Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

Lightweight and Resilient Signatures for Cloud-Assisted Embedded IoT Systems

Saif E. Nouma | **Attila A. Yavuz**

[1]Bellini College of AI, Cybersecurity, and Computing, University of South Florida, Tampa, FL, USA

Correspondence
Saif E. Nouma
Email: saifeddinenouma@usf.edu,
Attila A. Yavuz
Email: attilaayavuz@usf.edu

Funding Information
This research was supported by the This work is supported by Cisco Research Award (220159) and National Science Foundation NSF-SNSF 2444615.

Abstract

Digital signatures provide scalable authentication with non-repudiation and are vital tools for the Internet of Things (IoT). Many IoT applications harbor vast quantities of resource-limited devices often used with cloud services. However, key compromises (e.g., physical, malware) pose a significant threat to IoTs due to increased attack vectors and open operational environments. Forward security and distributed key management are critical breach-resilient countermeasures to mitigate such threats. Yet forward-secure signatures are exorbitantly costly for low-end IoTs, while cloud-assisted approaches suffer from centrality or non-colluding semi-honest servers. In this work, we create two novel digital signatures called *Lightweight and Resilient Signatures with Hardware Assistance* (`LRSHA`) and its Forward-secure version (`FLRSHA`). They offer a near-optimally efficient signing with small keys and signature sizes. We synergize various design strategies, such as commitment separation to eliminate costly signing operations and hardware-assisted distributed servers to enable breach-resilient verification. Our schemes achieve magnitudes of faster forward-secure signing and compact key/signature sizes without suffering from strong security assumptions (non-colluding, central servers) or a heavy burden on the verifier (extreme storage, computation). We formally prove the security of our schemes and validate their performance with full-fledged open-source implementations on both commodity hardware and 8-bit AVR microcontrollers.

KEYWORDS
Internet of Things (IoT), authentication, cloud computing, lightweight cryptography, digital signature, forward security.

1 | INTRODUCTION

The Internet of Things (IoT) is a fast-growing networked system that comprises of vast number of resource-constrained devices (e.g., RFID tags, sensors)[1]. The IoT applications involve domains like health, economy, personal, and military. As a result, the security of IoT devices is critical to achieving trustworthy cyber infrastructures. It becomes even more important when cloud servers are becoming the main resort of the sensitive data collected by IoT devices. The data and its surrounding security services are offloaded to the cloud-enabled ecosystems through emerging data analytic applications. For example, digital twins aim to conceive virtual replicas of cyber-physical systems (e.g., humans, institutions)[2] by monitoring the physical entities via IoT equipment (e.g., cameras, sensors, wearable). Healthcare digital twins deploy various wearables on patients to model and analyze medical functions with IoT devices[3]. For instance, resource-limited IoT devices (e.g., pacemakers) send electrical pulses to correct a slow heartbeat rate. Additionally, it enables professionals to monitor the patient's health status to prevent heart failures[4]. Some of these digital twin applications and their security services use cloud assistance[2].

Authentication and integrity are vital requirements to guarantee trustworthy IoT-supported systems[5,6]. Yet, it is a challenging task to offer these security services efficiently due to resource limitations, scalability issues, and advanced security requirements against system breaches[7]. Below, we outline some of *the highly desirable properties* that an ideal authentication and integrity mechanism must achieve for embedded IoT systems:

• *Lightweight and scalable signing*: The vast majority of the embedded IoT devices are resource-limited (e.g., memory, processing, battery)[7]. Hence, the authentication and integrity mechanisms must be lightweight to respect these limitations. Symmetric-key authentication (e.g., HMAC[8]) is computationally efficient. However, due to (pairwise) key distribution and management hurdles, they may not be scalable to large-scale and dynamic IoT applications. Moreover, it does not offer public verifiability and non-repudiation, which are crucial features for dispute resolution and audits. Digital signatures offer scalable and public verifiable authentication via public key infrastructures, which makes them ideal for large-scale IoTs. Yet, standard signatures are costly for low-end IoT devices[9]. The vast majority of signatures require Expensive Operations (ExpOps) such as modular exponentiation[10], Elliptic Curve (EC) scalar multiplication[11] or lattice-operations[12], which are shown to be energy and computation intensive for embedded IoTs[13,14].

Lightweight digital signatures[15,9,16,11] aim to minimize ExpOps to permit efficient signing. However, this generally comes at the cost of limits on the number of signatures[17], excessively large public keys[18], heavy memory consumption[19], weakened security[20] or extra assumptions[17,14]. The lightweight signing becomes especially challenging when additional security features such as compromise-resiliency and frequent signing (e.g., as in digital twins) are needed.

• *Key compromise-resiliency at IoT device*: IoT devices are vulnerable to key compromises via malware or physical access (like a smart-watch left in a public place or a medical handheld device left unattended in a hospital)[21]. Forward-security mitigates the impact of key compromises via key evolution techniques[22,23] by preventing already issued signatures from being forged after private key exposures. Thus, forward security is a vital security primitive for a wide range of applications. Most notably, secure logging from IoT devices to a remote cloud server, where attackers frequently attempt to alter log artifacts following system compromise events[24,25]. However, forward-secure signatures[26,27,28,29] are significantly more expensive than their conventional counterparts. The signing may involve multiple ExpOps with increased key/signature sizes. Even the optimal generic forward-secure transformations incur a logarithmic factor of cost expansion (excluding hidden constants)[30]. Hence, it is an extremely difficult task to create forward-secure signatures that are lightweight for the signer without putting exorbitant overhead on the verifier[18].

• *Compact and resilient operations at IoT device*: (i) The signature/key sizes must be small to respect the memory constraints of embedded devices. (ii) ExpOps require complex arithmetics, which increase the code size and memory footprint. Moreover, they are shown to be more vulnerable to side-channel attacks than simple arithmetic and hash calls[31]. Therefore, it is desirable to limit signing operations to only basic arithmetics and hash calls to avoid these hurdles. (iii) The low-end IoT devices cannot assume a trusted execution environment, and thus the signing logic should not require such special hardware (e.g., unlike[32]).

• *Resiliency at the Cloud-Assistance Services*: Many lightweight signatures leverage cloud-assistance to attain efficiency and/or advanced security[14,17,13,33,34,35]. However, the impacts of such cloud assistance must be assessed carefully. (i) The centralized security assistance is prone to a single point of failure, key escrow, and compromise problems. A distributed architecture can mitigate such risks provided that it does not impede the signer's efficiency. (ii) Decentralized signature assistance assumes semi-honest and collusion-risk-free parties, which may not hold in practice. Moreover, the lack of a cheating detection mechanism (e.g., a party injecting incorrect values) puts the trust at risk. Therefore, it is necessary to provide resiliency not only at the IoT but also at the cloud-assistance side to ensure a higher level of trust and security.

There is a significant gap in the state-of-the-art to achieve above properties simultaneously. It is extremely challenging to devise lightweight authentication primitive that achieves minimal computational overhead with a compact memory footprint. Prior works either incur ExpOps[3] or offload large storage overhead on verifier, and therefore limiting their feasibility and practicality in actual constrained IoT deployments. Below, we discuss most relevant state-of-art digital signatures to our work and outline the desirable properties of proposed schemes.

1.1 | Related Work

We now summarize the state-of-the-art techniques that are most relevant to our work. Our proposed schemes are lightweight forward-secure digital signatures for embedded IoTs with breach-resilient and decentralized verifier cloud-assistance. Hence, we select our counterparts through the lenses of these properties. Given that it is not possible to compare our schemes with every digital signature, we first focus on a broad class of seminal signatures. Later, we capture forward-secure signatures and lightweight constructions relying on special assumptions. Finally, we discuss some signatures with advanced properties that may receive benefit from our schemes or vice versa.

I) Prominent Class of Digital Signatures: Below, we outline some of the most foundational signatures used in IoTs (and compare our schemes with them in Section 6).

• *Elliptic Curve (EC)-based Signatures*: These are currently considered the most suitable class of schemes for resource-limited devices. ECDSA[36] and Ed25519[37] are examples of widely experimented schemes on IoT settings[38]. Despite their merits, they still incur at least one EC scalar multiplication at the signer (e.g.,[16,11,39]). It has been shown that even the most efficient EC signatures can be costly for low-end IoT settings (e.g., 8-bit microcontrollers), with a substantial impact on battery life (e.g.,[9,18]). In our experiments, we re-confirm this fact and then demonstrate that the overhead becomes impractical for low-end devices when advanced features (e.g., forward security) are considered. We further demonstrate the significant performance difference that lightweight signatures can offer over signatures relying on EC scalar multiplications in signing.

• *Pairing-based Signatures*: They offer some of the most compact signature and key sizes along with (cross-user) aggregation capability. Seminal pairing-based schemes like BLS[40] have been used in various applications such as secure routing[41], logging[22], blockchains[42], and IoTs[34]. Despite their compactness, the signature generation uses map-to-point and scalar multiplication, which are significantly slower than EC-based schemes. For example, we have shown that BLS signing is 18× slower than Ed25519, while other studies confirm performance hurdles of BLS on performance-aware networked settings[43,44]. Hence, we will focus on outperforming EC-based signatures in our work.

• *RSA Signatures*: It achieves a fast verification but highly expensive signing and large key sizes. It is even costlier than BLS-based signatures with larger keys, and therefore is not an ideal choice for our applications[10,45].

II) Signatures with Additional Properties and Assumptions:

• *Offline/Online (OO) and Pre-Computed Signatures*: These schemes shift expensive signing operations (e.g., EC-scalar multiplication) to an offline phase, thereby permitting faster signing but with extra storage and transmission. The generic OO schemes involve one-time signatures (e.g., HORS[46]) or special hashes (e.g., **?**), which are expensive for low-end devices. Some signatures such as ECDSA[36] and Schnorr[47] naturally permit commitments to be pre-computed[15], but require the signer to store a pre-computed token per message (i.e., linear storage overhead). Moreover, after depletion, the signer must re-generate these tokens. Due to these memory/bandwidth hurdles and replenishment costs, such OO approaches are not suitable for our target applications.

The BPV techniques[48] permit a signer to store a pre-computed table, from which commitments can be derived with only EC-scalar additions instead of an EC-scalar multiplication. It has been extensively used in low-end IoTs[14,49]. However, recent attacks[50] on BPV demands substantially larger security parameters, which reduces the performance gains. There are also pre-computation methods (e.g.,[51]) that speed up RSA and BLS, which require a large table storage and scalar additions (for BLS). A different line of work eliminates the commitment overhead from the signer by relying on a pre-defined set of one-time public keys at the verifier (e..g,[15,52,18]). Although signer efficient, they limit the number of signatures to be computed and incur a very large public key storage.

• *Lightweight Signatures with Cloud Assistance*: Cloud-assistance is used to elevate security in various protocols[33,34,35,2,53]. Various strategies are used to attain lightweight signatures with cloud assistance. In one line, a set of distributed servers supply verifiers with one-time commitments (e.g.,[14,13]). EC-based schemes[14] achieve high computational efficiency but with large key sizes due to BPV. Moreover, the servers are assumed to be semi-honest and non-colluding, without an explicit authentication on the commitment. Zhang et al.[53] propose a certificateless cloud-assisted digital signature scheme, with security based on the Strong Diffie-Hellman (SDH) assumption. While cloud support offloads the computational burden compared to prior works, the scheme still requires at least one scalar multiplication on resource-constrained IoT signers. Therefore, its efficiency is safely reduced to the EC-based class of signatures.

III) Forward-secure (FS) Digital Signatures: Seminal FS signatures such as Bellare-Miner[29], Itkis-Reyzin[28], and Abdalla-Reyzin[54] led to several asymptotically efficient designs (e.g.,[22,23]). However, they all require (generally multiple) ExpOps at the signing with large signatures, and therefore are not suitable for our use cases. Another alternative is to transform efficient EC-based signatures into FS with generic transformations (e.g., MMM[30]). MMM is an asymptotically optimal scheme that can transform any signature into an FS variant. However, it requires multiple calls to the underlying signing and key generation, leading to highly costly operations[23]. Akin to MMM, FS signatures such as XMSS[55] (and signer-efficient variants[27]) also rely on tree structures to attain multiple-time signatures from one-time hash-based schemes. However, as shown in our experiments, they are still magnitudes costlier than our constructions. Finally, to transform one-time signatures with multiple-time FS schemes, there are FS OO techniques (e.g.,[18]) and cloud-assisted approaches (e.g.,[13,17]). Despite their signing efficiency, they inherit limited usability, large public keys, and/or risks of single-point failure, as discussed above. Hence, there is a crucial need for FS signatures that avoid ExpOps without strict limits on the number of signatures or heavy one-time public key storage.

IV) Lighweight Signatures and Other Cryptographic Constructions with Trusted Execution Environment (TEE)-supported Cloud Assistance: Numerous digital signature schemes harness TEE-enabled cloud infrastructures to offload the computational overhead inherent in traditional software-only systems. For example, SCB [32] is a TEE-supported asymmetric cryptographic constructions based on symmetric cryptographic primitives. Although it significantly reduces ExpOps by leveraging lightweight symmetric algorithms, it assumes TEE availability on both endpoints (e.g., signer and verifier in a digital signature setting). Therefore, SCB is infeasible in IoT environments where signers are deemed to be resource-constrained devices (e.g., 8-bit microcontrollers). Another recent digital signature scheme delegates the generation of commitments and public keys to a TEE-supported cloud server [17]. Although it offers lightweight signing and small key sizes, it incurs large signatures. Moreover, this cloud assistance relies on a centralized TEE architecture, therefore prone to the key escrow and central root of trust vulnerabilities (as discussed in [56,53]). Conversely, TEEs have been instrumental in numerous cryptographic constructions, beyond digital signatures, most notably Oblivious Random Access Memory (ORAM) in Searchable Encryption (SE) [57], Attribute-Based Encryption (ABE) [58], and secure Multi-Party Computation (MPC) protocols [59]. These works complement ours and can serve as a privacy-enhancing layer.

V) Alternative Signatures with Potential Extensions: Identity-based and certificateless signatures [60] mitigate the overhead of certificate transmission and verification. They have been used in various IoT settings (e.g., [34,61,11]). Despite their merits, they still require ExpOp(s) at signers. It is possible to extend our schemes into these settings via proper transformations (e.g., [62]).

Puncturable digital signatures (e.g., [63,64]) involve key update strategies and can be built from ID-based signatures (e.g., [63]). Multi-signatures (e.g., [65]) and threshold signatures (e.g., [66]) can also be extended into forward-secure settings. However, our schemes are signer non-interactive, single-signer, and signer-optimal constructions, and therefore those signatures are not their counterparts. Finally, besides digital signatures, there are myriad other authentication techniques for IoTs, including but not limited to, multi-factor and/or user authentication (e.g., [35]). These works are complementary to ours. Our proposed schemes can serve as a building block when used as a signature primitive. Moreover, our schemes can support a myriad of network services, such as spectrum sensing [67]. Note that, we strictly aim to guarantee the public verifiability and non-repudiation of the embedded device by itself, but only let the cloud support the verification. Hence, the cloud-assisted authentication methods that defer the signature generation to the cloud (e.g., [68]) are also out of our scope. Finally, the protocols that offer confidentiality and availability for IoTs are out of our scope.

Given the limitations inherent in digital signature schemes and the research gap in simultaneously achieving the desirable performance and security goals, there is a critical need for a lightweight and resilient digital signature that leverage the existence of cloud servers and advancements in secure hardware (e.g., TEE) present in IoT ecosystems. This work attempts to address the following research questions:

RQ1. *Can we design a digital signature scheme that ensures computational efficiency and minimal energy consumption, enabling practical deployment on IoT endpoints with memory and power constraints?*

RQ2. *Can such a scheme be realized without incurring substantial storage overhead at the verifier or relying on strong, often impractical, trust assumptions about third-party cloud-assisted servers?*

RQ3. *Is it possible to leverage TEE support to offload ExpOps during signature generation while avoiding the central root of trust, single-point of failure, and rogue key attacks?*

RQ4. *Can we achieve FS for signers and breach resilience in cloud entities without affecting performance efficiency?*

1.2 | Our Contributions

In this paper, we propose two new digital signatures called *Lightweight and Resilient Signatures with Hardware Assistance* (`LRSHA`) and its forward-secure version as *Forward-secure* `LRSHA` (`FLRSHA`). Our schemes provide lightweight signing and near-optimally efficient forward security with small keys and signature sizes. They achieve this without relying on strong security assumptions (such as non-colluding or central servers) or imposing heavy overhead on the verifier (like linear public key storage or extreme computation overhead). Our methods introduce and blend different design strategies in a unique manner to achieve these advanced features simultaneously. Some key strategies include using the commitment separation method to eliminate expensive commitment generations and EC operations from the signer and utilizing distributed TEE-supported cloud-assisted servers to provide robust and dependable verification support at the verifier. We give the details of our schemes in Section 4 and outline the main idea and design principles of our proposed schemes further below:

Main Idea. The main performance bottleneck of EC-based signature generation arises from the elliptic curve scalar multiplication needed to generate commitments. Prior works (e.g., [75,17,14]) attempted to mitigate this cost through various commitment management strategies—such as using non-colluding distributed servers [14], a centralized root-of-trust server [17], or accepting linear storage overhead at the verifier side [75]. Our proposed scheme, LRSHA, removes the commitment-generation burden from the signer by introducing a distributed commitment strategy built upon a set of TEE-supported servers, called ComC servers. Each ComC server securely provides authenticated one-time EC commitments to verifiers on-demand or in batch during an offline phase. Furthermore, we extend LRSHA to a forward-secure variant, FLRSHA, which achieves breach resilience against key-compromise attacks through efficient key evolution without relying on heavy certification structures. To the best of our knowledge, (F)LRSHA are the first EC-based signature scheme that simultaneously achieves highly efficient signing, authenticated distributed verification, and collusion resilience. We outline the desirable properties of (F)LRSHA further below:

• *High Signing Computational Efficiency*: LRSHA and FLRSHA provide a near-optimal signature generation with compact key sizes, thanks to the elimination of ExpOps from signing. LRSHA outperform their counterparts by being 46× and 4× faster than Ed25519 [37] and its most signer-efficient counterpart HASES [17] on 8-bit AVR ATMega2560 microcontroller. The signing of FLRSHA is also faster than the forward-secure HASES, with a magnitude smaller signature size and without the central root of trust and key escrow limitations. The private key size of LRSHA is several magnitudes smaller than its fastest counterparts (e.g., [14,69,51]) that rely on pre-computed tables.

• *Forward Security and Tighter Reduction*: *(i) Forward Security:* As discussed in Section 1.1, FS signatures are generally significantly more expensive than their plain variants and not suitable for low-end devices. To the best of our knowledge, FLRSHA is one of the most efficient FS signatures in the literature, whose cost is almost as efficient as few symmetric MAC calls, with a compact signature and key sizes. These properties make it several magnitudes more efficient than existing FS signatures (e.g., [27,30]) and an ideal choice to be deployed on embedded IoTs. *(ii) Tighter Reduction:* Unlike traditional Schnorr-based signatures (e.g., [37]), the proof of our schemes avoids the forking lemma, thereby offering a tighter reduction factor.

• *Compact, Simple and Resilient Signing*: Our signing only relies on a few simple modular additions and multiplication, and cryptographic hash calls. Hence, it does not require intricate and side-channel-prone operations such as EC-scalar multiplication, rejection sampling, and online randomness generation. This simplicity also permits a small code base and memory footprint. These properties increase the energy efficiency of our schemes and make them more resilient against side-channel attacks (e.g., [70,31]) targeting complex operations, which are shown to be problematic, especially on low-end IoT devices.

• *Collusion-Resilient and Authenticated Distributed Verification with Offline-Online Capabilities:* Our technique avoids single-point failures and improves the collusion and breach robustness of the verification servers by using a distributed and hardware-assisted signature verification strategy. Furthermore, before signature verification, commitments can be generated and verified offline. In contrast to certain counterpart schemes that depend on servers providing assistance only in a semi-honest, non-colluding, or merely central manner, our systems are able to identify malicious injections of false commitments and provide far quicker signature verification during the online phase. All these characteristics allow our schemes to have reduced end-to-end delays and more reliable authentication than their counterparts with server-aided signatures.

• *Full-Fledged Implementation, Comparison, and Validation*: We implemented our schemes, compared them with our counterparts, and validated their efficiency on both commodity hardware and resource-constrained embedded devices. We open-sourced our full-fledged implementations for reproducibility and future adaptations: https://github.com/saifnouma/lrsha

2 | PRELIMINARIES

The acronyms and notations are described in Table 1.

Definition 1. A digital signature scheme SGN is a tuple of three algorithms (Kg, Sig, Ver) defined as follows:

- $(sk, PK, I) \leftarrow \texttt{SGN.Kg}(1^{\kappa})$: Given the security parameter κ, it returns a private/public key pair (sk, PK) and system parameters I (implicit input to all other interfaces).
- $\sigma \leftarrow \texttt{SGN.Sig}(sk, M)$: Given the private key sk and a message M, the signing algorithm returns signature σ.
- $b \leftarrow \texttt{SGN.Ver}(PK, M, \sigma)$: Given the public key PK, message M, and a signature σ, it outputs a bit b (if $b = 1$, the signature is valid, otherwise invalid).

TABLE 1 List of acronyms and notations

Notation/Acronym	Description
MCU	Micro-Controller Unit
TEE	Trusted Execution Environment (also called Secure enclave)
FS	Forward Security
(EC)DLP	(Elliptic Curve) Discret Logarithm Problem
ROM	Random Oracle Model
EUCMA	Existential Unforgeability against Chosen Message Attack
(F)HDSGN	FS Hardware-assisted Distributed Signature
(F)LRSHA	FS Lightweight and Resilient Signature with Hardware Assistance
PRF	Pseudo-Random Function
PPT	Probabilistic Polynomial Time
sk/PK	Private/Public key
r/R	Random nonce/Public commitment (Schnorr-like schemes)
ComC / C_j	Commitment Construct and signature C_j on a commitment
$\mathcal{S}^\ell$ / a^ℓ	Identity of the ℓ^{th} ComC server and its private key set
sk'^ℓ/PK'^ℓ	Private/public keys for certification for ℓ^{th} ComC server $\mathcal{S}^\ell$
L	Number of ComC servers in our system model
j/J	The algorithm state, and the maximum number of forward-secure signatures to be generated
$\|\|$ / $\|x\|$	String concatenation and bit length of a variable
$x \xleftarrow{\$} \mathcal{X}$ / κ	Random selection from a set $\mathcal{X}$ and security parameter
x_j^ℓ	Variable of server $\mathcal{S}^\ell$ for state j
$x_j^{\ell_1,\ell_2}$	Aggregate variable of $(x_j^{\ell_1}, x_j^{\ell_1+1}, \ldots, x_j^{\ell_2})$, where $\ell_2 \geq \ell_1$
$\vec{x}$	Vector contains finite set of elements $\{x_i\}_{i=1}^n$ where $n = \|\vec{x}\|$ represents of the number of elements in the vector
$\{0,1\}^*$	Set of binary strings of any finite length
$\{q_i\}_{i=0}^n$	Set of items q_i for $i = 0, \ldots, n$
$H : \{0,1\}^* \to \{0,1\}^\kappa$	Cryptographic hash function
$H^{(k)}(.)$	Return the output of k hash evaluations on the same input

Definition 2. A forward-secure signature `FSGN` has four algorithms (`Kg`, `Upd`, `Sig`, `Ver`) defined as follows:

- $(sk_1, PK, I) \leftarrow$ `FSGN.Kg`$(1^\kappa, J)$: Given κ and the maximum number of key updates J, it returns a private/public key pair (sk_1, PK) and system parameters I (including state $St \leftarrow (j = 1)$).
- $sk_{j+1} \leftarrow$ `FSGN.Upd`(sk_j, J): If $j \geq J$ then abort, else, given sk_j, it returns sk_{j+1}, delete sk_j and $j \leftarrow j + 1$.
- $\sigma_j \leftarrow$ `FSGN.Sig`(sk_j, M_j): If $j > J$ then abort, else it computes σ_j with sk_j on M_j, and $sk_{j+1} \leftarrow$ `FSGN.Upd`(sk_j, J).
- $b_j \leftarrow$ `FSGN.Ver`(PK, M_j, σ_j): If $j > J$ then abort, else given PK, M_j, and σ_j, it outputs a validation bit b_j (if $b_j = 1$, the signature is valid, otherwise invalid).

Definition 3. Let $\mathbb{G}$ be a cyclic group of order q, α be a generator of $\mathbb{G}$, and `DLP` attacker $\mathcal{A}$ be an algorithm that returns an integer in $\mathbb{Z}_q^*$. We consider the following experiment:

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

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

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

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

$$Adv^{DL}_{\mathbb{G},\alpha}(\mathcal{A}) = Pr[Expt^{DL}_{\mathbb{G},\alpha}(\mathcal{A}) = 1]$$

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

$$Adv^{DL}_{\mathbb{G},\alpha}(t') = \max_{\mathcal{A}}\{Adv^{DL}_{\mathbb{G},\alpha}(\mathcal{A})\}, \text{ where the maximum is over all } \mathcal{A} \text{ having time complexity } t.$$

Remark 1. Although we give some definitions for DLP, our implementation is based on Elliptic Curves (EC) for efficiency, and the definitions also hold under ECDLP[71].

3 | SYSTEM, THREAT, AND SECURITY MODELS

3.1 | System Model

As shown in Figure 1, our system model has three entities:

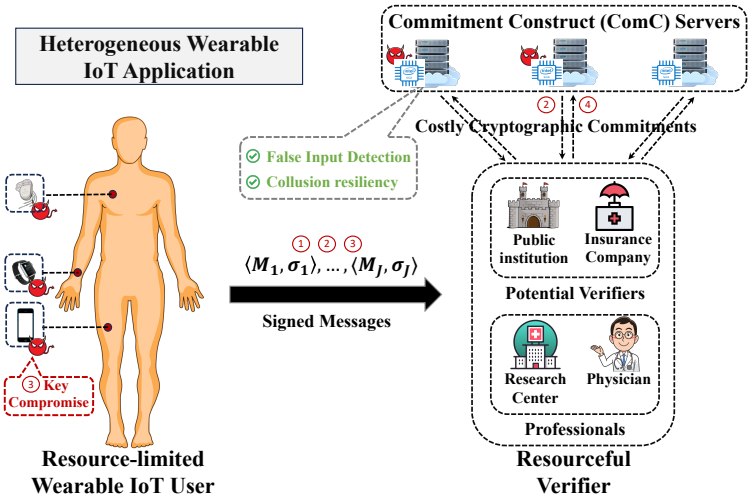

FIGURE 1 Our System Model

1) *Resource-limited Signer:* We focus on low-end IoT devices as signers. As depicted in Figure 1, we consider a secure wearable medical IoT application, in which the patient is equipped with sensors (e.g., a pacemaker) and wearable devices (e.g., a smart watch) that generate digital signatures on sensitive data to be authenticated by verifiers.
2) *Verifiers:* They can be any entity receiving the message-signature pair from signer. In our applications, verifiers (e.g., doctors, researchers, insurance companies) are equipped with commodity hardware (e.g., a laptop).
3) *ComC Servers:* ComC servers $\mathcal{S} = (\mathcal{S}^1, \ldots, \mathcal{S}^L)$, where each server is equipped with a TEE (i.e., secure enclave). We used Intel SGX due to its wide availability (e.g., Microsoft Azure). However, our model can be implemented with any TEE (e.g., ARM TrustZone, Sanctum).

3.2 | Threat and Security Model

Our threat model is based on adversary with the following capabilities:

1) *Passive attacks*: Monitor and interpret the output of the cryptographic interfaces sent from the IoT end devices and/or the ComC servers.
2) *Active attacks*: Attempt to intercept, forge, and modify messages, signatures and auxiliary values (e.g., commitments) sent from IoT devices and ComC servers.
3) *Key Compromise - resource-limited IoT side*: Attempt breaching the device to extract the cryptographic secret keys[7]. The attacker is then able to forge past signatures and impersonate the device by generating future signatures.
4) *Breach attempts on assisting clouds for verification services*: Attempt to gain access to assisted cloud services to tamper with the protocol such that: (i) Inject incorrect commitments. (ii) Forge certificates of the commitments. (iii) Force the cloud to collude (e.g., expose the secret keys).

Below, we first define the interfaces of our proposed schemes, and then present their security model that captures the above threat model as follows:

Definition 4. A hardware-assisted distributed digital signature scheme HDSGN consists of four algorithms (Kg, ComC, SGN, Ver) defined as follows:

- $(sk, PK, \vec{a}, I) \leftarrow \texttt{HDSGN.Kg}(1^\kappa, L)$: Given κ and the number of ComC servers L, it returns a private/public key pair (sk, PK), system parameter $(I, St \leftarrow j = 1)$, and private key of each ComC server $\vec{a} = \{a^\ell\}_{\ell=1}^L$.
- $(\vec{R}_j, \vec{C}_j) \leftarrow \texttt{HDSGN.ComC}(\{a^\ell\}_{\ell=1}^L, j)$: Given $St = j$ and $\vec{a}$, each server $\mathcal{S}^\ell$ generates a commitment R_j^ℓ and its signature $C_j^\ell = \texttt{SGN.Sig}_{a^\ell}(R_j^\ell)$. HDSGN.ComC returns $(\vec{R}_j = \{R_j^\ell\}_{\ell=1}^L, \vec{C}_j = \{C_j^\ell\}_{\ell=1}^L)$ as output.
- $\sigma_j \leftarrow \texttt{HDSGN.Sig}(sk, M_j)$: Given sk and a message M_j, it returns a signature σ_j and $j \leftarrow j + 1$.
- $b_j \leftarrow \texttt{HDSGN.Ver}(PK, M_j, \sigma_j)$: Given PK, M_j, and its signature σ_j, the verification algorithm calls $(\vec{R}_j, \vec{C}_j) \leftarrow \texttt{HDSGN.ComC}(\{a^\ell\}_{\ell=1}^L, j)$, and then it outputs a bit b_j (if $b_j = 1$, the signature is valid, otherwise invalid).

Definition 5. A forward-secure and hardware-assisted distributed digital signature scheme FHDSGN consists of five algorithms (Kg, ComC, Upd, Sig, Ver) defined as follows:

- $(sk_1, PK, \vec{a}, I) \leftarrow \texttt{FHDSGN.Kg}(1^\kappa, J, L)$: Given κ, L, and the maximum number of signatures J to be produced, it returns (sk_1, PK), $(I, St \leftarrow j = 1)$, and $\vec{a} = \{a^\ell\}_{\ell=1}^L$.
- $(\vec{Y}_j, \vec{R}_j, \vec{C}_j) \leftarrow \texttt{FHDSGN.ComC}(\vec{a}, j)$: Given $\vec{a}$ and state j, it returns a set of public key and commitment set $(\vec{Y}_j = \{Y_j^\ell\}_{\ell=1}^L, \vec{R}_j = \{R_j^\ell\}_{\ell=1}^L)$, a forward-secure signature on each pair as $\vec{C}_j = \{\texttt{FSGN.Sig}_{a^\ell}(Y_j^\ell \| R_j^\ell)\}_{j=1}^L$, and returns $(\vec{R}_j, \vec{C}_j)$.
- $sk_{j+1} \leftarrow \texttt{FHDSGN.Upd}(sk_j, J)$: As in Definition 2 update.
- $\sigma_j \leftarrow \texttt{FHDSGN.Sig}(sk_j, M_j)$: As in Definition 2 signing.
- $b_j \leftarrow \texttt{FHDSGN.Ver}(PK, M_j, \sigma_j)$: If $j > J$ then abort. Otherwise, given the public key PK, a message M_j, and its signature σ_j, the verification algorithm calls $(\vec{Y}_j, \vec{R}_j, \vec{C}_j) \leftarrow \texttt{FHDSGN.ComC}(\vec{a}, j)$, and then it outputs a bit b_j (if $b_j = 1$, the signature is valid, otherwise invalid).

The standard security notion for a digital signature SGN is the Existential Unforgeability against Chosen Message Attack (EU−CMA)[14]. It captures a Probabilistic Polynomial Time (PPT) adversary $\mathcal{A}$ aiming at forging signed messages. It corresponds to capabilities (1-2) stated in the threat model (passive or active attacks on message-signature pairs).

Definition 6. EU−CMA experiment $Expt_{\texttt{SGN}}^{\texttt{EU-CMA}}$ for SGN is as follows (in random oracle model (ROM)[72]):

- $(sk, PK, I) \leftarrow \texttt{SGN.Kg}(1^\kappa)$
- $(M^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.),\ \texttt{SGN.Sig}_{sk}(.)}(PK)$

$\mathcal{A}$ wins the experiment if $\texttt{SGN.Ver}(PK, M^*, \sigma^*) = 1$ and M^* was not queried to $\texttt{SGN.Sig}_{sk}(.)$ oracle. The EU−CMA advantage of $\mathcal{A}$ is defined as $Adv_{\texttt{SGN}}^{\texttt{EU-CMA}}(\mathcal{A}) = \Pr[Expt_{\texttt{SGN}}^{\texttt{EU-CMA}} = 1]$. The EU−CMA advantage of SGN is defined as $Adv_{\texttt{SGN}}^{\texttt{EU-CMA}}(t, q_H, q_s) = \max_{\mathcal{A}} Adv_{\texttt{SGN}}^{\texttt{EU-CMA}}(\mathcal{A})$. Note that the maximum is evaluated across possible $\mathcal{A}$ with time complexity t and maximum number of running queries q_H and q_s to the $RO(.)$ and $\texttt{SGN.Sig}_{sk}(.)$ oracles, respectively.

1. *Random Oracle RO*(.): It handles $\mathcal{A}$'s hash queries on any message M by returning a randomly uniformly distributed output $h \leftarrow RO(M)$. All cryptographic hashes used in our schemes are modeled as $RO(.)$[72].
2. $\texttt{SGN.Sig}_{sk}(.)$: It provides a signature σ on any queried message M computed as $\sigma \leftarrow \texttt{SGN.Sig}_{sk}(M)$.

We follow the formal security model of a hardware-assisted distributed digital signature scheme (HDSGN) as the Hardware-assisted Distributed Existential Unforgeability against Chosen Message Attack (HD−EU−CMA). It captures the capabilities (1-2, 4) in our threat model, including $\mathcal{A}$'s potential attacks on the ComC servers.

Definition 7. HD−EU−CMA experiment $Expt_{\texttt{HDSGN}}^{\texttt{HD-EU-CMA}}$ for a hardware-assisted distributed digital signature HDSGN = (Kg, ComC, Sig, Ver) is defined as follows:

- $(sk, PK, \{a^\ell\}_{\ell=1}^L, I) \leftarrow \texttt{HDSGN.Kg}(1^\kappa, L)$
- $(M^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.),\ \texttt{HDSGN.Sig}_{sk}(.),\ \texttt{HDSGN.ComC}_{\vec{a}}(.)}(PK)$

, where $\vec{a} = \{a^\ell\}_{\ell=1}^L$, denotes private key material of ComC server $\{\mathcal{S}^\ell\}_{\ell=1}^L$, respectively.

$\mathcal{A}$ wins the experiment if $\texttt{HDSGN.Ver}(PK, M^*, \sigma^*) = 1$ and M^* was not queried to $\texttt{HDSGN.Sig}_{sk}(.)$. The HD−EU−CMA advantage of $\mathcal{A}$ is defined as $Adv^{\text{HD-EU-CMA}}_{\text{HDSGN}}(\mathcal{A}) = \Pr[Expt^{\text{HD-EU-CMA}}_{\text{HDSGN}} = 1]$. The HD−EU−CMA advantage of HDSGN is defined as $Adv^{\text{HD-EU-CMA}}_{\text{HDSGN}}(t, q_H, q_s) = \max_{\mathcal{A}} Adv^{\text{HD-EU-CMA}}_{\text{HDSGN}}(\mathcal{A})$ with all possible adversary $\mathcal{A}$ having time complexity t and maximum queries q_H to $RO(.)$ and q_s to both of $\texttt{HDSGN.Sig}_{sk}(.)$ and $\texttt{HDSGN.ComC}_{\vec{a}}(.)$.

1. Oracles $RO(.)$ and $\texttt{HDSGN.Sig}_{sk}(.)$ works in as Def. 6.
2. $ComC_{\vec{a}}(.)$: Given state j, it generates a public commitment $\{R_j^\ell\}_{\ell=1}^L$ and corresponding signature $\{C_j^\ell \leftarrow \texttt{SGN.Sig}_{a^\ell}(R_j^\ell)\}_{\ell=1}^L$ for each ComC server $\{\mathcal{S}^\ell\}_{\ell=1}^L$.

The standard security notion for a forward-secure digital signature scheme FSGN is the Forward-secure EU−CMA (F−EU−CMA)[29]. It captures the key compromise capability 3) in our threat model.

Definition 8. F−EU−CMA experiment $Expt^{\text{F-EU-CMA}}_{\text{FSGN}}$ for a forward-secure signature scheme FSGN = (Kg, Upd, Sig, Ver) is defined as follows:

- $(sk_1, PK, I) \leftarrow \texttt{FSGN.Kg}(1^\kappa, J)$
- $(M^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.),\ \texttt{FSGN.Sig}_{sk_j}(.),\ \texttt{Break-In(.)}}(PK)$

$\mathcal{A}$ wins the experiment if $\texttt{FSGN.Ver}(PK, M^*, \sigma^*) = 1$ and M^* was not queried to $\texttt{FSGN.Sig}_{sk_j}(.)$. The F−EU−CMA advantage of $\mathcal{A}$ is defined as $Adv^{\text{F-EU-CMA}}_{\text{FSGN}}(\mathcal{A}) = \Pr[Expt^{F-EU-CMA}_{\text{FSGN}} = 1]$. The F−EU−CMA advantage of FSGN is defined as $Adv^{\text{F-EU-CMA}}_{\text{FSGN}}(t, q_H, q_s, 1) = \max_{\mathcal{A}} Adv^{\text{F-EU-CMA}}_{\text{FSGN}}(\mathcal{A})$, with all possible $\mathcal{A}$ having time complexity t and q_H, q_s, and one queries to $RO(.)$, $\texttt{FSGN.Sig}_{sk_j}(.)$, and Break−In(.) oracles, respectively. $RO(.)$ and $\texttt{FSGN.Sig}_{sk_j}(.)$ oracles are as in Definition 6. Break−In(.) oracle returns the private key sk_{j+1} if queried on state $1 \leq j < J$, else aborts.

We follow the formal security model of a forward-secure hardware-assisted distributed digital signature scheme (FHDSGN) as Forward-secure HD−EU−CMA (FHD−EU−CMA). It combines both security definitions and captures all abilities of the attacker (1-4) in our threat model. *This offers improved security over non-forward secure signature and/or cloud-assisted signature schemes that only rely on a semi-honest model.*

Definition 9. FHD−EU−CMA experiment $Expt^{\text{FHD-EU-CMA}}_{\text{FHDSGN}}$ for a forward-secure and hardware-assisted signature FHDSGN = (Kg, ComC, Upd, Sig, Ver) is defined as follows:

- $(sk_1, PK, I) \leftarrow \texttt{FHDSGN.Kg}(1^\kappa, J, L)$
- $(M^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.),\ \texttt{FHDSGN.Sig}_{sk_j}(.),\ \texttt{FHDSGN.ComC}_{\vec{a}}(.),\ \texttt{Break-In(.)}}(PK)$

$\mathcal{A}$ wins the experiment if $\texttt{FHDSGN.Ver}(PK, M^*, \sigma^*) = 1$ and M^* was not queried to $\texttt{FHDSGN.Sig}_{sk_j}(.)$ oracle. The FHD−EU−CMA advantage of $\mathcal{A}$ is defined as $Adv^{\text{FHD-EU-CMA}}_{\text{FHDSGN}} = \Pr[Expt^{FHD-EU-CMA}_{\text{FHDSGN}} = 1]$. The FHD−EU−CMA advantage of FHDSGN is defined as $Adv^{\text{FHD-EU-CMA}}_{\text{FHDSGN}}(t, q_H, q_s, 1) = \max_{\mathcal{A}} Adv^{\text{FHD-EU-CMA}}_{\text{FHDSGN}}(\mathcal{A})$, with all possible $\mathcal{A}$ having time complexity t and maximum queries equal to q_H, q_s, and one to $RO(.)$, both of $\texttt{FHDSGN.Sig}_{sk_j}(.)$ and $\texttt{FHDSGN.ComC}_{\vec{a}}(.)$, and Break−In(.), respectively. All oracles behave as in Definition 7, except forward-secure signatures are used for signing in $\texttt{FHDSGN.Sig}_{sk_j}(.)$, $\texttt{FHDSGN.ComC}_{\vec{a}}(.)$ and Break−In(.) (as in Definition 8).

Assumption 1. Each ComC server $\{\mathcal{S}^\ell\}_{\ell=1}^L$ securely provisions secret keys (before deployment) and runs their commitment construction functions via a secure Trusted Execution Environment (TEE) as described in the system model to offer colluding resistance and commitment authentication.

Discussion: The malicious security properties (i.e., capability 4 in our threat model) of our schemes at the assisting servers rely on the security of the underlying TEE. This offers enhanced mitigation to collusion and malicious tampering attacks on the stored private keys on the ComC servers. This is realized with a low cost and without having any impact on the signer performance. Unlike some related works (e.g.,[32]), we do not require a TEE on the signer. We realized our TEE with Intel SGX's secure enclaves. However, our system could also be instantiated using other isolated execution environments (e.g., Sanctum[73]). It is crucial to recognize the limitations of relying on trusted execution environments. For example, Intel SGX encountered various side-channel attacks (e.g.,[74]). Generic techniques for protection against enclave side-channel attacks are also under study

in various works (e.g.,[56]), therefore they are complementary to ours. Finally, even if TEE on some ComC servers is breached, the EU-CMA property of our LRSHA scheme will remain as secure as our counterpart cloud-assisted signatures (e.g., [13,14]), which assume a semi-honest server model with $(L-1, L)$-privacy. However, our forward-secure scheme in this case can only achieve EU-CMA as LRSHA. We further note that $\mathcal{A}$ successfully launching side-channel attacks against multiple TEEs on distinct ComC servers simultaneously assumes an extremely strong adversary.

4 | PROPOSED SCHEMES

We first outline our design principles and how we address some critical challenges of constructing a highly lightweight signature with hardware-supported cloud assistance. We then describe our proposed schemes in detail.

High-Level Idea and Design Principles: Fiat-Shamir type EC-based signatures (e.g., Ed25519[37], FourQ [71]) are among the most efficient and compact digital signatures. Their main overhead is the generation of a commitment $R \leftarrow \alpha^r \mod p$ (EC scalar multiplication) from one-time randomness r. In Section 1.1, we captured the state-of-the-art lightweight signatures that aim to mitigate this overhead via various commitment management strategies.

In our design, we exploit the commitment separation method (e.g.,[75,76]), but with various advancements to address the challenges of previous approaches. In commitment separation, the value R_j in $H(M_j||R_j)$ is replaced with one-time randomness x_j per message as $H(M_j||x_j)$. This permits R_j to be stored at the verifier before signature generation, provided that x_j is disclosed only after signing. While this approach eliminates ExpOps due to commitment generation, it has significant limitations: (i) The verifier must store a commitment per message to be signed that incurs linear public key storage overhead[15,18] (i.e., one-time commitments become a part of the public key). (ii) This limits the total number of signatures to be computed and puts a burden on signers to replenish commitments when depleted.

Our strategy is to *completely* eliminate the burden of commitments from the signer, but do so and by achieving advanced security properties such as *forward-security* and *malicious server detection with collusion-resistance*, which are not available in previous counterparts simultaneously:

(i) Our design uses a distributed commitment strategy, in which value r is split into L different shares $\{r^\ell\}_{\ell=1}^{L}$ each provided to a TEE-supported ComC server $\{\mathcal{S}^\ell\}_{\ell=1}^{L}$ along with other keys to enable advanced features (to be detailed in algorithmic descriptions). This approach mitigates single-point failures and key compromise/escrow problems in centralized cloud-assisted designs (e.g.,[17,15]). (ii) Our design does not rely on BPV[49] (unlike[14]) or signature-tables (unlike[51]), but only uses simple arithmetic and PRF operations. This permits both computational and memory efficiency. If we accept equal table storage as our counterparts, then this further boosts our speed advantage. (iii) Some previous EC-based server-assisted signatures rely on semi-honest servers, which are prone to collusion and lack the ability to detect servers supplying false commitments. Instead, we wrap our ComC servers with a TEE that not only mitigates the collusion risk, but also forces the attacker to breach multiple TEE instances to extract keys or coerce an algorithmic deviation. This substantially increases the practical feasibility of active attacks targeting ComC servers. Moreover, our ComC servers authenticate each commitment separately, permitting verifiers to detect the server(s) injecting a false commitment. With TEE support, after detection, we can also use attestation to further mitigate post-compromise damages. (iv) We have a new forward-secure variant with an efficient key evolution strategy that avoids heavy nested certification trees (e.g., unlike[55,30]) and costly public key evolutions (e.g.,[22]). Thanks to this, our scheme offers more than 15 times faster signing with 24 times smaller signatures compared to the most efficient (generic) forward-secure EC-based counterpart (see Section 6).

We now present our schemes LRSHA and FLRSHA.

4.1 | Lightweight and Resilient Signature with Hardware Assistance (LRSHA)

We created Lightweight and Resilient Signature with Hardware Assistance (LRSHA), which is outlined in Figure 2 and detailed in Algorithm 1. We further elaborate steps in Algorithm 1 as follows.

The key generation algorithm LRSHA.Kg accepts the security parameter κ and the number of ComC servers L. It first generates EC-related parameters I and main private/public key pair (Step 1-2), and then a commitment certification private/public key pair (sk'^ℓ, PK'^ℓ) for each server $\mathcal{S}^\ell$ (Step 4). Subsequently, it generates private key components $a^\ell = \langle sk'^\ell, r^\ell \rangle$ to be provisioned to each secure enclave of server $\mathcal{S}^\ell$ (step 5-6). Finally, sk and state $St = (j \leftarrow 1)$ are provided to signer (Step 7-8).

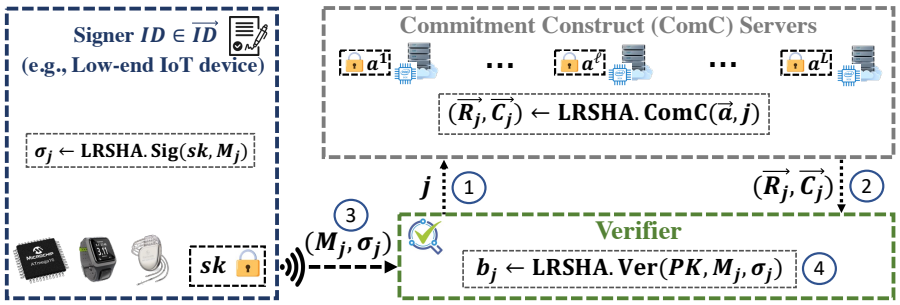

FIGURE 2 High-Level Overview of LRSHA.

Algorithm 1 Lightweight and Resilient Signature with Hardware Assistance (LRSHA)

$(sk, PK, \vec{a}, I) \leftarrow$ LRSHA.Kg$(1^\kappa, L)$:

Generate large primes q and p such that $q|(p-1)$. Select a generator α of the subgroup $\mathbb{G}$ of order q in $\mathbb{Z}_q^*$. Set $I \leftarrow (p, q, \alpha, St \leftarrow j = 1)$
$y \xleftarrow{\$} \mathbb{Z}_q^*$ and $Y \leftarrow \alpha^y \mod p$
for $\ell = 1, \ldots, L$ **do**
  $(sk'^\ell, PK'^\ell) \leftarrow$ SGN.Kg(1^κ)
  $r^\ell \xleftarrow{\$} \mathbb{Z}_q^*$
  $a^\ell \leftarrow \langle sk'^\ell, r^\ell \rangle$ is provisioned to enclave of server $\mathcal{S}^\ell$
$sk = \langle y, \vec{r} = \{r^\ell\}_{\ell=1}^L \rangle$
return $(sk, PK = (Y, \vec{PK'} = \{PK'^\ell\}_{\ell=1}^L), \vec{a} = \{a^\ell\}_{\ell=1}^L, I)$

$(\vec{R_j}, \vec{C_j}) \leftarrow$ LRSHA.ComC$(\vec{a}, j)$:

for $\ell = 1, \ldots, L$ **do**
  $R_j^\ell \leftarrow \alpha^{r_j^\ell} \mod p$, where $r_j^\ell \leftarrow \mathrm{PRF}_{r^\ell}(j) \mod q$
  $C_j^\ell \leftarrow$ SGN.Sig(sk'^ℓ, R_j^ℓ)
return $(\vec{R} = \{R_j^\ell\}_{\ell=1}^L, \vec{C} = \{C_j^\ell\}_{\ell=1}^L)$

$\sigma_j \leftarrow$ LRSHA.Sig(sk, M_j):

$r_j^{1,L} \leftarrow \sum_{\ell=1}^L r_j^\ell \mod q$, where $r_j^\ell \leftarrow \mathrm{PRF}_{r^\ell}(j) \mod q$
$e_j \leftarrow H(M_j \| x_j) \mod q$, where $x_j \leftarrow \mathrm{PRF}_y(j) \mod q$
$s_j \leftarrow r_j^{1,L} - e_j \cdot y \mod q$
Update $St \leftarrow j + 1$
return $\sigma_j \leftarrow \langle s_j, x_j, j \rangle$

$b_j \leftarrow$ LRSHA.Ver(PK, M_j, σ_j): Step 1-4 can be run offline.

$(\vec{R_j}, \vec{C_j}) \leftarrow$ LRSHA.ComC$(\vec{a}, j)$ ▷ Offline
for $\ell = 1, \ldots, L$ **do**
  if SGN.Ver$(PK'^\ell, R_j^\ell, C_j^\ell) = 0$ **then return** $b_j = 0$
$R_j^{1,L} \leftarrow \prod_{\ell=1}^L R_j^\ell \mod p$ ▷ Offline
$e_j \leftarrow H(M_j \| x_j) \mod q$
if $R_j^{1,L} = \alpha^{s_j} \cdot Y^{e_j} \mod p$, **return** $b_j = 1$, **else return** $b_j = 0$

In the signature generation algorithm LRSHA.Sig, given the state j, the signer first computes $r_j^{1,L}$ by aggregating values $\{r_j^\ell\}_{\ell=1}^L$ via PRF calls (Step 1). The one-time randomness x_j is used as the commitment (Step 2) instead of the public commitment R. Step 3 is as in Schnorr's signature, followed by a state update. Overall, our signing avoids any ExpOp, costly pre-computed tables (e.g., BPV or signature tables), or secure hardware requirements.

LRSHA.Ver is a cloud-assisted verification algorithm, and therefore calls LRSHA.ComC to retrieve L partial commitment values $\{R_j^\ell\}_{\ell=1}^L$ and their certificates from ComC servers (Step 1). In LRSHA.ComC, each server $\mathcal{S}^\ell$ first derives $\{R_j^\ell\}_{\ell=1}^L$ from their private keys $\vec{a} = \{a^\ell\}_{\ell=1}^L$ (step 2), puts a signature to certify them as $\{C_j^\ell\}_{\ell=1}^L$ (step 3) and returns these values to the verifier. The rest of LRSHA.Ver is similar to EC-Schnorr but with randomness x_j instead of commitment R_j in hash (steps 5-6). Note that the verifier can retrieve commitments and verify certificates offline (and even in batch) before message verification occurs. Hence, the overall online message verification overhead is identical to the EC-Schnorr signature. Moreover, LRSHA.Ver does

not require any pre-computed table, lets the verifier detect false commitments, and offers distributed security for assisting servers with enhanced collusion resiliency via TEE support in `LRSHA.ComC`.

Discussion: To address server downtime and TEE compromise issues, the signer can efficiently adjust the number of active servers $\{\mathcal{S}_\ell\}_{\ell=1}^{L}$ and *revoke* compromised ComC servers by updating the active servers' identities from a private key lookup table $\{r^\ell\}_{\ell=1}^{L_{max}}$, where L_{max} denotes the maximum number of supported ComC servers. The additional storage overhead is $(L_{max} - L) \cdot \kappa$, and the signer includes L ComC identities $\{ID^\ell\}_{\ell=1}^{L}$ (e.g., MAC address with each is of size 4 bytes) in the signature. Moreover, to support multi-user settings, the ComC servers can generate commitments for multiple signers by deriving each signer's private key elements $\{r^\ell\}_{\ell=1}^{L}$ from master private keys. This can be achieved, for example, via `PRF` calls as $r^{\ell,i} \leftarrow \texttt{PRF}(r_\ell \| ID_i)$ where ID_i is the signer identity (e.g., MAC address). The only extra overhead at the signer is adding its identity to the signature.

4.2 | Forward-secure Lightweight and Resilient Signature with Hardware Assistance (FLRSHA)

We now present our Forward-secure `LRSHA` (`FLRSHA`) as detailed in Algorithm 2 with an overview in Figure 3. We developed a key evolution mechanism for the signer and ComC servers that enables a highly lightweight yet compromise-resilient digital signature. Our introduction of distributed TEEs provided significant performance and security benefits, making `FLRSHA` the most efficient forward-secure alternative for low-end embedded devices (see in Section 6). Below, we outline `FLRSHA` algorithms by focusing on their differences with `LRSHA`.

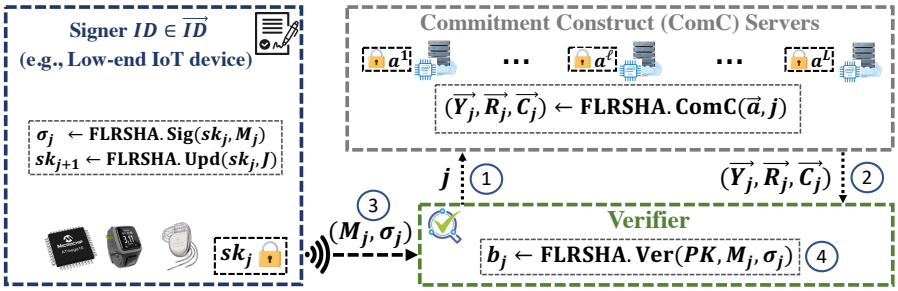

FIGURE 3 The high-level overview of `FLRSHA`.

The key generation `FLRSHA.Kg` works as in `LRSHA.Kg` but with the following differences: (i) It takes the maximum number of signatures J as an additional parameter, (ii) It generates a distinct forward-secure signature private/public key pair (step 3), (iii) It generates a private key tuple (y_1^ℓ, r_1^ℓ) (Step 4), for each $\{\mathcal{S}^\ell\}_{\ell=1}^{L}$. Unlike `LRSHA`, ComC will produce one-time public key pairs from those and certify them with a forward-secure signature.

In `FLRSHA.Sig`, unlike `LRSHA`, the signer calls a key update function `FLRSHA.Upd` (step 4), which evolves private key pairs $\{(y_j^\ell, r_j^\ell)\}_{\ell=1}^{L}$ by hashing and then deleting the previous key given $1 \leq j < J$ (steps 2-3). `FLRSHA.Upd` ensures a forward-secure private key pair is maintained per ComC server up to state J. `FLRSHA.Sig` then computes two aggregate private key components (step 1) instead of one, but uses this key pair as in `LRSHA.Sig` to compute the signature (steps 2-3). The cost of `FLRSHA.Sig` is mainly a few PRF calls and modular additions, therefore is highly efficient, as shown in Section 6.

`FLRSHA.Ver` works like `LRSHA.Ver` but with differences in aggregate keys and `ComC`: (i) `FLRSHA.ComC` provides a pair of one-time public key set and forward-secure signatures $(\vec{Y_j}, \vec{R_j}, \vec{C_j})$ for each $\{\mathcal{S}^\ell\}_{\ell=1}^{L}$, which can be retrieved and authenticated offline (before actual signature verification, only up to J) (steps 1-3). (ii) The verifier computes the aggregate pair $(Y_j^{1,L}, R_j^{1,L})$ (as opposed to only aggregate commitment in `LRSHA.Ver`), and the rest is as in `LRSHA.Ver`. Hence, the online overhead of `FLRSHA.Ver` is almost as efficient as that of `LRSHA.Ver` with only a small-constant number of (negligible) scalar addition cost differences.

• *Enhancing computational efficiency on ComC servers*: In Algorithm 2, ComC servers run a hash chain of length $0 \leq j < J$ on their private key components $(H^{(j)}(y_j^\ell), H^{(j)}(r_j^\ell))$ to generate public keys and commitments. To avoid the cost of hash recursion for long chains (e.g., $j \approx 2^{20}$), one can use a precomputed table of the private key components with interleaved indices at the ComC servers. This offers a computation-storage trade-off that ComC servers can decide. Note that the private keys are stored in a secure enclave. Given that modern enclaves offer large protected memory of up to 512 GB, the overhead of pre-computed tables is likely negligible. For instance, the total memory overhead of $(J = 2^{20})$ public commitments is equal to

Algorithm 2 Forward-secure Lightweight and Resilient Signature with Hardware Assistance (FLRSHA)

$(sk_1, PK, \vec{a}, I) \leftarrow$ FLRSHA.Kg$(1^\kappa, J, L)$:
Generate primes q and $p > q$ such that $q|(p-1)$. Select a generator α of the subgroup G of order q in $\mathbb{Z}_p^*$. Set $I \leftarrow \langle p, q, \alpha \rangle$ as system parameter.
for $\ell = 1, \ldots, L$ **do**
  $(sk'^\ell, PK'^\ell) \leftarrow$ FSGN.Kg(1^κ)
  $y_1^\ell \xleftarrow{\$} \mathbb{Z}_q^*$, and $r_1^\ell \xleftarrow{\$} \mathbb{Z}_q^*$
  $a^\ell \leftarrow \langle y_1^\ell, r_1^\ell, sk'^\ell \rangle$ is provisioned to TEE of server $\mathcal{S}^\ell$
$sk_1 \leftarrow \langle \vec{y_1} = \{y_1^\ell\}_{\ell=1}^L, \vec{r_1} = \{r_1^\ell\}_{\ell=1}^L \rangle$. The signer's initial state is $St \leftarrow j = 1$
return $(sk_1, PK = \{PK'^\ell\}_{\ell=1}^L, \vec{a} = \{a^\ell\}_{\ell=1}^L, I)$

$(\vec{Y_j}, \vec{R_j}, \vec{C_j}) \leftarrow$ FLRSHA.ComC$(\vec{a}, j)$: Each $S^1, \ldots, S^L$ executes in their independent TEE in isolation. It can be done in batch offline, or on demand online.
for $\ell = 1, \ldots, L$ **do**
  $Y_j^\ell \leftarrow \alpha^{y_j^\ell} \mod p$, where $y_j^\ell \leftarrow H^{(j-1)}(y_1^\ell) \mod q$
  $R_j^\ell \leftarrow \alpha^{r_j^\ell} \mod p$, where $r_j^\ell \leftarrow H^{(j-1)}(r_1^\ell) \mod q$
  $C_j^\ell \leftarrow$ FSGN.Sig$(sk'^\ell, Y_j^\ell \| R_j^\ell)$
return $(\vec{Y_j} = \{Y_j^\ell\}_{\ell=1}^L, \vec{R_j} = \{R_j^\ell\}_{\ell=1}^L, \vec{C_j} = \{C_j^\ell\}_{\ell=1}^L)$

$sk_{j+1} \leftarrow$ FLRSHA.Upd(sk_j, J): If $j \geq J$ then *abort*.
for $\ell = 1, \ldots, L$ **do**
  $y_{j+1}^\ell \leftarrow H(y_j^\ell) \mod q$
  $r_{j+1}^\ell \leftarrow H(r_j^\ell) \mod q$
Set $\vec{y}_{j+1} = \{y_{j+1}^\ell\}_{\ell=1}^L$, $\vec{r}_{j+1} = \{r_{j+1}^\ell\}_{\ell=1}^L$, and $St \leftarrow j + 1$
return $sk_{j+1} \leftarrow \langle \vec{y}_{j+1}, \vec{r}_{j+1} \rangle$

$\sigma_j \leftarrow$ FLRSHA.Sig(sk_j, M_j): If $j > J$ then *abort*.
$y_j^{1,L} \leftarrow \sum_{\ell=1}^L y_j^\ell \mod q$, and $r_j^{1,L} \leftarrow \sum_{\ell=1}^L r_j^\ell \mod q$
$e_j \leftarrow H(M_j \| x_j) \mod q$, where $x_j \leftarrow \mathrm{PRF}_{y_j^{1,L}}(j)$
$s_j \leftarrow r_j^{1,L} - e_j \cdot y_j^{1,L} \mod q$
$sk_{j+1} \leftarrow$ FLRSHA.Upd(sk_j, J)
return $\sigma_j \leftarrow \langle s_j, x_j, j \rangle$

$b_j \leftarrow$ FLRSHA.Ver(PK, M_j, σ_j): If $j > J$ then *abort*. Note that steps 1–4 can be run offline.
$(\vec{Y_j}, \vec{R_j}, \vec{C_j}) \leftarrow$ FLRSHA.ComC$(\vec{a}, j)$ ▷ Offline
for $j = 1, \ldots, L$ **do**
  if FSGN.Ver$(PK'^\ell, Y_j^\ell \| R_j^\ell, C_j^\ell) = 0$ **then return** $b_j = 0$
$Y_j^{1,L} \leftarrow \prod_{\ell=1}^L Y_j^\ell \mod p$, $R_j^{1,L} \leftarrow \prod_{\ell=1}^L R_j^\ell \mod p$ ▷ Offline
$e_j \leftarrow H(M_j \| x_j) \mod q$
if $R_j^{1,L} = \alpha^{s_j} \cdot (Y_j^{1,L})^{e_j} \mod p$ **then return** $b_j = 1$ **else** $b_j = 0$

only 32 MB. Alternatively, a Seekable Sequential Key Generator (SSKG)[77] can be integrated into FLRSHA as an efficient key update mechanism. This integration requires an average of L additional hash invocations per key update by the signer, while significantly reducing the storage and computational overhead to $\mathcal{O}(\log_d J)$ hash values and operations at each ComC server during FLRSHA.ComC, where d denotes the arity of the tree data structure in the underlying SSKG.

5 | SECURITY ANALYSIS

We prove that LRSHA and FLRSHA are HD-EU-CMA and FHD-EU-CMA secure, respectively (in random oracle model). We omit terms negligible to κ (unless expressed for clarity).

Theorem 1. *If a PPT adversary $\mathcal{A}$ can break the HD-EU-CMA-secure LRSHA in time t and after q_s signature and commitment queries to LRSHA.$Sig_{sk}(.)$ and LRSHA.$ComC_{\vec{a}}(.)$ oracles, and q_H queries to RO(.), then one can build a polynomial-time algorithm $\mathcal{F}$ that breaks the DLP in time t' (by Definition 7). The probability that any $\{S^l\}_{l=1}^L$ injects a false commitment without being detected is $Adv_{SGN}^{EU\text{-}CMA}(t'', q_H, L \cdot q_s)$, under the Assumption 1, in time t''.*

$$Adv_{LRSHA}^{HD\text{-}EU\text{-}CMA}(t, q_H, q_s) \leq Adv_{\mathbb{G},\alpha}^{DL}(t'),\ t' = \mathcal{O}(t) + L \cdot \mathcal{O}(q_s \cdot \kappa^3)$$

Proof: Let $\mathcal{A}$ be a LRSHA attacker. We construct a *DL*-attacker that uses $\mathcal{A}$ as a subroutine. We set ($y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow \alpha^y \mod p$) as in Definition 3 and $\vec{PK'} = \{(sk'^{\ell}, PK'^{\ell}) \leftarrow \texttt{SGN.Kg}(1^{\kappa})\}_{\ell=1}^{L}$ (as in LRSHA.Kg), where the rest of the key generation will be simulated in the *Setup phase* . $\mathcal{F}$ is run by Definition 7 (i.e., HD−EU−CMA) as follows:

Algorithm $\mathcal{F}(PK)$:

Setup: $\mathcal{F}$ maintains $\mathcal{LH}$,$\mathcal{LM}$, and $\mathcal{LR}$ to keep track of the query results in during the experiments. $\mathcal{LH}$ is a public hash list in form of pairs $\{M_i : h_i\}$, where (M_i, h_i) represents i^{th} data item queried to *RO*(.) and its corresponding answer, respectively. $\mathcal{LM}$ is a public message list that represents the messages queried by $\mathcal{A}$ to $\texttt{LRSHA.Sig}_{sk}(.)$ oracle. $\mathcal{LR}$ is a private list containing the randomly generated variables.

$\mathcal{F}$ initializes simulated public keys and *RO*(.) as follows:

- *Key Setup*: $\mathcal{F}$ injects challenge public key as $PK = (Y, \vec{PK'})$, and sets the parameters $I \leftarrow (p, q, \alpha, L)$. $\mathcal{F}$ generates $x_0 \xleftarrow{\$} \mathbb{Z}_q^*$, adds it to $\mathcal{LR}$, and sets the state as $St \leftarrow j = 1$.
- *RO*(.) *Setup*: $\mathcal{F}$ uses a function *H-Sim* that acts as a random oracle *RO*(.). If $\exists M : \mathcal{LH}[M] = h$, then *H-Sim* returns h. Otherwise, it returns $h \xleftarrow{\$} \mathbb{Z}_q^*$ and save it as $\mathcal{LH}[M] \leftarrow h$.

• *Execute* $(M^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.),\ \texttt{LRSHA.Sig}_{sk}(.),\ \texttt{LRSHA.ComC}_{\vec{a}}(.)}(PK)$:
$\mathcal{F}$ handles the queries of $\mathcal{A}$ as follows:

- Queries of $\mathcal{A}$: $\mathcal{A}$ can query *RO*(.) and $\texttt{LRSHA.Sig}_{sk}(.)$ on any message M of its choice up to q_H and q_s times, respectively. $\mathcal{A}$ can query $\texttt{LRSHA.ComC}_{\vec{a}}(.)$ oracle on the state j as input, and it returns the corresponding commitments $(\vec{R_j}, \vec{C_j})$ as the output. $\mathcal{F}$ handles $\mathcal{A}$'s queries as follows:
  1) *RO*(.) *queries*: $\mathcal{A}$ queries *RO*(.) on a message M. $\mathcal{F}$ calls $h \leftarrow$ *H-Sim*$(M, \mathcal{LH})$ and returns h to $\mathcal{A}$.
  2) $\texttt{LRSHA.Sig}_{sk}(.)$ *queries*: Insert M into $\mathcal{LM}$, and execute:
     i) $x_j \leftarrow$ *H-Sim*$(x_0 \| j, \mathcal{LH})$, where $(x_0 \leftarrow \mathcal{LR})$. If $M_j \| x_j \notin \mathcal{LH}$, then $\mathcal{F}$ calls *H-Sim*$(M_j \| x_j, \mathcal{LH})$, otherwise $\mathcal{F}$ *aborts*.
     ii) Retrieve s_j from $\mathcal{LR}$ if $s_j \in \mathcal{LR}$. Otherwise, $\mathcal{F}$ sets $s_j \xleftarrow{\$} \mathbb{Z}_q^*$, $e_j \xleftarrow{\$} \mathbb{Z}_q^*$, and $R_j^{1,L} \leftarrow \alpha^{s_j} \cdot Y^{e_j} \mod p$ and adds each of s_j, e_j, and $R_j^{1,L}$ to $\mathcal{LR}$.
     iii) $St = (j \leftarrow j + 1)$ and return $\sigma_j \leftarrow (s_j, x_j, j)$.
  3) *Handle* $\texttt{LRSHA.ComC}_{\vec{a}}(.)$: $\mathcal{A}$ can query $\texttt{LRSHA.ComC}_{\vec{a}}(.)$ on any index $1 \leq j \leq q_s$ of her choice. $\mathcal{F}$ handles these queries as follows: If $R_j^{\ell} \notin \mathcal{LR}$, then $\mathcal{F}$ generates $R_j^{\ell} \xleftarrow{\$} \mathbb{Z}_p^*$ and adds it to $\mathcal{LR}$, else fetch it from $\mathcal{LR}$ for $\ell = 1, \ldots, L-1$. $\mathcal{F}$ also checks if $R_j^{1,L} \in \mathcal{LR}$, then it retrieves. Otherwise, it generates $e_j \xleftarrow{\$} \mathbb{Z}_q^*$ and $s_j \xleftarrow{\$} \mathbb{Z}_q^*$, and sets $R_j^{1,L} \leftarrow \alpha^{s_j} \cdot Y^{e_j} \mod p$, $R_j^{L} \leftarrow \prod_{\ell_1=1}^{L-1} (R_j^{\ell_1})^{-1} \cdot R_j^{1,L} \mod p$, and add these values to $\mathcal{LR}$. Finally, $\mathcal{F}$ computes $\{C_j^{\ell} \leftarrow \texttt{SGN.Sig}_{\vec{sk'}}(R_j^{\ell})\}_{\ell=1}^{L}$ and returns $(\vec{R_j} \leftarrow \{R_j^{\ell}\}_{\ell=1}^{L}, \vec{C_j} = \{C_j^{\ell}\}_{\ell=1}^{L})$ to $\mathcal{A}$.
- Forgery of $\mathcal{A}$: Finally, $\mathcal{A}$ outputs a forgery for PK as (M^*, σ^*), where $\sigma^* = (s^*, x^*, j)$. By Definition 7, $\mathcal{A}$ wins the HD−EU−CMA-experiment for LRSHA if $\texttt{LRSHA.Ver}(PK, M^*, \sigma^*) = 1$ and $M^* \notin \mathcal{LM}$.
- Forgery of $\mathcal{F}$: If $\mathcal{A}$ fails, $\mathcal{F}$ also fails and *aborts*. Otherwise, given an LRSHA forgery $(M^*, \sigma^* \leftarrow (s^*, x^*, j))$ on PK: *(i)* $\mathcal{F}$ checks if $M^* \| x^* \notin \mathcal{LH}$ (i.e., $\mathcal{A}$ does not query *RO*(.)), then $\mathcal{F}$ *aborts*. *(ii)* $\mathcal{F}$ checks if $R_j^{1,L} \notin \mathcal{LR}$ (i.e., $\mathcal{F}$ did not query $\texttt{LRSHA.ComC}_{\vec{a}}(.)$), then $\mathcal{F}$ *aborts*. Otherwise, $\mathcal{F}$ continues as follows: Given $R_j^{1,L}$ computed by $\mathcal{A}$, the equation $R_j^{1,L} = Y^{e_j} \cdot \alpha^{s_j} \mod p$ holds, where e_j and s_j are derived from $\mathcal{LR}$. $\texttt{LRSHA.Ver}(PK, M_j^*, \sigma_j^*) = 1$ also holds, and so $R_j^{1,L} \equiv Y^{e_j^*} \cdot \alpha^{s_j^*} \mod p$ holds and $e_j^* \leftarrow$ *H-Sim*$(M_j^* \| x_j^*) \mod p$. Therefore, $\mathcal{F}$ can extract $y' = y$ by solving the below modular linear equations, where $Y = \alpha^{y'} \mod p$.

$$R_j^{1,L} \equiv Y^{e_j^*} \cdot \alpha^{s_j^*} \bmod p,\ R_j^{1,L} \equiv Y^{e_j} \cdot \alpha^{s_j} \bmod p,$$
$$r_j^{1,L} \equiv y' \cdot e_j^* + s_j^* \bmod q,\ r_j^{1,L} \equiv y' \cdot e_j + s_j \bmod q,$$

$Y = \alpha^{y'} \mod p$ holds as $\mathcal{A}$'s forgery is valid and non-trivial on PK. By Definition 3, $\mathcal{F}$ wins the *DL* experiment.

Execution Time Analysis: The runtime of $\mathcal{F}$ is that of $\mathcal{A}$ plus the time to respond to the queries of *RO*(.), $\texttt{LRSHA.Sig}_{sk}(.)$, and $\texttt{LRSHA.ComC}_{\vec{a}}(.)$. The dominating overhead of the simulations is modular exponentiation, whose cost is denoted as $\mathcal{O}(\kappa^3)$.

Each $\texttt{LRSHA.Sig}_{sk}(.)$ and $\texttt{LRSHA.ComC}_{\vec{a}}(.)$ query invokes approximately L modular exponentiation operations, making asymptotically dominant cost $L \cdot \mathcal{O}(q_s \cdot \kappa^3)$. Therefore, the approximate running time of $\mathcal{F}$ is $t' = \mathcal{O}(t) + L \cdot \mathcal{O}(q_s \cdot \kappa^3)$.

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

- $\overline{E1}$: $\mathcal{F}$ does not abort during the query phase.
- $E2$: $\mathcal{A}$ wins the HD−EU−CMA experiment for LRSHA.
- $\overline{E3}$: $\mathcal{F}$does not abort after $\mathcal{A}$'s forgery.
- *Win*: $\mathcal{F}$wins the *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 $M_j \| x_j \in \mathcal{LH}$ holds, *before* $\mathcal{F}$ inserts $M_j \| x_j$ into $\mathcal{LH}$. This occurs if $\mathcal{A}$ guesses x_j (before it is released) and then queries $M_j \| x_j$ to $RO(.)$ *before* it queries $\texttt{HDSGN.Sig}_{sk}(.)$. 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 $\mathcal{A}$'s simulated view is *indistinguishable* from $\mathcal{A}$'s real view.

Therefore, $Pr[E2|\overline{E1}] \approx Adv^{\text{HD-EU-CMA}}_{\text{LRSHA}}(t, q_H, q_s)$.

• *The probability that event $\overline{E3}$ occurs*: $\mathcal{F}$ does not abort if the following conditions are satisfied: (i) $\mathcal{A}$ wins the HD−EU−CMA experiment for LRSHA 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 (see event $\overline{E1}$). (ii) $\mathcal{A}$ wins the HD−EU−CMA experiment for LRSHA by querying $\texttt{LRSHA.ComC}_{\vec{a}}(.)$. The probability that $\mathcal{A}$ wins without querying $\texttt{LRSHA.ComC}_{\vec{a}}(.)$ is equivalent to forging SGN, which is equal to $Adv^{\text{EU-CMA}}_{\text{SGN}}(t'', q_H, q_s)$. (iii) After $\mathcal{F}$ extracts $y' = y$ by solving modular linear equations, the probability that $Y \not\equiv \alpha^{y'} \bmod p$ is negligible in terms κ, since $PK = (Y, \{PK'^{\ell}\}_{\ell=1}^{L})$ and $\texttt{LRSHA.Ver}(PK, M^*, \sigma^*) = 1$. Hence, $Pr[\overline{E3}|\overline{E1} \wedge E2] \approx Adv^{\text{HD-EU-CMA}}_{\text{LRSHA}}(t, q_H, q_s)$.

Indistinguishability Argument: The real-view of $\overrightarrow{A}_{real}$ is comprised of (PK, I), the answers of $\texttt{LRSHA.Sig}_{sk}(.)$ and $\texttt{LRSHA.ComC}_{\vec{a}}(.)$ (recorded in $\mathcal{LM}$ and $\mathcal{LR}$ by $\mathcal{F}$), and the answer of $RO(.)$ (recorded in $\mathcal{LH}$ by $\mathcal{F}$). All these values are generated by LRSHA algorithms, where $sk = (y, \vec{r})$ serves as 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, (s_j, e_j, x_0) (and so as x_j) are randomly drawn from $\mathbb{Z}_q^*$. This dictates the selection $\{R_j^{1,L}\}$ as random via the $\texttt{LRSHA.Sig}_{sk}(.)$ and $\texttt{LRSHA.ComC}_{\vec{a}}(.)$ oracles, respectively. That is, for each state $St = j$, the partial public commitments $\{R_j^{\ell}\}_{\ell=1}^{L-1}$ are randomly selected from $\mathbb{Z}_q^*$ while the last partial public commitment is equal to $R_j^L \leftarrow \prod_{\ell=1}^{L-1} (R_j^{\ell})^{-1} \cdot \alpha^{s_j} \cdot Y^{e_j} \mod p$. The aggregate commitment $R_j^{1,L} \leftarrow \prod_{\ell=1}^{L} R_j^{\ell} \mod p = (\prod_{\ell=1}^{L-1} R_j^{\ell}) \cdot (\prod_{\ell=1}^{L-1} (R_j^{\ell})^{-1}) \cdot \alpha^{s_j} \cdot Y^{e_j} = \alpha^{s_j} \cdot Y^{e_j} \mod p$. Thus, the correctness of aggregate commitments holds as in the real view. The joint probability distribution of these values is randomly and uniformly distributed and is identical to original signatures and hash outputs in $\overrightarrow{A}_{real}$, since the cryptographic hash function H is modeled as $RO(.)$ via *H-Sim*. ■

Theorem 2. *If a PPT adversary $\mathcal{A}$ can break the FHD−EU−CMA-secure FLRSHAin time t and after q_s signature and commitment queries to both $\texttt{FLRSHA.Sig}_{sk_j}(.)$ and $\texttt{FLRSHA.ComC}_{\vec{a}}(.)$ oracles, q_H queries to RO(.) and one query to Break−In(.) oracle, then one can build a polynomial-time algorithm $\mathcal{F}$ that breaks DLP in time t' (by Definition 9). The probability that $\{S^{\ell}\}_{\ell=1}^{L}$ injects a false commitment without being detected is $Adv^{EU-CMA}_{FSGN}(t'', q_H, L \cdot q_s)$, under the Assumption 1, in time t''.*

$$Adv^{FHD-EU-CMA}_{FLRSHA}(t, q_H, q_s, 1) \leq Adv^{DL}_{\mathbb{G},\alpha}(t'), t' = \mathcal{O}(t) + L \cdot \mathcal{O}(J \cdot \kappa^3)$$

Proof: Let $\mathcal{A}$ be a FLRSHA attacker and $(y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow \alpha^y \mod p)$ be a DLP challenge as in Definition 3. We set the certification keys $(\{sk'^{\ell}, PK'^{\ell}\}_{\ell=1}^{L}$ via $\texttt{FSGN.Kg}$. We then run the simulator $\mathcal{F}$ by Definition 5:

• *Setup:* $\mathcal{F}$ manages the lists $\mathcal{LM}$, $\mathcal{LH}$, and $\mathcal{LR}$, and handles $RO(.)$ queries via *H-Sim* as in Theorem 1. Per Definition 8, $\mathcal{F}$ selects the maximum number of signatures as J, and then selects an index $w \xleftarrow{\$} [1, J]$ hoping that $\mathcal{A}$ will output his forgery on. $\mathcal{F}$ continue as follows:

– *sk Simulation*: Set $sk_1 = \langle \vec{y}_1, \vec{r}_1 \rangle$ as in $\texttt{FLRSHA.Kg}$.

1: $a^{\ell} \leftarrow \langle y_1^{\ell}, r_1^{\ell}, sk'^{\ell} \rangle$, $\forall \ell = 1, \ldots, L$, and $\vec{a} = \{a^{\ell}\}_{\ell=1}^{L}$

2: $sk_{j+1} \leftarrow \texttt{FLRSHA.Upd}(sk_j), \forall j \in [1, w-2]$, where H in $\texttt{FLRSHA.Upd}$ is simulated via *H-Sim*.

3: $sk_{w+1} = \langle \vec{y}_{w+1}, \vec{r}_{w+1} \rangle$ as in FLRSHA.Kg.
4: $sk_{j+1} \leftarrow$ FLRSHA.Upd$(sk_j), \forall j \in [w+1, J-1]$.
5: Add sk_j to $\mathcal{LR}$, $\forall j \in [1, w-1] \cup [w+1, J]$ and $\vec{a}$ to $\mathcal{LR}$.

– *PK and ComC Simulation*:

1: Retreive $\{sk_j\}_{j\in[1,w-1]\cup[w+1,J]}$ and $\vec{a}$ from $\mathcal{LR}$.
2: **for** $j \in [1, w-1] \cup [w+1, J]$ **do**
3: $\vec{R}_j \leftarrow \{R_j^\ell \leftarrow \alpha^{r_j^\ell} \mod q\}_{\ell=1}^{L}$ and $\vec{Y}_j \leftarrow \{Y_j^\ell \leftarrow \alpha^{y_j^\ell} \mod q\}_{\ell=1}^{L}$. $\mathcal{F}$ adds $\vec{Y}_j$ and $\vec{R}_j$ to $\mathcal{LR}$.
4: $s_w \xleftarrow{\$} \mathbb{Z}_q^*$ and $e_w \xleftarrow{\$} \mathbb{Z}_q^*$. $\mathcal{F}$ adds s_w and e_w to $\mathcal{LR}$.
5: $Y_w^\ell \xleftarrow{\$} \mathbb{Z}_p^*$ and $R_w^\ell \xleftarrow{\$} \mathbb{Z}_q^*$, $\forall \ell = 1, \ldots L-1$.
6: $Y_w^L \leftarrow Y \cdot \prod_{\ell=1}^{L-1} (Y_w^\ell)^{-1} \mod p$
7: $R_w^L \leftarrow Y^{e_w} \cdot \alpha^{s_w} \cdot \prod_{\ell=1}^{L-1} (R_w^\ell)^{-1} \mod p$
8: Add $\vec{Y}_w \leftarrow \{Y_w^\ell\}_{\ell=1}^{L}$ and $\vec{R}_w \leftarrow \{R_w^\ell\}_{\ell=1}^{L}$ to $\mathcal{LR}$.

• *Query Phase:* $\mathcal{F}$ handles $RO(.)$ and Break-In(.) queries as in Theorem 1 and Definition 9, respectively. The rest of $\mathcal{A}$'s queries are as follows:

– FLRSHA.Sig$_{sk_j}$(.): If $j \neq w$, then it retrieves sk_j from $\mathcal{LR}$ and computes σ_j as in FLRSHA.Sig. Otherwise, it retrieves s_w from $\mathcal{LR}$ and returns $\sigma_j = (s_w, x_w \xleftarrow{\$} \{0,1\}^\kappa, w)$.

– FLRSHA.ComC$_{\vec{a}}$(.): $\mathcal{F}$ retrieves $(\vec{Y}_j, \vec{R}_j)$ from $\mathcal{LR}$ and computes $\vec{C}_j = \{ C_j^\ell \leftarrow$ FSGN.Sig$_{s\vec{k}'}(Y_j^\ell, R_j^\ell) \}_{\ell=1}^{L}$. Finally, $\mathcal{F}$ returns $(\vec{Y}_j, \vec{R}_j, \vec{C}_j)$ to $\mathcal{A}$.

• *Forgery and Extraction:* $\mathcal{A}$ outputs a forgery on PK as (M^*, σ^*), where $\sigma^* = (s^*, x^*, j)$. By Definition 9, $\mathcal{A}$ wins if FLRSHA.Ver$(PK, M^*, \sigma^*) = 1$, and $M^* \notin \mathcal{LM}$. $\mathcal{F}$ wins if (i) $\mathcal{A}$ wins, (ii) $\mathcal{A}$ produces a forgery on $j = w$ by querying FLRSHA.ComC$_{\vec{a}}$(.) (i.e., $(\vec{Y}_w, \vec{R}_w) \in \mathcal{LR}$) and $RO(.)$ (i.e., $(M^* \| x^*) \in \mathcal{LH}$). If these conditions hold, then $\mathcal{F}$ extracts y' as in Theorem 1 extraction phase.

• *Success Probability and Execution Time Analysis:* The analysis is similar to Theorem 1 except the forgery index must be on $j = w$. That is, the probability that $\mathcal{A}$ wins FHD-EU-CMA experiment against FLRSHA is equal to $Adv_{\text{FLRSHA}}^{\text{FHD-EU-CMA}}(t, q_H, q_s, 1)$. $\mathcal{F}$ wins the DLP experiment if $\mathcal{A}$ outputs his forgery on $j = w$. Since w is randomly drawn from $[1, J]$, the probability that $\mathcal{A}$ returns his forgery on $j = w$ is $1/J$. The probability that $\mathcal{A}$ wins the experiment without querying $RO(.)$ and LRSHA.ComC$_{\vec{a}}$(.) are $1/2^\kappa$ and $Adv_{\text{FSGN}}^{\text{EU-CMA}}(t'', q_H, L \cdot q_s)$, respectively. The execution time is asymptotically similar to that of Theorem 1, where $q_s = J$:

$$Adv_{\text{FLRSHA}}^{\text{FHD-EU-CMA}}(t, q_H, q_s, 1) \leq Adv_{\mathbb{G},\alpha}^{DL}(t'), t' = \mathcal{O}(t) + 2L \cdot J \cdot \mathcal{O}(\kappa^3)$$

• *Indistinguishability Argument:* $\mathcal{A}$'s real $\mathcal{A}_R$ and simulated $\mathcal{A}_S$ views are indistinguishable. The argument is as in Theorem 1, with the following differences: (i) In $\mathcal{A}_S$, the simulator uses private keys that are randomly generated except during $j = w$ where he injects the challenge Y. These random variables are identical to $\mathcal{A}_R$ since H is a random oracle. (ii) FSGN is used to sign commitments in the transcripts instead of SGN. ■

6 | PERFORMANCE ANALYSIS

We present a comprehensive performance evaluation of our schemes with a detailed comparison with their counterparts.

6.1 | Evaluation Metrics and Experimental Setup

Evaluation Metrics: We compare our proposed schemes and their counterparts based on: (1) signing computational overhead, (2) signature size, (3) private key size (including pre-computed tables), (4) verification overhead, (5) public key size, (6) performance in pre-computed, offline/online settings, (7) forward-security, (8) collusion resiliency / false input detection, (9) implementation features (e.g., online sampling, code base simplicity), (10) impact on battery life.

Selection Rationale of Counterparts: We follow our related work analysis in Section 1.1 as the guideline. Given that it is not possible to compare our schemes with every single digital signature, we focus on the most relevant categories to our work, especially the ones having an open-source implementation on low-end devices: (i) ECDSA [36], BLS [40], RSA [10] to cover the most prominent signatures, serving as a building block for others. (ii) BLISS as the lattice-based (due to its ability to run on an 8-bit MCU), SPHINCS+ [78] (hash-based on commodity hardware), and XMSS^{MT} [55] as the forward-secure standard. (iii) Alternative lightweight signatures with TEE and/or cloud assistance (e.g., [17,14]). (iv) We compare our forward-secure FLRSHA with MMM transformed versions of the most efficient signature schemes since it is proven to be an asymptotically optimal generic forward-secure transformation. We also compared FLRSHA with the most recent hardware-assisted counterparts [17]. (v) We also provided a comprehensive comparison when various signer optimizations are considered, especially with pre-computation methods for low-end devices (e.g., SCRA [51], BPV-variants, offline-online, etc.). We also included the pre-computed version ESEM$_2$ [14] as an ECDLP-based cloud-assisted digital signature with distributed verification.

Parameter Selection: We selected the security parameter as $\kappa = 128$ and ASCON-Hasha [79] as our cryptographic hash function H. We used the Curve25519 [37] (as NIST's FIPS 186-5 standard, 256-bit public keys) for our signature schemes. For BLS [40], we selected the curve BLS12-381, having an embedding degree equal to 12 and a 381 bit length. We selected XMSS^{MT} which allows $J = 2^{20}$ messages to be signed. We also set an equal signing capability $J = 2^{20}$ in our schemes, LRSHA and FLRSHA. We selected Ed25519 as our standard signature scheme SGN that serves to certify the commitments of LRSHA, while we opted for Ed25519 with optimal generic MMM [30] as a forward-secure FSGN for the ComC certification of FLRSHA. We discuss the parameters and specifics of other counterparts in Table 2.

Hardware/Software Configurations: We benchmark our proposed schemes on a high-performance workstation and highly constrained 8-bit MCU as follows: (i) We used a desktop with an Intel i9-9900K@3.6 GHz processor and 64 GB of RAM. We also used ASCON[†], OpenSSL[‡] and Intel SGX SSL[§] open-source libraries. (ii) We fully implemented LRSHA schemes on an 8-bit AVR ATmega2560 Micro-Controller Unit (MCU) at the signer side. This MCU is an 8-bit ATmega2560, having 256KB flash memory, 8KB SRAM, and 4KB EEPROM operating at a clock frequency of 16MHz. We used μNaCl[¶] open-source software library to implement the finite-field arithmetic operations and ASCON open-source software for cryptographic hash operations.

TABLE 2 Experimental Hardware and Software Configurations

Characteristic	x86_64 (Desktop)	AVR ATmega2560 (MCU)
CPU Architecture	64-bit CISC (Intel Core i9-9900K)	8-bit RISC (AVR)
Core Configuration	8 cores / 16 threads	Single-core
Frequency	3.6 GHz	16 MHz
Memory (SRAM/DRAM)	64 GB DDR4	8 KB SRAM
Flash Storage	SSD	256 KB Flash, 4 KB EEPROM
Crypto Acceleration	AES-NI, AVX2	None
Operating Environment	Ubuntu 22.04 LTS	Bare-metal firmware
Power Class	High-performance desktop	Ultra-low-power MCU

Implementation Details: Our implementation of the (F)LRSHA schemes follows the architecture illustrated in Figure 4. Each entity (i.e., the Signer, Verifier, and the set of ComC servers $\{\mathcal{S}_\ell\}$) is realized using a distinct combination of optimized arithmetic and cryptographic primitives. Our *Signer* is designed for extremely resource-constrained IoT devices and therefore relies solely on *finite-field arithmetic* and the lightweight cryptographic hash function *Ascon-Hash-256*. The modular operations $(\text{Add}_q, \text{Sub}_q, \text{Mul}_q)$ are performed over a prime field $\mathbb{F}_q$ (where $q = 2^{255} - 19$). The signer avoids any ExpOps including elliptic-curve (EC) or modular exponentiation operations. The *Verifier* checks a signature's validity where it performs EC-based validation (e.g., $\alpha^s Y^e \bmod p$), the verifier executes only double scalar multiplications and hash evaluations, which are widely available in standard cryptographic libraries. Each commitment constructor server $\{\mathcal{S}_\ell\}$ maintains its own share r_ℓ and produces certified commitments (R_ℓ, C_ℓ) via one EC scalar multiplication and a digital signature SGN that are later verified and aggregated by the verifier. Overall, the implementation of (F)LRSHA is backward-compatible with standard-compliant

[†] https://github.com/ascon/ascon-c
[‡] https://github.com/openssl/openssl
[§] https://github.com/intel/intel-sgx-ssl
[¶] https://munacl.cryptojedi.org/atmega.shtml

cryptographic libraries where finite-field arithmetic operations and the presence of a cryptographic hash function are available. We note that we open-source our implementation for reproducibility at https://github.com/saifnouma/lrsha.

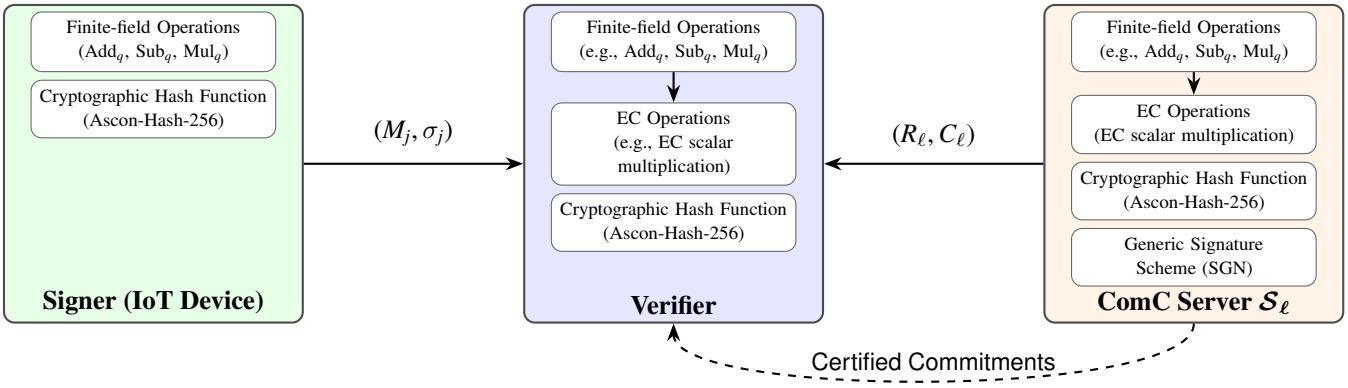

FIGURE 4 High-level overview of the (F)LRSHA building blocks and their interplay. Each entity's internal components show the main cryptographic and arithmetic methods used to implement each of the signature algorithms.

6.2 | Performance on Commodity Hardware

Table 3 illustrates the performance of LRSHA and their counterparts at the signer and verifier. Our main takeaways are as follows:

• *Signing Time:* LRSHA is 5× and 3.2× faster than ESEM$_2$ and standard Ed25519, respectively, but with a much smaller memory footprint than ESEM$_2$. FLRSHA offer forward security with only 1.65× decrease in speedup compared to LRSHA. Notably, FLRSHA is significantly faster than its forward-secure counterpart, XMSS$^{\text{MT}}$ by *several orders of magnitude*. HASES is also post-quantum-secure but suffers from a central root of trust that depends on a single hardware-supported ComC server in order to distribute large-sized public keys to verifiers. In contrast, FLRSHA distinguishes itself by employing a network of distributed ComC servers, thereby mitigating the risk associated with a single point of failure. Moreover, FLRSHA has *a magnitude times smaller signature* than that of HASES.

• *Signer Storage:* LRSHA consumes 200× less memory on resource-constrained signers compared to ESEM$_2$. This is attributed to the fact that ESEM stores a set of precomputed commitments to enhance signing efficiency. FLRSHA also outperforms the *optimal* generic forward-secure Ed25519-MMM and XMSS$^{\text{MT}}$ by having 241× and 27× lesser memory usage, respectively. FLRSHA consumes 7× more memory usage than its signer-efficient counterpart HASES, but without having a central root of trust on the ComC servers. Additionally, LRSHA schemes avoid costly EC-based operations by only executing simple arithmetic and symmetric operations.

• *Signature Size:* LRSHA has the smallest signature size among all counters, with a significant signing computational efficiency at its side. Only BLS have slightly larger (i.e., 1.5×) signature size, while its signing operates at an order of magnitude slower pace compared to our scheme. Similarly, FLRSHA surpasses its most signer-efficient forward-secure counterpart HASES with a 8.33× smaller signature. Consequently, LRSHA schemes prove to be the most resource-efficient in terms of processing, memory, and bandwidth.

• *Verification Time:* We consider: (i) computation of commitments at the ComC servers, (ii) network delay to transmit commitments and their signatures to the verifier, (iii) signature verification time at the verifier.

Unlike some other alternatives with distributed server support (e.g., [14]), our schemes permit an offline pre-computation of the commitments at ComC servers. This significantly reduces the verification delay to a mere 46μs for both LRSHA and FLRSHA. Moreover, verifiers may request public commitments in batches by sending a set of counters. FLRSHA's verification is slower than our fastest forward-secure hardware-assisted counterpart, HASES. However, in return, FLRSHA offers a magnitude of smaller signature sizes, faster signing, and resiliency against single-point failures.

• *Enhanced Security Properties:* We demonstrated that FLRSHA offers a superior performance trade-off. Our most efficient counterparts assume semi-honest and non-colluding server(s), are not forward-secure, and do not authenticate the commitment.

TABLE 3 Performance comparison of LRSHA and FLRSHA schemes and their counterparts on commodity hardware

Scheme	Signer					Verifier			ComC Servers				
	Signing Time (μs)	$\|sk\|$ (KB)	$\|\sigma\|$ (KB)	FS	OS	$\|PK\|$ (KB)	Ver Time (μs)	Storage Per Server (KB)	Comp. Per Server		CR	FID	OG
									Key Gen.	Cert. Gen.			
ECDSA [36]	17.07	0.03	0.06	×	✓	0.03	46.62						
Ed25519 [37]	16.34	0.03	0.06	×	✓	0.03	39.68						
Ed25519-BPV [37]	19.96	1.03	0.06	×	✓	0.03	39.68						
Ed25519-MMM [37]	82.32	53.09	1.2	✓	✓	0.03	267.04						
BLS [40]	278.6	0.06	0.05	×	✓	0.09	910.6						
RSA-3072 [10]	1235.74	0.5	0.25	×	✓	0.5	45.78			N/A			
SCRA-BLS [51]	15.31	16.06	0.05	×	×	0.09	43.52						
SCRA-RSA [51]	22.99	2 MB	0.27	×	×	0.53	51.2						
BLISS-I [12]	241.3	2.00	5.6	×	✓	7.00	24.61						
SPHINCS+ [78]	5,445.2	0.1	35.66	×	×	0.05	536.14						
XMSS [55]	10,682.35	5.86	4.85	✓	×	0.06	2,098.84						
HASES [17]	5.89	0.03	0.5	✓	×	32	10.41	32	624.64 μs	N/A	C	×	✓
$ESEM_2$ [14]	10.34	12.03	0.05	×	×	0.03	259.79	4.03	82.91 μs	N/A	SI	×	×
LRSHA	**3.23**	**0.06**	**0.05**	×	×	**0.03**	**45.96**	**0.03**	**2.56 *ms***	**22.67 μs**	**P**	✓	✓
FLRSHA	**5.35**	**0.22**	**0.05**	✓	×	**0.03**	**45.96**	**0.06**	**5.53 *ms***	**82.32 μs**	**P**	✓	✓

The input message size is 32 bytes. FS, OS, CR, FID, and OG denote forward security, online sampling, collusion resiliency, false input detection, and offline generation, respectively. In CR, (C, SI, and P) denote central, semi-honest, and protected, respectively. The maximum number of signing operations for FS schemes is set to $J = 2^{20}$. The number of ComC servers is $L = 3$. For SPHINICS+ parameters, $n = 16$, $h = 66$, $d = 22$, $b = 6$, $k = 33$, $w = 16$ and $\kappa = 128$. We benchmark the XMSS_SHA2_20_256 variant, allowing for 2^{20} signings. HASES parameters are ($l = 256, t = 1024, k = 16$). BPV parameters in $ESEM_2$ are ($n = 128, v = 40$). The parameter n in RSA is 3072-bit. For SCRA-BLS and SCRA-RSA, we set the optimal setting ($L = 32, b = 8$). The online verification time of LRSHA and FLRSHA is similar to that of Ed25519. The (offline) aggregation of commitments for LRSHA and FLRSHA is 9.1 μs and 16.99 μs, respectively. However, in $ESEM_2$, the verification includes the commitment aggregation (i.e., R). The ComC servers in ESEM require verifier input before generating the commitments. One can delegate the aggregation of partial commitments to a ComC server, but it will incur more network delay.

TABLE 4 Signing efficiency of LRSHA and FLRSHA schemes via offline-online technique

Scheme	**Commodity Hardware** (in μs)				**8-bit AVR ATmega2560 MCU** (in Cycles)				Additional Storage Cost (KB)*	FS
	Offline			Online	Offline			Online		
	Priv. Key Comp.	Priv. Key Upd.	Total	Signing	Priv. Key Comp.	Priv. Key Upd.	Total	Signing		
$ESEM_2$ [14]	6.25	0	6.25	4.09	1,006,144	0	1,006,144	549,236	96	×
Ed25519-BPV [80]	17.82	0	17.82	2.14	298,880	0	298,880	549,236	64	×
LRSHA	1.83	0	1.83	**1.4**	376,240	0	376,240	**121,949**	64	×
FLRSHA	2.04	1.95	3.99	**1.36**	712,800	767,872	1,480,672	**121,949**	128	✓

The running time is in μs for commodity hardware and in cycles for 8-bit AVR ATmega2560. The input message size is 32 Bytes. The number of ComC servers (i.e., L) is 3. The offline computation requires a storage penalty to save the computed keys in memory. SCRA-BLS and SCRA-RSA are not present due to the large private key size and expensive signing, respectively, compared to that of Ed25519-BPV, as in Table 3. SCRA-BLS and SCRA-RSA perform L EC point additions over a gap group and L modular multiplications over a large modulus (e.g., n is 3072-bit), respectively. Consequently, we considered Ed25519-BPV as the most efficient OO digital signature.
$\star$ LRSHA and FLRSHA require replenishment of additional stored data after every 2048 signings. For a total of 2^{20} signings, replenishment of additional data is 512 times.

In contrast, (i) LRSHA is forward-secure, (ii) leverages a set of SGX-supported ComC servers to ensure resiliency against collusions and single-point failures, (iii) authenticates all commitments to detect false inputs, (iv) avoids online sampling operations (unlike lattice-based schemes) and random number derivations, all of which are error-prone, especially on low-end embedded devices, (v) it avoids using Forking Lemma, and therefore has tighter security reduction than traditional Schnorr-based signatures (with the aid of distributed verification process).

• *Distinction from Cloud-Assisted Counterparts*: It is important to emphasize that the selected cloud-assisted hash-based digital signature scheme, HASES [17], lacks compatibility with distributed cryptographic settings due to its inherently non-algebraic structure. Otherwise, a security loss would occur, resulting from the sharing of the secret key among the L servers.

LRSHA schemes, as ECDLP variants, support additive homomorphism in the private key components ($y \leftarrow \sum y^{\ell}, r \leftarrow \sum r^{\ell}$), thus enabling secure key distribution without compromising the overall security guarantees.

6.3 | Performance on 8-bit AVR MCU

Table 5 compares the signing costs of our schemes with their counterparts on an 8-bit AVR MCU.

• *Signing:* LRSHA is 35.5× and 3× faster than the standard Ed25519 and ESEM, respectively. Table 4 showcases the signing efficiency of the most efficient candidate when pre-computation is considered. In our variant, the signer pre-computes symmetric keys via PRF calls to store them in memory offline and use them to generate signatures online. This strategy pushes the already efficient signing to the edge. For instance, the forward-secure FLRSHA with pre-computation is 8.5× and 3× faster than Ed25519-BPV and pre-computed ESEM_2, respectively, which are not FS.

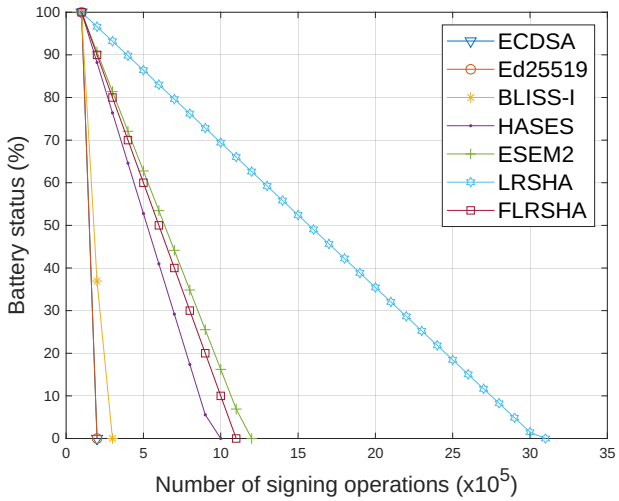

FIGURE 5 Impact of signing operations on the battery lifetime for LRSHA and FLRSHA schemes and their counterparts

• *Energy Consumption:* Table 5 depicts a comparative analysis of energy usage between LRSHA and FLRSHA schemes with their respective counterparts on the selected MCU device. Specifically, we connected a pulse sensor to the 8-bit MCU. Then, we contrast the energy usage of a single sampling reading from the pulse sensor with that of a single signature generation. Our experiments validate that LRSHA and FLRSHA exhibit superior performance when compared to the selected alternatives. This makes them the most suitable choices for deployment on resource-limited IoTs.

TABLE 5 Performance comparison of LRSHA and FLRSHA schemes and their counterparts on 8-bit AVR ATmega2560 MCU

Scheme	Private Key (KB)	Signature Size (KB)	Signing (Cycles)	Signing Energy (mJ)	N_{life} (#Signatures)	Signing/Sensor Energy Ratio (%)	FS	PF	SCB
ECDSA [36]	0.03	0.05	79, 185, 664	0.391	17.27×10^6	93.75	×	×	×
Ed25519 [37]	0.03	0.06	22, 688, 583	0.175	38.57×10^6	26.86	×	×	×
BLISS-I [12]	2.00	5.6	10, 537, 981	7.753	0.87×10^6	12.48	×	×	×
HASES [17]	0.05	0.5	1, 974, 528	0.696	9.70×10^6	2.34	✓	×	✓
ESEM_2 [14]	12.03	0.05	1, 555, 380	0.075	90.00×10^6	1.84	×	✓	✓
LRSHA	**0.06**	**0.03**	**498, 317**	**0.043**	$\mathbf{156.98 \times 10^6}$	**0.59**	×	✓	✓
FLRSHA	**0.22**	**0.06**	**1, 602, 749**	**0.089**	$\mathbf{75.84 \times 10^6}$	**1.9**	✓	✓	✓

The input message size is 32 Bytes. FS, PF, and SCB denote forward security, precomputation feasibility, and simple code base, respectively. ESEM_2 incurs a storage penalty of 12 KB at the signer side. The HORS parameters of HASES are ($l = 256, t = 1024, k = 16$).

Figure 5 illustrates the impact of signing operations on an 8-bit MCU. Consistent with Table 5, it reaffirms the efficacy of our schemes in prolonging low-end device battery life. LRSHA shows the longest battery life, depleting after 2^{30} signings. FLRSHA, though slightly less efficient than $ESEM_2$, offers collusion resilience and authenticated decentralized verification without a single root of trust or key escrow, extending battery life beyond HASES.

• *Energy Estimation using MICAz Model:* To further quantify the energy efficiency of our proposed schemes, we estimated the total energy consumption per signing operation and the corresponding battery lifetime using the *MICAz* energy model[81]. The MICAz node features an ATmega128L MCU operating at 16MHz, with 128KB flash and 4KB SRAM (i.e., similar computing capabilities to ATmega2560), powered by two AA batteries having a total energy capacity of $E_{\text{bat}} = 6750$J. Following the energy parameters in[81], the ATmega128L consumes approximately $E_{\text{cpu}} = 4.07$nJ per clock cycle, while the CC2420 ZigBee transceiver incurs $E_{\text{tx}} = 0.168\mu$J per transmitted bit. Using the cycle counts in Table 5, we derive the total computational energy for each scheme as $E_{\text{sign}} = N_{\text{cycles}} \cdot E_{cpu}$ and the total transmission cost for a s-bit signature as $E_{\text{tx,sign}} = s \cdot E_{\text{tx}}$. Therefore, the total energy per signing query is then $E_{\text{total}} = E_{\text{sign}} + E_{\text{tx, sign}}$, and the estimated number of signatures supported per battery is $N_{\text{life}} = E_{\text{bat}}/E_{\text{total}}$.

Under the evaluated setting, LRSHA consumes an energy of ≈ 2.03mJ per signing, whereas FLRSHA incurs a higher but still moderate cost of 6.52mJ per signing operation. Both values are markedly lower than the energy overheads observed in our selected counterparts, namely $ESEM_2$ (≈ 12.4mJ) and HASES (≈ 15.8mJ). Given double AA batteries as a power supply, the operational longevity of the MCU device becomes apparent, where LRSHA can support $3.3 \cdot 10^6$ signing operations, while FLRSHA supports around 10^6 signatures before full battery depletion occurs. Observe that this matches the results in Figure 5 where we did not account for signature transmission into the energy depletion.

When projected in terms of temporal operation, this translates into an estimated lifetime of approximately 6.3 years for LRSHA and 1.9 years for FLRSHA if the device generates one signature per minute. Even under a low-frequency signing of one signature generation per hour, the theoretical autonomy extends to several decades, without battery replacement. Therefore, we confirm the suitability of LRSHA and FLRSHA for long-term deployment in energy-constrained sensor networks.

7 | CONCLUSION

In this paper, we developed two new digital signatures referred to as *Lightweight and Resilient Signatures with Hardware Assistance* (LRSHA) and its forward-secure version (FLRSHA). Our schemes harness the commitment separation technique to eliminate the burden of generation and transmission of commitments from signers and integrate it with a key evolution strategy to offer forward security. At the same time, they introduce hardware-assisted ComC servers that permit an authenticated and breach-resilient construction of one-time commitments at the verifier without interacting with the signer. We used Intel-SGX to realize the distributed verification approach that mitigates the collusion concerns and reliance on semi-honest servers while avoiding single-point failures in centralized hardware-assisted signatures. Our distributed verification also permits the offline construction of one-time commitments before the signature verification, thereby offering a fast online verification.

Our new approaches translate into significant performance gains while enhancing breach resilience on both the signer and verifier sides. Specifically, to the best of our knowledge, FLRSHA is the only FS signature that has a comparable efficiency to a few symmetric MAC calls, with a compact signature size, but without putting a linear public key overhead or computation burden on the verifiers. Our signing process only relies on simple modular arithmetic operations and hash calls without online random number generation and therefore avoids complex arithmetics and operations that are shown to be prone to certain types of side-channel attacks, especially on low-end devices. We formally prove the security of our schemes and validate their performance with full-fledged open-source implementations on both commodity hardware and 8-bit AVR microcontrollers. We believe that our findings will foster further innovation in securing IoT systems and contribute to the realization of a more secure and resilient IoT infrastructure.

REFERENCES

1. Shafique K, Khawaja BA, Sabir F, Qazi S, Mustaqim M. Internet of things (IoT) for next-generation smart systems: A review of current challenges, future trends and prospects for emerging 5G-IoT scenarios. *Ieee Access.* 2020;8:23022–23040.
2. El Saddik A, Laamarti F, Alja'Afreh M. The potential of digital twins. *IEEE Instrumentation & Measurement Magazine.* 2021;24(3):36–41.
3. Marin E, Singelée D, Garcia FD, Chothia T, Willems R, Preneel B. On the (in) security of the latest generation implantable cardiac defibrillators and how to secure them. In: 2016:226–236.
4. Adamson PB. Pathophysiology of the transition from chronic compensated and acute decompensated heart failure: new insights from continuous monitoring devices. *Current heart failure reports.* 2009;6(4):287–292.
5. Glas B, Guajardo J, Hacioglu H, Ihle M, Wehefritz K, Yavuz AA. Signal-based Automotive Communication Security and Its Interplay with Safety Requirements. ESCAR, Embedded Security in Cars Conference, November 2012; 2012.

6. Rao PM, Deebak B. A Comprehensive Survey on Authentication and Secure Key Management in Internet of Things: Challenges, Countermeasures, and Future Directions. *Ad Hoc Networks.* 2023:103159.
7. Mudgerikar A, Bertino E. Iot attacks and malware. *Cyber Security Meets Machine Learning.* 2021:1–25.
8. Avoine G, Canard S, Ferreira L. Symmetric-key authenticated key exchange (SAKE) with perfect forward secrecy. In: 2020:199–224.
9. Nouma SE, Yavuz AA. Lightweight Digital Signatures for Internet of Things: Current and Post-Quantum Trends and Visions. In: IEEE. 2023:1–2.
10. Seo JH. Efficient digital signatures from RSA without random oracles. *Information Sciences.* 2020;512:471–480.
11. Peng C, Luo M, Li L, Choo KKR, He D. Efficient certificateless online/offline signature scheme for wireless body area networks. *IEEE Internet of Things Journal.* 2021;8(18):14287–14298.
12. Ducas L, Durmus A, Lepoint T, Lyubashevsky V. Lattice Signatures and Bimodal Gaussians. In: Canetti R, Garay JA., eds. *Advances in Cryptology – CRYPTO 2013: 33rd Annual Cryptology Conference. Proceedings, Part I* 2013:40–56.
13. Behnia R, Yavuz AA. *Towards Practical Post-Quantum Signatures for Resource-Limited Internet of Things*:119–130; New York, NY, USA: ACM . 2021.
14. Ozmen MO, Behnia R, Yavuz AA. Energy-Aware Digital Signatures for Embedded Medical Devices. In: IEEE. 2019.
15. Yavuz AA, Ozmen MO. Ultra Lightweight Multiple-time Digital Signature for the Internet of Things Devices. *IEEE Transactions on Services Computing.* 2019;15(1):215–227.
16. Chen X, Xu S, He Y, Cui Y, He J, Gao S. LFS-AS: lightweight forward secure aggregate signature for e-health scenarios. In: IEEE. 2022:1239–1244.
17. Nouma SE, Yavuz AA. Post-Quantum Forward-Secure Signatures with Hardware-Support for Internet of Things. In: IEEE. 2023:4540-4545.
18. Yavuz AA, Ning P, Reiter MK. BAF and FI-BAF: Efficient and Publicly Verifiable Cryptographic Schemes for Secure Logging in Resource-Constrained Systems. *ACM Trans on Information System Security.* 2012;15(2).
19. Yao ACC, Zhao Y. Online/offline signatures for low-power devices. *IEEE Transactions on Information Forensics and Security.* 2012;8(2):283–294.
20. Wang Q, Khurana H, Huang Y, Nahrstedt K. Time Valid One-Time Signature for Time-Critical Multicast Data Authentication. In: 2009.
21. Camara C, Peris-Lopez P, Tapiador JE. Security and privacy issues in implantable medical devices: A comprehensive survey. *Journal of Biomedical Informatics.* 2015;55:272 - 289.
22. Ma D. Practical forward secure sequential aggregate signatures. In: 2008:341–352.
23. Seyitoglu EUA, Yavuz AA, Ozmen MO. Compact and Resilient Cryptographic Tools for Digital Forensics. In: IEEE. 2020:1-9.
24. MITRE . Indicator Removal: Clear Linux or Mac System Logs. https://attack.mitre.org/techniques/T1070/002/; 2025. Accessed: April 5, 2025.
25. Liao W, Sun J, Wang H, Gu Z, Yang J. Semantic-Integrated Online Audit Log Reduction for Efficient Forensic Analysis. In: Springer. 2024:318–333.
26. Abdalla M, Benhamouda F, Pointcheval D. On the tightness of forward-secure signature reductions. *Journal of Cryptology.* 2019;32(1):84–150.
27. Hülsing A, Busold C, Buchmann J. Forward Secure Signatures on Smart Cards: Preliminary version. In: 2012:66–80.
28. Itkis G, Reyzin L. Forward-Secure Signatures with Optimal Signing and Verifying. In: 2001:332-354.
29. Bellare M, Miner S. A Forward-Secure Digital Signature Scheme. In: 1999:431–448.
30. Malkin T, Micciancio D, Miner SK. Efficient Generic Forward-Secure Signatures with an Unbounded Number Of Time Periods. In: Springer-Verlag 2002:400–417.
31. Espitau T, Fouque PA, Gérard B, Tibouchi M. Side-channel attacks on BLISS lattice-based signatures: Exploiting branch tracing against strongswan and electromagnetic emanations in microcontrollers. In: 2017:1857–1874.
32. Ouyang W, Wang Q, Wang W, Lin J, He Y. SCB: Flexible and Efficient Asymmetric Computations Utilizing Symmetric Cryptosystems Implemented with Intel SGX. In: 2021:1–8.
33. Yavuz AA, Nouma S. Hardware supported authentication and signatures for wireless, distributed and blockchain systems.; 2023. US Patent App. 18/188,749.
34. Liu J, Yang J, Wu W, Huang X, Xiang Y. Lightweight Authentication Scheme for Data Dissemination in Cloud-Assisted Healthcare IoT. *IEEE Transactions on Computers.* 2022;72(5):1384–1395.
35. Wang C, Wang D, Duan Y, Tao X. Secure and Lightweight User Authentication Scheme for Cloud-Assisted Internet of Things. *IEEE Trans on Information Forensics and Security.* 2023.
36. American Bankers Association*ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. 1999.
37. Bernstein DJ, Duif N, Lange T, Schwabe P, Yang BY. High-speed high-security signatures. *Journal of Cryptographic Engineering.* 2012;2(2):77–89.
38. Winderickx J, Braeken A, Singelée D, Mentens N. In-depth energy analysis of security algorithms and protocols for the Internet of Things. *J. of Cryptographic Engineering.* 2022;12(2):137–149.
39. Li T, Wang H, He D, Yu J. Permissioned blockchain-based anonymous and traceable aggregate signature scheme for Industrial Internet of Things. *IEEE Internet of Things Journal.* 2020;8(10):8387–8398.
40. Boneh D, Lynn B, Shacham H. Short Signatures from the Weil Pairing. *J. Cryptol.*. 2004;17(4):297–319.
41. Boldyreva A, Gentry C, O'Neill A, Yum D. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In: 2007:276–285.
42. Drijvers M, Gorbunov S, Neven G, Wee H. Pixel: Multi-signatures for Consensus.. In: 2020:2093–2110.
43. Baee MAR, Simpson L, Boyen X, Foo E, Pieprzyk J. On the efficiency of pairing-based authentication for connected vehicles: Time is not on our side!. *IEEE Transactions on Information Forensics and Security.* 2021;16:3678–3693.
44. Gouvêa CP, Oliveira LB, López J. Efficient software implementation of public-key cryptography on sensor networks using the MSP430X microcontroller. *Journal of Cryptographic Engineering.* 2012;2:19–29.
45. Vollala S. Energy efficient triple-modular exponential techniques for batch verification schemes. *Journal of Cryptographic Engineering.* 2024;14(2):295–309.
46. Reyzin L, Reyzin N. Better than BiBa: Short One-Time Signatures with Fast Signing and Verifying. In: Batten L, Seberry J., eds. *Information Security and Privacy: 7th Australasian Conference, ACISP 2002 Melbourne, Australia, July 3–5, 2002 Proceedings*Springer Berlin Heidelberg 2002; Berlin, Heidelberg:144–153.
47. Schnorr C. Efficient Signature Generation by Smart Cards. *Journal of Cryptology.* 1991;4(3):161–174.
48. Boyko V, Peinado M, Venkatesan R. Speeding up discrete log and factoring based schemes via precomputations. In: :221–235.
49. Ateniese G, Bianchi G, Capossele A, Petrioli C. Low-cost standard signatures in wireless sensor networks: a case for reviving pre-computation techniques?. In: 2013.

50. Coron JS, Gini A. A polynomial-time algorithm for solving the hidden subset sum problem. In: 2020:3–31.
51. Yavuz AA, Mudgerikar A, Singla A, Papapanagiotou I, Bertino E. Real-time digital signatures for time-critical networks. *IEEE Transactions on Information Forensics and Security.* 2017;12(11):2627–2639.
52. Zaverucha G, Stinson D. Short One-Time Signatures. Cryptology ePrint Archive, Report 2010/446; 2010.
53. Zhang Y, Deng RH, Zheng D, Li J, Wu P, Cao J. Efficient and robust certificateless signature for data crowdsensing in cloud-assisted industrial IoT. *IEEE Transactions on Industrial Informatics.* 2019;15(9):5099–5108.
54. Abdalla M, Reyzin L. A New Forward-Secure Digital Signature Scheme. In: Springer-Verlag 2000:116–129.
55. Cooper DA, Apon DC, Dang QH, et al. Recommendation for stateful hash-based signature schemes. *NIST Special Publication.* 2020;800:208.
56. Lang F, Wang W, Meng L, Lin J, Wang Q, Lu L. MoLE: Mitigation of Side-channel Attacks against SGX via Dynamic Data Location Escape. In: 2022:978–988.
57. Thang H, Ozmen MO, Jang Y, Yavuz AA. Hardware-Supported ORAM in Effect: Practical Oblivious Search and Update on Very Large Dataset. In: 2019.
58. Arshad H, Picazo-Sanchez P, Johansen C, Schneider G. Attribute-based encryption with enforceable obligations. *Journal of Cryptographic Engineering.* 2023;13(3):343–371.
59. Volgushev N, Schwarzkopf M, Getchell B, Varia M, Lapets A, Bestavros A. Conclave: secure multi-party computation on big data. In: 2019:1–18.
60. Zhou X, Luo M, Vijayakumar P, Peng C, He D. Efficient certificateless conditional privacy-preserving authentication for VANETs. *IEEE Trans on Vehicular Technology.* 2022;71(7):7863–7875.
61. Yang W, Wang S, Mu Y. An enhanced certificateless aggregate signature without pairings for e-healthcare system. *IEEE Internet of Things Journal.* 2020;8(6):5000–5008.
62. Behnia R, Yavuz AA, Ozmen MO, Yuen TH. Compatible Certificateless and Identity-Based Cryptosystems for Heterogeneous IoT. In: Springer International Publishing 2020; Cham:39–58.
63. Jiang M, Duong DH, Susilo W. Puncturable Signature: A Generic Construction and Instantiations. In: Springer. 2022:507–527.
64. Wang C, Ming Y, Liu H, Zhang S, Lu R. Puncturable Signature and Applications in Privacy-Aware Data Reporting for VDTNs. *IEEE Transactions on Services Computing.* 2025.
65. Chen X, Huang Q, Li H, Liao Z, Susilo W. A novel identity-based multi-signature scheme over NTRU lattices. *Theoretical Computer Science.* 2022;933:163–176.
66. Sedghighadikolaei K, Yavuz AA. A Comprehensive Survey of Threshold Digital Signatures: NIST Standards, Post-Quantum Cryptography, Exotic Techniques, and Real-World Applications. *arXiv preprint arXiv:2311.05514.* 2023.
67. Grissa M, Yavuz A, Hamdaoui B. An efficient technique for protecting location privacy of cooperative spectrum sensing users. In: IEEE. 2016:915–920.
68. Zheng W, Lai CF, He D, Kumar N, Chen B. Secure storage auditing with efficient key updates for cognitive industrial IoT environment. *IEEE Transactions on Industrial Informatics.* 2020;17(6):4238–4247.
69. Ateniese G, Bianchi G, Capossele AT, Petrioli C, Spenza D. Low-Cost Standard Signatures for Energy-Harvesting Wireless Sensor Networks. *ACM Trans. Embed. Comput. Syst..* 2017;16(3):64:1–64:23.
70. Genkin D, Valenta L, Yarom Y. May the fourth be with you: A microarchitectural side channel attack on several real-world applications of curve25519. In: 2017:845–858.
71. Costello C, Longa P. Schnorrq: Schnorr signatures on fourq. *MSR Tech Report.* 2016.
72. Katz J, Lindell Y. *Introduction to modern cryptography*. CRC press, 2020.
73. Costan V, Lebedev I, Devadas S. Sanctum: Minimal hardware extensions for strong software isolation. In: 2016:857–874.
74. Silva RD, Navaratna I, Kumarasiri M, Alawatugoda J, Wen CC. On power analysis attacks against hardware stream ciphers. *International Journal of Information and Computer Security.* 2022;17(1-2):21–35.
75. Yavuz AA. ETA: Efficient and Tiny and Authentication for heterogeneous wireless systems. In: WiSec '13. 2013:67–72.
76. Nouma SE, Yavuz AA. Practical Cryptographic Forensic Tools for Lightweight Internet of Things and Cold Storage Systems. In: 2023:340–353.
77. Marson GA, Poettering B. Even more practical secure logging: Tree-based seekable sequential key generators. In: 2014:37–54.
78. Bernstein DJ, Hülsing A, Kölbl S, Niederhagen R, Rijneveld J, Schwabe P. The SPHINCS+ signature framework. In: 2019:2129–2146.
79. Dobraunig C, Eichlseder M, Mendel F, Schläffer M. Ascon v1. 2: Lightweight authenticated encryption and hashing. *Journal of Cryptology.* 2021;34:1–42.
80. Ozmen MO, Yavuz AA. Dronecrypt - An Efficient Cryptographic Framework for Small Aerial Drones. In: 2018:1–6.
81. Piotrowski K, Langendoerfer P, Peter S. How public key cryptography influences wireless sensor node lifetime. In: 2006:169–176.