Sarah Meiklejohn, Joe DeBlasio, Devon O'Brien, Chris Thompson, Kevin Yeo, and Emily Stark

SoK: SCT Auditing in Certificate Transparency

Abstract: The Web public key infrastructure is essential to providing secure communication on the Internet today, and certificate authorities play a crucial role in this ecosystem by issuing certificates. These authorities may misissue certificates or suffer misuse attacks, however, which has given rise to the Certificate Transparency (CT) project. The goal of CT is to store all issued certificates in public logs, which can then be checked for the presence of potentially misissued certificates. Thus, the requirement that a given certificate is indeed in one (or several) of these logs lies at the core of CT. In its current deployment, however, most individual clients do not check that the certificates they see are in logs, as requesting a proof of inclusion directly reveals the certificate and thus creates the clear potential for a violation of that client's privacy. In this paper, we explore the techniques that have been proposed for privacy-preserving auditing of certificate inclusion, focusing on their effectiveness, efficiency, and suitability in a near-term deployment. In doing so, we also explore the parallels with related problems involving browser clients. Guided by a set of constraints that we develop, we ultimately observe several key limitations in many proposals, ranging from their privacy provisions to the fact that they focus on the interaction between a client and a log but leave open the question of how a client could privately report any certificates that are missing.

Keywords: Certificate Transparency, SCT auditing

DOI 10.2478/popets-2022-0075
Received 2021-11-30; revised 2022-03-15; accepted 2022-03-16.

1 Introduction

The basis for all secure communication on the Internet today is the ability of a domain operator to associate with their domain name a private key that they possess. This is achieved using digital certificates, which are issued by certificate authorities (CAs) and which websites can present to clients, who can check that they are valid by ensuring they are signed by, or have a signature chain rooted in, a trusted CA. If a CA is compromised, it can be used to issue false certificates that in turn would allow an attacker to eavesdrop on the communication between clients and a website. Furthermore, CAs may simply fail to fully verify a domain owner's identity and misissue a certificate. Both of these scenarios have happened [67], which illustrates the need for additional mechanisms to avoid trusting CAs unconditionally.

It was out of this need that Certificate Transparency (CT) was created, with the goal of increasing the visibility of certificates that are issued and thus reducing the time required to detect misissued certificates or other forms of CA misbehavior. Briefly, this goal is achieved in CT by requiring all issued certificates to be placed in one or more public logs. Upon receiving a certificate, a log operator checks its structural validity and — assuming it is valid — responds with a *signed certificate timestamp* (SCT), which acts as a promise to include the certificate in the log within some bounded delay. These SCTs are then presented alongside the certificate to the client, who checks that the SCTs are valid and that there are enough SCTs according to the policy set by their browser vendor. Separately, designated *auditors* and *monitors* are responsible for checking, respectively, that logs are append-only and globally consistent and that the actual content of the logs does not contain any inconsistencies or misissued certificates.

In order for CT to achieve its goal, it is crucial that clients check not only the validity of the SCTs they see, but also that the promises implicit in these SCTs have been fulfilled; i.e., that these certificates are in the log. In this way, assuming auditors are doing their job and everyone has access to the same log contents, clients can be sure that the certificates they see will eventually be examined by a monitor. If clients do not check that certificates are in the log, then even if they were misissued they may never be detected by a monitor. Having a client check this directly with a log operator is a clear privacy violation, however, as over time it amounts to having the client present the log operator with their browsing history. Thus, in the current deployment of CT, most clients do not perform this crucial check.

Related work. In 2013, Clark and van Oorschot systematically explored the space of known security issues

Sarah Meiklejohn, Joe DeBlasio, Devon O'Brien, Chris Thompson, Kevin Yeo, Emily Stark: Google LLC, {meiklejohn, jdeblasio, asymmetric, cthomp, kwlyeo, estark}@google.com

with HTTPS and evaluated proposals for improving the certificate infrastructure in terms of their security and privacy, deployability, and usability [21]. In 2020, Chuat et al. explored the landscape of solutions to the problems of delegation and revocation in the web public-key infrastructure [18]. Our paper is complementary to these works: rather than perform a broad exploration of secure communication on the Web and an evaluation of long-term replacements or improvements, we seek to identify a pragmatic solution to a specific problem, SCT auditing, that exists within the current HTTPS ecosystem. This could be used to further secure the ecosystem as it exists today.

Our contributions. Our contributions in this work are as follows:

- We survey the landscape of proposed solutions for the problem of privacy-preserving SCT auditing from the academic literature and the community of CT practitioners. We unify proposed solutions within a common framework of a *querying* and *reporting* phase.
- We compare SCT auditing to other privacy-sensitive problems in web browsers. We observe that while SCT auditing is a unique problem, some insights and approaches from other problems are applicable.
- We systematically evaluate proposed solutions on several dimensions: privacy, security, costs, and deployability.
- From our systematic evaluation, along with experimental data against real-world CT logs and insights from the deployment of CT and other related systems, we conclude that existing proposals suffer from crucial limitations. We extract a set of constraints that future SCT auditing solutions should adhere to, with the hope of guiding the community towards a complete solution.

2 SCT Auditing

Our description of Certificate Transparency (CT) and its components follows RFC 6962 [41]. We refer at various points to specific aspects of another RFC, 6962-bis [42], which describes "version 2.0 of the Certificate Transparency protocol." To the best of our knowledge, RFC 6962 is an accurate description of CT as it exists today and there have been no announced plans to migrate CT to RFC 6962-bis.

2.1 How Certificate Transparency works

At a high level, CT works as follows: a website operator requests a *certificate* from a CA, which should serve to bind together a domain name and a public key. CT requires that every such certificate must appear in at least one globally visible *log*. These logs are append-only, which means that once a certificate has been added it cannot be removed. CT thus exposes every certificate that has ever been issued.

The only restriction that logs place on the contents they store is that all certificate chains must lead back to one of a set of trusted root CAs, published by the log. For almost all CAs, this means that the current advice is to effectively broadcast a (pre)certificate to all log operators of whom they are aware. While certificates are typically submitted to logs by the CA at the time they are issued, any other party can also do so at any time after the certificate has been issued.

In order for a certificate to be valid, it should have at least one *signed certificate timestamp* (SCT) embedded into it, which proves that the certificate has been submitted to a log and promises that it will be included in the log within some *maximum merge delay* (MMD). Having the CA wait only to receive this promise has several advantages over having them wait until the certificate is actually included, such as allowing CAs to issue certificates without relying on logs to perform a computationally intensive and potentially time-consuming process; this freedom for logs to update less frequently can also have a positive impact on privacy. We discuss this in more detail in Sections 6.1 and 6.3.

These SCTs can be delivered to clients in three ways: (1) by embedding them in the certificate directly using its extensions (which requires the CA to have submitted a precertificate before issuance); (2) using a special TLS extension; and (3) using OCSP (Online Certificate Status Protocol) stapling, which allows the CA to provide (and potentially obtain) the SCT after the certificate has been issued. Importantly, these three options and the fact that certificates can be submitted to logs at any time means there is no canonical set of SCTs associated with a given certificate.

Once a certificate has been included in the log, any party can request an *inclusion proof* from the log, which proves the certificate's inclusion with respect to a timestamped commitment to the log's contents known as a *signed tree head* (STH). Anyone can furthermore request a *consistency proof* between two STHs, which proves that the log operates in an append-only manner. A party in possession of some known STH may need to obtain

Operator	Log name	Log size	Log shard		
			2021	2022	2023
Cloudflare	Nimbus	–	408.9	382.2	6.7
DigiCert	Nessie	–	99.1	134.1	22.1
	Yeti	–	305.3	66.5	22.2
Google	Argon	–	1356.2	671.4	25.5
	Xenon	–	1464.4	833.8	26.3
	Icarus	762.4	–	–	–
	Pilot	1077.3	–	–	–
	Rocketeer	1113.9	–	–	–
	Skydiver	309.2	–	–	–
Let's Encrypt	Oak	–	365.3	242.9	20.6
Sectigo	Sabre	199.9	–	–	–
	Mammoth	606.9	–	–	–
TrustAsia	Log	–	0.7	1.0	0.06

Table 1. All usable CT logs and their respective sizes, in terms of millions of entries, as of February 23, 2022. For the temporally sharded logs, we present the sizes of the 2021, 2022, and 2023 shards.

and verify both types of proofs in order to fully verify the inclusion of an entry in the log: first an inclusion proof with respect to a given STH, and then a consistency proof between this STH and their known one. The only way this second step would not be required is if the log operator happened to use the party's known STH in their inclusion proof (e.g., it had not yet issued a new STH for the log).

2.2 The CT ecosystem

The issuance process described above almost always takes place between the CA and the log, but there are other important participants in the CT ecosystem. All CT-enforcing user agents (i.e., browser vendors) have their own policy in terms of how many SCTs they require a certificate to have. For example, Apple (Safari) requires SCTs from at least two distinct log operators and Google (Chrome) requires one SCT from a Google-operated log and one from a non-Google log.

Table 1 contains a list of all currently usable CT logs and their operators.[1] As we can see, there are currently billions of certificates stored in these logs, and there are millions added every day [7]. To control the growth of any one log, new CT logs are required to be *temporally sharded* [1], with each log shard containing all certificates that expire within a given time range (typically one calendar year). Nevertheless, the sheer number of certificates across all logs makes it difficult for an individual domain owner to check for new (and possibly unauthorized) certificates being issued to their domain. In order to simplify this task, *monitors* act as mirrors and provide search and notification services [8, 44]. Some of the current prominent examples of monitors are crt.sh (https://crt.sh) and Cert Spotter (https://sslmate.com/certspotter/). While monitors thus help detect misbehavior on the part of CAs, *auditors* are responsible for detecting misbehavior on the part of logs. This role can be played by any entity with the ability to report or act on misbehavior; e.g., to initiate the process of removing a log from the ecosystem [26, 32, 52, 53, 60, 61]. Notably, individual browser instances are not in a position to act as auditors, at least not without the help of another entity such as the browser vendor.

In order to make sure that logs follow up on the promise implicit in an SCT, it is necessary to check that the certificate represented by this SCT is in fact included in the log, and act accordingly if not. We refer to this process as *SCT auditing*. Individual browser instances, however, are the only participants who reliably see SCTs "in the wild", by visiting websites and receiving certificates, and having them perform SCT auditing is a clear privacy problem: an SCT uniquely identifies a certificate and thus a domain (or set of domains if wildcards or alternative names are used), so reveals the website visited by the browser. Having an individual browser instance reveal SCTs to another party, whether it is the log or an auditor, thus reveals to that party some of the browsing history of that user. The problem of how to audit SCTs without compromising user privacy has been open since the introduction of CT.

2.3 Threat model

As described above, there are three core actors in an SCT auditing ecosystem: (1) users, whose browsers see SCTs as they visit websites; (2) logs, which store certificates and serve inclusion proofs for them; and (3) auditors, who are responsible for acting on evidence of log misbehavior. Ultimately, this means the goal of SCT auditing is for an honest auditor to learn about any SCTs whose implicit promises have been violated.

We assume that users, auditors, and logs can all be malicious; i.e., can attempt to deviate from the protocol in arbitrary ways. Importantly though, we assume that malicious parties cannot cause honest parties to deviate

1 This list is taken from https://www.gstatic.com/ct/log_list/v3/log_list.json.

from the protocol; e.g., if the auditor is a browser vendor they cannot inject malicious code into the browser to affect the behavior of an honest user. We also assume that all log operators use secure cryptographic standards, means they cannot form valid inclusion or consistency proofs unless, respectively, an entry really is in the log or the log really is append-only.

We break the problem of SCT auditing into two phases: *querying* and *reporting*. We model each phase as an interaction between a client and a server. In the querying phase, the server is the log operator, and the client's goal is to learn whether or not a specific entry is included in the log. In the reporting phase, the server is an auditor, and the goal is for them to learn about any entries that were not included in the log.

In both phases, the client is intentionally left generic, allowing it to represent individual browser instances but potentially other participants (e.g., auditors or web servers) as well. In terms of privacy, the information in a query or report — meaning the certificate or SCT it contains — is not sensitive; i.e., it does not inherently leak anything about individual clients. In either phase, the goal of an adversary is thus to either link together two queries (i.e., to learn that they came from the same user) or to link an individual user to a specific certificate; i.e., to learn that they queried on or reported a specific certificate.

3 Related Problems

We consider three problems that are related to the problem of SCT auditing, in terms of providing protection to users as they browse the Internet.

3.1 Safe Browsing

In order to protect users from phishing, their browsers can periodically check whether or not the sites they visit are on a blocklist maintained by Google. The Safe Browsing API [9] supports two types of interactions: basic lookups and updates. A basic lookup reveals the queried URL in the clear, and is analogous to a CT log's API for fetching inclusion proofs. The `Update` call allows clients to perform a local lookup before deciding whether or not to interact with a Safe Browsing endpoint directly. Briefly, clients can store the hash prefixes of URLs on the Safe Browsing list in a compressed data structure similar to a Bloom filter. When a client visits a URL, they hash it and check if its hash prefix is in the filter or not. If not, then the URL is definitely not on the Safe Browsing blocklist (or at least the version of it reflected by the client's filter). If it is, then they call the API on the hash prefix to get back a list of (full) hashes of all URLs on the blocklist that have that prefix. If the hash of the URL is on that list then it is on the blocklist (and thus considered unsafe), and otherwise it is not.

Currently, many browser vendors integrate with Safe Browsing, meaning their users are by default opted in to use the `Update` API and update their local filter every 30 minutes (they also have the ability to opt out). Users of Chrome may additionally opt in to "enhanced" Safe Browsing [56], which means their browsers query the lookup API in real time.

The problem that Safe Browsing is solving is in some sense the inverse of the problem in SCT auditing: in Safe Browsing users are looking for a match in a relatively small list of URLs (the blocklist), whereas in CT users are looking for a missing entry in a large list (the contents of a CT log). As we explore further in Section 4.4.2.1, this means the approach used in Safe Browsing cannot be used directly in CT.

3.2 Checking for certification revocation

When verifying a certificate, it is important to ensure that it has not been *revoked*. One way to check for revocation is using the Online Certificate Status Protocol (OCSP). This allows CAs to tell clients the status of a certificate via an endpoint that clients query directly. Using OCSP directly thus presents a privacy issue, as clients reveal the certificates they see to the CA.

There are two basic alternatives to OCSP that exist today. Using OCSP Stapling, the certificate holder (i.e., the web server) queries the CA rather than the client, and then "staples" the signed response from the CA to the certificate when it serves it. Since the web server can perform these queries periodically and cache the response until some expiration date, this improves not only privacy but also performance. In the other alternative, clients can query endpoints maintained by CAs to download certificate revocation lists (CRLs). Clients can then check certificates against locally cached versions of these CRLs, but this imposes a high storage overhead and requires clients to keep their lists up-to-date. To address these limitations, CRLite [34, 40] stores revocation data in a compressed data structure similar to a Bloom filter, while CRLSets [2] are revocation lists

of size at most 250kB that are curated by Google and pushed regularly to Chrome browsers.

As compared with SCT auditing, certificate revocation checks are designed to be performed in-band during the setup of a connection, whereas CT is designed to have asynchronous auditing. Furthermore, it is possible to have a canonical list of all revoked certificates but not possible to enumerate all SCTs/certificates that are not included in a CT log.

3.3 Checking for compromised credentials

Users who are concerned that their credentials may have been compromised can query and check whether or not these credentials are on a list of breached credentials, as provided by a service such as Google's Password Checkup [66] or Have I Been Pwned? (HIBP) [4]. While this is not directly related to browsing, many browsers have integrated some form of checking; e.g., the Firefox Monitor [3] uses HIBP, while Safari and Chrome use their own custom lists.

Perhaps surprisingly, this problem is the one that most closely resembles SCT auditing, as it also involves asynchronous querying in a large database. Nevertheless, there are also important differences, such as users needing to act in the case of matching rather than missing data (as with Safe Browsing). We discuss existing protocols for checking for compromised credentials (C3) and how they can be adapted for use in CT in more detail in Section 4.3.2.3.

4 Components of SCT Auditing

4.1 Literature review

Our goal is to identify proposed solutions for either phase of SCT auditing. Given that CT is a deployed project as well as an area of academic research, we identified solutions based on a manual review of three different sources: (1) academic literature published in computer security and networking conferences (ACM CCS, USENIX Security, IEEE S&P, NDSS, NSDI, and CNS), (2) experimental deployments in industry, and (3) posts on Certificate Transparency mailing lists [6] and standards documentation. The final list of proposals was compiled and categorized jointly by a set of three researchers, and was validated by a broader set of researchers and colleagues. In addition to looking at proposals for CT, we also looked at discussions of and proposed solutions for the three related problems introduced in the previous section.

4.2 Evaluation criteria

We consider seven main aspects that characterize a proposal for either phase of SCT auditing. A summary of all proposals against these evaluation criteria is in Table 2.

Integrity: We require that all proposals achieve integrity. In the querying phase, this means that the client accepts only entries that are included in the log. In the reporting phase, it means that the auditor acts only on SCTs that have been violated.

Privacy: We consider whether or not a proposal preserves privacy. Following Section 2.3, this means that it is difficult for the server to link a specific user (as represented by their browser instance) to a specific entry. We use ○ to indicate that no privacy is achieved; ◐ to indicate that k-anonymity is achieved, meaning the server knows either that one of k clients was interested in a specific entry or that a specific client was interested in one of k entries; and ● to indicate that unlinkability is provably achieved.

Client costs: We consider three costs that clients might incur: bandwidth, storage, and computation. We use ○ to indicate that there is no overhead; ◐ to indicate that there is some overhead but the client could still likely be run on a modern mobile device; and ● to indicate that there is enough overhead that it likely could not.

Certificate issuance latency: We consider any latency the protocol adds to the process of certificate issuance. We use 'none' to indicate that there is no added latency, and otherwise describe what is needed before a certificate can be issued.

Server costs: We consider the costs for the server (i.e., the log or auditor) to run the protocol. We again use ○ to indicate that there is no required overheaad, ◐ to indicate minimal overhead, and ● to indicate that the server would have additional requirements at least at the same scale as it does during normal operation (i.e., its requirements would at least be doubled).

Trust assumptions: We consider the assumptions needed for the protocol to satisfy both privacy and integrity, in terms of which participants have to trust which other participants to be sure that the correct information is communicated in a privacy-preserving

way. We use 'none' if there are no trust assumptions, and otherwise describe them.

Near-term deployability: We consider how possible it would be to deploy the protocol within the next 2-3 years. This factors in both costs and trust assumptions, in terms of whether or not there are natural participants in the CT ecosystem who can play these roles. We use ○ to indicate that there are major obstacles, ◐ to indicate significant but not insurmountable obstacles, and ● to indicate that near-term deployability is a reasonable expectation.

4.3 Proposals for querying

We first describe protocols for the querying phase, in which the goal is for a client to find out from a log operator whether or not a specific entry is included in the log without the log being able to link that specific entry to a specific client.

4.3.1 Network-level anonymization

We first discuss proposals in which the client provides the queried certificate/SCT in the clear, but their identity may be hidden from the log at the network layer.

4.3.1.1 Query directly

The simplest proposal for querying a log is to have the client do so directly; i.e., to request an inclusion proof from the log for a given certificate. This can be deployed easily, and similarly requires little overhead for an individual client so is performant. Following Section 2.3, the fact that the client receives an inclusion proof directly from the log means the protocol achieves integrity.

The protocol does not achieve any privacy for the client, as it reveals its certificates directly to the log. There are two possibilities: first, a client represents an individual browser, and the log thus learns the website visited by that browser. This achieves no privacy at all. Second, a client represents an auditor to whom a collection of individual browsers have reported one or multiple certificates; e.g., a browser vendor with whom some users have opted to share a portion of their browsing history. In this case, individual users achieve k-anonymity, where k is the total number of users of that auditor, but as they reveal their certificates directly to the auditor they must trust it to not share them externally. We discuss this second case in more detail in Section 5.3.

4.3.1.2 Proxy/mixnet

Rather than have each client contact the log directly, clients could route their queries through a single proxy server or a series of proxies; i.e., a mixnet, as mentioned by Eskandarian et al. [27] and as used implicitly in the CTor protocol for Tor clients due to Dahlberg et al. [23].

This protocol is performant and could be deployed in the near term; e.g., browsers could act as a proxy for CT queries just as some of them currently act as a proxy for Safe Browsing queries [10, 16]. As in the previous proposal, it achieves integrity as the client receives an inclusion proof from the log. It protects the privacy of the client as long as there is sufficient traffic, as clients achieve k-anonymity with respect to the set of clients using the proxy at a given point in time. This assumes, however, that the proxy servers are not colluding with the log, which is a problem for companies like Google that both offer a browser and run CT logs. Using more proxies makes it less likely that they are all colluding, but adds latency to the querying protocol.

4.3.1.3 DNS

Using a proxy improves privacy by avoiding direct communication between the client and the log. Instead of adding an external proxy server, we could identify a party who already knows of the client's interest in a given certificate, and then route the query through them. One such party is a DNS resolver. In 2015, Google proposed a DNS-based protocol for fetching inclusion proofs [43, 63]. Briefly, the protocol involves the client requesting records for domain names that encode information about leaf hashes. These special DNS records then provide inclusion proofs for these leaf hashes.

This protocol is performant and could be deployed in the near term; furthermore, the fact that the client receives back inclusion proofs means it satisfies integrity. In theory, the protocol also preserves the privacy of clients, who already reveal the domain names they are interested in to their local DNS resolvers; furthermore, the logs see the request in the clear but it comes from their configured DNS resolver, which should reveal nothing about the client. In practice, however, there are known privacy risks associated with this approach [49]. For example, SCT auditing is done asynchronously, so a client might send the SCT query hours after they actually visit the site. This might cause their query to be routed through a different DNS resolver, and thus reveal information. Even with the same resolver, DNS resolvers may do a form of prefetching; i.e., resolving domain names linked to on the site that the client is cur-

Proposal	Privacy	Client costs			Certificate issuance latency	Server costs	Trust assumptions	Near-term deployability
		Bandwidth	Storage	Computation				
Query directly (browser)	○	◐	○	◐	none	○	log	●
Query directly (auditor)	◐	◐	○	◐	none	○	auditor (depending on reporting phase)	●
Proxy/mixnet	◐	◐	○	◐	none	○	no collusion between proxy/mixnet and log	●
DNS	◐	◐	○	◐	none	○	no collusion between DNS resolvers and log	●
Fuzzy ranges	◐	◐ – ●	○	◐ – ●	wait for sequencing	◐	none	◐
PIR	●†	◐	○	◐	wait for sequencing	●	no collusion between replicated logs	○
C3-PSM (log)	◐†	◐ – ●	○	◐ – ●	none	●	none	◐
C3-PSM (third party)	◐	◐ – ●	○	◐ – ●	none	○	third party (for integrity)	◐
Local mirroring	●	●	●	●	none	◐	none	●
Fast embedding	◐†	◐	○	◐	wait for inclusion	○	none	◐
Slow embedding	◐†	◐	○	◐	wait for MMD	○	none	◐
OCSP stapling	◐†	◐	○	◐	none	○	none	○
Report directly	○	◐	○	◐	–	◐	auditor	●
Proxy/mixnet	◐	◐	○	◐	–	◐	no collusion between proxy/mixnet and auditor	●
Web server	●	◐	○	○	–	◐	no persistent MitM attack; website will report	○
C3-PSM	◐*	◐ – ●	○	◐ – ●	–	●	none	●
ZKP of non-inclusion	●	●	○	●	–	◐	none	◐

Table 2. Proposals for querying the log and, below the dashed line, for reporting to an auditor. Privacy is measured in terms of the difficulty of linking a specific client to a specific entry. The † superscript indicates that the protocol achieves this level of privacy only with respect to a covert adversary [13] rather than one that is fully malicious, and the $*$ superscript indicates that it achieves this level of privacy only with respect to an honest-but-curious adversary. The ◐ – ● range indicates that there is a tunable parameter that can increase or decrease the overhead (but, as we discuss in the respective sections for these proposals, this overhead is proportional to their privacy guarantees).

rently visiting. They thus know only the domain names that they have resolved for the client, so requesting an inclusion proof provides the additional information that the client is not only resolving but actually visiting a site. As such, Google no longer seems to be focusing on this protocol as a solution for SCT auditing [62].

4.3.2 Privacy-preserving queries

Rather than focus on anonymity, the next three proposals allow the client's identity to be known to the log but use cryptographic techniques to hide the specific certificate in which they are interested.

4.3.2.1 Fuzzy ranges

Instead of having a client query for a single leaf hash or index, they could ask to see inclusion proofs for all entries in a range that they know contains their specific certificate of interest. This type of request is implicit in the work of Eskandarian et al. [27]. To obtain this range, one could imagine having a client query for either all certificates within the range of a given timestamp, or for all entries between two indices in the underlying data structure. The former approach can be problematic for privacy, as a client does not know how many certificates were logged in a given time period, so may end up with a smaller anonymity set than desired. More importantly, this feature is not supported by the current CT API [41].

The only currently available option for this type of query is thus the latter option: having clients query for all entries between two indices. This has the upside for privacy that it fixes the size of the client's anonymity set, but the downside that in order to identify the right range the certificate would need to contain its index, or *sequence number*. This adds latency into the certificate issuance process, which is a limitation we discuss further in Section 6.1. More generally, there is an inverse relationship between performance and privacy: a client's communication and computation costs are $O(k \log(N))$, so increasing the size of the anonymity set means increasing these costs and thus degrading performance. Furthermore, to avoid revealing the exact index i of the certificate, a client would need to add some random offset; i.e., they would query for the range $[i+r-\frac{k}{2}, i+r+\frac{k}{2})$ for some random value $-\frac{k}{2} < r \leq \frac{k}{2}$. In the (likely) case where the client queried multiple logs on the same certificate, however, if those logs were colluding then they would be able to perform an intersection attack [58] to identify the certificate shared by both ranges, or at least to significantly reduce the set of candidate certificates.

4.3.2.2 Private information retrieval (PIR)

Private information retrieval (PIR) aims to achieve a notion of *client privacy* in which an adversarial server learns no information about the queries made by clients. PIR solutions do not typically achieve any notion of privacy for the server, but this is not needed for CT as the contents of all logs are designed to be globally visible. In CT, the value returned to the client is an inclusion proof for their specific entry of interest.

Lueks and Goldberg were the first to propose using PIR for CT [48], and a more performant solution was later proposed by Kales, Omolola, and Ramacher [35]. Recently, Kogan and Corrigan-Gibbs proposed a PIR solution, Checklist, for the related problem of Safe Browsing [36]. The first two solutions are "traditional" PIR protocols, while Checklist is an example of offline/online PIR [22]. To avoid the high performance overhead of single-server PIR, all three solutions operate in the two-server PIR model; i.e., they require two non-colluding servers to run identical copies of the log.

All three protocols provably achieve privacy for the client in retrieving a record from the database. For the CT-specific solutions, this record consists of a certificate and an inclusion proof, which means the protocol satisfies integrity as long as the PIR database is the only one maintained by the log. Providing an inclusion proof, however, opens the protocol up to the following attack by a fully malicious log [14]. Because an inclusion proof is formed with respect to a specific STH, in addition to a specific entry in the database, a malicious log could use a unique STH for each inclusion proof; this would create a one-to-one mapping between STHs and certificates. A client wanting to verify inclusion of a certificate would need to not only verify the inclusion proof returned by the log but also verify a consistency proof between the STH used for the inclusion proof and one that they already know and trust. If they query the log directly for this consistency proof, they reveal the STH and thus reveal the certificate it represents. We defer further discussion of this issue until Section 6.3, but briefly mention here that without an additional privacy-preserving method for retrieving consistency proofs this means privacy can be achieved only with respect to a *covert* adversary rather than one that is fully malicious [13].

In terms of performance, Lueks and Goldberg evaluate their protocol on a 3GB database, which they argue can store inclusion proofs for a log of 4 million cer-

tificates. Kales et al. evaluate their protocol for logs containing 2^{28} (268M) certificates, which as we see in Section 6 is in line with the size of many CT logs today. They find that the server's computation on a query takes roughly 1 second, the work for the client takes under a millisecond, and the protocol requires 6kB in communication costs. They also evaluate the work of Lueks and Goldberg and find that it requires 3.5 seconds for the server to respond to a query and 625kB in communication costs. Kogan and Corrigan-Gibbs evaluate Checklist on a database of size 3 million (in line with the smaller number of records used in Safe Browsing) and find that the offline phase takes 11s on both the client and the server, requiring roughly 10MB in communication costs. In contrast, the online phase takes at most 1ms for both the server and the client and requires roughly 1kB in communication costs. None of these costs are prohibitive given that SCT auditing is designed to be performed asynchronously, although of course further investigation would be needed to fully assess their practicality.

In terms of deployability, both CT-specific solutions require a certificate to contain its index in the log. As discussed in Section 4.3.2.1, this adds latency to the certificate issuance process, which is a limitation we discuss in Section 6.1. For all three solutions, there are significant deployment challenges associated with keeping two servers fully synchronized. Kales et al. suggest that the second log could be hosted on a cloud platform run by a competitor of the first log operator, but this would incur a significant financial cost.

4.3.2.3 Private set membership (PSM)

Private set membership (PSM) allows a client to learn whether or not an entry is in a list held by a server, with the client learning nothing about the list beyond (non)membership of the queried entry and the server learning nothing about the client's entry. (This differentiates PSM from PIR, which does not try to provide server privacy.) This is a specific case of private set intersection (PSI) where the size of the client's set is one.

We are not aware of any research using PSM for SCT auditing, but previous research has explored its usage in the related context of checking for compromised credentials (C3) [4, 46, 57, 66]. In these "C3-PSM" protocols, a user sends a prefix of the hash of their username (which is less privacy-sensitive than their password) and the server store lists of either credentials or password hashes in *buckets* according to their hash prefix. When a user queries on a prefix, the server can thus send back the relevant bucket and the user can check locally if their particular entry is in the bucket or not.

If we adapt this approach to CT, then each log would need to sort its certificates into buckets, according to a prefix of the certificate's hash. If the client simply ran the PSM protocol with the log, however, the overall protocol would not satisfy integrity, as a malicious log might store certificates in the buckets that are not actually in the log. To prove that it wasn't doing this and satisfy integrity, the log would also need to store associated inclusion proofs (with respect to a given STH) for each certificate in a bucket as well. This augmented protocol would thus impose a significant overhead on the log in the form of both storing all the buckets (which, with the associated inclusion proofs, use $N \log(N)$ space where N is the number of certificates in the log), and periodically recomputing an inclusion proof for every certificate and updating the buckets accordingly.

In C3 protocols, however, the client is assumed to be interacting with a party (such as their browser vendor) that they trust to maintain a complete list of password hashes. A more realistic solution here would thus be to adopt the same threat model as for C3: a client would trust their browser vendor or some other third party to maintain a complete list of certificates, and would engage in the PSM protocol with them instead of the log. The fact that the client trusts this third party for integrity removes the need for inclusion proofs and thus means the client can receive a simple yes/no response (as in a traditional PSM protocol).

The protocol between the client and the log achieves k-anonymity, where k is the size of the bucket, only if the log is covert rather than malicious; this is due to the same potential for a fully malicious log to use unique STHs for inclusion proofs that we discussed for PIR. Similarly, the protocol between the client and a semi-trusted third party also achieves k-anonymity. In either case, performance is inversely proportional to privacy: revealing larger prefixes results in smaller buckets, which means lower bandwidth overhead but a smaller anonymity set. While the protocol achieves k-anonymity for a single query, it is unclear how privacy would degrade over repeated querying on potentially related certificates, such as certificates for domains that fit into a common category [15, 55]. This may be an interesting and fruitful area for future research.

4.3.3 Avoiding client querying

Any solution that requires communication between the client and log imposes some performance overhead. In theory, avoiding having any querying protocol at all would have a positive impact on bandwidth (as the client and log do not need to communicate) and on privacy, as the client cannot reveal any information about the certificate if it does not perform any queries about it. In practice, these "non-interactive" protocols are limited in the privacy they can achieve due to the same limitation discussed in Sections 4.3.2.2 and 4.3.2.3: inclusion proofs are tied not only to specific entry but also to a specific STH, which a malicious log may exploit. We discuss this limitation further in Section 6.3.

4.3.3.1 Local mirroring

If clients mirrored every log, then they could check for inclusion locally and would never need to issue any queries. To some extent, this solution resembles maintaining a certificate revocation list (CRL).

While this solution achieves integrity and perfect privacy, it clearly imposes a significant storage requirement on the client, as there are billions of certificates stored across all CT logs; even CRLs, which are much smaller, are considered impractical for this reason. The bandwidth overhead would be even higher, as they would need to periodically update the list of entries they maintain for each log. Nevertheless, mirrors do exist today (for individual logs) so such a solution would be relatively easy to deploy in the short term.

4.3.3.2 Embedding, fast and slow

RFC 6962-bis briefly suggests that inclusion proofs can be embedded in a certificate via a custom extension [42, Section 7.1.2]. This leaves open several questions, however, one of which is who should be responsible for embedding the inclusion proof. The natural answer would seem to be certificate authorities (CAs), given the inclusion proofs by the logs, as otherwise this would require a mandatory change in web servers.

The next question is when CAs should obtain these inclusion proofs. If they do so as soon as the certificate gets included in the log (*fast* embedding), this adds latency to the process of issuing a certificate, as CAs have to wait until it is included in the log. The effect on privacy is also negative, due to the fact that the STH used in the inclusion proof is likely to represent a small number of certificates even in the case that the log is honest; in the case that the log is malicious, it can guarantee that the STH is unique to this certificate. Thus, even in the honest case this means that the protocol achieves k-anonymity for a relatively small value of k.

If instead CAs wait for some period to obtain the inclusion proof (*slow* embedding), such as the maximum merge delay (MMD), the latency involved in issuing a certificate increases significantly. In terms of privacy, this achieves k-anonymity for a larger k. We discuss in Section 6.3 the respective values of k that could be expected for these solutions, and discuss in Section 6.1 the challenges that they present for deployability.

4.3.3.3 OCSP stapling

RFC 6962-bis also briefly mentions serving inclusion proofs using OCSP stapling [42, Section 7.1.1], as described in Section 3.2. Here, the web server periodically requests not only the certificate status from a CA but also inclusion proofs for its certificates, and serves this signed information to the client alongside the certificate and its embedded SCTs. To achieve integrity, this additional information needs to be required and signed by the CA for all certificates, or adversarial web servers can simply choose to not provide anything.

This approach seemingly avoids the privacy issues raised by embedding an inclusion proof directly into the certificate: the only way to link a client to their requested certificate is for the CA to maintain a mapping from certificates to the STHs used for their inclusion proofs, and a colluding log to maintain a mapping from IP addresses to the STHs queried for consistency proofs; the combination of these two maps links a client directly to a certificate. While this level of collusion may seem far-fetched, as we saw in Table 1 most logs are already run by CAs. If CAs and logs are colluding, this solution achieves k-anonymity, where k is the number of certificates that the CA gives out for a given STH. As discussed previously, this means that if we model CAs and logs as fully malicious we cannot guarantee any privacy, so must instead treat them as covert adversaries only.

The protocol is efficient for all parties. In terms of deployability, the main change required is at the CA rather than individual web servers, but it does assume that web servers are set up to use OCSP stapling. We discuss the limitation of relying on changes in web servers further in Section 6.4; briefly, Liu et al. found in 2015 that at most 5% of certificates were served by hosts that supported OCSP stapling [47], and Scheitle et al. found in 2018 that 0.01% of their observed connections

had SCTs served via OCSP stapling. Furthermore, in order for this proposal to achieve integrity, web servers *must* provide this information. This means certificates must suppport OCSP "Must-Staple"; i.e., a version of OCSP stapling in which responses are not considered valid if they do not contain the requested information. Chung et al. found in 2018 that only 0.02% of certificates supported this [19].

4.4 Proposals for reporting

We now describe protocols for the reporting phase, in which the goal is for an auditor to learn about entries that were not included in the log without being able to link a specific reported entry to a specific client.

4.4.1 Network-level anonymization

As in the querying phase, these proposals make no effort to hide the client's certificate, but instead may hide their identity from the auditor at the network layer.

4.4.1.1 Report directly

As in the querying phase, the simplest approach for reporting is to just have clients send certificates to the auditor directly. These can be certificates for which the client has already performed the querying phase and found to not be in the log, or they can be all or some subset of their certificates, in which case the auditor can then perform the querying phase itself (in this latter case, the client must trust the auditor for integrity). Nordberg et al. mention the possibility of having individual browsers send SCTs directly to trusted auditors [51, Section 8.3].

This solution requires very little overhead for an individual client, but clearly requires them to trust the auditor fully as it does not achieve any privacy. We discuss in Section 5.3 how such trusted auditors may already exist for certain clients.

4.4.1.2 Proxy/mixnet

As in the querying phase, a natural attempt to improve privacy would be to have clients route their traffic through a single proxy or a series of proxies, rather than contact the auditor directly. Again, clients could either send only certificates for which they have already performed the querying phase and found to not be in the log, or all or some subset of their certificates. As with the previous solution, clients who do the latter must trust the auditor to perform the querying itself. This latter approach is used by Dahlberg et al. in their CTor protocol for Tor clients [23].

This solution has largely the same properties as the one in the querying phase: it is performant and could be deployed in the near term by browsers that already act as a proxy for the related problem of interacting with Safe Browsing endpoints. It also protects the privacy of the client as long as there is sufficient traffic and as long as the proxy servers are not colluding with the auditor. Unlike in the querying phase, however, there may not be a high volume of traffic going to the auditor. In particular, the first scenario has clients send only certificates that they have already determined are not in the log. We expect logs to violate the promise implicit in their SCTs with very low frequency, given the serious consequences if they are caught doing this, so in this scenario clients might need to periodically send cover traffic, as they do in other mixnet solutions [17].

4.4.1.3 Web servers

Nordberg et al. [51, Section 8.1] proposed having a browser report the SCTs that are relevant to a website it is currently visiting. It is then the responsibility of the web server to send these SCTs to an auditor.

This method of reporting has the advantage that it is hard to disrupt without also disrupting web browsing, and it preserves privacy as the web server already knows the browser is visiting the site. It assumes, however, that an attacker cannot run a persistent man-in-the-middle attack, as they say that "clients will send the same SCTs and chains to a server multiple times with the assumption that any man-in-the-middle attack eventually will cease, and an honest server will eventually receive collected malicious SCTs and certificate chains." Even without such an attack, integrity also relies on the web server honestly reporting their SCTs to an auditor, as a malicious web server could just decide not to report and there would be no way to detect that they hadn't. More generally, it would require a change in a significant fraction of web servers in order to capture certificate misissuance at a broad scale. We discuss this limitation further in Section 6.4.

4.4.2 Privacy-preserving reporting

As with the analogous querying proposals, the next set of solutions allows the client's identity to be known to the auditor but uses cryptographic techniques to hide the specific certificate being reported (or, for the first proposal, to hide it in all but exceptional cases).

4.4.2.1 Private set membership (PSM)

Google recently proposed a privacy-focused solution for having browsers report certificates to auditors [24]. This protocol imagines that an auditor (in their case Google) acts as a mirror for all CT logs and thus maintains a comprehensive list of all certificates; in the maintenance of this list the auditor plays a role analogous to a Safe Browsing endpoint maintaining a blocklist. Before interacting with the auditor, clients first use a sampling strategy to decide on a subset of certificates that they might report. This step eliminates interaction with the auditor in a manner similar to the Bloom filter used in Safe Browsing; as mentioned in Section 3.1, using a Bloom filter is not practical here due to the significantly larger size of the list and the fact that clients are looking for a missing rather than a matching entry. The client and the auditor then engage in a PSM protocol for each of the certificates in this sample, analogous to the protocol used to check for compromised credentials (C3). If at the end of the protocol the client is convinced that its certificate is in the list held by the auditor, they do not continue further. If not, the client reports the certificate to the auditor in the clear; i.e., sends it to the auditor directly. The protocol thus consists of a PSM "querying" phase (but with the auditor rather than a log operator) followed by a direct reporting phase.

In terms of privacy, the PSM phase of this interaction achieves k-anonymity, where k is the size of a bucket. If the certificate is not on the auditor's list the client reveals it directly to them, however, which means the overall protocol achieves no privacy. The proposal suggests that this case is unlikely to happen in practice "as Google maintains a comprehensive copy of all valid SCTs" and that in these rare cases it is thus "appropriate to break anonymity." This assumes that the auditor honestly maintains a comprehensive list of SCTs/certificates, which means the protocol achieves k-anonymity only if the auditor is honest-but-curious. Finally, as in Section 4.3.2.3, the protocol achieves k-anonymity for a single query but it is unclear how privacy would degrade over repeated querying, and in particular if the sampling strategy used in this proposal would resolve the privacy issues present in Safe Browsing [30, 36]. Furthermore, performance is inversely proportional to privacy, as achieving k-anonymity for larger values of k requires sending larger buckets.

While an honest-but-curious adversary may make sense anyway when modelling the role of a browser vendor, the proposal does contain additional mitigations. For example, it suggests that clients "maintain a strict limit of 3 total SCT reports" sent to the auditor, which means that even a malicious auditor could only ever see three certificates per client.

4.4.2.2 Proof of non-inclusion

Rather than provide a non-included certificate directly to an auditor, a client could instead provide a zero-knowledge proof of its non-inclusion, as proposed by Eskandarian et al. [27]. In other words, the client can prove knowledge of an SCT such that the timestamp falls (strictly) between those of two adjacent log entries.

This protocol achieves provable zero knowledge, which means it is private, and does not require any trust assumptions. It does not fully satisfy integrity, however, as the auditor learns only about the existence of a non-included entry. If multiple clients provide such a report for the same log, the auditor does not know if these reports represent the same certificate or repeated misbehavior and is not in a position to follow up directly with the log and find out. The auditor thus has no actionable evidence that it can use to further investigate the potential misbehavior, which is a limitation we discuss further in Section 6.5. In terms of performance, producing a proof requires over five seconds on the client side (on a laptop) and the proof itself is over 333kB. As Eskandarian et al. argue, however, the reporting protocol is likely to be run infrequently.

5 Full Proposals

In this section, we describe *full* proposals for SCT auditing; i.e., proposals that combine a querying and a reporting protocol. As compared with the many individual components described in the previous section, there are relatively few of these. As we discuss further in Section 6.5, this is perhaps due to the fact that most existing research has focused on the querying phase and not the reporting phase. We also see how a proposal in one phase can have weaker privacy as a full proto-

col due to mismatched trust assumptions and privacy guarantees in the other phase.

5.1 Proofs of non-inclusion

Eskandarian et al. proposed a protocol [27] that combines fuzzy ranges (Section 4.3.2.1) with proofs of non-inclusion (Section 4.4.2.2). Briefly, this means the client (a browser) queries the log for a range that should include the certificate they are interested in and then checks if their certificate is indeed in this range. If not, they provide a zero-knowledge proof of its non-inclusion to some publicly accessible auditor.

The protocol as a whole (provably) preserves a client's privacy with respect to the auditor but achieves only k-anonymity with respect to the log. Furthermore, a client's queries to different logs likely reveal patterns that shrink the anonymity set over time. In terms of performance, there is a significant bandwidth overhead for logs (who would be contacted by every individual browser), and reporting a zero-knowledge proof imposes significant costs in terms of both computation and bandwidth on the client but is expected to happen infrequently (only in the case of non-inclusion). Finally, in terms of functionality the protocol is not actionable in that the auditor knows only that a log has misbehaved but not where or how many times. We discuss this final limitation in more detail in Section 6.5.

5.2 SCT Feedback

SCT Feedback [51, Section 8.1] combines direct querying, between an auditor and a log, with the reporting mechanism described in Section 4.4.1.3. Briefly, this means the client (a browser) reports the SCTs that are relevant to a website it is currently visiting. The web server then collects these SCTs and passes them on to some publicly accessible auditor, who in turns queries the log for their inclusion.

The protocol as a whole preserves the privacy of the client, who reveals their SCTs only to a website they are already visiting. As discussed above, however, the protocol has the significant downside that it relies on websites to report certificates to an auditor. This means that clients may not be protected against malicious websites, and more generally that a significant fraction of web servers would be required to run the protocol in order to have a reasonable chance of identifying misbehavior. We discuss this limitation further in Section 6.4.

5.3 Opt-in SCT auditing

One active proposal by Google allows clients to *opt in* to SCT auditing [64]; in fact, this has been deployed in Chrome as of March 2021 [54]. This combines direct querying (Section 4.3.1.1), between the auditor and the log, with direct reporting (Section 4.4.1.1), from the client to the auditor. The auditor in this case is Google.

While direct reporting raises obvious privacy concerns, the clients who report SCTs already share their browsing history with Google by performing extended reporting as part of Safe Browsing. Furthermore, the proposal states that "Third-party logs don't receive any information about Chrome users' browsing history because we query Google-operated mirrors of CT data instead of querying the logs directly." Thus, as the auditor sits within the same trust boundary as the log, no information about the client's data is revealed to anyone except the auditor.

6 Discussion

In this section, we discuss the limitations of existing solutions, in terms of the assumptions they make and the requirements they impose. The particular issues we highlight are: (1) certificate issuance latency, (2) client constraints, (3) privacy, (4) significant changes to the Web infrastructure, and (5) reporting misbehavior to an auditor. We use these limitations to define a set of constraints for, and thus a clear definition of, the problem of performing SCT auditing in the existing Certificate Transparency ecosystem.

6.1 Issuance latency

Logs may take up to 24 hours to include a certificate in a log, given the current MMD for Certificate Transparency, and as reported by Gustafsson et al. [31] it can take logs up to 12 hours to publish a new STH (which signals the inclusion of a new batch of certificates).

Web hosting providers currently promise significantly faster issuance rates, ranging from minutes to several hours [5, 12, 20]. Furthermore, issuing a certificate quickly is important to these businesses, with "slow certificate deployment [leaving] customers with an unsatisfactory experience" [38]. Shortening the MMD would address this tension, but would place a significantly higher burden on log operators and—as we discuss below—could be harmful to privacy. This higher

burden would raise the barrier to entry for running a log, which would ensure that only large institutions would be able to act as log operators (as is already largely the case today).

It is thus infeasible for certificate authorities to wait for log inclusion before issuing a certificate, which means **inclusion proofs cannot be embedded into certificates**, as is required in the fast and slow embedding proposals (Section 4.3.3.2). Furthermore, the current most widely deployed log implementation, Trillian [11], has no distinction between sequencing an entry and including it. This also means that at least today, **sequence numbers cannot be embedded into certificates**, as is required in the fuzzy ranges (Section 4.3.2.1) and PIR (Section 4.3.2.2) querying proposals.

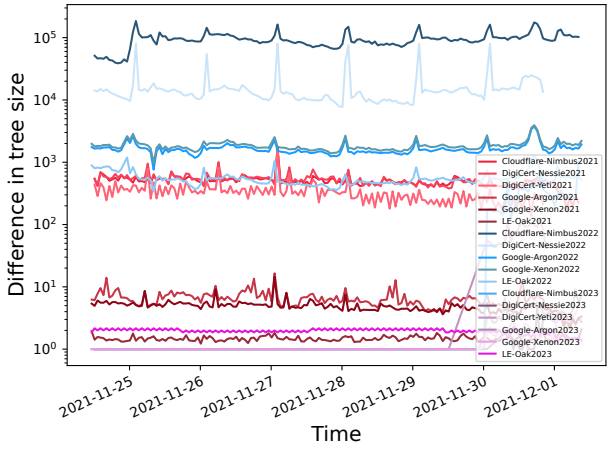

Fig. 1. The difference in tree sizes for time-sharded CT logs, averaged hourly over a week (November 24–December 1, 2021) and at log scale.

6.2 Client constraints

Clients are run on commodity laptops and, in an increasing majority of cases,[2] on mobile devices. These are computing environments that are limited in terms of the bandwidth, storage, and computational power available to them. Collectively, today's active time-sharded CT logs contain 5.8 billion certificates. Even if a client stored only this many hashes (which ignores storing all internal log hashes or the actual underlying data), this would amount to 185.6GB of data, which means that **clients cannot act as mirrors**, as is required in the local mirroring querying proposal (Section 4.3.3.1).

6.3 Privacy

Many querying proposals focus on privacy-preserving ways to retrieve inclusion proofs, which are inherently tied to the certificates for which they prove inclusion. As discussed first in Section 4.3.2.2, however, an inclusion proof is also tied to the STH with respect to which it proves inclusion, and for a client to fully convince themselves that an entry really is in the log they need to both (1) verify the inclusion proof for that entry with respect to its associated STH and (2) verify a consistency proof between that STH and one it currently holds and believes to be valid. If an STH were used to form only one or a small number of inclusion proofs, querying a log for a consistency proof with respect to this STH would reveal significant information about the certificate. As discussed earlier, it is possible for a malicious log to ensure that each inclusion proof is formed with respect to a unique STH, thus enabling this attack.

In the proposals that rely on network-level anonymization, this attack is not effective as clients can use the same anonymous communication tools to retrieve consistency proofs as they do to retrieve inclusion proofs. It is not clear, however, how to modify the PIR (Section 4.3.2.2), PSM with the log (Section 4.3.2.3), embedding (Section 4.3.3.2), and OCSP stapling (Section 4.3.3.3) querying proposals. For these proposals, we must thus assume weaker *covert adversaries* [13], who may deviate from the protocol but "do not wish to be 'caught' doing so." Given that monitors and auditors can both catch this form of log misbehavior, we believe this is a reasonable way to model adversaries in CT.

Even honest-but-curious logs may still unintentionally use a single STH to form only a small set of inclusion proofs. To understand the extent to which this might be a problem today, we sought to identify how many certificates are represented by a given STH. In particular, we performed two experiments as follows.

1. We queried each time-sharded log every 30 seconds for a week, starting on November 24, 2021, and included results for all logs for which we observed at least 95% availability. This meant excluding only the TrustAsia logs.
2. We queried each time-sharded log, this time once every second for a minute, for a total of four minutes spread between November 22nd and 25th, 2021.

2 https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet

Log name	Log shard 2021	2022	2023
Cloudflare Nimbus	1	1	1
DigiCert Nessie	1	1	1
DigiCert Yeti	1	1	1
Google Argon	22.8	60	3
Google Xenon	25.5	60	3
Let's Encrypt Oak	4	43	2

Table 3. The number of unique STHs seen per minute for each log shard, averaged across four runs of one query per second. The runs were conducted between November 22nd and 25th, 2021.

The first experiment is designed to evaluate the case in which a log server issues an inclusion proof with respect to the latest STH at the time a certificate is included, as in the fast embedding approach (Section 4.3.3.2). This achieves k-anonymity, where k is the difference in tree size between that STH and the previous one. As suggested by the results in Figure 1, there are many logs for which each STH represents a fairly small number of entries: the 2022 log shards saw differences in tree sizes in the hundreds or thousands, but for the 2023 log shards the average difference was 23.6 and the median was 1.75. (Across all logs and shards, the average difference was 8093 but the median was 385.) For the 2023 log shards, even forming inclusion proofs once per day would thus provide an anonymity set of only 100-200 certificates. Thus, **inclusion proofs cannot be embedded into certificates**, as is suggested in both the fast and slow embedding querying proposals (Section 4.3.3.2).

The second experiment is designed to evaluate the case in which a CA queries a log server for an inclusion proof at a given point in time, and again it returns an inclusion proof formed with respect to the latest STH. As suggested by Table 3, some logs form new STHs significantly faster than once per minute, and potentially even faster than once per second. As expected given that we performed this experiment in November 2021 (in which the vast majority of issued certificates would be expiring in 2022), this was especially true for the 2022 log shards. If we take the Let's Encrypt Oak 2022 log shard as an example, in which Figure 1 shows a difference in tree size of hundreds between STHs sampled at a 30-second interval, this suggests that the anonymity set for an STH retrieved at any given second would be at most tens of certificates.

Finally, if individual browser instances queried the log directly to retrieve inclusion proofs then even honest-but-curious logs would be able to learn the certificates seen by those clients. Thus, **browsers cannot directly query logs**, as is suggested by the first option in the direct querying proposal (Section 4.3.1.1).

6.4 Significant Web changes

Previous research has already shown that web servers are typically slow to adopt new protocols, such as HTTPS [28, 50, 59], OCSP stapling [19], newer versions of TLS [33, 37], and HSTS and HPKP [39]. Furthermore, web servers are slow to patch even significant and highly publicized security vulnerabilities, with a long tail never performing any patching at all [25, 45, 65].

More specifically to CT, Gasser et al. explored adoption of the gossiping endpoints proposed by SCT Feedback and found that at most 0.015% of domains had made them available [29]. Even if this number were higher, even in a longer timescale it is not feasible to imagine every single web server adopting a new protocol, so **security cannot rely on mandatory changes implemented in web servers**, as is required in the OCSP stapling querying proposal (Section 4.3.3.3) and the SCT Feedback reporting proposal (Section 4.4.1.3).

6.5 Reporting

The decisions that have thus far been made to remove logs from the CT ecosystem [52, 53, 60] have required significant discussion [26, 32, 61], including a "post-mortem" from the log operators identifying the conditions that led to their (unintended) misbehavior and considering how those conditions could be prevented in the future. As such, **auditors require actionable and concrete evidence of a log's misbehavior**, as opposed to just learning about its existence as in the ZKP of non-inclusion proposal (Section 4.4.2.2).

More generally, as we saw in Section 4, much of the research addressing the problem of SCT auditing has focused on the querying phase, with significantly fewer proposals for the reporting phase. While existing querying proposals could perhaps be extended using existing reporting proposals, we already saw in Section 5 that the security of the full proposal is only as secure as the weaker of the two phases. Furthermore, if we look at the viable proposals remaining in Table 2 we can see there are few natural pairings. The proposal for reporting directly requires full trust in the auditor, and the C3-PSM proposal achieves privacy only with respect to

an honest-but-curious auditor (and in that case achieves k-anonymity rather than full privacy).

This leaves proxying as the only option, which pairs naturally with the same proposal in the querying phase. Indeed, the fact that Brave and Apple already proxy queries to Safe Browsing endpoints suggests that this could be a viable solution in the near term for these browsers, in terms of acting as a proxy for both querying logs and reporting to auditors. Chrome users, however, comprise 65% of all browser users.[3] Given that Google acts as a CT log operator and is proposing to act as an auditor, it would thus need another entity to act as a proxy, which is a significant undertaking. Moreover, if other browser vendors chose to act as auditors too (which would of course be preferable to having Google be the only auditor), they would also require third-party proxying solutions.

6.6 Summary of constraints

To summarize the constraints we have identified above, the main limitation of many of the solutions we presented is that they address only the querying phase and have no solution for reporting. In particular, for almost all discussed solutions for the querying phase, it is an individual browser instance that learns whether or not a certificate is in the log, but there are both privacy and feasibility questions in terms of how individual browsers could then report log misbehavior to the broader CT ecosystem in a way that is actionable. Furthermore, as discussed in Section 6.5, most existing solutions for querying are not compatible in terms of their threat model with the existing solutions for reporting. Thus, the main guideline for future proposals is to consider both the querying and reporting phases and ensure that privacy is preserved across both types of interactions.

In terms of privacy, our main observation in Section 6.3 is that even for honest logs it is often possible for an STH itself to contain significant information in terms of the number of log entries it represents. It is thus important for querying proposals to consider not only how clients can preserve their privacy in obtaining inclusion proofs, but also how they can do so in obtaining consistency proofs. An alternative is to impose a rate limit on the number of STHs that can be produced within a given period of time, as is suggested in RFC 6962-bis [42, Section 4.10]. This requires substantial changes to log operation, however, which hinders near-term deployability. Furthermore, while limiting the number of available STHs would improve privacy with respect to honest-but-curious logs, malicious logs could still use specific STHs in the inclusion proofs for specific certificates to perform fingerprinting.

7 Conclusions

In this paper, we have systematically explored the techniques that have been proposed thus far for the problem of performing SCT auditing — which is central to the security guarantees that Certificate Transparency brings to the HTTPS ecosystem — in a way that preserves the privacy of individual users. In doing so, we have identified proposals from both academia and industry, many of which exist in quite different forms; e.g., posts on mailing lists, academic papers, and readmes in GitHub repositories. Despite these differences, we have brought these proposals together and explored them under a unified evaluation framework that considers their privacy, integrity, performance overheads, trust assumptions, and near-term deployability.

In doing so, we have identified several key limitations shared by multiple proposals, in terms of (1) the increased latency they cause for certificate issuance; (2) the excessive performance overheads they impose on clients; (3) the limited privacy they achieve due to their ability to privately retrieve inclusion proofs but not consistency proofs; (4) their need for change in a significant majority of web servers in order to achieve integrity; and (5) their lack of a proposed reporting component. In highlighting these limitations, our goal is to create a set of constraints that we hope serves as a useful guide to researchers interested in this problem. To further this goal, we have also highlighted both the similarities and differences with other problems associated with sensitive browsing-related information, most of which have been explored and evaluated more thoroughly than SCT auditing has to date.

Acknowledgements

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

3 https://gs.statcounter.com/browser-market-share

References

[1] Certificate Transparency Log Policy. https://chromium.github.io/ct-policy/log_policy.html.

[2] Crlsets. https://dev.chromium.org/Home/chromium-security/crlsets.

[3] Firefox Monitor. https://support.mozilla.org/en-US/kb/firefox-monitor.

[4] Have I Been Pwned. https://haveibeenpwned.com.

[5] How long will it take to issue my certificate? https://www.godaddy.com/help/how-long-will-it-take-to-issue-my-certificate-858.

[6] Mailing Lists. https://sites.google.com/site/certificatetransparency/mailing-lists.

[7] Merkle Town. https://ct.cloudflare.com/.

[8] Monitors: Certificate transparency. https://certificate.transparency.dev/monitors/.

[9] Safe Browsing APIs (v4). https://developers.google.com/safe-browsing/v4.

[10] Services We Proxy Through Brave Servers. https://github.com/brave/brave-browser/wiki/Deviations-from-Chromium-(features-we-disable-or-remove)#services-we-proxy-through-brave-servers.

[11] Trillian: General Transparency. http://github.com/google/trillian.

[12] Troubleshooting SSL certificates. https://cloud.google.com/load-balancing/docs/ssl-certificates/troubleshooting.

[13] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Proceedings of the Theory of Cryptography Conference (TCC)*, pages 137–156, 2007.

[14] A. Ayer. How will Certificate Transparency Logs be Audited in Practice?, Jan. 2018. https://www.agwa.name/blog/post/how_will_certificate_transparency_logs_be_audited_in_practice.

[15] S. Bird, I. Segall, and M. Lopatka. Why we still can't browse in peace: on the uniqueness and reidentifiability of web browsing histories. In *Proceedings of the 16th Symposium on Usable Privacy and Security (SOUPS)*, 2020.

[16] T. Broach. Apple redirects Google Safe Browsing traffic through its own proxy servers to prevent disclosing users' IP addresses to Google in iOS 14.5, Feb. 2021. https://the8-bit.com/apple-proxies-google-safe-browsing-privacy/.

[17] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: website fingerprinting attacks and defenses. In *Proceedings of ACM CCS*, 2012.

[18] L. Chuat, A. Abdou, R. Sasse, C. Sprenger, D. Basin, and A. Perrig. Sok: Delegation and revocation, the missing links in the Web's chain of trust. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, 2020.

[19] T. Chung, J. Lok, B. Chandrasekaran, D. Choffnes, D. Levin, B. M. Maggs, A. E. Mislove, J. P. Rula, N. Sullivan, and C. Wilson. Is the web ready for OCSP Must-Staple? In *Proceedings of the Internet Measurement Conference (IMC)*, pages 105–118, 2018.

[20] T. Cignetti. Easier certificate validation using DNS with AWS certificate manager, Nov. 2017. https://aws.amazon.com/blogs/security/easier-certificate-validation-using-dns-with-aws-certificate-manager/.

[21] J. Clark and P. C. van Oorschot. Sok: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2013.

[22] H. Corrigan-Gibbs and D. Kogan. Private information retrieval with sublinear online time. In *Proceedings of Eurocrypt 2020*, 2020.

[23] R. Dahlberg, T. Pulls, T. Ritter, and P. Syverson. Privacy-preserving & incrementally-deployable support for Certificate Transparency in Tor. *Proceedings on Privacy Enhancing Technologies*, 2021(2):194–213, 2021.

[24] J. DeBlasio. Opt-out SCT Auditing in Chrome. https://docs.google.com/document/d/16G-Q7iN3kB46GSW5b-sfH5MO3nKSYyEb77YsM7TMZGE.

[25] Z. Durumeric, F. Li, J. Kasten, J. Amann, J. Beekman, M. Payer, N. Weaver, D. Adrian, V. Paxson, M. Bailey, and J. A. Halderman. The matter of Heartbleed. In *Proceedings of the Internet Measurement Conference (IMC)*, 2014.

[26] G. Edgecombe. Wosign log failure to incorporate entry within the MMD. Certificate Transparency Policy mailing list, Dec. 2017. https://groups.google.com/a/chromium.org/g/ct-policy/c/-eV4Xe8toVk/m/pC5gSjJKCwAJ.

[27] S. Eskandarian, E. Messeri, J. Bonneau, and D. Boneh. Certificate Transparency with privacy. *Proceedings on Privacy Enhancing Technologies*, 2017(4):232–247, 2017.

[28] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel, and P. Tabriz. Measuring HTTPS adoption on the web. In *Proceedings of the 26th USENIX Security Symposium*, 2017.

[29] O. Gasser, B. Hof, M. Helm, M. Korczynski, R. Holz, and G. Carle. In log we trust: Revealing poor security practices with Certificate Transparency logs and internet measurements. In *Proceedings of the International Conference on Passive and Active Measurement (PAM)*, 2018.

[30] T. Gerbet, A. Kumar, and C. Lauradoux. A Privacy Analysis of Google and Yandex Safe Browsing. INRIA Research Report RR-8686, 2015. https://hal.inria.fr/hal-01120186v4/document.

[31] J. Gustafsson, G. Overier, M. Arlitt, and N. Carlsson. A first look at the CT landscape: Certificate Transparency logs in practice. In *Proceedings of the Passive and Active Measurement Conference (PAM)*, 2017.

[32] P. Hadfield. Google Aviator incident under investigation. Certificate Transparency Policy mailing list, Oct. 2016. https://groups.google.com/a/chromium.org/g/ct-policy/c/ZZf3iryLgCo/m/mi-4ViMiCAAJ.

[33] R. Holz, J. Hiller, J. Amann, A. Razaghpanah, T. Jost, N. Vallina-Rodriguez, and O. Hohlfeld. Tracking the deployment of TLS 1.3 on the Web. *ACM SIGCOMM Computer Communication Review*, July 2020.

[34] J. Jones. Introducing CRLite: All of the Web PKI's revocations, compressed, Jan. 2020. https://blog.mozilla.org/security/2020/01/09/crlite-part-1-all-web-pki-revocations-compressed/.

[35] D. Kales, O. Omolola, and S. Ramacher. Revisiting user privacy for Certificate Transparency. In *Proceedings of the 4th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[36] D. Kogan and H. Corrigan-Gibbs. Private blocklist lookups with Checklist. In *USENIX Security Symposium*, 2021.

[37] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero. Coming of age: A longitudinal study of TLS deployment. In *Proceedings of the Internet Measurement Conference (IMC)*, 2018.

[38] D. Kozlov. Announcing Cloudflare for SaaS for everyone, Apr. 2021. https://blog.cloudflare.com/cloudflare-for-saas/.

[39] M. Kranch and J. Bonneau. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning. In *Proceedings of NDSS 2015*, 2015.

[40] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. Crlite: A scalable system for pushing all TLS revocations to all browsers. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2017.

[41] B. Laurie, A. Langley, and E. Kasper. Certificate Transparency, 2013. https://tools.ietf.org/html/rfc6962.

[42] B. Laurie, A. Langley, E. Kasper, E. Messeri, and R. Stradling. Certificate Transparency version 2.0, 2019. https://tools.ietf.org/html/draft-ietf-trans-rfc6962-bis-34.

[43] B. Laurie, P. Phaneuf, and A. Eijdenberg. Certificate Transparency over DNS. https://github.com/google/certificate-transparency-rfcs/blob/master/dns/draft-ct-over-dns.md.

[44] 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 *Proceedings of ACM CCS*, 2019.

[45] F. Li, Z. Durumeric, J. Czyz, M. Karami, M. Bailey, D. McCoy, S. Savage, and V. Paxson. You've got vulnerability: exploring effective vulnerability nodifications. In *Proceedings of the 25th USENIX Security Symposium*, 2016.

[46] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart. Protocols for checking compromised credentials. In *Proceedings of ACM CCS*, 2019.

[47] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, and C. Wilson. An end-to-end measurement of certificate revocation in the Web's PKI. In *Proceedings of the Internet Measurement Conference (IMC)*, 2015.

[48] W. Lueks and I. Goldberg. Sublinear scaling for multi-client private information retrieval. In *Proceedings of the 19th Conference on Financial Cryptography and Data Security (FC)*, 2015.

[49] E. Messeri. Privacy implications of Certificate Transparency's DNS-based protocol, 2017. https://docs.google.com/document/d/1DY2OsrSJDzlRHY68EX1OwQ3sBIbvMrapQxvANrOE8zM.

[50] A. Mirian, C. Thompson, S. Savage, G. M. Voelker, and A. P. Felt. HTTPS adoption in the longtail, 2018. https://research.google/pubs/pub49037/.

[51] L. Nordberg, D. Gillmor, and T. Ritter. Gossiping in CT, 2018. https://tools.ietf.org/html/draft-ietf-trans-gossip-05.

[52] D. O'Brien. Upcoming CT Log Removal: StartCom. Certificate Transparency Policy mailing list, Jan. 2018. https://groups.google.com/a/chromium.org/g/ct-policy/c/W1Ty2gO0JNA/m/ZbQxlgRZAQAJ.

[53] D. O'Brien. Upcoming CT Log Removal: WoSign. Certificate Transparency Policy mailing list, Jan. 2018. https://groups.google.com/a/chromium.org/g/ct-policy/c/UcCqlxuz_1c/m/Mf_939xYAQAJ.

[54] D. O'Brien. Chrome CT 2021 Plans, Feb. 2021. https://groups.google.com/a/chromium.org/g/ct-policy/c/4puGir9pNFA/m/1caF3ilrBQAJ.

[55] L. Olejnik, C. Castelluccia, and A. Janc. Why Johnny can't browse in peace: on the uniqueness of web browsing history patterns. In *5th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETS)*, 2012.

[56] N. Parker, V. Khaneja, E. Mill, and K. C. Nair. Enhanced Safe Browsing Protection now available in Chrome, May 2020. https://security.googleblog.com/2020/05/enhanced-safe-browsing-protection-now.html.

[57] A. P. Security. Password Monitoring, Feb. 2021. https://support.apple.com/en-gb/guide/security/sec78e79fc3b/1/web/1.

[58] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In *International Workshop on Privacy Enhancing Technologies*, pages 41–53, 2002.

[59] S. Singanamalla, E. H. B. Jang, R. Anderson, T. Kohno, and K. Heimerl. Accept the risk and continue: measuring the long tail of government HTTPS adoption. In *Proceedings of the Internet Measurement Conference (IMC)*, 2020.

[60] R. Sleevi. Upcoming CT Log Shutdown: Aviator. Certificate Transparency Policy mailing list, Nov. 2016. https://groups.google.com/a/chromium.org/g/ct-policy/c/u87C79AY-E8/m/k-4sbTguCgAJ.

[61] R. Sleevi. StartCom Log misbehaving: Failure to incorporate SCTs. Certificate Transparency Policy mailing list, Dec. 2017. https://groups.google.com/a/chromium.org/g/ct-policy/c/92HIh2vG6GA/m/hBEHxcpoCgAJ.

[62] R. Sleevi. Unwind the CT DNS-based proof inclusion experiment, May 2019. https://bugs.chromium.org/p/chromium/issues/detail?id=506227#c59.

[63] R. Sleevi and E. Messeri. Certificate Transparency in Chrome: Monitoring CT logs consistency, 2015. https://docs.google.com/document/d/1FP5J5Sfsg0OR9P4YT0q1dM02iavhi8ix1mZIZe_z-ls/edit.

[64] E. Stark and C. Thompson. Opt-in SCT auditing, 2020. https://docs.google.com/document/d/1G1Jy8LJgSqJ-B673GnTYIG4b7XRw2ZLtvvSlrqFcl4A.

[65] B. Stock, G. Pellegrino, C. Rossow, M. Johns, and M. Backes. Hey, you have a problem: on the feasibility of large-scale web vulnerability notification. In *Proceedings of the 25th USENIX Security Symposium*, 2016.

[66] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh, and E. Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security Symposium*, 2019.

[67] J. Wolff. How a 2011 hack you've never heard of changed the Internet's infrastructure, Dec. 2016. https://slate.com/technology/2016/12/how-the-2011-hack-of-diginotar-changed-the-internets-infrastructure.html.