Private SCT Auditing, Revisited

Lena Heimberger*
Graz University of Technology
lena.heimberger@tugraz.at

Christopher Patton
Cloudflare Research
cpatton@cloudflare.com

Bas Westerbaan
Cloudflare Research
bas@cloudflare.com

ABSTRACT

In order for a client to securely connect to a server on the web, the client must trust certificate authorities (CAs) only to issue certificates to the legitimate operator of the server. If a certificate is mis-issued, it is possible for an attacker to impersonate the server to the client. The goal of Certificate Transparency (CT) is to log every certificate issued in a manner that allows anyone to audit the logs for mis-issuance. A client can even audit a CT log itself, but this would leak sensitive browsing data to the log operator. As a result, client-side audits are rare in practice.

In this work, we revisit private CT auditing from a real-world perspective. Our study is motivated by recent changes to the CT ecosystem and advancements in Private Information Retrieval (PIR). First, we find that checking for inclusion of Signed Certificate Timestamps (SCTs) in a log — the audit performed by clients — is now possible with PIR in under a second and under 100kb of communication with minor adjustments to the protocol that have been proposed previously. Our results also show how to scale audits by using existing batching techniques and the algebraic structure of the PIR protocols, in particular to obtain certificate hashes included in the log.

Since PIR protocols are more performant with smaller databases, we also suggest a number of strategies to lower the size of the SCT database for audits. Our key observation is that the web will likely transition to a new model for certificate issuance. While this transition is primarily motivated by the need to adapt the PKI to larger, post-quantum signature schemes, it also removes the need for SCT audits in most cases. We present the first estimates of how this transition may impact SCT auditing, based on data gathered from public CT logs. We find that large scale deployment of the new issuance model may reduce the number of SCT audits needed by a factor of 1,000, making PIR-based auditing practical to deploy.

1 INTRODUCTION

Certificates are used to add authenticity to the web by binding a domain name to a public key. Modern browsers only trust certificates issued by selected Certification Authorities (CAs) with a history of operating faithfully. To help mitigate mis-issuance, major browsers require every certificate to be logged before being issued by at least two public *Certificate Transparency* (CT) logs [1, 2, 32]. This is intend to add accountability to the certificate issuance process. For example, it allows a website owner to check if and when a certificate was issued for their domain name without their approval.

Upon accepting a certificate for publication, the log operator returns a *Signed Certificate Timestamp* (SCT), which the server can then present alongside its certificate to the client. The SCT acts as a kind of "promise" to the client that the certificate will eventually be included in the log. Logs are required to furnish proof of inclusion to auditors within the *Maximum Merge Delay* (MMD) window of 24 hours: if a log misses this deadline enough times, then it will eventually be distrusted by browsers. As of November 2024, six CT logs have sufficient uptime to be trusted by major browsers.[1]

Without SCT auditing, browsers would have to trust $n-1$ out of n operators to log all certificates they promise to log. This is because browsers only check for the presence of two SCTs; compromising two logs is sufficient to convince a client to trust an unlogged certificate. This is especially problematic given that many logs are operated by CAs and our goal is to detect mis-issuance by CAs. In addition, ongoing efforts to make CT logs easier to operate [18] may lead to more logs being created, which may in turn make it easier for the attacker to run logs themselves.

1.1 SCT Auditing

To audit an SCT, a client would wait until after the MMD has expired, then query the log to make sure the certificate was included as promised. However, directly checking inclusion with the log would be problematic, since the certificate includes the domain name visited by the client. Today SCT auditing is done in a very limited fashion by some browsers on a small fraction of certificates using differentially private methods [21]. A natural technique for fully private auditing is *Private Information Retrieval* (PIR) [15], which allows users to query a database privately without the database operator learning the query or the response. Earlier analysis by Meiklejohn et al. [41] of PIR for SCT auditing concludes this is impractical.

1.2 This paper

There have been two recent developments in the web public-key infrastructure that warrant a new look at PIR to tackle the SCT auditing problem:

First, with the imminent adoption of the new *static CT/Sunlight* API [18] and the updates to CT logs, certificates will now encode the sequence number at which the certificate appears in the log. This results in a dense database where the keys are consecutive sequence numbers, which facilitates much faster and more PIR-friendly queries. Previously, certificates had to be queried by hash [27].

The second is a side-effect of the ongoing transition to post-quantum cryptography. Post-quantum signature schemes are significantly larger than currently deployed schemes, which means we will have to re-engineer certificate issuance to accommodate them. There are currently a number of proposals under consideration to rework certificate issuance — most notably *Starlit Jellyfish* [30] and *Merkle Tree Certificates* (MTCs) [11]. These proposals adopt a *slow*

*Work partially performed while interning at Cloudflare Research.

[1] Chrome requires a log uptime of 99%. This is fulfilled by Cloudflare, Google, Let's Encrypt, Sectigo, TrustAsia and Digicert. https://googlechrome.github.io/CertificateTransparency/log_list.html

issuance approach for the bulk of certificates instead of the traditional X.509 certificates in use today. In the slow issuance approach, instead of presenting SCTs, the authenticating party presents a Merkle tree inclusion proof to a tree root of the log. To make sense of the authentication path, these distinguished Merkle tree roots have to be propagated out of band to the user by a trusted party, e.g., the browser vendor. This propagation takes time, and thus these slow-issuance certificates cannot be issued immediately. In addition, not every user will be able to update immediately. These cases are inevitable, hence it will always be necessary to fallback to to traditional X.509 certificates and SCTs. There are still ways reduce the number of signatures in the fallback path [29], but the SCTs are unavoidable.

In the remainder of this paper, we will refer to X.509 certificates with SCTs as *fast-issuance* certificates; and we will use the term *slow-issuance* for proposals like MTCs for which SCTs are not required. We remark that, as of this writing, slow-issuance certificates have not been deployed. However, it is likely that some slow-issuance mechanism will be needed to support the post-quantum transition.

1.2.1 Contributions. Taken together, these changes to the web public-key infrastructure warrant another look at the design space for private SCT auditing. In this work, we reconsider the practicality of PIR for this application in light of these changes, as well as advancements in the underlying cryptography. Our contributions are as follows:

- We concretize SCT auditing in the context of slow-issuance certificates, and define three scenarios where fast-issuance certificates are still necessary. Using publicly available data from Certificate Transparency logs (CT logs), we derive a loose upper bound of the probability that a random website falls back to a fast-issuance certificate. This gives us a basis for estimating the load a PIR-based auditing service would need to cope with, as well as the cost for clients using the service.
- Then, we revisit SCT auditing and define concrete requirements for PIR schemes. After giving an overview of the state of the art, we narrow the PIR schemes down to three concrete proposals.
- For each proposal, we describe how to restructure the database and/or change CT logs to make these schemes practical for deployments. We then benchmark implementations of these candidate PIR protocols against the best-known current solution using Bloom filters [27]. Our approach for a single audit are more efficient in terms of computation and communication.
- Finally, we discuss how to batch the audit to perform multiple audits at the cost of a single audit by leveraging the algebraic structure of the PIR schemes.

2 BACKGROUND AND RELATED WORK

When a user browses a website, they verify the identity based on the X.509 certificate they receive during the TLS handshake. If the provided X.509 certificate was issued by a trusted Certification Authority (CA), the servers identity is deemed secure. It is the CAs responsibility to diligently check the information presented when issuing certificates to ensure only the legitimate owner of a domain can obtain a valid certificate for the domain. Should the CA issue a certificate to someone else, either because the CA is malicious or makes a mistake, the certificate can be used by an attacker to intercept, read, and alter the traffic without the user having any capability to detect the attack. If the mis-issuance gets detected, then the CA would then be distrusted by browsers. This scenario is not hypothetical: in 2011, the CA DigiNotar wrongfully issued over 500 certificates, forcing major browsers to remove the CA from their root store programs.[2]

2.1 Certificate Transparency

In order to make such incidents easier to detect, Certificate Transparency (CT) requires all certificates to be publicly logged so anyone can check for the presence of potentially mis-issued certificates. Domain owners *monitor* CT logs to ensure the logs do not contain mis-issued certificates for their domains. When wrongful issuance is detected, the domain owner can flag the CA for misbehavior and revoke the certificate [33]. During the certificate issuance process, all certificates must be sent to at least two logs. Each log is required to check the certificate and return a Signed Certificate Timestamp (SCT) as proof they have seen the certificate and promise to ingest it into the log within the Maximum Merge Delay (MMD). The SCT is then returned to the CA, which includes the SCTs in the final certificate. Users verify these SCTs during the TLS handshake. They may also audit the logs directly to ensure the promise of inclusion has been fulfilled.

Log Structure and Proofs. CT logs are stored as append-only Merkle trees. Each hashed certificate forms a *leaf node* of the tree. To compute the tree, the leaf nodes are paired and hashed together to generate the next level of the Merkle tree. This process is repeated until a final root hash emerges. The Merkle tree log can provide two proofs: An inclusion proof that proves a specific node is included in the tree by providing all intermediary nodes to the root node, and a consistency proof that proves the append-only Merkle tree is consistent with a verifiable, known root by providing a path to the new, updated root. In the context of CT, the signed roots that are distributed are called *Signed Tree Heads* (STH) [33].

Current Deployment. At the time of writing, most CT logs use an implementation called Trillian [48]. In October 2024, Tessera [17] was announced as Trillians successor. Amongst a variety of new features, Tessera assigns sequence numbers [38] to data entries that are processed. As a result, each certificate gets a unique sequence number. Additionally, Tessera supports the StaticCT API [18], a redesign of the log that ensures there are no unsequenced SCTs, which is important in case of an incident where SCTs are issued but the corresponding certificates are not included in the log. StaticCT now allows immediate sequencing, where a sequence number is assigned to the certificate immediately, and the records and hashes are stored in *tiles* that group several entries at a level of the tree.

Tiled Logs. Tiles split the data structure of the Merkle tree into fixed-size chunks in Tessera and StaticCT. Each tile contains exactly 256 hashes. The tiles are addressed by the tuple (l, i) consisting of the Merkle tree level l and the i^{th} index of the tile at level l. Tiles

[2] https://en.wikipedia.org/w/index.php?title=DigiNotar&oldid=1162693847

at level 0 contain leaf hashes. The levels above contain the hashes of the full tiles below [18]. Tiled logs are significantly cheaper to operate [16, 28] as the tiles can be cached and the inclusion proof is computed by the client. Previously, the server would retrieve all hashes in the path of a certificate, compute an inclusion proof and return it to the user. Tiled logs use the fact that tiles are immutable once complete and shift the computation of the proof to the user. Now, the server only retrieves the tiles with the intermediary nodes in the tree, which can be easily cached, and returns them to the user, who computes the inclusion proof themselves.

2.2 Private SCT Auditing

In the current ecosystem, an adversary controlling two CT logs and a CA may serve a malicious certificate without immediate detection by the browser. Therefore, auditing that a received certificate has indeed been included in the log is important to detect an attack and flag misbehavior, especially in a scenario where the malicious certificate is only delivered to a small group of selected users. Meiklejohn et al.s SoK on SCT auditing [41] splits audit process into two parts: The *reporting* phase, where the SCT is collected and reported to some auditing infrastructure, and the *querying* phase that can be carried out by anyone who receives the SCT. While reporting an SCT is the duty of the user that received the certificate, auditing that the log is append-only and globally consistent can be done by anyone, most likely the browsers backend auditing infrastructure. This separation is important as reporting SCTs directly to the log leaks the sensitive browsing history of the user to the log.

For the audit, Meiklejohn et al.'s SoK [41] requires both a proof of consistency between a known Signed Tree Head (STH) and the current STH of the Merkle tree and a proof of inclusion from the known tree head to the STH. The consistency proof is necessary to prove the append-only property of the log. A malicious log may craft a special STH per user, causing a *split view* where several different tree heads are valid for the same certificate. This can only be detected with auditing.[3] Performing an audit with a user-specific STH can link an audit to a website visit by a specific user. Even auditing with an honest STH may *contain significant information in terms of the number of log records it represents.* [41]. To mitigate this concern, we argue in Section 4 that public logging of the SCT is sufficient, as the domain owner would be alerted of fraudulent behavior. A non-cryptographic strategy for SCT auditing leveraging the Tor browser infrastructure suggests collecting a random subset of certificates and sending them, after a time delay to mask the browsing behavior, to CT logs for logging [19].

Currently, six CT logs are publicly available and have the required uptime to be considered trusted. Most of them are run by CAs. In contrast, Firefox trusts 177 root certificates from 72 CAs[4], and Chrome trusts 134 certificates from 63 CAs[5]. Safari trusts 159 root certificates from 73 CAs. A certificate needs two SCTs according to browser policy. This number cannot be increased too much as that would lead to a large overhead in certificate sizes. Ongoing efforts to enable easier operation of CT logs (see Section 2.1, Tiled Logs) may lead to the creation of more logs, which means crafting a forged certificate may also become easier. As a result, auditing SCTs will be even more relevant.

2.3 SCT Auditing Proposals

An earlier SoK by Meiklejohn et al. [41] on SCT auditing identified five distinct proposals for SCT auditing with potential for *near-term deployability*. However, PIR was excluded from this category due to the absence of sequence numbers associated with the SCTs in the 2022 Trillian implementation.

2.3.1 Direct querying. , where the audit is performed with either the log or the auditor directly. Performing the audit with the log leaks the browsing behavior and assumes that the user can trust the log. In contrast, performing the audit with the auditor directly introduces a trust assumption with the auditor.

2.3.2 Proxying. the request through one or multiple servers that forward the request to the logs. The SoK identifies proxying as the most performant solution. The main problem with this approach is that it assumes that the proxy and the log do not collude, which does not hold in practice, as the log and the auditing infrastructure are likely run by the same entity. It also introduces a third party in the connection, which makes the audit more complicated and introduces possibilities of interference.

2.3.3 Querying via DNS. is a proposal by Google to use TXT records in internal resolvers to perform an audit. Since the resolver already knows that the user browses a website, no additional information is revealed when the DNS server performs the SCT audit. The request is made with the leaf hash of the SCT. The response from the DNS server uses special records and contains the inclusion proof.

While the protocol was deployed in 2015, deployment was stopped soon after. The reasons for this were "abnormally high failure rates" [23] where the reliance on Google servers meant "tying the two (DNS and CT) together unnecessarily, with grave privacy concerns". Auditing was also only possible for a short time after a certificate was received, as the request for an inclusion proof needs to go through the same resolver used to resolve the hostname to avoid another resolver learning which website was visited.

2.3.4 Local Mirorring. of all logs on the user. This solution is technologically simple and privacy-preserving, but mirrors even just the SCT hashes mean 96GB storage on the user for the 1.5 billion current active certificates with two active SCTs. The list needs to be updated regularly. Adding 340,000 new certificates per hour[6] requires hourly updates of 22MB. In a scenario where slow-issuance certificates are deployed widely and only 0.1% of the domains need SCTs, the storage and update overhead would be more manageable, but the SCTs would only be issued for certificates where the propagation of MTC tree heads is too slow. To avoid leaking browsing behaviour by requesting a specific MTC tree head, we would need another solution.

2.3.5 Private Set Membership (PSM). allows the user to learn whether an entry is in a list held by a server, learning nothing beyond the yes/no statement. The PSM proposal for SCT auditing is similar to

[3] An alternative is gossiping, but they have not been deployed widely https://emilymstark.com/2020/07/20/certificate-transparency-a-birds-eye-view.html

[4] https://hg.mozilla.org/releases/mozilla-beta/file/tip/security/nss/lib/ckfw/builtins/certdata.txt, retrieved on 2024-10-14

[5] http://g.co/chrome/root-store/

[6] Number of precertificates per hour according to MerkleTown, February 2025.

the protocol used for compromised credential checking (C3) [3, 46]. The user queries a partial prefix of the hash to retrieve a *bucket* containing the postfixes of hashes that start with the queried prefix. If the SCT is not in the bucket, the user reports the SCT.

Querying only the hashes may result in an integrity problem, as a malicious log may store certificates in buckets that are not in the log. The mitigation for this attack is to add inclusion proofs to the k-anonymous buckets, which adds communication overhead. Meiklejohn et al.'s SoK [41] suggests adopting the same threat model as in C3: the user trusts a third party, e.g. a browser vendor, to maintain a complete list of certificates. The PSM protocol is performed with the trusted third party instead of the log, which removes the need for inclusion proofs.

The protocol is k-anonymous for covert adversaries and therefore only allows a limited number of audits, and not full auditing. As described in Section 2.4, limited SCT auditing with PSM is currently deployed in Chrome. These proposals all provide different degrees of privacy, all of which fall short of what PIR can provide in theory. However, they were also all devised to cope with the rate of SCT auditing entailed by todays X.509 certificates.

2.4 SCT Auditing in Practice: Google Chome

Most major browsers (e.g., Firefox [32], Safari [1], Chrome [2], and Brave [40]) require at least two SCTs from distinct, trusted logs to validate a certificate. However, among these, only Chrome actively *audits* SCTs.

To start, Chrome stores valid SCTs from certificates of popular websites. In addition, the browser collects unknown SCTs to find indicators of misbehavior. Initially, users had to opt into SCT auditing and sharing their browser history by enabling it in the Enhanced Safe Browsing options. Chrome 90 started to perform k-anonymous SCT auditing by default, auditing each TLS connection with a probability of 0.1%. To provide k-anonymity, the query contains the hash prefix of the SCTs tree leaf, to which the server replies with the set of known hash postfixes that start with this prefix. If the postfix is not in the set, the SCT is reported.

The connection has to be *plausible* to be revealed to the auditing logic, which in the context of Safe Browsing means it belongs to an organization, a public DNS record, or another *realistic source* to avoid privacy leakage. In practice, only publicly trusted roots are audited.[7] Some users may enable *Enhanced Safe Browsing*, which sends all SCTs directly to the Safe Browsing Infrastructure, without anonymization. If a user opts out of Safe Browsing, no data is shared with the Safe Browsing Infrastructure.

To circumvent DoS, Chrome sends SCT auditing reports with a *persist-and-retry* mechanism for lookup queries. The mechanism tries to send the report at most 15 times, first spread over 30 minutes and then several days [47], until it is successful. While the logic prevents large-scale blocking, a targeted attack is still realistic. Blocking updates also stops SCT audits, as Chrome stops checking SCTs after the browser has not been updated for 70 days [21].

2.5 Private Information Retrieval

Private Information Retrieval (PIR) [15] is a two-party protocol for database record retrieval that gives query privacy to the user, meaning the server does not learn anything about the users query. A significant challenge for PIR for CT was that Trillian (see Section 2.1) did not distinguish between sequencing and including an entry. As a result, the lookup would have required using the hashed certificate. This, in turn, means the database needs to be preprocessed in a manner that makes them expensive to update. In 2024, Tessera, the direct successor of Trillian, along with the new StaticCT protocol (see Section 2.1), introduced sequence numbers to CT logs. Now, the lookup can query the sequence number and retrieve a hash of the certificate or Merkle tree leaf hash. The second challenge was that performant PIR protocols for SCT auditing required at least two perfectly synchronized servers to ensure security and performance [31, 39]. Recent advances in PIR [27] demonstrate that synchronization is no longer necessary. Combined with the introduction of sequencing numbers, which simplifies lookups, these recent developments make cryptographically secure PIR-based SCT auditing more efficient than current differentially private audits.

In state-of-the-art, PIR-based, SCT-auditing [27], the user queries the auditing infrastructure with a certificate hash and gets a 1-bit response indicating whether the hash is in the set of known SCT hashes. The known SCTs are hashed into a Bloom filter that hashes the SCT hash into multiple indices into the table and sets the corresponding bit in the filter. During the audit, the user locally computes the index and privately retrieves the bit from the server. Since the filter is known, an adversary could compute a collision string. Therefore, the SCT auditing infrastructure initializes several Bloom filters that are computed using independent hash functions. The user queries the filters at random. Using a large enough number of Bloom filters, the false positive rate drops to 50%, which is tolerable if we perform wider auditing. Concretely, we would need to audit at least twice as much to get the same level of security as for the differentially private lookup currently used in Chrome (see Section 2.4).

New certificates require recomputations of the Bloom filters on the server side. In addition, the audit can only tell if a certain hash is known or not, which should not be a problem when a collision-resistant hash function is used. A more natural approach is to query a sequence number and get the hashed certificate as a response. This eliminates the false positives from the probabilistic data structure and requires minimal preprocessing on the server side, but the 256-bit records would be significantly larger than the 1-bit records.

3 SLOW ISSUANCE CERTIFICATES

Todays ubiquitous elliptic curve signatures are very small: a P-256 signature is only 65 bytes. In stark contrast, post-quantum signatures are typically much larger: an ML-DSA-44 signature requires 2.4kB. There are many signatures involved in a TLS handshake. Using ML-DSA-44 as a drop-in replacement will add more than 14kB. Today, over half of non-resumed HTTP/3 connections handle at least a single request, transferring fewer than 8kB from server to user in total [52]. Earlier experiments from the CDN Cloudflare show that handshakes would be 60% slower [50]. Chromes product manager calls drop-in ML-DSA-44 *undeployable* [5]. For the transition to a post-quantum ecosystem, Chrome [44] formulated several design principles, including for size: adding 2kB is *very painful, but plausible*, but adding around 7kB is *implausible unless*

[7] https://groups.google.com/a/chromium.org/g/ct-policy/c/FddjjCNIrLo

a cryptographically relevant quantum computer (CRQC) is tangibly imminent. The same article considers switching to a *Non-X.509 PKI.*

3.1 The Problem of SCTs and Post-Quantum Certificates

Today in the web PKI, every certificate issued can be used immediately. The immediate issuance poses several challenges, including necessitating SCT audits and a number of signatures, which make certificates large when post-quantum signatures are necessary for authentication. To address the challenges, there are several proposals [11, 30] to give up immediate issuance. The majority of certificates would then be *slow-issuance certificates* that require some time before they can be used.

Slow-issuance certificates avoid the need for SCTs by encoding a *proof of inclusion* in a CT log directly in the certificate. When receiving the certificate, the proof has to be verified. This has already been suggested in RFC 6962-bis [33], Section 7.1.2. There are two possible ways of embedding the inclusion proof: using *fast embedding*, where the proof is fetched by the CA as soon as the certificate is in the log, and *slow embedding*, where the CAs await the MMD for the inclusion proof before issuing the certificate [41]. Shortening the MMD to remove the SCT is not possible, even when including the certificate in the log immediately and adding a Merkle Tree authentication path to the SCT itself. This moves the problem of SCT auditing to auditing the corresponding STH.

Slow-issuance certificates are small. For example, if all certificates today were issued by one CA, a Merkle Tree Certificate would still be well below 1,000 bytes. In addition, the slow issuance proposal is more positive for privacy, as the new tree heads propagate as a part of the protocol. Actively querying for STHs can have a negative effect on privacy due to the small number of certificates represented by an STH as mentioned in Section 2.2.

In a world where slow-issuance certificates are the norm, there are two possible ways of getting a certificate: a slow-issuance scenario where the certificate is requested in due time before it is used, and the tree head can be distributed to the user before the certificate starts to be valid, and a second, less likely *fast-issuance certificate* when slow issuance certificates are too slow. Two examples of slow issuance certificates are Merkle Tree Certificates (MTCs) [11] and Starlit Jellyfish [30]. We discuss them in Appendix A.1. There are a small number of possibilities where certificates need to be published immediately, in less than a minute. We discuss them in Section 3.2, and estimate the probability for the scenarios in Section 3.4.

3.2 Scenarios for Immediate Issuance

To estimate a slow issuance ecosystem, we find that in three scenarios a certificate would be needed immediately.

3.2.1 Brand new domains: A domain that did not exist previously. This is experienced exactly once for each domain. Domains usually take some time to become popular, e.g., all of the 100 most-visited domains existed before 2022 (see Section 3.4.2). Most new domains may not experience high traffic right away. In a slow-issuance ecosystem, a significant portion of new domains could use slow issuance if their certificates are requested sufficiently in advance.

3.2.2 Validity gaps: A configuration error or manual certificate renewal could result in a certificate expiring without an immediate replacement. In a world where slow issuance is standard, a fast-issuance certificate bridges the time between discovering the error and issuing a slow issuance certificate. The analogous error in todays certificate transparency system is a faulty configuration of automatic renewals or forgotten manual renewals of certificates.

3.2.3 Urgent re-issuance: When a domain is moved without sufficient notice, it needs a new certificate immediately. An example of urgent re-issuance is a domain that switches its provider on short notice because it is under a DDoS attack. The list above should cover the most plausible scenarios, but may not be exhaustive. When switching to slow-issuance certificates, like MTCs, regular renewals become more important. Ideally, all domain operators would opt for a well-configured automatic renewal process, eliminating the *validity gap* case.

3.3 User Support

Slow-issuance users can roughly be divided into three categories depending on their support level for slow issuance certificates.

Reliable users supporting slow issuance certificates *with* a reliable update mechanism. They would obtain new tree heads in a regular frequency, for instance every six hours, and rarely fallback to fast issuance. In contrast, *unreliable users* support slow issuance certificates *without* a reliable update mechanism. They would experience more frequent fallbacks, especially when a slow issuance certificate is relatively new. The probability of fallback would decrease inversely with the age of the certificate. Finally, a group of *users without slow issuance certificate support* will always require a fallback to X.509 certificates with SCTs.

3.3.1 User Staleness. In the first and second case, a user becomes *stale* when they do not update their tree heads regularily. They may fall back to using the legacy fast-issuance certificates until they receive the most up-to-date tree heads. An example of a situation where the user can be online but not updated is a restrictive firewall configuration or a captive portal. Problems with restricting configurations exist today and are addressed by a retry and persist mechanism (see Section 2.4). After the situation is resolved, the browser can request the new tree heads immediately.

3.3.2 User Support. In the third case, the user has yet to support slow-issuance certificates. In the meantime, the fallback mechanism needs to be used, i.e., X.509 certificates with SCTs. While browsers are reasonably regularly updated, the users may decline updates for an indefinite time. We expect updates for slow-issuance certificate support to be rolled out more than 70 days in advance, which is compliant with Chromes auditing policy (see Section 2.4).

3.4 Estimating a Slow-Issuance Ecosystem

Without an established slow-issuance ecosystem, it is impossible to know how many users will be reliable and how many will be unreliable. The concrete distribution of domains needing the fallback mechanism will likely also affect the update frequency. Over time, the third category of unreliable users will hopefully shrink when a slow-issuance ecosystem is deployed. In this paper, we will not focus on user behaviour, as the concrete access patterns to the

SCT database are oblivious. The overall number of queries to the database will be determined by the number of audits per user. Right now, the only audits performed by users are limited to three in Chrome, so it is difficult to give estimates. However, we discuss strategies to group audits into a single query in Section 7.

The server behavior is more clear. Assuming that current X.509 certificates are simply replaced with slow-issuance certificates where possible and fast-issuance certificates otherwise, we present the first estimate of a slow-issuance ecosystem. The resulting upper bound can help to facilitate decisions in the final design of slow-issuance certificates. We use data from public CT logs and the criteria from Section 3.2 to obtain a loose upper bound on how many certificates would need fast issuance. Overall, we find that 0.1% of the domains would need a fallback mechanism, using the conservative assumption that *it takes three days to propagate the Merkle tree heads to clients.*

3.4.1 Sampling Method. We use two datasets for our estimations. The first dataset contains certificates from random domains. Specifically, we include domains where the `sipHash64`[8] of one of the DNS names is congruent to zero modulo 10,000. The certificate validity spans 2.5 years, with their `notAfter` value ranging between 2022-01-01 and 2024-08-03. This *random domains* dataset contains 616,142 certificates corresponding to 123,773 unique domains.

A single website may have multiple valid URLs. For example, the domain `abc.xyz` also appear as `*.abc.xyz` or `www.abc.xyz`. However, the `sipHash64` used in the sampling method will only match one of these variations, introducing a significant number of false positives in our sample. In addition, our datasets may omit certain valid certificates. Consequently, our analysis offers only an approximate *upper bound*. The issue became evident in our analysis of validity gaps for misconfigurations in Section 3.4.3. Most domains have only one gap, which suggests our random domain analysis can be a loose upper bound at best. The duplicates could be detected with manual analysis, which is infeasible due to the large number of certificates, or querying logs directly with the possible subdomains, which would be very time-consuming.

The second dataset consists of certificates from 75 of the top 100 domains.[8] The remaining 25 domains, primarily Content Delivery Networks (CDNs), do not issue certificates. This *top domains* dataset contains 36,375 certificates.

3.4.2 Brand New Domains. To estimate how many new domains are created per fallback period, we measure the number of new domains added daily. These domains require the fallback mechanism until the new tree head is propagated to all relaying parties. To distinguish new domains from established ones, we analyze the timestamps of all certificates associated with each domain and order them chronologically. If the earliest timestamp is later than 2022-01-01, the domain is classified as *new*. Otherwise, it is considered *established*. This cutoff date was chosen because the dataset is incomplete before this period. In any case, extending the timeframe would be counterproductive, as our analysis indicates a steady rise in the number of new domains.

All top domains in our dataset existed before 2022-01-01. In the randomly sampled dataset, the most significant daily change occurred on 2023-11-27, when 216 certificates were seen for the first time, representing 0.15% of the domains. On a typical day, a median of 95 new certificates is observed in our dataset, covering 0.07% of domains. Since this is a one-in-a-thousand sample, we estimate that around 95,000 domains are added each day. Our findings align closely with Let's Encrypts reported growth statistics.[9] Between January 2022 and August 2024, the number of *registered domains active* increased from 90 million to 137.5 million, corresponding to an average of approximately 52,000 certificates per day. Let's Encrypts market share fluctuates between 45% (October 18^{th}) and 68% (October 27^{th}), with a median market share of 58%.[10] Overall, Let's Encrypt has issued approximately 70% of all the certificates observed in our analysis.

3.4.3 Validity Gaps. A certificate may expire without a new, valid certificate immediately replacing it, leading to errors for users attempting to access the domain. The slow issuance proposals crucially rely on automation to mitigate these oversights, especially given the shorter certificate lifetimes. Using our dataset, we estimate the distribution of such oversights by taking all validity periods and checking if they are continious. For the top domains, we found no expiry after cross-checking our dataset with the certificate database https://crt.sh. Due to the size of our dataset, cross-checking was not performed for the randomly sampled domains. However, our analysis estimates an upper bound of 3.11% for domain expiry, excluding domains for which we only observed a single certificate.

Figure 1 shows the time it takes for a domain to replace an expired certificate with a valid one if there is a validity gap. During our analysis, we observed that several certificates have gaps of only a few seconds. For example, a domain had a certificate that expired at 23:59:59 and replaced it with a new certificate to take its place that is valid from 00:00:00 for the next day, just one second later. This would most likely not become an error. To refine our analysis, we assume that automatic renewals are in place when the new certificate is valid less than 10 seconds after the expiry of the old certificate. Correcting these configurations would remove the need for a fallback mechanism. This adjustment lowers the failure probability for the randomly sampled domains to 3.03%. Over the 2.5 years covered by the data set, on average 0.003% of all websites had an expired certificate on any give date. Assuming the incident with the invalid certificate lasts for three days, the probability increases to 0.01%.

3.4.4 Moving Domains due to an attack. When a certificate remains active for a long time but is replaced by a new certificate from a different CA, this may indicate an unplanned domain move. In case of an unplanned move, the new certificate is typically used immediately, while the old certificate is left to expire, as certificate revocation is rare [51]. To estimate renewal patterns and identify indicators of unplanned domain moves, we plot the overlap in lifetimes between consecutive certificates, considering cases where the CA changes and where it does not. Figure 2 shows a clear trend in renewal patterns: most certificates are renewed 14, 30, or 90 days before their expiration. This pattern provides a basis for distinguishing planned from unplanned domain moves. If a

[8] Identified from https://radar.cloudflare.com/domains.

[9] https://letsencrypt.org/stats/#, October 2024

[10] Numbers from *Issuance per day by Certificate Authority* section of MerkleTown.

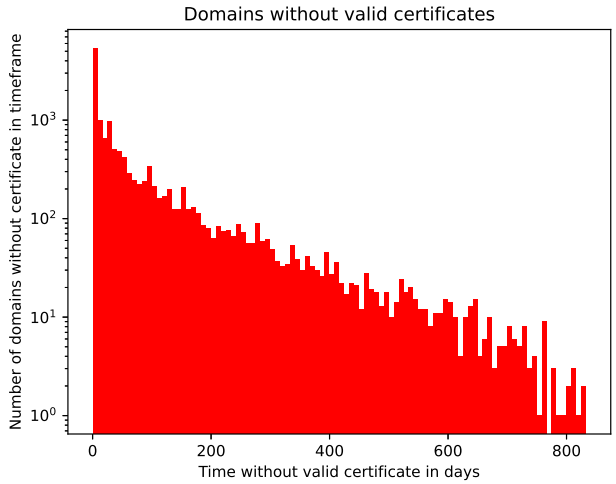

Figure 1: Time (in days) a domain is without a valid certificate. A majority of certificates are renewed within 24 hours of expiry, hinting at a wrong configuration.

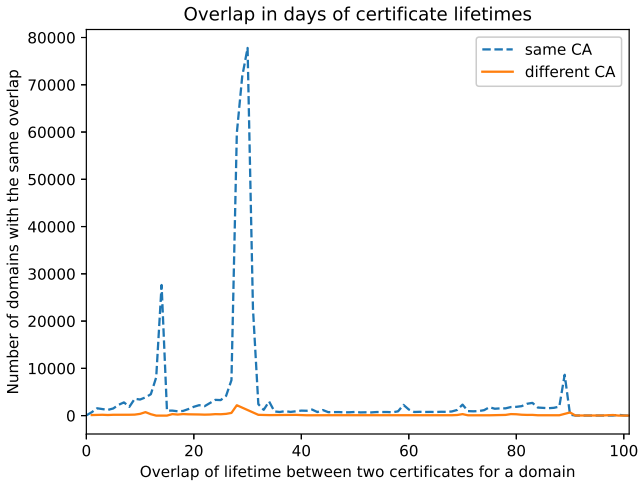

Figure 2: The overlap of a lifetime between two certificates is the time where a domain has two valid certificates. In the graph, we can see clear spikes at 14, 30, and 90 days that indicate an automatic renewal of certificates.

domain switches CAs and renews its certificate 14, 30, or 90 days before expiry, we classify the domain move as planned. Adding a tolerance window of ±24 hours results in a probability of 0.01% for an unplanned domain move.

Accounting for *backup certificates* is also important for these statistics. We classify a certificate as a backup certificate if two certificates have the same validity start time, and both CAs are accepted as a potential follow-up CA. Additionally, if we find an overlapping certificate from the same issuer with at least a second of overlap with the current certificate, we assume the move is planned and do not count it as a fallback case. Otherwise, we categorize the move as unplanned, indicating that we would need the fallback mechanism.

3.4.5 Probability of a Fallback. To conclude, assuming it takes three days for an MTC tree head to propagate, under a million fast-issuance certificates, which would make up around 0.07% of all certificates, are needed at any given time. They are composed of the following number of fast-issuance certificates:

- 300,000 from validity gaps,
- 285,000 from brand new domains, and
- 300,000 from unplanned domain moves.

We hope that the validity gap case disappears over time. In this case, the database with SCTs for fallback certificates size shrinks from $2^{19.7}$ to $2^{19.1}$.

4 TOWARDS FULLY PRIVATE AUDITING

To evaluate PIR for SCT auditing and to compare the existing schemes, we first establish a set of common criteria for PIR schemes in the context of SCT auditing.

4.0.1 Lookup requirements. We significantly relax the requirements for the lookup itself. The initial proposal for PIR with SCT auditing [39] stores membership proofs in a PIR database. Follow-up work [31] proposes retrieving an inclusion proof from the log instead for improved practicality. However, providing the inclusion proof opens the protocol for an attack by a fully malicious log [41], as an inclusion proof can be computed relative to a specific STH. This creates a direct mapping between the STH and certificates, requiring users to verify the consistency between the STH and a previously known, trusted STH. The verification process necessitates revealing the STH to the auditing infrastructure, which in turn reveals the query.

We propose an alternative: performing a private lookup with a trusted third party, such as the browsers infrastructure. This approach is reasonable, given that a certain amount of trust is already placed in the browser. A similar relaxation was suggested in Meiklejohn et al.'s SCT auditing SoK [41]. By trusting the browsers infrastructure, the lookup process can be simplified to verify the inclusion of an SCT and confirm it has been seen previously by the infrastructure. This reduces the problem to a straightforward hash lookup of the SCT by its sequence number. The lookup is queried by sequence number. The response can be either a 256-bit hash of the certificate or a single bit (sequence number is known/unknown), as we will discuss in Section 5.

4.0.2 Database preprocessing. Some PIR schemes rely on a per-user, data-independent preprocessing phase to get a more efficient online phase [13, 27]. However, with approximately five billion potential daily users, and frequent database updates, only minimal per-user preprocessing is feasible. In contrast, user-independent preprocessing of the database is more practical, as it can be performed near-real-time during database updates. An example of user-independent preprocessing is arranging the database in a structure optimized for efficient computations, such as a multiplication-friendly layout. In contrast, PIR schemes using hints perform per-user preprocessing.

4.0.3 Batching. Ideally, SCT audits should be performed before completing the TLS handshake since "any harm that would come from a mis-issued, improperly-attested certificate" [9] is prevented. For practicality, a retrospective batched lookup performed once per day is acceptable. PIR schemes often include batching mechanisms to amortize the high communication cost. For SCT auditing, a simplified batching approach, such as returning the sum of several records, may be sufficient. We show state-of-the-art PIR schemes in Table 1, and discuss batching strategies in Section 7.

Batching techniques may also lower both the server computation cost and overall communication cost when taking the *distribution of popular domains* into account [34]. At the cost of client-side storage, distributional PIR can reduce the server computation by a factor of 12 and a factor of 3 for communication by keeping distinguishing between popular and unpopular domains

4.0.4 Communication. Building on previous work [27], we assume the average user performs around 1,000 TLS connections per day. Each handshake of these connections requires 2.5 to 5kB of communication. The differentially private PIR lookup in Chrome [21] takes 240kB. Since we want to be strictly better, a daily fractional increase of 100kB for batched SCT auditing is acceptable. Immediate auditing at the time of the TLS connection brings more value and thus more communication is acceptable. There are no sharp limits, but doubling TLS handshake traffic (2.5kB) would be unpalatable.

A PIR audit needs to have strictly better, or sublinear, communication complexity than downloading the entire database to be acceptable. Depending on the exact auditing setting, this may be plausible, as an important consideration is timing: In a fast issuance setting (see Appendix A.1), the database would need to be updated frequently, for example once a second. The PIR online phase nonetheless needs to be better than updating a list of trusted tree heads of SCTs on SCT encounters.

4.0.5 Computation. Again, we follow the assumption of Meiklejohn et al.'s SoK [41] that the user is on a commodity laptop or a mobile phone. The server-side computational demands are less clearly defined. One core-second per user per day would require about a thousand 64-core servers in full use, which incurs a substantial but potentially acceptable multi-million dollar yearly cost. As discussed in the previous paragraph, these costs are easier to justify with immediate auditing. In addition, some proposals, namely the Bloom filter proposal [27], require regular database updates that recompute the database format. We want to minimize this overhead.

4.0.6 Reusing key material. In many PIR schemes, the users public key is relatively large compared to subsequent queries and responses. For daily batched lookups, reusing the users public key is acceptable. For immediate auditing, reusing key material introduces privacy concerns, as it could reveal when the user performs TLS handshakes. A potential mitigation are randomized delays to obscure timing correlations, a method previously proposed for proxy-based SCT auditing in the Tor browser [19]. The problem is further exacerbated if immediate auditing is only required for fast-issuance certificates, as that also leaks that the user visited a website requiring a fast-issuance certificate. This can be ameliorated somewhat by having the user use multiple public keys, each limited to a small number of uses.

4.1 PIR: State of the Art

Modern PIR schemes are based on lattices. The standard PIR protocol queries the database by performing an oblivious Matrix-Vector multiplication between the database matrix and a basis vector indicating the index of the database item to be retrieved. The form of the database and the query depend on the concrete PIR scheme. We now discuss the state of the art and the applicability of the schemes to SCT auditing. Since the publication of Meiklejohn et al.'s SoK [41] in 2022, PIR schemes have significantly improved. First, SimplePIR and DoublePIR [27] demonstrated that performant PIR is feasible with a single server. Further developments have reduced the online communication significantly [13] and eliminated the preprocessing from the protocol [26, 36], until the resulting protocol was almost as performant as the protocol with preprocessing for very small records [43].

While most PIR schemes benchmark against databases containing millions of records, the records are usually at least 256 bytes large. In contrast, SCT auditing requires much smaller records of 32 bytes or even a single bit, as discussed for the lookup requirements in Section 4, but a vastly larger database with at least 3 billion[11], or $2^{31.5}$, records. We discuss approaches to make the database size more manageable in Section 5.2 and Section 7.

PIR schemes can be categorized by the encryption scheme used to ensure privacy for the queries. The encryption either uses fully homomorphic encryption (FHE) or the Learning-With-Errors (LWE) problem. We now discuss both in the context of SCT auditing.

4.1.1 PIR from Fully Homomorphic Encryption. FHE allows arithmetic operations on ciphertexts, where operations performed on encrypted data correspond directly to the same operations on the corresponding plaintext data. For example, given a field element x, encrypting it $\mathsf{enc}(x)$, multiplying it with another field element y, and then decrypting the result is equivalent to multiplying y with the unencrypted field element, such that $\mathsf{dec}(\mathsf{enc}(x)y) \equiv xy$.

SealPIR [7], introduced in 2018, uses the SEAL homomorphic encryption library [45], which uses the BFV scheme [12, 25] to encrypt the ciphertexts. Using BFV in privacy-preserving protocols is common, for example in Apples private caller ID protocol [4]. SealPIR encrypts the index of the record that the PIR scheme should retrieve. The index is obliviously expanded to a vector where all indices are 0 except for the provided index on the server. For databases with entries so large that they do not fit in a single plaintext SEAL PIR offers two solutions: Either expanding multiple ciphertexts and concatenating them, or representing the database as a *d-dimensional hypercube*. In the hypercube representation, the dimension d significantly expands the database capacity. For example, for a lattice dimension $N = 4096$, increasing d from 1 to 2 changes the database representation from a vector to a matrix, allowing the database size to expand from 4096 to 16.7 million entries. Expanding to $d = 3$ allows up to over 68 billion database entries. However, this comes at an online cost, as a ciphertext is needed per dimension to correctly address the ciphertext, which in turn means that Server-side computation gets more expensive with each dimension increase, as the ciphertext expansion factor increases after processing each dimension. In addition, the database is padded with dummy plaintexts if

[11] In November 2024, 1.5 billion active certificates were counted by Cloudflares Merkle Town https://ct.cloudflare.com/. Each certificate has at least two SCTs.

the database does not perfectly fit in the hypercube. To better fit the database into the dimension, a hyperrectangle representation can be used instead.

Recent refinements add computation-communication tradeoffs to the SealPIR protocol [6] using additive recursion steps. The recursion compresses the secret key upload and download and improves oblivious query expansion techniques. Spiral [42] reduces the online cost by a factor of over 10× in comparison to SealPIR. Its online query size is independent of the database size, and the response size grows slowly with the size of the database. Spirals small online communication comes at the cost of large public parameters and higher computational cost, as the preprocessing cost of a query grows linear with the database size.

For databases with smaller records, Respire [13] adapts Spirals methods, but uses a subgroup for the query and the response and incorporates modulus switching to fix the noise growth. In addition, Respire supports dimension reduction. While Respire achieves the smallest communication overhead, it requires per-user storage in the form of a *user rotation key* that makes the schemes impractical for SCT auditing given the large user base.

4.1.2 PIR from Learning-With-Errors. PIR schemes based on LWE and its ring variant RLWE encrypt a base vector indicating the index and offer tradeoffs for communcation and computation: LWE enables faster computation, but RLWE elements are smaller, enabling more efficient communication. Over the past two years, a number of new schemes emerged that are promising for SCT auditing.

SimplePIR [27] shows efficient PIR can be achieved using a large, precomputed hint matrix as a reusable query key that is uploaded to the server before starting the queries. A drawback is that the hint in SimplePIR is several hundred megabytes large. DoublePIR, introduced in the same paper, reduces the size of the hint by adding a recursion step, at the cost of a higher per-query communication. However, updating the database requires updating the corresponding row of the hint, which may involve transmitting several megabytes per update.

FrodoPIR [20], developed concurrently with SimplePIR and DoublePIR, eliminates the hint by preprocessing the database on the server side without interacting with the user. Arithmetically, SimplePIR and FrodoPIR are very similar, but FrodoPIR formats the database as a row vector while SimplePIR formats the database as a square matrix to balance the user up- and download. As a result, the public parameters of FrodoPIR are significantly smaller than in SimplePIR, but FrodoPIR requires more online communication than SimplePIR.

HintlessPIR [35] and TiptoePIR [26] remove the need for a hint from SimplePIR by embedding the users secret key into the query and evaluating it homomorphically on the server. This approach increases the communication complexity, as the plaintext space of the outer RLWE scheme needs to be large enough to fit the ciphertext of the inner LWE scheme. The main difference between the TiptoePIR and HintlessPIR is in the performance and the database representation. TiptoePIR uses a column-major matrix-vector multiplication but has higher communication and computation requirements than HintlessPIR, which formats the database as a square matrix. Overall, removing the preprocessing from SimplePIR by using HintlessPIR instead increases online communication by a factor of four.

Finally, YPIR [43] improves hint packing by using the Chen-Dai-Kim-Song algebraic packing transformation [14] instead of bootstrapping. For small database records, YPIR has a slightly higher upload size than HintlessPIR, but outperforms HintlessPIR in every other metric. In terms of communication, YPIR does not quite outperform DoublePIR because extra key material is sent for key-switching but has the best communication complexity without preprocessing at the time of writing.

5 DOWNSIZING THE DATABASE

The communication and computation complexity of PIR schemes shrink significantly with the database size. In contrast, the ecosystem seems to be growing steadily. A common approach to handle large databases is to split, or *shard*, the database into multiple smaller, independent databases. Logs are currently sharded into epochs between six months and a year.[12] We now propose two ways to shard the SCT auditing database even further. The first one is inherent in the StaticCT structure, where entries are combined in 256 item large *tiles* that can be privately retrieved. The second proposal extends the idea of using the structure of the Merkle Tree by adding *distinguished roots* that are published in epochs rather than tree size.

5.1 Sharding by Structure: Tiled Logs

StaticCT, and, internally, Tessera, group entries into *tiles* for easier caching (see Section 2.1). Instead of auditing the entire database, the PIR lookup can verify whether a specific tile has been included in the tree by recomputing the tile and auditing the corresponding entry at the tile above, for example at level 1 instead of level 0. Auditing higher levels in the tree significantly cuts down the tree size. Each level higher in the tile hierarchy reduces the database size by a factor of 8. For example, auditing at level 1 reduces the tree size from $2^{31.5}$ to a more managable $2^{23.5}$.

To ensure the SCT has been included in the tree, the other 255 entries of the tile are needed to compute the hash of the parent tile. The hashes of these entries can be obtained in one of two ways: Either including the remaining entries directly in the PIR response, or embedding the hashes of the remaining entries within the certificate. The first proposal increases the response size by 8,192 bytes per level, while the second proposal increases the certificate size by 8,192 bytes per level and slows down the time until a certificate can be used, as the tile has to be filled before the certificate is issued.

5.2 Sharding by Epoch: STH Discipline

The high-level idea of canonical STH computation is that logs periodically, for example once per second[13] , compute a new canonical STH of all certificates they received in the meantime. The audit is then performed with the canonical STH instead of the STH of the full log. This significantly reduces the size of the database.

Auditing relative to an STH that is published periodically by a log, where each certificate is audited with respect to a *canonical STH* for which the log has an inclusion proof, has previously been proposed as *STH discipline* [9]. The SCT audit now performs against

[12] https://ct.cloudflare.com/logs

[13] Chrome currently accepts static CT logs only if they have an MMD of less than a minute [37], and considered 10 seconds or less as an MMD [22].

Table 1: An overview of relevant criteria for SCT auditing using state-of-the-art PIR schemes, taken from the original papers.

Criteria	SealPIR [6, 7]	HintlessPIR [36]	Respire [13]	DoublePIR [27]	YPIR [43]
Encryption	FHE	LWE	FHE	LWE	LWE
Simple Updates	✓	✓	✗	✗	✓
No per-user storage	✗	✓	✗	✓	✓
No Hints	✓	✓	✗	✗	✓
Dedicated support for small records	✗	✓	✓	✓	✓
Dedicated batching	✓	✗	✓	✓	✓
Potential candidate	✓	✓	✗	✗	✓

the canonical STH instead of the overall tree root. Adjusting the database structure significantly reduces the database size for the PIR computation, which becomes proportional to the frequency of computing the distinguished roots rather than the total number of certificates.

For example, with a certificate validity of 90 days and canonical STHs computed every second, there would be less than 2^{23} active canonical STHs at any given time. To optimize the database size even further, we propose computing an additional, daily root after 24 hours. This approach reduces the number of active roots even further while enabling near-instant auditing. This shrinks the database to 86,489 records at any time, as there are 86,400 seconds in a day. There are three possiblities to obtain the authentication path in this setting:

(1) A first certificate immediately contains an authentication path to the root published every second. The same certificate is modified to contain an authentication path to a daily root later.
(2) The certificate initially only includes the first authentication path, and is replaced with a second certificate that contains the authentication path to the daily root.
(3) The certificate immediately includes the first authentication path via a separate extension. The daily authentication path to the daily root is added and included at a later point by the server.

The first and second approach require updating the certificate at a later point, which makes the third approach a bit more appealing. The main deployment challenge for this approach is to determine a realistic period for computing and distributing canonical STHs. In addition, CT logs would need to add a new endpoint to retrieve the distinguished roots list.

6 BENCHMARKS

We measure the performance of implementations of the candidate schemes from Table 1 with set sizes for tiles and STH discipline in Table 2. The programs are run on Ubuntu 22.04 with a fixed-frequency AMD Ryzen 9 7900X 12-Core Processor with 125 GB RAM. The benchmark baseline is the state-of-the-art Bloom filter approach with YPIR(*plain YPIR*)[14] that retrieves a 1-bit record from a database with $2^{31.5}$ entries. The same database with 256-bit entries times out or is impossible for all considered PIR schemes.

We perform all benchmarks with the default configuration of the PIR scheme, and take the median over 10 runs. The measurements only cover the online time and exclude query-independent server preprocessing time. We measure SealPIR[15] twice with different database representations. Once with $d = 1$, representing the database as a vector with fewer elements packed in a plaintext for padding, which reduces the communication, but increases the server-side computation significantly. Representing the database as a hyperrectangle by setting $d = 2$ gives faster computation at the cost of increased communication. Heightening the dimension to $d = 3$ does not yield performance improvements. YPIR records can at most be 8 bits large. Combining YPIR with SimplePIR packing allows database records of 28672 bits or larger. We pack 112 records in a single database record to account for the larger modulus. E.g., for a for the database with $2^{15.5}$ records, the model database consists of 414 28672-bit records. Finally, HintlessPIR[16] is called with the preconfigured parameters.

Table 2 shows that YPIR with a Bloom filter ("plain YPIR") has the smallest response size overall. However, using other PIR schemes has better tradeoffs with the request sizes for all our proposals. Especially SealPIR performs surprisingly well, keeping several instantiations under 100 kB of overall communication with 46 kB per party for the tile-based embedding strategy, which performs the audit with the $(n-2)^{th}$ tile, going up two levels of tiles in the log and gives the best performance when the authentication path to the tile is embedded in the certificate. Retrieving the authentication path from the database instead is only possible with YPIR, and gives relatively bad performance, especially on the user side. STH discipline gives a better performance than tiled embedding with large records and does not increase the size of the certificates much, and only requires adding an API endpoint.

7 BATCH AUDITING

The previous section shows how to conduct a single PIR call efficiently enough to perform an SCT audit per day. Auditing 1,000 connections per day at even 46kB per audit, which we deem to be the lowest number currently possible in Table 2, is still prohibitive. Currently, Chrome clients audit one in a thousand connections [21]. Moreover, combining STH discipline with slow-issuance certificates would reduce the audits to a single audit per day on average, as 0.1% of the connections would use the fallback need to be audited.

[14] https://github.com/menonsamir/ypir

[15] https://github.com/microsoft/SealPIR

[16] https://github.com/google/hintless_pir

Table 2: Benchmarks for SCT auditing databases with $|DB|$ records. $2^{15.5}$ for tile-based embedding with the $(n-2)^{th}$ tile, 2^{17} for the canonical certificates *with* a daily new root, $2^{19.7}$ records of fast-issuance certificates in an MTC ecosystem, 2^{23} for canonical certificates without daily new roots, and 2^{24} for tile-based embedding with the $(n-1)^{th}$ tile as a distinguished root. The record size $|r|$ is either 256 bits when a hash is retrieved or includes a path to another tile. The best result is marked bold, and schemes that take less than 100 kB of communication or 1 second of computation are grey. Timed-out protocols are excluded.

Database parameters	Scheme	Request Size	Response Size	Computation User	Computation Server
$\|DB\| = 2^{31.5}$ $\|r\| = 1$	plain YPIR	735 kB	12 kB	884 ms	53 ms
$\|DB\| = 2^{15.5}$ $\|r\| = 256$	SealPIR $d = 1$	**46 kB**	46 kB	**1 ms**	112 ms
	SealPIR $d = 2$	93 kB	185 kB	**1 ms**	32 ms
	HintlessPIR	370 kB	5.9 MB	7 ms	790 ms
	YPIR-SimplePIR	487 kB	**12 kB**	34 ms	**28 ms**
$\|DB\| = 2^{15.5}$ $\|r\| = 131072$	SealPIR $d = 1$	-	-	-	-
	SealPIR $d = 2$	-	-	-	-
	HintlessPIR	-	-	-	-
	YPIR-SimplePIR	932 kB	61 kB	2.2 s	197 ms
$\|DB\| = 2^{17}$ $\|r\| = 256$	SealPIR $d = 1$	**46 kB**	46 kB	**1 ms**	311 ms
	SealPIR $d = 2$	93 kB	186 kB	**1 ms**	60 ms
	HintlessPIR	371 kB	6.0 MB	23 ms	788 ms
	YPIR-SimplePIR	487 kB	**12 kB**	34 ms	**26 ms**
$\|DB\| = 2^{19.7}$ $\|r\| = 256$	SealPIR $d = 1$	**46 kB**	46 kB	**1 ms**	217 ms
	SealPIR $d = 2$	93 kB	186 kB	**1 ms**	48 ms
	HintlessPIR	373 kB	6.1 MB	47 ms	794 ms
	YPIR-SimplePIR	530 kB	**12 kB**	34 ms	**29 ms**
$\|DB\| = 2^{23}$ $\|r\| = 256$	SealPIR $d = 1$	325 kB	46 kB	8 ms	18.6 s
	SealPIR $d = 2$	**93 kB**	185 kB	**2 ms**	1.5 s
	HintlessPIR	383 kB	18.2 MB	127 ms	1.7 s
	YPIR-SimplePIR	1.4 MB	**12 kB**	46 ms	**66 ms**
$\|DB\| = 2^{24}$ $\|r\| = 256$	SealPIR $d = 1$	650 kB	46 kB	16 ms	36.9 s
	SealPIR $d = 2$	**93 kB**	186 kB	**2 ms**	2.8 s
	HintlessPIR	389kB	24.3 MB	250 ms	2.3 s
	YPIR-SimplePIR	2.3 MB	**12 kB**	59 ms	**108 ms**

With slow-issaunce certificates widely deployed, it is unlikely that a client will need to audit more than a handful of fast-issuance certificates on a given day. Until slow-issuance certificates are deployed, *batching* the audits to perform all audits once per day in a single query amortizes the communication in the same way. Batching in the context of PIR efficiently packs several queries into a single ciphertext. This can amortize the cost of PIR by packing multiple SCT audits into a single private query. Table 1 shows that several PIR protocols support batching natively.

We observe that for the problem of SCT auditing, we do not care about the entries themselves, but only about gaining assurance whether the auditing infrastructure has already seen the certificates at hand. This observation gives rise to investigating weaker notions of batching that still allow us to audit several SCTs in a single query that can be used alongside traditional PIR batching to perform a larger number of queries in a single PIR protocol call.

7.1 Private Retrieval of Sums of Hashes

Several PIR schemes are straightforward to modify to query *the sum of* multiple entries by their indices. The standard vector-multiplication PIR protocol discussed in the beginning of Section 2.5 is a special instance of the LinPIR protocol [35] that has been suggested as a generalization of PIR protocols that allow querying arbitrary linear combinations of records such that the result of the protocol is $\sum_i a_i \mathsf{DB}_i$. The user can locally compute the sum of the hashes of the certificates and compare the result to the PIR response. If they differ, which could happen when at least one certificate is not in the database or the database knows a different certificate hash for a certificate, the user knows that there is a problem with at least one certificate.

7.1.1 Batching with LWE-based PIR. The LWE protocols from Section 4.1.2 encode the index for the query in a vector. A simple approach to batching is to encode *multiple* indices in the query to get the sum of multiple records as a response. However, each homomorphic addition adds noise to the server response ciphertext.

When the noise grows too large, the decryption of the server response may be incorrect. Lemma 17 of the LinPIR [35] analyzes the error growth for additive batched evaluation. However, the paper gives no concrete bounds for the number of entries that can be sum-batched per request. From the LWE-based PIR protocols, YPIR supports some batching to answer multiple queries at once, which may be of independent interest. Since HintlessPIR and YPIR do not give us the best performance, we leave the concrete sum-batched instantiation to future work.

7.1.2 Batching with FHE-based PIR. The FHE-based protocols from Section 4.1.1 encode the index i of the database record as a polynomial x^i, blind the polynomial, and then perform an oblivious expansion technique that results in the base vector for the multiplication. The expansion is a linear operation. To obtain the LinPIR combination of e.g. three elements i, j, k, the polynomial $x^i + x^j + x^k$ can be sent to the server instead. The result is an additive linear combination of database records.

7.2 Security of Retrieving Sums of Hashes

Without batching, we need our n-bit hash H to be collision-resistant when checking if the auditing infrastructure already knows of a certificate hash. When querying multiple hashes in a single PIR query, we instead need the *AdHASH* map $x_1, \ldots, x_k \mapsto \sum_{i=1}^{k} H(x_i)$ to be collision-resistant. The AdHASH map was introduced and studied by Bellare and Micciancio in [10]. They reduce its collision security to known hard lattice problems, but were unable to give guidance on how to translate parameters. Five years later, Wagner showed [49] there is an algorithm to find collisions in time and space $O(k2^{\frac{n}{2+\lg k}+1})$. Wagners attack initially becomes easier with larger k, but this changes when $k \geq 2 - \sqrt{n}$. Thus we can break a 16,284 batched query using 256-bit hashes with about 2^{32} time and space. Wagners attack would need 2^{128} time and space to break the same 16,284 batched query if we use a 1,808 bit hash. It is unclear whether Wagners attack is close to optimal. The information theoretic lower bound is $\Omega(2^{\frac{n}{2k}})$. We can get more assurances and efficiency by expanding the space of possible parameters.

7.3 Retrieving Signed Sums of Hashes

The straightforward mitigation of Wagners attack is using a hash function with longer outputs, which potentially makes the PIR scheme less efficient as the database records become larger. An alternative is to make it harder for the log to respond with a collision by guessing a query by adding more entropy to it. Instead of querying a vector of 0s and 1s, we can querying for a *signed sum*. Instead of retrieving the sum of entries, we propose the user picks random coefficients $s_i \in \{-1, 1\}$ and performs a query for the signed sum $\sum_{i=1}^{k} s_i x_i$. The user then checks whether the corresponding signed sum of SCT hashes matches the result. Clearly, for known signs s_i, Wagners attack still applies. Crucially, the attacker does not know the s_i and has to guess them. The attacker need not guess all of them: only for the subset for which the attacker has prepared a collision. Even then, the attacker is allowed to get the global sign wrong. For details see 7.3.1. If we include retries for guessing the sign, the attacker is expected to perform $O(2^{k+\lg k+\frac{n}{2+\lg k}})$ work.

7.3.1 Security analysis querying signed sums. We analyze the security of querying signed sums described in 7.3. The attacker can guess the signs $s_1, \ldots, s_n$ of a future audit, and then prepare two lists of certificates. The certificates in the first list are innocuous, legitimate, and publicly logged with SCT hashes $x_1, \ldots, x_n$. The second list contains the certificates that are presented during the active attack shadowing the legitimate first list with SCT hashes $y_1, \ldots, y_n$. Of those, lets say y_1 is for the certificate the attacker wants to mis-issue as its goal. To fool the user, the attacker must ensure $\sum_{i=1}^{n} s_i x_i = \sum_{i=1}^{n} s_i y_i$. Without loss of generality, we may assume $x_i \neq y_i$ for all i: Unchanged certificates don't affect the query or attack. Furthermore, we may assume the attacker uses the smallest collision: there is no $\emptyset \subset \Delta \subset \{1, \ldots, n\}$ with $\sum_{i\in\Delta} s_i x_i = \sum_{i\in\Delta} s_i y_i$. Indeed, if there were, the attacker could restrict their certificates to either Δ or $\{1, \ldots, n\} \setminus \Delta$. The attacker has some respite guessing the signs: if they guess every sign incorrectly, then the user is also fooled, as $-\sum_{i=1}^{n} s_i x_i = -\sum_{i=1}^{n} s_i y_i$.

But that is all: if the user instead uses $s'_1, \ldots, s'_n \in \{-1, 1\}$ with $s' \notin \{s, -s\}$, then the attackers meddling is discovered. Indeed, reasoning towards contradiction, assume $\sum_{i=1}^{n} s'_i x_i = \sum_{i=1}^{n} s'_i y_i$. Combining with the attackers preparation, we get $\sum_{i=1}^{n} \frac{s_i+s'_i}{2} x_i = \sum_{i=1}^{n} \frac{s_i+s'_i}{2} y_i$. Write $t_i \equiv \frac{s_i+s'_i}{2}$ and $\Delta \equiv \{i;\ s_i = s'_i\}$. Note that $t_i = s_i$ for all $i \in \Delta$. Thus, by our assumption of minimality, either $\Delta = \emptyset$ or $\Delta = \{1, \ldots, n\}$. In the former case, $s' = -s$ and in the latter $s' = s$. Contradiction.

8 CONCLUSION

In this paper, we studied the recent developments towards more private SCT auditing. Recent advances both in the certificate ecosystem and Private Information Retrieval seem to bring wide SCT auditing with PIR to the brink of practicality. In particular, we discuss some novel batching ideas and give guidelines towards a practical deployment with sharded databases that would enable private SCT audits in under 100kB per day in Table 2. In addition, we analyzed and estimated a slow-issuance ecosystem. The results show that SCT auditing in general, but especially in the context of slow issuance, is becoming a lot more plausible. In particular, leveraging slow issuance certificates like Merkle Tree Certificates or Starlit Jellyfish can lower the number of SCT audits to one in a thousand connections, a number that is already close to the current auditing rate of Chrome.

Future Work. Revisiting SCT auditing, we have shown that SCT auditing with the web browsers infrastructure and sequence numbers is feasible, and techniques like using distinguished roots and tiles can be leveraged to obtain a fast protocol. We also show how to batch audits by using the algebraic structure of PIR protocols, and provided a thorough analysis of batched SCT audits. The performance of batching may be improved by taking the popularity of domains into account [34].

Relaxing PIR to batched set membership may apply to related problems such as compromised credential checking [3, 46] and lookup of a known spam caller ID [4].

Additionally, we discuss how future deployments of slow-issuance certificates impact the certificate ecosystem, and provide scenarios where slow issuance is too slow, and discuss how SCTs could be

audited then. To get more accurate numbers on the slow-issuance fallbacks and the ecosystem in general, a practical deployment test is necessary. The test results can tighten the bound on the certificates, and give a more stable bound on fallbacks in Section 3.4.5, potentially even deriving a lower bound than our estimated 0.1% of overall connections.

For a complete deployment of PIR-based SCT audits, some practical challenges are still open: Networks may restrict traffic to the point that SCT auditing is refused. For example, captive portals tend to rewrite DNS responses to redirect to their websites and disallow any traffic. In this case, a non-blocking warning may be necessary, although there has been some progress in norming captive portals. We only cover the auditing phase, but give no mechanism for private reporting of non-included SCTs. Meiklejohn et al.'s SoK [41] emphasizes that a report needs to include *actionable and concrete evidence of logs misbehavior* while preserving the privacy of the reporter, for example submitting the non-included certificates reveal browsing history. Even a zero-knowledge proof of non-inclusion [24], where the auditor proves in zero knowledge that it has a valid SCT and a valid proof of non-inclusion by showing all leaves between the two timestamps [24], is not sufficient to prove that the log integrity failed. Concrete and actionable reports remain an open problem.

ACKNOWLEDGMENTS

We thank Ryan Lemkuhl for a productive discussion about the state of the art of PIR protocols in the context of SCT auditing. We are also thankful for Samir Menons comments on efficient instantiation of YPIR for our usecase. We used Grammarly to spell-check this paper.

REFERENCES

[1] 2023. Apple's Certificate Transparency policy. https://support.apple.com/en-us/103214.
[2] 2024. Chrome Certificate Transparency Policy. https://googlechrome.github.io/CertificateTransparency/ct_policy.html.
[3] 2024. Have I been pwned? https://haveibeenpwned.com/.
[4] 2024. Live Caller ID Lookup Example. https://github.com/apple/live-caller-id-lookup-example/.
[5] David Adrian. 2024. Post-quantum cryptography is too damn big. https://dadrian.io/blog/posts/pqc-signatures-2024.
[6] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. 2021. Communication-Computation Trade-offs in PIR. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 1811–1828.
[7] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 962–979. https://doi.org/10.1109/SP.2018.00062
[8] Jean-Philippe Aumasson and Daniel J. Bernstein. 2012. SipHash: A Fast Short-Input PRF. In *INDOCRYPT 2012 (LNCS, Vol. 7668)*, Steven D. Galbraith and Mridul Nandi (Eds.). Springer, Berlin, Heidelberg, 489–508. https://doi.org/10.1007/978-3-642-34931-7_28
[9] Richard Barnes. 2017. STH Discipline & Security Considerations. https://mailarchive.ietf.org/arch/msg/trans/Zm4NqyRc7LDsOtV56EchBIT9r4c/.
[10] Mihir Bellare and Daniele Micciancio. 1997. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In *EUROCRYPT'97 (LNCS, Vol. 1233)*, Walter Fumy (Ed.). Springer, Berlin, Heidelberg, 163–192. https://doi.org/10.1007/3-540-69053-0_13
[11] David Benjamin, Devon O'Brien, and Bas Westerbaan. 2024. *Merkle Tree Certificates for TLS.* Internet-Draft draft-davidben-tls-merkle-tree-certs-03. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-davidben-tls-merkle-tree-certs/03/ Work in Progress.
[12] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, Heidelberg, 868–886. https://doi.org/10.1007/978-3-642-32009-5_50
[13] Alexander Burton, Samir Jordan Menon, and David J. Wu. 2024. Respire: High-Rate PIR for Databases with Small Records. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 1463–1477. https://doi.org/10.1145/3658644.3690328
[14] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *ACNS 21International Conference on Applied Cryptography and Network Security, Part I (LNCS, Vol. 12726)*, Kazue Sako and Nils Ole Tippenhauer (Eds.). Springer, Cham, 460–479. https://doi.org/10.1007/978-3-030-78372-3_18
[15] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private Information Retrieval. In *36th FOCS*. IEEE Computer Society Press, 41–50. https://doi.org/10.1109/SFCS.1995.492461
[16] Russ Cox. 2019. Tiling a Log. https://research.swtch.com/tlog#tiling_a_log.
[17] Al Cutter and Martin Hutchinson. 2024. Introducing Trillian Tessera. transparency.dev presentation. https://transparency.dev/summit2024/tesseratalk.html.
[18] Al Cutter, Martin Hutchinson, and Filippo Valsorda. 2024. The Static Certificate Transparency API. The Community Cryptography Specification Project. c2sp.org/static-ct-api@v1.0.0.
[19] Rasmus Dahlberg, Tobias Pulls, Tom Ritter, and Paul Syverson. 2021. Privacy-Preserving & Incrementally-Deployable Support for Certificate Transparency in Tor. *PoPETs* 2021, 2 (April 2021), 194–213. https://doi.org/10.2478/popets-2021-0024
[20] Alex Davidson, Gonçalo Pestana, and Sofía Celi. 2023. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *PoPETs* 2023, 1 (Jan. 2023), 365–383. https://doi.org/10.56553/popets-2023-0022
[21] Joe DeBlasio. 2021. Opt-out SCT Auditing in Chrome. Google Doc. https://docs.google.com/document/d/16G-Q7iN3kB46GSW5b-sfH5MO3nKSYyEb77YsM7TMZGE/.
[22] Joe DeBlasio. 2024. Certificate Transparency Log Policy. https://groups.google.com/a/chromium.org/g/ct-policy/c/nuJOpwj06QA.
[23] eranm@chromium.org. 2015. Certificate Transparency: Audit logs by checking SCTs for inclusion. Chromium Issue Tracker. https://issues.chromium.org/issues/41186110.
[24] Saba Eskandarian, Eran Messeri, Joseph Bonneau, and Dan Boneh. 2017. Certificate Transparency with Privacy. *PoPETs* 2017, 4 (Oct. 2017), 329–344. https://doi.org/10.1515/popets-2017-0052
[25] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144
[26] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. 2023. Private web search with Tiptoe. In *Proceedings of the 29th symposium on operating systems principles*. 396–416.
[27] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905.
[28] Jay Hou. 2024. Tile-Based Transparency Logs. https://transparency.dev/articles/tile-based-logs/.
[29] Dennis Jackson. 2024. *Abridged Compression for WebPKI Certificates.* Internet-Draft draft-ietf-tls-cert-abridge-02. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-tls-cert-abridge/02/ Work in Progress.
[30] Dennis Jackson. 2024. The design space between CT and KT. transparency.dev presentation. https://transparency.dev/schedule/ctkt.html.
[31] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. 2019. Revisiting user privacy for certificate transparency. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 432–447.
[32] Dana Keeler. 2025. Certificate Transparency is now enforced in Firefox on desktop platforms starting with version 135. https://groups.google.com/a/mozilla.org/g/dev-security-policy/c/OagRKpVirsA/m/Q4c89XG-EAAJ.
[33] Ben Laurie, Adam Langley, and Emilia Kasper. 2013. Certificate Transparency. RFC 6962. https://doi.org/10.17487/RFC6962
[34] Ryan Lehmkuhl, Alexandra Henzinger, and Henry Corrigan-Gibbs. 2025. Distributional Private Information Retrieval. *Cryptology ePrint Archive* (2025).
[35] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz. 2024. Hintless Single-Server Private Information Retrieval. In *CRYPTO 2024, Part IX (LNCS, Vol. 14928)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 183–217. https://doi.org/10.1007/978-3-031-68400-5_6
[36] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. 2024. Hintless single-server private information retrieval. In *Annual International Cryptology Conference*. Springer, 183–217.
[37] Google LLC. 2025. Certificate Transparency Log Policy. https://googlechrome.github.io/CertificateTransparency/log_policy.html.
[38] Google LLC and Internet Security Research Group. 2025. Trillian Tessera. https://github.com/transparency-dev/trillian-tessera.

[39] Wouter Lueks and Ian Goldberg. 2015. Sublinear Scaling for Multi-Client Private Information Retrieval. In *FC 2015 (LNCS, Vol. 8975)*, Rainer Böhme and Tatsuaki Okamoto (Eds.). Springer, Berlin, Heidelberg, 168–186. https://doi.org/10.1007/978-3-662-47854-7_10
[40] Francois Marier. 2022. [Security] Add support for Certificate Transparency in Brave. https://github.com/brave/brave-browser/issues/22482.
[41] Sarah Meiklejohn, Joe DeBlasio, Devon O'Brien, Chris Thompson, Kevin Yeo, and Emily Stark. 2022. SoK: SCT Auditing in Certificate Transparency. *PoPETs* 2022, 3 (July 2022), 336–353. https://doi.org/10.56553/popets-2022-0075
[42] Samir Jordan Menon and David J Wu. 2022. Spiral: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 930–947.
[43] Samir Jordan Menon and David J. Wu. 2024. YPIR: High-Throughput Single-Server PIR with Silent Preprocessing. In *USENIX Security 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association.
[44] Chromium Projects. 2024. Building a Deployable Post-quantum Web PKI. https://www.chromium.org/Home/chromium-security/post-quantum-pki-design/.
[45] SEAL 2023. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..
[46] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. 2019. Protecting accounts from credential stuffing with password breach alerting. In *USENIX Security 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 1556–1571.
[47] Chris Thompson. 2020. Retrying and persisting SCT audit reports. Google Doc. https://docs.google.com/document/d/1YTUzoG6BDF1QIxosaQDp2H5IzYY7_fwH8qNJXSVX8OQ/.
[48] transparency.dev. 2018. Trillian: General Transparency. Github repository. https://github.com/google/trillian.
[49] David Wagner. 2002. A Generalized Birthday Problem. In *CRYPTO 2002 (LNCS, Vol. 2442)*, Moti Yung (Ed.). Springer, Berlin, Heidelberg, 288–303. https://doi.org/10.1007/3-540-45708-9_19
[50] Bas Westerbaan. 2021. Sizing up post-quantum signatures. https://blog.cloudflare.com/sizing-up-post-quantum-signatures/.
[51] Bas Westerbaan. 2024. Revocation Behaviour of Let's Encrypt. https://github.com/davidben/merkle-tree-certs/issues/41#issuecomment-2245218105.
[52] Bas Westerbaan and Luke Valenta. 2024. A look at the latest post-quantum signature standardization candidates. https://blog.cloudflare.com/another-look-at-pq-signatures/.

A SLOW-ISSUANCE CERTIFICATES

A.1 Merkle Tree Certificates

Merkle Tree Certificates (MTC) [11] starts with the slow issuance idea, and doubles down on it: it does not just work out the details, but it also makes bigger changes to the architecture of the PKI, see Figure 3. To obtain a certificate in MTC, the *authenticating party*, who operates a TLS server, requests one with the CA in advance (1). In MTC a certificate is split into an *assertion* and a *proof*. An informal example of an*assertion* is 'public key P is trusted to do TLS for domain `example.com`'— it is the certificate without the signature of the CA. On a set interval, say once an hour, the CA publishes a new *batch* of *assertions* it issued. For each batch, the CA also computes a Merkle tree. The assertions in each batch have the same implicit validity, say two weeks. Together with the latest batch, the CA also publishes a signature on the Merkle tree heads of the currently valid batches. Now the authenticating party can collect its certificate (3): it is formed by combining the assertion with a Merkle tree *inclusion proof* of the assertion into the batch. A relying party (browser) cannot make sense of this certificate without knowing the batch tree heads. This is where the *transparency service* (TS) comes in. The TS mirrors (2) the batches and signature of the CA; checks them for consistency; and passes the batch tree heads to the relying party (5). A TS could be run by a browser vendor or another third party. Now, when connecting to the TLS server (6), the relying party for each CA advertises the sequence number of the latest batch it knows the tree head for. The authenticating party

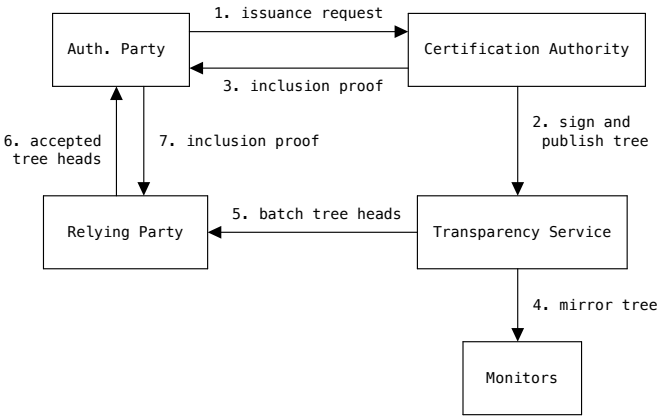

Figure 3: Parties and flow in a Merkle Tree certificate deployment [11]

returns either a MTC certificate if an appropriate one is available (7) or falls back to X.509. Finally, different TS mirroring the same CA are checked for consistency by Monitors (4).