Efficient Forward-Secure and Compact Signatures for the Internet of Things (IoT)

Efe Ulas Akay Seyitoglu
University of South Florida

Efficient Forward-Secure and Compact Signatures for the Internet of Things (IoT)

by

Efe Ulas Akay Seyitoglu

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Date of Approval:
February 6, 2020

Keywords: Storage efficiency, aggregation, unbounded message signing,
resource-constrained devices, audit logs

Dedication

It is very hard for me to put in words the positive impact my advisor Prof. Attila Altay Yavuz made to my life. He was always there for me throughout my grad school journey. His support and mentoring were (and still are) extremely invaluable for me. I feel very privileged and lucky to be his student.

I also would like to thank my committee members Prof. Jay Ligatti and Prof. Mehran Mozaffari Kermani for their time.

Lastly, I would like to thank my mother Emine Gul Kapci, my father Gurol Seyitoglu and my brother Deniz Ege Seyitoglu for their never-ending encouragement and belief in me. I consider myself very fortunate to be raised in a family environment where I was thought the value of accumulating scientific knowledge.

Table of Contents

List of Tables

List of Figures

Abstract

In the modern Internet of Things (IoT) applications, the system entities collect security-sensitive information that must be cryptographically protected. In particular, authentication and integrity, as foundational security services, are essential for any IoT applications. Digital signatures provide both authentication and integrity to these applications. Nevertheless, once an IoT device is comprised, its signature private key is leaked to an adversary. Forward-secure digital signatures mitigate the impact of such key compromises by incorporating a key-evolving mechanism into the authentication process. However, existing forward-secure signatures suffer from large tag signature/key sizes, heavy computational overhead, and some prominent variants that can only sign a limited number of messages. Hence, there is a critical need for forward-secure and compact digital signatures that can be used to authenticate large amounts of critical information.

In this work, we proposed two forward-secure signatures with signature and partial public key aggregation capabilities that we refer to as $CORE_{Base}^{K}$ and CORE-MMM. Our first scheme, to the best of our knowledge, is the first K-time forward-secure and aggregate signature scheme with a public-key aggregation feature. The idea is to use a hash-chain mechanism to evolve the keys and pre-compute the aggregated public key. For each message, we compute its signature and aggregate it. $CORE_{Base}^{K}$ offers compact public keys as well as compact signatures with low verification overhead. We fully implemented $CORE_{Base}^{K}$ in commodity hardware and tested for various performance metrics. We also compared $CORE_{Base}^{K}$ with its most efficient counterparts. For instance, $CORE_{Base}^{K}$ has 180x faster signature verification compared to its most verification-efficient counterpart, it also has 16.5x more compact public-keys compared to the most public-key compact counterpart. Our second scheme CORE-MMM, is a practically unbounded forward-secure signature scheme that leverages

$CORE_{Base}^{K}$. Our use of $CORE_{Base}^{K}$ is central to the design of $CORE_{Base}^{K}$ because we crafted $CORE_{Base}^{K}$ to optimize the performance of unbounded signing capability under the generic MMM transformation. To the best of our knowledge, this specific design led to the most efficient compromise-resilient and compact signature which we refer to as CORE-MMM. We also compared the performance of CORE-MMM with its state-of-art alternatives. Our analysis shows that CORE-MMM outperforms its state-of-art counterparts in most performance metrics. Some notable examples include small public keys (only 32 Bytes), more than two magnitudes more efficient key updates, compact signatures, and a magnitude smaller private keys compared to its most efficient counterparts for each metric.

Chapter 1: Introduction

Modern IoT applications record security events and system state changes in audit logs. If the attacker compromises the audit log of an IoT application, they can delete critical entries including their own traces. Attackers can also forge false entries on the audit log for malicious purposes. Standard digital signatures offer authentication and integrity services however, they are still vulnerable against dedicated attackers who can physically breach into the IoT device and capture the secret key. Therefore, it is critical to ensure that modern IoT applications have comprimise-ressiliency (i.e. the consequence of the secret key exposure should be minimal).

To achieve compromise resiliency, we need forward-security. Forward security is an important cryptographic technique that is used as a pre-caution for the possible leak of the secret keys. If a secret key is leaked and forward security is not employed, all of the signatures that are created via the secret key would be unreliable. This would create problems because individuals that wish to deny the ownership of a signature could intentionally leak the secret key they used for creating the signature (which violates the non-repudiation property). Moreover, if the system contains some private information that is encrypted by the secret key, the exposure of the secret key will lead to the leakage of the private information (violates the confidentiality property). Therefore, it is of paramount importance that we create forward-secure cryptographic schemes to limit the potential consequences that may stem from the leakage of the secret key [1, 2, 3].

The main idea of using forward-secure signatures is to leverage a key-evolving strategy to update the signing key in certain time intervals while keeping the public key unchanged. To accomplish this key-evolving strategy, the one-way property of hash functions are usually leveraged. In typical forward-secure signature schemes, after the time interval t_1 is over, the secret key used in t_1 gets replaced by it's hash ($sk' = H(sk)$). After this operation, the secret key that was used for time interval t_1 is deleted so that if the attacker compromises the system at the time interval t_2, they will not have access to the secret key used in the time interval t_1. Thus, all cryptographic operations that were done using the secret key in the time interval t_1 and before will be preserved.

Even though this key-evolving strategy makes forward-security possible, the standard forward-secure signatures suffer from large signatures and keys. Several forward-secure signature schemes with aggregation capabilities were proposed to reduce these large signatures and keys [4]. However, the most storage-efficient forward-secure and aggregate signature in the literature is still considerably high(the most storage-efficient forward-secure signature either has linear signature or public key or (signature + public key cost)). This storage cost is an important problem especially for resource-constrained devices.

The other important challenge of forward-secure and aggregate signatures is that they are bounded to work for a pre-defined number of periods (we will refer to this pre-defined number as K throughout the thesis). The problem with this is that after K signings, the IoT device would need to be manually reset to accommodate newer messages. Manually resetting the IoT device can be very problematic (or even impossible) for many real-world situations.

Our contribution in this paper is that, we propose two forward-secure and aggregate schemes. We named our first scheme as $CORE^K_{Base}$. To the best of our knowledge, our first scheme $CORE^K_{Base}$ is the first K-time forward-secure aggregate signature scheme with signature and public key aggregation capabilities. Our second scheme is named CORE-MMM. CORE-MMM leverages our $CORE^K_{Base}$ scheme to create an efficient signature scheme with unbounded number of time periods. Our CORE-MMM construction uses the Malkin-Micciancio-Miner (MMM) [5] algorithm to be able to sign practically unbounded number of messages. Details of our schemes will be explained in much more detail in the proposed schemes section of the thesis.

Chapter 2: Related Work

In this part of chapter of the thesis, we will discuss some of the constructions that are relevant to our contributions. This section will be composed of four parts. Firstly, we will discuss important secure audit logging schemes. Our discussion will continue with forward-secure signatures. It will be followed by a discussion on fundamental and state-of-art aggregation techniques for digital signatures. Lastly, we will discuss forward-secure and aggregate signatures (our scope will be both traditional and the state-of-art).

2.1 Secure Audit Logging Schemes

Audit logs are used in various applications due to their forensic value. One good example is a file storage system where the past states of the files are stored for versioning purposes. In this line of work, Peterson et. al. presented a new perspective on how to verify archived files that are stored by the third parties. The general idea is that whenever a certain version history of a file is to be saved, a corresponding message authentication code (MAC) is generated. Auditors that wish to verify the version history of a file can do so via verifying the corresponding message authentication code. Pederson et. al. uses incremental message authentication codes that allow MACs to be generated according to the differences between file updates. Leveraging this technique becomes advantageous because with incremental message authentication codes, the only data that needs to be read is the one in the cache. Therefore, the accesses to the disk are highly reduced [6]. Another example usage of the audit logs is with the database systems. Assume a user replaces an entry in the database, the database is now changed but the previous versions of the database are saved as the audit log following the change. To keep the database system audit logs tamper-evident (i.e. the logged events are not deleted and/or altered), Snodgrass et. al. proposes the use of notarization services. When there is a change on the database, their approach is to hash all the modified tuples and certify them with the notarization service. This way they are able to limit the number of calls they need to make to the notarization center (because they do not ask the notarization center to certify each tuple in the database but to certify the hash of all the modified data) while keeping the audit logs tamper-evident [7, 8].

In addition to constructions for securing audit logs that are used for a specific purpose, there are also generic constructions that could be applied to any audit log scheme to make it more reliable. One example for this is the use of hash-chains to detect tampering in the audit logs. Every time the audit log gets updated, this event is recorded on the hash chain. An auditor that wishes to check a specific modification on the audit log has to iterate through the hash chain to find what s/he is looking for. It is easy to see that this construction would require linear storage cost O(M) (where M is the number of modifications on the audit log). A good improvement on this is by Crosby and Wallach. They improve on the classic hash-chain constructions by leveraging a tree structure. They improve on the storage cost since their scheme requires logarithmic storage (O(log(M)) which is an important improvement considering hash chains require linear storage (O(M)).[8]

2.2 Forward Secure Signatures

Forward-secure signatures provide non-repudiation property that traditional digital signatures fail to provide. In other words, the traditional digital signatures cannot legitimize a signature that was signed with an exposed secret key. Anderson in 1997, came up with the first forward-secure signature scheme to mitigate the consequences that may be caused by an exposed secret key. Anderson proposed that all of the secret keys should have a time period that they are valid in. After the valid time period ends, the current secret-key must be replaced by the new secret-key. The new secret-key is created by a one-way function that takes the current secret-key as the input. After the new secret-key replaces the current secret-key, the replaced key is deleted. An adversary that breaks-in the system only has access to the signing key for that specific period. Therefore, it is impossible for the adversary to compromise the past secret keys because all of them have been deleted and it is extremely hard to reverse the one-way functions. [2, 9]. It is also important to note that the first forward-secure signature was formalized by Bellare and Miner [10, 11] and an important improvement that shortens the keys was proposed by Abdalla and Reyzin [2].

2.3 Aggregate Signatures

The first ever aggregate signature was proposed by Boneh et. al. The idea of aggregate signatures is that given k different signatures on k different messages, an aggregation of these k signatures will result in a single (compact) signature that can decide whether all of these signatures are valid or not [12]. If even one of the signatures among all the k signatures that are aggregated to produce the single compact signature is not legitimate, the aggregated signature will not verify. Following this invention, the first aggregate signature scheme that is provably secure without random oracles was proposed by Lu et. al. [13]. Another important improvement to the initial aggregate signature schemes was proposed by Bellare, Chanathip and Neven. Remember that the initial signature scheme of Boneh et. al. required k different signatures on k different messages for aggregation to occur [12]. Bellare, Chanathip and Neven proposed a scheme that would allow aggregation of signatures in the presence of duplicate messages. This is a very important improvement because there are many real life scenarios where aggregation of signatures would be extremely useful when the duplicate messages exist. The example scenario they give in their paper is the sensor network deployed to detect a natural disaster. They point out that the environmental data that is captured by the sensors can repeat. Without their improvement, the aggregation of signatures in such scenarios would not be possible [14]. After the discovery of forward-secure signatures and aggregate signatures, the combination of these signatures called forward-secure and aggregate signature schemes were proposed to aid resource-constrained devices. Ma et. al's work on forward secure and aggregate signature schemes is a fundamental example. Their idea is to leverage the signature aggregation capabilities of the BLS signatures [15] for compactness. They apply aggregation on all of the signatures that the forward-secure system creates.

2.4 Forward-Secure and Aggregate Signatures

Just like traditional aggregate signatures, if any of the intermediate signature in the forward-secure system is not legitimate, verification of the aggregate signature will fail. The drawback of this scheme is the cost of signature verification because of bilinear maps [11]. Another important example is called the BAF (Blind-Aggregate-Forward) scheme. BAF has strong features such has: (i) fast logging (only a constant number of hash functions are necessary), (ii) constant communication and storage overhead, (iv) being publicly verifiable and provably secure [16].

Chapter 3: Preliminaries

In this section of the thesis, we will also discuss some of the algorithms that are necessary to understand our contributions. To be more specific, we will discuss the ETA (Efficient and Tiny and Authentication for Heterogeneous Wireless Systems) signature, the Schnorr signature and the MMM (Malkin-Micciancio-Miner) algorithm. All of these algorithms play a role in either our $CORE_{Base}^{K}$ and/or CORE-MMM schemes. In the table 1, we explain the notations we use for certain variables.

Table 3.1 Table of Notations

Notation	**Description**
$\|\|$	Concatenation of two variables.
$\|a\|$	Bit length (e.g. $\|a\| = \log_2 a$).
i, $(x_0, \ldots, x_i) \overset{\$}{\leftarrow} \mathcal{S}$	$(x_0 \overset{\$}{\leftarrow} \mathcal{S}, \ldots, x_i \overset{\$}{\leftarrow} \mathcal{S})$.
$\{0,1\}^*$	Binary strings which have finite length.
$\{q_k\}_{k=0}^{t-1}$	All the items q_k for $k = 0, \ldots, t-1$
$\log x$	Logarithm with base 2 ($\log_2 x$)
K	Number of periods (upper bound)
H_1	$Z_q^* \to Z_q^*$
H_2	$\{0,1\}^* \to Z_q^*$
H_3	$\{0,1\}^\kappa \to \{0,1\}^\kappa$
q	A large prime
p	A large prime
α	Generator of a subgroup G (order q in $\mathbb{Z}_p^*$)
T	Maximum number of periods in the MMM [5]

Table 3.1 (Continued)

Hash Function: H	$\{0,1\}^* \rightarrow \{0,1\}^d$ (d is a fixed integer)
$\sum$	Sum composition operation in the MMM [5]
$levels$	Level of the tree
t	Number of periods so far in the MMM [5]
$a \overset{\$}{\leftarrow} \mathcal{S}$	Choose "a" in $\mathcal{S}$ randomly and uniformly.
ind	index
SGN	A generic signature scheme
$SGN.Kg$	Key generation algorithm of SGN
$SGN.UP$	Update algorithm of SGN
$SGN.Ver$	Signature verification algorithm of SGN
$SGN.Sig$	Signature generation algorithm of SGN

3.1 ETA Signature [17]

We do not directly use the ETA signature however, our $CORE^{K}_{Base}$ uses some of the critical ideas of ETA. ETA has a fixed number of messages it can sign to (we denote this number with K). For each message we create a public key (this means the public-key storage is linear with respect to number of messages that can be signed). It is also important to note that ETA is not forward-secure. It also does not have any aggregation feature.

The distinguishing feature of ETA is the signer side efficiency. This is done by pre-computing the public (and also private) keys in the key generation phase. Note that the key generation phase is only done once and is referred to as the "offline stage". Therefore, the pre-computation of the public keys in the key generation phase can be overlooked for many real-world applications.

In the next paragraphs, we will discuss the key generation, the signature generation, and signature verification algorithms of the ETA signature scheme. The idea in ETA key generation is that K (private and public) keys are pre-computed and stored. Starting from the initial secret key ($r_{init} \xleftarrow{\$} \mathbb{Z}_q^*$), the upcoming secret keys are computed by a hash chain mechanism. The public key components $\overline{pk} = pk_0, \ldots, pk_{K-1}$ (which are generated by a private key hash chain of K length) compose the private key along with the initial public key that was created with line 2. The secret key is composed of the initial private key in line 1 along with the starting element of the hash chain denoted by r_0 (using r_0, other private keys in the hash chain can be created).

Algorithm 1 ETA Key Generation [18]

```
    $\underline{(sk_0, PK) \leftarrow \mathsf{ETA.Kg}(1^\kappa, K)}$:
 1: $r_{init} \xleftarrow{\$} \mathbb{Z}_q^*$
 2: $l \xleftarrow{\$} \mathbb{Z}_q^*$
 3: $L \leftarrow \alpha^l \bmod q$
 4: $\mathsf{ind} \leftarrow 0$
 5: while $\mathsf{ind} \leq K - 1$ do
 6:     $R_j \leftarrow \alpha^{r_{\mathsf{ind}}} \bmod p$
 7:     $r_{\mathsf{ind}+1} \leftarrow H(r_{\mathsf{ind}})$
 8:     $pk_{\mathsf{ind}} \leftarrow H(R_{\mathsf{ind}})$
 9:     $\mathsf{ind} \leftarrow \mathsf{ind} + 1$
10: return $sk_0 \leftarrow (r_{init}, l)$ and $PK \leftarrow (\overline{pk} = pk_0, \ldots, pk_{K-1}, L)$
```

The signature generation algorithm in ETA is very similar to the Schnorr signatures. The value r_j is created by iterating through the hash chain using the initial key r_0. Furthermore, there is an additional component ind inside of the $e_{\mathsf{ind}} \leftarrow H(M_{\mathsf{ind}}||\mathsf{ind}||x_{\mathsf{ind}})$ to preserve the order of the signature generation process. The idea is in ETA signature verification is to exponentiate the public key over the hash of what was passed by as a parameter and multiply it by the signature. This should reveal the secret. The secret that is found by this operation is hashed and this hash is compared to the hash of the secret generated in the key generation phase. If they are the same, the signature verifies. if they are not, the signature does not. The signature verification algorithm in ETA is almost identical to SchnorrQ signature verification except the e_{ind} component $H(M_{\mathsf{ind}}||\mathsf{ind}||x_{\mathsf{ind}})$ contains an additional index value "ind" to preserve the order of signatures.

Algorithm 2 ETA Signature Generation [18]

```
$\underline{\sigma_{\mathsf{ind}} \leftarrow \mathsf{ETA.Sig}(sk_{\mathsf{ind}}, M_{\mathsf{ind}})}$:
1: if $\mathsf{ind} \leq K - 2$ then
2:     $x_{\mathsf{ind}} \xleftarrow{\$} \{0,1\}^{\kappa}$
3:     $s_{\mathsf{ind}} \leftarrow r_{\mathsf{ind}} - H(M_{\mathsf{ind}}||\mathsf{ind}||x_{\mathsf{ind}}) \cdot l \bmod q$
4:     $\sigma_{\mathsf{ind}} \leftarrow (s_{\mathsf{ind}}, x_{\mathsf{ind}}, \mathsf{ind})$
5:     $r_{\mathsf{ind}+1} \leftarrow H(r_{\mathsf{ind}})$
6:     return $\sigma_{\mathsf{ind}}$
7: else
8:     return NULL
```

Algorithm 3 ETA Signature Verification [18]

```
$\underline{(0,1) \leftarrow \mathsf{ETA.Ver}(PK, M_{\mathsf{ind}}, \sigma_{\mathsf{ind}})}$:
1: if $\mathsf{ind} \geq K$ then
2:     return 0
3: $R'_{\mathsf{ind}} \leftarrow L^{H(M_{\mathsf{ind}}||\mathsf{ind}||x_{\mathsf{ind}})} \cdot \alpha^{s_{\mathsf{ind}}}$
4: if $pk_{\mathsf{ind}} = H(R'_{\mathsf{ind}})$ then
5:     return 1
6: else
7:     return 0
```

3.2 Schnorr Signature [19]

We leverage the Schnorr signature in our CORE-MMM algorithm.Schnorr key generation is extremely efficient (only takes takes constant time). The only operations that we do are generating large primes and public/private key pairs.

Algorithm 4 Schnorr Key Generation [18]

```
$\underline{(l, L) \leftarrow \mathtt{Schnorr.Kg}(1^{\kappa})}$:
1: $q$ and $p$ are large primes tha holds the following two conditions: (i) $p > q$ and (ii) $q|(p-1)$.
2: $\alpha$ is the generator of the subgroup $G$ of order $q$ in $\mathbb{Z}_p^*$.
3: return $(l \xleftarrow{\$} \mathbb{Z}_q^*, L \leftarrow \alpha^l \bmod p)$
```

Just like Schnorr key generation, Schnorr signature generation is extremely efficient as well. The leading cost in Schnorr signature generation is the exponentiation that is done by the second line. However, when using the Schnorr signature, one must be careful not to use the same *rand* value twice to avoid a potential key leakage [18]. Schnorr signature verification is also very fast (the leading cost is exponentiation).

Algorithm 5 Schnorr Signature Generation [18]

```
    $\underline{\sigma = (s, e) \leftarrow \texttt{Schnorr.Sig}(l, M)}$:
1: $rand \xleftarrow{\$} \mathbb{Z}_q^*$
2: $A \leftarrow \alpha^{rand} \bmod p$.
3: $s \leftarrow (r - H(M||A) \cdot l) \bmod q$
4: **return** $\sigma = (s, e = (H(M||A)))$.
```

Algorithm 6 Schnorr Signature Verification [18]

```
    $\underline{(0, 1) \leftarrow \texttt{Schnorr.Ver}(\langle s, e\rangle, L, M)}$:
1: $R' \leftarrow L^e \alpha^s \bmod p$.
2: **if** $e \neq H_0(M||R')$ **then**
3:     **return** 0
4: **return** 1
```

Before ending our discussion on Schnorr signatures, we would like to emphasize that we utilize the Schnorr signatures as if it is a one-time signature.

3.3 MMM Algorithm [5]

In our second scheme (which we name CORE-MMM), we use the unbounded forward-secure signature generation strategy presented in MMM. MMM is an algorithm to create unbounded forward-secure signatures. Any traditional signature algorithm could be inserted into the MMM to create an unbounded forward-secure signature. In our constructions, we created specific building blocks to optimize the unbounded forward-secure signature generation strategy presented in MMM to create the state-of-art compact and comprimise-resilient signature that we refer as CORE-MMM.

The construction of CORE-MMM will be explained in the proposed schemes section of the thesis. The MMM algorithm is a combination of two algorithms called sum ($\sum$) and product composition. In the upcoming sections, we will investigate how $\sum$ composition and product composition work independently and how their use creates the MMM algorithm.

3.3.1 $\sum$ Composition Algorithms in MMM [5]

The idea of $\sum$ composition is to create a forward-secure signature scheme by combining two traditional signature schemes. To give an example, let σ_0 and σ_1 be two traditional signature schemes that have T_0 and T_1 time periods respectively.

After these two signature schemes are combined via the $\sum$ composition algorithm, the resulting signature will be a forward-secure signature that can sign messages up to T time periods ($T = T_0 + T_1$). Note that in the given constructions, G() is the length doubling psedurandom generator.

Algorithm 7 $\sum$ Key Generation

```
    $\underline{(sk, PK) \leftarrow \sum.\text{Kg}(r, levels, i)}$ :
 1: Given random seed $r$, generate $\kappa$-bit randomness $(r_0, r_1)$
 2: **if** $levels = 0$ **then**
 3:     $(sk_0, pk_0) \leftarrow \text{SGN.Kg}(r_0)$
 4:     $(sk_1, pk_1) \leftarrow \text{SGN.Kg}(r_1)$
 5:     $PK \leftarrow H(pk_0 || pk_1)$
 6:     $sk \leftarrow (sk_0, r_1, pk_0, pk_1)$
 7:     **return** $(sk, PK)$
 8: $(sk_0, pk_0) \leftarrow \sum.\text{Kg}(r_0, levels-1, i+1)$
 9: $(sk_1, pk_1) \leftarrow \sum.\text{Kg}(r_1, levels-1, i+1)$
10: $PK \leftarrow \text{H}(pk_0 || pk_1)$,
11: $sk \leftarrow (sk_0, r_1, pk_0, pk_1)$
12: **return** $(sk, PK)$
```

To start off the discussion on the $\sum$ composition, we would like to firstly elaborate on the the key generation algorithm. The $\sum$ composition key generation in the MMM algorithm is a recursive algorithm. The idea is to recursively create a key for each node in the tree. The base case of the algorithm executes when the leaf nodes are created (when $levels = 0$). Note that in the generic $\sum$ composition construction, two different signature algorithms could be presented inside the key generation. However, for our construction we will be using only one signature (SGN).

The idea of the signature generation algorithm of $\sum$ composition is to sign a message depending on the period that it is in. To put it differently, in the generic $\sum$ composition algorithm presented in MMM, it is easy to see that if the total number of periods is T and the periods of the signatures that make up the sum composition each have $T/2$ periods, the first signature scheme will be used to sign messages in the first $T/2$ periods, whereas the second signature scheme will be used to sign messages in the remaining periods.

However, for our constructions, we do not leverage the combination of two basic signatures in our signature generation algorithm of the $\sum$ composition. Therefore, only one signature (SGN) is presented in our pseudocode.

The idea of the $\sum$ composition update algorithm is to traverse the entire $\sum$ composition tree and update the nodes according to the time periods. However, just like in the other algorithms, only one signature scheme (denoted as SGN) is necessary for our construction.

Continuing on, the $\sum$ Composition Signature Verification Algorithm is a recursive algorithm that traverses the tree and verifies each node. The base case of this algorithm is the execution of the leaves.

Algorithm 8 $\sum$ Signature Generation

```
    $\underline{(\sigma, t) \leftarrow \sum.\text{Sig}(t, sk = \langle sk', r_1, pk_0, pk_1\rangle, M, levels, i, T, SGN)}$ :
 1: if $levels = 0$ then
 2:     if $t < T/2$ then
 3:         $t' \leftarrow t$
 4:     else
 5:         $t' \leftarrow t - T/2$
 6:     $\sigma' \leftarrow$ SGN.Sig$(sk', M)$, $\sigma \leftarrow \langle\sigma', pk_0, pk_1\rangle$
 7:     return $(\sigma, t')$
 8: else
 9:     if $t < T/2$ then
10:         $\sigma' \leftarrow \sum.\text{Sig}(t, sk', t, M, levels{-}1, i{+}1)$
11:     else
12:         $\sigma' \leftarrow \sum.\text{Sig}(t{-}T/2, sk', t, M, levels{-}1, i{+}1)$
13:     $\sigma \leftarrow \langle\sigma', pk_0, pk_1\rangle$
14:     return $(\sigma, t)$
```

Algorithm 9 $\sum$ Update

```
    $\underline{\sum.\text{UP}(sk = \langle sk', r_1, pk_0, pk_1\rangle, t, levels, i, \text{SGN})}$:
 1: if t + 1 < $T$ then
```

```
2:     $sk^{'} \leftarrow$SGN.UP($sk^{'}$)
3: else if $(t+1=T)$ then
4:     ($sk^{'}$,$pk^{'}$)$\leftarrow$ SGN.Kg($r_1$)
5:     delete $pk^{'}$
6:     $r_1 \leftarrow 0$
7: else
8:     $sk^{'} \leftarrow$SGN.UP($sk^{'}$)
9: if $t < T/2$ then
10:     $\sum$.UP($sk_{'}, r_1, pk_0, pk_1$,$t$, $levels$$-1, i$+1)
11: else
12:     $\sum$.UP($sk_{'}, r_1, pk_0, pk_1, t$$-$$T/2$, $levels$$-1, i+1$)
```

Algorithm 10 $\sum$ Signature Verification

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

3.3.2 Product Composition Algorithm in MMM [5]

Just like in the $\sum$ composition algorithm, the product composition algorithm takes in two signature schemes. For simplicity, let σ_0 denote the first and σ_1 denote the second signature scheme. We will call the periods of these signatures as T_0 and T_1 respectively. The result of the product composition is a signature that has a time period of $T_0 \cdot T_1$. Furthermore, the responsibilities of σ_0 and σ_1 are vastly different than the $\sum$ composition. The actual messages are signed via σ_1. σ_0 is responsible for certifying the public key of the σ_1. Our pseudocode of our construction of CORE-MMM is actually the product construction itself.

That is why we decided not to provide any pseudocode in this section as our proposed scheme (CORE-MMM) clearly demonstrates the idea of product construction.

3.3.3 MMM Construction [5]

In MMM, both the $\sum$ and the product composition are utilized. MMM is simply a combination of the upper tree and the lower trees. Each of these trees (whether it be one of the lower trees or the upper tree) is the result of a sum composition. The lower trees are responsible for signing the actual messages. The upper tree is responsible for creating leaves that would certify each of the lower trees. The lower trees are created on the run as the messages keep accumulating. Therefore, MMM will minimize the number of lower trees that needs to be created on the run. This significantly improves the performance of MMM . MMM enjoys constant public keys along with logarithmic or constant costs only.

Chapter 4: Proposed Schemes

In this section of the thesis, we will present our proposed schemes named $CORE_{Base}^{K}$ and CORE-MMM.

4.1 $CORE_{Base}^{K}$ Signature Scheme

Our first scheme, which we denote as $CORE_{Base}^{K}$, can aggregate signatures for every K messages. It can also aggregate the public keys (which are transmitted through the signature) $(pk_j, \ldots, pk_{j'})$ as long as the messages are the same. $(\overrightarrow{M}_i, \overrightarrow{PK}_i)$ denotes ith elements of the message and public key vectors, respectively. $\sigma_{1,K}$ is the aggregate signature on messages sampled during K periods. We will investigate the key generation, signature generation and signature verification algorithms of $CORE_{Base}^{K}$.

Our key generation algorithm of $CORE_{Base}^{K}$ starts with generating two K-size hash chains $(\bar{r},\ \bar{y})$ to create public values $H(R_{1,K})$ and $H(Y_{1,K})$. The secret key of the scheme is both (the first input to the hash chain) (y_1, r_1) and a random variable (κ bit) whereas the public key is the hash of $R_{1,K}$ and $Y_{1,K}$. Note that for K messages, the leading cost is only two exponentiations. The following algorithm demonstrates the key generation procedure of $CORE_{Base}^{K}$.
The values that constitute the public key are important ($R_{1,K}$ and $Y_{1,K}$) because after all, they are checked for correctness in the verification algorithm.

Our signature generation algorithm of $CORE_{Base}^{K}$ leverages signature aggregation and conditional public key aggregation (when the message to be signed does not change for the given interval $(1 \leq j < j' \leq t)$). To the best of our knowledge, $CORE_{Base}^{K}$ is the first forward-secure signature that leverages both conditional public key aggregation in addition to signature aggregation.

Algorithm 11 $CORE^{K}_{Base}$ Key Generation

```
$\underline{(sk_1, PK) \leftarrow CORE^{K}_{Base}.Kg(1^{\kappa}, K):}$
1: $y_1 \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
2: $r_1 \overset{\$}{\leftarrow} \mathbb{Z}_q^*$
3: $Y_1 \leftarrow \alpha^{y_1} \bmod p$
4: $\bar{r} \leftarrow \sum_{i=1}^{K} r_i \bmod p$, where $r_{i+1} \leftarrow H_1(r_i)$
5: $\bar{y} \leftarrow \sum_{i=1}^{K} y_i \bmod p$, where $y_{i+1} \leftarrow H_1(y_i)$
6: $Y_{1,K} \leftarrow \alpha^{\bar{r}}$
7: $R_{1,K} \leftarrow \alpha^{\bar{y}}$
8: $x_1 \overset{\$}{\leftarrow} \{0,1\}^{\kappa}$
9: $sk_1 \leftarrow (y_1, r_1, x_1)$
10: $PK \leftarrow (R' = H_1(R_{1,K}), Y' = H_1(Y_{1,K}))$
```

Our signature generation algorithm is composed of three execution paths. The first execution path is when the new message is not the same as the previous message (lines 16-20). The corresponding signature is aggregated (line 24) while the size of the public key is increased by one. The second execution path is when the message to be signed is the same as the previous message, we leverage both signature and public key aggregation (lines 22-31). This means there will be both individual public keys and partially aggregated public keys. Furthermore, lines 24-28 are the update operations for the next iteration. Lastly, the third execution path is when lines 2 - 15 execute the sealing function for situations where the signer wants to verify the messages signed so far. A sealing message marking the time stamp of the final message and its index gets created. The signer then generates all private keys from it's current index to K, aggregates them, and the signs the sealing message with this aggregated private key. The signer then generates the corresponding public key for this aggregated private key, updates the list, and finalizes the K-time signature generation. This is done so that the public key vector in the key generation will match the one in the signature generation. Furthermore, an appropriate signature will be created to verify K messages.

Our signature verification algorithm for $CORE^{K}_{Base}$ does three checks: (i) a check on the variables' that attained their values in the signature generation algorithm. (ii) a check on whether the aggregated public key generated in the signature generation algorithm is the same with the one generated in the key generation. (iii) Finally, the secret key verification. Note that the signature verification is extremely efficient especially when the frequency of messages is small.

Algorithm 12 $CORE_{Base}^{K}$ Signature Generation

```
$\underline{(sk_j, \langle st_j, \sigma_{1,j} \rangle) \leftarrow CORE_{Base}^{K}.Sig(sk_j, \sigma_{1,j-1}, m_j, st_j, c):}$
1: Let $(i = j = j' = 1, c = 0)$, $m_1 = M_1$, $Y_1^{0,1} = 1$, $s_{1,0} = 0$ and $(\overrightarrow{M}, \overrightarrow{PK}, st_1) = NULL$
2: if $c = 1$ then
3:     $x_{i+1} \leftarrow H_3(x_i)$
4:     $i \leftarrow i + 1$
5:     $M_i \leftarrow$ "$seal||timestamp||j||K$"
6:     $y_{j,K} \leftarrow \sum_{l=j}^{K} H_1(y_l) \bmod q$, where $y_{l+1} \leftarrow H_1(y_l)$
7:     $r_{j,K} \leftarrow \sum_{l=j}^{K} H_1(r_l) \bmod q$, where $r_{l+1} \leftarrow H_1(r_l)$
8:     $s_{j,K} \leftarrow r_{j,K} - H_1(M_i||j+1||x_i) \cdot y_{j,K} \bmod q$
9:     $s_{1,K} \leftarrow s_{1,j-1} + s_{j,K} \bmod q$
10:    $Y_i^{j,K} \leftarrow \alpha^{y_{j,K}} \bmod p$
11:    Delete $(y_j, ., y_K, y_{j,K}, r_j, ., r_K, r_{j,K}, x_i, x_{i+1}, s_{j,K}, s_{1,j-1})$
12:    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
13:    $j \leftarrow K + 1$ and update $st_j = (\langle i, j-1, j' \rangle, \overrightarrow{M}, \overrightarrow{PK}))$
14:    $\sigma_{1,K} \leftarrow (s_{1,K}, x_1)$
15:    return the final state $(sk_j = \perp, \langle st_j, \sigma_{1,K} \rangle)$ and exit
16: else if $m_j \neq M_i$ then
17:    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 $st_{j+1} = (\langle i, j-1, j' \rangle, \overrightarrow{M}, \overrightarrow{PK})$
18:    $j' \leftarrow j$, $x_{i+1} \leftarrow H(x_i)$
19:    $i \leftarrow i + 1$, $M_i \leftarrow m_j$
20:    $s_j \leftarrow r_j - H_1(M_i||j'||x_i) \cdot y_j \bmod q$
21:    $Y_i^{j,j'} \leftarrow Y_j$
22: else
23:    $s_j \leftarrow r_j - H_1(M_i||j'||x_i) \cdot y_j \bmod q$
24:    $Y_i^{j,j'} \leftarrow Y_i^{j-1,j'} \cdot Y_j$, delete $Y_i^{j-1,j'}$ and $Y_j$
25: $s_{1,j} \leftarrow s_{1,j-1} + s_j \bmod q$
26: Delete $(s_{1,j-1}, s_j)$
27: $y_{j+1} \leftarrow H_1(y_j)$, $r_{j+1} \leftarrow H_1(r_j)$
28: Delete $(y_j, r_j)$
29: $Y_{j+1} \leftarrow \alpha^{y_{j+1}} \bmod p$, $j \leftarrow j + 1$
30: 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 $st_j = (\langle i, j-1, j' \rangle, \overrightarrow{M}, \overrightarrow{PK})$
31:    return the final output as $(sk_j = \perp, \langle st_j, \sigma_{1,K} = (s_{1,K}, x_1) \rangle)$
32: return $(sk_j = \perp, \langle st_j, \sigma_{1,j} = (s_{1,j}, x_i) \rangle)$
```

To put it differently, when the messages do not change with high frequency, the public key vector shrinks which in turn makes the signature verification extremely efficient.

Algorithm 13 $CORE^{K}_{Base}$ Signature Verification

```
$(0,1) \leftarrow CORE^{K}_{Base}.Ver(st, \sigma_{1,j}, PK)$:
1: if $j \geq 1$ & $j' \leq i$ & $i \leq j$ & $j-1 \neq K$ then
2:     return 0
3: else if $H_1(\prod_{l=1}^{i} \overrightarrow{PK}_l \bmod p) \neq Y'$ then
4:     return 0
5: 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
6:     return 0
7: else return 1
```

4.2 CORE-MMM Signature Scheme

We began with a naive $CORE^{K}_{Base}$ scheme and leveraged the MMM algorithm to extend our $CORE^{K}_{Base}$ signature scheme for signing unbounded number of messages. In our CORE-MMM construction (see fig 4.1), the upper tree leaves are Schnorr, the lower tree leaves are $CORE^{K}_{Base}$ signatures. Schnorr signatures are used to certify the lower trees while our $CORE^{K}_{Base}$ signatures sign the actual messages. This permits the use of aggregation capabilities of $CORE^{K}_{Base}$ along with making the certification process extremely efficient.

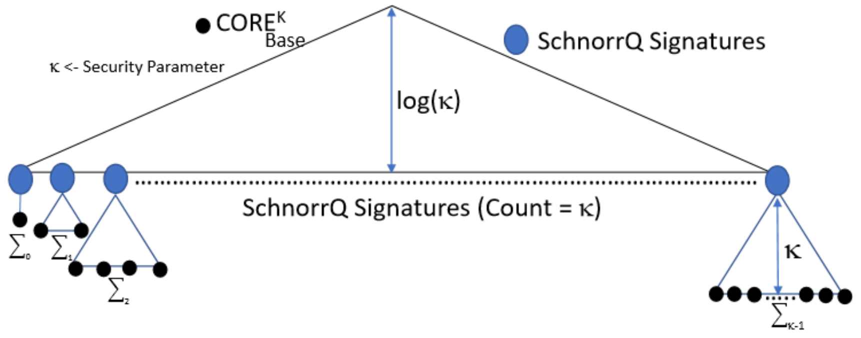

Figure 4.1 Demonstration of CORE-MMM.

As can be seen from the above figure, a mixture of $CORE^{K}_{Base}$ and Schnorr signatures are necessary to form CORE-MMM. The following subsections will discuss the key generation, signature generation, update and signature verification algorithms of CORE-MMM. Note that the lower trees are created on the run while the upper tree is fixed ($\log \kappa$ number of leaves).

In our key generation algorithm for CORE-MMM, every Schnorr signature, an instance of a $CORE^{K}_{Base}$ tree is generated. The height of this tree is kept track by the variable l. The height of the tree increases one by one in each iteration to handle more messages (please refer to [5] for details of the tree).

Algorithm 14 CORE-MMM Key Generation

```
    $\underline{(sk,\ PK) \leftarrow \texttt{CORE-MMM.Kg}(r)\text{:}}$
 1: $t_0 = 0$
 2: $t_1 = 0$
 3: $l = 1$
 4: $(r_0, r_1) \leftarrow \mathrm{G}(r)$
 5: $(r'_0, r'_1) \leftarrow \mathrm{G}(r_1)$
 6: $(sk_0, PK) \leftarrow \sum.\mathrm{Kg}(r_0, 7, 0, \mathsf{SchQ})$
 7: $(sk_1, pk_1) \leftarrow \sum.\mathrm{Kg}(r_0, l, 0, CORE^{K}_{Base})$
 8: $\sigma \leftarrow \sum.\mathrm{Sig}(0, sk_0, pk_1, l, 0, \mathsf{SchQ})$
 9: $sk_0 \leftarrow \sum.\mathrm{UP}(sk_0, 0, l, 0, \mathsf{SchQ})$
10: return $(sk{=}\langle sk_0, \sigma, sk_1, pk_1, r''_1 \rangle, PK)$
```

Note that each lower tree will have increasing heights. In the figure 4.1, we denoted that the Schnorr signatures will be placed on the root nodes of the lower trees and the remaining nodes on the lower tree will be $CORE^{K}_{Base}$. The Schnorr signature will be used to certify the lower trees and the $CORE^{K}_{Base}$ will be used to sign the messages (which will be discussed in detail on the upcoming sections).

Our signature generation algorithm of CORE-MMM is relatively simpler than the key generation process. In our scheme CORE-MMM, the $CORE^{K}_{Base}$ signing alorithm is used to sign the messages. For K messages an aggregated $CORE^{K}_{Base}$ signature is generated. This algorithm also increases the current period since a call to $CORE^{K}_{Base}$ is made.

Furthermore, our $CORE^{K}_{Base}$ has its own update operations embedded inside the signature generation algorithm (please see lines 24-28 of $CORE^{K}_{Base}$ signature generation). To not mistaken this algorithm with the update operations embedded in $CORE^{K}_{Base}$ signature generation we would like to clarify that this particular algorithm shows the update operations that are happening right after each leaf (i.e. right after each K messages have been processed) and right after each lower tree is processed.

Algorithm 15 CORE-MMM Signature Generation

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

The base case of this algorithm is the first if statement which executes after all the leaf nodes have been processed. Finally, our signature verification algorithm of CORE-MMM is a combination of $CORE^K_{Base}$ and SchnorrQ verifications.

Algorithm 16 CORE-MMM Update

```
    CORE-MMM.UP$(t, \langle sk = \langle sk_0, \sigma_0, sk_1, pk_1, r\rangle, t_0)$:
1:  if t + 1 $\neq$ 0 mod $T_1$ then
2:      $\sum$.UP$(sk_1, t \ mod \ T_1, t_0, 0, CORE^K_{Base})$
3:      return
4:  else
5:      $l = l + 1$
6:      $(r') \leftarrow G(r)$
7:      $(r) \leftarrow G(r)$
8:      $t_0 = t_0 + 1$
9:      $(sk_1, pk_1) \leftarrow \sum$.Kg$(r', l, i, CORE^K_{Base})$
10:     $\sigma \leftarrow \sum$.Sig$(\lfloor t/T_1 \rfloor, sk_0, M, t_0, 0, \mathsf{SchQ})$
11:     $sk_0 \leftarrow \sum$.UP$(sk_0, \lfloor t/T_1 \rfloor, t_0, 0, \mathsf{SchQ})$
```

Algorithm 17 CORE-MMM Signature Verification

```
    (0,1) $\leftarrow$CORE-MMM.Ver$(pk, M, \langle \sigma = \langle pk_1, \sigma, \sigma'\rangle, t, t_0, t_1)$:
1:  $B_0 \leftarrow \sum$.Ver$(pk_1, M, \sigma, \lfloor t/T \rfloor, t_0, 0, \mathsf{SchQ})$
2:  $B_1 \leftarrow \sum$.Ver$(pk_1, M, \sigma', t-T, t_1, 0, CORE^K_{Base})$
3:  return $B_0$ & $B_1$
```

The reason is that SchnorrQ signatures certify the lower tree and $CORE^K_{Base}$ certifies the individual messages (verifies messages K at a time).

4.3 CORE-MMM Design: Our Choice of $CORE_{Base}^{K}$ and $Schnorr$ Signatures

Our $CORE_{Base}^{K}$ scheme is specifically tailored to maximize the efficiency of the unbounded forward-secure signature generation methodologies. Following are the distinguishing features of $CORE_{Base}^{K}$ among it's counterparts: (i) $CORE_{Base}^{K}$ has very compact signatures and public keys, which helps to minimize the signature and secret key size, when it is initialized with MMM. In MMM, the public key = hash output, and base public keys have logarithmic storage as the secret key which becomes the signature. $CORE_{Base}^{K}$'s compact the public key size (32 Bytes), coupled with the conditional aggregation on its signature provides storage benefits tailored for MMM. For instance, CORE-MMM has 56× and 1.22× smaller secret key sizes; and 58365× and 111× more compact signature sizes (when $f = 2^3$) compared to HORS and SchnorrQ respectively. (ii) $CORE_{Base}^{K}$ has the best key generation performance compared to it's counterparts (only two exponentiations for K messages). This is critical in MMM because the cost of update operations are dominated by the key generation cost. (iii) Unlike other store and forward systems that rely on batch signing, $CORE_{Base}^{K}$ has an optional sealing feature where the messages that are processed so far could be authenticated by aggregating the remaining public keys and creating a single signature. The dominating cost of this process is only one exponentiation. (iv) $CORE_{Base}^{K}$ signature verification is also extremely efficient, depending on the frequency of messages changed (f). Compared to it's most efficient counterpart (HORS-MMM), $CORE-MMM$ has 51× faster verification when $f = 2^3$.

Furthermore a use-case diagram for our CORE cryptographic tool is presented. Note that in these diagrams, the end device can be anything ranging from a security monitoring camera, to a wireless sensor or a personal bank account system.

The use-case diagram in figure 4.2 demonstrates how the end-devices create signatures for the given messages. To give a real-world example, we can think of the end-device as a security camera and the messages as camera snapshots that need to be signed.

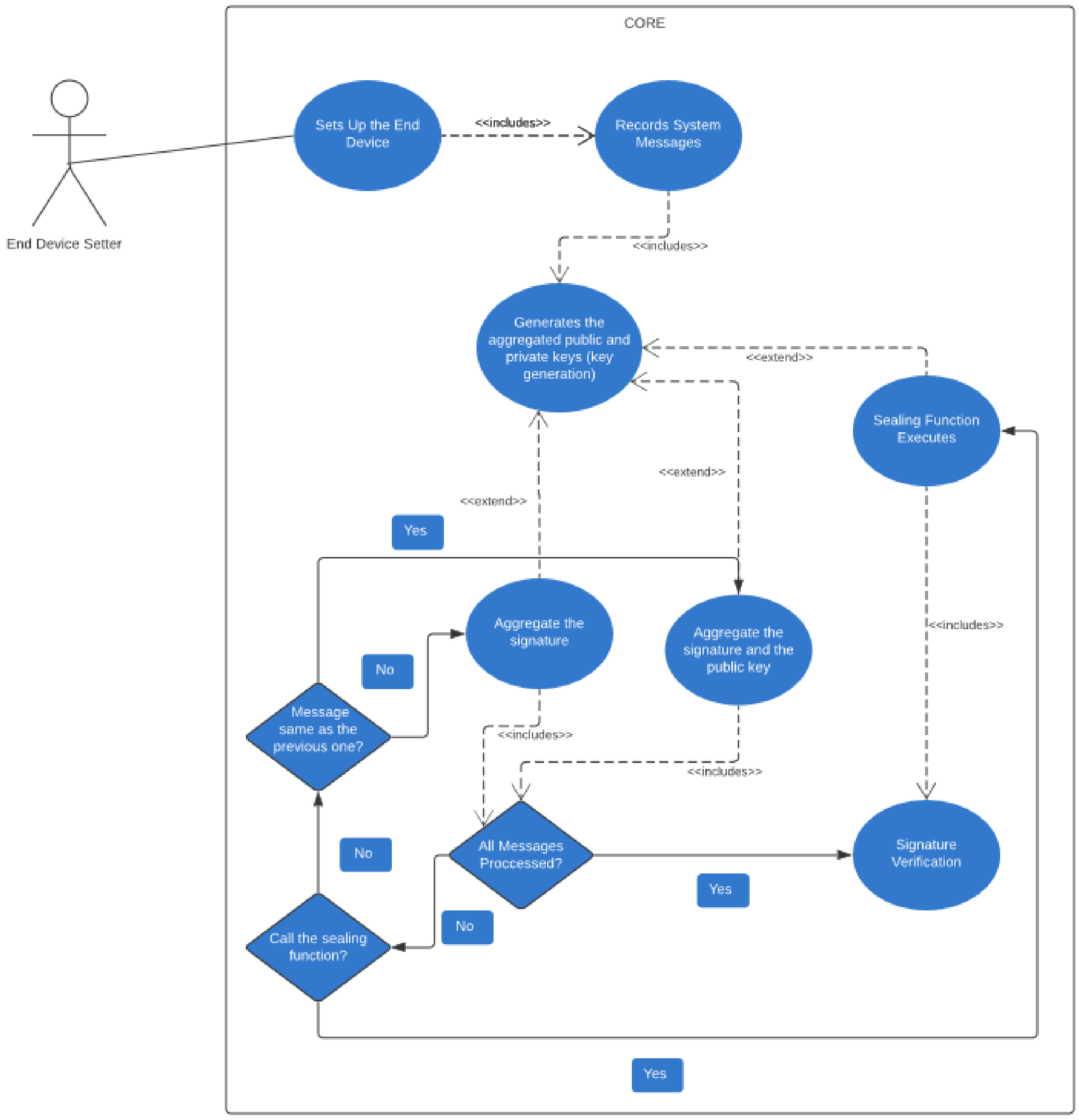

Figure 4.2 The 'verify messages' use-case of the CORE schemes.

Chapter 5: Models

In this chapter of the thesis, we will elaborate on our store-and-forward system model (with an illustration on figure 5.1) and use-cases.

5.1 The System Model of CORE

Our signature schemes ($CORE^{K}_{Base}$ and CORE-MMM) embody the "store-and-forward" model. The idea in a "store-and-forward" model is that K messages (and its corresponding signatures) need to be processed before verification can take place. In other words, to verify a specific signature, one needs to wait for all K signatures to be created.

An important point about our system model is that, we assume honest signers. Furthermore, any stakeholder in our model can do the verification.

We also assume that message changes (system state alterations) do not occur frequently. This permits the use of public-key aggregation capabilities in our schemes. Without the public-key aggregation capabilities, our schemes may not be the best option for real-time systems where a signature needs to be verified as soon as it is created.

Furthermore, we build our system considering a very capable adversary. In our model, the adversary can get the secret-key of our signatures at any time-period s/he wishes. In our assumption, an attacker will be permitted to do any malicious activity it can with the secret key it compromises. These malicious activities may include: forging, modifying and or deleting signatures.

5.2 Use Cases of CORE

Consider a security monitoring application (e.g. a security camera) of a relatively inactive venue. The messages that need to be signed by the application are the same for many intervals. This favors the use of our applications because of the low state transitions. Our schemes will provide significant space advantages for such situations.

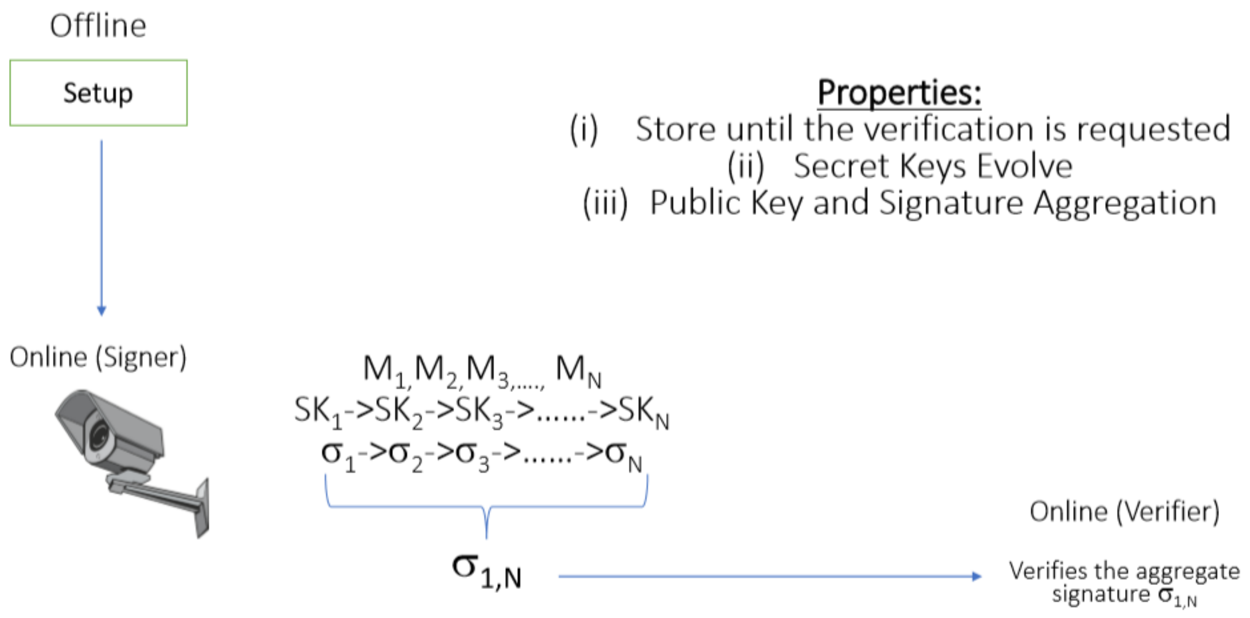
Offline
Setup
Properties:
(i) Store until the verification is requested
(ii) Secret Keys Evolve
(iii) Public Key and Signature Aggregation
Online (Signer)
$M_1, M_2, M_3, \ldots, M_N$
$SK_1 \to SK_2 \to SK_3 \to \ldots \to SK_N$
$\sigma_1 \to \sigma_2 \to \sigma_3 \to \ldots \to \sigma_N$
$\sigma_{1,N}$
Online (Verifier)
Verifies the aggregate signature $\sigma_{1,N}$

Figure 5.1 Model definition of our scheme.

The above model illustrates our system model. The end device (e.g. security camera) takes as input messages $\mathsf{M}_1, \mathsf{M}_2, ..$ and a corresponding signature is generated for each message. We aggregate all of these corresponding signatures to create one compact signature. Public keys can also be aggregated with the condition that consecutive messages do not change.

Chapter 6: Performance Analysis

In this chapter of the thesis, we will show the comparison results of our schemes (both $CORE_{Base}^{K}$ and CORE-MMM) with their state-of-art counterparts (both analytically and experimentally). We will also disclose our evaluation metrics and explain in detail how we conducted the experiments.

6.1 Evaluation Metrics with Hardware and Software Specifications

Our schemes are compared with their state-of-art counterparts in terms of public and private key sizes, signature sizes, key and signature generation times and signature verification times. We assume K (the number of messages $CORE_{Base}^{K}$ or any other K-time scheme can accommodate) to be 2^{10}. We also denote the frequency in which our messages change via f.

Our implementation utilizes the FourQlib[1] library. The reason we chose this library is because elliptic curve operations are very fast via this library. BLAKE2b [20] was used for hash functions (because it is very fast specifically on laptops).

Our implementation can be found at [2]:

www.github.com/efeUlasAkay/COREBASE

We did our experiments on commodity hardware (a laptop). Our laptop had 12 GB of random access memory (RAM) along with the Intel i7-6700HQ 2.6 GHz central processing unit (CPU).

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

[2]Our implementation will be open source but currently it is on a private repository for confidentiality purposes. Implementation can be disclosed to the thesis committee upon request

6.2 Performance Assessment of our Schemes (comparison with the state-of-art)

In this part of the thesis, we will provide a performance assessment on our schemes by comparing it against its state-of-art counterparts.

In general, when analyzing the performance of a digital signature, the key generation cost of can be overlooked because key-generation is a one-time operation as well as it is considered to be an offline operation. However, the key generation cost of $CORE^{K}_{Base}$ highly influences the update cost of our CORE-MMM scheme. This is the primary reason why we included the key generation cost of our $CORE^{K}_{Base}$ scheme in our performance tables. As the tables 6.1 and 6.2 demonstrate, $CORE^{K}_{Base}$ has the best key generation cost among its counterparts (two exponentiations for K messages).

Our findings on tables 2 and 3 also show that $CORE^{K}_{Base}$ offers very small signature sizes. This is because of the signature aggregation feature of our $CORE^{K}_{Base}$ (please see the proposed schemes section for details). Compared to its state-of-art alternatives, $CORE^{K}_{Base}$ is among the best alternatives in terms of signature size when f is relatively small (message changes are relatively less). The main competitors of $CORE^{K}_{Base}$ in term of signature size are the `BAF` and `FssAgg-BLS` signatures.

Signature verification is another metric that $CORE^{K}_{Base}$ is extremely efficient in. The fastest state-of-art competitor of $CORE^{K}_{Base}$ in terms of the signature verification time is the HORS signature. Considering our use-cases, $CORE^{K}_{Base}$ does 180x faster verification than the HORS signature.

$CORE^{K}_{Base}$ also has very compact public keys (64 KB). The smallest public-key counterpart of $CORE^{K}_{Base}$ is the SchnorrQ signature (32MB). Note that the reason our $CORE^{K}_{Base}$ is able to achieve small public key size is due to the fact that we aggregate all the K public keys in the key generation phase. Note that the size of our public key is always constant regardless of the message frequency change rate. Other metrics such as signature generation time and private key size are competitive (even better in some comparisons) compared to state-of-art counterparts of $CORE^{K}_{Base}$. Considering all of these advantages, we argue that

$CORE^{K}_{Base}$ is the best signature alternative in our system model.Taking these results into account, we believe that our $CORE^{K}_{Base}$ signature could potentially be very fitting in other system models that are out of the scope of this thesis as well.

In tables 6.3 and 6.4 we give analytical as well as experimental performances of CORE-MMM. We also evaluate and compare the state-of-art MMM instantiations with CORE-MMM. To the best of our knowledge, our performance analysis shows that our construction CORE-MMM is the best forward-secure unbounded signature option among its counterparts. Before analyzing specific performance metric of CORE-MMM, we would like to note that the performance of MMM depends on the signatures signed so far (which we denote by t). In our analysis, we consider $t = 2^{30}$. To elaborate on what this means, with $t = 2^{30}$, it is possible to sign 3.4 messages every second for 10 years. Therefore, it is safe to say that $t = 2^{30}$ is a sufficient number for our schemes.

Table 6.1 Private/public key sizes, signature size and signature generation/verification costs of CORE and its counterparts

Scheme	**Signer**		
	Key Generation Time	**Private Key Size**	**Signature Size**
HORS [21]	$2 \cdot \kappa \cdot K \cdot H$	κ	$K \cdot \kappa \cdot u$
SchnorrQ [22]	$K \cdot EMul$	$\|q\|$	$K \cdot 2\|q\|$
XMSS [23]	$K \cdot (3 + l \cdot (w + 2)) \cdot H$	κ	$K \cdot (l + \|K\|) \cdot \|H\|$
*`BAF` [16]	$2 \cdot K \cdot EMul$	$4\|q\|$	$2\|q\|$
*`FssAgg-BLS` [11]	$K \cdot EMul$	$\|q\|$	$\|q\|$
*$CORE^{K}_{Base}$	$2 \cdot EMul$	$2\|q\| + \kappa$	$\|q\| \cdot f + 2\|q\|$

Table 6.1 (Continued)

Scheme	**Signer**	**Verifier**	
	Signing Time [‡]	**Public Key Size**	**Verification Time**
HORS [21]	$(u+1) \cdot H$	$t' \cdot \|H\| \cdot K$	$K \cdot (u+1) \cdot H$
SchnorrQ [22]	$EMul$	$K \cdot \|q\|$	$K \cdot 1.3 \cdot EMul$
XMSS [23]	$(((\|K\|+2) \cdot (\|K\| +l \cdot (w+2)))/2 + 4 \cdot \|K\|) \cdot H$	$(2(\|K\|+\|l\|) +1) \cdot \|H\|$	$K \cdot (\|K\|+l \cdot (w+1)) \cdot H$
*BAF [16]	$Mulq + 4H$	$(4K-1) \cdot \|q\|$	$K \cdot EMul$
*FssAgg-BLS [11]	$2H + EMul + Mul_q$	$K \cdot \|q\|$	$K \cdot PR$
$*CORE^K_{Base}$	$EMul + ((K-f)/K) \cdot Eadd$	$2\|q\|$	$(1.3+f) \cdot EMul$

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 (elliptic curve) 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 [21]. w is the Winternitz parameter and l is the tree parameter in XMSS [23]. PR denotes for the pairing cost (a1 curve is used for FssAgg-BLS). For HORS [21] and SchnorrQ. [22], 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.

One of the impressive metrics for our scheme CORE-MMM is the signature size. In MMM constructions the public key goes inside of the signature. Thus, CORE-MMM's signature size becomes very compact compared to its counterparts since $CORE^K_{Base}$ has very optimal public keys (64 KB).

Another impressive metric of our CORE-MMM scheme is the update time it has. Note that the update time of the MMM schemes is highly dependent on the key generation algorithm of their base schemes. Since $CORE^K_{Base}$ has the best key generation time, the update time of CORE-MMM becomes extremely more efficient than its counterparts. CORE-MMM also has the best private key size compared to its counterparts (thanks to the underlying $CORE_{Base}^{K}$ algorithm we use). We should also note that the public key sizes are constant for all MMM schemes. This is optimization is due to underlying structure of the MMM algorithm. It does not have anything to do with our design choices. As a note, we would like to stress that the efficiency of CORE-MMM is also competitive in signature generation. Furthermore, CORE-MMM's signature verification time is also the best option among its counterparts. This efficiency should be attributed to the fast signature verification capability of the $CORE_{Base}^{K}$ signatures. Please also note that the reason why CORE-MMM's signature verification time gets better as the message change frequency gets lower is thanks to the $CORE_{Base}^{K}$ signature. In the proposed schemes section of the thesis, we have illustrated that the size of the public key vector shrinks as the message change rate decreases. This fact reduces the running time of the signature verification algorithm of $CORE_{Base}^{K}$ signature. As the verification time of $CORE_{Base}^{K}$ signature gets better, it reduces the verification time of CORE-MMM in-turn. Another optimization was related to the integration of the SchnorrQ signatures in our CORE-MMM scheme.

Table 6.2 Experimental performance comparison of CORE and its counterparts ($K = 2^{10}$)

Scheme	f	**Key Generation Time (μs)**	**Signing ‡ Time (μs)**	**Private Key Size (Byte)**
HORS [21]	N/A	36700.16	6.47	16
SchnorrQ [22]		10362.88	10.89	32
XMSS [23]		55728	322.41	16
*`BAF` [16]		20725.76	1.21	128
*`FssAgg-BLS` [11]		10362.88	911.02	32
*$CORE_{Base}^{K}$	1	**27.38**	11.11	80
	2^3		10.87	
	2^9		10.61	
	2^{10}		10.28	

Table 6.2 (Continued)

Scheme	f	**Signature Size (KB)**	**Verification Time** (ms)	**Public Key Size** ‡
HORS [21]	N/A	384	3.60	32 MB
SchnorrQ [22]		64	21.69	32 KB
XMSS [23]		2 368	45.63	1 056 Byte
*`BAF` [16]		0.0625	16.21	131040 Byte
*`FssAgg-BLS` [11]		0.03125	11250.69	32768 Byte
*$CORE^K_{Base}$	1	**0.09375**	**0.02**	**64 Byte**
	2^3	**0.3125**	**0.15**	
	2^9	16.0625	8.52	
	2^{10}	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.

We leveraged the SchnorrQ signatures to verify each of the lower trees. The reason we chose SchnorrQ instead of $CORE^K_{Base}$ to verify each of the sum composition trees is because $CORE^K_{Base}$ is a K-time signature and only one message needs to be verified by the top node of the sum composition node.

To give a more concrete example, the figure below [3] comparing the space efficiency that could be gained by using our forward-secure and aggregate signature scheme vs traditional digital forensic mechanisms is presented.

In our example, there are 2000 messages to be signed. It is easy to see that using the $CORE^K_{Base}$ scheme is much more efficient than using SchnorrQ. In a real-world security camera, many more snapshot of images may be necessary. This would increase the advantage we would be getting by using $CORE^K_{Base}$. Even though we provide numeric values for the comparison $CORE^K_{Base}$ and SchnorrQ in the figure 6.1, similar efficiency gains would be

[3]The copyright-free camera image is taken from "pixabay"

obtained if we compared CORE-MMM and any other MMM based scheme. It should be noted that figure 6.1 is suitable for our system model because the message changes are not that much. There is only one messages that is in between the message space. Note that when the message changes are not done, our schemes leverage public key aggregation. That is why it is possible to see from our example that the public key of the $CORE^{K}_{Base}$ scheme is much more compact compared to SchnorrQ. Table 6.2 demonstrates how we can compare the cost of $CORE^{K}_{Base}$ with other state-of-art signature schemes as well. To do a comparison and decide which signature would provide good performance to a system, the message change frequency (f) and K need to be adjusted. We show that $CORE^{K}_{Base}$ scheme is the best option for many metrics including key generation, signature size, signature verification time and public key size in our use-cases.

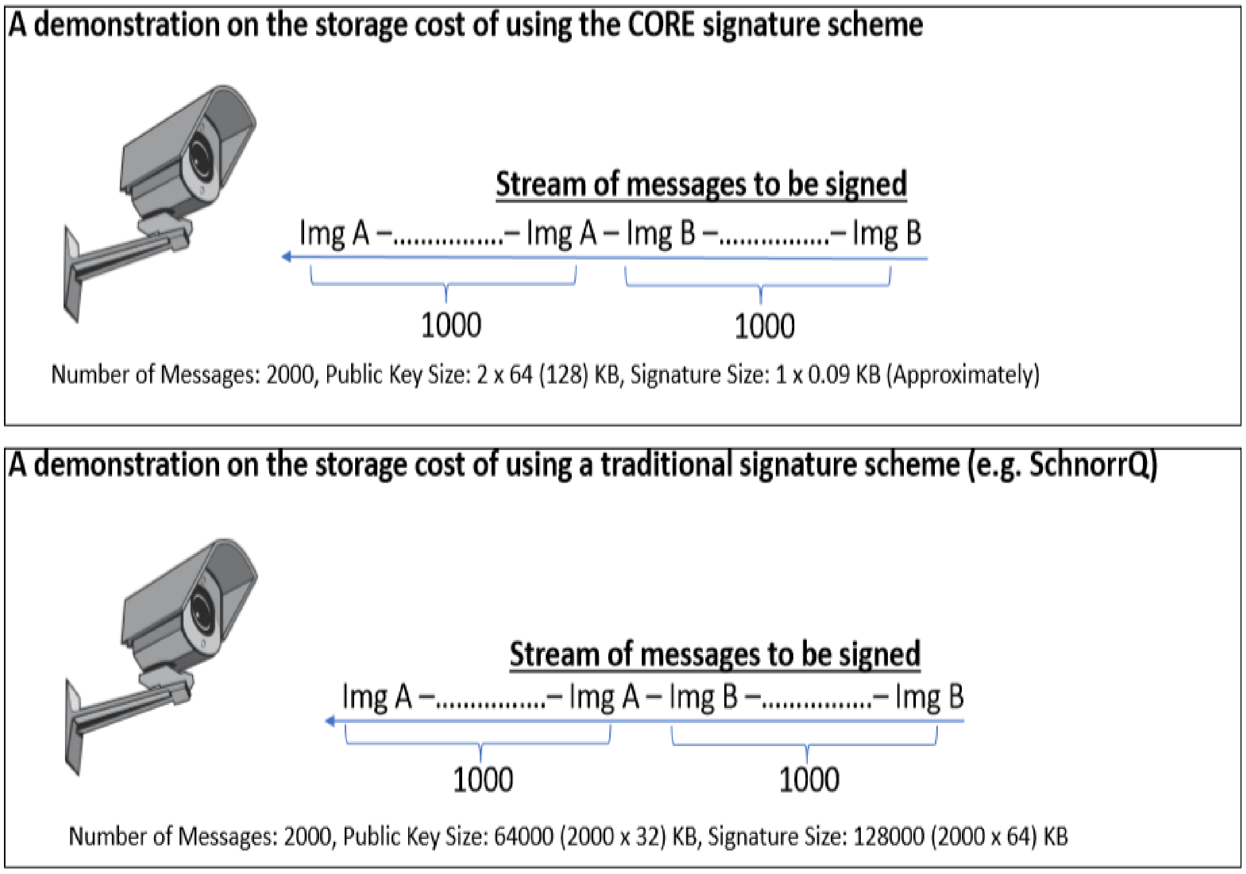
A demonstration on the storage cost of using the CORE signature scheme
Stream of messages to be signed
Img A –................– Img A – Img B –................– Img B
1000
1000
Number of Messages: 2000, Public Key Size: 2 x 64 (128) KB, Signature Size: 1 x 0.09 KB (Approximately)
A demonstration on the storage cost of using a traditional signature scheme (e.g. SchnorrQ)
Stream of messages to be signed
Img A –................– Img A – Img B –................– Img B
1000
1000
Number of Messages: 2000, Public Key Size: 64000 (2000 x 32) KB, Signature Size: 128000 (2000 x 64) KB

Figure 6.1 Use of $CORE^{K}_{Base}$ vs SchnorrQ signatures in an end device.

Table 6.3 Analytical comparison of CORE-MMM and its standard unbounded forward-secure counterparts

Scheme	**Signing** ‡ **Time** (μs)	**Private Key Size (Byte)**	**Update Time**
HORS-MMM	$(u+1)\cdot H$	$(5\|\kappa\|+3log(t)+6t'+2)$ $\cdot\|H\|+(2+u)\cdot\kappa$	$2\cdot\kappa\cdot H$
SchnorrQ-MMM	$EMul$	$(5\|\kappa\|+3log(t)+2)$ $\cdot\|H\|+10\|q\|$	$EMul$
BAF-MMM	$Mulq+4H$	$(5\|\kappa\|+3log(t/K)+2)$ $\cdot\|H\|+(12K+4)\cdot\|q\|$	$2\cdot EMul$
CORE-MMM	$EMul$ $+((K-f)/K)\cdot Eadd$	$(5\|\kappa\|+3log(t/K)+2)$ $\cdot\|H\|$ $+14\|q\|+\kappa$	$(2/K)\cdot EMul$

Scheme	**Signature Size (Byte)**	**Verification Time** (μs)	**Public Key Size (Byte)**
HORS-MMM	$K\cdot((2\|\kappa\|+2\|log(t)+4t'+1)$ $\cdot\|H\|+2\kappa\cdot u)$	$K\cdot(2u+2)\cdot H$	$\|H\|$
SchnorrQ-MMM	$K\cdot((2\|\kappa\|+2log(t)+1)\cdot\|H\|$ $+8\|q\|)$	$K\cdot 2.6\cdot EMul$	$\|H\|$
BAF-MMM	$(2\|\kappa\|+2log(t/K)+1)\cdot\|H\|$ $+(8K+4)\cdot\|q\|$	$(1.3+K)\cdot EMul$	$\|H\|$
CORE-MMM	$(2\|\kappa\|+2log(t/K)+1)$ $\cdot\|H\|$ $+(10+f)\cdot\|q\|$	$(2.6+f)\cdot EMul$	$\|H\|$

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 6.4 Performance comparison of CORE-MMM and its standard unbounded forward-secure counterparts

Scheme	f	**Signing** [‡] **Time** (μs)	**Private Key Size (Byte)**	**Update Time** (μs)
HORS-MMM	N/A	3.51	201088	35.84
SchnorrQ-MMM		10.22	4384	10.12
BAF-MMM		1.21	396448	20.24
CORE-MMM	1	11.11	**3568**	**0.03**
	2^3	10.87		
	2^9	10.61		
	2^{10}	10.28		

Scheme	f	**Signature Size (KB)**	**Verification Time** (ms)	**Public Key Size (Byte)**
HORS-MMM	N/A	134240	7.21	32
SchnorrQ-MMM		2656	44.56	32
BAF-MMM		257.8	16.23	32
CORE-MMM	1	**2.1**	**0.04**	32
	2^3	**2.3**	**0.17**	
	2^9	**18**	**8.54**	
	2^{10}	**34**	17.09	

The costs of MMM instantiations are calculated based on the base schemes and analytical performances as in Table 2 (parameters are as in Table 3). Recall that the signature verification is given for K items in all compared schemes.

Chapter 7: Conclusion

Traditional signature schemes provide security guarantees, but their storage overhead becomes a bottleneck especially for resource-constrained devices. Furthermore, the limitation on the number of messages that the traditional signature schemes can sign is an impactful limitation because many IoT devices cannot tolerate manual resetting.

To overcome these problems, we developed two forward-secure signatures which we name $CORE^{K}_{Base}$ and CORE-MMM. To the best of our knowledge, our $CORE^{K}_{Base}$ signature scheme is the first forward-secure signature scheme that leverages public key and signature aggregation. These aggregation capabilities of $CORE^{K}_{Base}$ provides extra storage efficiencies that would especially be useful for resource-constrained devices. We fully implemented and extensively tested our $CORE^{K}_{Base}$ scheme. Our second signature which we denote as CORE-MMM is a scheme that can sign practically unbounded number of messages. CORE-MMM leverages our first scheme $CORE^{K}_{Base}$ due to the fact that $CORE^{K}_{Base}$ is crafted to optimize the generic MMM construction. We hope that the optimizations we make in the base scheme of the MMM construction will serve as a foundation for future discoveries of unbounded forward-secure signatures.

Chapter 8: Future Work

In this chapter of the thesis, we will be discussing the possible improvements that could be made to our signature schemes.

We would like to experiment with our schemes on real end-devices to get a better understanding of how much storage/execution efficiency we would be able to get in a real-world setting. For instance, integration of our signature schemes on a security camera would shed light on how much storage efficiency our signature schemes would be able to get in a setting where the frequency of message changes are not uniform.

Another important point is that, even though we have done the full implementation and performance analysis of our $CORE_{Base}^{K}$ scheme on the commodity hardware, due to time constraints we have estimated our CORE-MMM scheme. An implementation of our CORE-MMM scheme on commodity hardware would very helpful for future real-world deployments of CORE-MMM.

Finally we would like to stress that our current signature schemes are not resistant to quantum attacks. A potential improvement would be to create signatures that would be resilient to quantum attacks while preserving at least some of the advantages of $CORE_{Base}^{K}$ and CORE-MMM.

References

[1] G. Itkis and L. Reyzin, “Forward-secure signatures with optimal signing and verifying,” in *Advances in Cryptology (CRYPTO ’01).* Springer-Verlag, 2001, pp. 332–354.

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

[3] M. Bellare and B. S. Yee, “Forward-security in private-key cryptography,” in *Proceedings of the The Cryptographers Track at the RSA Conference (CT-RSA ’03)*, 2003, pp. 1–18.

[4] 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.

[5] 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.

[6] Z. N. J. Peterson, R. Burns, G. Ateniese, and S. Bono, “Design and implementation of verifiable audit trails for a versioning file system,” in *Proceedings of the 5th USENIX Conference on File and Storage Technologies*, ser. FAST ’07. USA: USENIX Association, 2007, p. 20.

[7] R. S. Shilong, R. T. Snodgrass, S. S. Yao, and C. Collberg, “Tamper detection in audit logs,” in *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB ’04)*, 2004, pp. 504–515.

[8] 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.

[9] R. Anderson, “Two remarks on public-key cryptology, invited lecture,” Proceedings of the 4th ACM conference on Computer and Communications Security (CCS ’97), 1997.

[10] M. Bellare and S. Miner, “A forward-secure digital signature scheme,” in *Advances in Crpytology (CRYPTO ’99).* Springer-Verlag, 1999, pp. 431–448.

[11] 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.

[12] 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.

[13] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, “Sequential aggregate signatures and multisignatures without random oracles,” in *Proc. of the 25th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT ’06)*. Springer-Verlag, 2006, pp. 465–485.

[14] M. Bellare and G. Neven, “Multi-signatures in the plain public-key model and a general forking lemma,” in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS ’06. New York, NY, USA: ACM, 2006, pp. 390–399.

[15] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the weil pairing,” in *Advances in Cryptology — ASIACRYPT 2001*, C. Boyd, Ed. Springer Berlin Heidelberg, 2001, pp. 514–532.

[16] 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.

[17] 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.

[18] A. A. Yavuz and M. O. Ozmen, “Ultra lightweight multiple-time digital signature for the internet of things devices,” *IEEE Transactions on Services Computing*, pp. 1–1, 2019.

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

[20] 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

[21] 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.

[22] 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.

[23] 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