AccuRevoke: Enhancing Certificate Revocation with Distributed Cryptographic Accumulators

Munshi Rejwan Ala Muid   Taejoong Chung   Thang Hoang

Virginia Tech
{*munshira, tijay, thanghoang*}*@vt.edu*

Abstract—**Certificate revocation is essential for maintaining the security of the Public Key Infrastructure (PKI), ensuring that compromised or untrustworthy certificates are invalidated promptly. Traditional revocation mechanisms like Certificate Revocation Lists (CRLs) and the Online Certificate Status Protocol (OCSP) face significant challenges, including scalability issues, high bandwidth consumption, privacy concerns, and reliance on centralized infrastructure that can become points of failure.**

In this paper, we introduce AccuRevoke, a novel revocation scheme that leverages cryptographic accumulators and edge computing to address these challenges effectively. AccuRevoke enables clients to verify the revocation status of certificates efficiently without the need to contact Certificate Authorities (CAs) directly for each validation. By utilizing distributed accumulators and threshold cryptography, AccuRevoke ensures authenticity and integrity of revocation information, even when responses are generated by third-party Edge Compute Providers (ECPs).

Our scheme significantly reduces bandwidth consumption by providing compact revocation proofs—approximately 21 bytes for membership proofs and 61 bytes for non-membership proofs—which are substantially smaller than traditional OCSP responses. To further optimize performance, especially in generating non-membership witnesses, we employ GPU acceleration, achieving considerable improvements in processing times.

We compare AccuRevoke with existing revocation mechanisms, demonstrating advantages in bandwidth efficiency, reliability, auditability, and potential enhancements in privacy. Our evaluation shows that AccuRevoke offers a scalable and practical solution for revocation checking, improving the security and performance of TLS/PKI deployments. We plan to open-source our design and implementation to facilitate adoption and encourage further research in this area.

1. Introduction

Transport Layer Security (TLS) and the Public Key Infrastructure (PKI) are foundational to secure Internet communications, enabling authenticated and encrypted interactions between clients and servers. A critical component of this ecosystem is the ability to revoke certificates that have been compromised [43] or are no longer trustworthy [4]. Effective certificate revocation mechanisms are essential for maintaining the integrity and security of TLS/PKI deployments. However, disseminating revocation information at scale poses significant challenges due to the vast number of certificates and the associated performance overhead.

Traditional methods for certificate revocation, such as Certificate Revocation Lists (CRLs) [9] and the Online Certificate Status Protocol (OCSP) [39], suffer from scalability and latency issues. CRLs can become excessively large as the number of revoked certificates grows, leading to increased bandwidth consumption and delayed updates. OCSP provides real-time revocation status but introduces latency due to the need for clients to query OCSP responders for each certificate validation. To mitigate these challenges, many Certificate Authorities (CAs) have leveraged third-party services for distributing revocation information.

Content Delivery Networks (CDNs) have historically been employed to enhance the scalability and performance of revocation information dissemination. For example, Let's Encrypt utilizes Akamai's CDN services to distribute its CRLs and OCSP responses [25]. Similarly, Mozilla's CRLite [28], which compresses revocation information using Bloom filters, relies on Akamai's CDN to distribute these filters to clients [21].

While CDNs improve availability and distribution efficiency, their function as reverse proxies introduces inherent limitations. Since CDNs forward requests to the origin server, the origin server (i.e., the CA's infrastructure) must maintain near-perfect uptime–which, in practice, often falls short [11]. If the CA's origin server becomes unavailable due to outages or maintenance, the CDN cannot serve OCSP responses unless the data is cached, potentially leaving clients unable to verify certificate revocation status. This dependency implies that, despite leveraging CDNs, the system's availability remains tightly coupled with the origin server's reliability.

To address this limitation, some CAs have adopted delegated OCSP responder certificates [31], allowing CDNs to generate and sign OCSP responses directly without needing to contact the origin server for each request. While this approach improves availability by reducing dependency on the origin server, it introduces new trust assumptions into the PKI model. Specifically, it extends the CA's trust boundary to include third-party CDNs, which are not traditional

2025 IEEE Symposium on Security and Privacy (SP) | 979-8-3315-2236-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/SP61157.2025.00129

authentication authorities. If a CDN is compromised, it could issue fraudulent OCSP responses, and clients would be unable to detect such tampering due to the lack of mechanisms for independent verification. This shift raises concerns about the integrity and security of the revocation process, as it relies on entities outside the established trust framework of the PKI.

In recent years, CDNs have evolved into *edge computing providers* [40], offering computational capabilities at the edge of the network. Platforms such as *Cloudflare Workers* [16] and *Fastly Compute@Edge* [20] not only distribute content efficiently but also perform computations close to the clients. These edge computing providers, being also CDNs, can compute and deliver results rapidly, combining the benefits of content distribution and edge computation. This evolution opens new opportunities to rethink how revocation information is disseminated and verified.

We posit that leveraging edge computing can address the limitations of traditional revocation mechanisms. By deploying computational logic at the edge, third parties can participate in secure protocols without compromising trust. Specifically, we can empower edge servers to collaboratively generate cryptographic proofs of certificate revocation status, enhancing both performance and security.

In this paper, we propose a novel revocation system, AccuRevoke, that utilizes *cryptographic accumulators* [10] in conjunction with edge computing. A cryptographic accumulator is a compact data structure that allows one to succinctly represent a set of elements and efficiently prove whether an element is a member (revoked) or a non-member (non-revoked) of the set. Importantly, the size of the accumulator remains constant regardless of the number of elements it contains, making it an attractive option for handling large revocation datasets.

However, traditional accumulator schemes typically assume a trusted first party (accumulator manager) responsible for maintaining the accumulator and generating proofs, which is not ideal in a distributed environment involving third-party entities. By harnessing edge computing, we can distribute the computational tasks required for witness generation among multiple edge servers. This distribution not only improves performance by leveraging the computational power of edge servers but also enhances security by ensuring that no single entity holds the entire secret required to generate valid proofs.

Our contributions are threefold:

- **Design of a Distributed and Auditable Accumulator**: We introduce an accumulator framework where multiple edge compute providers collaboratively generate membership and non-membership proofs. Each provider holds a share of the secret key—using techniques like Shamir's Secret Sharing—rather than the entire secret. This ensures that no single compromised edge server can generate valid proofs on its own. By working together, the edge servers can generate the necessary proofs *without* direct communication with the CA. Our scheme also enables clients and the CA to audit the authenticity of the proofs, ensuring that any incorrect information provided by third parties can be detected and mitigated.
- **Leveraging Edge Computing for Scalability and Performance**: By utilizing the computational capabilities of edge servers, our system reduces latency and improves the responsiveness of revocation checks. Clients can obtain proofs from nearby edge servers, minimizing network delays. The ability to process computations at the edge allows the system to scale effectively, handling a large number of revocation queries without overloading central servers.
- **Performance Optimization through Parallelism**: Recognizing the computational challenges associated with accumulators, particularly for generating non-membership proofs, we optimize our accumulator design for parallel computation. By leveraging the inherent parallelism in cryptographic operations, we implement our scheme on GPUs at the edge. Our implementation demonstrates that the time to generate non-membership proofs is significantly reduced. This addresses a common criticism of accumulator-based systems concerning their scalability and performance.

We validate our proposed system through rigorous analysis and experimental evaluation. Our results indicate that the distributed accumulator not only enhances the security and trustworthiness of revocation information dissemination but also achieves practical performance suitable for real-world deployment. By integrating edge computing into the PKI infrastructure, we provide a pathway for more secure and efficient certificate revocation mechanisms.

To foster reproducibility and further research into improving the TLS revocation ecosystem, we publicly release our implementation code to the research community at

https://accurevoke.netsecurelab.org

2. Background

In this section, we provide a brief background on certificates, detail the protocols for certificate revocation used in practice, and cryptographic foundations underlying our proposed scheme, AccuRevoke.

2.1. Certificates

Digital certificates are fundamental components of web security, serving to establish a trusted association between entities—typically domain names—and their cryptographic public keys. Issued by Certificate Authorities (CAs), these certificates enable clients to verify the identity of servers and establish secure communications over the Internet.

Certificates are organized into a hierarchical chain of trust, starting from a self-signed root certificate, followed by intermediate certificates, and culminating in the leaf certificate presented by the server during the TLS handshake. Each certificate in the chain is validated by verifying the digital signature of its issuer, ensuring that it was issued by a trusted authority and has not been tampered with. Validation also involves checking the certificate's validity period, confirming that it has not expired, and determining whether it has been revoked prior to its expiration date.

The de facto standard for web certificates is defined by the X.509 specification, which outlines the structure and encoding of certificates using Abstract Syntax Notation One (ASN.1) [9], [19].

2.2. TLS Certificate Revocation in Practice

Numerous solutions have been proposed to enhance the revocation system [12], [13], [37], [41], [42]; however, in practice, three main revocation protocols are predominantly used, encompassing *all* revocation information.

2.2.1. Certificate Revocation Lists (CRLs). CRLs are ASN.1-encoded files maintained by CAs that contain a list of revoked certificates [9]. Each entry in a CRL includes the certificate's serial number, the revocation timestamp, and the revocation reason. CAs include a URL in the certificate's CRL Distribution Points extension, allowing clients to locate and download the CRL associated with a certificate.

CAs are responsible for publishing updated CRLs regularly, even if no new certificates have been revoked, to ensure that the validity period remains current.

However, CRLs are often criticized for their inefficiency. Clients must download the entire CRL even if they are only interested in the revocation status of a single certificate. This can result in significant bandwidth consumption and increased latency during certificate validation. Prior work has demonstrated that CRLs can be quite large, with some exceeding 76 MB in size [30].

2.2.2. Online Certificate Status Protocol (OCSP). To address the limitations of CRLs, the OCSP [39] enables clients to obtain the real-time revocation status of specific certificates. Clients use the Authority Information Access (AIA) extension in certificates to locate the OCSP responder's URL. An OCSP request includes the certificate's serial number and issuer information, allowing CAs to verify the request. The responder provides a signed response indicating the certificate's status (Good, Revoked, or Unknown) and validity period. OCSP reduces data transfer compared to CRLs but introduces additional latency due to the extra network request during the TLS handshake. It also raises privacy concerns, as OCSP requests can reveal clients' browsing habits to CAs.

To address these issues, OCSP Stapling [3] was introduced, allowing web servers to periodically fetch OCSP responses from the CA and include them in the TLS handshake. Additionally, the OCSP Must-Staple extension was designed to enforce the use of OCSP stapling. However, studies have shown that only a small fraction of servers support OCSP stapling, many of which are misconfigured, and an even smaller proportion of certificates include the Must-Staple extension [11].

2.2.3. CRLite. Unlike traditional methods where clients retrieve revocation status directly from CAs, CRLite [28] leverages Mozilla—a third-party client entity—to collect revocation information from all CAs and push it to the client browsers. Mozilla aggregates revocation data by monitoring Certificate Transparency logs and collecting revocation information from various CAs. The aggregated data is then compressed using CRLite's Bloom filter structures and distributed to Firefox users through browser updates.

By integrating CRLite into Firefox, clients can perform revocation checks locally without additional network requests during the TLS handshake.

2.3. Roles of CDNs in Dissemination of Revocation Status

CAs commonly use CDNs to host and distribute their CRLs and facilitate OCSP responses; for example, Let's Encrypt and Certum rely on Akamai's CDN services, while Sectigo, ComodoCA, and GlobalSign utilize Cloudflare for their OCSP endpoints [25]. Traditionally, CDNs act as reverse proxies for OCSP responses, forwarding client requests to the CA's origin server. Due to security concerns, CAs typically do not allow CDNs to sign OCSP responses on their behalf. This reliance on the origin server introduces a single point of failure: if the origin server experiences downtime, the CDN cannot serve OCSP responses, leaving clients unable to verify certificate revocation status [11].

To improve availability, some CAs (e.g., TrustCor or WISeKey) use delegated certificates [31] to allow CDNs to sign OCSP responses *without* accessing the CA's main signing keys. While this reduces dependency on the origin server, it extends the CA's trust boundary to third-party CDNs, raising security concerns about potential misuse or compromise of delegated keys. Additionally, response sizes increase due to the inclusion of the delegate certificate. A recent solution, CRLite [28], compresses revocation information using Bloom filters, in practice, Mozilla integrates CRLite into Firefox and distributes filter cascades or delta updates through Akamai's CDN [21]. While CAs can audit these filters, clients rely on pre-generated data without the ability to independently verify its completeness or correctness, introducing a trust assumption on Mozilla.

These methods highlight a trade-off between availability and trust. Delegating signing authority to CDNs enhances availability but raises security concerns; client-side solutions like CRLite depend on third parties and limit clients' ability to verify revocation data independently.

To address these challenges—ensuring availability, authenticity, and integrity even during origin (i.e., , CA) server outages—and to harness the potential of edge computing providers, we propose a new approach. By leveraging cryptographic techniques such as Secure Multiparty Computation (MPC) and cryptographic accumulators, we design a distributed and secure revocation system.

The following background introduces these cryptographic concepts, which are foundational to our proposed solution.

Algorithm 1 Shamir Secret Sharing Scheme

```
$(\langle\alpha\rangle_1, \ldots, \langle\alpha\rangle_\ell) \leftarrow \mathsf{SSS.Create}(\alpha, t-1)$: Create $l$-private shares of $\alpha$
1: $(a_1, \ldots, a_{t-1}) \xleftarrow{\$} \mathbb{F}_p$
2: for $i = 1, \ldots, \ell$ do
3:     $\langle\alpha\rangle_i \leftarrow \alpha + \sum_{j=1}^{t-1} a_j \cdot x_i^j$
4: return $(\langle\alpha\rangle_1, \ldots, \langle\alpha\rangle_\ell)$
```

Algorithm 2 Finding the Share of the Inverse

```
$\langle s\rangle \leftarrow \mathsf{InvSharesElement}(\langle\alpha\rangle, x)$ : Compute share of $(x+\alpha)^{-1}$
1: $\langle r\rangle \leftarrow$ Share of pre-computed random element $r$
2: $\langle z\rangle \leftarrow x + \langle\alpha\rangle$
3: $\langle p\rangle \leftarrow \langle r\rangle \boxtimes \langle z\rangle$
4: $p \leftarrow$ open $\langle p\rangle$
5: $\langle s\rangle \leftarrow p^{-1} \cdot \langle r\rangle$
6: return $\langle s\rangle$
```

2.4. Secure Multiparty Computation

Secure multi-party computation (MPC) allows multiple parties to jointly compute a function over their inputs while keeping those inputs private. A key cryptographic primitive in MPC is Shamir Secret Sharing, which provides a secure method for distributing and reconstructing secrets. The following section details its underlying mechanism.

2.4.1. Shamir Secret Sharing. We recall shamir secret sharing [38], where a secret is divided into multiple shares. No less than threshold number of shares reveal any information about the secret.

In Algorithm 1, we produce l shares of the secret value α based on a polynomial of degree $t-1$, where t denotes the minimum number of data points required to reconstruct the secret. To construct this polynomial, we randomly select $t-1$ coefficients from the finite field $\mathbb{F}_p$. These coefficients, together with the secret α as the constant term, fully define the polynomial. Each share is then obtained by evaluating the polynomial at unique, non-zero identities i in $\{1, 2, \ldots, l\}$ This setup ensures that any subset of t shares suffices to recover the secret, while subsets smaller than t reveal no information about α.

When a secret is shared among multiple parties using Shamir secret sharing scheme, Lagrange interpolation is a fundamental method used to reconstruct a secret from at least threshold number of shares. Each party holds a share, and given threshold or more distinct data points where the secret share is the value of the evaluation of the polynomial and the evaluation point is a public value which denotes the identity of the secret holder. The Lagrange polynomial $g(x)$ is computed as

$$g(x) \leftarrow \mathsf{LagrangeInterpolation}\left(\{(x_i, \langle\alpha\rangle_{x_i})\}_{i=1}^{t}\right)$$

Given t distinct data points, it returns a Lagrange polynomial $g(x)$ of degree at most $t-1$ where $g(x) = \sum_{i=1}^{t} \langle\alpha\rangle_{x_i} \cdot \prod_{\substack{1\le j\le t \\ j\neq i}} \frac{x-x_j}{x_i-x_j}$

2.4.2. Computation of Share of Multiplicative Inverse. Each participating party can efficiently and securely determine its share of the multiplicative inverse of an element using its own share of the secret.

In Algorithm 2, each party utilizes its secret share, denoted as $\langle\alpha\rangle$, to compute the share of the multiplicative inverse of a combined element, $x+\alpha$, where x is a public value and α is secret-shared. The resulting share of the inverse is denoted as $\langle s\rangle$.

The secure and efficient multiplication of shares, $\langle r\rangle$ and $\langle z\rangle$, denoted as $\boxtimes$ in Algorithm 2, leverages Beaver's trick [1]. Here, $\langle z\rangle$ represents the share of the element $x+\alpha$, and $\langle r\rangle$ represents the share of a random, precomputed element r. This multiplication is performed using Beaver multiplication triples (a, b, c), where $c = a \cdot b$ is precomputed during an offline phase.

The shares of Beaver's triple, $\langle a\rangle, \langle b\rangle, \langle c\rangle$, are generated and distributed among all parties in the MPC setup. The objective of this multiplication $\boxtimes$ is to compute the share of $r \cdot z$ for each party in an efficient and secure manner.

Each party computes:

$$\langle u\rangle = \langle r\rangle - \langle a\rangle,$$
$$\langle v\rangle = \langle z\rangle - \langle b\rangle,$$

using their respective shares. Then, the values u and v become public after opening. Utilizing these public values, u and v, along with their shares of Beaver triples, each party computes its share of $r \cdot z$ as:

$$\langle p\rangle = uv + u\langle b\rangle + v\langle a\rangle + \langle c\rangle.$$

When any party opens p, where $p = r \cdot z$, no party can infer any information about z because it is securely masked by r.

2.5. Accumulator

Cryptographic accumulators have seen substantial interest over the decades as a means of creating compact, binding commitments which we call the value of the accumulator from a set of elements.

2.5.1. Dynamic Universal (threshold) Secret-shared Distributed Accumulator. An accumulator that generates short membership and non-membership witnesses for verifying element inclusion or exclusion is called a universal accumulator. If it supports both the addition and deletion of elements, it is referred to as a dynamic accumulator.

A dynamic universal accumulator requires a secret key, known only to the accumulator manager, to securely add or remove elements and generate membership or non-membership witnesses. Building on this concept, a distributed accumulator employs (MPC) to share the secret key among multiple managers [22]. This approach ensures that no single entity has full control or knowledge of the secret key, thereby preserving privacy and security. Such distributed accumulators can operate in a threshold-based manner using Shamir's Secret Sharing, allowing collaborative operations such as element addition, deletion, and proof generation, even if some participants are unavailable. This ensures both resilience and functionality.

In this paper, we extend the concept of a Dynamic Universal (threshold) Secret-Shared Distributed Accumulator [22] by introducing a hybrid architecture with distinct roles for a master accumulator manager (first party i.e., CA) and multiple edge servers (servant accumulator managers). Unlike prior work [22] where all accumulator operations—including addition, deletion, and witness generation—were performed in a fully distributed manner, our design assigns specific tasks to two separate entities.

The CA, acting as the master accumulator manager, holds the secret key and is trusted to handle critical operations such as initializing the accumulator, securely managing element addition and deletion, and distributing secret key shares to the edge servers. By centralizing these tasks, the CA reduces operational complexity while ensuring secure and reliable management of the accumulator.

To decentralize control of witness generation, the CA delegates this responsibility to multiple edge servers. The edge servers, operating in an MPC setting, collaboratively generate membership and non-membership witnesses using the secret key shares provided by the CA. Importantly, the edge servers are restricted to witness generation alone, with no involvement in managing the accumulator or its elements. This division of responsibilities enhances both scalability and security, as no single edge server can independently produce a valid witness.

The final verification of membership or non-membership is conducted by the client (e.g., a web browser), which uses the generated witnesses to validate the inclusion or exclusion of a specified element. By segregating tasks between the CA and the edge servers, our architecture achieves a balance between trust centralization and secure distributed collaboration.

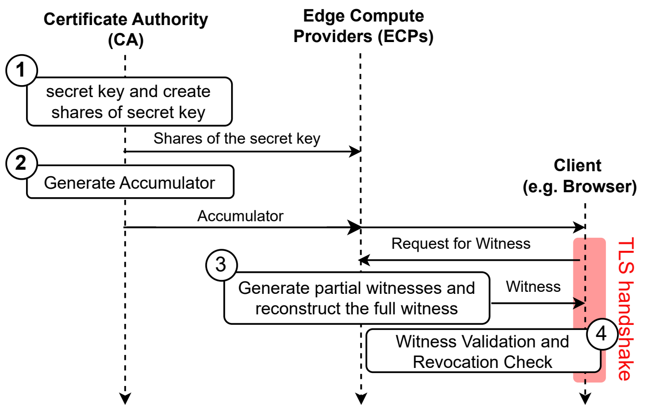
3. System

In this section, we present the architecture and components of our proposed revocation system, AccuRevoke, which leverages cryptographic accumulators to enhance the security and efficiency of certificate revocation in TLS/PKI environments. Our system is composed of three primary entities: the *First Party* (i.e., Certificate Authority), *Third Parties* (e.g., Edge Compute Providers), and *Clients* (e.g., web browsers). We also introduce the nomenclature and provide a detailed overview of the system's mathematical foundations.

3.1. System Components

An overview of the system architecture is depicted in Figure 1. Table 1 also summarizes the notation used in the AccuRevoke model and security analysis.

The CA generates the cryptographic accumulator and secret-shares the secret key with the Edge Compute Providers (ECP_i). The ECP_is collaboratively generate witnesses for certificates, which they can store in cache for efficient retrieval. Clients request the witnesses from multiple ECP_is to verify the validity of certificates.



Certificate Authority (CA)
Edge Compute Providers (ECPs)
Client (e.g. Browser)
1
secret key and create shares of secret key
Shares of the secret key
2
Generate Accumulator
Accumulator
Request for Witness
3
Generate partial witnesses and reconstruct the full witness
Witness
4
Witness Validation and Revocation Check
TLS handshake

Figure 1. The overview of AccuRevoke: ① The CA uses threshold secret sharing protocol to divide its secret key among ECPs, ensuring that no single ECP can generate a witness alone. (§3.1.1); ② The CA generates an accumulator of revoked certificates and distributes it to ECPs and clients. This accumulator is updated by the CA and distributed to ECPs and clients when a certificate is revoked. (§3.1.1); ③ During the TLS handshake (the TLS server is omitted in the figure for brevity), the client requests a witness from an ECP to perform revocation checking. ECPs collaboratively generate partial witnesses using their secret shares, combine them to reconstruct a full witness, and return it to the client. Note that this process does not require ECPs to communicate with the CA during operation unlike traditional OCSP. (§3.1.2); ④ Upon receiving the full witness, the client validates it and completes the revocation check, allowing the TLS handshake to conclude successfully. (§3.1.3)

TABLE 1. NOMENCLATURE

Symbol	Description
CA	Certificate Authority (First Party)
ECP_i	Edge Compute Provider i (Third Parties)
α	Accumulator's secret key (trapdoor)
pk_Λ	Public parameters of the accumulator
$\mathcal{X}$	Set of revoked certificate serial numbers
$\Lambda_\mathcal{X}$	Accumulator value for $\mathcal{X}$
x	Revoked certificate's serial number
y	Non-revoked certificate's serial number
G_1, G_2, G_T	Groups used
e	Non-degenerate bilinear map
$g_1, g_2, e(g_1, g_2)$	Generators of G_1, G_2 and G_T respectively
ℓ	Number of edge compute providers (ECP_i)
t	Threshold for secret sharing
$\langle\alpha\rangle_i$	Share of the secret key held by ECP_i
w_x	Membership Witness
(w_y, u_y)	Non-membership witness

3.1.1. First Party (Certificate Authority). The First Party, typically a CA, holds the authoritative responsibility for issuing, managing, and revoking digital certificates within the PKI ecosystem. The CA performs the following key functions:

- Key Generation: The CA generates the secret key α and public parameters pk_Λ.

- Accumulator Management: The CA initializes and updates the cryptographic accumulator $\Lambda_{\mathcal{X}}$ representing the set of revoked certificates using the secret key α.
- Secret Sharing: to enable distributed witness generation, the CA employs Threshold Secret Sharing (Shamir) scheme to distribute *shares* of the accumulator's secret key to multiple edge compute providers (ECP_i). This ensures that no single ECP_i possesses the entire secret, thereby enhancing the system's resilience against compromises.
- Accumulator Dissemination: The CA periodically publishes the current state of the accumulator to all registered ECPs and clients, ensuring that revocation information is up-to-date and accessible. Note that the accumulator size is very small, such as 21 bytes [27].

3.1.2. Third Parties (Edge Compute Providers). Edge Compute Providers (ECP_i) are computational entities responsible for generating and distributing witnesses. Their responsibilities include:

- Partial Witness Generation: Each ECP_i computes share as $\langle w_x \rangle_i$ or $(\langle w_y \rangle_i, u_y)$ for a certificate using their secret share $\langle \alpha \rangle_i$.
- Collaborative Witness Construction: The ECP_i collaboratively combine their shares of a witness to reconstruct full witness w_x or (w_y, u_y) for a certificate in a distributed setting. If a single ECP_i has t number of shares from other ECP_is, it can reconstruct the full witness providing resiliency and robustness. This process can be performed in advance for multiple certificates, and the resulting witnesses can be stored in cache for efficient retrieval.
- Witness Distribution: Upon receiving a request from a client, an ECP_i retrieves the full witness w_x or (w_y, u_y) from cache and sends it to the client.

3.1.3. Client (e.g., Web Browser). A client, typically a web browser, interacts with the revocation system to verify the validity of certificates during TLS handshakes. The client's responsibilities include:

- Witness Request: A client sends a request for the witness of a certificate with serial number x or y to an ECP_i. If the requested ECP_i is out of service, the client can look for another ECP_i.
- Witness Verification: A client verifies the received reconstructed (by ECP_i) witness w_x or (w_y, u_y) using public parameters. Optionally, the client may request witnesses from multiple ECP_i for redundancy but is not required to manage the collaboration process.

3.2. Threat Model

The security of the AccuRevoke system is paramount, given its role in maintaining the integrity of TLS/PKI communications. We adopt the Dolev-Yao adversary model [17] to analyze potential threats, where the adversary has full control over the communication channels and can attempt to compromise system components. The primary threats considered are as follows:

3.2.1. Compromised Third Parties (Edge Compute Providers). An adversary may compromise one or more ECP_i, gaining access to their shares $\langle \alpha \rangle_i$ of the accumulator's secret key. If sufficient ECP_i are compromised, the adversary might attempt to generate fraudulent witnesses. We mitigate this by (1) employing Shamir's Secret Sharing with a threshold t, our system ensures that no single ECP_i possesses the entire secret key. A threshold number of ECP_i must collaborate to generate a valid witness, making it significantly harder for an adversary to compromise the system by targeting individual ECP_i. The system's resilience parameter t can be also tuned to require a minimum number of ECP_i to generate witnesses. This allows system administrators to balance between performance and security based on the threat landscape.

3.2.2. Integrity of the Accumulator Value. We assume that clients can securely obtain the latest accumulator value $\Lambda_{\mathcal{X}}$ from the CA. This can be achieved by leveraging existing infrastructures such as Certificate Transparency Logs (CTLogs) [7], which provide an append-only, publicly auditable log of certificates and associated data. The accumulator value can be published in these logs or distributed via secure channels.

Since the accumulator value is relatively small (e.g., a couple of bytes), it can be efficiently distributed and stored by clients. By ensuring the integrity and authenticity of $\Lambda_{\mathcal{X}}$, clients can trust that the verification process is based on the correct and up-to-date state of the revocation information. Clients can verify the authenticity of the accumulator value using digital signatures from the CA or by cross-referencing with trusted public logs.

This assumption allows us to focus on the security of the witness generation and verification processes without needing to consider attacks that tamper with the accumulator value during transmission. By leveraging trusted mechanisms for obtaining $\Lambda_{\mathcal{X}}$, we mitigate the risk of accumulator value compromises.

Also, it is computationally infeasible for an adversary to compute the value of the accumulator without the knowledge of the secret key α even if it knows the set of elements accumulated $\mathcal{X}$ and the current accumulator value $\Lambda_{\mathcal{X}}$.

4. AccuRevoke: Protocols and Implementation

In this section, we introduce the security goals of AccuRevoke and the detailed workings of AccuRevoke by focusing on the interactions and responsibilities of the three main entities and explain the mathematical foundations that enable secure and efficient revocation.[1]

4.1. Security Goals

The primary security goals of AccuRevoke are:

1. For formal proof of AccuRevoke, please refer to the §A.

Algorithm 3 CA Key Generation and Setup

```
    Input: Security parameter $\kappa$, size of the domain of elements $q$, threshold $t$, total number of $ECP$s $\ell$.
    Output: Public parameters $pk_\Lambda$, secret shares $\{\langle\alpha\rangle_i\}$.
1:  Generate bilinear pairing parameters $BG$ as $BG = (p, G_1, G_2, G_T, e, g_1, g_2)$ using security parameter $\kappa$ and size of the domain of elements $q$.
2:  Choose a secret key $\alpha \xleftarrow{\$} \mathbb{Z}_p^*$.
3:  Compute $h \leftarrow g_2^\alpha$.
4:  Set public parameters $pk_\Lambda = (BG, h)$.
5:  Use Shamir's Secret Sharing to divide $\alpha$ into shares $\{\langle\alpha\rangle_i\}$ with threshold $t$.
6:  Securely distribute $\langle\alpha\rangle_i$ to each $ECP_i$.
```

- **Authenticity and Integrity**, which ensure that clients receive authentic and untampered revocation information, preventing attackers from misleading clients about the revocation status of certificates.
- **Confidentiality of Secret Keys**, which protects the accumulator's secret key α and its shares $\langle\alpha\rangle_i$ from unauthorized access, ensuring that only authorized entities can generate valid witnesses.
- **Robustness against Compromised Entities**, which maintians security even if some ECPs are compromised, preventing attackers from generating fraudulent witnesses or disrupting the system.
- **Availability**, which ensures that clients can reliably obtain revocation information and verify certificates, even in the presence of network failures or malicious entities.

4.2. Overview of the AccuRevoke Protocol

AccuRevoke leverages bilinear pairing based dynamic universal (threshold) secret-shared distributed cryptographic accumulator to provide a scalable and secure certificate revocation mechanism. We can consider various types of cryptographic accumulators, such as RSA accumulators and Merkle Tree accumulators. However, we chose to utilize an elliptic curve accumulator due to its efficiency and smaller proof sizes. Elliptic curve accumulators offer shorter element representations and faster computation times compared to other types, which is essential for handling large-scale revocation lists in practical deployments.

The protocol consists of the following key phases:

1) **Setup and Initialization**: The CA generates cryptographic parameters, including a secret key and public parameters for the accumulator. The secret key is divided and securely distributed to multiple ECPs using a threshold secret sharing scheme.
2) **Accumulator Management**: The CA maintains an accumulator that represents the set of *revoked certificates*. The CA might construct the accumulator using non-revoked certificates, but having revoked certificates is more advantageous because the number of revoked certificates is significantly smaller than the total number of issued certificates [26]. Accumulating only revoked certificates reduces the frequency of accumulator updates and minimizes witness updates, as revocations occur less often than new certificate issuances.
   Certificates rarely transition from revoked to non-revoked status, and the performance impact of such updates is minimal [32]. Furthermore, certificates automatically expire at the end of their validity period, rendering them invalid without requiring explicit revocation or removal from the accumulator. This further reduces the need for updates when certificates expire, improving efficiency.
   By minimizing updates to the accumulator, witnesses remain valid for longer durations, enhancing caching efficiency for ECPs and clients. When a certificate is revoked, the CA updates the accumulator and distributes the updated version to ECPs and clients. However, new certificate issuances do not affect the accumulator, as it is based solely on revoked certificates.
3) **Witness Generation and Distribution**: ECPs collaboratively compute witnesses for certificates using their secret shares. They precompute and cache these witnesses to serve client requests efficiently.
4) **Client Verification**: Clients request witnesses from ECPs when verifying a certificate during a TLS handshake. Upon receiving a witness, the client verifies its validity using the public parameters and the accumulator.

Algorithm 4 Accumulator Evaluation

```
    Input: Secret Key $\alpha$, list of revoked serial numbers $\mathcal{X}$, public parameters $pk_\Lambda$.
    Output: Accumulator $\Lambda_\mathcal{X}$.
1:  parse the public parameters $pk_\Lambda$ as $(p, G_1, G_2, G_T, e, g_1, g_2, h)$
2:  for each serial number $x \in \mathcal{X}$ do
3:      $\Lambda_\mathcal{X} \leftarrow \Lambda_\mathcal{X}^{(x+\alpha)}$
4:  Disseminate $\Lambda_\mathcal{X}$ to all $ECP_i$ and clients.
```

4.3. First Party (i.e., CA) Operations

The CA plays a central role in (1) initializing the system and (2) managing the accumulator. Note that the CA is not involved with the generation of witnesses.

4.3.1. Key Generation. To initialize the system, the CA generates the cryptographic keys and parameters required for the accumulator. In Algorithm 3, the CA generates cryptographic parameters necessary for the accumulator's operation. By using bilinear pairings [8], the system benefits from efficient verification and security properties. The secret key α is critical for accumulator updates and witness generation; by employing threshold secret sharing scheme, the system ensures that no single ECP can reconstruct α, enhancing security against compromise. The CA is also tasked with defining the appropriate threshold t value.

Algorithm 5 Accumulator Update

```
    **Input**: Current accumulator $\Lambda_{\mathcal{X}}$, certificate serial number $x$, operation $\mathsf{op} \in \{\mathsf{add}, \mathsf{delete}\}$.
    **Output**: Updated accumulator $\Lambda_{\mathcal{X}'}$.
1: if $\mathsf{op} = \mathsf{add}$ then
2:     $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_{\mathcal{X}}^{(x+\alpha)}$.
3:     Update $\mathcal{X}' \leftarrow \mathcal{X} \cup \{x\}$.
4: else if $\mathsf{op} = \mathsf{delete}$ then
5:     $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_{\mathcal{X}}^{(x+\alpha)^{-1}}$.
6:     Update $\mathcal{X}' \leftarrow \mathcal{X} \setminus \{x\}$.
7: Disseminate $\Lambda_{\mathcal{X}'}$ to all $ECP_i$ and clients.
```

4.3.2. Accumulator Initialization. The CA evaluates the accumulator $\Lambda_{\mathcal{X}}$ representing the set of revoked certificates $\mathcal{X}$. Algorithm 4 details how the accumulator is evaluated initially with the set of revoked serial numbers.

4.3.3. Accumulator Update. The CA maintains the accumulator $\Lambda_{\mathcal{X}}$ representing the set of revoked certificates $\mathcal{X}$. The accumulator is updated whenever a certificate is revoked or a revoked certificates changes to non-revoked, using operations that depend on the secret key α. Algorithm 5 details how the accumulator is updated; when a certificate is revoked, the accumulator is exponentiated by the element $(x + \alpha)$, effectively incorporating the new certificate. For the transition of a certificate's status from revoked to non-revoked, the accumulator is exponentiated by the multiplicative inverse $(x+\alpha)^{-1}$ of the element $(x+\alpha)$, removing the certificate from the accumulator.

4.4. Third Parties (ECPs) Operations

The ECPs are responsible for collaboratively generating witnesses and efficiently serving them to clients.

4.4.1. Partial Membership Witness Computation. Each ECP_i computes partial membership witnesses using their secret share $\langle\alpha\rangle_i$ without requiring knowledge of the entire set of the revoked certificates $\mathcal{X}$ or the secret key α. In Algorithm 6, each ECP computes a partial witness by raising the accumulator to the exponent r where r is the evaluation of the lagrange polynomial $g(x)$ at 0. To compute the lagrange polynomial $g(x)$ per Shamir secret share, ECP_i only needs its share of the multiplicative inverse of $(x+\alpha)$ which is $\langle s\rangle_i$ and identity i. The computation is secure because $\langle\alpha\rangle_i$ is only a share of the secret key, and individual ECPs cannot reconstruct α alone. This approach contrasts with traditional accumulators where witness generation often requires central coordination or full knowledge of the accumulator set [10].

Algorithm 6 ECP Partial Membership Witness Computation

```
    **Input**: Secret share $\langle\alpha\rangle_i$, accumulator $\Lambda_{\mathcal{X}}$, revoked certificate's serial number $x$.
    **Output**: Partial membership witness $\langle w_x\rangle_i$.
1: Compute share of the mulitplicative inverse $(x+\alpha)^{-1}$ as $\langle s\rangle_i \leftarrow \mathsf{InvSharesElement}(\langle\alpha\rangle_i, x)$
2: Compute polynomial $g(x) \leftarrow \mathsf{LagrangeInterpolation}(i, \langle s\rangle_i)$
3: Evaluate the Lagrange polynomial at 0 as $r \leftarrow g(0)$
4: Compute share of the membership witness $\langle w_x\rangle_i \leftarrow \Lambda_{\mathcal{X}}^r$
5: return $\langle w_x\rangle_i$
```

Algorithm 7 ECP Partial Non-membership Witness Computation

```
    **Input**: Secret share $\langle\alpha\rangle_i$, accumulator $\Lambda_{\mathcal{X}}$, non-revoked certificate's serial number $y$, public key $pk_\Lambda$, set of accumulated revoked serial numbers $\mathcal{X}$.
    **Output**: Partial non-membership witness $(\langle w_y\rangle_i, u_y)$.
1: parse the public parameters $pk_\Lambda$ as $(BG, h) = ((p, G_1, G_2, G_T, e, g_1, g_2), h)$
2: Compute share of the mulitplicative inverse of $(y+\alpha)^{-1}$ as $\langle s\rangle_i \leftarrow \mathsf{InvSharesElement}(\langle\alpha\rangle_i, y)$
3: Compute $u_y \leftarrow -\prod_{x\in\mathcal{X}}(x-y)$
4: Compute $wit_y \leftarrow g_1^{u_y} \cdot \Lambda_{\mathcal{X}}$
5: Compute polynomial $g(x) \leftarrow \mathsf{LagrangeInterpolation}(i, \langle s\rangle_i)$
6: Evaluate the Lagrange polynomial $g(x)$ at 0 as $r \leftarrow g(0)$
7: Compute $\langle w_y\rangle_i \leftarrow wit_y^r$
8: return $(\langle w_y\rangle_i, u_y)$
```

4.4.2. Partial Non-membership Witness Computation. Like partial membership witness generation, each ECP_i computes partial non-membership witnesses using their secret share $\langle\alpha\rangle_i$ without requiring knowledge of the secret key α. In Algorithm 7, each ECP computes a partial non-membership witness which constitutes two parts. The first part $\langle w_y\rangle_i$ is computed using the quite similar approach like partial membership witness generation except with depenedency on u_y. The second part u_y is a public value and doesn't depend on the secret share $\langle\alpha\rangle_i$ of the ECP_i.

4.4.3. Collaborative Witness Construction.

ECPs combine their partial witnesses to construct the full witness for each certificate using simple multiplication. Algorithm 8 outlines how ECPs reconstruct the full witness.

Algorithm 8 ECP Collaborative Witness Construction

```
    **Input**: Partial membership witnesses $\{\langle w_x\rangle_i\}$ or non-membership witnesses $\{\langle w_y\rangle_i, u_y\}$ from at least $t$ ECPs.
    **Output**: Full membership witness $w_x$ or non-membership witness $(w_y, u_y)$.
1: Randomly pick at least $t$ number of shares of membership or non-membership witnesses.
2: if $x \in \mathcal{X}$ then
3:     $w_x \leftarrow \prod_{i=1}^{t}\langle w_x\rangle_i$
4: else
5:     $w_y \leftarrow \prod_{i=1}^{t}\langle w_y\rangle_i$
6: Store $w_x$ or $(w_y, u_y)$ in cache for efficient retrieval.
```

Algorithm 9 Client Non-Membership Witness Verification

```
    Input: Public parameters $pk_\Lambda$, accumulator $\Lambda_{\mathcal{X}}$, non-revoked certificate's serial number $y$, non-membership witness $(w_y, u_y)$.
    Output: Verification result (Valid or Invalid).
1: parse the public parameters $pk_\Lambda$ as $(BG, h) = ((p, G_1, G_2, G_T, e, g_1, g_2), h)$
2: if $e(\Lambda_{\mathcal{X}} \cdot g_1^{u_y}, g_2) = e(w_y, g_2^y \cdot h)$ then
3:     Return Valid
4: else
5:     Return Invalid
```

Algorithm 10 Client Membership Witness Verification

```
    Input: Public parameters $pk_\Lambda$, accumulator $\Lambda_{\mathcal{X}}$, revoked certificate's serial number $x$, membership witness $w_x$.
    Output: Verification result (Valid or Invalid).
1: parse the public parameters $pk_\Lambda$ as $(BG, h) = ((p, G_1, G_2, G_T, e, g_1, g_2), h)$
2: if $e(\Lambda_{\mathcal{X}}, g_2) = e(w_x, g_2^x \cdot h)$ then
3:     Return Valid
4: else
5:     Return Invalid
```

This distributed approach to witness generation is more scalable and secure compared to traditional methods that rely on a single authority. No single ECP can infer anything about the shares of the witness of the other ECPs from the partial witnesses due to the discrete logarithm problem. [6]

4.4.4. Witness Distribution. Upon receiving a request from a client, an ECP retrieves the full witness from cache and sends it to the client rapidly, leveraging their CDN infrastructure. By precomputing and caching witnesses, ECPs can respond to client requests with minimal latency. This is particularly effective because ECPs, functioning as CDNs, are geographically distributed and close to clients, ensuring fast and efficient delivery of witnesses.

4.5. Client Operations

Clients verify the revocation status of certificates during TLS handshakes by obtaining and validating witnesses.

4.5.1. Witness Request and Retrieval. Clients send requests for witnesses to ECPs without needing to be aware of the underlying collaboration or threshold parameters.

The client simplifies its interaction by requesting the witness from a single ECP. The complexity of witness generation and ECP collaboration is abstracted away, enhancing usability and efficiency on the client side.

4.5.2. Witness Verification. The client verifies the validity of the received witness using public parameters and the bilinear pairing.

Algorithm 10 and 9 presents the verification process; the client checks the equality of two pairing computations using the public parameters and the received witness. If the equation holds, it confirms that the witness for membership (i.e., revoked status) or non-membership (i.e., non-revoked status) corresponds to the certificate, and that both the accumulator value and the witness are correct.

This verification step allows *the client to independently validate the legitimacy of the witness, even though it was received from a third party (ECP/CDN)*. If the validation fails, it indicates that either the accumulator value or the witness is incorrect or has been tampered with. This capability enhances the *auditability* of the system, as clients do not need to fully trust the third party serving the witness; they can verify its correctness themselves.

5. AccuRevoke: Implementation and Evaluation

We implement AccuRevoke in C++, utilizing several external libraries to handle cryptographic operations, network communication, and performance optimizations; the core implementation of AccuRevoke consists of approximately 3,000 lines of C++ code. For modular arithmetic and cryptographic computations, we used (1) Shoup's NTL library v11.5.1 [34] for our cryptographic accumulator and multiparty computation operations, (2) Pairing-Based Cryptography (PBC) library [36] to support elliptic curve cryptography and pairing-based operations, and (3) the ZeroMQ library [2] for socket programming to facilitate communication between the CA, client and ECPs.

In the following experiments, we evaluate the performance of AccuRevoke from multiple perspectives, examining the role of each entity. Our testing environment consists of an Intel Xeon Platinum 8360Y CPU with 48 cores.

5.1. First-party: Accumulator Management

We first evaluate the time required for accumulator construction and updates, as well as the scalability of these operations with respect to the number of elements and the utilization of parallel processing.

5.1.1. Accumulator Implementation. For AccuRevoke, we choose parameters based on the pairing-friendly Barreto-Naehrig (BN) curve which has an embdedding degree 12. This high embdedding degree allows us to reach a security level of 256 bits in extended field F_q^{12}; this choice results in an accumulator size of 21 bytes and witness sizes of 21 bytes (for membership proof) and 61 bytes (for non-membership proof). These sizes remain *constant* regardless of the number of accumulated elements, ensuring scalability as the number of elements increases.

We first measure the CPU time required to construct the accumulator as the number of revoked certificates increases, scaling up to 10 million elements. It is worth noting that Let's Encrypt, one of the largest CAs in terms of certificate issuance volume, reported approximately 800,000 revoked certificates [41]. We extend our evaluation up to 10 million elements to rigorously assess the performance of our system

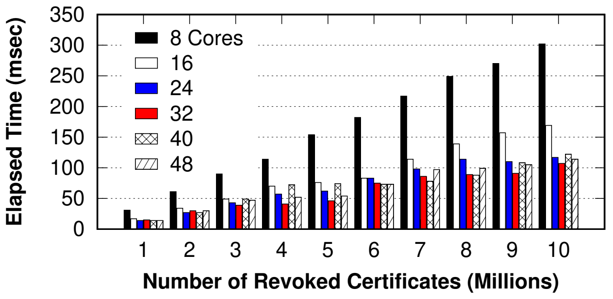


Figure 2. Accumulator generation time on CPU with multiple cores and different numbers of revoked certificates.

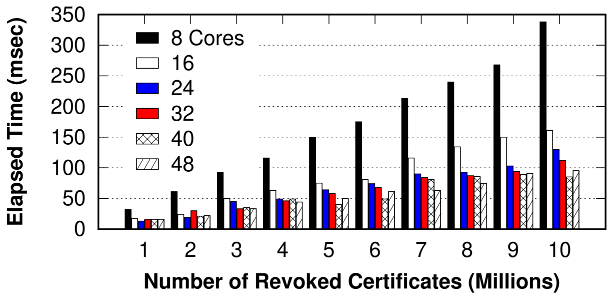


Figure 3. Non-membership witness generation time on CPU with multiple cores and different numbers of revoked certificates.

under larger workloads. We conduct tests across a range of processor configurations, varying from 8 to 48 cores.

As shown in Figure 2 we find that the CPU time for accumulator construction grows almost linearly with the number of elements. However, by leveraging multicores on the CPU, we achieve significant performance improvements; for example, constructing an accumulator with 10 million elements took less than 120 msec when utilizing 48 cores, underscoring the effectiveness of parallel processing in this context. Considering that the generation of the accumulator is a *one-time process*, we believe this initial construction time is acceptable for practical deployment.

5.1.2. Accumulator Updates. Now, we measure the time required to add or delete a single certificate from the accumulator; note that these operations occur only when a certificate is revoked or when a previously revoked certificate is reinstated. Since we manage only revoked certificates in the accumulator, the issuance of new certificates does not require updates to the accumulator.

The accumulator can theoretically hold an extremely large number of members (i.e., 205 duodecillions), which is nearly closer to the total number of serial numbers used in X.509 certificates (which are typically 160 bits, allowing for 2^{160} unique serials) [9]. Therefore, the CA may even choose *not* to remove expired revoked certificates from the accumulator to minimize operational overhead. However, there are cases where revoked certificates are determined to be valid again due to misconfiguration or error [32]. In such scenarios, it is necessary to delete members from the accumulator. Thus, we measure the performance overhead of both adding and deleting certificates in the accumulator.

From experiments, we find that the time to add or delete a single certificate remains constant regardless of the accumulator size; both addition and deletion operations take approximately *0.47 milliseconds* on average. This consistency is due to the design of AccuRevoke, where updates involve fixed-size exponentiation operations that are independent of the total number of accumulated elements as outlined in line 2 and 5 of Algorithm 5.

5.2. Third-party: Witness Generation and Reconstruction

We now evaluate the performance of witness generation for both membership (revoked certificates) and non-membership (non-revoked certificates) proofs. Note that this process is done by ECPs.

5.2.1. Witness for membership. When generating a single membership witness, we find that the CPU time remains constant at approximately 0.46 milliseconds using a single core with 1 million revoked certificates. Moreover, this time remains constant *regardless of the number of revoked certificates*. This is because generating a membership witness in our accumulator scheme involves computations that are independent of the size of the accumulated set (as noted in line $1-4$ in Algorithm 6); specifically, the process requires exponentiating the accumulator with the multiplicative inverse of an element related to the individual revoked certificate, rather than iterating over the entire set of revoked certificates.

This indicates that ECPs *do not* need to pre-generate membership witnesses for revoked certificates and can efficiently handle witness generation on the fly. The constant-time performance ensures scalability and responsiveness, making real-time witness generation practical even as the number of revoked certificates grows.

5.2.2. Witness for non-membership. In contrast, non-membership witness generation may face performance challenges because the accumulator must iterate over all its members to generate the proof, as outlined in line 3 of Algorithm 7.

Figure 3 presents the results of our experiments. We observe significantly slower performance compared to membership witness generation. For example, using 8 cores, the time required to generate a non-membership witness is approximately 70 times slower (0.46 vs. 31 msec) than generating a membership witness with an accumulator containing 1 million revoked certificates.

However, the performance improves as we utilize more CPU cores. By employing 48 cores, we find that the generation time decreases to 16 milliseconds, effectively making

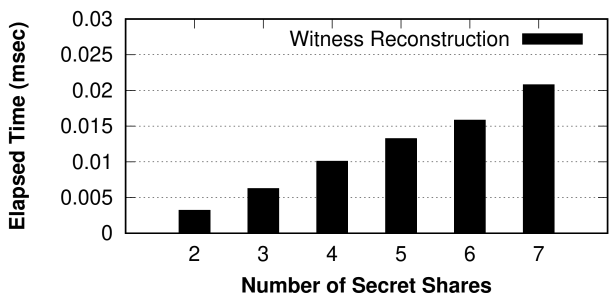


Figure 4. Witness reconstruction time with the different numbers of secret shares (t).

it over two times faster than with 8 cores; this improvement demonstrates that parallelism can effectively reduce the computation time for non-membership witness generation, especially as the number of revoked certificates increases.

Despite these improvements, the CPU time for non-membership witness generation is relatively higher than the typical time required to generate an OCSP response, which takes around 1 millisecond in our testbed. This disparity indicates that further optimization is necessary to make non-membership witness generation practical for real-world applications. We will discuss techniques to improve the generation time of non-membership proofs in the next section.

5.2.3. Witness reconstruction. After generating the partial witnesses from multiple ECPs, the ECPs initially contacted by the client is responsible for collecting these partial witnesses to construct the full witness; this operation must be efficient to avoid introducing performance overhead when returning the witness to the client.

We analyze the effect of varying the threshold parameter t in Shamir's Secret Sharing on the performance of witness reconstruction. Note that the full witness can be reconstructed by the client or by an ECP that aggregates the partial witnesses.

Figure 4 shows the witness reconstruction time as the threshold t increases. As outlined in Algorithm 8, the witness reconstruction depends only on the threshold t and is independent of the number of elements in the accumulator.

The results indicate a linear increase in reconstruction time with higher thresholds due to the additional computations required for mulitiplication of G_1's element; however, even with higher thresholds, the CPU reconstruction time remains within 0.02 milliseconds. Thus, we believe the overhead for reconstruction remains within acceptable limits for practical applications.

5.3. Client: Witness Validation

The client can validate the full witness using the accumulator fetched from the CA. As shown in Algorithm 10 and 9, the validation time is independent of the number of elements in the accumulator because the verification process involves two bilinear pairing operation. Specifically, the validation requires a constant number of exponentiation and pairing operations that do not depend on the size of the accumulated set or the number of ECPs.

Our measurements indicate that the validation time is approximately 24 milliseconds; this performance is acceptable for practical applications, as it introduces minimal overhead on the client side.

6. Overcoming the Non-membership Witness Generation Bottleneck

While AccuRevoke performs efficiently in accumulator management and membership witness generation, generating non-membership witnesses for revoked certificates can present a performance bottleneck when the number of revoked certificates surges due to a security incident that requires massive revocation, such as the Heartbleed vulnerability [24] or a CAA bug [14].

One possible solution is to use *double accumulators*, maintaining two accumulators: one for revoked certificates and another for non-revoked certificates. This allows ECPs to generate faster membership proofs for non-revoked certificates instead of slower non-membership proofs for revoked ones. However, this approach may introduce substantial overhead; the CA must frequently update the non-revoked certificates accumulator *whenever new certificates are issued*, disseminate updated accumulator values to clients and ECPs, and provide updated membership witnesses. This continuous updating is impractical and not scalable.

Alternatively, maintaining a single accumulator for revoked certificates and optimizing non-membership witness generation is more feasible. Our experiments (Figure 3) suggest a potential performance improvements in non-membership witness generation through increased parallelism using multicore processors.

6.1. GPU Acceleration

Modern edge computing providers, such as Cloudflare, now offer GPU acceleration services [15], which we can leverage to enhance the performance of our system.

6.1.1. Insights. In our implementation, AccuRevoke employs elliptic curve-based accumulators. The primary computational bottleneck in generating non-membership witnesses is computing the product $\prod_{x\in\mathcal{X}}(x-y)$, as specified in line 4 of Algorithm 7. This operation has a time complexity of $O(|\mathcal{X}|)$, where $\mathcal{X}$ represents the set of accumulated revoked certificates. To enhance performance, we leverage parallelism in calculating this product; each term in the product is independent, allowing distribution across multiple processing units.[2]

2. Notably, such parallelization is challenging with RSA accumulators for non-membership witness generation. RSA accumulators require computing Bézout coefficients using the Extended Euclidean Algorithm [29], an inherently sequential process that limits opportunities for parallel execution.

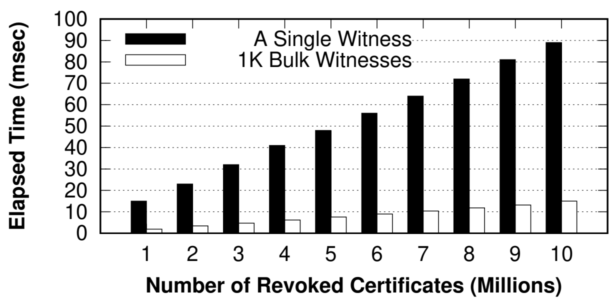


Figure 5. Non-Membership witness generation time (single vs. 1K Bulk) on GPUs with different numbers of revoked certificates.

We utilize the CUDA-based CGBN (C Arbitrary Precision Integer Library for GPUs) [35] from NVIDIA Labs to handle modular operations on large numbers, employing NVIDIA A100 GPUs for computation. Our implementation addresses the challenges associated with adapting cryptographic algorithms to GPU architectures, such as managing memory hierarchies and optimizing parallel workloads. The result is a performant system capable of handling large-scale cryptographic computations efficiently; we plan to open-source our implementation to support further research and development in this area. Our implementation consists of around 900 lines of C++ code.

6.1.2. Non-Membership Witness Generation on GPU.

Now we evaluate the performance of witness generation for non-membership on GPU. Figure 5 shows the average time required to generate a non-membership witness. Initially, the performance gains appear modest; for example, generating a non-membership proof with 1 million revoked certificates takes 15 milliseconds on the GPU, compared to 16 milliseconds using 48 CPU cores. This limited improvement is primarily due to the overhead of memory copying over the PCI Express (PCIe) bus to the GPU, which constitutes the majority of the processing time. Excluding the memory copy cost, the GPU computation achieves 8.5 milliseconds, highlighting the potential for significant speedup.

To mitigate the memory transfer overhead, we batch the processing of non-revoked certificates. By copying and processing batches of 1,000 certificates at once, we amortize the memory copy cost over multiple computations. This approach reduces the average time per non-membership proof to 1.8 milliseconds, achieving an 8.9-fold speedup compared to CPU-based processing. We believe that ECPs can effectively utilize GPUs to enhance performance in this manner.

7. Security Analysis

In this section, we discuss how AccuRevoke mitigates potential attacks across various scenarios.

Attack 1: Fooling Clients into Accepting Revoked Certificates. An attacker may attempt to trick clients into accepting a revoked certificate as valid by providing a fraudulent witness. In AccuRevoke, generating a valid non-membership witness (w_y, u_y) for a revoked certificate $x \in \mathcal{X}$ (i.e., x is a member of the accumulator) requires knowledge of the secret key α. Since α is securely shared among ECPs using Shamir's Secret Sharing with threshold t, and no single ECP possesses enough shares to reconstruct α, an attacker cannot generate a valid witness without compromising at least t ECPs.

Furthermore, the accumulator value $\Lambda_{\mathcal{X}}$ is signed by the CA and disseminated to clients and ECPs. Clients verify the authenticity of $\Lambda_{\mathcal{X}}$ using the CA's signature, ensuring that they use the correct accumulator. Any attempt to provide a tampered or fake accumulator would fail the signature verification, and subsequent witness verification would fail.

Attack 2: Generating Membership Witnesses for Non-Member Certificates. Conversely, an attacker may try to generate a membership witness for a certificate that was never issued or has not been revoked (i.e., $x \notin \mathcal{X}$). Without knowledge of α, or access to sufficient secret shares, it is computationally infeasible to generate a valid witness for such a certificate due to the hardness of the underlying cryptographic assumptions (e.g., the Strong Diffie-Hellman assumption). Even if the attacker has the oracle access to all the algorithms, it's infeasible for the attacker due to the collision freeness of the accumulator.

Attack 3: Compromised Edge Compute Providers. If one or more ECPs are compromised, the attacker gains access to their secret shares $\langle\alpha\rangle_i$. However, unless the attacker compromises at least t ECPs, they cannot reconstruct α or generate valid witnesses independently. Our use of threshold secret sharing ensures that the system remains secure as long as fewer than t ECPs are compromised.

In the event that compromised ECPs provide incorrect partial witnesses, clients will fail to reconstruct a valid full witness, and *the verification will fail*. Clients can request partial witnesses from multiple ECPs and use redundancy to ensure that they can obtain enough correct shares to reconstruct the witness.

The client-side verification process is inherently robust again collusion among t or more ECPs. Since verification depends on the trusted accumulator $\Lambda_{\mathcal{X}}$ from CA and the specific serial numbers of either revoked (x) or non-revoked (y) certificates, any atttempt to generate a fraudulent witness would result in a verification failure. This ensures that fabricated witnesses are easily detectable.

Attack 4: Denial of Service. An attacker may attempt to disrupt the availability of the system by preventing clients from obtaining witnesses or the accumulator. To mitigate this, AccuRevoke leverages the distributed nature of ECPs, which are geographically dispersed and can serve clients from multiple locations. Clients can request witnesses from multiple ECPs, increasing the likelihood of obtaining the necessary information even in the presence of network disruptions or targeted attacks.

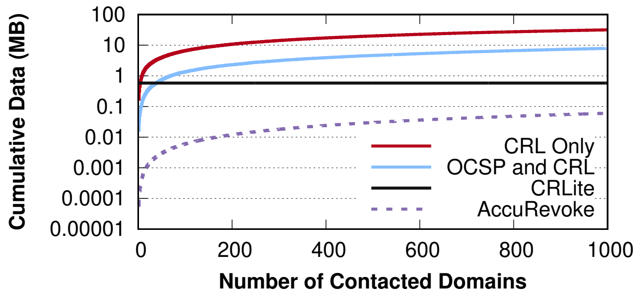


Figure 6. Cumulative data downloaded by clients using different revocation strategies.

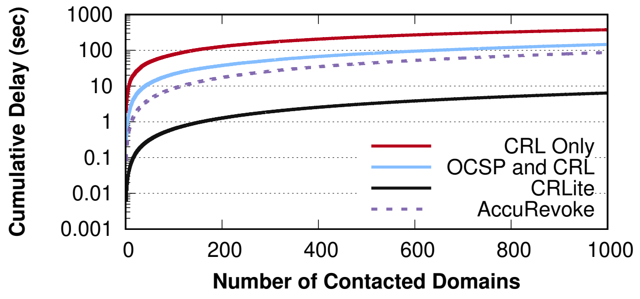


Figure 7. Cumulative delay experienced by clients using different revocation strategies.

8. Comparison with Other Approaches

Table 2 summarizes the comparison between AccuRevoke and three existing revocation dissemination schemes: CRLs, CRLite, and OCSP.

Bandwidth Consumption. Existing approaches to revocation incur significant data overheads. CRLs require downloading entire lists of revoked certificates—roughly 173 KB each [41]—and must be refreshed periodically. CRLite lowers this overhead by distributing compact filters, yet it still demands around 580 KB of daily delta updates [28]. OCSP responses are smaller (about 1.3 KB) [11], [28], but each certificate validation triggers a dedicated query. In contrast, AccuRevoke drastically reduces bandwidth by using *fixed-size* proofs (approximately 21 bytes for membership and 61 bytes for non-membership), independent of the total number of revoked certificates.

To evaluate real-world performance, we simulate a user visiting 1,000 HTTPS domains according to a Zipf distribution (exponent = 1.26) favoring popular sites following a similar setup in [28]. We ran 100 such simulations, each contacting around 650 unique domains on average, and computed the mean results. We compared four revocation strategies:

- *CRL Only*: The browser downloads entire CRLs from certificate authorities and caches them until expiration.
- *OCSP and CRL*: The browser uses OCSP by default, reverting to CRLs only if OCSP is unavailable.
- *CRLite*: The browser maintains a local "filter cascade" [28] with daily incremental updates, checking locally for revocations.
- *AccuRevoke*: The browser holds the latest accumulator value from the CA, and each domain supplies a short membership or non-membership proof.

Figure 6 shows the cumulative data downloaded as users visit up to 1,000 domains. Among these four strategies, *CRL Only* yields the highest bandwidth usage—reaching over 10 MB on average—due to frequent, large CRL downloads. *OCSP and CRL* uses smaller OCSP responses (1.3 KB) but still suffers from high overhead (a 10% fallback to full CRLs). Although *CRLite* avoids per-domain growth by leveraging a locally stored filter, AccuRevoke is ultimately *even more efficient*, keeping total downloads under 0.1 MB for 1,000 domains through its compact proofs.

Latency. Figure 7 highlights the cumulative delay in the same simulation. Following the modeling parameters of CRLite [28], we assume OCSP checks take 50–100 ms and CRLs 50–4,000 ms to download, reflecting real-world network variability. For CRLite, local lookups typically incur 10 ms if the certificate chain requires a filter-cascade check and 6 ms if it is already in the LRU cache [28].

In keeping with the bandwidth trends, *CRL Only* again demonstrates the largest latency—at times approaching 1,000s—due to the long-tail delays of downloading large CRLs. *OCSP and CRL* performs better overall, but fallback instances and the cumulative effect of many OCSP queries still add up. *CRLite* exhibits very low latency because checks are largely local once the filter is updated. Similarly, AccuRevoke requires a short proof validation per domain visit; although this introduces a small per-domain overhead, the proof size is minimal (tens of bytes), allowing AccuRevoke to surpass the latency performance of OCSP and CRL.

In practice, AccuRevoke offers CRLite-like data efficiency without forcing daily downloads of a large global filter. Unless a user visits tens of thousands of domains in a single day, AccuRevoke's smaller per-domain overhead remains highly attractive in typical browsing scenarios.

Privacy. CRLs and CRLite offer full privacy since clients check revocation status locally. OCSP poses privacy concerns because clients query the CA's OCSP responder, exposing their visited websites. Although CDNs may somewhat obscure client queries, they do not fully address privacy issues. AccuRevoke can enhance privacy by allowing clients to obtain proofs from ECPs and, due to the small proof sizes, potentially leverage alternative channels like DNS to mask client queries. However, leveraging such methods is beyond the scope of this paper.

Reliability. CRLs and CRLite function even if the origin server is unavailable, as clients can use cached revocation data. OCSP depends on the CA's origin server; if it is down and the CDN lacks fresh data, clients cannot obtain revocation status. AccuRevoke improves reliability by enabling

TABLE 2. Comparison of revocation dissemination schemes. We indicate privacy with a ▲ where schemes rely on OCSP stapling (rather than vanilla OCSP). In AccuRevoke, the CA itself cannot track queries, but the ECP may infer which certificate is being checked based on the request. CRLite's auditing can become cumbersome if clients (such as Mozilla) build their own filters without aggregating certificates from all CAs. Similarly, for OCSP, auditing may be complicated when a CDN uses a delegated certificate to sign responses.

Scheme	Revocations	Pull	Bandwidth	For 1K Websites		Privacy	Works with	Authenticity	Failure
	Covered	Model	Cost	Bytes Downloaded	Delay	Dead Origin	Auditable	Model	
CRL	All	✗	173 KB per CRL [41]	31.7 MB	378.7 sec	●	●	●	Hard-fail
CRLite	All	✗	580 KB per day [28]	0.58 MB	6.4 sec	●	●	●	Hard-fail
OCSP	All	●	1.3 KB per request [11], [28]	1.30 MB	74.8 sec	▲	✗	▲	Soft-fail
AccuRevoke	All	●	21 or 61 B per request	0.06 MB	86.9 sec	▲	●	●	Soft-fail

ECPs to generate proofs *independently* of the CA's origin server, ensuring clients can still perform revocation checks even when the origin is unreachable.

Failure Model. The failure model refers to the system's behavior when revocation information is unavailable. CRLs and CRLite use a *hard-fail* model: if the client cannot obtain revocation data, it treats the certificate as invalid, enhancing security but potentially affecting availability. OCSP employs a *soft-fail* model, where the client may proceed without revocation information, assuming the certificate is valid. While this improves availability, it may expose clients to risks if a revoked certificate is accepted. To address this issue, extensions like OCSP Stapling and Must-Staple have been introduced; however, the deployment rate of these extensions is low, mainly due to challenges in implementation and mismanagement by server operators [11]. AccuRevoke mitigates the risks associated with the soft-fail model by distributing revocation information across ECPs, reducing the likelihood of unavailability and enhancing the reliability of revocation checks.

Auditability. CRLs are auditable as clients can verify the authenticity of revocation lists from the CA. CRLite aims for auditability, but clients may not fully audit the filter generation process; in practice, the filter generation process is centralized, and clients may not have the means to fully audit it. Specifically, Mozilla collects all revocation information to generate the CRLite filters, and clients rely on these pre-generated filters without the ability to independently verify the completeness or correctness of the revocation data included. This reliance limits the clients' ability to audit the revocation information fully. OCSP's auditability is limited, especially when CDNs use delegated certificates [31]; clients cannot detect fraudulent responses if a CDN is compromised.

AccuRevoke maintains auditability by enabling clients to verify the authenticity of revocation proofs using the accumulator and public parameters, ensuring trust even when responses are served by third parties.

Deployability. AccuRevoke aligns with existing TLS/PKI deployment practices, where CAs rely on third-party CDNs (e.g., Akamai, Cloudflare) to distribute revocation data [5], [23], [33]. However, AccuRevoke refines ECPs' role: they cryptographically collaborate to generate proofs without holding full authority, reducing trust assumptions while retaining performance benefits. This creates compelling incentives for ECP adoption:

- Synergy with Existing Infrastructure. ECPs' globally distributed Points of Presence (PoPs) minimize latency for witness retrieval, aligning with their core CDN functionality. The threshold secret-sharing model (§3.1.1) allows ECPs to reuse existing edge compute resources (e.g., GPUs for parallelized operations in §6.1), avoiding costly infrastructure overhauls.
- Risk Mitigation. The threshold model (§4.4.3) ensures that no single ECP bears full responsibility for witness generation. This distributes liability and reduces the reputational risk of compromise, making participation safer for ECPs.
- Service Differentiation. By supporting privacy-preserving, auditable revocation checks with minimal bandwidth overhead, ECPs position themselves as leaders in secure edge computing. We believe this attracts security-conscious CAs and enterprises, differentiating them from competitors offering basic CDN services.

In practice, ECPs already partner with numerous CAs; AccuRevoke effectively formalizes these relationships, allowing them to provide authenticated revocation proofs without gaining access to a CA's private key. From a cost–benefit perspective, the additional computation and memory overhead is small relative to typical CDN workloads, while the potential benefits—service differentiation, new revenue channels, and enhanced trust—can be significant.

9. Conclusion

We introduced AccuRevoke, a novel scheme that enhances the certificate revocation process by ensuring the authenticity and integrity of revocation information, even when responses are generated by third parties without direct CA communication. Leveraging distributed cryptographic accumulators and threshold cryptography, AccuRevoke enables ECPs to securely provide authenticated revocation proofs.

AccuRevoke addresses key challenges in the revocation ecosystem with a scalable and efficient solution. It reduces proof sizes to 21 bytes for membership proofs and 61 bytes for non-membership proofs, significantly smaller than current OCSP implementations. Our scheme generates an accumulator containing 1M revoked certificates in 12 ms, updates it in 0.47 ms, and generates witnesses in 0.46 ms (membership) and 16 ms (non-membership). GPU acceleration further optimizes non-membership witness generation, reducing processing time to 8.5 ms.

Integrating AccuRevoke into existing infrastructure enhances the reliability, privacy, and efficiency of TLS revocation checking. Compact proof sizes and distributed computation reduce bandwidth consumption, mitigate single points of failure, and preserve client privacy by decoupling revocation checks from the CA.

Acknowledgments

We thank anonymous reviewers and shepherd for their helpful comments. This research was supported in part by NSF grants CNS-2247306, CNS-2350215, and the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) [RS-2023-00215700, Trustworthy Metaverse: blockchain-enabled convergence research].

References

[1] Beaver's Multiplication Trick. https://medium.com/partisia-blockchain/beavers-trick-e275e79839cc.

[2] ZeroMQ an open-source universal messaging library. https://zeromq.org.

[3] D. E. 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. RFC 6066, IETF, 2011.

[4] C. Arthur. DigiNotar SSL certificate hack amounts to cyberwar, says expert. The Guardian. http://www.theguardian.com/technology/2011/sep/05/diginotar-certificate-hack-cyberwar.

[5] J. Aas. OCSP systems at scale are complex. 2024. https://news.ycombinator.com/item?id=41047832.

[6] M. Alfred and M. Alfred. The Discrete Logarithm Problem. Elliptic Curve Public Key Cryptosystems, 1993.

[7] L. e. al. Certificate Transparency. RFC 6962 (Proposed Standard), IETF, 2013.

[8] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. AsiaCrypt, 2001.

[9] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, IETF, 2008. http://www.ietf.org/rfc/rfc5280.txt.

[10] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. CRYPTO, 2002.

[11] T. Chung, J. Lok, B. Chandrasekaran, D. Choffnes, D. Levin, B. Maggs, A. Mislove, J. Rula, N. Sullivan, and C. Wilson. Is the Web Ready for OCSP Must Staple? IMC, 2018.

[12] A. A. Chariton, E. Degkleri, P. Papadopoulos, P. Ilia, and E. P. Markatos. DCSP: Performant Certificate Revocation a DNS-based approach. EuroSec, 2016.

[13] CA/Revocation Checking in Firefox. https://wiki.mozilla.org/CA/Revocation_Checking_in_Firefox.

[14] CAA Rechecking Bug. https://community.letsencrypt.org/t/2020-02-29-caa-rechecking-bug/114591.

[15] Cloudflare Powers Hyper-Local AI Inference with NVIDIA Accelerated Computing. https://www.cloudflare.com/press-releases/2023/cloudflare-powers-hyper-local-ai-inference-with-nvidia.

[16] Cloudflare Workers. https://workers.cloudflare.com/.

[17] D. Dolev and A. C. Yao. On the Security of Public Key Protocols. IEEE Transactions on Information Theory, 29(2), 1983.

[18] D. Derler, C. Hanser, and D. Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. CT-RSA, Springer, 2015.

[19] O. Dubuisson. ASN.1 communication between heterogeneous systems. Morgan Kaufmann, 2001.

[20] Fastly Compute. https://www.fastly.com/products/edge-compute.

[21] S. Gibson and L. Laporte. Cascading Bloom Filters. 2024. https://www.grc.com/sn/sn-989.htm.

[22] L. Helminger, D. Kales, S. Ramacher, and R. Walch. Multi-party revocation in sovrin: Performance through distributed trust. CT-RSA, Springer, 2021.

[23] S. Helme. Let's Encrypt to end OCSP support in 2025. 2024. https://scotthelme.co.uk/lets-encrypt-to-end-ocsp-support-in-2025/.

[24] Heartbleed Bug. http://heartbleed.com.

[25] A. Josh, B. Richard, C. Benton, D. Zakir, E. Peter, F.-L. Alan, H. J. Alex, H.-A. Jacob, K. James, R. Eric, S. Seth, and W. Brad. Let's Encrypt: An Automated Certificate Authority to Encrypt the Entire Web. CCS, 2019.

[26] N. Korzhitskii and N. Carlsson. Revocation statuses on the Internet. PAM, 2021.

[27] B. Lynn. Type F Internals. https://crypto.stanford.edu/pbc/manual/ch08s08.html.

[28] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson. CRLite: a Scalable System for Pushing all TLS Revocations to Browsers. IEEE S&P, 2017.

[29] J. Li, N. Li, and R. Xue. Universal accumulators with efficient nonmembership proofs. ACNS, 2007.

[30] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, A. Schulman, and C. Wilson. An End-to-End Measurement of Certificate Revocation in the Web's PKI. IMC, 2015.

[31] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. IETF RFC 2560, IETF, 1999.

[32] Z. Ma, A. Faulkenberry, T. Papastergiou, Z. Durumeric, M. D. Bailey, A. D. Keromytis, F. Monrose, and M. Antonakakis. Stale TLS Certificates: Investigating Precarious Third-Party Access to Valid TLS Keys. IMC, 2021.

[33] Mozilla: CRLite: Speeding Up Secure Browsing. https://blog.mozilla.org/security/2020/01/21/crlite-part-3-speeding-up-secure-browsing/.

[34] NTL: A Library for doing Number Theory. https://libntl.org.

[35] NVIDIA CGBN Liabrary. https://github.com/NVlabs/CGBN/tree/master.

[36] The Pairing-Based Cryptography Library. https://crypto.stanford.edu/pbc/.

[37] A. Schulman, D. Levin, and N. Spring. RevCast: Fast, Private Certificate Revocation over FM Radio. CCS, 2014.

[38] A. Shamir. How to share a secret. CACM, 22(11), 1979.

[39] S. Santesson, M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Proposed Standard), IETF, 2013.

[40] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu. Edge Computing: Vision and Challenges. IEEE Internet of Things, 3(5), 2016.

[41] S. Trevor, D. Luke, and S. Kent. Let's revoke: Scalable global certificate revocation. NDSS, 2020.

[42] The Chromium Projects. https://www.chromium.org/Home/chromium-security/crlsets/.

[43] L. Zhang, D. Choffnes, T. Dumitraş, D. Levin, A. Mislove, A. Schulman, and C. Wilson. Analysis of SSL certificate reissues and revocations in the wake of Heartbleed. IMC, 2014.

Appendix A. Formal Definitions

In this section, we first present the security definitions for Bilinear Dynamic Universal accumulator based on $q-SDH$ and later we prove the security of $\mathsf{AccuRevoke}$.

Assumption 1 (q-Strong Diffie-Hellman). Let G_1 and G_2 be two cyclic groups of prime order p, respectively generated by g_1 and g_2. In the bilinear group pair (G_1, G_2), the $q-SDH$ problem is stated as follows:

Given as input a $(q+3)$-tuple of elements $(g_1, g_1^x, g_1^{x^2}, \ldots, g_1^{x^q}, g_2, g_2^x) \in G_1^{q+1} \times G_2$, output a pair $(c, g_1^{1/(x+c)}) \in \mathbb{Z}_p^* \times G_1$ for a freely chosen value $c \in \mathbb{Z}_p^* \setminus \{-x\}$.

For $q > 0$, we define the advantage of an adversary $\mathcal{A}$ as

$$\mathrm{Adv}_{q,A}^{q-SDH} := \Pr\left[A(g_1, (g_1^{x^i})_{i\in[q]}, g_2, g_2^x) = (c, g_1^{1/(x+c)})\right]$$

The $q-SDH$ assumption holds if $\mathrm{Adv}_{\mathrm{BG},A}^{q-SDH}$ is a negligible function for all PPT adversaries $\mathcal{A}$.

Definition 1 (Bilinear Dynamic Universal accumulator based on $q-SDH$). [18] A dynamic universal accumulator is a tuple of PPT algorithms (KeyGen, Eval, Add, Delete, MemWit, NonMemWit, VerMem, VerNonMem):

$\underline{(\alpha, pk_\Lambda) \leftarrow \mathsf{KeyGen}(1^\kappa, q)}$: Given a security parameter κ, the maximum number of elements that can be accumulated q, it computes $BG = (p, G_1, G_2, G_T, e, g_1, g_2)$ using q, κ and returns private key α and public parameters pk_Λ where $pk_\Lambda \leftarrow (BG, g_2^\alpha)$.

$\underline{\Lambda_\mathcal{X} \leftarrow \mathsf{Eval}(\alpha, pk_\Lambda, \mathcal{X})}$: Given the private key α, public parameters pk_Λ and a set $\mathcal{X}$ that has to be accumulated, it computes $\Lambda_\mathcal{X} \leftarrow g_1^{\prod_{x\in\mathcal{X}}(x+\alpha)}$ and returns the accumulator $\Lambda_\mathcal{X}$.

$\underline{(\Lambda_{\mathcal{X}'}, \mathcal{X}') \leftarrow \mathsf{Add}(\alpha, \Lambda_\mathcal{X}, x, \mathcal{X})}$: Given the private key α, an accumulator $\Lambda_\mathcal{X}$, an element $x \in \mathbb{Z}_p^*$ to be added, set of accumulated elements $\mathcal{X}$. If $x \in \mathcal{X}$, it return $\perp$ otherwise it computes $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^{(x+\alpha)}$ and returns the updated accumulator $\Lambda_{\mathcal{X}'}$ and updated set $\mathcal{X}' \leftarrow \mathcal{X} \cup x$.

$\underline{(\Lambda_{\mathcal{X}'}, \mathcal{X}') \leftarrow \mathsf{Delete}(\alpha, \Lambda_\mathcal{X}, x, \mathcal{X})}$: Given the private key α, an accumulator $\Lambda_\mathcal{X}$, an element $x \in \mathbb{Z}_p^*$ to be deleted, set of accumulated elements $\mathcal{X}$. If $x \notin \mathcal{X}$, it return $\perp$ otherwise it computes $\Lambda_{\mathcal{X}'} \leftarrow \Lambda_\mathcal{X}^{(x+\alpha)^{-1}}$ and returns the updated accumulator $\Lambda_{\mathcal{X}'}$ and updated set $\mathcal{X}' \leftarrow \mathcal{X} \setminus x$.

$\underline{w_x \leftarrow \mathsf{MemWit}(\alpha, \Lambda_\mathcal{X}, x)}$: Given the private key α, an accumulator value $\Lambda_\mathcal{X}$, an element x for which membership witness has to be generated where $x \in \mathcal{X}$, it computes $w_x \leftarrow \Lambda_\mathcal{X}^{(x+\alpha)^{-1}}$ and returns the membership witness w_x.

$\underline{(w_y, u_y) \leftarrow \mathsf{NonMemWit}(\alpha, pk_\Lambda, \Lambda_\mathcal{X}, \mathcal{X}, y)}$: Given the private key α, the public parameters pk_Λ, an accumulator value $\Lambda_\mathcal{X}$, the set of accumulated elements $\mathcal{X}$, and an element $y \notin \mathcal{X}$, it computes $u_y \leftarrow -\prod_{x\in\mathcal{X}}(x-y)$ and $w_y \leftarrow (\Lambda_\mathcal{X} \cdot g_1^{u_y})^{(y+\alpha)^{-1}}$ and returns the non-membership witness (w_y, u_y).

$\underline{\{0,1\} \leftarrow \mathsf{VerMem}(pk_\Lambda, \Lambda_\mathcal{X}, x, w_x)}$: Given the public parameters pk_Λ, the accumulator $\Lambda_\mathcal{X}$, the membership element x, the membership witness w_x, it returns 1 if $e(\Lambda_\mathcal{X}, g_2) = e(w_x, g_2^x \cdot g_2^\alpha)$, otherwise return 0.

$\underline{\{0,1\} \leftarrow \mathsf{VerNonMem}(pk_\Lambda, \Lambda_\mathcal{X}, y, w_y, u_y)}$: Given the public parameters pk_Λ, the accumulator $\Lambda_\mathcal{X}$, the non-membership element y, the non-membership witness (w_y, u_y), it returns 1 if $e(\Lambda_\mathcal{X} \cdot g_1^{u_y}, g_2) = e(w_y, g_2^y \cdot g_2^\alpha))$, otherwise return 0.

This bilinear dynamic universal accumulator is correct and sound under the q-strong Diffie-Hellman assumption defined as follows:

Definition 2 (Correctness). An accumulator is correct if there exists a negligible function $\epsilon(\cdot)$ in the security parameter κ such that the following holds:

$$\Pr\left[\begin{array}{l}(\alpha, pk_\Lambda) \leftarrow \mathsf{KeyGen}(1^\kappa, q),\\ w_x \leftarrow \mathsf{MemWit}(\alpha, \Lambda_\mathcal{X}, x):\\ \mathsf{VerMem}(pk_\Lambda, \Lambda_\mathcal{X}, x, w_x) = 1 \wedge x \in \mathcal{X}\end{array}\right] \geq 1 - \epsilon(\kappa),$$

and

$$\Pr\left[\begin{array}{l}(\alpha, pk_\Lambda) \leftarrow \mathsf{KeyGen}(1^\kappa, q),\\ (w_y, u_y) \leftarrow \mathsf{NonMemWit}(\alpha, pk_\Lambda, \Lambda_\mathcal{X}, y, \mathcal{X}):\\ \mathsf{VerNonMem}(pk_\Lambda, \Lambda_\mathcal{X}, y, w_y, u_y) = 1 \wedge y \notin \mathcal{X}\end{array}\right] \geq 1 - \epsilon(\kappa),$$

Correctness implies that any element in/not in the accumulator, a corresponding membership/non-membership witness must prove the membership/non-membership witness successfully.

Definition 3 (Collision Freeness). Let $\mathcal{O} = \{\mathsf{Eval}(\alpha, pk_\Lambda, \cdot), \mathsf{MemWit}(\alpha, \cdot, \cdot), \mathsf{NonMemWit}(\alpha, pk_\Lambda, \cdot, \cdot), \mathsf{Add}(\alpha, \cdot, \cdot, \cdot)$ and $\mathsf{Delete}(\alpha, \cdot, \cdot, \cdot)\}$ be the oracles such that α, pk_Λ are hard-coded and adversary is allowed to choose other parameters for the oracles (denoted as "$\cdot$"). Let $\Lambda^\star \leftarrow \mathsf{Eval}(\alpha, pk_\Lambda, \mathcal{X}^\star)$, be the value of the accumulator from the aforementioned oracle, where $\mathcal{X}^\star$ is a set of elements to be accumulated chosen by the adversary. A cryptographic accumulator is collision-free, if for all PPT adversaries $\mathcal{A}$ that have access to $\mathcal{O}$, there exists a negligible function $\epsilon(\cdot)$ such that:

$$\Pr\left[\begin{array}{l}(\alpha, pk_\Lambda) \leftarrow \mathsf{KeyGen}(1^\kappa, q),\\ (w_x, x^\star, \mathcal{X}^\star) \leftarrow \mathcal{A}^\mathcal{O}(pk_\Lambda):\\ \mathsf{VerMem}(pk_\Lambda, \Lambda^\star, x^\star, w_x) = 1 \wedge x^\star \notin \mathcal{X}^\star\end{array}\right] \leq \epsilon(\kappa),$$

and

$$\Pr\left[\begin{array}{l}(\alpha, pk_\Lambda) \leftarrow \mathsf{KeyGen}(1^\kappa, q),\\ (w_y, u_y, y^\star, \mathcal{X}^\star) \leftarrow \mathcal{A}^\mathcal{O}(pk_\Lambda):\\ \mathsf{VerNonMem}(pk_\Lambda, \Lambda^\star, y^\star, w_y, u_y) = 1 \wedge y^\star \in \mathcal{X}^\star\end{array}\right] \leq \epsilon(\kappa),$$

Soundness (collision resistance) prevents a dishonest party from generating a non-membership witness for a member or membership witness for a non-member.

Security Model. We define the security of $\mathsf{AccuRevoke}$ using Ideal/Real paradigm, such that the adversary $\mathcal{A}$ should not be able to make the witness verification unsuccessful (correctness) when the witnesses created by the semi-honest ECPs correctly in an honest majority settings. Also, an adversary $\mathcal{A}$ statically corrupting ECPs can not create a membership witness for a non-member and vice-versa (collision freeness). Let $\mathcal{F}$ be an ideal functionality that answers all the queries honestly. Let $\mathcal{S}$ be an ideal simulator that emulates the view of the real-world adversary. Let $\mathcal{E}$ be the environment that provides inputs for all entities and receives the corresponding outputs. Also, $\mathcal{E}$ can get any adversarial views at any time. We define our Ideal world as below:

Ideal:

1) **Gen:** On receiving input $(1^\kappa, q)$, it runs $(\alpha, pk_\Lambda) \leftarrow \mathsf{KeyGen}(1^\kappa, q)$ and sends (α, pk_Λ) to the CA. $\mathcal{F}$ notifies $\mathcal{S}$ about the event and $\mathcal{F}$ sends pk_Λ to $\mathcal{S}$.
2) **Eval:** On receiving input $(\alpha, pk_\Lambda, \mathcal{X})$ and it evaluates $\Lambda_\mathcal{X} \leftarrow \mathsf{Eval}(\alpha, pk_\Lambda, \mathcal{X})$ and sends $\Lambda_\mathcal{X}$ to the CA. $\mathcal{F}$ notifies $\mathcal{S}$ about the event and $\mathcal{F}$ sends $\Lambda_\mathcal{X}$ to $\mathcal{S}$.
3) **Add:** On receiving input $(\alpha, \Lambda_\mathcal{X}, x, \mathcal{X})$ and it updates $(\Lambda_{\mathcal{X}'}, \mathcal{X}') \leftarrow \mathsf{Add}(\alpha, \Lambda_\mathcal{X}, x, \mathcal{X})$ and sends $(\Lambda_{\mathcal{X}'}, \mathcal{X}')$ to the CA. $\mathcal{F}$ notifies $\mathcal{S}$ about the event and $\mathcal{F}$ sends $\Lambda_{\mathcal{X}'}$ to $\mathcal{S}$.
4) **Delete:** On receiving input $(\alpha, \Lambda_\mathcal{X}, x, \mathcal{X})$ and it updates $(\Lambda_{\mathcal{X}'}, \mathcal{X}') \leftarrow \mathsf{Delete}(\alpha, \Lambda_\mathcal{X}, x, \mathcal{X})$ and sends $\Lambda_{\mathcal{X}'}$ to the CA. $\mathcal{F}$ notifies $\mathcal{S}$ about the event and $\mathcal{F}$ sends $\Lambda_{\mathcal{X}'}$ to $\mathcal{S}$.
5) **MemWit:** On receiving x, $\mathcal{F}$ runs $w_x \leftarrow \mathsf{MemWit}(\alpha, \Lambda_\mathcal{X}, x)$. $\mathcal{F}$ notifies $\mathcal{S}$ about $\Lambda_\mathcal{X}$, $\mathcal{X}$, w_x and x. If $\mathcal{S}$ can simulate the view of adversary, it sends ok to $\mathcal{F}$ otherwise $\perp$. $\mathcal{F}$ sends w_x as output to the client if it receives ok from $\mathcal{S}$.
6) **NonMemWit:** On receiving y, $\mathcal{F}$ runs $(w_y, u_y) \leftarrow \mathsf{NonMemWit}(\alpha, \Lambda_\mathcal{X}, x)$. $\mathcal{F}$ notifies $\mathcal{S}$ about $\Lambda_\mathcal{X}$, (w_y, u_y), $\mathcal{X}$ and x. If $\mathcal{S}$ can simulate the view of adversary, it sends ok to $\mathcal{F}$ otherwise $\perp$. $\mathcal{F}$ sends (w_y, u_y) as output to the client if it receives ok from $\mathcal{S}$.

Let $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ denote the random variable describing the output of environment $\mathcal{E}$ when interacting with adversary $\mathcal{A}$ and parties running protocol $\mathsf{AccuRevoke}$-Π.

Let $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ denote the random variable describing the output of environment $\mathcal{E}$ when interacting with simulator $\mathcal{S}$ and ideal functionality $\mathcal{F}$.

Definition 3 (AccuRevoke Security): We say that the protocol Π securely UC-emulates the ideal functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{E}$, the distributions of $EXEC_{\Pi,\mathcal{A},\mathcal{E}}$ and $EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}$ are indistinguishable. That is, on any input, the probability that $\mathcal{E}$ outputs 1 after interacting with $\mathcal{A}$ and parties running Π differs by at most a negligible amount from the probability that $\mathcal{E}$ outputs 1 after interacting with $\mathcal{S}$ and $\mathcal{F}$, formally:

$$|\Pr[EXEC_{\Pi,\mathcal{A},\mathcal{E}}] - \Pr[EXEC_{\mathcal{F},\mathcal{S},\mathcal{E}}]| \leq \epsilon(\kappa)$$

Appendix B. Security Proof

Theorem 1 (**AccuRevoke security**). *AccuRevoke is UC-secure by Definition 3.*

Proof. We construct a simulator $\mathcal{S}$ such that no PPT environment $\mathcal{E}$ can distinguish between its view in the Ideal and Real. $\mathcal{E}$ can statically corrupt $t-1$ ECPs to view the execution transcript. On receiving the notification from $\mathcal{F}$, the simulator $\mathcal{S}$ functions as follows:

Gen:

1) $\mathcal{S}$ generates random $t-1$ elements as $(\alpha_1, \dots, \alpha_{t-1}) \xleftarrow{\$} \mathbb{F}_p$
2) $\mathcal{S}$ outputs $(\alpha_1, \dots, \alpha_{t-1})$

Eval, Add and Delete:

1) As these operations are performed by the trusted CA, the simulator does not need to simulate the adversary view for these commands.

MemWit:

1) $\mathcal{S}$ generates $t-1$ random inputs $(\alpha_1, \dots, \alpha_{t-1}) \xleftarrow{\$} \mathbb{F}_p$ to execute the $\mathsf{InvSharesElement}$ protocol on each input, obtaining corresponding outputs $(s_1, \dots, s_{t-1})$.
2) $\mathcal{S}$ computes lagrange polynomial $g(x)_i = s_i \cdot \prod_{\substack{1\leq j\leq t-1 \\ j\neq i}} \frac{x-x_j}{x_i-x_j}$ for each of the generated outputs from step 1.
3) $\mathcal{S}$ computes $g(0)_i$ for each of the polynomials from step 2.
4) $\mathcal{S}$ outputs $(g_1^{g(0)_1}, g_1^{g(0)_2}, \dots, g_1^{g(0)_{t-1}})$

NonMemwit:

1) $\mathcal{S}$ generates $t-1$ random inputs $(\alpha_1, \dots, \alpha_{t-1}) \xleftarrow{\$} \mathbb{F}_p$ to execute the $\mathsf{InvSharesElement}$ protocol on each input, obtaining corresponding outputs $(s_1, \dots, s_{t-1})$.
2) $\mathcal{S}$ computes Lagrange polynomial $g(x)_i = s_i \cdot \prod_{\substack{1\leq j\leq t-1 \\ j\neq i}} \frac{x-x_j}{x_i-x_j}$ for each of the generated outputs from step 1.
3) $\mathcal{S}$ computes $g(0)_i$ for each of the polynomials from step 2.
4) $\mathcal{S}$ outputs $(g_1^{g(0)_1}, g_1^{g(0)_2}, \dots, g_1^{g(0)_{t-1}})$

We now define a sequence of hybrid experiments. We show that the Real and Ideal are indistinguishable. All algorithms (KeyGen, Eval, Add, Delete, MemWit, NonMemWit, VerMem, VerNonMem) are non-interactive.

Hybrid 0. It is a real-world protocol $EXEC_{\Pi,\mathcal{S},\mathcal{E}}$ between an adversary $\mathcal{A}$ and environment $\mathcal{E}$ presented in Algorithms 3- 8. Without loss of generality, we assume $ECP_1, \dots, ECP_{t-1}$ are corrupted.

Hybrid 1. This game is the same as Hybrid 0 except that we make the following changes. First, we introduce the ideal functionality $\mathcal{F}$ to answer the $\mathcal{E}$'s request honestly. Second, in the Gen algorithm, the CA computes l random values, instead of using Shamir secret sharing. Third, in step 1 of MemWit or $\mathsf{NonMemWit}$ the ECP executes $\mathsf{InvSharesElement}$ protocol using dummy inputs rather than the one provided by the environment.

We claim that Hybrid 0 and Hybrid 1 are indistinguishable. Specifically, due to the security of Shamir secret sharing, shares generated in Gen algorithm in Hybrid 0 are indistinguishable from random elements generated in Hybrid 1. Second, due to the security of $\mathsf{InvSharesElement}$

protocol that makes use of Beaver's trick to perform Shamir share multiplication, the transcript being generated from the dummy input in Hybrid 1 is indistinguishable from the transcript generated from the actual input in Hybrid 0.

Note that Hybrid 1 is identical to the simulator $\mathcal{S}$ in Ideal, which shows that Ideal and Real are indistinguishable under the view of $\mathcal{E}$ and completes the proof. $\square$

Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

This paper designs a system that addresses many issues in certificate revocation. The presented system, called AccuRevoke, uses cryptographic accumulators and Shamir secret sharing to reduce the reliance on a single point of failure for providing revocation information to clients.

The system requires participation by three distinct entities: a CA, multiple edge compute providers (ECPs), and clients. It works by the CA generating an initial accumulator representing all the known revoked certificates. Any time a new certificate is revoked, the CA can update the accumulator. Whenever the accumulator is updated, the CA splits the accumulator into secret shares and distributes them among the ECPs. When a client wants to know if a certificate has been revoked, they request the witness of the accumulator from an ECP. The ECP then collects the locally generated shares of the witness from k-of-n other ECPs, recombines them and gets the actual witness that they return to the client.

C.2. Scientific Contributions

- Addresses a Long-Known Issue
- Provides a Valuable Step Forward in an Established Field

C.3. Reasons for Acceptance

1) This paper addresses the long-known issues with certificate revocation on the internet. They provide a scheme that significantly reduces bandwidth as compared to Certificate Revocation Lists (CRL) and reduces latency compared to the Online Certificate Status Protocol (OCSP).
2) This paper provides a valuable step forward in an established field. They build on prior work by Helminger et al. (2021) who designed the cryptographic mechanisms for revoking certificates in the general case and applied that scheme to the new domain of TLS certificate revocation on the internet—including some of the complexities of deployment and security concerns.

C.4. Noteworthy Concerns

A formal security treatment is provided in the appendix, however, as it is not in the main body and was added during shepherding, it has not been peer reviewed. As a result, the guarantees remain to be fully analysed.