Compact and Resilient Cryptographic Tools for Digital Forensics

Efe U. A. Seyitoglu
University of South Florida
Tampa, Florida, USA
efe3@mail.usf.edu

Attila A. Yavuz
University of South Florida
Tampa, Florida, USA
attilaayavuz@usf.edu

Muslum Ozgur Ozmen*
Purdue University
West Lafayette, Indiana, USA
mozmen@purdue.edu

***Abstract*—Audit logs play a crucial role in the security of computer systems and are targeted by the attackers due to their forensic value. Digital signatures are essential tools to ensure the authentication/integrity of logs with public verifiability and non-repudiation. Especially, forward-secure and aggregate signatures (FAS) offer compromise-resiliency and append-only features such that an active attacker compromising a computer cannot tamper or selectively delete the logs collected before the breach. Despite their high-security, existing FAS schemes can only sign a small pre-defined number (K) of logs, and their key-size/computation overhead grows linearly with K. These limitations prevent a practical adoption of FAS schemes for digital forensics.**

In this paper, we created new signatures named *COmpact and REsilient (*CORE*)* schemes, which are (to the best of our knowledge) the first FAS that can sign (practically) unbounded number of messages with only a sub-linear growth in the key-size/computation overhead. Central to CORE is the creation of a novel K-time signature $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ that has a small-constant key generation overhead and public key size. We then develop CORE−MMM that harnesses $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ via forward-secure transformations. We showed that CORE−MMM significantly outperforms its alternatives for essential metrics. For instance, CORE−MMM provides more than two and one magnitudes faster key updates and smaller signatures, respectively, with smaller private keys. CORE−MMM also offers extra efficiency when the same messages are signed with evolving keys. We formally prove that CORE schemes are secure. Our analysis indicates that CORE schemes are ideal tools to enhance the trustworthiness of digital forensic applications.

***Index Terms*—Authentication, digital signatures, digital forensics, audit logs, forward-security, aggregation.**

I. Introduction

Audit logs are one of the most prevalent forensic analysis methodologies in computer systems [1]. Logs provide critical information on the past states of a machine such as system events, security incidents and failures. They have a central role in law investigations and legal dispute resolution. Due to their forensic value, logs are among the primary targets of the attackers. For instance, an attacker breaching into a computer can erase its own traces and/or implicate innocent users by tampering audit logs. It is of vital importance to ensure the tamper-resistance of audit logs in computer systems.

Cryptographic digital forensic mechanisms form the foundation of tamper-resistant and trustworthy audit logging [2], [3], [4]. They can provide essential services such as authentication and integrity while also offering some advanced security features (e.g., forward-security). Specifically, an ideal cryptographic digital forensic mechanism should offer (at minimum) the following properties [4], [5], [6], [7], [8]:

(i) Public verifiability and non-repudiation: They permit *any* entity to verify the trustworthiness of logs via public keys. Hence, real-life use-cases that involve a public audit or legal dispute resolution (e.g., financial, healthcare) strictly need these features, which are mainly achieved via digital signatures [9]. *(ii) Compromise-resiliency*: The digital signature should offer some security even in the presence of active attackers who may breach into the computer system (e.g., malware, physical intervention). Especially, a forward-secure signature [10] ensures that despite the current key is compromised, data items signed before the attack remain unforgeable. *(iii) Aggregation and Holistic Integrity*: The cryptographic storage and transmission overhead of tamper-resistant mechanisms can be substantial due to the high volume of logs collected from a large number of users. The signature aggregation [11] in conjunction with forward-security can compress the signatures thereby mitigating this overhead. It can also not only guarantee the integrity of individual logs, but the entire log trail to prevent re-ordering and selective deletion [6]. *(iv) High Efficiency*: The cryptographic mechanism should offer fast verification to audit sheer amount entires, an efficient signing and constant-size private/public key sizes to minimize overall storage burden.

There is a critical research gap in the state-of-art towards achieving all these desirable properties simultaneously.

II. Related Work and the Limitations of the State-of-the-art

There are two main lines of cryptographic digital forensic (also referred to as secure audit logging) techniques.

• *Symmetric Cryptography based Primitives*: Many symmetric key based audit logging techniques [3], [12], [13] rely on Message Authentication Codes [9]. Despite their efficiency, they cannot achieve non-repudiation and public verifiability, since the verifier(s) shares the same key with signer(s). As discussed, these properties are vital for digital forensic applications [1]. The rest of this paper focuses on the public key based secure auditing techniques.

*Work done when the third author was employed at University of South Florida.

• *Public Key Cryptography (PKC) based Primitives*: Digital signatures offer public verifiability and non-repudiation, but the traditional signatures [14] do not offer advanced features such as forward security [10], aggregation [11] and high performance. Later, *forward-secure and aggregate signatures (FAS)* for secure audit logging have been developed with several performance and security trade-offs. FssAgg [3], [5], [6], [15] were the first *(FAS)* schemes, but their variants with sub-linear signature and public key sizes were flawed [16]. FssAgg-BLS [3], [6] is the only secure version, which is a K-time signature scheme with $O(K)$ public key size. It also requires a cryptographic pairing per message verification, and therefore is very costly. Later, BAF schemes [4], [7], [17] have been proposed, which are currently the most computationally efficient alternatives but still require $O(K)$ public key size. A recent FAS in [18] pre-defines the number of messages to be signed (i.e., K) and is extremely computationally costly since the signature generation/verification requires exponentiations over a large module (e.g. $|N| = 3072$-bit) per message with a size of ≈ 100 KB. Finally, forward-secure schemes with selective verification capabilities (e.g., order-free [2], verifiable excerpts [1]) can complement the aforementioned secure logging schemes.

• *Research Challenges*: There are two obstacles towards a wide adoption of existing *FAS*-based digital forensic tools in real-life applications: (i) The total number of messages are pre-defined (i.e., K-time signatures). This is a severe limitation, since it might not be possible to know beforehand how many log entries a computer system may generate in its life-time. Moreover, once all K signatures are consumed, the whole system must be re-initialized, which might be impossible or extremely costly for some real-life applications. For instance, consider a router or security camera generating a new signature on every new measurement that they capture. Such devices may generate vast amount of log entries in relatively short period of time, thereby quickly depleting K-time signatures. This forces secure audit tool to be re-initialized frequently, which introduces heavy transmission/computation and security vulnerabilities to the system. (ii) The total cryptographic storage overhead, which is dominated by the size of signature plus the public key, is always $O(K)$. For example, in BAF and FssAgg-BLS, despite the aggregate signature has $O(1)$ size for K entries, the total cryptographic overhead remains $O(K)$ at the verifier (e.g., auditor, digital archive). *There is a significant need for new cryptographic schemes that can address these limitations while preserving the desirable properties of forward-secure and aggregate signatures.*

III. Our Contributions

We created new forward-secure and aggregate signatures named as *COmpact and REsilient (*CORE*)* schemes, which can address the aforementioned limitations towards enabling fully practical cryptographic digital forensic tools.

Main Idea: We observed that achieving a practical unbounded forward-secure signature requires: (i) The signer must generate a fresh private/public key pair per update, and therefore the key generation overhead must be minimized. This phase is extremely costly (i.e., $O(K)$) in previous constructions. (ii) New public keys must be relayed to the verifier for each update, and therefore both the signature and public key sizes must be minimized. Based on these observations, we first created a new K-time *FAS* scheme $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ that achieves efficient key generation time and small (and constant) public key size while preserving the signing and verification efficiency. We then develop CORE−MMM, which is (to the best of our knowledge) the first practical *FAS* scheme with unbounded signing capability. We elaborate the desirable properties of CORE schemes as below:

• *Compact Public Key, Signature and Efficient Key Generation*: $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ has several new design features that makes it an ideal building block for unbounded signing with forward-security and aggregation: (i) $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ derives a pair of accumulated public keys, which can batch verify K-signatures, from one-way private key sequences. The size of this K-time public key is only 64 byte and can be generated with two exponentiations (i.e., elliptic curve scalar multiplications), as opposed to $O(K)$ size and computation overhead of its counterparts. (ii) $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ not only provides full K-time signature aggregation but also conditional public key aggregation if the same message is signed consecutively with evolving private keys. This offers extra compactness and verification advantages for use-cases when the state (message) transitions are infrequent. (iii) Unlike previous alternatives that must wait all K-items to be signed, $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ permits "sealing" feature, which can fast-forward signing with only a small-constant overhead, in case an early verification is needed.

• *Practically Unbounded Signing with High Efficiency*: We developed CORE−MMM, which is (to the best of our knowledge) the first *FAS* scheme with unbounded signing, by harnessing our $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ via MMM with optimizations. We also analyzed unbounded signing transformations of existing alternatives, and showed that CORE−MMM offers a vastly superior performance over them. For instance, CORE−MMM offers $337\times$ faster update (w.r.t., SchnorrQ−MMM), $7.5\times$ smaller signature (w.r.t., BAF−MMM), with 80% smaller private key, compared with the most efficient alternative for each category. This is achieved with an equal public key size and competitive signing/verification speed. Moreover, the performance of CORE−MMM is further pushed to the edge once the application signs the same measurements consecutively. We elaborate on some relevant use-cases and give a comprehensive performance analysis in Section VIII.

• *Provable Security and Implementation*: We formally prove that CORE schemes are *Forward-secure Aggregate Existentially Unforgeable against Chosen Message Attack (FAEU-CMA)* [4] (in random oracle model), and present a full implementation of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ on commodity hardware.

• *Potential Use-cases*: CORE is ideal for digital forensic applications (e.g., [5], [12], [13], [19]) that require store-and-forward integrity and authentication (e.g., [4], [6], [16]). For example, Internet of Things (IoT) devices (e.g., security camera, sensors) collect critical measurements, digitally sign,

and store them, until the receiver requests those for the verification. Similarly, some system monitoring applications (e.g., hypervisors, financial logs) collect, secure and store forensic data until it is transferred for an analysis. In this process, an active adversary might compromise the device and then forge the previous signatures by extracting the private key.

CORE−MMM prevents forgery and selective deletion of the pre-compromise data by offering forward-security and aggregation (i.e., append-only) without limiting the number of items to be secured. CORE−MMM outperforms its counterparts in many metrics, and shows even a better performance when the system state remains unchanged for some time intervals. For instance, the number of transactions on a personal bank account per day is limited, and there are multiple and contagious forward-secure signing periods, in which the signed data remains the same. Similarly, a camera/sensor at a remote street may capture only a few movements in a given hour, while readings remain the same between these rare state changes.

IV. Preliminaries

Operators $||$ and $|x|=\log_2 x$ denote the concatenation and the bit length of variable x, respectively. $x \xleftarrow{\$} \mathcal{S}$ means variable x is randomly and uniformly selected from set $\mathcal{S}$. We denote $\{0,1\}^*$ by the set of binary strings of any finite length. The set of items q_i for $i=0,\ldots,l-1$ is denoted by $\{q_i\}_{i=0}^{l-1}$. $H:\{0,1\}^* \rightarrow \{0,1\}^d$ (d is a small integer), $H_1:\mathcal{Z}_q^* \rightarrow \mathcal{Z}_q^*$, $H_2:\{0,1\}^* \rightarrow \mathcal{Z}_q^*$ and $H_3:\{0,1\}^\kappa \rightarrow \{0,1\}^\kappa$ are cryptographic hash functions treated as random oracles [20]. $G(.)$ is a length-doubling pseudo-random number generator. q and $p>q$ are large primes such that $q|(p-1)$. α is a generator of the subgroup G of order q in $\mathbb{Z}_p^*$.

Standard and K-time Digital Signatures: We use a standard digital signature [9] as in Definition 1. We also use Efficient and Tiny Authentication *(ETA)* [21] (see Algorithm 1).

Definition 1. A signature scheme consists of three algorithms $\mathsf{SGN}=(\mathsf{Kg},\mathsf{Sig},\mathsf{Ver})$ defined as follows.

- $(sk, PK) \leftarrow \mathsf{SGN.Kg}(1^\kappa)$: Given the security parameter κ, it outputs the private and public key pair (sk, PK).
- $\sigma \leftarrow \mathsf{SGN.Sig}(m, sk)$: Given the message m and the signer's private key sk, it outputs the signature σ.
- $b \leftarrow \mathsf{SGN.Ver}(m, \sigma, PK)$: Given a message-signature pair (m,σ), and public key PK, outputs $b \leftarrow \{0,1\}$

Algorithm 1 Efficient and Tiny Authentication Scheme [21]

$(sk_0, PK) \leftarrow \mathsf{ETA.Kg}(1^\kappa, K)$:
1: $r_0 \xleftarrow{\$} \mathbb{Z}_q^*, y \xleftarrow{\$} \mathbb{Z}_q^*, Y \leftarrow \alpha^y \bmod q$
2: **for** $j=0,\ldots,K-1$ **do**
3: $R_j \leftarrow \alpha^{r_j} \bmod p$, $r_{j+1} \leftarrow H(r_j)$, $v_j \leftarrow H(R_j)$
4: **return** $sk_0 \leftarrow (y, r_0)$ and $PK \leftarrow (Y, \overline{v}=v_0,\ldots,v_{K-1})$

$\sigma_j \leftarrow \mathsf{ETA.Sig}(sk_j, M_j)$: **if** $j \geq K$ **return** $\perp$ **else**
1: $x_j \xleftarrow{\$} \{0,1\}^\kappa$, $e_j \leftarrow H(M_j||j||x_j)$, $s_j \leftarrow r_j - e_j \cdot y \bmod q$, $r_{j+1} \leftarrow H(r_j)$ and **return** $\sigma_j=(s_j, x_j, j)$

$b \leftarrow \mathsf{ETA.Ver}(PK, M_j, \sigma_j)$: **if** $j \geq K$ then **return** $b=0$
1: **if** $v_j = H(Y^{H(M_j||j||x_j)} \cdot \alpha^{s_j})$ **return** $b=1$ **else return** $b=0$

*Malkin, Micciancio and Miner (*MMM*) Signature Scheme [22]*: This is a generic algorithm that can transform a standard digital signature SGN into a forward-secure signature [22].

MMM is composed of the sum (referred to as $\sum$) and product composition algorithms. We give the $\sum$ composition in Algorithm 2.

Algorithm 2 Sum Composition ($\sum$) Algorithms in MMM [22]

$(sk, PK) \leftarrow \sum.\mathsf{Kg}(1^\kappa, r, levels, i, \mathsf{SGN})$:
1: $(r_0, r_1) \leftarrow G(r)$
2: **if** $levels = 0$ **then**
3: $(sk_0, pk_0) \leftarrow \mathsf{SGN.Kg}(1^\kappa, r_0)$
4: $(sk_1, pk_1) \leftarrow \mathsf{SGN.Kg}(1^\kappa, r_1)$
5: $PK \leftarrow H(pk_0||pk_1)$, $sk \leftarrow (sk_0, r_1, pk_0, pk_1)$
6: **return** (sk, PK)
7: $(sk_0, pk_0) \leftarrow \sum.\mathsf{Kg}(1^\kappa, r_0, levels-1, i+1, \mathsf{SGN})$
8: $(sk_1, pk_1) \leftarrow \sum.\mathsf{Kg}(1^\kappa, r_1, levels-1, i+1, \mathsf{SGN})$
9: $PK \leftarrow H(pk_0||pk_1)$, $sk \leftarrow (sk_0, r_1, pk_0, pk_1)$
10: **return** (sk, PK)

$(\sigma, \bar{t}) \leftarrow \sum.\mathsf{Sig}(\bar{t}, sk=\langle sk', r_1, pk_0, pk_1\rangle, M, levels, i, \mathsf{T}, \mathsf{SGN})$:
1: **if** $levels = 0$ **then**
2: **if** $\bar{t} < T/2$ **then** $\overline{t'} \leftarrow \bar{t}$ **else** $\overline{t'} \leftarrow \bar{t} - T/2$
3: $\sigma' \leftarrow \mathsf{SGN.Sig}(sk', M)$, $\sigma \leftarrow \langle \sigma', pk_0, pk_1\rangle$
4: **return** $(\sigma', \overline{t'})$
5: **else**
6: **if** $\bar{t} < T/2$ **then**
7: $(\sigma', \bar{t}) \leftarrow \sum.\mathsf{Sig}(\bar{t}, sk', \bar{t}, M, levels-1, i+1, \mathsf{SGN})$
8: **else** $\bar{t} \leftarrow \bar{t} - T/2$
9: $(\sigma', \bar{t}) \leftarrow \sum.\mathsf{Sig}(\overline{t'}, sk', \bar{t}, M, levels-1, i+1, \mathsf{SGN})$
10: $\sigma \leftarrow \langle \sigma', pk_0, pk_1\rangle$ and **return** $(\sigma, \bar{t})$

$\sum.\mathsf{Up}(sk=\langle sk', r_1, pk_0, pk_1\rangle, \bar{t}, levels, i, \mathsf{SGN})$:
1: **if** $\bar{t} + 1 < T$ **then** $sk' \leftarrow \mathsf{SGN.UP}(sk')$
2: **else if** $(\bar{t}+1 = T)$ **then**
3: $(sk', pk') \leftarrow \mathsf{SGN.Kg}(r_1)$, $r_1 \leftarrow 0$, **delete** pk' **else** $sk'' \leftarrow \mathsf{SGN.UP}(sk')$
4: **if** $\bar{t} < T/2$ **then**
5: $\sum.\mathsf{Up}(sk'', r_1, pk_0, pk_1, \bar{t}, levels-1, i+1, \mathsf{SGN})$
6: **else** $\sum.\mathsf{Up}(sk'', r_1, pk_0, pk_1, \bar{t}-T/2, levels-1, i+1, \mathsf{SGN})$

$b \leftarrow \sum.\mathsf{Ver}(pk, M, \sigma=\langle \sigma', pk_0, pk_1\rangle, \bar{t}, levels, \mathrm{i}, \mathsf{SGN})$:
1: **if** $H(pk_0||pk_1) \neq pk$ **then return** $b=0$
2: **if** $levels = 0$ **then**
3: **if** $\bar{t} < T/2$ **then** $\mathsf{SGN.Ver}(pk_0, M, \sigma)$
4: **else** $\mathsf{SGN.Ver}(pk_1, M, \sigma)$
5: **else**
6: **if** $\bar{t} < T/2$ **then**
7: **return** $\sum.\mathsf{Ver}(pk_0, M, \sigma=\langle \sigma', pk_0, pk_1\rangle, \bar{t},$
8: $levels-1, i+1, \mathsf{SGN})$
9: **else return** $\sum.\mathsf{Ver}(pk_1, M, \sigma=\langle \sigma', pk_0, pk_1\rangle,$
$\bar{t}-T/2, levels-1, i+1, \mathsf{SGN})$

Using Algorithm 2, we instantiated our CORE−MMM (Algorithm 4) by leveraging our $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$(Algorithm 3) via product composition and optimized MMM strategies. In our instantiation of the $\sum$ composition, we only use one type of standard signature SGN. Hence, given a SGN with $T/2$ time periods (i.e., signing capability), the $\sum$ composition produces a signature SGN′ with $T = T/2 + T/2$ time periods. In product composition, given two SGN with $T/2$ time periods, it produces a signature SGN′ with $T = (T/2)^2$ time periods. The overall MMM construction has a single upper tree and lower trees, which are created with the iterative execution of sum composition. Lower trees are generated on the run with increasing levels of height. This minimizes the key generation/update cost and makes them depend on "messages signed so far". Thus, the lower trees are created as needed on the run. MMM increases the levels in each iteration to sign more messages with a slight increase in cost (generation of a higher tree). Overall, in MMM, the costs are either logarithmic or constant with respect to $t < T$ (i.e., the total number of messages signed so far) and the size of the PK is a small-constant.

V. System, Threat and Security Models

System and Threat Model: Our system model relies on the standard digital signature based broadcast authentication with a store-and-forward model including two entities: (i) Signers who are honest until they are compromised by the adversary, (ii) verifiers who can be any entity. In the store-and-forward model, the signer collects, signs and then stores K messages and their corresponding signatures until they are requested for a verification. These messages and signatures later are uploaded to a verifier for a public verification. Our threat model assumes an active adversary $\mathcal{A}$ who can compromise the secret key, interact and observe signatures before its compromise. After the signer and secret key are compromised by $\mathcal{A}$, they may attempt to forge signatures, or modify/re-order/delete the signatures issued before the compromise.

Security Model: We define our formal security model that correspond to our threat and system model as follows.

CORE schemes rely on the intractability of *Discrete Logarithm Problem (DLP)* [9], which is defined below.

Definition 2. *A cyclic group G of order prime q and a generator α of G, let algorithm $\mathcal{A}$ return an element of Z_q^*:*
Experiment $Expt^{DL}_{G,\alpha}(\mathcal{A})$
$y \xleftarrow{\$} \mathbb{Z}_q^*$, $Y \leftarrow \alpha^y \bmod q$, $y' \leftarrow \mathcal{A}(Y)$,
If $\alpha^{y'} \bmod p = Y$, *return* 1, *else, return* 0.

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

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

The DL-advantage of (G, α) *in this experiment is defined as*

$$Adv^{DL}_{G,\alpha}(t) = \max_{\mathcal{A}}\{Adv^{DL}_{G,\alpha}(\mathcal{A})\},$$

where the max is over all $\mathcal{A}$ having time t.

$\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ is a K-time single-signer forward-secure and aggregate signature (FAS) [4]. We will not repeat FAS interface, as it is identical to that of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$(see Algorithm 3), wherein the key updates are subsumed in the signature generation. A FAS scheme is proven to be *Forward-secure Aggregate Existentially Unforgeable against Chosen Message Attack (FAEU-CMA)* [4] based on the experiment defined in Definition 3. $\mathcal{A}$ is given with three oracles:

$RO(.)$: A random oracle from which $\mathcal{A}$ can request the hash of any message m of their choice up to K' messages. Our hash functions are modeled as a random oracle [20] via $RO(.)$.

$FAS.Sig_{sk}(.)$: A $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$*signing oracle* from which $\mathcal{A}$ can query up to K messages $\overrightarrow{M}$ of their choice adaptively, until they decide to "break-in". For each query on $m_j, 1 \leq j \leq K$, $FAS.Sig_{sk}(.)$ updates sk_j to sk_{j+1}(deletes sk_j), and returns an aggregate signature $\sigma_{1,j}$ on $\overrightarrow{M} = \{m_j\}_{j=1}^{K}$ also including a state st and a partial aggregated public key $\overrightarrow{PK}$. As in [4], we consider a batch verification of multiple messages $\overrightarrow{M}'$, in which $\sigma_{1,K}$ is verified only under the aggregate public key PK on $\{sk_j\}_{j=1}^{K}$.

Break-in: If $\mathcal{A}$ queried $l < K$ messages to $FAS.Sig_{sk}(.)$, then *Break-in* oracle returns $(l+1)$-th private key to $\mathcal{A}$, else it rejects the query (all sk were used).

Definition 3. $FAEU\text{-}CMA$ experiment *is defined as follows:*
Experiment $Expt^{FAEU\text{-}CMA}_{FAS}(\mathcal{A})$
$(sk_1, PK) \leftarrow FAS.Kg(1^\kappa, K)$,
$(st^*, \sigma^*_{1,l}) \leftarrow \mathcal{A}^{RO(.), FAS.Sig_{sk}(.), Break\text{-}in}(PK)$,
If $FAS.Ver(PK, st^* = \langle \overrightarrow{M}^*, \overrightarrow{PK}^* \rangle, \sigma^*_{1,l}) = 1$ *and* $\exists z \in \{1, \ldots, l\} : \overrightarrow{M}^*[z] \notin \overrightarrow{M}$ *holds, then return* 1, *else return* 0.
FAEU-CMA-advantage of $\mathcal{A}$ *is defined as*

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

FAEU-CMA-advantage of FAS *is defined as*

$$Adv^{FAEU\text{-}CMA}_{FAS}(t, L', L) = \max_{\mathcal{A}}\{Adv^{FAEU\text{-}CMA}_{FAS}(\mathcal{A})\}$$

where the max is over all $\mathcal{A}$ with time t, making at most K' queries to both $RO(.)$ and $FAS.Sig_{sk}(.)$.

VI. Proposed Schemes

We first give our K-time scheme $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ and then its extension CORE−MMM with (practically) unbounded signing.

A. Description of the $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ Scheme

In key generation algorithm of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$.Kg, we create a pair of accumulated (small constant-size) public keys with only two exponentiations. This pair can verify an aggregate signature computed via K distinct private keys. Specifically, we generate two hash chains from (y_1, r_1) of size K, and then compute accumulated private keys $(\bar{r}, \bar{y})$ from them (step 2-3). We then embed them into $PK \leftarrow (R' = H_1(R_{1,K}), Y' = H_1(Y_{1,K})$, and set $sk = (y_1, r_1, x_1)$ (step 4-5).

The signing algorithm $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$.Sig offers K-time forward-secure signature aggregation and conditional public key aggregation (when the message to be signed does not change for consecutive intervals). If the message m_j is

different than the previous M_i, we compute an individual signature s_j and add/update state st_j including individual (Y_j) partial aggregate public key $\overrightarrow{PK}_i$ with corresponding messages $\overrightarrow{M}_i$ (steps 13-16). Otherwise, we can also aggregate Y_j into previous one (Step 19). We then aggregate s_j into previous signature (step 20) and update private/public keys (step 21-22), until final aggregate signature $\sigma_{1,K}$ computed. In steps 1-12, we capture a special condition that we refer to as "sealing". Specifically, the accumulated PK requires a batch verification of all K messages (as in previous K-time FAS schemes (e.g., [4], [6])) as it contains all K private keys. If a verification is requested before all private keys are used, we invoke "seal" mechanisms that fast-forwards the signature generation, in the line of key generation, by computing a full aggregate signature and public key on a special command $M_i = seal||timestamp||j||K$ that locks the time, order (j) and sealing of data items accumulated so far. The sealing requires only one exponentiation and therefore is efficient.

The verification algorithm $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Ver}$ takes PK, state st including messages $\overrightarrow{M}$, their corresponding partial aggregate public key $\overrightarrow{PK}$ and final counter values (i, j, j') with aggregate signature $\sigma_{1,j} = (s_{1,j}, x_j)$. We first check counters (e.g., number of items, in step 1) and then verify if the partial aggregate public keys matches with the accumulated PK (step 2). If $\overrightarrow{PK}$ are verified, then we use them and $\sigma_{1,j}$ to verify the messages $\overrightarrow{M}$ (step 3). Notice that our verification gains speed advantage from partial public key aggregation (if it applies).

Algorithm 3 $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ Algorithm

$(sk_1, PK) \leftarrow \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Kg}(1^\kappa, K)$:

1: $(y_1, r_1) \xleftarrow{\$} \mathbb{Z}_q^*$, $x_1 \xleftarrow{\$} \{0,1\}^\kappa$,$sk_1 \leftarrow (y_1, r_1, x_1)$
2: $\bar{r} \leftarrow \sum_{i=1}^{K} r_i \bmod p$, where $r_{i+1} \leftarrow H_1(r_i)$
3: $\bar{y} \leftarrow \sum_{i=1}^{K} y_i \bmod p$, where $y_{i+1} \leftarrow H_1(y_i)$
4: $Y_{1,K} \leftarrow \alpha^{\bar{y}} \bmod p$, $R_{1,K} \leftarrow \alpha^{\bar{r}} \bmod p$
5: $PK \leftarrow (R' = H_1(R_{1,K}), Y' = H_1(Y_{1,K}))$

$(st_j, \sigma_{1,j}) \leftarrow \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Sig}(sk_j, \sigma_{1,j-1}, m_j, st_j, c)$: Let $(i = j = j' = 1, c = 0)$, $m_1 = M_1$, $Y_1^{0,1} = 1$, $s_{1,0} = 0$ and $(\overrightarrow{M}, \overrightarrow{PK}, st_1)$ are empty.

1: **if** $c = 1$ **then**
2: $x_{i+1} \leftarrow H_3(x_i)$, $i \leftarrow i + 1$
3: $M_i \leftarrow$ "$seal||timestamp||j||K$"
4: $y_{j,K} \leftarrow \sum_{l=j}^{K} H_1(y_l) \bmod q$, where $y_{l+1} \leftarrow H_1(y_l)$
5: $r_{j,K} \leftarrow \sum_{l=j}^{K} H_1(r_l) \bmod q$, where $r_{l+1} \leftarrow H_1(r_l)$
6: $s_{j,K} \leftarrow r_{j,K} - H_1(M_i||j+1||x_i) \cdot y_{j,K} \bmod q$
7: $s_{1,K} \leftarrow s_{1,j-1} + s_{j,K} \bmod q$
8: $Y_i^{j,K} \leftarrow \alpha^{y_{j,K}} \bmod p$
9: Delete $(y_j, .., y_K, y_{j,K}, r_j, .., r_K, r_{j,K}, x_i, x_{i+1}, s_{j,K}, s_{1,j-1})$, and add $(Y_i^{j-1,j'}, Y_i^{j,K})$ and $(M_i^{j-1,j'}, M_i^{j,K} = M_i||j'+1)$ to $\overrightarrow{PK}_i$ and $\overrightarrow{M}_i$, respectively.
10: $j \leftarrow K + 1$ and update $st_j = (\langle i, j-1, j' \rangle, \overrightarrow{M}, \overrightarrow{PK}))$
11: $\sigma_{1,K} \leftarrow (s_{1,K}, x_1)$
12: **return** the final state $(sk_j = \perp, \langle st_j, \sigma_{1,K} \rangle)$ and **exit**
13: **else if** $m_j \neq M_i$ **then**
14: Add $Y_i^{j-1,j'}$ and $M_i^{j-1,j'} = M_i||j'$ to $\overrightarrow{PK}_i$ and $\overrightarrow{M}_i$, respectively, and update $st_{j+1} = (\langle i, j-1, j' \rangle, \overrightarrow{M}, \overrightarrow{PK})$
15: $j' \leftarrow j$, $x_{i+1} \leftarrow H(x_i)$, $i \leftarrow i + 1$, $M_i \leftarrow m_j$
16: $s_j \leftarrow r_j - H_1(M_i||j'||x_i) \cdot y_j \bmod q$, $Y_i^{j,j'} \leftarrow Y_j$
17: **else**
18: $s_j \leftarrow r_j - H_1(M_i||j'||x_i) \cdot y_j \bmod q$
19: $Y_i^{j,j'} \leftarrow Y_i^{j-1,j'} \cdot Y_j$, delete $Y_i^{j-1,j'}$ and Y_j
20: $s_{1,j} \leftarrow s_{1,j-1} + s_j \bmod q$, delete $(s_{1,j-1}, s_j)$
21: $y_{j+1} \leftarrow H_1(y_j)$, $r_{j+1} \leftarrow H_1(r_j)$ delete (y_j, r_j)
22: $Y_{j+1} \leftarrow \alpha^{y_{j+1}} \bmod p$, $j \leftarrow j + 1$
23: **if** $j > K$ **then** add $Y_i^{j-1,j'}$ and $M_i^{j-1,j'} = M_i||j'$ to $\overrightarrow{PK}_i$ and $\overrightarrow{M}_i$, respectively, update $st_j = (\langle i, j-1, j' \rangle, \overrightarrow{M}, \overrightarrow{PK})$ **return** $(\langle st_j, \sigma_{1,K} = (s_{1,K}, x_1) \rangle)$, $sk_j = \perp$, and **exit**
24: **return** $(st_j, \sigma_{1,j} = (s_{1,j}, x_i))$

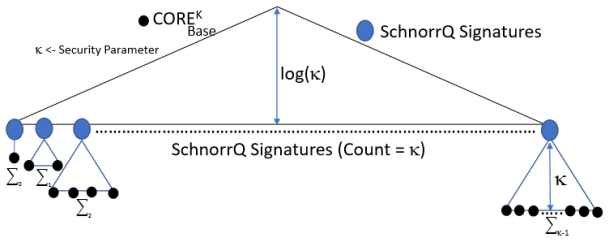
$(0,1) \leftarrow \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Ver}(PK, st{=}(\overrightarrow{M}, \overrightarrow{PK}), \sigma_{1,j})$:

1: **if** $j \geq 1$ and $j' \leq i$ and $i \leq j$ and $j - 1 \neq K$ **then return** 0
2: **else if** $H_1(\prod_{l=1}^{i} \overrightarrow{PK}_l \bmod p) \neq Y'$ **then return** 0
3: **else if** $R' \neq \prod_{l=1}^{i} \overrightarrow{PK}_l^{H_1(\overrightarrow{M}_l||x_l)} \cdot \alpha^{s_{1,K}} \bmod p$, $(x_{l+1} \leftarrow H_3(x_l))$ **then return** 0 **else return** 1

B. CORE−MMM *Scheme*

Our CORE−MMM scheme transforms our $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ scheme into a practically unbounded FAS scheme via MMM [22]. The Algorithm 4 describes instantiation of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ with $\sum$ (see Algorithm 2) and product compositions (see Section IV) with an optimization as in Figure 1. The upper and lower leaves in MMM tree are computed with SchnorrQ [23] and $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$, respectively. Lower trees are created on the run while the upper tree is fixed. $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ signs messages while SchnorrQ certifies lower trees. Leveraging this strategy not only makes the aggregation capabilities of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ possible but it also makes implicit certification very fast.

Fig. 1: Signature Tree Structure of CORE−MMM



We further emphasize the features of CORE−MMM among it's counterparts that makes it ideal for (practically) unbounded forward-secure transformation thanks to the special design of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$: (i) In MMM, despite the final public key is just a hash output, the public keys of base schemes are stored logarithmically as a secret key and then released as a part the signature regularly. Since $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ minimizes the

public key size (only 32 Byte) while also offering signature aggregation, it offers the most compact cryptographic tag sizes (i.e., $|\sigma| + |PK|$) among its counterparts when extended with MMM. (ii) The computational overhead in MMM strictly depends on the key generation, which dictates the cost of key update. Unlike existing FAS alternatives incurring $O(K)$ exponentiations, $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ only requires two for key generation, which makes CORE−MMM key update vastly more efficient than its alternatives. (iii) $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ features "sealing" and batch verification thanks to the partial public key aggregation that enhances the computational efficiency of CORE−MMM.

Algorithm 4 CORE−MMM Algorithm

$(sk,\ PK) \leftarrow \mathsf{CORE{-}MMM.Kg}(r, 1^\kappa)$:
1: $\overline{t_0} = 0, \overline{t_1} = 0, l = 1$
2: $(r_0, r_1) \leftarrow G(r)$, $(r'_0, r'_1) \leftarrow G(r_1)$
3: $(sk_0, PK) \leftarrow \sum.\mathsf{Kg}(r_0, 7, 0, \mathsf{SchQ}, 1^\kappa)$
4: $(sk_1, pk_1) \leftarrow \sum.\mathsf{Kg}(r_0, l, 0, \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}, 1^\kappa)$
5: $\sigma \leftarrow \sum.\mathsf{Sig}(0, sk_0, pk_1, l, 0, \mathsf{SchQ})$
6: $sk_0 \leftarrow \sum.\mathsf{Up}(sk_0, 0, l, 0, \mathsf{SchQ})$
7: **return** $\langle sk{=}\langle sk_0, \sigma, sk_1, pk_1, r'_1\ \rangle, PK\rangle$

$\sigma \leftarrow \mathsf{CORE{-}MMM.Sig}(\bar{t}, sk = \langle sk_0, \sigma_0, sk_1, pk_1, r\rangle, M, \overline{t_1})$:
1: $\overline{t_1} \leftarrow \overline{t_1} + 1$
2: $\sigma_1 \leftarrow \sum.\mathsf{Sig}(0, sk_0, pk_1, 0, \overline{t_1}, \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}})$
3: **return** $(\sigma = \langle pk_1, \sigma_0, \sigma_1\rangle, \bar{t})$

$\mathsf{CORE{-}MMM.UP}(\bar{t}, sk = \langle sk_0, \sigma_0, sk_1, pk_1, r\rangle, \overline{t_0})$:
1: **if** $\bar{t} + 1 \neq 0 \bmod T$ **then**
2: $\quad \sum.\mathsf{Up}(sk_1, \bar{t}\ \ mod\ \ T, t_0, 0, \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}})$ and **exit**
3: **else**
4: $\quad l \leftarrow l + 1$, $t_0 \leftarrow \overline{t_0} + 1$, $(r', r) \leftarrow G(r)$
5: $\quad (sk_1, pk_1) \leftarrow \sum.\mathsf{Kg}(r', l, i, \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}})$
6: $\quad \sigma \leftarrow \sum.\mathsf{Sig}(\lfloor \bar{t}/T_1 \rfloor, sk_0, pk_1, \overline{t_0}, 0, T, \mathsf{SchQ})$
7: $\quad sk_0 \leftarrow \sum.\mathsf{Up}(sk_0, \lfloor \bar{t}/T_1 \rfloor, \overline{t_0}, 0, \mathsf{SchQ})$

(0,1) $\leftarrow \mathsf{CORE{-}MMM.Ver}(pk, M, \langle \sigma{=}\langle pk_1, \sigma, \sigma'\rangle, \bar{t}, \overline{t_0}, \overline{t_1})$:
1: $B_0 \leftarrow \sum.\mathsf{Ver}(pk_1, M, \sigma, \lfloor \bar{t}/T \rfloor, \overline{t_0}, 0, Sch)$
2: $B_1 \leftarrow \sum.\mathsf{Ver}(pk_1, M, \sigma', \bar{t}{-}T, \overline{t_1}, 0, \mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}})$
3: **return** B_0 and B_1

VII. Security Analysis

We prove that $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ is a *FAEU-CMA* signature scheme in Theorem VII.1 in the random oracle model (ROM) [20].

Theorem VII.1.

$$Adv^{FAEU\text{-}CMA}_{\mathsf{CORE(p,q,}\alpha\mathsf{)}}(t, K', K) \quad \leq \quad L \cdot Adv^{DL}_{G,\alpha}(t'),$$

where $O(t') = O(t + K \cdot \kappa^3 + K' \cdot \kappa)$ *(in ROM).*

Proof: Let $\mathcal{A}$ be a $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ *attacker* and ($y \xleftarrow{\$} \mathbb{Z}^*_q, Y \leftarrow \alpha^y \bmod p$) as in Definition 2). We build a *DLP-attacker* $\mathcal{F}$ that uses $\mathcal{A}$ as a sub-routine as follows:

Setup: $\mathcal{F}$ creates three lists $\mathcal{HL}$, $\mathcal{LM}$, and $\mathcal{LS}$ to maintain the query results during the experiment. $\mathcal{HL}$ keep tracks data items queried to $RO(.)$ and their corresponding answers. $\mathcal{LM}$ and $\mathcal{LS}$ keep tracks of data items and their corresponding answers given by $FAS.Sig_{sk}(.)$, respectively.

1) $RO(.)$ *Oracle*: $\mathcal{F}$ runs the function $h' \leftarrow H\text{-}Sim(m, i \in 0, 1, 2, 3)$ to simulate $RO(.)$ answers for (H, H_1, H_2, H_3) on message m. If $(m, i) \in \mathcal{HL}$ then return the previously given answer in $\mathcal{HL}$ corresponding to m. Otherwise, return one of ($h \xleftarrow{\$} \{0,1\}^d, h_1 \xleftarrow{\$} \mathbb{Z}^*_q, h_2 \xleftarrow{\$} \mathbb{Z}^*_q, h_3 \xleftarrow{\$} \{0,1\}^\kappa$) for the query on (H, H_1, H_2, H_3), respectively.
2) *Initialize* $FAS.Sig_{sk}(.)$ *Oracle*: Generate $(y_1, r_1) \xleftarrow{\$} \mathbb{Z}^*_q$ and $x_1 \xleftarrow{\$} \{0,1\}^\kappa$, and set partial private key $sk' = (y_1, r_1, x_1)$ as in $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Kg}(1^\kappa, K)$.
3) *Simulate* PK: $\mathcal{F}$ select a random target forgery index $w \xleftarrow{\$} [1, K]$ and embeds Y into w-th public key as $Y_w = Y$. Since $\mathcal{F}$ does not know the private key (y_w, r_w) corresponding to (Y_w, R_w), $\mathcal{F}$ simulates PK via (sk', Y_w, R_w) as follows:
   - Add $(y_{w+1} \xleftarrow{\$} \mathbb{Z}^*_q, r_{w+1} \xleftarrow{\$} \mathbb{Z}^*_q)$ into sk'
   - $\bar{y}' \leftarrow \sum_{i=1}^{w-1} y_i \bmod q$, $y_{i+1} \leftarrow H\text{-}Sim(y_i, 1)$
   - $\bar{y}'' \leftarrow \sum_{i=w+1}^{K} y_i \bmod q$, $y_{i+1} \leftarrow H\text{-}Sim(y_i, 1)$
   - $\bar{r}' \leftarrow \sum_{i=1}^{w-1} r_i \bmod q$, $r_{i+1} \leftarrow H\text{-}Sim(r_i, 1)$
   - $\bar{r}'' \leftarrow \sum_{i=w+1}^{K} r_i \bmod q$, $r_{i+1} \leftarrow H\text{-}Sim(r_i, 1)$
   - $\bar{y} \leftarrow \bar{y}' + \bar{y}''\ \bmod q$, $\bar{r} \leftarrow \bar{r}' + \bar{r}''\ \bmod q$
   - $(\theta, \gamma) \xleftarrow{\$} \mathbb{Z}^*_q$
   - $R_w \leftarrow (Y^\theta)^{-1}\alpha^\gamma \bmod p$
   - $Y_{1,K} \leftarrow (\alpha^{\bar{r}}) \cdot Y_w \bmod p$, $R_{1,K} \leftarrow (\alpha^{\bar{y}}) \cdot R_w \bmod p$
   - $PK \leftarrow (R' = H\text{-}Sim(R_{1,K}, 1),\ Y' = H\text{-}Sim(Y_{1,K}, 1)$, return PK to $\mathcal{A}$

Execute $\mathcal{A}^{RO(.), FAS.Sig_{sk}(.), Break\text{-}in}(PK)$: $\mathcal{F}$ executes $RO(.)$ for hash queries via $H\text{-}Sim$ function as in *Setup* Step 1. $\mathcal{F}$ responds $\mathcal{A}$'s signature and break-in queries as follows:

1) $FAS.Sig_{sk}(.)$ *Oracle*: (i) $\mathcal{F}$ answers signature queries via $sk' = \{\langle(y_1, r_1)\rangle, \langle(y_{w+1}, r_{w+1})\rangle, x_1\}$ and $RO(.)$ by following $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Sig}$ algorithm, except a query on $(y_w = y)$ corresponding the challenge $Y = Y_w$. (ii) For the query m_w on $y_w = y$, if $M_w = (m_w||w||x_w) \in \mathcal{HL}$ then $\mathcal{F}$ *aborts* and return 0. Otherwise, it sets $s_w \leftarrow \theta$, and programs $H\text{-}Sim$ by inserting γ into $\mathcal{HL}$ for the hash answer of M_w. $\mathcal{F}$ can compute the rest of aggregate signature by using sk' as in (i).
2) *Break-in Oracle*: Given l queries to $FAS.Sig_{sk}(.)$ up to now, if $l = K$ then reject the query and proceed to the *Forgery* . Otherwise, if $l \leq w$ then *abort* and return 0. Otherwise, give the current private key (y_l, r_l, x_l) to $\mathcal{A}$.

Forgery and Extraction: $\mathcal{A}$ outputs a forgery for PK as $(st^*, \sigma^*_{1,l})$. By Definition 3, $\mathcal{A}$ wins if $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Ver}(PK,\ st^* = \langle \overrightarrow{M}^*, \overrightarrow{PK}^*\rangle, \sigma^*_{1,l}) = 1$ and $\exists z \in \{1, \ldots, l\} : \overrightarrow{M}^*[z] \notin \overrightarrow{M}$ holds. If $\mathcal{A}$ loses in the *FAEU-CMA* experiment, then $\mathcal{F}$ also loses in the $DL\text{-}experiment$, and therefore $\mathcal{F}$ *aborts* and returns 0. Otherwise, $\mathcal{F}$ continues as follows: If $(z < w)$ then $\mathcal{F}$ *aborts* and return 0, where $z = |\overrightarrow{M}^*|$ (i.e., $\mathcal{A}$'s forgery is

valid but it is not on the values $Y = Y_w$). Otherwise, $\mathcal{F}$ proceeds for the discrete log extraction as follows: The forged aggregate signature $s^*_{1,l}$ is valid on PK, and $\mathcal{F}$ knows sk except $(y_w = y, r_w)$, which are in the forged signature γ^*. Therefore, $\mathcal{F}$ can extract γ^* from $s^*_{1,l} \in \sigma^*_{1,l}$ by using private key values at hand. (i) $\alpha^{\gamma} \equiv Y_w^{\theta} \cdot R_w \bmod p$ holds due to the simulation. (ii) Since $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Ver}(PK, st^* = \langle \overrightarrow{M}^*, \overrightarrow{PK}^* \rangle, \sigma^*_{1,l}) = 1$ holds, $\alpha^{\gamma^*} \equiv Y_w^{-h^*_w} \cdot R_w \bmod p$ also holds, where $(Y_w = Y) \in PK$ and $h^*_w \leftarrow H\text{-}Sim(\overrightarrow{M}^*[w]||w||x_w)$. This implies the below modular linear equations with only unknowns $(y_w = y, r_w)$. $\mathcal{F}$ then can extract y_w by solving them.

$$\begin{aligned} \gamma &\equiv \theta \cdot y_w + r_w \bmod q, \\ \gamma^* &\equiv -h^*_w \cdot y_w + r_w \bmod q, \end{aligned}$$

$Y \equiv \alpha^{y_w} \bmod p$ holds, as $\mathcal{A}$'s forgery is valid and non-trivial on Y. By Definition 2, $\mathcal{F}$ wins the $DL\text{-}experiment$.

Indistinguishability, Probability and Running Time Analysis: The real-view of $\mathcal{A}$ is comprised of $(PK, st^* = \langle \overrightarrow{M}^*, \overrightarrow{PK}^* \rangle, \sigma^*_{1,l} = \langle s_{1,l}, x_l \rangle)$. In simulation, the values in $sk' = (y_1, r_1, y_{w+1}, r_{w+1})$ are randomly selected from Z^*_q, and intermediate keys derived from sk' are computed via $H\text{-}Sim$ acting as a random oracle. The simulated signature $\sigma = \theta \cdot y_w + r_w$ has identical distribution to that of the real signature s_w, since $y_w = y$ corresponds to *DL-Challenge* $Y \xleftarrow{\$} \mathbb{Z}^*_q$. Due to the simulation and $\theta \xleftarrow{\$} \mathbb{Z}^*_q$, r_w is a random element from Z^*_q. Hence, the values in $\mathcal{A}$'s simulated-view has an identical joint probability distribution to that of $\mathcal{A}$'s real-view as hash functions are modeled as a random oracle.

(a) $\mathcal{F}$ does not abort during $\mathcal{A}$'s queries: (i) if $\mathcal{F}$ queries $M_w = (m_w||w||x_w)$ to $RO(.)$ oracle before querying it to $FAS.Sig_{sk}(.)$ oracle, then $\mathcal{F}$ aborts. This requires $\mathcal{A}$ to guess x_1 or x_w, which only occurs with a negligible probability $1/2^{\kappa}$. If $\mathcal{F}$ queries $Break\text{-}in$ oracle on $l \leq w$th private, then $\mathcal{A}$ aborts as they don't know y_w. Therefore, the probability that $\mathcal{F}$ does not abort is l/K. (b) $\mathcal{A}$ does not abort due to the $\mathcal{F}$'s query replies, as $\mathcal{A}$'s real-view is perfectly indistinguishable than $\mathcal{A}$'s simulated view. Therefore, the probability that $\mathcal{A}$ produces a forgery is $Adv^{FAEU\text{-}CMA}_{\mathsf{CORE(p,q,}\alpha\mathsf{)}}(t, K', K)$. (c) $\mathcal{F}$ does not abort during the extraction, if $\mathcal{A}$'s forgery includes wth (simulated public) key, which occurs with at least $1/l$. The probability that $\mathcal{A}$ produces a forgery without querying $RO(.)$ oracle is negligible in terms of κ. $\mathcal{F}$ wins the experiment if all the above events in (a,b,c) happens, and therefore the upper bound on *FAEU-CMA-advantage if* $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ is:

$$Adv^{FAEU\text{-}CMA}_{\mathsf{CORE(p,q,}\alpha\mathsf{)}}(t, K', K) \quad \leq \quad L \cdot Adv^{DL}_{G,\alpha}(t')$$

The running time of $\mathcal{F}$ is that of $\mathcal{A}$ plus the overhead of setup, query handling and extraction. The setup requires only a small-constant number of costly operations (e.g., exponentiation) and therefore it is negligible. The simulation and extraction have similar costs to $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Sig}$ and $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}.\mathsf{Ver}$ overhead plus random number generations for $RO(.)$. Hence, the estimated running time is $O(t') = O(t + K \cdot \kappa^3 + K' \cdot \kappa)$, where the cost of an exponentiation and random number generation are κ^3 and κ, respectively.

In Corollary VII.1, we show that the security of $\mathsf{CORE}-\mathsf{MMM}$ follows from that of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ and generic MMM transformation. We do not repeat the security theorems in MMM for brevity, and refer curios readers to [22].

Corollary VII.1.

$$Adv^{FEU\text{-}CMA}_{\mathsf{CORE\text{-}MMM}}(t, K', K) \quad \leq \quad \bar{t} \cdot Adv^{FAEU\text{-}CMA}_{\mathsf{CORE(p,q,}\alpha\mathsf{)}}(t, K', K),$$

Proof: Let $\mathsf{CORE}-\mathsf{MMM}$ be an MMM instantiation of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$via sum and product compositions for the maximum T time period with the number of update operations so far denoted as $\bar{t}$. $Adv^{FEU\text{-}CMA}_{\mathsf{CORE\text{-}MMM}}(t, K', K)$ is a *Forward-secure* $\mathsf{EU\text{-}CMA}$ *(FAEU-CMA)* signature scheme, since (i) $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ is a $FAEU\text{-}CMA$ secure by Theorem VII.1, (ii) *SchnorrQ* is $\mathsf{EU\text{-}CMA}$ secure [23], and sum and product composition are secure by Theorem 1 and Theorem 5 as in [22], respectively. Hash functions are modeled as random oracle in $\mathsf{CORE}-\mathsf{MMM}$ and therefore are collision-resilient as required by [22]. The advantage of $\mathcal{A}$ against $\mathsf{CORE}-\mathsf{MMM}$ is bounded by the total number of updates so far $\bar{t}$ with that of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$. SchnorrQ is invoked only $1 < K << \bar{t}$, and therefore $\mathcal{A}$'s advantage and execution time costs are dominated by $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$. The rogue-key attacks in multi-user settings [24] do not apply, since $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ is in single-signer setting and PK is certified at the key generation.

VIII. Performance Analysis

In this section, we compare the performance of CORE schemes with their counterparts.

A. Evaluation Metrics and Experimental Setup

We compare CORE schemes and their counterparts in terms of (i) private, signature and public key sizes, and (ii) key generation, signature generation, update and verification execution times. We fully implemented $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ on elliptic curves on an Intel i7-6700HQ 2.6 GHz processor with 12 GB of RAM as the commodity hardware in our experiments. FourQlib[1] is utilized for curve & arithmetic operations. We used BLAKE2b [25] as our hash function. Base implementations (if available) are used for the counterparts or we conservatively measured their performance based on the unit costs. Our open-source implementation is located at:

www.github.com/efeseyitoglu/COREBASE

B. Performance Evaluation and Comparisons

We present the performances of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ and its K-time counterparts in Table I and Table II, analytically and experimentally, respectively. Recall that a fast key generation and compact public key are essential requirements for an efficient MMM transformation. As seen in Table II, for $K = 2^{10}$, the key generation and public key size of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ is $2035\times$ faster and $16\times$ smaller, respectively, than its closest counterpart (i.e., XMSS). It also has a highly competitive signing (including key update) and verification speeds with a private key size of only 80 bytes. This makes it the best

[1]https://github.com/Microsoft/FourQlib

TABLE I: Private/public key sizes, signature size and signature generation/verification costs of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ and its counterparts

Scheme	Signer				Verifier	
	Key Generation Time	Private Key Size	Signature Size	Signing Time ‡	Public Key Size	Verification Time
HORS [26]	$2{\cdot}\kappa{\cdot}\ K\ {\cdot}H$	κ	$K \cdot \kappa \cdot u$	$(u+1) \cdot H$	$\overline{t'} \cdot \lvert H\rvert \cdot K$	$K \cdot (u+1) \cdot H$
SchnorrQ [23]	$K \cdot EMul$	$\lvert q\rvert$	$K \cdot 2\lvert q\rvert$	$EMul$	$K \cdot \lvert q\rvert$	$K \cdot 1.3 \cdot EMul$
XMSS [27]	$K \cdot (3 + l \cdot (w+2)) \cdot H$	κ	$K \cdot (l + \lvert K\rvert) \cdot \lvert H\rvert$	$(((\lvert K\rvert + 2) \cdot (\lvert K\rvert + l \cdot (w+2)))/2 + 4 \cdot \lvert K\rvert) \cdot H$	$(2(\lvert K\rvert + \lvert l\rvert) + 1) \cdot \lvert H\rvert$	$K \cdot (\lvert K\rvert + l \cdot (w+1)) \cdot H$
*BAF [4]	$2 \cdot K \cdot EMul$	$4\lvert q\rvert$	$2\lvert q\rvert$	$Mulq + 4H$	$(4K-1) \cdot \lvert q\rvert$	$K \cdot EMul$
*FssAgg−BLS [3]	$K \cdot EMul$	$\lvert q\rvert$	$\lvert q\rvert$	$2H + EMul + Mul_q$	$K \cdot \lvert q\rvert$	$K \cdot PR$
$*\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$	$2 \cdot EMul$	$2\lvert q\rvert + \kappa$	$\lvert q\rvert \cdot f + 2\lvert q\rvert$	$EMul + ((K-f)/K) \cdot Eadd$	$2\lvert q\rvert$	$(1.3 + f) \cdot EMul$

K is the maximum number of signatures that can be generated for K-time signature schemes. f denotes the frequency of message changes out of K messages to be signed. $Emul$ and $Eadd$ denote the costs of EC scalar multiplication over modulus p, and EC addition, respectively. H and Mul_q denote a cryptographic hash and a modular multiplication over modulus q, respectively. We omit the constant number of negligible operations if there is an expensive operation (e.g., hashing, $Eadd$ are omitted if there is an $Emul$). We use double-point scalar multiplication for verifications of ECC based schemes ($1.3 \cdot Emul$ instead of $2 \cdot Emul$). Integers t' and u denote the parameters used in HORS [26]. w is the Winternitz parameter and l is the tree parameter in XMSS [27]. PR denotes for the pairing cost (a1 curve is used for FssAgg-BLS). For HORS [26] and SchnorrQ. [23], we deterministically generate the private keys from a seed (i.e., via a keyed hash). $*$ denotes forward secure and aggregate signatures. We have presented the costs for the signature size, signature verification and key generation for K messages and signature generation for a single message. ‡ denotes that cost is calculated per message.

TABLE II: Experimental performance comparison of $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ and its counterparts, where $K = 2^{10}$

Scheme	f	Key Generation Time (μs)	Signing ‡ Time (μs)	Private Key Size (Byte)	Signature Size (KB)	Verification Time (ms)	Public Key Size ‡
HORS [26]	N/A	36700.16	6.47	16	384	3.60	32 MB
SchnorrQ [23]		10362.88	10.89	32	64	21.69	32 KB
XMSS [27]		55728	322.41	16	2 368	45.63	1 056 Byte
*BAF [4]		20725.76	1.21	128	0.0625	16.21	131040 Byte
*FssAgg−BLS [3]		10362.88	911.02	32	0.03125	11250.69	32768 Byte
$*\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$	1	**27.38**	11.11	80	**0.09375**	**0.02**	**64 Byte**
	2^3		10.87		**0.3125**	**0.15**	
	2^9		10.61		16.0625	8.52	
	2^{10}		10.28		32.0625	17.07	

The signing costs, which also includes the key update cost (relatively small), are given *per message*. $K = 2^{10}$, $p = 2^{127} - 1$, t' = 1024, u = 24, w = 4, l = 67, hash output = 32 Bytes, |q| = 32 Bytes. The cost of hash-based schemes are calculated based on the cost of a single hash operation.

TABLE III: Analytical comparison of CORE−MMM and its standard unbounded forward-secure counterparts

Scheme	Signing ‡ Time (μs)	Private Key Size (Byte)	Update Time	Signature Size (Byte)	Verification Time (μs)	Public Key (Byte)
HORS−MMM	$(u+1) \cdot H$	$(5\lvert\kappa\rvert + 3log(\bar{t}) + 6\overline{t'} + 2) \cdot \lvert H\rvert + (2+u) \cdot \kappa$	$2 \cdot \kappa \cdot H$	$K \cdot ((2\lvert\kappa\rvert + 2\lvert log(\bar{t}) + 4\overline{t'} + 1) \cdot \lvert H\rvert + 2\kappa \cdot u)$	$K \cdot (2u+2) \cdot H$	$\lvert H\rvert$
SchnorrQ−MMM	$EMul$	$(5\lvert\kappa\rvert + 3log(\bar{t}) + 2) \cdot \lvert H\rvert + 10\lvert q\rvert$	$EMul$	$K \cdot ((2\lvert\kappa\rvert + 2log(\bar{t}) + 1) \cdot \lvert H\rvert + 8\lvert q\rvert)$	$K \cdot 2.6 \cdot EMul$	$\lvert H\rvert$
BAF−MMM	$Mulq + 4H$	$(5\lvert\kappa\rvert + 3log(\bar{t}/K) + 2) \cdot \lvert H\rvert + (12K+4) \cdot \lvert q\rvert$	$2 \cdot EMul$	$(2\lvert\kappa\rvert + 2log(\bar{t}/K) + 1) \cdot \lvert H\rvert + (8K+4) \cdot \lvert q\rvert$	$(1.3 + K) \cdot EMul$	$\lvert H\rvert$
CORE−MMM	$EMul + ((K-f)/K) \cdot Eadd$	$(5\lvert\kappa\rvert + 3log(\bar{t}/K) + 2) \cdot \lvert H\rvert + 14\lvert q\rvert + \kappa$	$(2/K) \cdot EMul$	$(2\lvert\kappa\rvert + 2log(\bar{t}/K) + 1) \cdot \lvert H\rvert + (10+f) \cdot \lvert q\rvert$	$(2.6 + f) \cdot EMul$	$\lvert H\rvert$

$\bar{t}$ denotes the signatures signed so far, since MMM costs depends on this value. In MMM, the signing execution time includes key generation times.

TABLE IV: Performance comparison of CORE−MMM and its standard unbounded forward-secure counterparts

Scheme	f	Signing ‡ Time (μs)	Private Key Size (Byte)	Update Time (μs)	Signature Size (KB)	Verification Time (ms)	Public Key Size (Byte)
HORS−MMM	N/A	3.51	201088	35.84	134240	7.21	32
SchnorrQ−MMM		10.22	4384	10.12	2656	44.56	32
BAF−MMM		1.21	396448	20.24	257.8	16.23	32
CORE−MMM	1	11.11	**3568**	**0.03**	**2.1**	**0.04**	32
	2^3	10.87			**2.3**	**0.17**	
	2^9	10.61			**18**	**8.54**	
	2^{10}	10.28			**34**	17.09	

The costs of MMM instantiations are calculated based on the base schemes and analytical performances as in Table III (parameters are as in Table II). Recall that the signature verification is given for K items in all compared schemes.

candidate for MMM transformation among its alternatives as shown in the performance analysis of CORE−MMM below.

We now showcase the analytical and experimental performance comparison of CORE−MMM and its counterparts in Table III and Table IV, respectively. Note that the performance of MMM schemes depends on signatures signed so far, which makes it practically unbounded. We consider up to $t = 2^{30}$ messages to be signed, that corresponds to 3.4 signatures per second (non-stop) for 10 years. We consider that this should suffice for almost all practical audit logging applications. We set $K = 2^{10}$ for all base schemes as in Table I and Table II.

All MMM extensions have a small constant-size public key of 32 Bytes. CORE−MMM has the fastest update (337× better than SchnorrQ−MMM) and therefore the most efficient signing plus update time among its alternatives. It has also the smallest signature (7.5× better than BAF−MMM) and private key size. Finally, CORE−MMM has a competitive verification time compared to its alternative, albeit being

slightly less efficient than HORS−MMM and BAF−MMM. Note that CORE−MMM gains substantial performance advantage when the same message is being consecutively signed with evolving keys. For instance, when the frequency of message changes is $f = 8$ over $K = 2^{10}$, CORE−MMM can offer approximately two magnitudes of smaller signature sizes and faster verification than its best counterparts. In summary, CORE−MMM vastly outperforms its alternatives in almost all metrics except signature verification, and for all metrics when the same message is being signed repeatedly for some time interval with example parameters $1 \leq f \leq 2^9, K = 2^{10}$.

Remark: The signature size and verification costs are calculated cumulatively over K messages. Update costs are amortized as in MMM [22].

IX. Conclusion

In this work, towards addressing the authentication and integrity needs of audit logging applications, we created new forward-secure and aggregate schemes that we call as COmpact and REsilient (CORE) signatures. Our base scheme is a novel K-time forward-secure signature $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ that has a small constant-size public key with signature and conditional public key aggregation. We then developed CORE−MMM that relies on $\mathsf{CORE}^{\mathsf{K}}_{\mathsf{Base}}$ to achieve a practically unbounded number of signatures with high performance. CORE−MMM vastly outperforms its counterparts for essential metrics such as key update, private key and signature sizes with an equal public key size and security. Moreover, the performance of CORE schemes is further boosted for secure logging applications with infrequent state transitions, for which CORE outperforms its alternatives for almost all metrics. Overall, our analysis suggests that CORE is an ideal cryptographic tool to enhance the security of audit logging applications with low overhead and practically unbounded signing capability.

Acknowledgements: We would like to thank the anonymous reviewers for their valuable feedback. This work was supported by the NSF CAREER Award CNS-1652389 and an unrestricted gift via Cisco Research Award.

References

[1] G. Hartung, "Secure audit logs with verifiable excerpts," in *Topics in Cryptology - CT-RSA 2016*, K. Sako, Ed. Cham: Springer International Publishing, 2016, pp. 183–199.

[2] G. A. Marson and B. Poettering, "Even more practical secure logging: Tree-based seekable sequential key generators," in *Computer Security - ESORICS 2014*, M. Kutyłowski and J. Vaidya, Eds. Cham: Springer International Publishing, 2014, pp. 37–54.

[3] D. Ma and G. Tsudik, "Forward-secure sequential aggregate authentication," in *Proceedings of the 28th IEEE Symposium on Security and Privacy (S&P '07)*, May 2007, pp. 86–91.

[4] A. A. Yavuz, P. Ning, and M. K. Reiter, "BAF and FI-BAF: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems," *ACM Transaction on Information System Security*, vol. 15, no. 2, 2012.

[5] D. Ma and G. Tsudik, "A new approach to secure logging," in *Proc. of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC '08)*, 2008, pp. 48–63.

[6] ——, "A new approach to secure logging," *Trans. Storage*, vol. 5, no. 1, pp. 2:1–2:21, Mar. 2009. [Online]. Available: http://doi.acm.org/10.1145/1502777.1502779

[7] A. A. Yavuz and P. Ning, "BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems," in *Proceedings of 25th Annual Computer Security Applications Conference (ACSAC '09)*, 2009, pp. 219–228.

[8] G. A. Marson and B. Poettering, "Practical secure logging: Seekable sequential key generators," in *Computer Security – ESORICS 2013*, J. Crampton, S. Jajodia, and K. Mayes, Eds. Springer Berlin Heidelberg, 2013, pp. 111–128.

[9] M. Bellare and P. Rogaway, "Introduction to modern cryptography," in *UCSD CSE Course*, 1st ed., 2005, p. 207, http://www.cs.ucsd.edu/~mihir/cse207/classnotes.html.

[10] M. Abdalla and L. Reyzin, "A new forward-secure digital signature scheme," in *Advances in Crpytology (ASIACRYPT '00)*. Springer-Verlag, 2000, pp. 116–129.

[11] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. of the 22th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '03)*. Springer-Verlag, 2003, pp. 416–432.

[12] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," *ACM Transaction on Information System Security*, vol. 2, no. 2, pp. 159–176, 1999.

[13] S. Crosby and D. S. Wallach, "Efficient data structures for tamper evident logging," in *Proceedings of the 18th conference on USENIX Security Symposium*, August 2009.

[14] C. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

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

[16] G. Hartung, "Attacks on secure logging schemes," in *Financial Cryptography and Data Security*. Cham: Springer International Publishing, 2017, pp. 268–284.

[17] P. Kampanakis and A. A. Yavuz, "BAFi: A practical cryptographic secure audit logging scheme for digital forensics," *Security and Communication Networks*, March 2015.

[18] J. Kim and H. Oh, "Fas: Forward secure sequential aggregate signatures for secure logging," *Information Sciences*, vol. 471, pp. 115 – 131, 2019.

[19] L. Babun, A. K. Sikder, A. Acar, and A. S. Uluagac, "Iotdots: A digital forensics framework for smart environments." *CoRR*, vol. abs/1809.00745, 2018. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1809.html#abs-1809-00745

[20] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proceedings of the 1st ACM conference on Computer and Communications Security (CCS '93)*. NY, USA: ACM, 1993, pp. 62–73.

[21] A. A. Yavuz, "Eta: efficient and tiny and authentication for heterogeneous wireless systems," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*, ser. WiSec '13. New York, NY, USA: ACM, 2013, pp. 67–72.

[22] T. Malkin, D. Micciancio, and S. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods," in *Advances in Cryptology - Eurocrypt 2002*, ser. Lectture Notes in Computer Science, vol. 2332, IACR. Amsterdam, The Netherlands: Springer-Verlag, April 28-May 2 2002, pp. 400–417.

[23] C. Costello and P. Longa, "Schnorrq: Schnorr signatures on fourq," MSR Tech Report, 2016. Available at: https://www. microsoft. com/en-us/research/wp-content/uploads/2016/07/SchnorrQ. pdf, Tech. Rep., 2016.

[24] D. Boneh, M. Drijvers, and G. Neven, "Compact multi-signatures for smaller blockchains," in *Advances in Cryptology – ASIACRYPT 2018*. Cham: Springer International Publishing, 2018, pp. 435–464.

[25] J.-P. Aumasson, L. Henzen, W. Meier, and R. C.-W. Phan, "Sha-3 proposal blake," Submission to NIST (Round 3), 2010. [Online]. Available: http://131002.net/blake/blake.pdf

[26] L. Reyzin and N. Reyzin, "Better than BiBa: Short one-time signatures with fast signing and verifying," in *Proceedings of the 7th Australian Conference on Information Security and Privacy (ACIPS '02)*. Springer-Verlag, 2002, pp. 144–153.

[27] A. Huelsing, D. Butin, S.-L. Gazdag, J. Rijneveld, and A. Mohaisen, "XMSS: eXtended Merkle Signature Scheme," RFC 8391, May 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8391.txt