Rasmus Dahlberg*, Tobias Pulls, Tom Ritter, and Paul Syverson

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

Abstract: The security of the web improved greatly throughout the last couple of years. A large majority of the web is now served encrypted as part of HTTPS, and web browsers accordingly moved from positive to negative security indicators that warn the user if a connection is insecure. A secure connection requires that the server presents a valid certificate that binds the domain name in question to a public key. A certificate used to be valid if signed by a trusted Certificate Authority (CA), but web browsers like Google Chrome and Apple's Safari have additionally started to mandate Certificate Transparency (CT) logging to overcome the weakest-link security of the CA ecosystem. Tor and the Firefox-based Tor Browser have yet to enforce CT.

In this paper, we present privacy-preserving and incrementally-deployable designs that add support for CT in Tor. Our designs go beyond the currently deployed CT enforcements that are based on blind trust: if a user that uses Tor Browser is man-in-the-middled over HTTPS, we probabilistically detect and disclose cryptographic evidence of CA and/or CT log misbehavior. The first design increment allows Tor to play a vital role in the overall goal of CT: detect mis-issued certificates and hold CAs accountable. We achieve this by randomly cross-logging a subset of certificates into other CT logs. The final increments hold misbehaving CT logs accountable, initially assuming that some logs are benign and then without any such assumption. Given that the current CT deployment lacks strong mechanisms to verify if log operators play by the rules, exposing misbehavior is important for the web in general and not just Tor. The full design turns Tor into a system for maintaining a probabilistically-verified view of the CT log ecosystem available from Tor's consensus. Each increment leading up to it preserves privacy due to and how we use Tor.

Keywords: Certificate Transparency, Tor

*Corresponding Author: Rasmus Dahlberg: Karlstad University, E-mail: rasmus.dahlberg@kau.se
Tobias Pulls: Karlstad University, E-mail: tobias.pulls@kau.se
Tom Ritter: E-mail: tom@ritter.vg

1 Introduction

Metrics reported by Google and Mozilla reveal that encryption on the web skyrocketed the past couple of years: at least 84% of all web pages load using HTTPS [26, 44]. An HTTPS connection is initiated by a TLS handshake where the client's 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 is really connecting to Mozilla, and not, say, Google. A certificate specifies the cryptographic key-material for a given domain name, and it is considered valid if it is digitally signed by a Certificate Authority (CA) that the web browser trusts.

It is a long-known problem that the CA trust model suffers from weakest-link security: web browsers allow hundreds of CAs to sign arbitrary domain-name to key-bindings, which means that it suffices to compromise a single CA to acquire any certificate [9, 18]. Motivated by prominent CA compromises, such as the issuance of fraudulent certificates for *.google.com, *.mozilla.org and *.torproject.org by DigiNotar [49], multiple browser vendors mandated that certificates issued by CAs must be publicly disclosed in Certificate Transparency (CT) logs to be valid. The idea behind CT is that, by making all CA-issued certificates transparent, mis-issued ones can be detected *after the fact* [34, 36, 37]. The appropriate actions can then be taken to keep the wider web safe, e.g., by investigating the events that lead up to a particular incident, removing or limiting trust in the offending CA, and revoking affected certificates. Google Chrome and Apple's Safari currently enforce CT by augmenting the TLS handshake to require cryptographic proofs from the server that the presented certificate *will appear* in CT logs that the respective web browsers trust [3, 24].

In addition to increased encryption on the web, the ability to access it anonymously matured as well. Tor with its Tor Browser has millions of daily users [16, 40], and efforts are ongoing to mature the technology for

Paul Syverson: U.S. Naval Research Laboratory, E-mail: paul.syverson@nrl.navy.mil

wider use [43]. Tor Browser builds on-top of Mozilla's Firefox: it relays traffic between the user and the web server in question by routing everything through the Tor network, which is composed of thousands of volunteer-run relays that are located across the globe [62]. Just like attackers may wish to break security properties of HTTPS, it may also be of interest to break the anonymity provided by Tor. A common technique for deanonymization (known to be used in practice) is to compromise Tor Browser instead of circumventing the anonymity provided by Tor [5, 10, 22, 69]. Web browsers like Firefox (or forks thereof) are one of the most complex software types that are widely used today, leading to security vulnerabilities and clear incentives for exploitation. For example, the exploit acquisition platform Zerodium offers up to $100,000 for a Firefox zero-day exploit that provides remote code execution and local privilege escalation (i.e., full control of the browser) [70].

An attacker that wishes to use such an exploit to compromise and then ultimately deanonymize a Tor Browser user has to deliver the exploit somehow. Since the web is mostly encrypted, this primarily needs to take place over an HTTPS connection where the attacker controls the content returned by the web server. While there are numerous possible ways that the attacker can accomplish this, e.g., by compromising a web server that a subset of Tor Browser users visit, another option is to *impersonate* one or more web servers by acquiring fraudulent certificates. Due to the Tor network being run by volunteers, getting into a position to perform such an attack is relatively straightforward: the attacker can volunteer to run malicious exit relays [68]. The same is true for an attacker that wishes to man-in-the-middle connections made by Tor Browser users. In some cases a Tor Browser exploit may not even be needed for deanonymization, e.g., the attacker can observe if the user logs-on to a service linking an identity.

1.1 Introducing CTor

We propose an incrementally deployable and privacy-preserving design that is henceforth referred to as CTor. By bringing CT to Tor, HTTPS-based man-in-the-middle attacks against Tor Browser users can be detected *after the fact* when conducted by attackers that:
1. can acquire any certificate from a trusted CA,
2. with the necessary cryptographic proofs from enough CT logs so that Tor Browser accepts the certificate as valid without the attacker making it publicly available in any of the controlled logs, and

3. with the ability to gain full control of Tor Browser shortly after establishing an HTTPS connection.

The first and third capabilities are motivated directly by shortcomings in the CA ecosystem as well as how the anonymity of Tor Browser is known to be attacked. The second capability assumes the same starting point as Google Chrome and Apple's Safari, namely, that the logs are trusted to *promise* public logging, which is in contrast to being untrusted and thus forced to *prove* it. This is part of the gradual CT deployment that avoided breakage on the web [55]. Therefore, we start from the assumption that Tor Browser accepts a certificate as valid if accompanied by two independent promises of public logging. The limitation of such CT enforcement is that it is trivially bypassed by an attacker that controls two seemingly independent CT logs. This is not to say that trusting the log ecosystem would be an insignificant Tor Browser improvement when compared to no CT at all, but CTor takes us several steps further by relaxing and ultimately eliminating the trust which is currently (mis)placed in today's browser-recognized CT logs. We already observed instances of CT logs that happened to violate their promises of public logging [41], show inconsistent certificate contents to different parties [52, 53], and get their secret signing keys compromised due to disclosed remote code-execution vulnerabilities [50].

The first design increment uses the CT landscape against the attacker to ensure a non-zero (tweakable) probability of public disclosure *each time* a fraudulent certificate is used against Tor Browser. This is done by randomly adding a subset of presented certificates to CT logs that the attacker may not control (inferred from the accompanied promises of public logging). Such *certificate cross-logging* distributes trust across all CT logs, raising the bar towards unnoticed certificate misissuance. Motivated by factors like privacy, security and deployability, Tor Browser uses Tor relays as intermediates to cache and interact with CT logs on its behalf. Such deferred auditing is a fundamental part of our setting unless future distributed auditing mechanisms turn out to be non-interactive from the browser's perspective.

The next incremental step is to not only cross-log certificates but also their promises of public logging. While it requires an additional CT log API endpoint, it facilitates auditing of these promises if some logs are trustworthy. The full design also holds logs accountable but without any such assumption: Tor relays challenge the logs to prove correct operation with regards to a single fixed view in Tor's consensus, and potential issues are reported to auditors that investigate them further.

1.2 Contribution and Structure

Section 2 introduces background on the theory and practise of CT, as well as the anonymity network Tor. Section 3 motivates the intended attacker and presents a unified threat model for CT and Tor. Section 4 describes the full CTor design that *eliminates all trust in the browser-recognized CT logs* by challenging them to prove certificate inclusion cryptographically, and would result in a *single probabilistically-verified view of the CT log ecosystem available from Tor's consensus*. This view could be used by other browsers as the basis of trust, *greatly improving the security posture of the entire web*. The security analysis in Section 5 shows that one of the best bets for the attacker would be to take network-wide actions against Tor to avoid public disclosure of certificate mis-issuance and log misbehavior. Such an attack is trivially detected, but it is hard to attribute unless reactive defenses are enabled at the cost of trade-offs.

The full design involves many different components that add deployment burdens, such as the requirement of reliable CT auditors that investigate suspected log misbehavior further. Therefore, we additionally propose two initial increments that place *some trust in CT logs* (Section 6). The first increment *provides evidence to independent CT logs that fraudulent certificates were presented while preserving privacy*. This greatly impacts risk-averse attackers because one part of their malicious behavior becomes transparent *if the randomly selected log operator is benign*. For example, the targeted domain name is disclosed as part of the cross-logged certificate, and awareness of the event draws unwanted attention.

The next increment is minor from the perspective of Tor, but requires CT logs to support an additional API. Similar changes were proposed in the context of CT gossip [23]. If supported, Tor relays could expose both the mis-issued certificates and the operators that promised to log them publicly *without the complexity of ever distinguishing between what is benign and fraudulent*. This API change happens to also build auditor infrastructure directly into CT log software, thereby paving the path towards the missing component of the full design. We argue that CTor can be deployed incrementally: complete Firefox's CT enforcement [4], add our cross-logging increments, and finally put the full design into operation. Each part of CTor would *greatly contribute to the open question of how to reduce and/or eliminate trust in browser-recognized log operators*, which is caused by the lack of an appropriate gossip mechanism as well as privacy issues while interacting with the logs [20, 23, 46].

We show that circuit-, bandwidth- and memory-*overheads are modest* by computing such estimates in Section 7. Therefore, we do not investigate performance further in any experimental setting. Section 8 discusses privacy aspects of our design choices with a focus on the essential role of the Tor network's distributed nature to preserve user privacy as well as the overall security. In gist, *a similar approach would be privacy-invasive without Tor*, e.g., if adopted by Google Chrome. Section 9 outlines related work. Section 10 concludes the paper.

2 Background

The theory and current practise of CT is introduced first, then Tor and its privacy-preserving Tor Browser.

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 [34]. By making the set of issued certificate chains[1] transparent, anyone that inspect the logs can detect certificate mis-issuance *after the fact*. It would be somewhat circular to solve issues in the CA ecosystem by adding trusted CT logs. Therefore, the cryptographic foundation of CT is engineered to avoid any such reliance. Google's *gradual* CT roll-out started in 2015, and evolved from downgrading user-interface indicators in Chrome to the current state of hard failures unless a certificate is accompanied by a signed *promise* that it will appear in two CT logs [55]. Unlike Apple's Safari [3], these two logs must additionally be operated by Google and not-Google to ensure independence [24].

The lack of mainstream verification, i.e., beyond checking signatures, allows an attacker to side-step the current CT enforcement with minimal risk of exposure *if the required logs are controlled by the attacker*. CTor integrates into the gradual CT roll-out by starting on the premise of pairwise-independently trusted CT logs, which avoids the risk of bad user experience [55] and significant system complexity. For example, web pages are unlikely to break, TLS handshake latency stays about

---

1 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 [18]. Such a *certificate chain* is valid if it ends in a trusted anchor that is shipped in the user's system software.

the same, and no robust management of suspected log misbehavior is needed. Retaining the latter property as part of our incremental designs simplifies deployment.

2.1.1 Cryptographic Foundation

The operator of a CT log maintains a tamper-evident append-only Merkle tree [36, 37]. 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 [17].

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 [8, 14, 46, 58].

2.1.2 Standardization and Verification

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 to obtain the most recent STH and to fetch proofs [36, 37]. 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 `add-chain` in Section 6, as well as several other endpoints in Section 4 to fetch proofs and STHs. It might be helpful to know that an inclusion proof is fetched based on two parameters: a certificate hash and the tree size of an STH. The former specifies the log entry of interest, and the latter with regards to which view inclusion should be proven. The returned proof is valid if it can be used in combination with the certificate to reconstruct the STH's tree head.

The CT landscape provides a limited value unless it is verified that the logs play by the rules. What the rules are changed over time, but they are largely influenced by the major browser vendors that define *CT policies*. For example, what is required to become a recognized CT log in terms of uptime and trust anchors, and which criteria should pass to consider a certificate CT compliant [3, 24]. 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. A party that follows-up on inclusion and consistency proofs is said to *audit* the logs.

Widespread 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, requesting an inclusion proof is privacy-invasive because it leaks browsing patterns to the logs, and reporting suspected log misbehavior comes with privacy [20] as well as operational challenges. Found log incidents are mostly reported manually to the CT policy list [11]. This is in contrast to automated *CT monitors*, which notify domain owners of newly issued certificates based on what actually appeared in the public logs [12, 38].

2.2 Tor

Most of the activity of Tor's millions of daily users starts with Tor Browser 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. Tor Browser 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 destinations, 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.

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 weights as well as flags such as `guard` or `exit`. It also assigns 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 [15]. 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 [60].

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, and it 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 [21]. 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 a strong attacker who is targeting all or a subset of users visiting a particular website over Tor. It is generally difficult to perform a targeted attack on a single particular Tor user because one needs to identify the user's connection before performing the attack—something that Tor's anonymity properties frustrate. However, it is not difficult to perform an attack on all or a subset of unknown users of a particular service. A net-

work vantage point to perform such an attack is easily obtained by operating an exit relay (for a subset of Tor users) or by compromising the network path of multiple exit relays or the final destination. Once so positioned, the encrypted network traffic can be intercepted using a fraudulent certificate and associated SCTs. The subsequent attack on decrypted network traffic may be passive (to gather user credentials or other information) or active. Typical examples of active attacks are to change cryptocurrency addresses to redirect funds to the attacker or to serve an exploit to the user's browser for *user deanonymization*. Without the ability to intercept encrypted traffic, these attacks become more difficult as the web moves towards deprecating plaintext HTTP.

All of the components of such an attack have been seen in-the-wild numerous times. Untargeted attacks on visitors of a particular website include Syria's interception of Facebook traffic using a self-signed 512-bit RSA key in 2011 [19], Iran's interception of Bing and Google traffic using the DigiNotar CA [34, 49], and the 2018 MyEtherWallet self-signed certificate that was used as part of a BGP hijack [51]. The latter is also an example of redirecting routing as part of an attack (either suspected or confirmed). Other examples of this are Iran hijacking prefixes of Telegram (an encrypted messaging application) in 2018 [47], another attack on cryptocurrency in 2014 this time targeting unencrypted mining traffic [57], and hijacks that may have been intelligence-gathering (or honest mistakes) including hijacks by Russian ISPs in 2017 and China Telecom in 2018 and 2019 [66]. Finally, there are several examples of law enforcement serving exploits to Tor Browser users to deanonymize and subsequently arrest individuals [28, 65].

With the attacker's profile in mind, we consider someone that controls a CA, enough CT logs to pass Tor Browser's SCT-centric CT policy, some Tor clients, and a fraction of Tor relays. For example, it is possible to issue 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 [32, 41]. 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 Tor Browser as a starting point [16, 48]. Any attacker that can reliably disrupt CT and/or Tor well beyond Tor's threat model is therefore not within ours.

Given that we are in the business of enforcing CT, the attacker needs to hide mis-issued certificates and

SCTs from entities that audit the CT log ecosystem. 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. Note that the goal of *detection* is inherited from CT's threat model, which aims to remedy certificate mis-issuance *after the fact*; not prevent it [34].

We identify and analyze specific attack vectors that follow from our threat model and design as part of the security analysis in Section 5, namely, attack vectors related to timing as well as relay flooding and tagging.

4 Design

A complete design—a design that detects misbehavior by both CAs and CT logs within our strong threat model—requires a considerable degree of complexity. In this section we present such a full design by breaking it up into four phases as shown in Figure 1, demonstrating the need for the involved complexity in each step. Section 6 presents two incremental versions of the full design that are less complicated. The first increment comes as the cost of having a weaker threat model and security goal. The second increment does not have a weaker security goal but requires a new CT log API.

A design that starts by validating SCT signatures like Apple's Safari is promising and assumed [3, 67], but it does not stand up against a malicious CA and two CT logs that work in concert. If the logs cannot be trusted blindly, the presented SCTs need to be audited.

4.1 Phase 1: Submission

The least complicated auditing design would be one where Tor Browser receives a TLS certificate and accompanying SCTs (we will refer to this bundle as an SCT Feedback Object, or SFO for short) and talks to the corresponding logs, over Tor, requesting an inclusion proof for each SCT. In an ordinary browser, this would be an unacceptable privacy leak to the log of browsing behavior associated with an IP address; performing this request over Tor hides the user's IP address but still leaks real-time browsing behavior.

An immediate problem with this design is that a primary requirement of Tor Browser is to persist no data about browsing behavior after the application exits. If we assume that browsers are not left running for long periods of time, the inclusion proof request can be easily circumvented by the attacker by using a fresh SCT whose MMD has not completed—thus no inclusion proof needs to be provided (yet) by the log as per the CT standard. A second problem is that the STH that an inclusion proof refers to exists in a *trust vacuum*: there is no way to know that it is consistent with other STHs and not part of a split view (assuming that there is no proactive STH gossip [14, 58], which is not deployed).

We can evolve the design by adding two components: a list of STHs that Tor Browser receives over a trusted channel and the participation of a trusted third party with the ability to persist data and perform auditing actions at a later point in time.

A single third party used by all users of Tor Browser would receive a considerable aggregation of browsing behavior and would need to scale in-line with the entire Tor network. A small number of auditors presents privacy and single-point-of-failure concerns. A large number would be ideal but presents difficulties in curation and independent management and still requires scaling independent of the Tor network. These concerns do not entirely preclude the design, but they can be easily avoided by reusing relays in the Tor network as our trusted third parties: we call the relays so designated Certificate Transparency Relays (CTRs).

Now, when the browser is completing the TLS handshake, it simultaneously either passes the SFO to a CTR (if the MMD of the SCT has not elapsed) or queries the log itself for an inclusion proof to a trusted STH. However, if we presume the attacker can serve an exploit to the browser, the latter behavior is immediately vulnerable. The log, upon receiving an inclusion proof request for an SCT that it knows is malicious, can delay its response. The TLS connection in the browser, having succeeded, will progress to the HTTP request and response, at which point the exploit will be served, and the SFO (containing the cryptographic evidence of CA and log misbehavior) will be deleted by the exploit code. While blocking the TLS connection until the CT log responds is an option, experience related to OCSP hard-fail indicates that this notion is likely doomed to fail [33].

The final change of the design has Tor Browser submit the SFO to the CTR immediately upon receipt
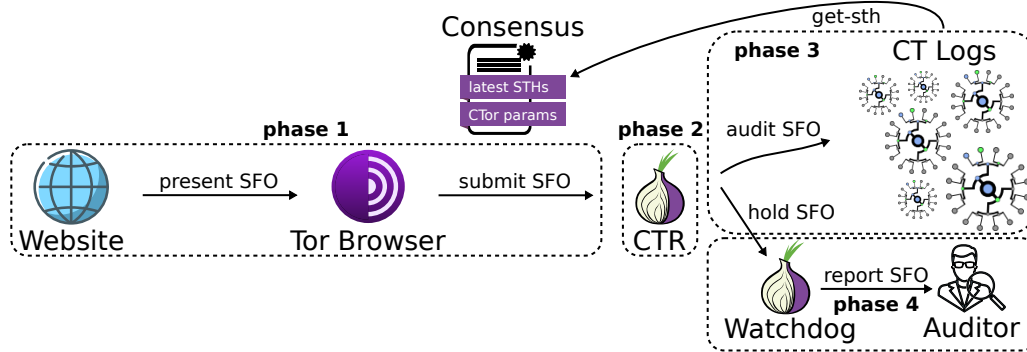
Fig. 1. An overview of the four phases of the full CTor design. In phase 1 Tor Browser submits an SFO (SCT Feedback Object) to a Certificate Transparency Relay (CTR), followed by phase 2 where the CTR buffers the SFO. In phase 3 the relay attempts to audit the SFO, and in case of failure, it reports the SFO to an auditor with the help of a watchdog CTR in phase 4.

(with some probability) in all cases. A consequence of this shift is that the trusted STH list no longer needs to be delivered to the browser but rather the CTRs. To mitigate the risk of a browser exploit being able to identify the CTR to the attacker (who could then target it), we prepare *CTR circuits* ahead of time that are closed and discarded as soon as the SFO is sent. This allows the SFO submission to race with the TLS connection completion and HTTP request/response. An added detail is to block the TLS connection in the case that an SFO is unusually large, as defined by a parameter `ct-large-sfo-size`. A large SFO may indicate an attempt to win the race between SFO submission and exploitation. The parameter can be set such that it happens extremely rarely on legitimate connections, as shown in Section 7.

We summarize phase 1 with the following algorithm that provides more explicit steps and details, including the addition of a parameter `ct-submit-pr` that indicates a probability that an SFO is submitted to a CTR. This provides probabilistic security while providing the ability to adjust submission rates to account for CTR and more general network scaling/health issues. Given an incoming SFO s, Tor Browser should:

1. Raise a certificate error and stop if the certificate chain of s is not rooted in Tor Browser's trust store.
2. Raise a certificate transparency error and stop if the SCTs of s fail Tor Browser's CT policy.
3. If $\mathsf{len}(s) < $ `ct-large-sfo-size`, accept s and conduct the remaining steps in the background while the TLS connection and subsequent HTTP request/response proceed. If $\mathsf{len}(s) \geq$ `ct-large-sfo-size` pause the TLS handshake, complete the remaining steps, accept s as valid and then continue the handshake.
4. Flip a biased coin based on `ct-submit-pr` and stop if the outcome indicates no further auditing.

5. Submit s to a random CTR on a pre-built circuit. The circuit used for submission is closed immediately without waiting for any acknowledgment.

4.2 Phase 2: Buffering

Once received, the most straightforward thing for a CTR to do would be to contact the issuing log and request an inclusion proof relative to a trusted STH. (And if the SCT's MMD has not elapsed, hold the SFO until it has.) However, this proposal has two flaws, the first of which leads us to the actual design of phase 2.

Immediately contacting the log about an SFO (i) allows the log to predict when exactly it will receive a request about an SFO and (ii) discloses real-time browsing behavior to the log. The former problem means that an attacker can position resources for perpetuating an attack ahead-of-time, as well as letting it know with certainty whether a connection was audited (based on `ct-submit-pr`). The latter is some amount of information leakage that can help with real-time traffic analysis.

Because a CTR must support buffering SCTs regardless (due to the MMD), we can schedule an event in the future for when each SFO should be audited. Adding a per-SFO value sampled from `ct-delay-dist` effectively adds stop-and-go mixing [30] 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 audited at different times and that SFOs submitted to that CTR by other honest clients are more likely to be mixed with these.

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

Fig. 2. Algorithm that computes an `audit_after` timestamp t.

In addition to buffering SFOs for mixing effects, we also add a layer of caching to reduce the storage overhead, prevent unnecessary log connections, and limit the disclosure to logs. With regards to some CT circuit, an incoming SFO s is processed as follows by a CTR:

1. Close the circuit to enforce one-time use.
2. Discard all SCTs in the SFO for logs the CTR is not aware of; if no SCT remains then discard the SFO.
3. Stop if s is cached or already pending to be audited in the buffer. See caching details in Section 7.2.
4. Sample a CT log l that issued a remaining SCT in s.
5. Compute an `audit_after` time t, see Figure 2.
6. Add (l, t, s) to a buffer of pending SFOs to audit.

What makes a CT log known to the CTR is part of the Tor consensus, see Section 4.5. It implies knowledge of a trusted STH for the sampled CT log l, which refers to an entity that (i) issued an SCT in the submitted SFO, and (ii) will be challenged to prove inclusion in phase 3 sometime after the `audit_after` timestamp t elapsed. We choose one SCT (and thus log) at random from the SFO because it is sufficient to suspect only one misbehaving log so long as we report the entire SFO, allowing us to identify the other malicious CT logs later on (a risk averse-attacker would not conduct an attack without controlling enough logs, i.e., one benign log would otherwise make the mis-issued certificate public).

The `audit_after` timestamp specifies the earliest point in time that an SCT from an SFO will be audited in phase 3, which adds random noise that obfuscates real-time browsing patterns in the Tor network and complicates predictions of when it is safe to assume no audit will take place. If memory becomes a scarce resource, pending triplets should be deleted at random [46]. Figure 2 shows that t takes the log's MMD into account. This prevents an *early signal* to the issuing CT logs that an SFO is being audited. For example, if an SFO is audited before the MMD elapsed, then the issuing CT log could simply merge the underlying certificate chain to avoid any MMD violation. However, by taking the MMD into account, this results in a relatively large time window during which the attacker can attempt to *flood* all CTRs in hope that they delete the omitted SFO at random before it is audited. We discuss the threat of flooding further in Section 5, noting that such an

attack can be detected if CTRs publish two new metrics in the extra-info document: `ct-receive-bytes` and `ct-delete-bytes`. 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` and `write-history`.

4.3 Phase 3: Auditing

As alluded to in phase 2, there is a second problem why the simple behavior of "contact the log and request an inclusion proof" is unacceptable. We include the ability to DoS an individual Tor relay in our threat model—if the log knows which CTR holds the evidence of its misbehavior, it can take the CTR offline, wiping the evidence of the log's misbehavior from its memory.

We can address this concern in a few ways. The simple proposal of contacting the log over a Tor circuit will not suffice: a log can tag each CTR by submitting unique SFOs to them all, and recognize the CTR when they are submitted (see Section 5). Even using a unique Tor circuit for each SFO might not suffice to prevent effective tagging attacks. For example, after tagging all CTRs, a malicious log could ignore all but innocuous untagged requests and tagged requests matching tags for whichever CTR it decides to respond to first. If some kind of back-off is supported (common to delay retransmissions and avoid congestion), the rest of the CTRs will likely be in back-off so that there is a high probability that the first CTR is the one fetching proofs. The log can repeat this process—alternating tagged CTRs it replies to—until it receives the offending SFO from an identifiable CTR with high probability. CTRs may report the log as inaccessible for days, but that is not the same as direct cryptographic evidence of misbehavior.

While there are ways to detect this attack after-the-fact, and there may be ways to mitigate it, a more robust design would tolerate the disclosure of a CTRs identity to the log during the auditing phase without significant security implications. A simple appealing approach is to write the data to disk prior to contacting the log; however, Tor relays are explicitly designed not to write data about user behavior to disk unless debug-level logging is enabled. Relay operators have expressed an explicit desire to never have any user data persisted to disk, as it changes the risk profile of their servers with regards to search, seizure, and forensic analysis.

The final design is to have the CTR work with a partner CTR—we call it a *watchdog*—that they choose at random and contact over a circuit. Prior to attempt-

ing to fetch a proof from a log, the CTR provides the watchdog with the SFO it is about to audit. After an appropriate response from the log, the CTR tells the watchdog that the SFO has been adequately addressed.

In more detail, each CTR maintains a single shared circuit that is used to interact with all CT logs known to the CTR (we are not using one circuit per SFO given the overhead and unclear security benefit noted above). For *each* such log l, the CTR runs the following steps:

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 $l' = l$ and $t <=$ now():
   (a) Share s with the current watchdog.
   (b) Challenge the log to prove inclusion to the closest STH in the Tor consensus where $t \leq$ STH.timestamp. Wait ct-log-timeout time units for the complete proof before timing out.
     – On valid proof: send an acknowledgment to the watchdog, cache s and then discard it.
     – On any other outcome: close circuit to the watchdog CTR, discard s, and go to step 1.

4.4 Phase 4: Reporting

At any given time, a CTR may be requesting inclusion proofs from logs and act as a watchdog for one or more CTRs. A CTR acting as a watchdog will have at most one SFO held temporarily for each other CTR it is interacting with. If an acknowledgement from the other CTR is not received within ct-watchdog-timeout, it becomes the watchdog's responsibility to report the SFO such that it culminates in human review if need be.

Because human review and publication is critical at this end-stage, we envision that the watchdog (which is a Tor relay that cannot persist any evidence to disk and may not be closely monitored by its operator) provides the SFO to an independent CT auditor that is run by someone that closely monitors its operation. When arriving at the design of the CTR being a role played by a Tor relay, we eschewed separate auditors because of the lack of automatic scaling with the Tor network, the considerable aggregation of browsing behavior across the Tor network, and the difficulties of curation and validation of trustworthy individuals. SFOs submitted to auditors at this stage have been filtered through the CTR layer (that additionally backs-off if the logs become unavailable to prevent an open pipe of SFOs from being reported), resulting in an exponentially smaller

load and data exposure for auditors. This should allow for a smaller number of them to operate without needing to scale with the network.

While we assume that most auditors are trusted to actually investigate the reported SFOs further, the watchdog needs to take precautions talking to them because the network is not trusted.[2] The watchdog can contact the auditor immediately, but must do so over an independent Tor circuit.[3] If a successful acknowledgement from the auditor is not received within ct-auditor-timeout, the SFO is buffered for a random time using ct-delay-dist before being reported to the same auditor again over a new independent Tor circuit.

When an auditor receives an SFO, it should persist it to durable storage until it can be successfully resolved to a specific STH.[4] Once so persisted, the auditor can begin querying the log itself asking for an inclusion proof. If no valid inclusion proof can be provided after some threshold of time, the auditor software should raise the details to a human operator for investigation.

Separately, the auditor should be retrieving the current Tor consensus and ensuring that a consistency proof can be provided between STHs from the older consensus and the newer. If consistency cannot be established after some threshold of time, the auditor software should raise the details to a human operator for investigation. An auditor could also monitor a log's uptime and report on excessive downtime. Finally, it is paramount that the auditor continuously monitors its own availability from fresh Tor-circuits by submitting known SFOs to itself to ensure that an attacker is not keeping watchdogs from connecting to it.

4.5 Setup

There are a number of additional details missing to setup phases 1–4 for the design. Most of these details relate to the Tor consensus. Directory authorities in-

---

2 While our threat model, and Tor's, precludes a global network adversary, both include partial control of the network.

3 This is also important because CTRs are not necessarily exits, i.e., the exiting traffic must be destined to another Tor relay.

4 The fetched inclusion proof must be against the first known STH that should have incorporated the certificate in question by using the history of STHs in Tor's consensus: the mis-issued certificate might have been merged into the log reactively upon learning that a CTR reported the SFO, such that a valid inclusion proof can be returned with regards to a more recent STH but not earlier ones that actually captured the log's misbehavior.

fluence the way in which Tor Browser and CTRs behave by voting on necessary parameters, such as the probability of submission of an SFO (`ct-submit-pr`) and the timeout used by CTRs when auditing CT logs (`ct-log-timeout`), as introduced earlier as part of the design. See Appendix A for details on these parameters and their values that were previously used. Next, we briefly introduce a number of implicitly used parts from our design that should also be part of the consensus.

In the consensus, the existing `known-flags` item determines the different flags that the consensus might contain for relays. We add another flag named `CTR`, which indicates that a Tor relay should support CT-auditing as described here. 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 8 discusses trade-offs in the assignment of the `CTR` flag.

The consensus should also capture a fixed view of the CT log ecosystem by publishing STHs from all known logs. A CT log is known 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 Tor Browser accepts, the CT logs recognized by Tor Browser 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.

5 Security Analysis

We consider four types of impact for an attacker that conducted HTTPS-based man-in-the-middle attacks on Tor Browser. Other than *none*, these impact types are:

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 design leads to significant and catastrophic impact events, but does unfortunately not preclude minor ones. It is possible to overcome this shortcoming at different trade-offs, e.g., by tuning CTor parameters reactively (phase 2 below) or relying on different trust assumptions as in the incremental cross-logging designs (Section 6).

Probability of Detection. Suppose the attacker mis-issued a certificate that Tor Browser trusts, and that it is considered valid because it is accompanied by enough SCTs from CT logs that the attacker controls. The resulting SFO is then used to man-in-the-middle a single Tor Browser user, i.e., for the purpose of our analysis we consider *the most risk-averse scenario possible*. Clearly, none of the attacker's CT logs plan to keep any promise of public logging: that would trivially imply significant impact events. The risk of exposure is instead bound by the probability that *any* of the four phases in our design fail to propagate the mis-issued SFO to a pinned CT auditor that is benign.

Phase 1: Submission. The probability of detection cannot exceed the probability of submission (`ct-submit-pr`). We analyze the outcome of submitting the mis-issued SFO from Tor Browser to a CTR. There are two cases to consider, namely, the mis-issued SFO is either larger than `ct-large-sfo-size` or it is not.

If the SFO is larger than `ct-large-sfo-size`, Tor Browser blocks until the SFO is submitted and its CT circuit is closed. As such, it is impossible to serve a Tor Browser 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 Tor Browser are unreliable,[5] the sampled CTR identity also cannot be revealed with high certainty afterwards by compromising Tor Browser. The attacker may know that the SFO is buffered by *some CTR* based on timing, i.e., blocking-behavior could be 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-large-sfo-size` there is a race between (i) the time it takes for Tor Browser 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,

---

5 "tor" (aka "little-t tor") is the tor process Tor Browser 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.

the attacker would maximize the time it takes to perform (i) by sending an SFO that is `ct-large-sfo-size`. 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).

Note that the attack surface could, in theory, be eliminated by setting `ct-large-sfo-size` to zero. However, that is likely too costly in terms of latency [33].

Phase 2: Buffering. The probability of detection cannot exceed $1 - (f_{ctr} + f_{dos})$, where f_{ctr} is the fraction of malicious CTRs and f_{dos} the fraction of CTRs that suffer from DoS. We analyze the outcome of SFO reception at a genuine CTR.

The time that an SFO is buffered depends on if the log's MMD elapsed or not. The earliest point in time that a newly issued SCT can be audited (and the log is expected to respond) is an MMD later, whereas the normal buffer time is otherwise only governed by smaller randomness in the `audit_after` timestamp (minutes). A rational attacker would therefore maximize the buffer time by using a newly issued SCT, resulting in an attack window that is *at least* 24 hours for today's CT logs [24].

Following from Tor's threat model, the mis-issued SFO must be stored in volatile memory and not to disk. Two risks emerge due to large buffer times: the CTR in question might be restarted by the operator independently of the attacker's mis-issued SFO being buffered, and given enough time the attacker might find a way to cause the evidence to be deleted. While a risk-averse attacker cannot rely on the former to avoid detection, we emphasize that the CTR criteria must include the `stable` flag to reduce the probability of this occurring.

The latter is more difficult to evaluate. It depends on the attacker's knowledge as well as capabilities. Phase 1 ensured that the attacker *does not know which CTR to target*. As such, any attempt to intervene needs to target all CTRs. While a network-wide DoS against Tor would be effective, it is not within our threat model. A less intrusive type of DoS would be to *flood* CTRs by submitting massive amounts of SFOs: just enough to make memory a scarce resource, but without making Tor unavailable. This could potentially *flush* a target SFO from the CTR's finite memory, following from the delete-at-random strategy in Section 4.2. Assuming that a CTR has at most 1 GiB of memory available for SFOs (conservative and in favour of the attacker), Appendix C shows that the attacker's flood must involve at least 2.3 GiB per CTR to accomplish a 90% success certainty. This means that it takes 7.9–39.3 minutes if the relay bandwidth is between 8–40 Mbps. So it is im-

practical to flush all CTRs within a few minutes, and hours are needed not to make everyone unavailable at once.

The CTR criteria set in Section 4.5 matches over 4000 Tor relays [62]. A network-wide flush that succeeds with 90% certainty therefore involves 8.99 TiB. It might sound daunting at first, but distributed throughout an entire day it only requires 0.91 Gbps. Such an attack is within our threat model because it does not make Tor unavailable. Notably the ballpark of these numbers do not change to any significant degree by assuming larger success probabilities, e.g., a 99% probability only doubles the overhead. Further, the needed bandwidth scales linearly with the assumed memory of CTRs. This makes it difficult to rely on the finite volatile memory of CTRs to mitigate network-wide flushes. As described in Section 4.2, we ensure that flushes are *detected* by publishing the number of received and deleted SFO bytes throughout different time intervals as extra-info.

Once detected, there are several possible *reactions* that decrease the likelihood of a minor impact scenario. For example, Tor's directory authorities could lower MMDs to, say, 30 minutes, so that the SFO is reported to an auditor before it is flushed with high probability. This has the benefit of implying significant impact because the mis-issued certificate is detected, but also the drawback of allowing the logs to merge the certificate before there is any MMD violation to speak of. The most appropriate response depends on the exact attack scenario and which trade-offs one is willing to accept.

Phase 3: Auditing. By the time an SFO enters the audit phase, the log in question is expected to respond with a valid inclusion proof. There is no such proof if the log violated its MMD, and it is too late to create a split-view that merged the certificate in time because the CTR's view is already fixed by an STH in the Tor consensus that captured the log's misbehavior. In fact, creating any split-view within Tor is impractical because it requires that the consensus is forged or that nobody ever checks whether the trusted STHs are consistent. This leaves two options: the attacker either responds to the query with an invalid inclusion proof or not at all. The former is immediately detected and starts phase 4, whereas the latter forces the CTR to wait for `ct-watchdog-timeout` to trigger (which is a few seconds to avoid premature auditor reports). A rational attacker prefers the second option to gain time.

Clearly, the attacker knows that *some* CTR holds evidence of log misbehavior as it is being audited. The relevant question is whether the *exact CTR identity* can be inferred, in which case the attacker could knock it

offline (DoS). Motivated by the threat of *tagging*, where the attacker sends unique SFOs to all CTRs so that their identities are revealed once queried for, we erred on the safe side and built watchdogs into our design: it is already too late to DoS the querying CTR because the evidence is already replicated somewhere else, ready to be reported unless there is a timely acknowledgement. The attacker would have to *break into an arbitrary CTR within seconds* to cancel the watchdog, which cannot be identified later on (same premise as the sampled CTR in phase 1). Such an attacker is not in Tor's threat model.

Phase 4: Reporting. At this stage the process of reporting the mis-issued SFO to a random CT auditor is initiated. Clearly, the probability of detection cannot exceed $1 - f_{\text{auditor}}$, where f_{auditor} is the fraction of malicious CT auditors. Fixating the sampled CT auditor is important to avoid the threat of an eventually successful report only if it is destined to the attacker's auditor because our attacker is partially present in the network. Gaining time at this stage is of limited help because the CTR identity is unknown as noted above, and it remains the case throughout phase 4 due to reporting on independent Tor circuits (and independently of if other SFO reports succeeded or not). Without an identifiable watchdog, the attacker needs a network-wide attack that is already more likely to succeed in the buffer phase.

6 Incremental Deployment

Section 4 covered the full design that places zero-trust in the CT landscape by challenging the logs to prove certificate inclusion with regards to trusted STHs in the Tor consensus. If no such proof can be provided, the suspected evidence of log misbehavior is reported to a trusted CT auditor that follows-up on the incident, which involves human intervention if an issue persists. The proposed design modifies the Tor consensus, Tor relays, and Tor Browser. It also requires development and operation of a trusted auditor infrastructure. The current lack of the latter makes it unlikely that we will see adoption of CTor in its full potential anytime soon, and begs the question of increments that help us get there in the future. Therefore, we additionally propose two incremental designs in this section.

Without the ability to rely on CT auditors, trust needs to be shifted elsewhere because we cannot expect relay operators to take on the role. At the same time, an incremental proposal needs to improve upon the status quo of pairwise-independently trusted CT logs. These observations lead us towards the trust assumption that *at least some* of the CT logs are trustworthy. Such an assumption is suboptimal, but it does provide a real-world security improvement by significantly raising the bar from weakest-link(s) to quite the opposite.

The smallest change of the full design would be for watchdogs to report suspected certificate mis-issuance to all CT logs, simply by using the public add-chain API to make the SFO's certificate chain transparent. This has the benefit of holding the CA accountable if *some* log operator is benign. Given that our attacker is risk-averse, reporting to a single independent log[6] that issued none of the accompanied SCTs would likely be sufficient. There is also room for further simplification: there is no point in challenging the logs to prove inclusion if the fallback behavior of no response only makes the issued certificate public, not the associated SCTs. Thus, CTRs could opt to cross-log immediately *without ever distinguishing between certificates that are benign and possibly fraudulent*. This results in the incremental design shown in Figure 3, which initially removes several system complexities such as extra-info metrics, auditor infrastructure, watchdog collaborations, and inclusion proof fetching against trusted STHs in Tor's consensus.

The drawback of certificate cross-logging is that the misbehaving CT logs cannot be exposed. There is also a discrepancy between cross-logging and encouraging the CT landscape to deploy reliable CT auditors. We therefore suggest a minimal change to the basic cross-logging design that addresses both of these concerns. This change is unfortunately to the API of CT logs and not Tor. The proposed change is to allow cross-logging of a certificate's issued SCTs, e.g., in the form of an add-sfo API that would replace add-chain in Figure 3. This means that CTRs could expose both the mis-issued certificate and the logs that violated their promises of public logging. At the same time, the infrastructural part of a CT auditor is built directly into existing CT logs: accepting SFOs that need further investigation. Such an API would be an ecosystem improvement in itself, providing a well-defined place to report suspected log misbehavior on-the-fly *casually*, i.e., without first trying to resolve an SFO for an extended time period from many different vantage points and then ultimately reporting it manually on the CT policy mailing list.

---

6 The independent log need not be trusted by the browser, i.e., it could be specified separately in the Tor consensus. An operator that runs such a log would help distribute trust and facilitate auditing. Appendix B provides details on today's log ecosystem.
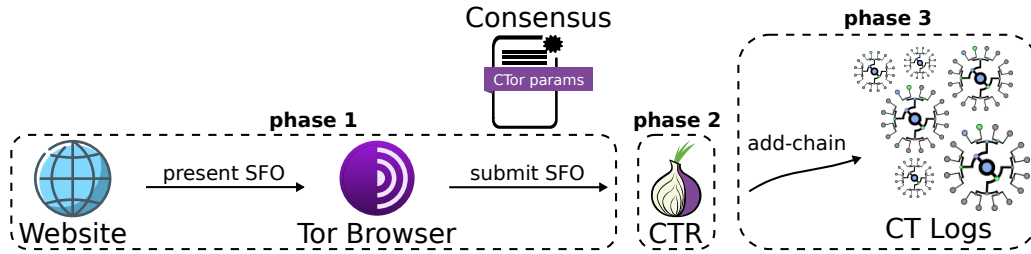
Fig. 3. Incremental design that can be deployed without any trusted CT auditors. Tor Browser still submits SFOs to CTRs on independent Tor circuits for the sake of privacy and security. After CTR buffering, the submitted certificates are *cross-logged* by adding them to independent CT logs (selected at random) that the attacker does not control (inferred from accompanied SCTs).

Security Sketch. There are no changes to phase 1 because cross-logging is instantiated at CTRs. Phases 3–4 are now merged, such that the encountered certificates are added to independent CT logs that the attacker does/may not control. Watchdogs are no longer needed since either the certificates are added to a log that the attacker controls, or they are not (which makes them public). The other main difference takes place in phase 2, during which CTRs buffer SFOs. The buffer time used to be lengthy due to taking early signals and MMDs into account, but it is now irrelevant as no inclusion proofs are fetched. The expected buffer time can therefore be shortened down to *minutes* that follow only from the randomness in the `audit_after` timestamp (for the sake of privacy), making network-wide flushes impractical while at the same time reducing the time that a mis-issued certificate stays unnoticed: a benign log is likely to add an entry before all MMDs elapsed.

The extended cross-logging also aims to expose log misbehavior. As such, it is paramount that no cross-logged SFO becomes public before the issuing CT logs can merge the mis-issued certificate reactively to avoid catastrophic impact. This could be assured by buffering newly issued SFOs longer as in the full design, which brings back the threat and complexity of minor impact scenarios. Another option that is appealing for Tor (but less so for CT) is to operate the `add-sfo` API with the expectation of *delayed merges* that account for MMDs before making an SFO public, effectively moving lengthy buffering from CTRs to CT logs with persistent storage. Trillian-based CT logs already support delayed merges of (pre)certificates, see `sequencer_guard_window` [25].

7 Performance

The following analysis shows that CTor's overhead is modest based on computing performance estimates from concrete parameter properties and two public data sets.

7.1 Setup

Mani *et al.* derived a distribution of website visits over Tor and an estimation of the number of circuits through the network [40]. We use their results to reason about overhead as the Tor network is under heavy load, assuming 140 million daily website visits (the upper bound of a 95% confidence interval). Our analysis also requires a distribution that captures typical SFO properties per website visit. Therefore, we collected an SFO data set by browsing the most popular webpages submitted to Reddit (r/frontpage, all time) on December 4, 2019. The data set contains SFOs from 8858 webpage visits, and it is available online as an open access artifact together with the associated scripts [13]. Notably we hypothesized that browsing actual webpages as opposed to front-pages would yield more SFOs. When compared to Alexa's list it turned out to be the case: our data set has roughly two additional SFOs per data point. This makes it less likely that our analysis is an underestimate.

We found that an average certificate chain is 5440 bytes, and it is seldom accompanied by more than a few SCTs. As such, a typical SFO is in the order of 6 KiB. No certificate chain exceeded 20 KiB, and the average number of SFOs per webpage was seven. The latter includes 1–2 SFOs per data point that followed from our client software calling home on start-up (Chromium 77).

We assume no abnormal CTor behavior, which means that there will be little or no CTR back-offs due to the high uptime requirements of today's CT logs: 99%. We set `ct-large-sfo-size` conservatively to avoid blocking in the TLS handshake (e.g., 20 KiB), and use a 10% submission probability as well as a 10 minute random buffer delay on average. It is likely unwarranted to use a higher submission probability given that the intended attacker is risk-averse. Shorter buffer times would leak finer-grained browsing patterns to the logs, while longer ones increase the attack surface in phase 2. Therefore, we selected an average for `ct-delay-dist` that satisfies none of the two extremes. The remaining

CTor parameters are timeouts, which have little or no performance impact if set conservatively (few seconds).

7.2 Estimates

The incremental cross-logging designs are analyzed first without any caching. Caching is then considered, followed by overhead that appears only in the full design.

Circuit Overhead. Equation 1 shows the expected circuit overhead from Tor Browser over time, where p is the submit probability and $\bar{d}$ the average number of SFOs per website visit. The involved overhead is linear as either of the two parameters are tuned up or down.

$$p\bar{d} \tag{1}$$

Using $p \leftarrow \frac{1}{10}$ and our approximated SFO distribution $\bar{d} \leftarrow 7$ yields an average circuit overhead of 0.70, i.e., for every three Tor Browser circuits CTor adds another two. Such an increase might sound daunting at first,[7] but these additional circuits are short-lived and lightweight; transporting 6 KiB on average. Each CTR also maintains a long-lived circuit for CT log interactions.

Bandwidth Overhead. Equation 2 shows the expected bandwidth overhead for the Tor network over time, where V is the number of website visits per time unit, p the submit probability, $\bar{d}$ the average number of SFOs per website visit, and $\bar{s}$ the average SFO byte-size.

$$6V p\bar{d}\bar{s} \tag{2}$$

$V p\bar{d}$ is the average number of SFO submissions per time unit, which can be converted to bandwidth by weighting each submission with the size of a typical SFO and accounting for it being relayed six times: three hops from Tor Browser to a CTR, then another three hops from the CTR to a CT log (we assumed symmetric Tor relay bandwidth). Using $V \leftarrow 140$ M/day, $p \leftarrow \frac{1}{10}$, $\bar{d} \leftarrow 7$, $\bar{s} \leftarrow 6$ KiB and converting the result to bps yields 334.5 Mbps in total. Such order of overhead is small when compared to Tor's capacity: 450 Gbps [61].

Memory Overhead. Equation 3 shows the expected buffering overhead, where V_m is the number of website visits per minute, t the average buffer time in

minutes, R the number of Tor relays that qualify as CTRs, and $\bar{s}$ the typical SFO size in bytes.

$$\frac{V_m t}{R}\bar{s} \tag{3}$$

$V_m t$ represent incoming SFO submissions during the average buffer time, which are randomly distributed across R CTRs. Combined, this yields the expected number of SFOs that await at a single CTR in phase 2, and by taking the byte-size of these SFOs into account we get an estimate of the resulting memory overhead. Using $V_m \leftarrow \frac{140 \text{ M}}{24 \cdot 60}$, $t \leftarrow 10$ m, $R \leftarrow 4000$ based on the CTR criteria in Section 4.5, and $\bar{s} \leftarrow 6$ KiB yields 1.42 MiB. Such order of overhead is small when compared to the recommended relay configuration: at least 512 MiB [64].

A cache of processed SFOs reduces the CTR's buffering memory and log interactions proportionally to the cache hit ratio. Mani *et al.* showed that if the overrepresented torproject.org is removed, about one third of all website visits over Tor can be attributed to Alexa's top-1k and another one third to the top-1M [40]. Assuming 32 byte cryptographic hashes and seven SFOs per website visit, a cache hit ratio of $\frac{1}{3}$ could be achieved by a 256 KiB LFU/LRU cache that eventually captures Alexa's top-1k. Given that the cache requires memory as well, this is mainly a bandwidth optimization.

Full Design. For each CTR and CT log pair, there is an additional watchdog circuit that transports the full SFO upfront before fetching an inclusion proof. The expected bandwidth overhead is at most $9V p\bar{d}\bar{s}$, i.e., now also accounting for the three additional hops that an SFO is subject to. In practise the overhead is slightly less, because an inclusion query and its returned proof is smaller than an SFO. We expect little or no watchdog-to-auditor overhead if the logs are available, and otherwise one light-weight circuit that reports a single SFO for each CTR that goes into back-off. Such overhead is small when compared to all Tor Browser submissions. Finally, the required memory increases because newly issued SFOs are buffered for at least an MMD. Only a small portion of SFOs are newly issued, however: the short-lived certificates of Let's Encrypt are valid for 90 days [1], which is in contrast to 24 hour MMDs [24].

8 Privacy

There is an inherent privacy problem in the setting due to how CT is designed and deployed. A browser, like Tor Browser, that wishes to validate that SFOs presented to

---

[7] Circuit establishment involves queueing of onionskins [63] and it is a likely bottleneck, but since the introduction of ntor it is not a scarce resource so such overhead is acceptable if it (i) serves a purpose, and (ii) can be tuned. Confirmed by Tor developers.

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 (PIR) [7] that require server-side support or introduction of additional parties and trust assumptions [29, 39], 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 Tor Browser, 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 IP-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 Tor Browser user browsing with CT log interaction harder for attackers—and safely maintains a dynamic cache of the most commonly already verified SFOs. While browsers like Chrome could maintain a cache, Tor Browser's security and privacy goals (Section 2.2) prohibit such shared (persisted) dynamic state.

In terms of privacy, the main limitation of CTor is that CTor continuously leaks to CT logs—and to a *lesser extent* auditors (depending on design)—a fraction of certificates of websites visited using Tor Browser 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 Tor Browser or DNS resolution at exit-relays [27], both of which indicate visit activity in real time to a small number of entities.

Another significant 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 full design, see Section 4.5—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 7). Depending on the health of the network and the exact incremental deployment of CTor, there are different trade-offs.

9 Related Work

The status quo is to consider a certificate CT compliant if it is accompanied by two independent SCTs [24, 67]. Therefore we proposed that Tor Browser should do the same, but unlike any other CT-enforcing web browser CTor also provides concrete next steps that relax the centralized trust which is otherwise misplaced in CT logs [41, 50, 52, 53]. Several proposals surfaced that aim to do better with regards to 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 [35]. 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 [37]. Lueks and Goldberg proposed that a separate database of inclusion proofs could be maintained that supports information-theoretic PIR [39]. 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 [29]. Nordberg *et al.* avoid inclusion proof fetching by hanging on to presented SFOs, handing them back to the same origin at a later time [46]. In contrast, CTor protects the user's privacy without any persistent browser state by submitting SFOs on independent Tor circuits to CTRs, which in turn add random noise before there is any log interaction. 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 [6], or, browser vendors could push a verified view [54]. Such trusted auditor relationships may work for some but not others [46]. Chuat *et al.* proposed that HTTPS clients and HTTPS servers could pool STHs and consistency proofs, which are gossiped on website visits [8]. Nordberg *et al.* suggested a similar variant, reducing the risk of user tracking by pooling fewer and recent STHs [46]. 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 [14]. Syta *et al.* proposed an alternative to reactive gossip mechanisms by showing how an STH can be cosigned efficiently by many independent witnesses [58]. A smaller-scale version of witness cosigning could be instantiated by cross-logging STHs in other CT logs [23], or in other append-only ledgers [59]. CTor's full design (Section 4) ensures that anyone connected to the Tor network is on the same view by making STHs public in the Tor consensus. In contrast, the first incremental design (Section 6) is not concerned with catching log misbehavior, while the second incremental design (also Section 6) exposes misbehaving logs *without* first trying to fetch inclusion proofs.

Nordberg proposed that Tor clients could enforce public logging of consensus documents and votes [45]. Such an initiative is mostly orthogonal to CTor, as it strengthens 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 Tor Browser could check self-signed TLS certificates for exact matches on independent Tor circuits [68]. 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 double-checking, except that there are normally no TLS handshake blocking, browser warnings, or strict assumptions regarding the attacker's location.

In parallel Stark and Thompson proposed that Chrome could submit a random subset of encountered SCTs to a trusted auditor that Google runs [56]. CTor also propagates a random subset of SCTs to a trusted auditor, but does so while preserving privacy because of and how Tor is used. Meiklejohn additionally proposed witness cosigning on-top of consistent STHs [42]. CTor adds signatures on-top of STHs too, but only as part of the Tor consensus that directory authorities sign.

10 Conclusion

We proposed CTor, a privacy-preserving and incrementally-deployable design that brings CT to Tor. Tor Browser should start by taking the same proactive security measures as Google Chrome and Apple's Safari: require that a certificate is only valid if accompanied by at least two SCTs. Such CT enforcement narrows down the attack surface from the weakest-link security of the CA ecosystem to a relatively small number of trusted log operators *without negatively impacting the user experience to an unacceptable degree*. The problem is that a powerful attacker may gain control of the required logs, trivially circumventing enforcement without significant risk of exposure. If deployed incrementally, CTor relaxes the currently deployed trust assumption by distributing it across all CT logs. If the full design is put into operation, such trust is completely eliminated.

CTor repurposes Tor relays to ensure that today's trust in CT logs is not misplaced: Tor Browser probabilistically submits the encountered certificates and SCTs to Tor relays, which cross-log them into independent CT logs (incremental design) or request inclusion proofs with regards to a single fixed view (full design). It turns out that delegating verification to a party that can defer it is paramount in our setting, both for privacy and security. Tor and the wider web would greatly benefit from each design increment. The full design turns Tor into a system for maintaining a probabilistically-verified view of the entire CT log ecosystem, provided in Tor's consensus for anyone to use as a basis of trust. The idea to cross-log certificates and SCTs further showcase how certificate mis-issuance and suspected log misbehavior could be disclosed casually without any manual intervention by using the log ecosystem against the attacker.

The attacker's best bet to break CTor involves any of the following: operating significant parts of the CTor infrastructure, spending a reliable Tor Browser zero-day that escalates privileges within a tiny time window, or targeting all Tor relays in an attempt to delete any evidence of certificate mis-issuance and log misbehavior. The latter—a so-called network-wide flush—brings us to the border of our threat model, but it cannot be ignored due to the powerful attacker that we consider. Therefore, CTor is designed so that Tor can *adapt* in response to interference. For example, in Tor Browser the `ct-large-sfo-size` could be set reactively 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. The distribution that adds random buffering delays could also be updated, as well as log operator relationships during the auditing phase.

Acknowledgements

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 *CCS*, 2019.

[2] M. Alicherry and A. D. Keromytis. DoubleCheck: Multi-path verification against man-in-the-middle attacks. In *ISCC*, 2009.

[3] Apple Inc. Apple's certificate transparency log program, January 2019. https://support.apple.com/en-om/HT209255, accessed 2020-12-15.

[4] Bugzilla. Implement certificate transparency support (RFC 6962), 2020. https://bugzilla.mozilla.org/show_bug.cgi?id=1281469, accessed 2020-12-15.

[5] Catalin Cimpanu. Exploit vendor drops Tor Browser zero-day on Twitter, 2018. https://web.archive.org/web/20200529194530/https://www.zdnet.com/article/exploit-vendor-drops-tor-browser-zero-day-on-twitter/, accessed 2020-12-15.

[6] M. Chase and S. Meiklejohn. Transparency overlays and applications. In *CCS*, 2016.

[7] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *FOCS*, 1995.

[8] L. Chuat, P. Szalachowski, A. Perrig, B. Laurie, and E. Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *CNS*, 2015.

[9] J. Clark and P. C. van Oorschot. SoK: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements. In *IEEE S&P*, 2013.

[10] 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), 2016.

[11] CT policy mailing list (n.d.). Certificate transparency policy. https://groups.google.com/a/chromium.org/forum/#!forum/ct-policy, accessed 2020-12-15.

[12] R. Dahlberg and T. Pulls. Verifiable light-weight monitoring for certificate transparency logs. In *NordSec*, 2018.

[13] R. Dahlberg, T. Pulls, T. Ritter, and P. Syverson. SFO distribution artificat, December 2020. https://github.com/rgdd/ctor/tree/master/artifact, accessed 2020-12-15.

[14] R. Dahlberg, T. Pulls, J. Vestin, T. Høiland-Jørgensen, and A. Kassler. Aggregation-based certificate transparency gossip. In *SECURWARE*, 2019.

[15] G. Danezis and P. Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In *PETS*, 2008.

[16] R. Dingledine, N. Mathewson, and P. F. Syverson. Tor: The second-generation onion router. In *USENIX Security*, 2004.

[17] B. Dowling, F. Günther, U. Herath, and D. Stebila. Secure logging schemes and certificate transparency. In *ESORICS*, 2016.

[18] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman. Analysis of the HTTPS certificate ecosystem. In *IMC*, 2013.

[19] P. Eckersley. A Syrian man-in-the-middle attack against Facebook, 2011. https://www.eff.org/deeplinks/2011/05/syrian-man-middle-against-facebook, accessed 2020-12-15.

[20] S. Eskandarian, E. Messeri, J. Bonneau, and D. Boneh. Certificate transparency with privacy. *PoPETs*, 2017(4), 2017.

[21] N. S. Evans, R. Dingledine, and C. Grothoff. A practical congestion attack on Tor using long paths. In *USENIX Security*, 2009.

[22] 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 2020-12-15.

[23] Google LLC. Minimal gossip, May 2018. https://github.com/google/trillian-examples/blob/master/gossip/minimal/README.md, accessed 2020-12-15.

[24] Google LLC. Chromium certificate transparency policy, October 2020. https://github.com/chromium/ct-policy/blob/master/README.md, accessed 2020-12-15.

[25] Google LLC. Trillian log signer, June 2020. https://github.com/google/trillian/blob/master/cmd/trillian_log_signer/main.go, accessed 2020-12-15.

[26] Google LLC. (n.d.). HTTPS encryption on the web. https://transparencyreport.google.com/https/overview?hl=en, accessed 2020-05-19.

[27] B. Greschbach, T. Pulls, L. M. Roberts, P. Winter, and N. Feamster. The effect of DNS on Tor's anonymity. In *NDSS*, 2017.

[28] K. Hill. How did the FBI break Tor?, 2014. https://www.forbes.com/sites/kashmirhill/2014/11/07/how-did-law-enforcement-break-tor/#6cf2ed594bf7, accessed 2020-12-15.

[29] D. Kales, O. Omolola, and S. Ramacher. Revisiting user privacy for certificate transparency. In *IEEE EuroS&P*, 2019.

[30] D. Kesdogan, J. Egner, and R. Büschkes. Stop-and-Go MIXes: Providing probabilistic anonymity in an open system. In *IH*, 1998.

[31] N. Korzhitskii and N. Carlsson. Characterizing the root landscape of certificate transparency logs. In *IFIP Networking*, 2020.

[32] A. Langley. Enhancing digital certificate security, 2013. https://security.googleblog.com/2013/01/enhancing-digital-certificate-security.html, accessed 2020-12-15.

[33] A. Langley. No, don't enable revocation checking, 2014. https://www.imperialviolet.org/2014/04/19/revchecking.html, accessed 2020-12-15.

[34] B. Laurie. Certificate transparency. *Commun. ACM*, 57(10), 2014.

[35] B. Laurie. Certificate transparency over DNS, March 2016. https://github.com/google/certificate-transparency-rfcs/blob/master/dns/draft-ct-over-dns.md, accessed 2020-12-15.

[36] B. Laurie, A. Langley, and E. Kasper. Certificate transparency. RFC 6962, IETF, 2013.

[37] 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, 2019.

[38] 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 *CCS*, 2019.

[39] W. Lueks and I. Goldberg. Sublinear scaling for multi-client private information retrieval. In *FC*, 2015.

[40] A. Mani, T. Wilson-Brown, R. Jansen, A. Johnson, and M. Sherr. Understanding Tor usage with privacy-preserving measurement. In *IMC*, 2018.

[41] B. McMillion. Un-incorporated SCTs from GDCA1, 2018. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/Emh3ZaU0jqI, accessed 2020-12-15.

[42] S. Meiklejohn, P. Kalinnikov, C. S. Lin, M. Hutchinson, G. Belvin, M. Raykova, and A. Cutter. Think global, act local: Gossip and client audits in verifiable data structures, 2020. https://arxiv.org/pdf/2011.04551.pdf, accessed 2020-12-15.

[43] Mozilla. Mozilla research grants 2019H1, 2019. https://web.archive.org/web/20200528174708/https://mozilla-research.forms.fm/mozilla-research-grants-2019h1/forms/6510, accessed 2020-12-15.

[44] Mozilla (n.d.). SSL ratios. https://docs.telemetry.mozilla.org/datasets/other/ssl/reference.html, accessed 2020-05-19.

[45] L. Nordberg. Tor consensus transparency, June 2014. https://github.com/torproject/torspec/blob/master/proposals/267-tor-consensus-transparency.txt, accessed 2020-12-15.

[46] L. Nordberg, D. K. Gillmor, and T. Ritter. Gossiping in CT. Internet-draft draft-ietf-trans-gossip-05, IETF, 2018.

[47] P. H. O'Neill. Telegram traffic from around the world took a detour through Iran, 2018. https://www.cyberscoop.com/telegram-iran-bgp-hijacking/, accessed 2020-12-15.

[48] M. Perry, E. Clark, S. Murdoch, and G. Koppen. The design and implementation of the Tor Browser [DRAFT], June 2018.

[49] J. Prins. DigiNotar certificate authority breach "operation black tulip". Interim report, Fox-IT, 2011.

[50] J. Rowley. CT2 log compromised via Salt vulnerability, 2020. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/aKNbZuJzwfM, accessed 2020-12-15.

[51] A. Siddiqui. What happened? The Amazon Route 53 BGP hijack to take over Ethereum cryptocurrency wallets, 2018. https://www.internetsociety.org/blog/2018/04/amazons-route-53-bgp-hijack/, accessed 2020-12-15.

[52] R. Sleevi. Upcoming CT log removal: Izenpe, 2016. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/qOorKuhL1vA, accessed 2020-12-15.

[53] R. Sleevi. Upcoming log removal: Venafi CT log server, 2017. https://groups.google.com/a/chromium.org/forum/#!topic/ct-policy/KMAcNT3asTQ, accessed 2020-12-15.

[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 2020-12-15.

[55] E. Stark, R. Sleevi, R. Muminovic, D. O'Brien, E. Messeri, A. P. Felt, B. McMillion, and P. Tabriz. Does certificate transparency break the web? Measuring adoption and error rate. In *IEEE S&P*, 2019.

[56] E. Stark and C. Thompson. Opt-in SCT auditing, September 2020. https://docs.google.com/document/d/1G1Jy8LJgSqJ-B673GnTYIG4b7XRw2ZLtvvSlrqFcl4A/edit, accessed 2020-12-15.

[57] J. Stewart. BGP hijacking for cryptocurrency profit, 2014. https://www.secureworks.com/research/bgp-hijacking-for-cryptocurrency-profit, accessed 2020-12-15.

[58] 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 S&P*, 2016.

[59] A. Tomescu and S. Devadas. Catena: Efficient non-equivocation via Bitcoin. In *IEEE S&P*, 2017.

[60] Tor Project. Getting up to speed on Tor's past, present, and future. https://2019.www.torproject.org/docs/documentation.html.en, accessed 2020-12-15.

[61] Tor project. Advertised and consumed bandwidth by relay flag, 2020. https://metrics.torproject.org/bandwidth-flags.html, accessed 2020-05-30.

[62] Tor project. Relays by relay flag, 2020. https://metrics.torproject.org/relayflags.html, accessed 2020-05-29.

[63] Tor Project (n.d.). Functions to queue create cells for processing. https://src-ref.docs.torproject.org/tor/onion___queue_8c_source.html, accessed 2020-12-15.

[64] Tor project (n.d.). Relay requirements. https://community.torproject.org/relay/relays-requirements/, accessed 2020-05-29.

[65] U.S. Dept. of Justice. More than 400 .onion addresses, including dozens of 'dark market' sites, targeted as part of global enforcement action on Tor network, 2014. https://www.fbi.gov/news/pressrel/press-releases/more-than-400-.onion-addresses-including-dozens-of-dark-market-sites-targeted-as-part-of-global-enforcement-action-on-tor-network, accessed 2020-12-15.

[66] Wikipedia contributors. BGP hijacking—Wikipedia, the free encyclopedia, 2020. https://en.wikipedia.org/w/index.php?title=BGP_hijacking&oldid=964360841, accessed 2020-12-15.

[67] C. Wilson. CT days 2020. https://groups.google.com/a/chromium.org/g/ct-policy/c/JWVVhZTL5RM, accessed 2020-12-15.

[68] P. Winter, R. Köwer, M. Mulazzani, M. Huber, S. Schrittwieser, S. Lindskog, and E. R. Weippl. Spoiled onions: Exposing malicious Tor exit relays. In *PETS*, 2014.

[69] Zerodium. Tor Browser zero-day exploit bounty (expired), 2017. https://web.archive.org/web/20200528175225/https://zerodium.com/tor.html, accessed 2020-12-15.

[70] Zerodium. Our exploit acquisition program, 2020. https://web.archive.org/web/20200521151311/https://zerodium.com/program.html, accessed 2020-05-21.

A Detailed Consensus 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 10th SFO is sent to a random CTR on average.

ct-large-sfo-size: A natural number that determines how many wire-bytes a normal SFO should not exceed. As outlined in Section 4.1, 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., 5 seconds. As outlined in Section 4.3, a timeout causes the watchdog to send an SFO to the auditor.

ct-delay-dist: A distribution that determines how long a CTR should wait at minimum before auditing a submitted SFO. As outlined in Section 4.2, 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 4.3, CTRs audit pending SFOs in batches at random time intervals to spread out log overhead.

ct-watchdog-timeout: A natural number that determines how long time at most a watchdog waits before considering an SFO for reporting. Prevents the watchdog from having to wait for a circuit timeout caused by an unresponsive CTR. Should be set with `ct-backoff-dist` in mind.

ct-auditor-timeout A natural number that determines how long time at most a watchdog waits for an auditor to acknowledge the submission of an SFO.

attacker does not control. Google's CT policy currently qualifies 36 logs that are hosted by Cloudflare, DigiCert, Google, Let's Encrypt, Sectigo, and TrustAsia [24]. No log accepts all roots, but the overlap between root certificates that are trusted by major browser vendors and CT logs increased over time [31]. This trend would likely continue if there are user agents that benefit from it, e.g., Tor Browser. Despite relatively few log operators and an incomplete root coverage, the basic and extended cross-logging in CTor still provide significant value as is:

– Even if there are no independent logs available for a certificate issued by some CA, adding it again *to the same logs* would come with practical security gains. For example, if the attacker gained access to the secret signing keys but not the logs' infrastructures the mis-issued certificate trivially makes it into the public. If the full SFO is added, the log operators could also notice that they were compromised.

– Most log operators only exclude a small fraction of widely accepted root certificates: 1–5% [31]. 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 likely harder to take control of Let's Encrypt which some logs and operators exclude due to the sheer volume of issued certificates than, say, a smaller CA that law enforcement may coerce.

Browser-qualified or not, the availability of independent logs that accept the commonly accepted root certificates provides significant ecosystem value. Log misbehavior is mostly reported through the CT policy mailing list. Thus, it requires manual intervention. Wide support of certificate chain and SCT cross-logging allows anyone to *casually* disclose suspected log misbehavior on-the-fly.

B Log Operators & 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" [36, 37]. Apple further claims that a log in their CT program "must trust all root CA certificates included in Apple's trust store" [3]. This bodes well for the incremental CTor designs: we assumed that the existence of independent log operators implies the ability to at least add certificate chains and possibly complete SFOs into logs that the

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 4, which can be used to compute the number

of SFO submissions that the attacker needs to flush a buffer of $n > 2$ entries with some probability $p \in [0, 1)$.

$$k = \frac{\log(1 - p)}{\log(1 - \frac{1}{n})} \tag{4}$$

It is recommended that a non-exit relay should have at least 512MB of memory. If the available bandwidth exceeds 40Mbps, it should have at least 1GB [64]. Given that these recommendations are lower bounds, suppose the average memory available to store SFOs is 1GiB. Section 7 further showed that the average SFO size is roughly 6KiB. This means that the buffer capacity is $n \leftarrow 174763$ SFOs. Plugging it into Equation 4 for $p \leftarrow \frac{9}{10}$, the attacker's flood must involve $k \leftarrow 402406$ submissions. In other words, 2.3GiB must be transmitted to flush a single CTR with 90% success probability.

As a corner case and implementation detail it is important that Tor Browser and CTRs *reject* SFOs that are bogus in terms of size: it is a trivial DoS vector to load data indefinitely. If such a threshold is added the required flushing bandwidth is still 2.3GiB (e.g., use 1MiB SFOs in the above computations). What can be said about bandwidth and potential adversarial advantages is that a submitted SFO yields amplification: twofold for cross-logging, and slightly more for proof-fetching as the SFO is pushed up-front to a watchdog. Note that such amplification is smaller than a typical website visit.