Post-Quantum Forward-Secure Signatures with Hardware-Support for Internet of Things

Saif E. Nouma and Attila A. Yavuz
Department of Computer Science and Engineering, University of South Florida, Tampa, FL, USA
{saifeddinenouma, attilaayavuz}@usf.edu

***Abstract*—Digital signatures provide scalable authentication with non-repudiation and therefore are vital tools for the Internet of Things (IoT). IoT applications harbor vast quantities of low-end devices that are expected to operate for long periods with a risk of compromise. Hence, IoT needs post-quantum cryptography (PQC) that respects the resource limitations of low-end devices while offering compromise resiliency (e.g., forward-security). However, as seen in NIST PQC efforts, quantum-safe signatures are extremely costly for low-end IoT. These costs become prohibitive when forward security is considered.**

We propose a highly lightweight post-quantum digital signature called *HArdware-Supported Efficient Signature* (`HASES`) that meets the stringent requirements of resource-limited signers (processor, memory, bandwidth) with forward security. `HASES` transforms a key-evolving one-time hash-based signature into a polynomially unbounded one by introducing a public key oracle via secure enclaves. The signer is non-interactive and only generates a few hashes per signature. Unlike existing hardware-supported alternatives, `HASES` does not require a secure-hardware on the signer, which is infeasible for low-end IoT. `HASES` also does not assume non-colluding servers that permit scalable verification. We proved that `HASES` is secure and implemented it on the commodity hardware and the 8-bit AVR ATmega2560 microcontroller. Our experiments confirm that `HASES` is $271\times$ and $34\times$ faster than (forward-secure) XMSS and (plain) Dilithium. `HASES` is more than twice and magnitude more energy-efficient than (forward-secure) ANT and (plain) BLISS, respectively, on an 8-bit device. We open-source `HASES` for public testing and adaptation.

***Index Terms*—Authentication; Internet of Things; post-quantum security; embedded devices; lightweight cryptography.**

I. Introduction

The Internet of Things (IoT) is comprised of vast quantities of connected computing devices that collect, process, and transmit sensitive data at large scales. Authentication and integrity are fundamental security services to protect sensitive data and critical IoT infrastructures from various attacks such as man-in-the-middle, impersonation, data tampering, and others.

Digital signatures provide scalable authentication and integrity via public key infrastructures. Furthermore, they offer non-repudiation and public verifiability, which are useful for various real-life applications (e.g., financial, legal). Therefore, digital signatures are essential primitives to ensure security and trust for the IoT. However, emerging IoT applications demand security and performance features beyond what classical digital signatures offer. An ideal digital signature for the emerging IoT should offer (at minimum) the following properties:

(i) *Post-quantum Security*: With the arrival of quantum-computers, Shor's algorithm [1] can break cryptosystems that rely on conventional intractability assumptions (e.g., ECDSA [2], SchnorrQ [3]). IoT infrastructures are expected to operate for long periods of time and offer long-term security for sensitive data. Therefore, IoT ideally needs Post-Quantum (PQ) digital signatures to achieve long-term trust and security. Despite their merits, PQ signatures are significantly more expensive than conventional pre-quantum signatures [4].

(ii) *Compromise-resiliency*: In IoT applications such as smart-city and smart-building, sensors/actuators work in open environments that make them vulnerable to compromise. Moreover, military IoT needs to work in hostile environments. Overall, IoT devices are vulnerable to breaches via either physical means or malware [5]. Hence, it is important for a digital signature to offer compromise-resiliency properties like forward security. This property guarantees authenticity and integrity before a system breach point [6]. The forward-security involves updating the secret key periodically. Forward-secure (FS) signatures are (in some cases significantly) costlier than their standard (plain) signature counterparts.

(iii) *High Efficiency and Architectural Feasibility*: IoT applications harbor a large number of resource-limited devices [5], which have limited processing, memory, and battery capacities. Hence, it is paramount for a digital signature to be lightweight in terms of computation, communication, and space. However, PQ signatures are extremely costly for low-end devices, even without considering features like forward-security. Moreover, such low-end devices cannot harbor sophisticated hardware (e.g., secure enclaves) or participate in highly interactive protocols that usually benefit resourceful architectural entities only.

A. Related Work and Limitations of the State-of-the-Art

NIST PQC Standards: NIST recently announced the PQ digital signatures to be standardized [11], namely Dilithium [4], Falcon [11], and SPHINCS+ [12]). The hash-based SPHINCS+ offers high security, but with very large signature sizes (e.g., 35.66 KB) and signer computation, making it prohibitively costly for low-end devices. Dilithium and Falcon are lattice-based signatures, where Dilithium offers better signer efficiency. However, it is significantly more expensive than conventional (pre-quantum) solutions (e.g., ECDSA [2]) on commodity hardware. It is extremely costly for low-end devices (no open-source implementation available on 8-bit hardware). To the best of our knowledge, BLISS-I [7] is the only lattice-based scheme with an open-source implementation on 8-bit devices but was eliminated from the NIST competition.

ICC 2023 - IEEE International Conference on Communications | 978-1-5386-7462-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICC45041.2023.10279236

TABLE I
PERFORMANCE COMPARISON OF HASES SCHEMES AND THEIR COUNTERPARTS ON AVR ATMEGA2560 MICROCONTROLLER

Scheme	Signing Cycles	Private Key (KB)	Signature Size (KB)	CPU Energy (mJ)	Post-Quantum Promise	Forward Security	Rejection/Gaussian (Sampling)	Deterministic Signing
ECDSA [2]	$81,324,870$	0.03	0.06	508.28	×	×	×	×
SchnorrQ [3]	$5,211,321$	0.03	0.06	32.57	×	×	×	×
BLISS-I [7]	$10,537,981$	2	5.6	65.86	✓	×	✓	×
ANT-FS-II [8]	718,678	0.09	0.432	4.49	✓	✓	×	✓
HASES	**306,762**	**0.03**	**0.424**	**1.92**	✓	✓	×	✓

We have excluded both EPID [9] and SCB [10], as they are not inline with our system model (i.e., signers are resource-limited devices that cannot encapsulate secure enclaves). The details of experiment settings, hardware/software configurations, and cryptographic parameters are given in Section V.

Forward-secure (FS) PQ Signatures: FS signatures employ a key-evolution strategy by periodically updating the private key (and managing the corresponding public key(s)). NIST PQC standards are not FS. One can transform them into FS by applying generic transformations (e.g., [6]). However, the most efficient generic method introduces a $log_2(J)$ signing overhead blow-up (J is the maximum number of signatures). For example, Dilithium [4] with $J = 2^{20}$ will be at least $20\times$ costlier, which is impractical for low-end devices. XMSS^{MT} [13] is currently the only FS and PQ candidate being considered for future recommendation by NIST. However, it is more than a magnitude times costlier than Dilithium, and is currently impractical for low-end platforms.

Distributed Verification: One line of research (e.g., lattice-based ANT [8]) enables signers to delegate the public key construction to a set of distributed servers to achieve PQ and FS signatures. Despite their merits, such approaches assume non-colluding servers, which might be a risky assumption for some real-life applications. Moreover, distributed verification introduces heavy network delays and outage risks.

Secure Hardware-based Solutions: Another line of research exploits the availability of trusted execution environments in modern architectures to achieve higher cryptographic functionalities: (i) It is possible to emulate asymmetric cryptosystems from symmetric-key only schemes (e.g., MACs) with a secure enclave (e.g., Intel Software Guard (SGX) [14]). While efficient, such approaches require each party to have a local secure enclave (e.g., SCB [10]), which is not practical for low-end IoT devices. Moreover, they provide restricted public verifiability (only SGX-enabled devices) and lack non-repudiation (due to shared symmetric keys), which is a critical need for numerous real-life applications. (ii) Initially formalized by IRON [15], and followed by many others (e.g., IBBE-SGX [16]), it is possible to build a generic functional encryption framework based on secure enclaves. Alternatively, EPID [9] proposes a group signature that achieves anonymous signing by using secure enclaves on the signer side. However, the vast majority of these works require secure enclaves on the senders, which is not feasible for low-end IoT. Moreover, they focus on privacy enhancement, which is orthogonal to our work.

We aim to address the following research challenges: *How to achieve forward-secure and post-quantum signatures that are practical for highly resource-limited IoT devices? How to achieve this goal without assuming a secure enclave on the signer or non-colluding distributed servers at the verifier?*

B. Our Contribution

We created a new highly lightweight PQ signature called *HArdware-Supported Efficient Signature* (HASES) that achieves near signer optimal efficiency and forward-security without assuming non-colluding distributed verification or secure enclaves at the signer. Our approach is to transform near-optimal one-time hash-based signatures [17], which form the basis of NIST PQC finalist SPHINCS+ [12] and recommendation XMSS^{MT} [18], into a multiple-time hash-based signature, but without consorting with heavy sub-tree construction or secure enclaves at the signer. Instead, we eliminate the burden of public key generation/transmission from the signer via the public key oracle PUKO, which is realized via secure enclaves. We outline the desirable properties of HASES as below:

(1) Signer Computation/Energy Efficiency: HASES executes only a small-constant number of hash calls (e.g., 13) per forward-secure signing. This makes it $271\times$ and $34\times$ faster than XMSS^{MT} and Dilithium, which is the only FS and the most efficient NIST PQC finalist candidate (not FS), respectively. As shown in Table I, HASES is $34\times$ and $2.4\times$ more energy efficient than BLISS and ANT, which are the only feasible alternatives with an implementation on an 8-bit device, with standard and distributed verification, respectively. Moreover, HASES is even $16\times$ more energy efficient than SchnorrQ [3], which is neither PQ nor FS. Hence, HASES is the most energy-efficient PQ and FS alternative.

(2) Compact Signatures: The signature size of HASES is identical to HORS, which makes it the most compact signature among its counterparts. Notably, its signature is a magnitude smaller than XMSS^{MT} (only hash-based FS alternative), while still being smaller than ANT without non-colluding servers.

(3) Non-Interactive and Scalable Multi-Users: HASES alleviates the burden of conveying/certifying public keys from the signer, thereby making it independent from PUKO and verifiers. PUKO can supply any public key of a valid signer to verifiers either beforehand or on-demand, with adjustable storage overhead, permitting a scalable public key management service for massive-size IoT networks with millions of users.

(4) Architectural Feasibility: (i) Unlike some secure-hardware-based solutions (e.g., [9], [10]), HASES does not require a secure-enclave on the signer, which is not feasible for low-end IoT. (ii) Unlike ANT [8], which is currently the only feasible FS and PQ signature on 8-bit devices, HASES does not assume non-colluding servers while also being faster.

(5) High Security: i) HASES only relies on hash calls, and therefore is free from rejection/Gaussian sampling that permits devastating side-channel attacks [19] in its lattice-based counterparts (e.g., BLISS-I [7]). (ii) HASES signature generation is deterministic and therefore avoids vulnerabilities of weak pseudo-random generators typically found in low-end IoT devices. (iii) HASES schemes achieve both PQ and FS properties.

(5) Full-fledge Implementation: We fully implemented HASES on both 8-bit AVR microcontroller and commodity hardware at the signer side, and with Intel SGX as the PUKO, and with commodity hardware at the verifier side. Our implementation can be found below: https://github.com/SaifNOUMA/HASES

We note that, compared to existing PQ signatures (Table 1 and Section V), HASES attains its significant performance advantages and forward-security by introducing a public key oracle via secure enclaves at the verifier. Our analysis shows that this is a highly favorable trade-off given the signer optimality, forward security, avoidance of secure hardware at the signer, verification flexibility (e.g., no non-colluding servers), and the availability of secure enclaves in the modern clouds.

II. Preliminaries

Notation: $\|$ means string concatenation. $|x|$ denotes the bit length of variable x. $x \xleftarrow{\$} \mathcal{S}$ means selecting x randomly from set $\mathcal{S}$. We denote by $\{0,1\}^*$ the set of binary strings of any finite length. The set of items q_i for $i = 0, \ldots, n-1$ is denoted by $\{q_i\}_{i=0}^{n-1}$. $f : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ is one-way function. $H_i : \{0,1\}^* \rightarrow \{0,1\}^\kappa, i \in \{0,1,2\}$ are cryptographic hash functions. $H^k(.)$ means k consecutive applications of H to form a hash chain on the given input.

Definition 1 A Hardware-assisted Forward-Secure Multi-User signature scheme consists of four algorithms HFMU-SGN $=$ (Kg, Sig, PKConstr, Ver):

- $(msk, \overrightarrow{sk_1}, \overrightarrow{sk_p}, I) \leftarrow$ HFMU-SGN.Kg$(1^\kappa, \overrightarrow{ID}, J)$: Given the security parameter κ, the signer identifier list $\overrightarrow{ID}$ and the maximum number of signatures to be computed J, it returns the master key msk, the initial and precomputed private key set $\overrightarrow{sk_1}$ and $\overrightarrow{sk_p}$, respectively, for $\overrightarrow{ID}$ and the system-wide parameters I.
- $\sigma_j^i \leftarrow$ HFMU-SGN.Sig(sk_j^i, M_j^i): Given the private key sk_j^i of ID_i and a message M_j^i, it returns the signature σ_j^i, updates $sk_j^i \leftarrow sk_{j+1}^i$ and $j = j + 1$, and deletes sk_j^i.
- $PK_j^i \leftarrow$ HFMU-SGN.PkConst$(msk, \overrightarrow{sk_p}, ID_i, j)$: Given the signer identifier $ID_i \in \overrightarrow{ID}$ and state j, it returns the corresponding public key PK_j^i under msk.
- $b \leftarrow$ HFMU-SGN.Ver$(PK_j^i, M_j^i, \sigma_j^i)$: Given PK_j^i, a message M_j^i, and its corresponding signature σ_j^i, it returns a bit b, with $b = 1$ meaning *valid*, and $b = 0$ otherwise.

Definition 2 Hash to Obtain Random Subset [17] HORS $=$ (Kg, Sig, Ver) is defined as follows:

- $(sk, PK, I) \leftarrow$ HORS.Kg(1^κ): Given the security parameter κ, it first generates $I \leftarrow (l, k, t)$, and then t random l-bit strings $\{s_i\}_{i=1}^t$ and $v_i \leftarrow f(s_i)$,$\forall i = 1, \ldots, t$. It sets $sk \leftarrow \{s_i\}_{i=1}^t$ and $PK \leftarrow \{v_i\}_{i=1}^t$.
- $\sigma \leftarrow$ HORS.Sig(sk, M): Given (sk, M), it computes $h \leftarrow H_0(M)$, splits it as $\{h_j\}_{j=1}^k$ where $|h_j| = \log t$ and interprets them as integers $\{i_j\}_{j=1}^k$. It sets $\sigma \leftarrow \{s_{i_j}\}_{j=1}^k$.
- $b \leftarrow$ HORS.Ver(PK, M, σ): Given PK, M, and σ, it computes $\{i_j\}_{j=1}^k$ as in HORS.Sig(.) and checks if $f(s'_j) = v_{i_j}$, $j = 1, \ldots, k$, returns $b = 1$, else $b = 0$.

III. System and Security Models

System Model: There are three types of entities in the system.

(i) The signers are storage, computational, bandwidth, and power-limited IoT devices (e.g., medical implants, RFID tags). Therefore, signing efficiency and compact signature/key sizes are in our setting. The signers are not required to have secure enclaves. They only broadcast messages and signatures without interacting with any other entity or conveying public keys. (ii) Verifiers can be any (untrusted) entity (laptop, cloud server) that receives signatures from the signers and public keys from a supplier. (iii) We introduce a public key management service that we refer to as *PUblic Key Oracle* (PUKO). We realize PUKO with secure enclave (e.g., Intel SGX, ARM Trustzone [1]) due to its broad availability in modern clouds. Unlike the state-of-the-art rely on non-colluding servers to replenish public keys (e.g., [8]), our approach permits PUKO to be on the verifier (immediate public access) or just with a single round of request to a remote cloud server. We implemented HASES with Intel SGX. However, HASES can be realized with *any* secure hardware offering a trusted execution with standard cryptographic hash functions.

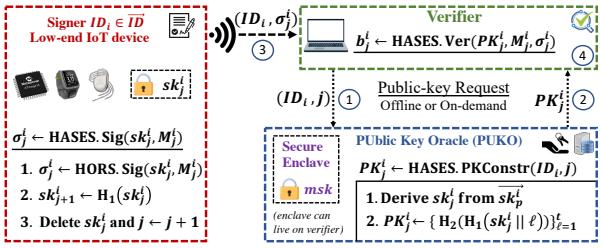

Fig. 1. High-level description of HASES scheme

Security Model: We follow the standard Forward-secure Existential Unforgeability Under Chosen Message Attack (*FEU-CMA*) [6] by incorporating a public oracle as in [8].

Definition 3 *FEU-CMA* experiment $Expt_{\text{HFMU-SGN}}^{FEU\text{-}CMA}$ is as:

- $(msk, \overrightarrow{sk_1}, \overrightarrow{sk_p}, I) \leftarrow$ HFMU-SGN.Kg(1^κ)
- $(M^*, \sigma^*) \leftarrow \mathcal{A}^{\text{Sig}_{sk_j}(.), \text{PKConstr}_{msk}(.), \text{Break-In(j)}}(.)$
- If $1 =$ HFMU-SGN.Ver(PK^*, M^*, σ^*) and M^* was not queried to $\text{Sig}_{sk_j}(.)$ where PK^* is output of $\text{PKConstr}_{msk}(.)$, return 1, else, return 0.

The *FEU-CMA* advantage of the adversary $\mathcal{A}$ is defined as

$$Adv_{HFMU}^{FEU\text{-}CMA}(t, q_s, 1) = Pr[Expt_{\text{HFMU-SGN}}^{FEU\text{-}CMA}(\mathcal{A})] = 1]$$

[1] https://www.arm.com/technologies/trustzone-for-cortex-a

, where $\mathcal{A}$ is having time complexity t, making at most q_s queries and a single query to $(\texttt{Sig}_{sk_j}(.), \texttt{PKConstr}_{msk}(.))$ and `Break-In(j)` oracles, respectively.

The oracles reflect how a `HFMU-SGN` scheme works: The signing oracle $\texttt{Sig}_{sk_j}(.)$ returns signature σ_j to $\mathcal{A}$ under sk_j on a message M_j. $\texttt{PKConstr}_{msk}(.)$ acts like `PUKO` and returns PK_j of $ID \in \overrightarrow{ID}$ under msk. Given the state $1 < j < J$, `Break-In(j)` oracle returns sk_{j+1} to $\mathcal{A}$.

• *EU-CMA experiment for HORS*: It is as in *FEU-CMA* experiment but without `Break-In(j)` and $\texttt{PKConstr}_{msk}(.)$. $\texttt{Sig}_{sk_j}(.)$ is queried once as `HORS` is a one-time signature.

• *Assumption 1*: `PUKO` has a secure enclave as described in the system model. The post-quantum `PUKO` is easily achieved by using a quantum-safe signature in the enclave.

IV. Proposed Schemes

We now present our proposed scheme `HASES`. We outline the main signature functionalities based on a high-level depiction in Fig.1 and a formal description in Alg.1.

In `HASES.Kg`(.), for a given set of users $\overrightarrow{ID}$, first generates the master key msk (Step 1) and then derives the initial private key $sk_1^i \in \overrightarrow{sk_1}$ for each $ID_i \in \overrightarrow{ID}$ (Step 2-3). It also accepts as input (J_1, J_2) as the number of precomputed and interleaved keys, respectively. According to J_1, it derives the precomputed keys $\overrightarrow{sk_p^i}$ from the initial sk_1^i (Step 4-6). Each private seed sk_1^i is sent to its corresponding signer (Step 7), while (msk, $\overrightarrow{sk_p^i}$) are solely placed on the secure enclave of `PUKO` (Step 1).

`HASES.PKConstr` harnesses `PUKO` to offer a trustworthy and flexible public key and identity management service for the verifiers, without interacting with signers. The verifier requests public key PK_j^i of ID_i for state j. `PUKO` first identify the corresponding precomputed key (Step 1) and then derive the j^{th} secret key (Step 2). Finally, it generates the public key PK_i^j and returns it to the verifier (Step 3-5). `PUKO` can derive any public key PK_j^i of any $ID_i \in \overrightarrow{ID}$ on demand, making `HASES` *fully scalable for millions of users with an adjustable* $\mathcal{O}(J_1)$ *cryptographic data storage*. Moreover, the verifier can obtain any public key(s) $1 \leq j \leq J$ from `PUKO` in batches before receiving signatures (`HASES.PKConstr` is independent from the signer). This permits verifiers to immediately verify signatures. Also, `PUKO` can either present on the verifier machine (e.g., laptop), or a nearby edge-cloud server, and therefore can effectively deliver public keys on demand.

As shown in Fig.1, the signing algorithm `HASES.Sig` relies on `HORS` signature generation but with a forward-secure pseudo-random number generation. Specifically, given sk_j^i, the signer computes subset resilient indexes and derives `HORS` one-time signature components (steps 1-4 in Alg. 1). The current private key is then updated and the previous key is deleted. The signing is near-optimal with respect to forward-secure hash-based signatures since it only requires one `HORS` call (around 10 hash calls) and a single hash for the update. The signer does not require any usage or interaction with secure hardware. The algorithm `HASES.Ver` also relies on `HORS`, except that it invokes `HASES.PKConstr` to obtain the public key (Phase (1-2) in Fig. 1). We remind that `HASES` distinct itself from symmetric-key approaches (e.g., MACs alone or use of secure-hardware to verify/compute MACs) via achieving public verifiability and non-repudiation. The verifier can check the received signatures with an offline interaction with `PUKO`, which only supplies public keys but does *not* perform verification itself.

Algorithm 1 HArdware-Supported Optimal Signature (`HASES`)

$(msk, \overrightarrow{sk_1}, \overrightarrow{sk_p}, I) \leftarrow \texttt{HASES.Kg}(1^\kappa, \overrightarrow{ID} = \{ID_i\}_{i=1}^N, J_1, J_2)$:

1: Generate the master key $msk \xleftarrow{\$} \{0,1\}^\kappa$ and set $I \leftarrow (l, k, t)$ as in Definition 2 and ($J \leftarrow J_1 \cdot J_2$). msk is provisioned to `PUKO`.
2: **for** $i = 1, \ldots, N$ **do**
3: $\quad sk_1^i \leftarrow H_0(msk \| ID_i)$
4: $\quad$ **for** $j_1 = 2, \ldots, J_1 - 1$ **do**
5: $\quad\quad sk_{j_1 \cdot J_2 + 1}^i \leftarrow H_1^{J_2}(sk_{(j_1-1)\cdot J_2+1}^i)$
6: $\quad \overrightarrow{sk_p^i} \leftarrow \{sk_{j_1 \cdot J_2+1}^i\}_{j_1=1}^{J_1-1}$
7: $\overrightarrow{sk_1} \leftarrow \{sk_1^i\}_{i=1}^N$, where sk_1^i is provisioned to ID_i.
8: $\overrightarrow{sk_p} \leftarrow \{\overrightarrow{sk_p^i}\}_{i=1}^N$, are provisioned to `PUKO`, as precomputed seeds.
9: **return** $(msk, \overrightarrow{sk_1}, \overrightarrow{sk_p}, I)$

$\sigma_j^i \leftarrow \texttt{HASES.Sig}(sk_j^i, M_j^i)$: Init $(j = 1)$ and require $j \leq J$

1: $h \leftarrow H_0(M_j^i)$ and split h into k substrings $\{h_i\}_{i=1}^k$ such that $|h_i| = \log_2 t$, where each h_i is interpreted as an integer x_i
2: **for** $\ell = 1, \ldots, k$ **do**
3: $\quad s_\ell \leftarrow H_1(sk_j^i \parallel x_\ell)$
4: $\sigma_j^i \leftarrow (s_1, s_2, \ldots, s_k, j, ID_i)$
5: Update $j \leftarrow j + 1$ and sk_j^i as $sk_{j+1}^i \leftarrow H_1(sk_j^i)$, and delete sk_j^i
6: **return** σ_j^i

$PK_j^i \leftarrow \texttt{HASES.PKConstr}(msk, \overrightarrow{sk_p}, ID_i, j)$: Require $ID_i \in \overrightarrow{ID}$ and $j \leq J$

1: $j_1 \leftarrow \lfloor \frac{j-1}{J_2} \rfloor$ and $j_2 \leftarrow (j-1) \mod J_2$
2: Load $sk_{j_1 \cdot J_2+1}^i$ from $\overrightarrow{sk_p^i}$ and derive $sk_j^i \leftarrow H_1^{j_2}(sk_{j_1 \cdot J_2+1}^i)$
3: **for** $\ell = 1, \ldots, t$ **do**
4: $\quad v_\ell \leftarrow H_2(s_\ell)$, where $s_\ell \leftarrow H_1(sk_j^i \| \ell)$
5: **return** $PK_j^i \leftarrow (v_1, v_2, \ldots, v_t)$

$b \leftarrow \texttt{HASES.Ver}(PK_j^i, M_j^i, \sigma_j^i)$: Step 1 can be run offline.

1: $PK_j^i \leftarrow \texttt{HASES.PKConstr}(ID_i, j, msk, \overrightarrow{sk_p})$
2: Execute Step (1) in `HASES.Sig`
3: **if** $H_0(s_\ell) = v_{x_\ell}, \forall \ell \in [1, k]$ and $1 \leq j \leq J$ **then return** $b_i^j = 1$ **else return** $b_i^j = 0$

V. Performance Analysis and Comparison

Parameters: We choose $I \leftarrow \{l = 256, t = 11966, k = 13\}$ for $\kappa = 128$. Our choice prioritizes optimal signer efficiency as it only requires $k = 13$ hash calls for signing. In `HASES`, the resource-limited signer is non-interactive and do not communicate the public keys, and therefore the parameter t does impact the signer performance. We set $N = 2^{20}$ signers and $J = 2^{20}$ signing capability (as in XMSS$^{\text{MT}}$ [18]). The users' identifiers $\overrightarrow{ID}$ are considered as the MAC addresses of signers. The hash functions $\{H_i\}_{i=0}^2$ are instantiated with SHA-256.

TABLE II
PERFORMANCE COMPARISON OF HASES VARIANTS AND ITS COUNTERPARTS ON A COMMODITY HARDWARE

Scheme	Signing Cycles	Private Key	Signature Size	Verification Cycles	Verifier Storage (GB)	Post-Quant Promise	Forward Security	Rejection/Gaussian (Sampling)	Deterministic Signing
ECDSA [2]	691,768	0.03	0.06	691,768	0.09	×	×	×	×
SchnorrQ [3]	32,016	0.03	0.06	60,453	0.09	×	×	×	×
SPHINCS+ [12]	112,080,752	0.1	35.66	10,249,926	35.71	✓	×	×	✓
BLISS-I [7]	870,770	2	5.6	748	12.6	✓	×	✓	×
Dilithium-II [4]	210,188	2.53	2.42	88,400	3.73	✓	×	✓	×
XMSS^{MT} [18]	3,011,720	3.11	2.61	3,944,441	3.36	✓	✓	×	✓
HASES	**11,101**	**0.03**	**0.42**	**13,041** $+ \Delta$	**6 MB**	✓	✓	×	✓

The private/public key and signature sizes are in KB. We benchmarked the XMSSMT-SHA20/2_256 variant. For SPHINCS+ parameters, $n = 16, h = 66, d = 22, b = 6, k = 33, w = 16$ and $\kappa = 128$. Δ denotes the PUKO delay for public key construction. It depends on the storage overhead of the pre-computed secret keys, per user, $\overrightarrow{sk_p^i}$. For example, if $J_1 = 1$ then $\Delta = 172.69\ ms$. If $J_1 = \sqrt{J} = 2^{10}$ and the public key was requested in *offline* mode, then $\Delta = 4.27\ ms$. Otherwise, if it was in *online* mode, $\Delta = 0.2\ ms$.

Configurations: We used a desktop equipped with an SGX-enabled Intel i9-9900K @ 3.6 GHz processor and 64 GB of RAM. We implemented HASES with OpenSSL[2] and the Intel SGX SDK v2.15.1 with C/C++ on the commodity hardware and PUKO, respectively. We utilize Intel Software Guard Extensions SSL[3] for cryptographic operations in the enclave. For the low-end devices at signer, we used an Arduino Mega 2560 board, which is based on low-power 8-bit ATmega2560 microcontroller and equipped with 256KB flash memory, 8KB SRAM and 4KB EEPROM, with clock frequency of 16 MHz. We implemented HASES using the Arduino Cryptography library[4].

Performance on Commodity Hardware: Table II shows the performance comparison of HASES with its counterparts.

• *Signature Generation and Size*: HASES offers a speedup of 271× over its only hash-based FS and PQ counterpart XMSS^{MT} [18]. It is also 19× faster than the most efficient (non-forward-secure) lattice-based NIST PQC Finalist Dilithium-II [4], while two magnitudes faster than stateless SPIHINCS+ [12]. The signature size of HASES is the smallest among all PQ variants. Overall, HASES offers the most efficient and compact PQ signature among all alternatives with forward-security.

• *Public Key Construction*: The delay that incur from the PUKO computation is parameterized by Δ in Table II. Verifiers can request the public keys in an *Offline* or *Online* mode. In offline mode, the ver time is independent of the PUKO delays and therefore achieving a significant efficiency over HASES's counterparts. We observe that storing only 3.2 KB of precomputed keys at PUKO, yields to a negligible delay $\Delta = 4\ ms$. As for online mode, verifiers can only request the k components of the user's public key to perform verification. Remarkably, the PUKO delay is only 30 μs for a storage overhead of 320 KB, which only consumes 0.25% of the overall SGX trusted memory. Such delay becomes bounded by HASES online ver time. Thus, it does not impact the overall verification time.

• *Verification*: HASES verification significantly outperform its counterparts. For example, HASES is 6.78× and 302.47× than Dilithium-II (non-forward-secure) and XMSS^{MT} (forward-secure), respectively, if the public key's retrieval is offline.

[2]https://github.com/openssl/openssl
[3]https://github.com/intel/intel-sgx-ssl
[4]https://rweather.github.io/arduinolibs/crypto.html

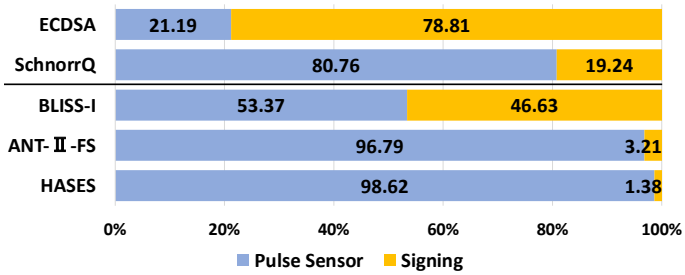

Fig. 2. Energy consumption of signing vs pulse sensor. ANT-II-FS assumes distributed non-colluding servers while others not.

• *Verifier Storage overhead*: By harnessing PUKO as a public key manager, HASES unleash a significant storage burden. Indeed, verifiers need to store only 6 MB as users' identities for a wide network that encapsulates *millions* of low-end devices. We note that the counterparts need to store the public keys + their certificates (e.g., ≈ 10 GB for 2^{20} users) which is not practical for non-resourceful verifiers (e.g., smartphones).

Performance on 8-bit AVR Microcontroller: Table I and Figure 2 compare the signer performance of HASES scheme and its counterparts on the 8-bit microcontroller.

To provide longer battery life, it is particularly important to reduce the energy consumption of cryptography in IoT. We compared the energy overhead with a pulse sensor that has a sampling time of 10 sec and a read time of 1 ms. HASES reduces the energy usage remarkably to 1.38% compared to that of the pulse sensor. This is 2.3× more energy-efficient than its closest counterpart ANT, which assumes multiple non-colluding servers to generate public keys. HASES is 34× more energy-efficient than BLISS-I, which is not FS and suffers from side-channel attacks due to Gaussian/rejection sampling while introducing a large signature size. HASES is even 16× faster than EC-based SchnorrQ that is not PQ or FS. Therefore, our experiments confirm that HASES is significantly more energy-efficient than their counterparts with advanced security features.

VI. SECURITY ANALYSIS

We formally prove the security of our scheme as follows.

Theorem 1 *If a polynomial-time adversary $\mathcal{A}$ can break the F-EU-CMA secure HASES in time t and after q_s signature and public key queries with a break-in query, then one can build*

polynomial-time algorithm $\mathcal{F}$ that breaks the EU-CMA secure HORS *in time t' and q'_s queries under Assumption 1.*

$$Adv^{FEU\text{-}CMA}_{HASES}(t, q_s, 1) \quad \leq \quad J \cdot Adv^{EU\text{-}CMA}_{HORS}(t', q'_s),$$

, where $q'_s = q_s + 1$ and $\mathcal{O}(t') = \mathcal{O}(t) + k \cdot H_0$.

Proof: $\mathcal{F}$ is given the challenge public key PK', where $(sk', PK', I) \leftarrow \texttt{HORS.Kg}(1^\kappa)$. $\mathcal{F}$ is given a HORS signing oracle, to which it can query a signature computed with sk'.

Algorithm $\mathcal{F}^{\texttt{HORS}_{sk'}(\cdot)}(PK')$: $\mathcal{F}$ is run per Definition 3:

• *Setup:* $\mathcal{A}$ selects an ID and gives it to $\mathcal{F}$. If $ID \notin \overrightarrow{ID}$, then $\mathcal{F}$ *aborts*, else continues the setup (index i is omitted for the brevity). $\mathcal{F}$ runs $\texttt{HASES.PKConstr}(msk, \overrightarrow{sk_p}, ID, j)$ to obtain sk_j (step 2) and PK_j (step 5), for $j = 1, \ldots, J$. $\mathcal{F}$ randomly selects a target forgery index $w \in [1, J]$ and overrides $PK_w = PK'$. $\mathcal{F}$ stores these values in the list $\mathcal{LS}$.

• *Queries:* $\mathcal{F}$ handles $\mathcal{A}$'s queries as follows.

(1) $\texttt{Sig}_{sk_j}(.)$: If $\mathcal{A}$ queries $\mathcal{F}$ on M_w then $\mathcal{F}$ returns $\sigma_j \leftarrow \texttt{HORS}_{sk'}(M_j)$ by querying HORS signing oracle. Otherwise, $\mathcal{F}$ returns $\sigma_j \leftarrow \texttt{HASES.Sig}(sk_j \in \mathcal{LS}, M_j)$.

(2) $\texttt{PKConstr}_{msk}(.)$: $\mathcal{F}$ returns $PK_j \in \mathcal{LS}$ to $\mathcal{A}$.

(3) $\texttt{Break-In(j)}$: If $1 < j \leq w$ then $\mathcal{F}$ *aborts*, else it returns $sk_j \in \mathcal{LS}$ and ends the experiment.

• *Forgery:* $\mathcal{A}$ outputs a forgery (M^*, σ^*) on PK^*. $\mathcal{F}$ wins the experiments if $\mathcal{A}$ wins the experiments by producing a valid forgery on PK_w. That is, $\mathcal{F}$ returns 1 if $PK^* = PK_w$ and $1 = \texttt{HORS.Ver}(PK^*, M^*, \sigma^*)$ and M^* was not queried to $\texttt{Sig}_{sk_j}(.)$. Otherwise, $\mathcal{F}$ returns 0 and *aborts*.

• *Success Probability and Indistinguishability*: Assume that $\mathcal{A}$ wins *FEU-CMA* experiment against HASES with the probability $Adv^{FEU\text{-}CMA}_{\texttt{HASES}}(t, q_s, 1)$. $\mathcal{F}$ wins the *EU-CMA* experiments against HORS if and only if $\mathcal{A}$ produces a forgery on the challenge public key PK_w and does not abort during the experiment. Since $w \in [1, J]$ is selected randomly, the success probability of $\mathcal{A}$ can be bounded that of $\mathcal{F}$ as

$$Adv^{FEU\text{-}CMA}_{\texttt{HASES}}(t, q_s, 1) \quad \leq \quad J \cdot Adv^{EU\text{-}CMA}_{HORS}(t', q'_s)$$

$\mathcal{A}$'s real $\mathcal{A}_\mathcal{R}$ and simulated $\mathcal{A}_\mathcal{S}$ views are indistinguishable. $\mathcal{F}$'s transcripts in $\mathcal{LS}$ are identical to the real execution except that PK_w is replaced with HORS public key PK'. Hence, $sk_w = sk'$ is not the part of hash chain generated from sk_1 via H_1. As in HORS, H_1 is a random oracle. Therefore, w^{th} element of $\mathcal{LS}$ in $\mathcal{A}_\mathcal{R}$ and $\mathcal{A}_\mathcal{S}$ have identical distributions. HORS is post-quantum secure, and so as HASES. ■

VII. Conclusion

We created a novel post-quantum forward-secure signature HASES that achieves the highest signer efficiency among its counterparts. It is at least two and one magnitudes faster than the best forward-secure hash-based signature XMSS$^{\text{MT}}$ and non-forward-secure lattice-based signature BLISS-I, respectively. HASES is also more than twice efficient over its fastest lattice-based alternative ANT without requiring non-colluding servers or secure enclaves on signers. HASES have the smallest signature and private key sizes among its alternatives. We formally proved that HASES is secure and open-sourced its implementation to enable public testing and broader adaptation.

Acknowledgment

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

References

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM Review*, vol. 41, no. 2, pp. 303–332, 1999.

[2] *ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*, American Bankers Association, 1999.

[3] C. Costello and P. Longa, "SchnorrQ: Schnorr signatures on FourQ," Tech. Rep., 2016.

[4] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-dilithium: A lattice-based digital signature scheme," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 238–268, 2018.

[5] L. Tawalbeh, F. Muheidat, M. Tawalbeh, and M. Quwaider, "IoT Privacy and security: Challenges and solutions," *Applied Sciences*, vol. 10, no. 12, p. 4102, 2020.

[6] T. Malkin, D. Micciancio, and S. K. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods," in *Proc. of the 21th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '02)*, 2002, pp. 400–417.

[7] T. Pöppelmann, T. Oder, and T. Güneysu, "High-performance ideal lattice-based cryptography on 8-bit ATxmega microcontrollers," in *Int. conf. on cryptology and information security in Latin America*, 2015, pp. 346–365.

[8] R. Behnia and A. A. Yavuz, "Towards practical post-quantum signatures for resource-limited internet of things," in *Annual Computer Security Applications Conference*, 2021, pp. 119–130.

[9] D. Boneh, S. Eskandarian, and B. Fisch, "Post-quantum EPID signatures from symmetric primitives," in *CT-RSA Conference*, 2019, pp. 251–271.

[10] W. Ouyang, Q. Wang, W. Wang, J. Lin, and Y. He, "SCB: Flexible and Efficient Asymmetric Computations Utilizing Symmetric Cryptosystems Implemented with Intel SGX," in *2021 IEEE International Performance, Computing, and Communications Conference (IPCCC)*, 2021, pp. 1–8.

[11] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, C. Miller, D. Moody, R. Peralta *et al.*, "Status report on the third round of the nist post-quantum cryptography standardization process," *National Institute of Standards and Technology, Gaithersburg*, 2022.

[12] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS+ signature framework," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 2129–2146.

[13] D. A. Cooper, D. C. Apon, Q. H. Dang, M. S. Davidson, M. J. Dworkin, C. A. Miller *et al.*, "Recommendation for stateful hash-based signature schemes," *NIST Special Publication*, vol. 800, p. 208, 2020.

[14] Intel, "Intel® Software Guard Extensions," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, accessed: Oct 6, 2022.

[15] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: functional encryption using Intel SGX," in *Proc of the 2017 ACM SIGSAC Conf on Computer and Communications Security*, 2017, pp. 765–782.

[16] S. Contiu, R. Pires, S. Vaucher, M. Pasin, P. Felber, and L. Réveillère, "IBBE-SGX: Cryptographic group access control using trusted execution environments," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 207–218.

[17] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," in *Information Security and Privacy: 7th Australasian Conference*, July 2002, pp. 144–153.

[18] A. Hülsing, L. Rausch, and J. Buchmann, "Optimal parameters for XMSS MT," in *International conference on availability, reliability, and security*. Springer, 2013, pp. 194–208.

[19] X. Lou, T. Zhang, J. Jiang, and Y. Zhang, "A survey of microarchitectural side-channel vulnerabilities, attacks, and defenses in cryptography," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.