Practical Cryptographic Forensic Tools for Lightweight Internet of Things and Cold Storage Systems

SAIF E. NOUMA, University of South Florida, USA
ATTILA A. YAVUZ, University of South Florida, USA

Internet of Things (IoT) and Storage-as-a-Service (STaaS) continuum permit cost-effective maintenance of security-sensitive information collected by IoT devices over cloud systems. It is necessary to guarantee the security of sensitive data in IoT-STaaS applications. Especially, log entries trace critical events in computer systems and play a vital role in the trustworthiness of IoT-STaaS. An ideal log protection tool must be scalable and lightweight for vast quantities of resource-limited IoT devices while permitting efficient and public verification at STaaS. However, the existing cryptographic logging schemes either incur significant computation/signature overhead to the logger or extreme storage and verification costs to the cloud. There is a critical need for a cryptographic forensic log tool that respects the efficiency requirements of the IoT-STaaS continuum.

In this paper, we created novel digital signatures for logs called *Optimal Signatures for secure Logging* (OSLO), which are the first (to the best of our knowledge) to offer both small-constant signature and public key sizes with near-optimal signing and batch verification via various granularities. We introduce new design features such as one-time randomness management, flexible tag aggregations along with various optimizations to attain these seemingly conflicting properties simultaneously. Our experiments show that OSLO offers 50× faster verification (for 2^{35} entries) than the most compact alternative with equal signature sizes, while also being several magnitudes of more compact than its most logger efficient counterparts. These properties make OSLO an ideal choice for the IoT-STaaS continuum, wherein lightweight logging and efficient batch verification of massive-size logs are vital for the IoT edge and cold storage servers, respectively.

Additional Key Words and Phrases: Authentication, secure logs, cold storage, digital signatures

ACM Reference Format:
Saif E. Nouma and Attila A. Yavuz. 2023. Practical Cryptographic Forensic Tools for Lightweight Internet of Things and Cold Storage Systems. In *International Conference on Internet-of-Things Design and Implementation (IoTDI '23), May 9–12, 2023, San Antonio, TX, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3576842.3582376

1 INTRODUCTION

System logs are vital tools for any security-critical applications [12]. They capture important events (e.g., user activity, errors, security breaches), making them an important target for attackers. Recent cyberattacks employ anti-forensics techniques to hide any evidence, namely by deleting or modifying log files. As such, administrators and/or verifiers cannot identify the source of errors during an incident investigation. Thereby, ensuring the trustworthiness of log files is a well-known topic for both authorities [1] and practitioners [20, 30].

[1] https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/

The emerging IoT harbors a sheer amount of IoT devices (e.g., sensors) that collect sensitive information (e.g., financial, health, personal) from the environment. These sensitive data and their metadata (i.e., log files) must be protected against such cyber attacks (e.g., impersonation, tampering) by ensuring their authentication, integrity, and confidentiality.

However, IoT devices are known to be resource-limited, rendering them more vulnerable to such cyber attacks. Indeed, IoT devices do not have the necessary storage capacity to keep locally the log files. Additionally, they are more vulnerable to (especially cyber-physical) attacks, and therefore there is a major risk of log tampering.

A common practice is to securely offload log streams to a cloud storage solution for future analytics and forensic investigation. Storage-as-a-Service (STaaS) [2] offers advanced data storage and infrastructure for end-users. However, it is highly expensive to retain append-only files (i.e., logs) on fast-access cloud servers which are usually dedicated to frequently accessed data. *Cold storage solution* [15] is a new type of data warehouse, designed to host large-scale archives. As such, Cold-STaaS becomes the best alternative to keep such rarely used yet valuable log files.

An ideal secure log authentication scheme for IoT-STaaS should offer (at minimum) the following properties:

• *Scalability, Public Verifiability, and Non-Repudiation*: (i) The cryptographic solution should be scalable to large IoT networks. (ii) It should allow any entity to verify the trustworthiness of information (e.g., meta-data, logs) by external parties. (iii) It should provide non-repudiation feature, which is essential for digital forensics and legal dispute resolution (e.g., financial, health). These features are usually offered by digital signatures [11, 24].

• *Logger Efficiency:* The cryptographic mechanisms must respect the limited resources (e.g., battery, memory, CPU) for low-end IoT devices (e.g., sensors), which are expected to operate for long durations without a replacement. (i) The authentication process should introduce a low computational overhead that translates into minimum energy consumption. (ii) The signatures should be compact to reduce the memory and transmission overhead. (iii) A small cryptographic code size is desirable to reduce the memory footprint.

• *Cloud Storage and Verification Efficiency*: Cold storage systems maintain sheer sizes of data (e.g., order of TBs). This requires an ability to compress cryptographic data, while periodic security controls necessitate fast batch verification.

• *Flexible Verification Granularity*: There is usually a performance and precision trade-off for secure log verification. For example, the authentication of the entire log stream with a single condensed tag offers minimal storage and fast verification time. However, having a single altered log entry renders the overall authentication invalid. Alternatively, signatures can be kept individually, per log entry, for the highest precision, but with high storage overhead. Hence, the cryptographic solution should permit for both logger and cold storage to adjust the storage granularity and verification precision depending on the application requirements [10, 19] .

It is a highly challenging task to devise a digital signature scheme that meets the stringent performance and security requirements of both IoT devices and STaaS simultaneously. The state-of-the-art techniques prioritize the needs of either the logger or verifier side while omitting performance and security features for the other side. In the following, we outline the research gap in the existing secure logging schemes by focusing on digital signatures.

1.1 Related Work and Research Gap

We first discuss the closest related works to our solutions with a focus on digital signature-based approaches. We then discuss other relevant and complementary works.

Related Work in our Scope: `OSLO` follows a prominent aggregate signature (AS)-based secure logging models (e.g., [10–12, 14, 18, 19, 24, 30]), where the logger compute an aggregate signature on its log entries so they can be attested

[2] https://www.intel.com/content/www/us/en/cloud-computing/storage-as-a-service.html

later. Digital signatures offer public verifiability and non-repudiation via Public Key Infrastructures (PKI). Therefore, they are suitable tools to provide scalable authentication for IoT and cold storage systems. Hereby, we outline the state-of-art signatures that are applicable in our context.

The standard digital signatures (e.g., RSA, Ed25519 [3]) involve expensive operations (e.g., modular exponentiation, Elliptic Curve (EC) scalar multiplication), which are costly for resource-limited IoTs. They do not offer aggregation property. Therefore, they introduce $O(T)$ signature overhead for T log entries putting a heavy storage burden on cold storage. Finally, the majority of them do not offer batch verification, which is important for fast authentication.

Aggregate Signatures (AS) [4, 29] can aggregate multiple distinct signatures into a single compact tag. Some aggregate signatures offer batch verification. Hence, they are instrumental tools for building cryptographic forensic schemes [12, 19, 20, 24, 30]. The Condensed-RSA (C-RSA) [29] and BLS [4] are two essential aggregate signatures but with a costly computation in both signing and verification. BLS requires highly expensive pairings and EC scalar multiplication with a heavy special hash function at the verifier and signer sides, respectively. C-RSA requires costly modular exponentiation with large key sizes. As shown in our experiments, they are highly costly for our envisioned IoT-STaaS applications.

Forward-secure and Aggregate Signatures (FAS) [14, 24, 30] offer breach-resiliency and signature aggregation. Despite their merits, FAS schemes introduce significant computational and storage overhead either at the signer and/or verifier sides. Some of these are signer-efficient signatures [30, 31], which makes them ideal choices for secure logging in resource-constrained IoT. However, this comes at the cost of a linear public key size. Our experiments proved that this introduces costly cloud storage overhead. Moreover, they cannot offer storage at different granularities due to fixed public key sizes. Hence, they are not suitable for emerging cold storage applications.

Recent AS schemes with extended properties for IoTs (e.g., [16, 25, 26, 32]) are either based on BLS [4] or Schnorr [6]. Hence, they inherit similar computational overhead (e.g., pairing, EC scalar mul) at the signer, which was demonstrated by our analysis that it might not be suitable for highly resource-limited devices. Additionally, we observe the absence of performance evaluations on low-end devices (e.g., 8-bit ATMega2560). In our comparisons, we focus on Ed25519 [3], SchnorrQ [6] and BLS [4] to represent the signer overhead schemes that rely on such cryptographic operations.

Other/Complementary Related Work: Our proposed scheme (`OSLO`) is a special class of aggregate digital signature, and therefore does not offer data confidentiality that can be achieved by: (i) data encryption at the logger [8], (ii) private-auditing at STaaS side, (iii) privacy-enhancing tools like searchable encryption [27].

There is a line of work focuses on Proof of Data Possession (PDP) [2] and Proof of Retrievability (PoR) [1] on the outsourced user data. Some works cope with privacy-preserving public auditing [28]. These works differ from our system model and primary performance objective. IoT devices do not compute a signature, but just transfer log files to STaaS, without initiating data authentication/integrity check. Rather, administrators (or STaaS) initiates usually an interactive integrity check protocol to audit the outsourced data, whereas AS-based schemes are generally non-interactive. PoR/PDP schemes offer fast audit time that is achieved by homomorphic linear authenticators (HLA) [28]. These enable an external entity to audit the data without having to retrieve the entire set. However, it comes at the cost of a very high computational overhead on IoT devices since the most deployed HLAs (i.e., BLS, RSA) suffer from expensive signing (see Table 1 and Fig. 7). In a different line, Li et al. in [17] proposed a public auditing protocol with data sampling for IoT networks.

Herein, our goal is to achieve optimal signing and small cryptographic payload for IoT devices, while offering compact storage and plausible verification efficiency at STaaS. By doing so, we permit low-end IoT to actively compute signatures, thereby ensuring public verifiability and non-repudiation. We note that `OSLO` can be transformed into a homomorphic

authenticable signature (via Map-to-Point operation akin to BLS [4]). However, this would result in a high computational signing, due again to reliance on BLS/RSA. Therefore, our scope and counterparts are AS-based secure logging schemes.

1.2 Our Contribution

In this work, we created a new series of secure logging schemes that we refer to as *Optimal Signatures for secure Logging* (OSLO). To the best of our knowledge, OSLO schemes are the first AS-based secure logging schemes that achieve small-constant tag and public key sizes with near-optimal signing and batch verification via various granularities. These features make them ideal for IoT-STaaS applications, wherein efficient signing and batch verification are critical for the resource-limited IoTs and cold storage servers, respectively.

Main Idea: Elliptic-Curve (EC)-based signatures usually offer the most compact tag sizes with a better signing efficiency compared to RSA-based [29] and pairing-based [4] alternatives. However, the most efficient EC-based signatures (e.g, Ed25519 [3], SchnorrQ [6]) still require at least one expensive operation (i.e., EC scalar multiplication) during signing. Many techniques attempted to address this bottleneck. A naive approach is to pre-compute private/public commitments during the key generation. This is at the cost of a linear storage overhead on the signer. As the number of log entries grows, such storage becomes infeasible on resource-constrained devices.

An alternative approach is to eliminate the public commitments (both computation and storage) from signature generation, by replacing them with one-time random seeds [21, 30]. Despite being highly signer efficient, these approaches require linear public key storage at the verifier, which incurs extreme overhead on Cold-STaaS (e.g., ≈ 3.3 TB for 2^{35} log entries). Overall, AS scheme is either efficient for the signer but with the expense of extreme storage cost and verification overhead, or expensive for the low-end device in terms of signing and storage overhead. In Section 1.1 and Section 6, we discuss AS-based signatures in terms of their conundrums.

In OSLO, we attempt to address these limitations by putting forward several new design approaches. (i) We introduce a new randomness management mechanism that achieves $O(\log_2 T)$ intermediate and $O(1)$ final one-time seed storage and computation. Our approach eliminates the linear server storage while preserving optimal and deterministic signing via a tree-based seed data structure that respects the post-signature disclosure requirement of EC-based signatures. (ii) Our schemes can aggregate additive and multiplicative homomorphic signature components separately with any desired granularity. This permits us to compress tags either at the IoT side per epoch, and/or compact them individually at the verifier. (iii) We propose two instantiations of OSLO: Signer-Optimal Coarse-grained OSLO (SOCOSLO) and FIne-grained Public-key OSLO (FIPOSLO). OSLO significantly outperforms their counterparts for the cold storage and verification time, with various granularities and high signer performance. In Table 1, we show a high-level comparison of OSLO with their counterparts (selection rationale to be discussed in Section 6) and outline their desirable properties below:

• *Compact Cold Cryptographic Storage and Fast Verification*: We compared our schemes with their alternatives for cryptographic storage and total verification times for 2^{35} entries (each is of size 32 bytes). OSLO achieves the fastest verification and compact storage among their counterparts. *(i)* They enable total storage of just 0.10 KB, which is *several magnitudes more compact* than alternative EC-based signatures (e.g., Ed25519, FI-BAF) with TBs of storage. *(ii)* SOCOSLO has 7× smaller signature than C-RSA and the same size as BLS, but with 9× and 50× faster verification, respectively. It is 24× faster than its most signer-efficient counterpart FI-BAF.

• *Flexible Verification Granularities and Architectures*: *(i)* In some IoT applications, IoT devices periodically stream their sensing reports to a verifier. In SOCOSLO, the logger signs each entry as collected and sequentially aggregates into a single "umbrella signature" to be uploaded to the verifier per epoch. SOCOSLO has a compact signature with the fastest

Table 1. Performance comparison of OSLO and its counterparts on embedded IoT and cold storage servers

Scheme	Logger (Signer) IoT Device: AtMega2560 (8-bit)			Cold Storage Server Commodity Hardware (Desktop)			Ver time (ms) (per epoch)	Dynamic Granularity	Granularity Level	Initial/Final Public Key
	Signing (in sec) (per item)	Cryptographic Payload (KB)	Priv Key Size (KB)	Cold Cryptographic Data: Entire Sig/PK Set (for 2^{35} entries)	Cold Cryptographic Data: One Sig (KB)	Ver Time (hours)				
Ed25519 [3]	1.45	16.38	0.03	2.20 TB	0.10	2243.12	56.78	×	Fine	$O(1)$ / $O(1)$
SchnorrQ [6]	0.27	16	0.03	2.20 TB	0.10	154.92	4.16	×	Fine	$O(1)$ / $O(1)$
FI-BAF [30]	0.01	0.05	0.10	3.30 TB	0.77	164.90	4.44	×	Coarse	$O(T)$ / $O(T)$
C-RSA [29]	83.26	0.25	0.51	0.77 KB	4.72	73.22	2.05	✓	Coarse/Fine	$O(1)$ / $O(1)$
BLS [4]	4.08	0.03	0.03	0.10 KB	0.10	432.55	15.15	✓	Coarse/Fine	$O(1)$ / $O(1)$
SOCOSLO	**0.01**	**0.05**	**0.06**	**0.10** KB	**0.11**	**8.33**	**0.17**	✓	**Coarse**	$O(T/L_1)$ / $O(1)$
FIPOSLO	**0.09**	**16**	**65.6**	**0.10** KB	**0.10**	**8.12**	**3.80**	✓	**Fine**	$O(1)$ / $O(1)$

The details of experiment settings, hardware/software configurations, and cryptographic parameters are given in Section 6. We chose our counterparts to cover the primary signature schemes, deployed for secure logging in the IoT domain. More details about our selection rationale can be found in Section 6.1. The total number of entries and the size of an epoch are $T = 2^{35}$ and $L_2 = 2^8$, respectively. At the cold storage server, the cryptographic storage (i.e., cold cryptographic data) is the total size of signatures and public keys needed to verify T entries. The verification time (in hours) is the total runtime of the batch verifying T items. At the logger (signer), the signature size is measured for an epoch. The signing time (in seconds) is given for a single entry. The verification time (in ms) is for all the collected items in a given epoch.

verification (89× than BLS) for an epoch level (e.g., $L_1 = 256$ items) of granularity (i.e., coarse-grained). However, it requires $O(L_1)$ initial public keys at the verifier but with $O(1)$ final public key at the cold storage. *(ii)* FIPOSLO keeps every signature separately to be authenticated and aggregated at the distiller. This enables the highest level of granularity (i.e., fine-grained) and $O(1)$ public key size. *(iii)* OSLO introduces a distillation process, in which the entries are verified and organized with a desired degree of granularities. The distillation can be done with an intermediate verifier (e.g., an edge cloud) or by the cold storage server itself.

• *Near-optimal Logging Efficiency*: OSLO schemes are highly signing efficient makes them ideal alternatives for logging in the resource-limited IoT devices. (i) SOCOSLO achieves a near-optimal signing by eliminating costly operations (e.g., EC multiplication). This makes it 27× and 40× faster than the most compact traditional and aggregate counterparts SchnorrQ and BLS, respectively. While as fast as FI-BAF, SOCOSLO is also many magnitudes more compact at the cold storage with 20× faster verification. (ii) FIPOSLO is the second-fastest alternative at the signer but with the finest granularity and $O(1)$ public-key storage advantage over SOCOSLO and FI-BAF. It has the largest private key to enable pre-computation, but this can be replaced with scalar multiplication for a compact private key.

• *Full-fledge Implementation*: We implemented OSLO schemes on a low-end IoT device and commodity hardware and compared their performance with that of their counterparts. Our experiments confirm that the asymptotic advantages of OSLO translate into practical performance. We open-source our implementation for public testing and adaptation purposes in the following online repository: https://github.com/SaifNOUMA/OSLO

2 PRELIMINARIES

Notation: || and $|x|$ denote concatenation and the bit length of variable x, respectively. $x \xleftarrow{\$} \mathcal{S}$ means variable x is randomly selected from the finite set $\mathcal{S}$ using a uniform distribution. $|\mathcal{S}|$ denotes the cardinality of set $\mathcal{S}$. $\{0,1\}^*$ denotes a set of binary strings of any finite length. $\{x_i\}_{i=1}^{n}$ denotes the set of items $(x_1, x_2, \ldots, x_n)$. $\log x$ denotes $\log_2 x$. $H_i : \{0,1\}^* \rightarrow \{0,1\}^\kappa, i \in \{0,1\}$ are distinct Full Domain Hash Functions [13], where κ is the security parameter. T denotes the maximum number of items to be signed in a given signature scheme. Our schemes operates over epochs, in which L_2 items are processed, with a total L_1 epochs available such that $T = L_1 \cdot L_2$. The variable $s_i^{j_0,j_1}$ denotes the aggregated (or derived) value of s for the iteration range $j_0 \leq j \leq j_1$ in the epoch i. $M_i^j \in \overrightarrow{M_i}$ means that M_i^j belongs to set of items $\overrightarrow{M_i}$. $\overrightarrow{M} = \{\overrightarrow{M_i}\}_{i \in \overrightarrow{i}}$ denotes a super vector where each $\overrightarrow{M_i}$ contains L_2 messages and $\overrightarrow{i}$ are epoch indices of $\overrightarrow{M}$.

Definition 2.1. An aggregate signature scheme ASGN consists of four algorithms (Kg, Agg, ASig, AVer) as follows:

- $(I, sk, PK) \leftarrow \texttt{ASGN.Kg}(1^\kappa, T)$: Given the security parameter κ and the maximum number of messages to be signed T, it returns a private/public key pair (sk, PK) with a public parameter I.
- $\sigma_{1,u} \leftarrow \texttt{ASGN.Agg}(\sigma_1, \ldots, \sigma_u)$: Given a set of signatures $\{\sigma_i\}_{i=1}^u$, it combines them and outputs an aggregate tag $\sigma_{1,u}$.
- $\sigma_i \leftarrow \texttt{ASGN.ASig}(sk, M_i)$: Given the secret key sk and a message M_i, it returns a signature σ_i as output.
- $b \leftarrow \texttt{ASGN.AVer}(PK, \{M_i\}_{i=1}^u, \sigma_{1,u})$: Given the public key PK, a set of messages $\{M_i\}_{i=1}^u$ and their corresponding aggregated signature $\sigma_{1,u}$, it outputs $b = 1$ if $\sigma_{1,u}$ is valid or $b = 0$ otherwise.

OSLO schemes rely on the intractability of *Discrete Logarithm Problem (DLP)* [13].

Definition 2.2. Let $\mathbb{G}$ be a cyclic group of order q, let α be a generator of $\mathbb{G}$, and let DLP attacker $\mathcal{A}$ be an algorithm that returns an integer in $\mathbb{Z}_q$. We consider the following experiment:

Experiment $Expt_{\mathbb{G},\alpha}^{DL}(\mathcal{A})$:

$b \xleftarrow{\$} \mathbb{Z}_q^*$, $B \leftarrow \alpha^b \mod q$, $b' \leftarrow \mathcal{A}(B)$,

If $\alpha^{b'} \mod p = B$ then return 1, else return 0

The DL-advantage of $\mathcal{A}$ in this experiment is defined as:

$$Adv_G^{DL}(\mathcal{A}) = Pr[Expt_{\mathbb{G},\alpha}^{DL}(\mathcal{A}) = 1]$$

The *DL* advantage of $(\mathbb{G}, \alpha)$ in this experiment is defined as follows:

$$Adv_G^{DL}(t) = \max_{\mathcal{A}}\{Adv_G^{DL}(\mathcal{A})\}, \text{ where the maximum is over all } \mathcal{A} \text{ having time complexity } t.$$

SOCOSLO uses Boyko-Peinado-Venkatesan (BPV) generator [5]. It reduces the computational cost of expensive operations (e.g., EC scalar mul.) via pre-computation technique. It consists of two algorithms described as follows:

1) $(\Gamma, v, k) \leftarrow \texttt{BPV.Offline}(1^\kappa, p, q, \alpha)$: It chooses BPV parameters (v, k) as the size of the pre-computed table and number of randomly selected elements, respectively. Then, it generates the pre-computed table $\Gamma = \{r_i, R_i\}_{i=1}^v$.
2) $(r, R) \leftarrow \texttt{BPV.Online}(\Gamma)$: It generates a random set $S \in \{1, \ldots, v\}$ of size $|S| = k$. Then, it computes a one-time commitment pair $(r \leftarrow \sum_{i \in S} r_i \mod q\,,\ R \leftarrow \prod_{i \in S} R_i \bmod p)$.

3 MODELS

System Model: Our system model follows a well-known AS-based secure logging models (e.g., [11, 12, 18, 19, 24, 30]), in which logger (i.e., IoT device) computes authentication tags on its log entries to be publicly verified later. Specifically, we consider an *IoT-Cloud continuum* wherein vast quantities of IoT devices generate log streams and report them to an (edge) cloud for analysis. As depicted in Fig. 1, our model consists of three main entities:

(i) Logger (Signer): represent the end-user IoT devices (e.g., medical sensors). They collect sensitive information (e.g., personal, health), and periodically upload them with their corresponding log entries to a nearby edge server (e.g., access point). They are expected to be resource-limited in terms of computation, storage, battery, and bandwidth.

(ii) Distiller: Any authorized entity can verify the log files and their digital signatures via corresponding public keys. For example, in a smart-building application, the IoT sensors can upload their periodic sensing reports to a nearby edge cloud. Before placing logs and signatures into the cold storage, we consider that the edge cloud performs a distillation process. That is, it maintains *Cold Cryptographic Data (CCD)* that harbors "valid" batches of log entries with their compressed (and adjustable) tags in various granularities. We assume that it keeps the "invalid" log entry-signature pairs individually. Remark that in the vast majority of real-life applications, the number of "invalid" (flagged) entries usually form only a negligible part of the entire log set. Hence, the "valid" entries dominate the storage of *CCD*. After the distillation, the edge cloud uploads *CCD* to the cold storage servers for long-term maintenance and check.

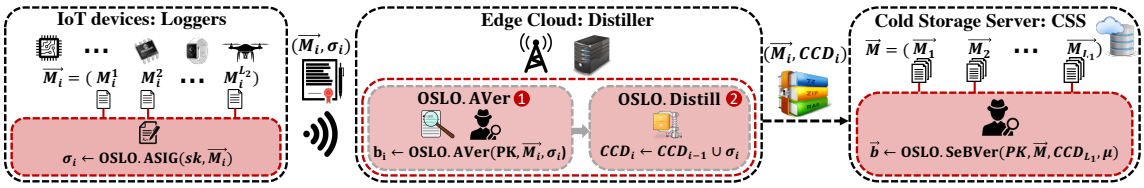


Fig. 1. A high-level illustration of OSLO system model and algorithms

(iii) Cold Storage Server (CSS) : It gives a STaaS service for our IoT-STaaS continuum. As discussed in Section 1, STaaS need regular audits to prove that their digital archives are trustworthy [7]. Hence, verifiers periodically check the authentication and integrity of logs maintained in CSS. For simplicity, verifiers are part of CSS.

Threat and Security Model: We follow the threat model of cryptographic audit log techniques originally introduced by Schneier et al. in [22] and then improved in various subsequent cryptographic works [10, 12, 18, 30]. In this model, the adversary is an active attacker that aims to forge and/or tamper audit logs to implicate other users. The state-of-the-art cryptographic secure logging schemes rely on digital signatures to thwart such attacks with public verifiability and non-repudiation. As stated in Section 1, we focus on signer-efficient (EC-based) aggregate signature-based approaches due to their compactness and fast batch verification properties.

We follow the *Aggregate Existential Unforgeability Under Chosen Message Attack* (A−EU−CMA) [4] security model that captures our threat model. A−EU−CMA considers the homomorphic properties of aggregate signatures and can offer desirable features such as log order preservation (if enforced) and truncation detection for signature batches. OSLO schemes are single-signer aggregate signatures, and therefore we do not consider inter-signer aggregations.

Definition 3.1. A−EU−CMA experiment for ASGN is as follows:

Experiment $Expt^{\text{A-EU-CMA}}_{\text{ASGN}}(\mathcal{A})$

$(I, sk, PK) \leftarrow$ ASGN.Kg$(1^\kappa, T)$,

$(M^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.),\, \text{ASGN.ASig}_{sk}(\overrightarrow{M})}(PK)$,

If ASGN.AVer$(PK, M^*, \sigma^*) = 1$ & $M^* \notin \{\overrightarrow{M_j}\}_{j=1}^{L_1}$, return 1 else 0.

The A−EU−CMA of $\mathcal{A}$ is defined as

$$Adv^{\text{A-EU-CMA}}_{\text{ASGN}}(\mathcal{A}) = Pr[Expt^{\text{A-EU-CMA}}_{\text{ASGN}}(\mathcal{A}) = 1].$$

The A−EU−CMA advantage of ASGN is defined as

$$Adv^{\text{A-EU-CMA}}_{\text{ASGN}}(t, T', T) = \max_{\mathcal{A}}\{Adv^{\text{A-EU-CMA}}_{\text{ASGN}}(\mathcal{A})\},$$

where the maximum is over $\mathcal{A}$ having time complexity t, with at most T' queries to $RO(.)$ and T queries to ASGN.ASig(.).

The oracles reflect how OSLO works as ASGN scheme. The signing oracle ASGN.ASig(.) returns an aggregate signature σ on a batch of messages $\overrightarrow{M} = (\overrightarrow{M_1}, \ldots, \overrightarrow{M_{L_1}})$ computed under sk. ASGN.Agg(.) aggregates the individual (or batch) signatures of these messages. ASGN.Agg(.) can be performed during the signing or before verification (e.g., in the distillation). It can aggregate additive or multiplicative components $\delta_i \in \sigma_i$. $RO(.)$ is a random oracle from which $\mathcal{A}$ can request the hash of any message of her choice up to T' messages. In our proofs (see Section 5), cryptographic hash functions are modeled as a random oracle [13] via $RO(.)$.

4 PROPOSED SCHEMES

Our goal is to create new cryptographic secure logging schemes that can meet the stringent requirements of low-end IoT devices with efficient signing and compact signatures while achieving fast verification and optimal storage in the cloud. We aim to achieve: *(i)* A near-optimal signer computational efficiency with no costly EC-scalar multiplication or modular exponentiation. *(ii)* Compact aggregate tag storage and transmission. *(iii)* $O(1)$ final cryptographic storage for the cold storage, which means $O(1)$ public key and signature size for the highest level of compression. *(iv)* Fast batch verification for a large number of messages. *(v)* Ability to aggregate tags in any desired granularity at the signer and/or verifier sides.

We observe that, among the existing aggregate signatures, EC-based signer-efficient variants (e.g., FI-BAF [30]) have the best potential for IoT, yet lack the necessary compactness and fast verification for cold storage applications (see Section 1 for a recap). They transform the Schnorr signature [23] into a one-time aggregate signature, in which the generation and storage of costly commitments ($R \leftarrow \alpha^r \bmod q, r \xleftarrow{\$} \mathbb{Z}_q^*$) are shifted to the key generation and verifier, respectively. In a nutshell, the signing process separates the message M from the commitment by replacing $H(M||R)$ with $H(M||x)$, where x is one-time randomness. "x" cannot be disclosed before signing and does not admit aggregation. Hence, it enforces $O(T)$ storage and expensive batch verification, which are extremely costly as shown in Section 6.

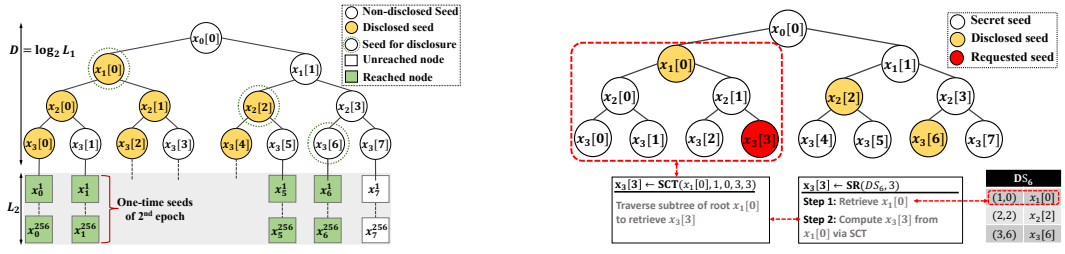
We developed several new techniques that address the *signer versus verifier bottleneck* conundrum. In Fig. 1, we outline our system model and OSLO's high-level functionalities. We first describe our data structures and new seed management strategy to cope with linear seed storage in Section 4.1. We then present our proposed schemes SOCOSLO and FIPOSLO that offer efficient signing, compact server storage, and batch verification with various granularity options.

4.1 OSLO Data Types and Seed Management

OSLO Data types: OSLO Tree-based structure (OSLOT) is a hash-based tree for seed storage and management, in which the leaves are one-time random seeds x, and the left and right children are computed via $H_{0,1}$, respectively. Let L_1 and $D = \log L_1$ be the maximum number of leaves and tree depth, respectively. OSLOT nodes $x_d[i]$ at depth $d, 0 \le d \le D$, for index $i, 0 \le i < 2^d$, are computed as:

$$x_d[i] = \begin{cases} H_0(x_{d-1}[\lfloor \frac{i}{2} \rfloor]), & \text{if } i \equiv 0 \mod 2 \\ H_1(x_{d-1}[\lfloor \frac{i}{2} \rfloor]), & \text{if } i \equiv 1 \mod 2 \end{cases}$$

Disclosed Seeds (DS) is a hash table structure. It maintains the disclosed nodes as values and their coordinates (i.e., depth d, index i) as keys. Formally, it is presented as follows: $DS : (d, i_d) \rightarrow x_d[i_d]$, where $0 \le d \le D$ and $0 \le i_d \le 2^d - 1$.



(a) Tree-based seed managment (b) Verifier's view

Fig. 2. Illustration of tree-based seed management functionalities

The Seed Management Functions (SMF) are formalized in Fig. 3: (i) *Seed Computation (SC)* takes the source node $x_{d_0}[i_0]$ and computes the requested child $x_d[i]$ by traversing OSLOT tree. (ii) *Seed Storage Optimizer* (SSO) discloses ancestor nodes progressively when the logger completes a given number of epochs. Given leaf index i and the OSLOT root $x_0[1]$, it outputs a compact DS_i. SSO seeks the seeds that share the same ancestor node, thereby ensuring (at most) $O(\log L_1)$ storage. (iii) Seed Retrieval (SR) returns the seed $x_D[i]$ if DS contains an ancestor for the leaf of index i.

An instance of OSLOT is provided in Fig. 2a, where $(L_1 = 2^3, L_2 = 2^8)$. It shows the OSLOT status after completing the 6^{th} epoch. The seeds, to be disclosed, are highlighted. They can be determined by running SSO algorithm. The SSO output is: $DS_6 \leftarrow \texttt{SSO}(x_0[1], 6)$ where $DS_6 = \{(1,0) : x_1[0];\ (2,2) : x_2[2];\ (3,6) : x_3[6]\}$.

The advantage of OSLOT seed management is apparent over the linear disclosure of one-time commitments in Schnorr-like schemes. It transforms $O(T)$ of both logger transmission and verifier storage into (at most) $O(\log L_1)$. Upon finishing all epochs, the logger discloses the OSLOT root $x_0[1]$, enabling $O(1)$ verifier storage.

$x_d[i] \leftarrow \texttt{SC}(x_{d_0}[i_0], d_0, i_0, d, i)$:

1: Set $x_p \leftarrow x_{d_0}[i_0]$ and $i \leftarrow i_d - (i_0 - 1) \cdot 2^{d-d_0}$
2: **for** $j = d - d_0 - 1, \ldots, 0$ **do**
3: $\quad x_p = \begin{cases} H_0(x_p), & \text{if } \lfloor i/2^j \rfloor \equiv 0 \mod 2 \\ H_1(x_p), & \text{if } \lfloor i/2^j \rfloor \equiv 1 \mod 2 \end{cases}$
4: **return** $(x_d[i] \leftarrow x_p)$

$DS_i \leftarrow \texttt{SSO}(x_0[0], i)$:

1: Let $\beta = (\beta_1, \ldots, \beta_D)$ be the binary representation of i
2: $J \leftarrow \{j, j \in \{0, \ldots, D\} \setminus \beta_j = 0\}$, $DS_i \leftarrow \{\}$ and counter $l \leftarrow 0$
3: **for** $j \in J$ **do**
4: $\quad d \leftarrow D - j$; $i_d \leftarrow \lfloor l/2^j \rfloor$
5: $\quad x_d[i_d] \leftarrow \texttt{SC}(x_0[0], 0, 0, d, i_d)$
6: $\quad$ Add $\{(d, i_d) : x_d[i_d]\}$ to DS_i and increment $l \leftarrow l + 2^j$
7: **return** DS_i

$x_D[i] \leftarrow \texttt{SR}(DS, i)$:

1: **if** $\exists (d, i_d) \in DS$, where $x_d[i_d]$ is an ancestor node of $x_D[i]$ **then**
2: $\quad x_D[i] \leftarrow \texttt{SC}(x_d[i_d], d, i_d, D, i)$
3: **else** $x_D[i] \leftarrow \perp$
4: **return** $x_D[i]$

Fig. 3. Seed Management Functions (SMF)

4.2 Signer-Optimal Coarse-grained OSLO (SOCOSLO)

SOCOSLO offers a near-optimal signing efficiency in terms of both computational and storage overhead. It offloads an aggregate tag upon signing an epoch of individual log entries. Unlike previous EC-based signature designs, SOCOSLO pre-stores a $O(L_1)$ sublinear number of public commitments (R) at the verifier side, and compact them after receiving the authenticated logs from IoT devices. In the following, we explain the formal description of SOCOSLO routines.

SOCOSLO Digital Signature Algorithms: We give the aggregate signature functions of SOCOSLO in Fig. 4a.

In SOCOSLO.Kg(.), for a given T, we first select the number of epochs and items to be signed in an epoch as L_1 and L_2, respectively (Step 1). We then generate the initial ephemeral randomness r_0 and the root of OSLOT tree $x_0[0]$ (Step 2). These values will be used to generate ephemeral public commitments (R) and one-time randomness (x) for a given epoch state $St : (i)$. We generate EC-based parameters (p, q, α) and private/public key pair (y, Y) (Step 3-4). SOCOSLO is coarse-grained, and therefore we combine the commitments for each epoch as in Step (5-7), which results in initial $O(L_1)$ and final $O(1)$ storage at the verifier via aggregation. The private/public key and parameters are as in Steps (8-9).

$(I, sk, PK) \leftarrow$ SOCOSLO.Kg$(1^\kappa, T)$:

1: Select integers (L_1, L_2) such that $L_1 \cdot L_2 = T$ and L_1 is a power of 2
2: $x_0[0] \xleftarrow{\$} \{0,1\}^\kappa$; $r_0 \xleftarrow{\$} \mathbb{Z}_q^*$; $D \leftarrow \log(L_1)$; $\rho \xleftarrow{\$} [0,1]$
3: Generate large primes q and $p > q$ such that $q|(p-1)$. Select a generator α of the subgroup G of order q in $\mathbb{Z}_q^*$.
4: $y \xleftarrow{\$} \mathbb{Z}_q^*$; $Y \leftarrow \alpha^y \bmod p$
5: **for** $i = 0, \ldots, L_1 - 1$ **do**
6: $\quad r_i^{1,L_2} \leftarrow \sum_{j=1}^{L_2} r_i^j \bmod q$, where $r_i^j \leftarrow H_0(r_0 \parallel i \parallel j)$
7: $\quad R_i^{1,L_2} \leftarrow \alpha^{r_i^{1,L_2}} \bmod p$
8: $sk \leftarrow (y, r_0, x_0[0])$; $PK \leftarrow (Y, \overrightarrow{R})$, where $\overrightarrow{R} \leftarrow \{R_i^{1,L_2}\}_{i=0}^{L_1-1}$
9: The system-wide param $I \leftarrow (p, q, \alpha, L_1, L_2, T, D, St : (i = 0))$
10: **return** (I, sk, PK)

$\delta_{1,u} \leftarrow$ SOCOSLO.Agg$(\{\delta_j \in \sigma_j\}_{j=1}^u)$:

1: **if** $\delta \in \mathbb{Z}_q^*$ **then** $\delta_{1,u} \leftarrow \sum_{j=1}^u \delta_j \bmod q$ **else** $\delta_{1,u} \leftarrow \prod_{j=1}^u \delta_j \bmod p$
2: **return** $\delta_{1,u}$

$\sigma_i^{1,L_2} \leftarrow$ SOCOSLO.ASig$(sk, \overrightarrow{M_i})$: require $i < L_1$ and $\overrightarrow{M_i} = \{M_i^j\}_{j=1}^{L_2}$

1: $x_D[i] \leftarrow$ SC$(x_0[0], 0, 0, D, i)$ and $s_i^{0,0} \leftarrow 0$
2: **for** $j = 1, \ldots, L_2$ **do**
3: $\quad r_i^j \leftarrow H_0(r_0 \parallel i \parallel j) \bmod q$
4: $\quad e_i^j \leftarrow H_0(M_i^j \parallel x_i^j) \bmod q$, where $x_i^j \leftarrow H_0(x_D[i] \parallel j)$
5: $\quad s_i^{1,j} \leftarrow$ SOCOSLO.Agg$(s_i^{1,j-1}, s_i^j)$, where $s_i^j \leftarrow r_i^j - e_i^j \cdot y \bmod q$
6: $\sigma_i^{1,L_2} \leftarrow \langle s_i^{1,L_2},$ $DS_i \leftarrow$ SSO$(x_0[0], i)\rangle$ and $St : (i \leftarrow i+1)$
7: **return** σ_i^{1,L_2}

$b \leftarrow$ SOCOSLO.AVer$(PK, \overrightarrow{M}, \sigma)$: Set of messages $\overrightarrow{M} = \{\overrightarrow{M_i}\}_{i \in \overrightarrow{i}}$ and require $|\overrightarrow{M_i}| \equiv 0 \bmod L_2, \forall i \in \overrightarrow{i}$

1: **if** $R \notin \sigma$ **then** $R \leftarrow$ SOCOSLO.Agg$(\{R_i^{1,L_2} \in PK\}_{i \in \overrightarrow{i}})$
2: $e \leftarrow 0$
3: **for** $i \in \overrightarrow{i}$ **do**
4: $\quad x_D[i] \leftarrow$ SR(DS, i)
5: $\quad$ **for** $j = 1, \ldots, L_2$ **do**
6: $\quad\quad x_i^j \leftarrow H_0(x_D[i] \parallel j)$
7: $\quad\quad e \leftarrow e + H_0(M_i^j \parallel x_i^j) \bmod q$
8: **if** $R = Y^e \cdot \alpha^s \bmod p$ **then return** $b = 1$ **else return** $b = 0$

(a) Digital signature algorithms

$CCD_i \leftarrow$ SOCOSLO.Distill$(PK, CCD_{i-1}, \overrightarrow{M_i}, \sigma_i^{1,L_2})$: Initialize $s_d = 0$ and $R_d = 1$

1: $b_i \leftarrow$ SOCOSLO.AVer$(PK, \overrightarrow{M_i}, \sigma_i^{1,L_2})$
2: **if** $b_i = 1$ **then**
3: $\quad s_d \leftarrow$ SOCOSLO.Agg(s_d, s_i^{1,L_2})
4: $\quad R_d \leftarrow$ SOCOSLO.Agg(R_d, R_i^{1,L_2})
5: $\quad$ **if** $i \mod \rho \cdot L_1 = 0$ **then**
6: $\quad\quad s_A \leftarrow$ SOCOSLO.Agg(s_A, s_d)
7: $\quad\quad R_A \leftarrow$ SOCOSLO.Agg(R_A, R_d)
8: $\quad\quad \sigma_A \leftarrow \langle s_A, R_A \rangle$
9: $\quad\quad \sigma_u = \langle s_d, R_d \rangle$
10: $\quad\quad$ Reset $(\sigma_d = 0,\ R_d = 1)$
11: $\quad\quad CCD_i^V \leftarrow \{\sigma_A\}$
12: $\quad\quad CCD_i^U \leftarrow CCD_i^U \cup \{\sigma_u, \lfloor \frac{i}{\rho \cdot L_1} \rfloor\}$
13: **else**
14: $\quad \sigma_i^{1,L_2} \leftarrow \langle s_i^{1,L_2}, R_i^{1,L_2} \rangle$
15: $\quad CCD_i^I \leftarrow CCD_{i-1}^I \cup \{\sigma_i^{1,L_2}, i\}$
16: Delete R_i^{1,L_2} from PK
17: $CCD_i \leftarrow (CCD_i^V, CCD_i^U, CCD_i^I, DS_i)$
18: **return** CCD_i

$\overrightarrow{b} \leftarrow$ SOCOSLO.SeBVer$(PK, \overrightarrow{M}, CCD, \mu)$: require $|\overrightarrow{M}| \equiv 0 \bmod L_2$

1: Attach DS to every signature σ to be verified
2: **switch** (μ)
3: $\quad$ **case** "V":
4: $\quad\quad \overrightarrow{M} \leftarrow \{\overrightarrow{M_i}\}_{i \in \overrightarrow{i} \setminus CCD^I}$
5: $\quad\quad b^V \leftarrow$ SOCOSLO.AVer$(PK, \overrightarrow{M}, \sigma_A)$
6: $\quad\quad \overrightarrow{b} = b^V$
7: $\quad$ **case** "U":
8: $\quad\quad$ **for** $(\sigma_\ell, i_\ell) \in CCD^U$ **do**
9: $\quad\quad\quad \overrightarrow{M} \leftarrow \{\overrightarrow{M_i}\}_{i \in \overrightarrow{i} \setminus CCD^I}$
10: $\quad\quad\quad b_\ell^U \leftarrow$ SOCOSLO.AVer$(PK, \overrightarrow{M}, \sigma_\ell)$
11: $\quad\quad \overrightarrow{b} = \{b_\ell^U\}_{\ell \in |CCD^U|}$
12: $\quad$ **case** "I":
13: $\quad\quad$ **for** $(\sigma_\ell, i_\ell) \in CCD^I$ **do**
14: $\quad\quad\quad b_\ell^I \leftarrow$ SOCOSLO.AVer$(PK, \overrightarrow{M_{i_\ell}}, \sigma_\ell)$
15: $\quad\quad \overrightarrow{b} = \{b_\ell^I\}_{\ell \in |CCD^I|}$
16: **return** $\overrightarrow{b}$

(b) Distillation and selective batch verification

Fig. 4. Signer-Optimal Coarse-grained OSLO (SOCOSLO)

SOCOSLO.Agg(.) is a keyless signature aggregate function with a dual signature combination mode. That is, given tag element $s \in \sigma$ or $R \in \sigma$, it performs additive or multiplication aggregation, respectively.

SOCOSLO.ASig(.) is an aggregate signature generation that signs each entry and sequentially aggregates into a single umbrella signature (i.e., the tag representing all items in the given epoch). The seed $x_D[i]$ is computed once per epoch i (Step 1) and used to derive one-time seeds x_i^j (Step 4). The aggregate signature $s_i^{1,j}$ is computed with only a few hash calls and modular additions plus a modular multiplication (Step 3-5). This makes SOCOSLO the most signer efficient alternative. At the end of epoch i, the logger determines a set of disclosed seeds DS_i via SSO, updates its internal state (Step 6), and outputs the condensed signature σ_i^{1,L_2} (Step 6-7).

SOCOSLO.AVer(.) receives the public key PK, a set of messages $\overrightarrow{M}$, and their corresponding aggregate signature σ as input. The verifier checks if messages comply with the epoch size, and then identifies the format of the aggregate signature to choose component R (Step 1). SOCOSLO.AVer(.) can be invoked by the edge cloud or CSS as the final

verifier. This difference dictates if the aggregate commitment R is included in the initial public key PK or the aggregate signature σ. Below, we will elaborate further that the SOCOSLO.Distill(.) function can be used to verify the entries and then compact them according to a granularity parameter ρ. Hence, if the verification is done during the distillation, the verifier already has $R_i^{1,L_2} \in \overrightarrow{R}$ as part of PK and this value is used in the verification (Step 8). Otherwise, if the verification is run by the CSS, then "R" can be found as a part of the signature in CCD. The verifier retrieves the seeds in the given epoch (Step 4) and then computes the aggregate hash component e (Steps 2-7). Finally, the aggregate signature is verified (Step 8). Fig 2b depicts the mechanism for seed retrieval. It consists of the verifier's view after finishing 6^{th} epoch. It illustrates the request to retrieve the seed of the 3^{rd} epoch.

SOCOSLO Distillation and Selective Batch Verification: The verification involves two entities of our system model (as in Section 3) *(i) Distiller (ii) Cold Storage Server (CSS)*. The cryptographic data structure (CCD) is maintained by CSS and updated by distillers. Fig. 4b formally describes the distillation and batch verification processes. First, both entities initialize CCD as empty sets of signatures. SOCOSLO.Distill(.) updates CCD structure by aggregating valid signatures in CCD^V (Step 2-12). It keeps valid tags according to the granularity parameter ρ. Hence, CSS maintains a condensed tag σ_A, set of umbrella signatures CCD^U, and individual invalid signatures CCD^I (Step 17). SOCOSLO.SeBVer(.) is a selective batch verification routine that can be run in three modes: (i) Mode "V" verify the valid set which consists of one aggregate signature for all valid entries (ii) Mode "U" checks partial umbrella signatures in case the overall authentication (mode "V") is failed. Depending on the application requirements, CCD^U storage overhead can be adjusted according to the granularity parameter ρ. (iii) Mode "I" checks the invalid set by verifying separately each entry. The generic SOCOSLO.AVer(.) enables both the verifier and CSS to use it in the distillation and verification processes, respectively.

4.3 FIne-grained Public-key OSLO (FIPOSLO)

FIPOSLO employs BPV pre-computation [5] to pre-store a constant size of one-time commitments at the logger. Previous works [21] have shown that the incurred storage is negligible for low-end IoT. This is important for immediate and fine-grained verification at the distiller. More importantly, it enables CSS to authenticate log entries individually, thereby achieving accurate investigation and optimal recovery. We describe our fine-grained variant (FIPOSLO) in Fig. 5.

$\underline{(I, sk, PK) \leftarrow \texttt{FIPOSLO.Kg}(1^\kappa, T)}$: Step 1-4 are identical to SOCOSLO.Kg, the rest is as follows:

1: $(\Gamma, v, k) \leftarrow \texttt{BPV.Offline}(1^\kappa, p, q, \alpha)$
2: $sk \leftarrow (y, x_0[0], r_0, \Gamma)$; $PK \leftarrow Y$
3: The system-wide param $I \leftarrow (p, q, \alpha, v, k, L_1, L_2, T, D, \rho, St : (t = 1))$
4: **return** (I, sk, PK)

$\underline{\sigma_I \leftarrow \texttt{FIPOSLO.Sig}(sk, M_t)}$: require $t \leq T$

1: $i \leftarrow \lfloor \frac{t}{L_2} \rfloor$; $j \leftarrow t \bmod L_2$
2: **if** $j = 1$ **then** $x_D[i] \leftarrow \texttt{SC}(x_0[0], 0, 0, D, i)$
3: $x_t \leftarrow H_0(x_D[i] \parallel j)$
4: $(r_t, R_t) \leftarrow \texttt{BPV.Online}(\Gamma, v, k)$
5: $e_t \leftarrow H_0(M_t \parallel x_t) \bmod q$
6: $s_t \leftarrow r_t - e_t \cdot y \bmod q$
7: **if** $j = L_2$ **then** $\sigma_I \leftarrow (s_t, R_t, x_t, DS_t \leftarrow \texttt{SSO}(x_0[0], i))$
8: **else** $\sigma_I \leftarrow (s_t, R_t, x_t)$
9: $St \leftarrow t + 1$
10: **return** σ_t

$\underline{b \leftarrow \texttt{FIPOSLO.AVer}(PK, \overrightarrow{M}, \sigma)}$:

1: **if** $|\overrightarrow{M}| = 1$ **then** $e \leftarrow H(M \parallel x) \bmod q$, where $\overrightarrow{M} = M$
2: **else** execute SOCOSLO.AVer steps 3-8
3: **if** $R = Y^e \cdot \alpha^s \bmod p$ **then return** $b = 1$ **else return** $b = 0$

Fig. 5. FIne-grained Public-key OSLO (FIPOSLO)

FIPOSLO provides various performance advantages at the distiller side. For instance, it offers an immediate verification of each message within an epoch by attaching the seed x_i^j to the signature as shown in FIPOSLO.Sig(.) (Step 7). Unlike SOCOSLO, it permits a $O(1)$ public-key storage at the distiller. That is, the signer generates commitment value R_t via BPV (Step 4) and includes it in the signature (Step 8). Therefore, by introducing the BPV generator, FIPOSLO eliminates the initial $O(L_1)$ public key storage and enables the highest level of granularity by verifying signatures individually.

The distillation and selective batch verification functionalities are similar to SOCOSLO with minor differences and therefore are not repeated. Indeed, the verifier aggregate every signature separately. Thereby, the invalid set CCD^I contains individual signatures (highest granularity) making CSS verify each invalid entry separately. As such, FIPOSLO offer better verification precision than SOCOSLO, but with slightly slower verification time.

5 SECURITY ANALYSIS

We prove that OSLO schemes are A−EU−CMA signature schemes in Theorem 5.1 (in the random oracle model [13]) and Lemma 5.1. We ignore terms that are negligible in terms of κ.

THEOREM 5.1. *$Adv^{A-EU-CMA}_{SOCOSLO(p,q,\alpha)}(t, T', T) \leq Adv^{DL}_{G,\alpha}(t')$, where $t' = O(t) + O(T \cdot (\kappa^3 + RNG))$.*

Proof: Let $\mathcal{A}$ be a SOCOSLO attacker. We construct a *DL-attacker* $\mathcal{F}$ that uses $\mathcal{A}$ as a sub-routine. That is, we set ($b \xleftarrow{\$} \mathbb{Z}_q^*$, $B \leftarrow \alpha^b \bmod p$) as defined in *DL-experiment* (i.e., Definition 2.2) and then run the simulator $\mathcal{F}$ by Definition 3.1 (i.e., A−EU−CMA experiment) as follows:

Algorithm $F(B)$

Setup: $\mathcal{F}$ maintains $\mathcal{LH}$, $\mathcal{LM}$, and $\mathcal{LS}$ to keep track of query results in the duration of the experiment. $\mathcal{LH}$ is a hash list in form of tuples (M_l, h_l, k), where M_l and h_l denote the l^{th} data item queried to $RO(.)$ and its corresponding $RO(.)$ answer, respectively, while $k \in \{0, 1\}$ refers to the selected cryptographic hash function H_k. $\mathcal{LH}[l, 0, k]$ and $\mathcal{LH}[l, 1, k]$ denote the access to the element M_l, h_l, respectively via the hash function H_k. $\mathcal{LM}$ is a list of messages, in which each of its elements $\mathcal{LM}[i]$ is a message set $\overrightarrow{M}_i$ (i.e., the i^{th} batch query). $\mathcal{LS}$ is a signature list that is used to record answers given by SOCOSLO.ASig_{sk}.

- $\mathcal{F}$ creates a simulated SOCOSLO public key PK as follows:
  a) $Y \leftarrow B$ and $x_0[1] \xleftarrow{\$} \{0, 1\}^\kappa$
  b) **for** $l = 1, \ldots, T$ **do**
     i) $R_l \leftarrow Y^{e_l} \cdot \alpha^{s_l} \bmod p$ where $(s_l, e_l) \xleftarrow{\$} \mathbb{Z}_q^*$
  c) **for** $i = 1, \ldots, L_1$ **do**
     i) $R_i^{1,L_2} \leftarrow \prod_{j=1}^{L_2} R_{(i-1)\cdot L_1 + j} \bmod p$
  d) Set (L_1, L_2, ρ) as in SOCOSLO.Kg(.).
  e) Set $PK \leftarrow (Y, \overrightarrow{R})$, where $\overrightarrow{R} \leftarrow \{R_i^{1,L_2}\}_{i=1}^{L_1}$
  f) Set $I \leftarrow (p, q, \alpha, L_1, L_2, T, D = \log L_1)$ and init $l \leftarrow 0,\ i \leftarrow 0$

Execute $\mathcal{A}^{RO(.),\text{SOCOSLO.ASig}_{sk}(.)}(PK)$:

- Queries: $\mathcal{A}$ queries the SOCOSLO.$\text{ASig}_{sk}(.)$ oracle on T messages of her choice. It also queries $RO(.)$ oracle on up to T' messages of her choice. These queries are handled as follows:

  • *How to Handle $RO(.)$ Queries*: $\mathcal{F}$ implements a function $H\text{-}Sim(\delta, k)$ that works as $RO(.)$ as follows: If $\exists l' : \delta \in \mathcal{LH}[l', 0, i]$ then return $\mathcal{LH}[l', 1, i]$. Otherwise, return $h \xleftarrow{\$} \mathbb{Z}_q^*$ as the answer for H_k, insert new tuple (δ, h) to $\mathcal{LH}$

as ($\mathcal{LH}[l,0,k] \leftarrow \delta, \mathcal{LH}[l,1,k] \leftarrow h$) and then update $l \leftarrow l+1$. That is, cryptographic hash functions $H_{k=0,1}$ used in SOCOSLO are modeled as random oracles.

When $\mathcal{A}$ queries $RO(.)$ on a message M_l, $\mathcal{F}$ returns $h_l \leftarrow H\text{-}Sim(M_l, k)$ as described above. Any call for SC or SSO functions invoke H_0 call to traverse OSLOT tree (per Fig. 3) that are all simulated via *H-Sim* as described.

• *How to respond to* $SOCOSLO.ASig_{sk}(.)$ *Queries:*

- For each batch query $\overrightarrow{M_i}$, $\mathcal{A}$ queries SOCOSLO.ASig(.) on $\{M_i^j\}_{j=1}^{L_2}$ of her choice. If $i > L_1$ $\mathcal{F}$ rejects the query (i.e., the query limit is exceeded), else $\mathcal{F}$ continues as follows:
  a) $\mathcal{F}$ computes $x_D[i] \leftarrow \mathtt{SC}(x_0[1], 0, 1, D, i)$.
  b) Initialize $s_i^{1,0} \leftarrow 0$
  c) **for** $j = 1, \ldots, L_2$ **do**
    i) $\mathcal{F}$ sets $x_i^j \leftarrow H\text{-}Sim(x_D[i] \parallel j, 0)$, if $(M_i^j \parallel x_i^j) \in \mathcal{LH}$ then $\mathcal{F}$ *aborts*, else inserts $(M_i^j \parallel x_i^j, 0)$ to $\mathcal{LH}$.
    ii) $\mathcal{F}$ computes $s_i^{1,j} \leftarrow \mathtt{SOCOSLO.Agg}(s_i^{1,j-1}, s_i^j)$
  d) $\mathcal{F}$ sets $\sigma_i \leftarrow \langle s_i^{1,L_2}, DS_i = \mathtt{SSO}(x_0[1], i)\rangle$, inserts $(\overrightarrow{M_i}, \sigma_i)$ to ($\mathcal{LM}$,$\mathcal{LS}$) and $i \leftarrow i+1$.
- Forgery of $\mathcal{A}$: Eventually, $\mathcal{A}$ outputs a forgery on PK as $(\overrightarrow{M}^*, \sigma^*)$, where $\overrightarrow{M}^* = \{\overrightarrow{M_i^*}\}_{i \in \overrightarrow{i}}$ and $\sigma^* = (s^*, DS^*)$. By definition 3.1, $\mathcal{A}$ wins A-EU-CMA experiment for SOCOSLO if $\mathtt{SOCOSLO.AVer}(PK, \overrightarrow{M^*}, \sigma^*) = 1$ and $\overrightarrow{M^*} \notin \mathcal{LM}$ hold. If these conditions hold, $\mathcal{A}$ returns 1, else, returns 0.
- Forgery of $\mathcal{F}$: If $\mathcal{A}$ loses in the A-EU-CMA experiment for SOCOSLO, $\mathcal{F}$ also loses in the *DL* experiment, and therefore $\mathcal{F}$ *aborts* and returns 0. Otherwise, if $\overrightarrow{M^*} \in \mathcal{LH}$ then $\mathcal{F}$ *aborts* and returns 0 (i.e., $\mathcal{A}$ wins the experiment without querying $RO(.)$ oracle). Otherwise, $\mathcal{F}$ continues as follows:

$R \equiv Y^e \cdot \alpha^s \bmod p$ holds for the aggregated variables (R, e, s). That is, given the indices of corresponding previous messages $\overrightarrow{i}$, $\mathcal{F}$ retrieves (s_i, r_i) from $(\mathcal{LS}, \mathcal{LH})$, and then computes $e = \sum_{i \in \overrightarrow{i}} \sum_{j=1}^{L_2} e_{(i-1)\cdot L_1 + j} \bmod q$ and $s = \mathtt{SOCOSLO.Agg}(\{s_i^{1,L_2}\}_{i \in \overrightarrow{i}})$. Moreover, $\mathtt{SOCOSLO.AVer}(PK, \overrightarrow{M^*}, \sigma^*) = 1$ holds, and therefore $R \equiv Y^{e^*} \cdot \alpha^{s^*} \bmod p$ also holds. Note that $\mathcal{A}$ queries $\mathcal{F}$ on L_1 batches and T messages in total. Hence, $\mathcal{F}$ disclosed the root of OSLOT tree, from which required seeds can be derived. $\mathcal{F}$ calls $x_i[D] \leftarrow \mathtt{SR}(DS^*, i)$, $\forall i \in \overrightarrow{i}$, where SR function invoke SC which already simulated via *H-Sim*. It then computes $e^* = \sum_{i \in \overrightarrow{i}} \sum_{j=1}^{L_2} H\text{-}Sim(M_i^{j*} \| x_i^{j*}, 0)$ where $x_i^j \leftarrow H\text{-}Sim(x_i[D] \| j, 0)$. Thus, the following equations hold: $R \equiv Y^e \cdot \alpha^s \bmod p$, $R \equiv Y^{e^*} \cdot \alpha^{s^*} \bmod p$,
$\mathcal{F}$ then extracts $y' = b$ by solving the below modular linear equations (note that only unknowns are y and r), where $Y = B$ as defined in the public key simulation: $r \equiv y' \cdot e + s \bmod q$, $r \equiv y' \cdot e^* + s^* \bmod q$
$B' \equiv \alpha^b \bmod p$ holds, since $\mathcal{A}$'s forgery is valid and non-trivial on $B' = B$. By Def. 2.2, $\mathcal{F}$ wins the *DL-experiment*.

The execution time and probability analysis are as follows:

Execution Time Analysis: In this experiment, the runtime of $\mathcal{F}$ is that of $\mathcal{A}$ plus the time it takes to respond $RO(.)$ queries.

- *Setup phase:* $\mathcal{F}$ draws $2T+1$ random numbers, performs $2T$ modular exponentiations and multiplications. Hence, the total cost of this phase is $(2T) \cdot O(\kappa^3 + \kappa^2) + (2T+1) \cdot \mathtt{RNG}$, where $O(\kappa^3)$ and $O(\kappa^2)$ denote the cost of modular exponentiation and modular multiplication, respectively. RNG denotes the cost of drawing a random number.
- *Query phase:* $\mathcal{F}$ draws $L_1 \cdot \log L_1 \cdot \mathtt{RNG}$ to compute the epoch seeds and $T \cdot \mathtt{RNG}$ to derive one-time random keys. It also draws $T \cdot \mathtt{RNG}$ to handle $\mathcal{A}$'s $RO(.)$ queries. The cost of query phase is bounded as $O(T) \cdot \mathtt{RNG}$.

Therefore, the approximate total running time of $\mathcal{F}$ is $t' = O(t) + O(T \cdot (\kappa^3 + RNG))$.

Success Probability Analysis: $\mathcal{F}$ succeeds if all below events occur.

- $\overline{E1}$: $\mathcal{F}$ does not abort during the query phase.
- $E2$: $\mathcal{A}$ wins the A−EU−CMA experiment for SOCOSLO.
- $\overline{E3}$: $\mathcal{F}$ does not abort after $\mathcal{A}$'s forgery.
- Win: $\mathcal{F}$ wins the A−EU−CMA experiment for *DL-experiment*.
- $Pr[Win] = Pr[\overline{E1}] \cdot Pr[E2|\overline{E1}] \cdot Pr[\overline{E3}|\overline{E1} \wedge E2]$

• *The probability that event $\overline{E1}$ occurs*: During the query phase, $\mathcal{F}$ aborts if $(M_i^j || x_i^j) \in \mathcal{LH},\ 1 \leq i \leq L_1,\ 1 \leq j \leq L_2$ holds, *before* $\mathcal{F}$ inserts $(M_i^j \parallel x_i^j)$ into $\mathcal{LH}$. This occurs if $\mathcal{A}$ guesses x_i^j (before it is released) and then queries $(M_i^j \parallel x_i^j)$ to $RO(.)$ *before* querying it to SOCOSLO.ASig(.). The probability that this occurs is $\frac{1}{2^\kappa}$, which is negligible in terms of κ. Hence, $Pr[\overline{E1}] = (1 - \frac{1}{2^\kappa}) \approx 1$.

• *The probability that event E2 occurs*: If $\mathcal{F}$ does not abort, $\mathcal{A}$ also does not abort since the $\mathcal{A}$'s simulated view is *indistinguishable* from $\mathcal{A}$'s real view (see the indistinguishability analysis). Thus, $Pr[E2|\overline{E1}] = Adv^{\text{A−EU−CMA}}_{\text{SOCOSLO}(p,q,\alpha)}(t, T', T)$.

• *The probability that event $\overline{E3}$ occurs*: $\mathcal{F}$ does not abort if the following conditions are satisfied: (i) $\mathcal{A}$ wins the A−EU−CMA experiment for SOCOSLO on a message M^* by querying it to $RO(.)$. The probability that $\mathcal{A}$ wins without querying M^* to $RO(.)$ is as difficult as a random guess. (ii) After $\mathcal{F}$ extracts $y' = b$ by solving modular linear equations, the probability that $Y' \not\equiv \alpha^{y'} \bmod p$ is negligible in terms κ, since $(Y = B) \in PK$ and SOCOSLO.AVer$(PK, M^*, \sigma^*) = 1$. Hence, $Pr[\overline{E3}|\overline{E1} \wedge E2] = Adv^{\text{A−EU−CMA}}_{\text{SOCOSLO}(p,q,\alpha)}(t, T', T)$. Omitting the terms that are negligible in terms of κ, the upper bound on *A−EU−CMA-advantage of SOCOSLO* is as follows:

$$Adv^{\text{A−EU−CMA}}_{\text{SOCOSLO}(p,q,\alpha)}(t, T', T) \leq Adv^{DL}_{G,\alpha}(t'),$$

Indistinguishability Argument: The real-view of $\overrightarrow{A}_{real}$ is comprised of the public key PK, parameters I, the answers of SOCOSLO.ASig$_{sk}(.)$ (recorded in $\mathcal{LS}$ by $\mathcal{F}$) and the answer of $RO(.)$ (recorded in $\mathcal{LH}$ by $\mathcal{F}$). All these values are generated by SOCOSLO algorithms as in the real system, where $sk = (x_0[1], r_0, y)$ serves as the initial randomness. The joint probability distribution of $\overrightarrow{A}_{real}$ is random uniform as that of sk.

The simulated view of $\mathcal{A}$ is as $\overrightarrow{A}_{sim}$, and it is equivalent to $\overrightarrow{A}_{real}$ except that in the simulation, values (s_l, e_l) for $l = 1, \ldots, T$ are randomly selected from $\mathbb{Z}_q^*$. This then dictates the selection of R_l for $l = 1, \ldots, T$ as random via the public key simulation (step c)-ii). Note that the joint probability distribution of these variables is also random uniformly distributed and is identical to the original signature and hash outputs (since $H_{0,1}$ is modeled as $RO(.)$ via *H-Sim*). SOCOSLO.Distill(.) and SOCOSLO.SeBVer(,) use SOCOSLO.Agg(.) and SOCOSLO.AVer(.), which are invoked in the signature simulation and forgery/extraction phases. Since *CCD* only contains the values produced in the simulation, $\overrightarrow{A}_{sim}$ for SOCOSLO.Distill(.) and SOCOSLO.SeBVer(.) are indistinguishable from that of $\overrightarrow{A}_{real}$. □

LEMMA 5.1. *FIPOSLO is as secure as SOCOSLO.*

Proof: In the sketch proof, we first show that FIPOSLO public key and signature simulations produce random uniformly distributed values as in SOCOSLO. We then show that the forgery and extraction phases in A−EU−CMA experiment for both variants are identical. Finally, we provide an indistinguishability argument for the A−EU−CMA for FIPOSLO.

• *Public Key Simulation*: FIPOSLO.Kg(.) Step 1-4 are identical to that of SOCOSLO, except commitment value R are generated via BPV generator. Therefore, $\mathcal{F}$ runs the public key simulation as in SOCOSLO, expect $\overrightarrow{R}$ is not pre-stored as a part of the public key. All $\{s_l, R_l, e_l\}_{l=1}^T$ values are as in SOCOSLO simulation.

• *Signature Simulation*: $\mathcal{F}$ sets $(\sigma_l = \langle s_l, R_l, x_l \rangle)$, where (s_l, R_l) are as defined above, and (e_l, x_l) are obtained through $RO(.)$ as in SOCOSLO via *H-Sim* function. FIPOSLO.Sig(.) queries are individual, and therefore σ_l is not aggregated via SOCOSLO.Agg(.). The abort conditions in both SOCOSLO and FIPOSLO are the same.

• *Forgery and Extraction*: SOCOSLO and FIPOSLO verifications are identical except for the first step, which identifies if the signature is on a single or batch of messages. If the forgery is an aggregate signature on a batch message, FIPOSLO.AVer(.) verifies it by performing aggregation as in SOCOSLO.AVer(.). Hence, the forgery and extraction are identical, wherein $\mathcal{A}$ might return a batch or individual forgery (σ^*, M^*). $\mathcal{F}$ retrieves (s, R, e) from $\mathcal{LS}$ since R components are the part of signatures but not *PK* (unlike SOCOSLO).

• *Indistinguishability Argument*: $\overrightarrow{A}_{real}$ of FIPOSLO is as in SOCOSLO except that $\{R\}_{l=1}^{T}$ (generated via BPV) are not part of *PK* but in individual signatures $\{\sigma_l = (\langle s_l, R_l, DS_l \rangle\}_{l=1}^{T}$. The joint probability distribution of the values in $\overrightarrow{A}_{real}$ are random uniformly distributed as all derived from *sk* (as in SOCOSLO). Remark that each R_l is also random uniform because the distribution of BPV output r_l is statistically close to the uniform random distribution with an appropriate choice of parameters (v, k) [5]. $\overrightarrow{A}_{sim}$ is identical to $\overrightarrow{A}_{real}$ since public key and signature simulations produce random uniformly distributed values with equal size to $\overrightarrow{A}_{real}$. As in SOCOSLO, FIPOSLO.Distill(.) and FIPOSLO.SeBVer(.) call FIPOSLO.Agg(.) and FIPOSLO.AVer(.), in which *CCD* values are produced by FIPOSLO.Sig(.) and *H-Sim.* □

6 PERFORMANCE ANALYSIS

In this section, we give a detailed performance comparison of OSLO schemes with that of their counterparts.

6.1 Evaluation Metrics and Experimental Setup

Evaluation Metrics: We compare OSLO schemes and their counterparts in terms of *(i)* logger's energy usage, *(ii)* private/public key sizes and signature size, *(iii)* batch verification time and cryptographic cloud storage,

We select our main counterparts such that they reflect the performance of primary families of aggregate signatures (ASs) *(i)* Factorization-based: C-RSA [29] is a AS scheme with a near-optimal signature verification. *(ii)* ECDLP-based: SchnorrQ [6] is one of the fastest EC-based signature (compared to ECDSA/Ed25519 [3]) with a high-performance on embedded devices. FI-BAF [30] is a signer-optimal FAS scheme, which is our closest logger-efficient. *(iii)* Pairing-based: BLS [4] is a multi-user AS scheme that relies on bilinear maps. It is the most compact-storage alternative. Also, we observe BLS is the most deployed signature in recent AS schemes with extended properties (e.g., [16, 26]) in the IoT networks, thereby inheriting similar efficiency advantage of OSLO over BLS.

Parameter Selection: We set the security parameter as $\kappa = 128$. We used FourQ curve [6] and set $|q| = 256$ for the EC-based schemes. The BPV parameters are $(v, k) = (1024, 16)$. The composite modulo size in C-RSA is $|n| = 2048$.

Hardware/Software Configuration: We fully implemented OSLO, for the signer and verifier sides, on a desktop equipped with an Intel i9-9900K@3.6 GHz processor and 64 GB of RAM. On the logger, we implemented OSLO on a low-end device, AVR ATMega 2560 microcontroller, due to its low energy consumption and extensive use in practice. It is equipped with 256KB flash memory, 8KB SRAM, and 4KB EEPROM, with a clock frequency of 16MHz. Our comparisons are based on the following software libraries: *(i)* MIRACL[3] for C-RSA [29] and BLS [4]. *(ii)* FourQlib[4] for the EC-based schemes (i.e., SchnorrQ [6], FI-BAF [30], and OSLO schemes). *(iii)* We used OpenSSL[5] to implement the cryptographic hash functions $H_{i=0,1}$ via SHA-256. We open source our implementation for public testing purposes (see Section 1.2).

[3] https://github.com/miracl/MIRACL
[4] https://github.com/microsoft/FourQlib
[5] https://github.com/openssl/openssl

6.2 Performance Evaluation and Comparison

In this section, we give a performance comparison of OSLO with its counterparts analytically and experimentally.

6.2.1 Analytical Performance Comparison. We present an analytical performance analysis of our schemes with their counterparts. In Table 2, we give the overhead of the main signature functions at the signer and distiller sides w.r.t our evaluation metrics. It also provides the distillation cost w.r.t the failure rate. In Table 3, we provide an analytical comparison for the CSS cost w.r.t the cryptographic storage and the batch verification. We highlight takeaways from our analysis below.

• *Seed Management Overhead Analysis*: One of OSLO's contributions is the seed management (see Section 4.1) that enable both near-optimal signer efficiency and $O(1)$ storage at the CSS and distiller. The amortized seed management overhead of OSLO signing algorithms across T messages is on average one hash call based on the derivation and disclosure of seeds by SC and SSO algorithms, respectively. The resulting average amortized cost is $(\frac{\log L_1 \cdot (4+\log L_1)}{4L_2} \cdot H)$, which corresponds to less than a single hash call, and therefore we conservatively accept it as H in our performance analysis.

The average seed storage is $O(\frac{\log L_1}{L_2})$ at CSS. At the end of last epoch, the signer disclosed the OSLOT root, with which the CSS can verify any prior log entry-signature pair with $O(1)$ final storage.

Table 2. Private/public key and signature sizes, and signature generation/verification costs of OSLO and its counterparts

Scheme	Logger (Signer)			Verifier		
	Sig Gen	Private Key	Sig Size	Public Key	Sig Ver ($\times L$)	Distill & Agg ($\times \tau_S \cdot L$)
SchnorrQ [6]	$2H + Add_q + Mul_q + EMul$	$\|q\|$	$2\|q\|$	$\|q\|$	$H + 1.3 \cdot EMul$	N/A
FI-BAF [30]	$3H + 2Add_q + Mul_q$	$2 \cdot (\|q\| + \kappa)$	$\|q\| + \kappa$	$2L \cdot (\|q\| + \kappa)$	$2 \cdot (H + Add_q) + 2.3 \cdot EMul$	Add_q
C-RSA [29]	$H + Exp_{\|n\|}^{\|d\|}$	$2\|n\|$	$\|n\|$	$2\|n\|$	$H + Exp_{\|n\|}^{\|e\|}$	Mul_n
BLS [4]	$MtP + EMul'$	$\|q\|$	$\|q\|$	$2\|q\|$	$MtP + Pr$	Mul_q
SOCOSLO	$3H + 2Add_q + Mul_q$	$\|q\| + 2\kappa$	$\|q\|$	$L_1 \cdot \|q\|$	$3H + Add_q + 1.3 \cdot EMul/L_2$	$Add_q + EAdd/L_2$
FIPOSLO	$3H + Add_q + Mul_q$ $+k \cdot (Add_q + EAdd)$	$2 \cdot v \cdot \|q\| + \kappa$	$2\|q\| + \kappa$	$\|q\|$	$H + 1.3 \cdot EMul$	$Add_q + EAdd$

Add_q and Mul_q denote modular addition and multiplication, respectively, with modulus q. $EMul$, $EMul'$ are EC scalar multiplication on Four$\mathbb{Q}$ and pairing-based curves, respectively. We used double-point scalar multiplication (e.g., $1.3EMul$ instead of $2EMul$ for Four$\mathbb{Q}$). Pr is a pairing operation. $Exp_{|y|}^{|x|}$ denotes modular exponentiation with exponent x and modulus y. L denotes the batch size of signatures.

Table 3. Storage and computation costs of OSLO variants and its counterparts at the cold storage side

Scheme	Cold Storage Server (CSS)				
	Cold Cryptographic Data (*CCD*)			Verification Valid Time	Umbrella Verification Signatures
	Valid Storage		Invalid Storage		
	Pub Key	Sig	Pub Key		
SchnorrQ [6]	$\|q\|$	$2 \cdot \tau_S \cdot T \cdot \|q\|$	$\|q\|$	$\tau_S \cdot T \cdot (H + 1.3 \cdot EMul)$	$\tau_S \cdot T \cdot (H + 1.3 \cdot EMul)$
FI-BAF [30]	$2 \cdot \tau_S \cdot T \cdot (\|q\| + \kappa)$	$\|q\| + \kappa$	$2 \cdot \tau_F \cdot T \cdot (\|q\| + \kappa)$	$\tau_S \cdot T \cdot (2H + 2Add_q + 1.3 \cdot EMul') + 1.3 \cdot EMul'$	$\tau_S \cdot T \cdot (2H + 2Add_q + 1.3 \cdot EMul') + \frac{1}{\rho} \cdot EMul'$
C-RSA [29]	$2\|n\|$	$\|n\|$	$2\|n\|$	$\tau_S \cdot T \cdot (H + Mul_n) + Exp_{\|n\|}^{\|e\|}$	$\tau_S \cdot T \cdot (H + Mul_n) + \frac{1}{\rho} \cdot Exp_{\|n\|}^{\|e\|}$
BLS [4]	$2\|q\|$	$\|q\|$	$2\|q\|$	$\tau_S \cdot T \cdot (MtP + Mul_q) + Pr$	$\tau_S \cdot T \cdot (MtP + Mul_q) + \frac{1}{\rho} \cdot Pr$
SOCOSLO	$2\|q\|$	$\|q\|$	$\tau_F \cdot L_1 \cdot \|q\|$	$\tau_S \cdot T \cdot (3H + Add_q) + 1.3 \cdot EMul$	$\tau_S \cdot T \cdot (3H + Add_q) + \frac{1.3}{\rho} \cdot EMul$
FIPOSLO	$\|q\|$	$2\|q\|$	$\|q\|$		$\tau_S \cdot T \cdot (3H + Add_q) + \frac{1.3}{\rho} \cdot EMul$

τ_F and τ_S denote, respectively, the valid and invalid rates. The signature size in the invalid set and the verification invalid time are as in in table 2 multiplied by $\tau_F \cdot T$.

Table 4. Bandwidth overhead and signature generation time of OSLO variants and its counterparts at the signer side

Scheme \ Epoch size	Analyticl complexity	Cryptographic payload (KB)					Signing (in sec) (per item)
		16	32	64	128	256	
SchnorrQ [6]	$2 \cdot L_2 \cdot \|q\|$	1	2	4	8	16	0.27
FI-BAF [30]	$\|q\| + \kappa$	0.05	0.05	0.05	0.05	0.05	0.01
C-RSA [29]	$\|n\|$	0.25	0.25	0.25	0.25	0.25	83.26
BLS [4]	$\|q\|$	0.03	0.03	0.03	0.03	0.03	4.08
SOCOSLO	$\|q\| + \kappa$	0.05	0.05	0.05	0.05	0.05	0.01
FIPOSLO	$2 \cdot L_2 \cdot \|q\|$	1	2	4	8	16	0.09

• *Logger (signer):* Table 2 show that the SOCOSLO signature generation only requires 3 hash calls (in average), two and one modular additions and multiplication, respectively. This makes it as lightweight as its most signer efficient counterpart FI-BAF, but with vastly superior performance at CSS. SOCOSLO is significantly more logger efficient than all other alternatives in terms of runtime, with a highly compact signature and small key sizes. FIPOSLO is the second most signer efficient alternative requiring constant number (e.g., 16) of *EAdd* operations. It relies on a pre-computed BPV table, which increases its private key size in exchange for better signing efficiency. Note that the use of BPV can be avoided by accepting single *EMul*, which makes FIPOSLO signing cost equal to that of SchnorrQ. We remind that FIPOSLO accepts extra signing/verification cost over SOCOSLO in exchange for finer granularity.

• *Verifier/CSS:* Table 3 shows the overall performance of OSLO schemes and their counterparts at the server side. The aggregate signatures offer batch ver. for all valid entries (i.e., the success rate $\tau_S = 1$) and umbrella signatures. The batch ver. of all valid entries requires only one *EMul*, while umbrella signatures are based on granularity parameter ρ. In terms of storage, the final public key and aggregate tag sizes of OSLO schemes are as efficient as the most compact alternative BLS, but with a faster runtime since they do not require expensive pairing (Pr) and map-to-point (MtP) operations.

6.2.2 Experimental Evaluation. In Table 1 (see Section 1), we outlined the experimental performance of OSLO schemes and their counterparts at a high level. We now provide details on the signing energy efficiency at the logger side. Then, we outline the computational/storage performance at the cold storage (i.e., CSS).

• *Logger:* Figure 7 showcases the energy usage of OSLO schemes and their counterparts compared to that of sensors typically found in IoT devices. Specifically, we compared the energy usage of a single signature generation with that of sampling via pulse[6] and pressure[7] sensors (10s per sampling time with $1msec$ reading time).

SOCOSLO and FIPOSLO have remarkably low energy usage with 0.88% and 7.38%, respectively, compared to that of the pulse sensor. For SOCOSLO, this translates into 4.5× and 9× lower energy usage than the most efficient standard SchnorrQ and verifier compact BLS, respectively. SOCOSLO is equally energy efficient to FI-BAF, but with substantial gains on the cold storage to be further discussed below. FIPOSLO is the second most energy-efficient alternative, while offering a fine granularity and higher verification efficiency.

Traffic Variation and Bandwidth Usage: Table 1 depicts the signer cryptographic payload, by enabling full aggregation (per epoch). The signer-efficient variant, SOCOSLO, has equal and lesser bandwidth overhead compared to the short signature scheme BLS and the most signer-efficient counterpart FI-BAF, respectively. SOCOSLO is the most suitable during a low-frequency upload since it has a lightweight signature generation with a compact signature size. For a high-frequency upload and/or more available battery lifetime, FIPOSLO offer higher precision by uploading individual signatures to a nearby edge cloud, to be verified and distilled separately. Table 4 depicts the variation of the signer's cryptographic payload under different epoch sizes. Recall that the epoch size represents the number of individual tags to be aggregated. That is, the low-end devices can increase the epoch size when low bandwidth is observed. FIPOSLO have equal cryptographic payload compared to SchnorrQ, while having 3× faster signature generation time. Similarly, SOCOSLO have equal bandwidth overhead compared to the most signer-efficient counterpart FI-BAF but with constant and flexible storage at the distiller and CSS sides. SOCOSLO is considered the best scheme to offer both low bandwidth overhead and fast signature generation on the signer side.

One can adopt the sign-aggregate-forward approach in a hop-by-hop setting, wherein each IoT device signs a set of log entries, aggregates the individual signatures, and forward the resulting tag to the next IoT device. Another possible design

[6]https://pulsesensor.com/
[7]https://cdn-shop.adafruit.com/datasheets/1900_BMP183.pdf

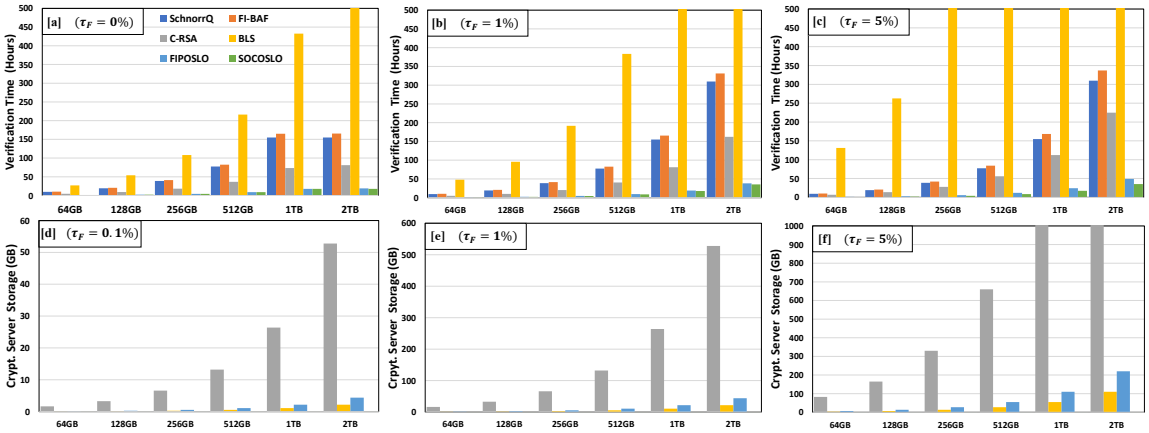


Fig. 6. Comparison of OSLO schemes and their counterparts at the cold storage side

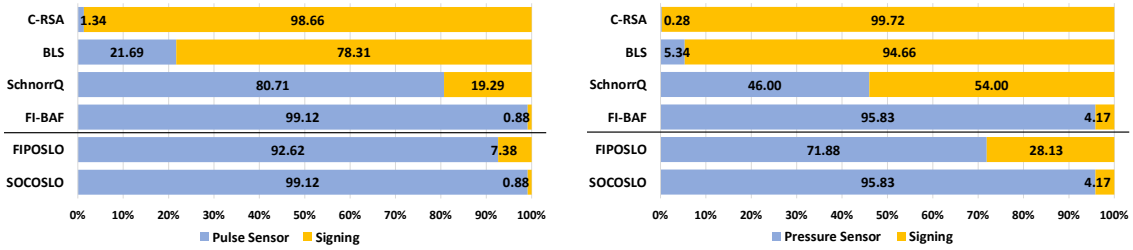


Fig. 7. Energy consumption of OSLO schemes and their counterparts at the logger side

is to employ a clustering approach [9] wherein the IoT devices elect a cluster leader to communicate the authenticated log entries to the distiller. The leader adjusts the cryptographic payload based on the network conditions. For instance, for a set of 2^{10} loggers and 2^8 of epoch size, the bandwidth overhead for a maximum compression across multiple signers is 16.03KB, which is 3× and 171× smaller than single-signer aggregate and non-aggregate approaches, respectively.

• *Distiller:* The distiller storage overhead is more cumbersome than the cold storage server, especially when the hardware is not a resourceful device (e.g., hotspot). The latter receives sets of authentication tags, from a large number of IoT devices, to be verified and aggregated following a pre-determined policy. Thus, the authentication mechanism must have a low-cost verification algorithm and a flexible aggregation capability. By introducing the granularity parameter ρ, the distiller can adjust the tag sizes depending on its resource capabilities and/or the network conditions.

• *CSS:* Fig. 6 shows the verification time and the storage overhead for different sizes of log entry set (each entry is 32 bytes) and failure rates τ_F. Recall that τ_F denotes the ratio of entries with "invalid" verification. As discussed in Sec. 3, in the vast majority of real-world applications, the "invalid" logs (flagged events) are expected to be only a small fraction of the entire log. Therefore, it is preferable to not compress invalid tags, so that they can be attested individually.

In the case of full signature aggregation (i.e., $\tau_F = 0$), we refer the reader to Table 1 that summarizes the verification time and storage advantages of our schemes. In Fig. 6, we further investigate the efficiency of compared schemes for varying failure (τ_F) and granularity (ρ) rates. Specifically, we vary $\tau_F = 0, 1, 5\%$ to observe verification time and storage overhead in Fig. 6-([a],[b],[c]) and Fig. 6-([d],[e],[f]), respectively. We increase the size of log entries from 64 GB (2^{31} entries) to 2 TB (2^{36} entries). We eliminated the counterparts, having linear server storage (i.e., SchnorrQ and FI-BAF) from the storage graphs. In our experiments, for large log sizes, we processed them in small batches and included repeated

disk I/O time in our results. We experimented with ρ from $10^{-7}\%$ to 1%. We observed that ρ has a minimal impact on performance in these margins, and further increase mainly impacts storage with only a slight increase in verification time.

Disk I/O and Cold Storage Cost: Considering a large IoT network where several low-end IoT devices are offloading their authenticated log entries to a remote edge cloud, and ultimately to the cold storage server (CSS). The overall storage at both the edge clouds and CSS become exponentially large and costly. Recall that log files are infrequently accessed data, and therefore it is preferred to store them at cold line solution (e.g., Google cloud solutions [8]), which is relatively low-priced (i.e., $49.15/year for each Terabyte). However, we argue that OSLO is able to offer the best trade-off between low-cost compact server storage, low disk I/O, and fast verification. According to Table 1, SOCOSLO's cryptographic storage overhead is only 0.10KB for 1TB of log entries, whereas it is 3.3TB for the most signer-efficient counterpart FI-BAF. Thus, SOCOSLO have lower disk I/O time and cheaper storage cost since both metrics are directly proportional to the storage overhead. Additionally, OSLO optimizes the disk memory access time by only loading the overall aggregate tag to verify the set of log files. In case the verification is failed, the partially condensed signatures are loaded to locate the invalid log file. The storage cost at the distiller side is more expensive than that of the cold storage server. As the distiller represents the medium between low-end devices and CSS, its stored data is frequently accessed since it receives the authenticated log entries, and distill them after performing the verification. Afterward, it offloads the log files along with the associated cryptographic payload upon finishing a pre-defined set of epochs. This fits the standard storage for data stored within only brief periods of time. Based on the Google cloud solution, the storage cost of one Terabyte is equal to $245.76/year. Similarly, the disk I/O becomes a key metric since the distiller is frequently accessing the stored data.

OSLO schemes prove excellent verification performance for both runtime and storage. They outperform all of their counterparts in both metrics for varying failure rates. For instance, OSLO schemes are significantly faster than their most storage-efficient alternative BLS and much more compact than their fastest counterpart C-RSA with significant speed superiority. Therefore, OSLO schemes are the most efficient alternatives for secure logging in IoT-StaaS applications.

7 CONCLUSION

In this work, we created a new series of aggregate signatures, called OSLO, for secure logging in resource-constrained IoT networks. To the best of our knowledge, OSLO offer the best trade-off between security guarantees and computational/storage efficiency. OSLO embeds a new seed management design via tree-based structure and post-signature disclosure of one-time commitments in EC-based schemes. This enables a compact public key and signature with significant speedup gains for both signing and verification, compared to the state-of-the-art. To avoid losing verification granularity, we introduce an adjustable parameter to keep additional condensed tags after the distillation. This allows more flexibility for verifiers to control the verification precision and the cryptographic data stored on the cold storage servers. We presented an extensive performance analysis with state-of-the-art on both commodity hardware and low-end IoTs. Our experiments show that OSLO represents the best secure logging candidate for numerous recent works in the IoT domain. We formally proved that OSLO is secure and our implementation is open-source for public testing and adaptation.

ACKNOWLEDGMENT

This research is supported by the unrestricted gift from the Cisco Research Award (220159), and the NSF CAREER Award CNS-1917627.

[8] https://cloud.google.com/storage

REFERENCES

[1] Gaspard Anthoine, Jean-Guillaume Dumas, Mélanie de Jonghe, Aude Maignan, Clément Pernet, Michael Hanling, and Daniel S Roche. Dynamic proofs of retrievability with low server storage. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 537–554, 2021.

[2] Giuseppe Ateniese, Roberto Di Pietro, Luigi V Mancini, and Gene Tsudik. Scalable and efficient provable data possession. In *Proceedings of the 4th international conference on Security and privacy in communication netowrks*, pages 1–10, 2008.

[3] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, Sep 2012.

[4] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, sep 2004.

[5] Victor Boyko, Marcus Peinado, and Ramarathnam Venkatesan. Speeding up discrete log and factoring based schemes via precomputations. In *EUROCRYPT '98*, pages 221–235, May 1998.

[6] Craig Costello and Patrick Longa. Schnorrq: Schnorr signatures on fourq. *MSR Tech Report*, 2016.

[7] Rebecca Frank. Risk in trustworthy digital repository audit and certification. *Archival Science*, 22:1–31, 03 2022.

[8] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proc of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.

[9] Mohamed Grissa, Attila A Yavuz, and Bechir Hamdaoui. Trustsas: A trustworthy spectrum access system for the 3.5 ghz cbrs band. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 1495–1503. IEEE, 2019.

[10] Gunnar Hartung. Secure audit logs with verifiable excerpts. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016*, pages 183–199, Cham, 2016. Springer International Publishing.

[11] Gunnar Hartung. Attacks on secure logging schemes. In *Financial Cryptography and Data Security*, pages 268–284, Cham, 2017. Springer International Publishing.

[12] Gunnar Hartung. *Advanced Cryptographic Techniques for Protecting Log Data*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2020.

[13] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2020.

[14] Jihye Kim and Hyunok Oh. Fas: Forward secure sequential aggregate signatures for secure logging. *Information Sciences*, 471:115 – 131, 2019.

[15] Justin J Levandoski, Per-Åke Larson, and Radu Stoica. Identifying hot and cold data in main-memory databases. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 26–37. IEEE, 2013.

[16] Tian Li, Huaqun Wang, Debiao He, and Jia Yu. Permissioned blockchain-based anonymous and traceable aggregate signature scheme for industrial internet of things. *IEEE Internet of Things Journal*, 8(10):8387–8398, 2020.

[17] Xin Li, Huazhe Wang, Ye Yu, and Chen Qian. An iot data communication framework for authenticity and integrity. In *2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pages 159–170. IEEE, 2017.

[18] D. Ma. Practical forward secure sequential aggregate signatures. In *Proceedings of the 3rd ACM symposium on Information, Computer and Communications Security (ASIACCS '08)*, pages 341–352, NY, USA, 2008. ACM.

[19] Di Ma and Gene Tsudik. A new approach to secure logging. *Trans. Storage*, 5(1):2:1–2:21, March 2009.

[20] Giorgia Azzurra Marson and Bertram Poettering. Even more practical secure logging: Tree-based seekable sequential key generators. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 37–54, Cham, 2014. Springer International Publishing.

[21] Muslum Ozgur Ozmen, Rouzbeh Behnia, and Attila A. Yavuz. Energy-aware digital signatures for embedded medical devices. In *7th IEEE Conf. on Communications and Network Security (CNS), Washington, D.C., USA, June*, 2019.

[22] B. Schneier and J. Kelsey. Secure audit logs to support computer forensics. *ACM Transaction on Information System Security*, 2(2):159–176, 1999.

[23] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

[24] Efe U. A. Seyitoglu, Attila A. Yavuz, and Muslum O. Ozmen. Compact and resilient cryptographic tools for digital forensics. In *2020 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, 2020.

[25] Thokozani F. Vallent, Damien Hanyurwimfura, and Chomora Mikeka. Efficient certificate-less aggregate signature scheme with conditional privacy-preservation for vehicular adhoc networks enhanced smart grid system. *Sensors*, 21(9), 2021.

[26] Girraj Kumar Verma, Neeraj Kumar, Prosanta Gope, BB Singh, and Harendra Singh. Scbs: a short certificate-based signature scheme with efficient aggregation for industrial-internet-of-things environment. *IEEE Internet of Things Journal*, 8(11):9305–9316, 2021.

[27] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou. Secure ranked keyword search over encrypted cloud data. In *2010 IEEE 30th international conference on distributed computing systems*, pages 253–262. IEEE, 2010.

[28] Cong Wang, Sherman SM Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE transactions on computers*, 62(2):362–375, 2011.

[29] Attila A. Yavuz. Immutable authentication and integrity schemes for outsourced databases. *IEEE Trans. Dependable Sec. Comput.*, 15(1):69–82, 2018.

[30] Attila A. Yavuz, Peng Ning, and Michael K. Reiter. BAF and FI-BAF: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems. *ACM Trans on Information System Security*, 15(2), 2012.

[31] Attila A. Yavuz and Muslum O. Ozmen. Ultra lightweight multiple-time digital signature for the internet of things devices. *IEEE Transactions on Services Computing*, (01):1–1, jul 5555.

[32] Attila Altay Yavuz, Anand Mudgerikar, Ankush Singla, Ioannis Papapanagiotou, and Elisa Bertino. Real-time digital signatures for time-critical networks. *IEEE Transactions on Information Forensics and Security*, 12(11):2627–2639, 2017.