and the extended design in Section 7 exposes logs that misbehave *without* fetching inclusion proofs.

Nordberg proposed that Tor clients could enforce public logging of consensus documents and votes [32]. Such an initiative is mostly orthogonal to CTor, as it strenghtens the assumption of a secure Tor consensus by enabling detection of compromised signing keys rather than mis-issued TLS certificates. Winter *et al.* proposed that TB could check self-signed TLS certificates for extact matches on independent Tor circuits [58]. Alicherry *et al.* proposed that any web browser could double-check TLS certificates on first encounter using alternative paths and Tor, again, looking for certificate mismatches and generating warnings of possible man-in-the-middle attacks [2]. The submission phase in CTor is similar to such double-checking, expect that there is no normal-case TLS handshake blocking, browser warnings, or strict assumptions regarding the attacker's location.

11 Conclusion

We proposed privacy-preserving and incrementally-deployable support for CT in Tor: CTor. Our design is composed of a base that discloses mis-issued certificates to public scrutiny by using the CT landscape against the attacker, and two orthogonal extensions that additionally catch misbehaving CT logs. The use of Tor relays that cache TB-observed SFOs and add random auditing delays is central within our setting, both for privacy and security. Our analysis shows that the attacker's best bet to break the base design and the following log extension is to control and/or DoS a large majority of the entire CTor system, or, access a reliable TB zero-day that escalates privileges within a tiny time window. Our auditor extension does not rely on trust in the CT landscape, but at the cost of making flooding attacks that flush the network practical. Network-wide flushes are trivially detectable but not necessarily attributable.

Mitigation of network-wide attacks take us outside of Tor's threat model and therefore outside of ours too. That said, we cannot ignore such attacks given the strong attacker we consider, namely, one that controls a trusted CA and CT logs. To this end, CTor is designed so that Tor can *adapt* in response to interference. For example, in TB the `ct-max-sfo-bytes` could reactively be set such that all SFOs must be sent to a CTR before accepting any HTTPS application-layer data to counter zero-days, and the submit probability `ct-submit-pr` could be increased if ongoing attacks are suspected. When it comes to the storage phase, the consensus can minimize or maximize the storage time by tuning a log's MMD in the `ct-log-info` item, as well as updating the distributions that add random auditing delays. The behavior of the auditing phase can also be tuned, e.g., by updating log operator relationships.

Our base design lets Tor play a vital role in CT's overall goal of detecting certificate mis-issuance. It also exemplified an new style of auditing, where certificate chains are cross-logged rather verified for inclusion. The extended versions of CTor involve increased deployment burdens, either within Tor or the CT landscape. The wider web could greatly benefit from both, however. The log extension provides a single coherent place to report suspected log misbehavior, making it similar to a well-known auditor. The auditor extension turns Tor into a system for maintaining a probabilistically-verified view of the entire CT landscape, provided in Tor's consensus for anyone to use as a basis of trust on the wider web.

References

[1] J. Aas, R. Barnes, B. Case, Z. Durumeric, P. Eckersley, A. Flores-López, J. A. Halderman, J. Hoffman-Andrews, J. Kasten, E. Rescorla, S. D. Schoen, and B. Warren. Let's encrypt: An automated certificate authority to encrypt the entire web. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2473–2487. ACM, 2019.

[2] M. Alicherry and A. D. Keromytis. Doublecheck: Multi-path verification against man-in-the-middle attacks. In *Proceedings of the 14th IEEE Symposium on Computers and Communications (ISCC 2009), July 5-8, Sousse, Tunisia*, pages 557–563. IEEE Computer Society, 2009.

[3] J. Amann and R. Sommer. Exploring tor's activity through long-term passive TLS traffic measurement. In T. Karagiannis and X. A. Dimitropoulos, editors, *Passive and Active Measurement - 17th International Conference, PAM 2016, Heraklion, Greece, March 31 - April 1, 2016. Proceedings*, volume 9631 of *Lecture Notes in Computer Science*, pages 3–15. Springer, 2016.

[4] M. Chase and S. Meiklejohn. Transparency overlays and applications. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 168–179. ACM, 2016.

[5] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 41–50. IEEE Computer Society, 1995.

[6] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri. Efficient Gossip Protocols for Verifying the Consistency of Certificate Logs. In *2015 IEEE Conference on Communications and Network Security, CNS 2015, Florence, Italy, September 28-30, 2015*, pages 415–423. IEEE, 2015.

[7] J. Clark and P. C. van Oorschot. Sok: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 511–525. IEEE Computer Society, 2013.

[8] M. Conti, S. Crane, T. Frassetto, A. Homescu, G. Koppen, P. Larsen, C. Liebchen, M. Perry, and A. Sadeghi. Selfrando: Securing the tor browser against de-anonymization exploits. *PoPETs*, 2016(4):454–469, 2016.

[9] R. Dahlberg and T. Pulls. Verifiable light-weight monitoring for certificate transparency logs. In N. Gruschka, editor, *Secure IT Systems - 23rd Nordic Conference, NordSec 2018, Oslo, Norway, November 28-30, 2018, Proceedings*, volume 11252 of *Lecture Notes in Computer Science*, pages 171–183. Springer, 2018.

[10] R. Dahlberg, T. Pulls, J. Vestin, T. Høiland-Jørgensen, and A. Kassler. Aggregation-based certificate transparency gossip. In *The Thirteenth International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*, pages 120–127, October 2019.

[11] G. Danezis and P. Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In N. Borisov and I. Goldberg, editors, *Privacy Enhancing Technologies: Eighth International Symposium, PETS 2008*, pages 151–166. Springer-Verlag, LNCS 5134, July 2008.

[12] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In M. Blaze, editor, *Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA*, pages 303–320. USENIX, 2004.

[13] B. Dowling, F. Günther, U. Herath, and D. Stebila. Secure logging schemes and certificate transparency. In I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. A. Meadows, editors, *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*, volume 9879 of *Lecture Notes in Computer Science*, pages 140–158. Springer, 2016.

[14] D. Drysdale. Minimal gossip, May 2018. https://github.com/google/certificate-transparency-go/blob/master/gossip/minimal/README.md, accessed 2020-05-27.

[15] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In K. Papagiannaki, P. K. Gummadi, and C. Partridge, editors, *Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013*, pages 291–304. ACM, 2013.

[16] S. Englehardt and A. Narayanan. Online tracking: A 1-million-site measurement and analysis. In E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1388–1401. ACM, 2016.

[17] S. Eskandarian, E. Messeri, J. Bonneau, and D. Boneh. Certificate transparency with privacy. *PoPETs*, 2017(4):329–344, 2017.

[18] N. S. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on Tor using long paths. In *Proceedings of the 18th USENIX Security Symposium*, pages 33–50, Montreal, Canada, August 2009. USENIX Association.

[19] B. Greschbach, T. Pulls, L. M. Roberts, P. Winter, and N. Feamster. The effect of DNS on Tor's anonymity. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017.* The Internet Society, 2017.

[20] D. Kales, O. Omolola, and S. Ramacher. Revisiting user privacy for certificate transparency. In *IEEE European Symposium on Security and Privacy, EuroS&P 2019, Stockholm, Sweden, June 17-19, 2019*, pages 432–447. IEEE, 2019.

[21] D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-Go MIXes: Providing probabilistic anonymity in an open system. In D. Aucsmith, editor, *Information Hiding: Second International Workshop, IH 1998*, pages 83–98, Portland, Oregon, USA, April 1998. Springer-Verlag, LNCS 1525.

[22] N. Korzhitskii and N. Carlsson. Characterizing the root landscape of certificate transparency logs. *CoRR*, abs/2001.04319:1–9, January 2020.

[23] A. Langley. Enhancing digital certificate security, January 2013. https://security.googleblog.com/2013/01/enhancing-digital-certificate-security.html, accessed 2020-05-28.

[24] B. Laurie. Certificate transparency. *Commun. ACM*, 57(10):40–46, 2014.
[25] B. Laurie. Certificate transparency over DNS. Design document draft-ct-over-dns-01, Google Inc., March 2016.
[26] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency. RFC 6962, IETF, 2013.
[27] B. Laurie, A. Langley, E. Kasper, E. Messeri, and R. Stradling. Certificate Transparency Version 2.0. Internet-draft draft-ietf-trans-rfc6962-bis-34, IETF, November 2019. work in progress.
[28] B. Li, J. Lin, F. Li, Q. Wang, Q. Li, J. Jing, and C. Wang. Certificate transparency in the wild: Exploring the reliability of monitors. In L. Cavallaro, J. Kinder, X. Wang, and J. Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2505–2520. ACM, 2019.
[29] W. Lueks and I. Goldberg. Sublinear scaling for multi-client private information retrieval. In R. Böhme and T. 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.
[30] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr. Understanding tor usage with privacy-preserving measurement. In *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*, pages 175–187. ACM, 2018.
[31] B. McMillion. Un-incorporated scts from gdca1, August 2018. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/Emh3ZaU0jqI, accessed 2020-05-20.
[32] L. Nordberg. Tor consensus transparency, June 2014.
[33] L. Nordberg, D. K. Gillmor, and T. Ritter. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05, IETF, 2018. work in progress.
[34] D. O'Brien. Certificate transparency enforcement in Google Chrome, 2018. https://groups.google.com/a/chromium.org/forum/#!msg/ct-policy/wHILiYf31DE/iMFmpMEkAQAJ, accessed 2020-03-12.
[35] M. Perry, E. Clark, S. Murdoch, and G. Koppen. The design and implementation of the Tor Browser [draft], June 2018.
[36] J. R. Prins and B. U. Cybercrime. Diginotar certificate authority breach "operation black tulip". *Fox-IT, November*, page 18, 2011.
[37] Apple Inc. Apple's certificate transparency policy, 2019. https://web.archive.org/web/20190321072334/https://support.apple.com/en-us/HT205280, accessed 2020-03-12.
[38] Bugzilla. Implement certificate transparency support (rfc 6962), 2020. https://web.archive.org/web/20200202042716/https://bugzilla.mozilla.org/show_bug.cgi?id=1281469, accessed 2020-05-29.
[39] Catalin Cimpanu. Exploit vendor drops tor browser zero-day on twitter | zdnet, 2018. https://web.archive.org/web/20200529194530/https://www.zdnet.com/article/exploit-vendor-drops-tor-browser-zero-day-on-twitter/, accessed 2020-05-29.
[40] firstwatch at sigaint.org. [tor-talk] javascript exploit, 2016. https://web.archive.org/web/20200529194407/https://lists.torproject.org/pipermail/tor-talk/2016-November/042639.html, accessed 2020-05-29.
[41] Google Inc. (n.d.). HTTPS encryption on the web. https://transparencyreport.google.com/https/overview?hl=en, accessed 2020-05-19.
[42] Mozilla. Mozilla research grants 2019h1, 2019. https://web.archive.org/web/20200528174708/https://mozilla-research.forms.fm/mozilla-research-grants-2019h1/forms/6510, accessed 2020-05-28.
[43] Mozilla (n.d.). SSL ratios. https://docs.telemetry.mozilla.org/datasets/other/ssl/reference.html, accessed 2020-05-19.
[44] Zerodium. Zerodium - tor browser zero-day exploit bounty 2017 (expired), 2017. https://web.archive.org/web/20200528175225/https://zerodium.com/tor.html, accessed 2020-05-28.
[45] Zerodium. Zerodium - how to sell your 0day exploit to zerodium, 2020. https://web.archive.org/web/20200521151311/https://zerodium.com/program.html, accessed 2020-05-21.
[46] Apple Inc. Apple's certificate transparency log program, January 2019. https://support.apple.com/en-om/HT209255, accessed 2020-05-31.
[47] Tor project. Advertised and consumed bandwidth by relay flag, May 2020. https://metrics.torproject.org/bandwidth-flags.html, accessed 2020-05-30.
[48] Tor project. Relays by relay flag, May 2020. https://metrics.torproject.org/relayflags.html, accessed 2020-05-29.
[49] Tor project (n.d.). Relay requirements. https://community.torproject.org/relay/relays-requirements/, accessed 2020-05-29.
[50] J. Rowley. CT2 log compromised via salt vulnerability, May 2020. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/aKNbZuJzwfM, accessed 2020-05-20.
[51] A. Serjantov, R. Dingledine, and P. Syverson. From a trickle to a flood: Active attacks on several mix types. In F. A. Petitcolas, editor, *Information Hiding: 5th International Workshop, IH 2002*, pages 36–52. Springer-Verlag, LNCS 2578, 2002.
[52] R. Sleevi. Upcoming CT log removal: Izenpe, May 2016. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOorKuhL1vA, accessed 2020-05-20.
[53] R. Sleevi. Upcoming log removal: Venafi CT log server, March 2017. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/KMAcNT3asTQ, accessed 2020-05-20.
[54] R. Sleevi and E. Messeri. Certificate transparency in Chrome: Monitoring CT logs consistency, March 2017. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZlZe_z-ls/edit?pref=2&pli=1, accessed 2020-05-27.
[55] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 526–545. IEEE Computer Society, 2016.
[56] A. Tomescu and S. Devadas. Catena: Efficient non-equivocation via bitcoin. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 393–409. IEEE Computer Society, 2017.

[57] Getting up to speed on Tor's past, present, and future. https://2019.www.torproject.org/docs/documentation.html.en. Retrieved May 25, 2020.

[58] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. R. Weippl. Spoiled onions: Exposing malicious tor exit relays. In E. D. Cristofaro and S. J. Murdoch, editors, *Privacy Enhancing Technologies - 14th International Symposium, PETS 2014, Amsterdam, The Netherlands, July 16-18, 2014. Proceedings*, volume 8555 of *Lecture Notes in Computer Science*, pages 304–331. Springer, 2014.

A CT Trust Anchors

The standardized CT protocol suggests that a log's trust anchors should "usefully be the union of root certificates trusted by major browser vendors" [26, 27]. Apple further claims that a log in their CT program "must trust all root CA certificates included in Apple's trust store" [46]. This bodes well for CTor: we assumed that the existence of independent log operators implies the ability to add certificate chains (Section 4) or complete SFOs (Section 7) into logs that the attacker does not control. The preprint of Korzhitskii and Carlsson show that the overlap between root certificates that are trusted by major browser vendors and CT logs increased over time, but, *no log* accepts all roots to date [22]. Let's Encrypt is likely a commonly excluded CA, following from the large volume of issued certificates which might deter some logs (or operators) from including them as a trust anchor. All is not lost, however. A few notes:

- Even if there are no independent logs available for some CA, there is still significant value in practise to (re)add the presented certificate chain and SCTs to the same logs that supposedly logged them. For example, the attacker might have access to signing keys but not the actual infrastructure. In such a scenario the mis-issued certificate makes it into the public. Given SCTs, the log operators also learn that their signing keys were compromised at some point.
- Most log operators only exclude a small fraction of widely accepted root certificates: 1–5% [22]. This narrows down the possible CAs that the attacker must control by 1–2 orders of magnitude. In other words, to be entirely sure that CTor would (re)add a mis-issued SFO to the attacker-controlled CT logs, this smaller group of CAs must issue the underlying certificate. It is arguably harder to take control of Let's Encrypt than, say, a smaller CA that may be coerced to misbehave by national law enforcement.
- Given that the coverage of CT logs increased over time, such a trend would likely continue if the ecosystems around CT benefit from it (e.g., CTor).
- The two designs that rely on independent CT logs also permit community-driven CT logs that are not recognized by any major browser vendor. Such logs, if included in the Tor consensus, can be viewed as parties facilitating auditing of the CT landscape. Such interest exist already, e.g., in the form of emerging CT monitoring services [9, 28]. It could also be an alternative for CAs that, somehow, need to stay relevant in light of today's *freely available* certificate issuance as provided by Let's Encrypt [1].

B Trusted Auditor

An trusted auditor is expected to accept SFOs at a dedicated endpoint, endeavoring to validate the inclusion status of each SCT with regards to the first STH in the Tor consensus that elapsed the log's MMD. If a log appears to function correctly except for some evidence that cannot be resolved despite multiple attempts that span a given time period, it is paramount that the auditor software alerts its operator who can investigate the issue further before submitting a full report to the CT-policy mailing list. Cases of misbehavior, in detail:

- If the log fails to provide a valid inclusion proof for an SCT with regards to the first applicable STH in the Tor consensus (certificate omission).
- If the log fails to provide a valid consistency proof between two STHs in the Tor consensus (split-view).
- If a published STH is future-dated or backdated more than an MMD (general log misbehavior).
- If a log ignores some CTRs (uptime misbehavior).

An unresponsive log can be suspected by observing the watchdog report frequency, and possibly confirmed by querying the log in question from different vantage points. Among other extra-info metrics that go beyond received and deleted SFO-bytes that the announced CT auditors should check, it could be valuable to publish the extent to which CTR-to-log interactions fail.

While not within our threat model, we do encourage that the announced auditors verify whether STHs in the Tor consensus are consistent with external views of the CT landscape. For example, operate an STH pollination endpoint [33] and fetch STHs actively from many diverse vantage points using Tor, VPN services, DoH resolvers, and RIPE Atlas (to mention a few options). This goes back to the point of general ecosystem value.

C Flushing a Single CTR

Let n be the number of SFOs that a CTR can store in its buffer. The probability to sample a target SFO is thus $\frac{1}{n}$, and the probability to not sample a target SFO is $q = 1 - \frac{1}{n}$. The probability to not sample a target SFO after k submissions is q^k. Thus, the probability to sample the relevant buffer index at least once is $p = 1 - q^k$. Solving for k we get: $k = \frac{\log(1-p)}{\log(q)}$. Substituting q for $1 - \frac{1}{n}$ yields Equation 1, which was to be shown.