ABSTRACT

YAVUZ, ATTILA ALTAY. Efficient, Compromise Resilient and Compact Cryptographic Constructions for Digital Forensics. (Under the direction of Dr. Peng Ning.)

Audit logs are a fundamental digital forensic mechanism for providing security in computer systems; they are used to keep track of important events regarding the system activities. In current large distributed systems, protecting audit logs is a challenging task, especially in the presence of active attackers. It is critical for such a system to be compromise-resilient (i.e., having forward security and append-only integrity properties) such that when an attacker compromises a logging machine, she cannot forge or selectively delete the log entries accumulated before the compromise. Unfortunately, existing secure audit logging schemes have limitations that make them impractical for real-life applications: On the one hand, the symmetric schemes are not publicly verifiable, demand high storage, require online Trusted Third Party (TTP) support, and are also vulnerable to certain attacks. On the other hand, Public Key Cryptography (PKC)-based schemes require several Expensive Operations (ExpOps) (e.g., pairing), and thus are impractical for task-intensive and/or resource-constrained systems.

In this dissertation, we address the above problems by developing a series of novel cryptographic constructions that achieve the most desirable properties of both symmetric and PKC-based schemes simultaneously.

First, we propose a new class of signature schemes for Unattended Wireless Sensor Networks (UWSN) called Hash-Based Sequential Aggregate and Forward-Secure Signature (HaSAFSS). Using existing verification delays as an opportunity to introduce asymmetry, HaSAFSS schemes achieve high efficiency, while still preserving public verifiability, forward security and compactness. The HaSAFSS schemes are the only schemes in which both signers and verifiers get equal benefits of computational efficiency. Symmetric HaSAFSS (Sym-HaSAFSS) and Elliptic Curve Cryptography-based HaSAFSS (ECC-HaSAFSS) achieve the optimal (constant) signer and optimal verifier storage efficiency, respectively. Self-SUstaining HaSAFSS (SU-HaSAFSS) achieves an optimal storage at both the signer and the verifier sides by introducing a little more computation overhead.

Second, we develop a novel forward-secure and aggregate signature scheme called Blind-Aggregate-Forward (BAF) to address the secure audit logging needs of resource-constrained devices. BAF can address both real-time and non-real-time applications by achieving the public verifiability without requiring any online TTP support or time factor. BAF is the only scheme that can produce a publicly verifiable signature with very low computational, storage, and communication costs for the loggers. Moreover, a variant of BAF (i.e., Fast-Immutable BAF) enables fine-grained log verification by preserving the optimal logger efficiency and security.

Third, we propose a new signature scheme called Log Forward-secure and Append-only Signature (LogFAS) to address the secure logging needs of task-intensive applications with a large number of

loggers. LogFAS is the only secure audit logging scheme that can verify L log entries with always a small and constant number of ExpOps regardless of the value of L. It is also the only alternative in which each verifier stores only a small and constant size public key independent from the number of loggers and the number of log entries to be verified. In addition, a variation of LogFAS can identify the corrupted log entries with a sub-linear number of ExpOps when most entries are intact.

We prove that all of our schemes are secure under appropriate computational assumptions (in the random oracle model). We also show that they are significantly more efficient and practical than all the previous cryptographic secure audit logging schemes.

Efficient, Compromise Resilient and Compact Cryptographic Constructions for Digital Forensics

by
Attila Altay Yavuz

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2011

APPROVED BY:

Dr. Huaiyu Dai	Dr. Douglas S. Reeves
Dr. Ting Yu	Dr. Peng Ning Chair of Advisory Committee

DEDICATION

Gratefully devoted to my mother Ayse Yavuz by heart, who left her family and country behind to help me recover my health problems in US. She always gives her best to me in each aspect, and even her existence here is an invaluable support for me. I am also grateful to my father Dr. Ilhami Yavuz for always supporting me in any mean. His supports are invaluable. I also would to like to thank my elder sister Dr. Lale Yavuz for her morale support. Without the support of my family, this work would be impossible.

BIOGRAPHY

Attila A. Yavuz was born in Ankara, Turkey. He received his M.S. degree from the Department of Computer Science, Bogazici University, in 2006, and his B.S. degree from the Department of Computer Science and Engineering, Yildiz Technical University, in 2004, both in Istanbul, Turkey. He is interested in design, analysis and applications of cryptographic primitives and protocols to enhance the security of computer networks and systems. His current research focuses on the development of efficient cryptographic primitives to provide the security in digital forensics and wireless networks.

ACKNOWLEDGEMENTS

First, I am grateful to my PhD advisor Dr. Peng Ning, who devotes a significant effort and time to help me for improving my research and presentation skills. I have a special thank to Dr. Michael Reiter, who provided me an invaluable training in the theoretical cryptography. I thank my committee members, alphabetically, Dr. Huaiyu Dai, Dr. Douglas Reeves and Dr. Ting Yu for their valuable suggestions to improve my dissertation.

I would like to thank my officemates Sangwon Hyun, Jung Ki So, Young-Hyun Oh, Dr. An liu, Dr. Young Hee Park and Yao Liu for their supports.

I would like to thank to all of my friends at North Carolina State University. I am only able to list a few here: Mustafa Berke Yelten, Can Babaoglu, Namik Temizer, Dr. Nurcan Tezcan, Harun Demirci, Hicran Koc, Elif Karayaka...

Finally, I would like to express my deepest gratefulness to my family. Their endless support, help, encouragement and patience cannot be described with any word!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Audit logs are used to track important events such as user activities and program executions in modern computer systems, providing invaluable information about the state of the systems (e.g., intrusions, crashes). Due to their forensic value, audit logs are an attractive target for attackers. For instance, an experienced attacker may erase traces of her malicious activities from the logs, or modify the log entries to implicate other users after compromising the system. Therefore, ensuring the integrity, authenticity and accountability of audit logs is critical for any modern computer system [38, 57, 79, 104].

In current large and ubiquitous computer systems (e.g., Wireless Sensor Networks (WSN) [121]), protecting audit logs is a difficult task, especially in the presence of active adversaries. In particular, secure audit logging in task-intensive [79] and/or resource-constrained applications (e.g., WSNs [76], RFID tags [9]) is even more challenging. Unfortunately, existing secure audit logging solutions have significant limitations, and therefore cannot meet the requirements of such applications.

In this chapter, we first identify the limitations of naive (non-cryptographic) secure audit logging approaches and then enumerate the desirable properties of an ideal cryptographic secure audit logging scheme. We then identify the limitations of existing cryptographic secure audit logging schemes. Last, we present our solutions that address all these limitations.

1.1 Motivation

The central role of audit logs in computer forensics lead to the proposal of numerous log protection techniques. However, none of the previous solutions can address the requirements of secure audit logging in task-intensive and/or resource-constrained applications.

1.1.1 The Limitations of Naive (Non-Cryptographic) Approaches

Early naive secure audit logging approaches protect audit logs by preventing the adversary from accessing and/or modifying log entries. These approaches and their limitations have been identified

in [18, 43, 78, 79, 94, 103, 120] as follows:

- One approach is to rely on an omni-reliable and secure end-to-end real-time communication channel between each logging machine and a remote Trusted Third Party (TTP) to transmit log entries before the adversary compromises the logging machine. However, in modern IT systems, it is impractical to assume an omni-reliable and secure end-to-end real-time connection between each logging machine and a TTP [79, 104]. For instance, many real-life application driven by the Internet based services are considered Delay Tolerant Networks (DTNs) [50, 51, 53], which are characterized by the lack of such a continuous end-to-end communication channel. Similarly, various ubiquitous systems such as Unattended WSN (UWSN) [46, 76, 121] cannot guarantee the presence of such a communication channel.

- A straightforward approach is to back up log entries to a write-only medium (e.g., CD/DVD) assuming that the backup occurs before the logging machine is compromised. However, even for a moderate volume of logging activity, relying on a write-only medium such as CD/DVD for log protection is impractical.

- Another approach is to assume both the presence and "bug-freeness" of a tamper resistant hardware in each logging machine that prevents the adversary from reaching audit logs. However, in heterogeneous systems (e.g., compute clouds), it is impractical to assume both the presence and "bug-freeness" of a tamper resistant hardware on all types of platforms (e.g., wireless sensors [76], commercial off-the-shelf systems [18]). For instance, the recently introduced Write-Once-Read-Many (WORM) drives [34] were rapidly adopted for secure audit purposes [115]. However, several vulnerabilities of WORM drives were later identified (e.g., [58, 94]). This further confirms the danger in simply relying on a tamper-resistant hardware.

1.1.2 Desirable Properties of an Ideal Cryptographic Secure Audit Logging Scheme

The above limitations highlight the need for developing cryptographic mechanisms that can protect audit logs on physically unprotected machines without assuming continuous end-to-end real-time communication. There has been extensive research invested in this direction (e.g., [18, 43, 47, 57, 75, 78, 79, 103, 104, 120]). It is desirable for such schemes to have the following properties:

- *Forward Security and Append-only (aggregate) Properties:* Since the log verifiers are not necessarily available to verify the log entries once they are generated, a logger may have to accumulate log entries over a period of time. If the adversary takes full control of the logging machine in this duration, no cryptographic mechanism can prevent her from modifying the post-attack log entries (due to her control over the system)[1]. However, the integrity of log entries accumulated

[1]Post-compromise data can only be recovered/controlled if a remote online TTP or a locally trusted intrusion resilient hardware periodically checks the untrusted machine (e.g., key insulated schemes [49]) [79].

before the attack should be protected [18, 43, 57, 75, 78, 79, 120] (i.e., forward security [2]). Note that this protection should not only guarantee the integrity of individual log entries but also the integrity of log stream as a whole. That is, no selective deletion or truncation of log entries should be possible. This is achieved with signature aggregation (i.e., append-only property) [75, 76, 79] in addition to forward security.

- *Efficiency:* Computational overhead introduced by the secure audit logging scheme must be low at both the logger and the log verifier sides.

- *Public Verifiability:* Unlike schemes that enable only a few privileged entities who share secrets with loggers to verify the integrity of log entries, a scheme supporting public verifiability permits any entity to do so. Public verifiability is a desirable property for some critical applications, such as electronic voting, where logs need to be audited and verified by the public [20], and financial applications, where financial books of publicly held companies need to be verified by current and potential future share holders [57, 79].

- *Provable Security:* It is necessary for a scheme to have formal security assessments for its security properties, including attacks specific to audit logging systems such as truncation and delayed detection attacks.[2] (The details of these attacks are given in Chapter 4.)

- *Independence of Continuous Trusted Server Support:* Ideally, a secure audit logging scheme should not rely on a continuous online trusted server, though an offline server or a passive server may be used from time to time. This is desirable since any dependence on the continuous communication with a trusted server could be exploited by an adversary to defeat the scheme (e.g., delayed detection attacks [78, 79, 120]).

1.1.3 The Limitations of Symmetric Cryptography based Secure Audit Logging Schemes

The first cryptographic schemes (e.g., [17,18,103,104]) addressing the forward-secure audit logging rely on the symmetric cryptography to achieve the computational efficiency. These schemes adopted various symmetric primitives such as forward-secure Message Authentication Code (MAC) [69], forward-secure Pseudo Random Number Generators (PRNGs) and cryptographic hash chains [19, 70]. Despite their computational efficiency, these symmetric schemes have critical drawbacks:

- The symmetric nature of these schemes does not allow public verifiability, and therefore they cannot address applications requiring public auditing [57, 75, 76, 120] (e.g., financial auditing for cooperations [4] and secure auditing for electronic voting [20]).

[2] Delayed detection attack occurs if the log protection mechanism cannot the achieve immediate verification property. For instance, some secure audit logging schemes (e.g., [104]) require an online TTP support to verify the log entries. In such schemes, log verifiers cannot detect whether the log entries are manipulated until their TTP provides necessary keying information to them.

- They either need full symmetric key distribution (e.g., FssAgg-MAC in [76]), which incurs high storage overhead, or they require continuous remote trusted server support [103], which entails costly maintenance and attracts potential single-point of failures.

- They are vulnerable to certain attacks such as truncation and delayed detection attacks [75, 79].

1.1.4 The Limitations of PKC-based Secure Audit Logging Schemes

To address the limitations of symmetric cryptography based secure audit logging solutions, several PKC-based secure audit logging schemes have been proposed. Logcrypt extends the forward-secure MAC strategy to the PKC domain; it is publicly verifiable and secure against the delayed detection attack without requiring online TTP support [57]. However, Logcrypt incurs high storage/communication overheads and it is also still vulnerable to the truncation attack. Ma and Tsudik proposed FssAgg schemes ([75,76,78,79]), which use forward-secure signatures and aggregate signatures in an integrated way. These schemes require only a single-final aggregate signature for all the accumulated log entries (due to the ability to aggregate individual signatures into a single compact signature), and therefore are signature storage/transmission efficient. This approach also provides security against the truncation attack (i.e., "all-or-nothing" property) [75].

However, despite their advantages, all existing PKC-based secure audit logging schemes share drawbacks that make them impractical for certain applications:

- These schemes require several Expensive Operations (ExpOps)[3] to compute and verify the signatures of accumulated log entries. Therefore, they are impractical for secure audit logging in task-intensive and/or resource-constrained applications.

- To verify a particular log entry, all these schemes (e.g., [75, 76, 78]) force log verifiers to verify the entire set of log entries, which entails a linear number of ExpOps. A failure in this verification does not give any information about which log entry(ies) is (are) responsible for the failure. The iFssAgg schemes [79] mitigated these problems by allowing a more fine-grained verification. However, iFssAgg schemes double the signature computation and verification costs of their base FssAgg schemes to prevent the truncation attack. Moreover, the verification of a subset of given log entries still requires linear ExpOps in terms of the size of the given subset. Also, even if a small fraction of log entries are damaged, detecting damaged entry(ies) requires a linear number of ExpOps.

- All these schemes rely on only heuristic security arguments to justify their security against the truncation attack. However, security-critic applications demand provable security.

[3]For brevity, in this dissertation, we refer to an expensive cryptographic operation such as modular exponentiation [109] and pairing [82] as an ExpOp.

1.2 Our Contributions

The above discussion indicates that existing secure audit logging schemes cannot meet the requirements of task-intensive and/or resource-constrained applications.

In this dissertation, we developed a series of efficient, compromise resilient and compact cryptographic constructions to meet these requirements. We summarize our proposed schemes and their properties as follows:

1. *Hash-Based Sequential Aggregate and Forward-Secure Signature (HaSAFSS) for Unattended Wireless Sensor Networks (UWSNs) [121, 122]*: Resource-constraints [84] and the lack of real-time communication [50] make secure audit logging a challenging task in UWNSs, especially in the presence of active adversaries [76]. To address this problem, we propose a new class of signature schemes, which we refer to as *Hash-Based Sequential Aggregate and Forward-Secure Signature (HaSAFSS)*.

   HaSAFSS schemes utilize existing verification delays in UWSNs to introduce asymmetry between signers and verifiers. In this way, they integrate the efficiency of MAC-based aggregate signatures and the public verifiability of bilinear map based signatures by preserving the forward security via Timed-Release Encryption (TRE) [36].

   We develop three specific HaSAFSS schemes, Symmetric HaSAFSS *(Sym-HaSAFSS)*, Elliptic Curve Cryptography (ECC) based HaSAFSS *(ECC-HaSAFSS)* and self-SUstaining HaSAFSS *(SU-HaSAFSS)*.

   The desirable properties of HaSAFSS schemes are summarized below:

   (a) HaSAFSS schemes achieve public verifiability and computational efficiency simultaneously.

   (b) HaSAFSS schemes are the only schemes that allow both signers and verifiers to get equal benefits of the optimal (i.e., ExpOp-free) computational efficiency.

   (c) HaSAFSS schemes allow a signer to sequentially generate a compact and fixed-size signature.

   (d) Sym-HaSAFSS and ECC-HaSAFSS complement each other in terms of their storage requirements by offering different alternatives for different applications. SU-HaSAFSS achieves the optimal (i.e., constant key/signature sizes) storage efficiency for both signers and verifiers by introducing a little more computation. However, it still remains more efficient than all the previous publicly verifiable alternatives.

   HaSAFSS schemes are ideal solutions for UWSNs and other applications having similar Non-Real-Time (NRT) communication characteristics.

2. *BAF and FI-BAF: Efficient, Compact and Compromise Resilient Cryptographic Schemes for Secure Audit Logging in Resource-Constrained Devices [120, 123]*: Publicly verifiable secure logging in resource-constrained devices is a hard task due to the computation and storage limitations of such devices. Unfortunately, all previous forward-secure and/or compact (aggregate) cryptographic tools are computationally costly (e.g., BM [12], BLS [26], FssAgg [75, 79]).

   Despite being computationally efficient, the HaSAFSS schemes cannot achieve immediate verification, which restricts their use for real-time applications. Moreover, the HaSAFSS schemes need a passive TTP support, which might not be available in certain applications.

   To fulfill the need of an optimal signer efficient secure logging scheme with the immediate verification property, we developed a new forward-secure and aggregate cryptographic construction called *Blind Aggregate Forward (BAF)* along with its extension *Fast-Immutable BAF (FI-BAF)*. The main idea is to use simple but efficient algebraic blinding operations to compute individual signatures of log entries, which can be aggregated with just a modular addition operation. The blinding and aggregation operations preserve forward security, verifiability and indistinguishability without requiring any ExpOp at the logging device. This approach offers the following desirable properties:

   (a) BAF does not require a time factor to be publicly verifiable. Therefore, it can address real-time applications by achieving the immediate verification property.

   (b) BAF does not need online TTP support to enable the signature verification. This eliminates the potential single point of failure risks stemming from TTP dependence, and also makes BAF more scalable than the symmetric schemes (e.g., [18, 76, 103, 104]) and the HaSAFSS schemes.

   (c) In BAF, signing a single log entry requires only a few cryptographic hash operations. Hence, it is significantly more computationally efficient than all PKC-based schemes at the signer side.

   (d) BAF can produce publicly verifiable forward-secure and aggregate signatures with near-zero storage and communication costs for the loggers, even if the signing procedure is ExpOp-free.

   (e) BAF is proven to be ForWard-secure Aggregate Existentially Unforgeable against Chosen Message Attack (FAEU-CMA) in the Random Oracle Model (ROM) [13] based on the intractability of Discrete Logarithm Problem (DLP) [109]. Moreover, BAF provides a formal proof against the truncation attack without the help of an external signature scheme (the details are given in Chapter 4).

   (f) An extension of BAF, Fast-Immutable BAF (FI-BAF), allows the verification of a particular log entry without compromising the security of the original BAF as well as preserving its

computational efficiency. FI-BAF is also more efficient than its immutable counterparts (i.e., iFssAgg schemes [75]) at the signer side.

All the above properties make BAF an ideal alternative for secure logging on resource-constrained devices.

3. *LogFAS: Efficient, Compromise Resilient and Append-only Cryptographic Schemes with Verifier Efficiency*: In various applications, system auditors (i.e., verifiers) are required to monitor a large numbers of logging machines in real time. This requirement forces verifiers to process a large numbers of audit logs simultaneously, and also incurs heavy public key/certificate storage overhead. All previous publicly verifiable secure audit logging schemes are costly at the verifier side. Despite being signer efficient, BAF also requires an ExpOp per-item at the verifier side, which might be costly for some applications. As discussed before, HaSAFSS schemes are not suitable for real-time log verification.

   To address these problems, we propose a new forward-secure and append-only signature scheme called *Log Forward-secure and Append-only Signature (LogFAS)*. LogFAS utilizes various cryptographic primitives including incremental hashing [11], DSA tokens [92] and Schnorr signature [105] in novel ways, which offers the following unique properties:

   (a) LogFAS is the only PKC-based forward-secure and append-only construction that can verify L items with a small and constant number of ExpOp(s) regardless of the value of L. All the previous PKC-based alternatives require a linear number of ExpOps with respect to the number of log entries L. Hence, LogFAS is significantly more computational-efficient than all previous PKC-based schemes at the verifier side.

   (b) LogFAS requires each verifier to store only a small and constant size public key. This makes LogFAS the most verifier-storage-efficient among the publicly verifiable secure audit logging schemes.

   (c) Similar to BAF, LogFAS achieves the public verifiability *without* requiring any online TTP support or a time factor.

   (d) LogFAS is proven to be secure in ROM [13] based on the existential unforgeability [16] of the Schnorr signature [105] and the Target Collision Resistance (TCR) property of incremental hashing IH [11].

   (e) A variant of LogFAS can identify the corrupted log entries with a *sub-linear* number of ExpOps when most log entries are intact. In contrast, the other schemes always require a linear number of ExpOps.

   All the above properties make LogFAS an ideal choice for task-intensive log verification in real-time applications.

1.3 Organization of Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 gives background information and related work. Chapter 3 presents the HaSAFSS schemes in detail. Chapter 4 provides the details of BAF and FI-BAF. Chapter 5 presents LogFAS and its extension in detail. Chapter 6 concludes this dissertation and discusses future work.

Chapter 2

Background and Related Work

In this chapter, we provide background information and related works. We first give a review of forward-secure and/or aggregate signatures, which constitute the basis of our schemes. We then discuss works related to the HaSAFSS schemes [121, 122] including time-related cryptographic primitives and self-healing techniques in UWSNs. We review the previous cryptographic secure audit logging schemes, which are then compared with our schemes in detail in their own chapters. Lastly, we briefly discuss Elliptic Curve Cryptography (ECC), on which ECC-HaSAFSS in [121] and the implementation of BAF [120] are constructed.

2.1 Forward Secure and/or Aggregate Signature Schemes

We briefly investigate the previous forward-secure and/or aggregate signature schemes below.

2.1.1 Forward-Secure Signature Schemes

A forward-secure signature aims to minimize the effect of key compromises. This goal is achieved via the *sign-evolve-delete* strategy, which can be exemplified by the first forward-secure signature scheme proposed by Anderson in [8]. Anderson's scheme can be constructed from any standard signature scheme SGN (e.g, [93]) generically. The intuition is as follows:

The signer first generates L private/public key pairs as $sk \leftarrow (sk_0, \ldots, sk_L)$ and $PK \leftarrow (PK_0, \ldots, PK_L)$, respectively, where L denotes the maximum number of time periods that a signer can use. Each $\{sk_j, PK_j\}_{j=0}^{L}$ is associated with a time period t_j.

For each time period $t_j, 0 \leq j \leq L$, the signer computes the signature on data items D_j received in the duration of t_j as $\sigma_j \leftarrow SGN.Sig_{sk_j}(D_j)$. The signer then updates the private key by simply deleting sk_j. Assume that the signer computed $(\sigma_0, \ldots, \sigma_l)$ on data items $(D_0, \ldots, D_l)$ until she is compromised by the adversary in t_{l+1}. After the compromise, the adversary cannot forge $(\sigma_0, \ldots, \sigma_l)$,

since their associated private keys were deleted. That is, forging data items accumulated before the compromise is as difficult as breaking SGN.

The limitation of this construction is that all performance characteristics grow linearly with the value of L. That is, the storage overhead is linear with respect to the value of L at both the signer and the verifier sides. Moreover, the signature generation and signature verification require a $SGN.Sig$ and a $SGN.Ver$ operation for per-item, respectively (i.e., $O(L)ExpOp$ at both the signer and the verifier sides).

Several new schemes were later proposed to address these limitations. We can classify these schemes in two categories: Schemes based on a specific number theoretic assumption, and generic schemes that can be constructed from any standard signature scheme. We give a brief overview of these approaches below.

Non-generic Forward-Secure Signature Schemes

Bellare and Miner (BM) proposed the first formalized forward-secure signature scheme in [12] by following Anderson's scheme [8]. The BM scheme relies on the intractability of factoring a Blum-Williams integer. Most notably, in this scheme, the number of private and public keys as well as the length of signature do not grow linearly with L. However, the sizes of these individual private and public keys are large. Abdalla and Reyzin (AR) improved the performance of BM [12] in [2] by shortening the lengths of private and public keys. However, the signature generation and signature verification of AR are more costly than that of BM.

Kozlov and Reyzin (KR) developed a forward-secure signature in [67] that achieves a computation-efficient key update (i.e., one modular exponentiation for each key update). This property is useful for the scenarios, in which time intervals are evolved frequently. Itkis and Reyzin (IR) developed a forward-secure signature scheme in [63], which is based on Guillou-Quisquater signature [54]. In AR, the computational cost of the signature generation and verification is only two modular exponentiations with short exponents. However, the key update operation of IR is computationally expensive.

Boyen et al. proposed a forward-secure signature scheme in [28], where key update can be performed on encrypted keys as a second layer of security. Later, other schemes with similar properties were also proposed (e.g., [71]).

The main advantage of these non-generic signature schemes is that, they show a better performance for certain scenarios than a generic scheme (e.g., frequent key evolve [67] and constructions specifically designed for secure audit logging [76, 121] and WNSs [84]).

Generic Forward-Secure Signature Schemes

The first generic forward-secure signature is Anderson's scheme [8], which was described in subsection 2.1.1. Bellare and Miner (BM) developed a generic construction in [12] (in addition to their non-generic

construction), which utilizes a binary tree approach to achieve constant public key and signature sizes. However, its private key size and signing/verifying operations are all logarithmic with respect to the value of L (instead of a single ExpOp per-item). Later, Malkin et al. developed a highly efficient generic construction referred to as the MMM scheme [80]. The MMM scheme utilizes Merkle tree [88] based certification chains combined with the binary tree scheme of BM [12]. This enables the MMM scheme to achieve a practically unbounded number of time periods. We briefly discuss MMM scheme later in Section 3.1.

Krawczyk proposed another generic construction in [68], which utilizes Forward-secure Pseudo Random Number Generators (FWPRNGs) and a standard signature scheme SGN. In this scheme, the signer generates an initial private key sk_0, and then generates L public keys and their corresponding certificates. These public keys and certificates are given to the verifiers before the deployment. In signature generation, for each time period $t_j, 0 \leq j \leq L$, the signer computes $\sigma_j \leftarrow SGN.Sig_{sk_j}(D_j)$ on D_j, and then updates the private key sk_j as $sk_{j+1} \leftarrow F(sk_j)$, where F is a FWPRNG function (such a function can be obtained from a standard cryptographic hash function such as SHA-1 [109]). The signer then deletes sk_j from the memory permanently. The verification is done as in Anderson's scheme [8]. Our schemes the HaSAFSS [121, 122] and the BAF [120] also follow a similar key update strategy.

The generic forward-secure signature schemes have two main advantages. First, they achieve provable security just by assuming a secure standard signature scheme exists [5]. In contrast, non-generic schemes can be proven to be secure in the Random Oracle Model (ROM) [13]. Second, they can be constructed from any standard signature scheme. Hence, they can trade-off computational and storage efficiencies by using different base standard signature schemes with different performance characteristics [5].

Key Update Strategies for Forward-secure Signatures

A forward-secure signature scheme may operate in one of the following two key update models:

1. *Per-item model*: Sign each individual entry D_j as soon as it is received.
2. *Per-period model*: Sign a group of entry D'_j for each time period t_j, where D'_j denotes all individual data items received in t_j.

In terms of the key evolving strategy, *per-item* and *per-period* key update models are identical. However, they offer a *storage-security trade-off* [75] that can be decided according to the requirements of audit logging applications. That is, the per-item model provides the forward security of each individual data item, but it demands more storage overhead. In contrast, the per-period model provides the forward security only across time periods, but it demands less storage overhead.

2.1.2 Aggregate Signature Schemes

Another important digital signature primitive is the aggregate signature, which aggregates n individual signatures associated with n different data items into a single, compact signature. That is, assume that each user $j, 0 \leq j \leq L$ possesses a private/public key pair (sk_j, PK_j), and wants to attest to a message D_j. Each user first signs D_j by obtaining σ_j, and these signatures can be combined into a single aggregate signature $\sigma_{0,L}$ by any party. The aggregate signature verification algorithm verifies $\sigma_{0,L}$ on all messages $(D_0, \ldots, D_L)$ for all public keys $(PK_0, \ldots, PK_L)$. Therefore, $\sigma_{0,L}$ provides the non-repudiation of all messages $(D_0, \ldots, D_L)$ simultaneously [73].

Aggregate digital signatures are useful primitives for various real-life applications including certificate chains and secure routing. For instance, consider a Public Key Infrastructure (PKI) of depth L. Here, the certificate $Cert_{ID_i}$ on PK_i associated with user ID_i is comprised of a certificate chain, which is issued by a hierarchy of Certification Authorities (CAs). This certificate chain is long in complex CA hierarchies, and the user ID_i is required to include $Cert_{ID_i}$ in his all (new) signature/public key transmissions. It is therefore desirable to keep the length of authentication tags in $Cert_{ID_i}$ constant-size [73].

The first aggregate signature scheme was proposed in [26], which utilizes the BLS (Boneh-Lynn-Shacham) signatures [27]. This scheme is proven to be secure in ROM under the Bilinear Diffie-Hellman (BDH) problem [66]. It uses a full domain hash function and a single modular exponentiation under large prime p to compute an individual signature for a given data item. The aggregation of two such individual signatures is just a modular multiplication operation under mod p. Moreover, the sizes of private key and aggregate signature are small, especially when the scheme is implemented in elliptic curves (see Section 2.4). However, the signature verification requires a pairing operation (see Section 3.1) for each item to be verified, which makes it computationally costly.

Various new aggregate signature schemes have been developed to offer different properties. Lysyanskaya et al. [73] proposed a sequential aggregate signature, in which the order of signers are preserved by the signature scheme. This scheme is based on a family of certified trapdoor permutations (TPDs) [10]. Lysyanskaya et al. gave a formal definition of sequential aggregate signatures, and then provided tight security reductions in ROM for homomorphic and claw-free TDPs. Zhu et al. gave applications of aggregate signatures to the secure routing discovery [126] and reliable broadcast communication [127] problems. Lu et al. proposed a sequential aggregate signature and a multi-signature scheme in [72] that are proven to be secure in standard security model [31] instead of ROM [13]. Mu et al. [90] proposed a compact sequential aggregate signature scheme based on RSA [102] with formal security definitions.

Boldyreva et al. developed multisignatures and identity-based sequential aggregate signatures in [21] for secure routing applications. Their ordered multi-signature (OMS) allows signers to attest a common message sequentially with a higher computational and storage efficiency than traditional sequential aggregate signatures. The security of OMS relies on the intractability of the Computational Diffie-Hellman (CDH) problem [64]. The identity-based sequential signature (IBSAS) aims to address bandwidth ef-

ficient secure routing, since identity-based primitives do not require to transmit large public keys and certificates. Boldyreva et al. claimed that IBSAS is secure based on an assumption called as M-LRWS (the modified version of LRWS introduced in [74]). However, Hwang et al. [59] showed that IBSAS is universally forgeable due to the fact that the intractability assumption of M-LRWS does not hold.

2.1.3 Forward-Secure and Aggregate Signature Schemes

Ma et al. proposed two forward-secure and aggregate signature schemes in [76]: FssAgg-BLS and FssAgg-MAC. These schemes are integrated signature schemes, which achieve signature aggregation and forward security simultaneously. FssAgg-BLS uses forward-secure hash chains and BLS signatures [26] to compute and verify aggregate signatures. However, similar to the traditional BLS-based aggregate signature schemes, FssAgg-BLS is also computationally costly.

In FssAgg-MAC, a TTP distributes a secret chain root sk_0 to the signer and the verifier before the system deployment. In the signature generation phase, assume that the signer has accumulated i data items $(D_0, D_1, ..., D_{i-1})$ and computed their aggregate signature $\sigma_{0,i-1}$ at a time period t_w. When the signer collects data item D_i, she first computes the individual signature of D_i as $\sigma_i \leftarrow MAC_{sk_i}(D_i)$ and aggregates σ_i into $\sigma_{0,i-1}$ by simply computing $\sigma_{0,i} \leftarrow H(\sigma_{0,i-1}||\sigma_i)$. The signer then updates the chain key as $sk_{i+1} \leftarrow H(sk_i)$ and deletes sk_i when she advances to the next time period t_{w+1}, where H denotes a cryptographic hash function (e.g., SHA-1 [109]). Since the verifier knows the secret chain root, she can verify $\sigma_{0,i}$ by following the same procedure as in the signature generation. Despite its effectiveness and simplicity, FssAgg-MAC does not allow signatures to be publicly verifiable.

To reduce storage/computational overhead of the pioneering FssAgg primitives, Ma proposed two additional FssAgg schemes [75] called FssAgg-Bellare-Miner (FssAgg-BM) and FssAgg-Abdalla-Reyzin (FssAgg-AR), which were derived from the forward-secure signature schemes in [12] and in [2], respectively. FssAgg-BM utilizes a technique that allows individual BM [12] signature pairs to be aggregated sequentially without destroying their verifiability and forward security. The FssAgg-AR follows a closely related strategy to sequentially aggregate the forward-secure signature pairs of the AR scheme [2]. The security of FssAgg-BM and FssAgg-AR relies on the assumption that a squaring operation is a one-way function, and the factorization of a Blum-Williams integer into two primes is intractable for appropriate parameter sizes. FssAgg-AR is more storage efficient than FssAgg-BM, but FssAgg-BM requires fewer squaring operations, and therefore it is more computationally efficient than FssAgg-AR. Despite both FssAgg-BM and FssAgg-AR are more computationally efficient than FssAgg-BLS in [76] at the verifier side, they are still computationally expensive for secure audit logging in task-intensive and/or resource-constrained applications.

2.2 Time-Related Cryptographic Primitives and Self-Healing Techniques

Our proposed HaSAFSS schemes [121,122] rely on a time factor to achieve their intended goals. Specifically, HaSAFSS schemes utilize the Timed-Release Encryption (TRE) concept to introduce the desired asymmetry between the signers and verifiers to achieve public verifiability and forward security simultaneously. Therefore, we first briefly discuss the TRE concept. We then summarize TESLA, which also uses a time factor to introduce an asymmetry between the signers and verifiers.

We finally discuss the self-healing techniques in UWSNs that are distantly related to HaSAFSS schemes.

2.2.1 Timed-Release Encryption (TRE)

The purpose of TRE [85] is to encrypt a message in such a way that no entity including the intended receivers can decrypt it until a pre-defined future time. TRE has several applications in real-life [41]:

- *Confidential Commitments*: In various applications such as sealed-bid auctions, electronic lotteries and legal wills, the ciphertext can be considered as a commitment of the sender. This commitment should be available to its intended recipients no earlier than a designed future time. The properties of TRE satisfies these requirements. That is, after the sender provides the ciphertext (i.e., the commitment) to the recipients, she cannot change the original message anymore. At the same time, receivers cannot obtain any information about the message content before its intended time. Once the time trapdoor information is released, the receivers can recover the original message from the ciphertext.

- *Bulk Information Pre-distribution*: TRE enables a sender to send a bulk ciphertext beforehand without the risk of an information leakage. Since the trapdoor information is significantly smaller than the ciphertext (and even than the original message), this allows a rapid dissemination (i.e., the recovery) of the encrypted content at its desired availability time. In this way, the sender can avoid several problems that may stem from the burst transmission of bulk data (e.g., network/server bottlenecks). Such applications include the distribution of entertainment media, software license and scheduled payments.

One approach to obtain a TRE scheme is to force recipients to perform intensive computations on a given challenge (i.e., the ciphertext). For instance, the sender can encrypt the message with a short symmetric key. However, this approach imposes an intolerable computational overhead to the recipients, and also the release-time cannot be controlled precisely, since the computation power may change from one recipient to another.

The majority of modern TRE schemes are based on a Trusted Agent (TA), in which a time server provides universally accepted time reference and trapdoor information to the users [101]. Hence, users

can decrypt the ciphertext when its related trapdoor information is released by the TA. Some preliminary TA-based schemes require a multi-round interaction with the TA (e.g., [45]). Blake et al. developed the first TRE scheme in [36] that does not require an interaction with the TA. However, they did not provide a formal security analysis of their construction. Later, non-interactive TRE schemes with formal security assessments were proposed (e.g., [33, 37]).

In some applications, recipients might need to decrypt the ciphertext before its intended time (with the permission of the sender). To address this need, TRE schemes with a pre-open capability were developed [41, 60]. That is, the sender helps recipients to decrypt the ciphertext by publishing a pre-open key that enables the decryption without interacting with the TA. In these schemes, it is important to prevent the sender from manipulating the ciphertext in a predictable way via this auxiliary key, and the security model should capture this threat as well. Later, generic TRE schemes, which can be constructed from any traditional public key encryption scheme or an IBE scheme [25], were proposed (e.g., [83]).

Our schemes Sym-HaSAFSS and ECC-HaSAFSS only use the basic TRE concept [85] to fulfill the optimal computational efficiency goal. Our other scheme SU-HaSAFSS inspires from the time trapdoor mechanism used in [113] to achieve the self-sustainability and optimal storage efficiency, but it is more computationally costly than Sym-HaSAFSS and ECC-HaSAFSS.

2.2.2 TESLA

Another cryptographic scheme that utilizes the time factor is TESLA [96]. TESLA is an efficient broadcast authentication protocol that also uses delayed disclosure of the keying material, assuming that senders and receivers are loosely synchronized. However, TESLA cannot address the cryptographic secure logging problem properly due to the following limitations:

- TESLA cannot achieve forward security, since an active adversary compromising a sender can forge MACs valid for the given time interval.
- TESLA does not use a signature aggregation strategy, and therefore its communication overhead is linear with respect to the number of data items to be signed.
- TESLA cannot be used for some UWSN applications, in which a loose time synchronization cannot be guaranteed for the network entities.

2.2.3 Self-Healing Techniques

Recently, a series of studies [77, 98, 99] based on self-healing techniques have been proposed to achieve data survival in UWSNs. These studies proposed mobile adversary models, in which the adversary compromises the sensors and deletes the data accumulated in them. To confront such an adversary, they propose collaborative techniques, in which non-compromised sensors collectively attempt to recover a

compromised sensor [77, 99] by introducing local randomness (with a PRNG) to their neighborhood. DISH [77] assumes a read-only adversary and targets data secrecy. POSH [99] allows for constrained write-only adversaries and targets data survival. Pietro et. al. [98] elaborated the adversary models given in [77, 99] and provided experimental/analytical results for them.

Note that the adversary models and security goals in [77, 98, 99] are different from our schemes. In HaSAFSS schemes (similar to FssAgg [76]), the goal of the adversary is to forge the data. However, the goal of the adversary in [77, 98, 99] is to prevent the data from reaching the sink (not modifying or forging it).

2.3 Secure Audit Logging Schemes

We first survey the schemes that are based on forward-secure stream integrity and digital signatures. We then briefly discuss secure auditing approaches with different properties than those of our own schemes.

2.3.1 Forward-secure Stream Integrity and Digital Signature based Solutions

The pioneering studies addressing the forward-secure stream integrity for audit logging were presented in [17, 18]. The main focus of these schemes is to formally define and analyze forward-secure MACs and PRNGs. Based on their forward-secure MAC construction, they also presented a secure logging scheme, in which log entries are tagged and indexed according to evolving time periods.

Schneier et al. [103, 104] proposed secure logging schemes that use one-way hash chains together with forward-secure MACs to avoid using tags and indexes. In these schemes, a trusted server provides auxiliary keying materials to the verifiers to enable log verification. While reducing the need for trust in the verifiers, this approach introduces a trusted server that has to be online for each verification. This brings architectural difficulties as well as making the system open to a single point of failure.

Logcrypt [57] extended the idea given in [18, 103] to the PKC domain by replacing MACs with digital signatures and ID-based cryptography. Logcrypt can also be considered as an extension of Anderson's scheme [8] with the addition of a confidentiality service. In this scheme, each signer generates a private/public key pair for a digital signature and/or PKC-based encryption scheme (traditional or Identity-Based Encryption (IBE)). For each log entry, the signer computes a signature and also (optionally) encrypts this log entry. The private keys are evolved as in Anderson's scheme.

Finally, Ma et al. proposed a set of comprehensive secure audit logging schemes in [78, 79] based on their forward-secure and aggregate signature schemes given in [75, 76], whose properties were outlined in 2.1.3. The detailed analysis and comparison of all these schemes with ours will be given in their corresponding chapters.

2.3.2 Audit Logging Schemes Aiming with Different Properties to Our Own

Itkis proposed cryptographic tamper resistance techniques in [62] that can detect tampering even if all the keying material is compromised. Our schemes can be combined with the Itkis model as any forward-secure signature [62].

Davis et al. proposed time-scoped search techniques on encrypted audit logs [47]. Waters et al. proposed an audit logging scheme [116] relying on IBE, which enables an encrypted search on logs without relying on a time factor. There are several other encrypted search schemes (e.g., [24, 44, 107]) that can be used for the same purpose. All these schemes can be coupled with our schemes to provide confidentiality and effective log utilization.

There is a line of work that relies on authenticated data structures to protect audit logs in distributed systems [7, 81, 95]. While being computationally efficient, these approaches do not provide forward security. Furthermore, any authenticated data structure can be strengthened with a forward-secure signature [43]. Therefore, our schemes can serve these authenticated data structures as a forward-secure and aggregate digital signature primitive.

Apart from the above schemes, there is another line of work that utilizes a trusted hardware to enhance the security of audit logs. Chong et al. proposed an extension to the scheme in [103] by using tamper-resistant hardware [39]. Xu et al. proposed SAWS [118], which relies on a Trusted Computing Base (TCB) to protect private keys used for PKC operations.

Accorsi gave surveys on existing secure audit logging schemes in [3, 4].

2.4 Elliptic Curve Cryptography (ECC)

Elliptic curve cryptography (ECC) is an approach to public-key cryptography based on the algebraic structure of elliptic curves over finite fields [56, 109, 117]. Elliptic curves used in cryptography are typically defined over two types of finite fields: prime fields F_p, where p is a large prime number, and binary extension fields F_{2^m}. In our schemes, we only use elliptic curves over F_p.

An elliptic curve over F_p is defined by a cubic equation $y^2 = x^3 + ax + b$, where $a, b \in F_p$ are constants such that $4a^3 + 27b^3 \neq 0$ [56]. The points on the elliptic curve $y^2 = x^3 + ax + b$ consist of all pairs of affine coordinates (x, y) for $x, y \in F_p$ that satisfy the equation $y^2 = x^3 + ax + b$ and a point $\mathcal{O}(x, \infty)$ at infinity. These points form an abelian group with respect to a special addition operation, where $\mathcal{O}$ is the additive identity of this group. The formulas defining point addition, point doubling, and other details can be found in [56, 109].

For a generator point G on an elliptic curve, the set $\{\mathcal{O}, G, 2G, 3G, ...\}$ is a cyclic group [56]. The calculation of $k \cdot G$, where k is an integer, is called a *scalar multiplication*. The problem of finding k given points $(k \cdot G)$ and G is called the *Elliptic Curve Discrete Logarithm Problem (ECDLP)*. It is computationally infeasible to solve ECDLP for appropriate parameters (e.g., for G with a large prime

order) [56]. The hardness of ECDLP allows several cryptographic schemes based on elliptic curves, such as ECDH and ECDSA [87].

2.5 Conclusion

In summary, forward-secure and aggregate signatures are ideal cryptographic tools to achieve compact and compromise-resilient authentication and integrity for audit logs. However, all previous PKC-based forward-secure and aggregate secure logging schemes are highly costly, and their symmetric counterparts are not publicly verifiable as well as being open to certain attacks.

In the following chapters, we present our forward-secure and aggregate (append-only) cryptographic mechanisms that overcome these limitations.

Chapter 3

Efficient, Compromise Resilient and Compact Cryptographic Constructions for UWSNs

An Unattended Wireless Sensor Network (UWSN) [76, 77, 86, 98, 99] is a Wireless Sensor Network (WSN) in which continuous end-to-end real-time communication is not possible for sensors (senders) and their receivers (e.g., mobile collectors, static sinks). In other words, receivers might not be available for sensors from time to time, sometimes for long time periods. Sensors accumulate the sensed data in these time periods, and transmit it to the receivers whenever they become available (e.g., visits of mobile collectors [76, 77]).

Examples of UWSNs can be found in military WSN applications (e.g., [98, 111]), where sensors are deployed to an adversarial and unattended environment to gather information about enemy activities. One illustrative example is LANdroids [46], a recent U.S. Defense Advanced Research Projects Agency (DARPA) research project, which designs smart robotic radio relay nodes for the battlefield deployment. These nodes are expected to be deployed in hostile environments to gather military information and upload to ally vehicles (e.g., UAV, soldier) upon their arrivals.

The lack of real-time communication and the resource constraints of UWSNs bring several security and performance challenges, especially when an UWSN is deployed in a hostile environment as described above. In particular, inability to off-load the sensed data forces sensors to accumulate a large amount of data along with their authentication information. More importantly, unattended settings make the UWSN highly vulnerable to active [76] and/or mobile adversaries [77, 98]. Such an adversary can physically compromise sensors and gain access to the accumulated data as well as the existing cryptographic keys. When a sensor is compromised, the adversary can always use the cryptographic keys learned from the sender to generate forged messages after the attack. However, it is critical to prevent the adversary from modifying the data accumulated before the adversary takes control of the sender [76].

Forward-secure and aggregate signatures are ideal cryptographic tools for achieving data integrity and authentication for UWSN applications in the presence of active adversaries [76] (motivations and properties of forward-secure and aggregate signatures were discussed in Chapter 1 and Chapter 2, respectively). However, all existing PKC-based forward-secure and aggregate signature schemes (e.g., [26, 75, 76]) impose extreme computational overheads on the network entities, which are intolerable for resource-constrained UWSN applications. Another alternative is to rely on symmetric key cryptography via hash chains and Message Authentication Codes (MAC) as in FssAgg-MAC [76]. However, such an approach requires full symmetric key distribution and does not allow signatures to be publicly verifiable. This makes it unscalable and impractical for large distributed UWSN applications. Thus, it is necessary to seek highly efficient and flexible forward secure and aggregate signatures for UWSN applications.

To address this problem, we propose a new class of cryptographic schemes for UWSN applications, which we call *Hash-Based Sequential Aggregate and Forward Secure Signatures (HaSAFSS, pronounced as "Hasafass").* We develop three specific HaSAFSS schemes, a *Symmetric HaSAFSS* scheme (called *Sym-HaSAFSS*), *ECC-based HaSAFSS* scheme (called *ECC-HaSAFSS*), and a *self-SUstaining HaSAFSS* scheme (called *SU-HaSAFSS*).

A nice property of these schemes is that they achieve several seemingly conflicting goals, computational efficiency, public verifiability, forward security, flexibility and sustainability at the same time. To achieve this, HaSAFSS schemes introduce asymmetry between the senders and receivers using the time factor via Timed-Release Encryption (TRE) [101]. Using this asymmetry, our schemes achieve high efficiency by minimizing costly Expensive Operations (ExpOps), while still remaining publicly verifiable and forward-secure. In addition to these common properties, each HaSAFSS scheme also achieves unique properties specific to it.

We summarize the properties of HaSAFSS schemes as follows:

1. Our schemes achieve near-optimal computational efficiency and public verifiability at the same time. They achieve the computational efficiency by adopting cryptographic hash functions to compute aggregate and forward-secure signatures, and thus are much more efficient than all the existing schemes (e.g., [26], FssAgg-BLS in [76], FssAgg-AR/BM [75]), with the exception of FssAgg-MAC in [76]. When compared with FssAgg-MAC [76], our schemes further achieve public verifiability by eliminating symmetric key distribution.

2. In our schemes, both signers (senders) and verifiers (receivers) get equal benefits of computational efficiency, while most existing schemes incur heavy computational overhead on either the signer or verifier side. This property is especially useful for UWSN applications in which both the signers and verifiers need to process large amounts of data efficiently.

3. Since our schemes achieve signature aggregation, a signer always stores and transmits only a single compact signature, regardless of the number of time periods or data items to be signed. This offers bandwidth efficiency.

4. Sym-HaSAFSS relies on an omni-symmetric design even if it is public verifiable. Hence, it is the most computationally efficient scheme among of all its counterparts. It is also the most verifier storage friendly scheme by requiring only a small and constant storage for the verifiers. However, it requires a linear storage at the signer side.

5. ECC-HaSAFSS requires storing one key for each signer by offering a signer storage friendly scheme. However, it requires an ExpOp to initialize each time interval (but still requires only three hash operations to sign/verify per-item), and also demands quadratic storage overhead at the verifier side.

6. Despite their simplicity and efficiency, Sym-HaSAFSS and ECC-HaSAFSS put a linear bound on the maximum number of time periods that a signer can use. Moreover, they require a pre-determined and fixed data delivery schedule that all signers have to agree upon before deployment.

   SU-HaSAFSS addresses these limitations by offering following properties:

   - SU-HaSAFSS enables a signer to use (practically) unbounded number of time periods (this implies the ability of generating unbounded number of signatures) without requiring any re-keying after the deployment. This allows a signer to operate in a hostile environment for a long time without requiring re-deployment/re-keying support. This also offers only a constant key storage at the signer side and a linear key storage at the verifier side.
   - SU-HaSAFSS enables each sender to decide her own data delivery schedule dynamically (after the deployment) without requiring any (online) communication with other signers or a trusted third party. Therefore, SU-HaSAFSS can support applications in which a pre-determined data delivery schedule cannot be determined.

   To achieve these properties, SU-HaSAFSS requires a few ExpOps for per interval (for the initialization purpose), and therefore is more computationally costly than Sym-HaSAFSS and ECC-HaSAFSS. However, SU-HaSAFSS is significantly more efficient than all other PKC-based schemes (e.g., FssAgg) that require an ExpOp per data item (SU-HasAFSS requires only three hash operations per data item to sign or verify in given time interval).

HaSAFSS schemes utilize already existing verification delays in the envisioned UWSN applications as an opportunity to achieve the aforementioned properties. Thus, they are ideal solutions for UWSN applications in which high computational, storage, or bandwidth efficiency is more important than immediate verification.

The remainder of this chapter is organized as follows. Section 3.1 provides the preliminaries. Section 3.2 presents our assumptions as well as the security and data models. Section 3.3 describes the proposed schemes in detail. Section 3.4 provides the security analysis of the proposed schemes. Section 3.5 gives performance analysis and compares the proposed schemes with previous approaches. Section 3.6 gives a summary of this chapter.

3.1 Preliminaries

$\mathbb{G}_1$ is a cyclic additive group generated by generator G on an Elliptic Curve *(EC)* over a prime field $\mathbb{F}_p$, where p is a large prime number and q is the order of G and $\mathbb{G}_1$. kG, where k is an integer, denotes a *scalar multiplication*. $\mathbb{G}_2$ is a cyclic multiplicative group with the same order q.

$\mathcal{E}$, $\mathcal{D}$, $||$, and $|x|$ denote symmetric encryption function, symmetric decryption function, concatenation operation, and the bit length of variable x, respectively.

H_1 and H_2 are two distinct cryptographic hash functions, which are both defined as $H_1/H_2 : \{0,1\}^n \rightarrow \{0,1\}^{|H|}$, where n denotes the bit length of randomly generated input key and $|H|$ denotes the output bit length of the selected hash function. H_3 is used to compute aggregate signatures and is defined as $H_3 : \{0,1\}^* \rightarrow \{0,1\}^{|H|}$. H_4 is used to map an input key to a point on the EC, i.e., $H_4 : \{0,1\}^n \rightarrow \alpha G$. $H_5 : \{0,1\}^{|t|} \rightarrow Z_q^*$ is used to hash a t-bit time instance $T \in \{0,1\}^{|t|}$ (e.g., T="22:43, June 21 2011"). H_6 is defined as $H_6 : \mathbb{G}_2 \rightarrow Z_q^*$. We also use a secure MAC to compute individual signatures of data items, defined as $MAC_{sk} : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^{|H|}$.

t_w denotes a single time interval, which is formed from two consecutive pre-determined time points T_{w-1} and $T_w > T_{w-1}$. $t_{w,w'}$ denotes a unified time interval, which starts at the beginning of t_w and ends at the end of $t_{w'}$, $w' \geq w$.

SGN denotes a standard digital signature scheme (e.g., Schnorr [105], DSA [87]) and MMM denotes a Malkin Micciancio Miner (MMM) generic forward-secure signature construction [80] instantiated from an appropriate base scheme (e.g, [6,63]). $(\overline{sk}, \overline{pk}) = SGN.Kg(1^\kappa)$, $\sigma = SGN.Sig_{\overline{sk}}(m)$ and $\{success, failure\} = SGN.Ver_{\overline{pk}}(\sigma, m)$ denote the key generation for security parameter κ, signature generation on message m with private $\overline{sk}$ and verification of σ on m with public key $\overline{pk}$, respectively.

MMM signature scheme, in addition to the above standard algorithms, also has a key update algorithm, denoted $MMM.Upd(sk_w, w)$. That is, given the current private key sk_w, the update algorithm generates one or several new key instances to be used in the future. This is done by using a sum composition and a product composition iteratively based on a special tree structure. The details of this update procedure can be found in [80].

Definition 3.1 *$\widehat{e}$ is a bilinear pairing $\widehat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, i.e., an admissible map with the following properties:*

1. Bilinearity*: $\widehat{e}(aP, bQ) = \widehat{e}(bP, aQ) = \widehat{e}(abP, Q) = \widehat{e}(P, abQ) = \widehat{e}(P, Q)^{ab}$, $\forall P, Q \in \mathbb{G}_1$ and $\forall a, b \in Z_q^*$.*

2. Non-degeneracy*: $\exists P, Q \in \mathbb{G}_1$ such that $\widehat{e}(P, Q) \neq 1$. In our settings, we select prime order groups in which $\forall P, Q \in \mathbb{G}_1, \widehat{e}(P, Q) \neq 1$, and therefore $\widehat{e}(G, G)$ (G is the generator of $\mathbb{G}_1$) is a generator of $\mathbb{G}_2$.*

3. Efficiency*: There exists an efficient algorithm to compute $\widehat{e}(P, Q), \forall P, Q \in \mathbb{G}_1$.*

Definition 3.2 *Elliptic Curve Discrete Logarithm Problem (ECDLP) [56] is defined as follows: Given a prime p, a generator $G \in \mathbb{G}$, and a random point $Q \in E(F_p)$, find the integer k, $0 \leq k \leq p-2$, such that $Q \equiv kG$. ECDLP is (τ, ϵ)-hard, if no algorithm running in time less than τ can solve the ECDLP with a probability more than ϵ. where ϵ is computed over the random choices of (G, k).*

Definition 3.3 *q-Bilinear Diffie-Hellman Inversion (q-BDHI) problem [35] is defined as follows: Given $(q+1)$-tuple $(G, aG, a^2G, \ldots, a^qG) \in \mathbb{G}_1^{q+1}$ for some $a, q \in Z_p^*$, compute $\widehat{e}(G,G)^{a^{-1}} \in \mathbb{G}_2$. q-BDHI is (τ, ϵ)-hard, if no algorithm running in time less than τ can solve the q-BDHI with a probability more than ϵ, where ϵ is computed over the random choices of G.*

3.2 Models

We first give our threat model and security model including the HaSAFSS security definition and complexity/system assumptions. We then present our data model.

3.2.1 Threat Model and Security Model

Our treat model is based on a resourceful but Probabilistic Polynomial Time (PPT) bounded adversary $\mathcal{A}$ with the following abilities: (i) passive attacks against output of cryptographic operations, (ii) active attacks including packet interception/modification, and (iii) physically compromising senders/receivers (called as *"break-in"*) and extracting the cryptographic keys from the compromised nodes.

$\mathcal{A}$ aims to produce an existential forgery against the forward-secure and aggregate signature of the accumulated data that he obtained after the break-in. $\mathcal{A}$ may use any cryptographic key and data that she extracted from the compromised signers and verifiers.

Before giving the HaSAFSS security model, we review the *Quality of Forward Security (QoF)* concept [75]:

Definition 3.4 QoF *is a performance-forward security quality trade-off, which is decided according to the following two key update methods:*

- Per-item QoF*: Each individual data item D_j is signed as soon as it is collected.*
- Per-interval QoF*: A group of data item D'_j is signed as a single data item for each time period t_j, where D'_j denotes all individual data items collected in t_j.*

In terms of the key evolving strategy, these two methods are the same. However, they enable users to establish a performance-security trade-off that can be decided according to the requirements of application. That is, *per-item QoF* provides the *highest quality of forward security* (i.e., forward-security of each data item individually), but it incurs *high* computational and storage overhead to the signers and

verifiers. In contrast, *per-interval QoF* provides a *low* quality of forward security (i.e., only for across time periods), but it also incurs *less* computational and storage overhead to the the signers and verifiers.

HaSAFSS Security Objectives: The goal of HaSAFSS schemes is to achieve secure signature aggregation and forward security simultaneously. We follow the example of previous forward-secure and aggregate signature schemes (i.e., [75, 76]), which focus on forward-security, existential unforgeability and authentication properties, to analyze our schemes in Section 3.4.

Remark that HaSAFSS schemes exploit the already existing delays (i.e., time factor) in UWSNs to achieve its desirable properties. Thus, the forward-security objective of HaSAFSS schemes is slightly different than that of previous forward-secure and aggregate schemes (i.e., [75,76,79]). That is, HaSAFSS aims to achieve a *time-valid forward-security* instead of a *permanent* forward-security.

Based on our threat model and security goals, the security of HaSAFSS schemes is defined as follows:

Definition 3.5 *The security of a HaSAFSS scheme is defined as the non-existence of a PPT bounded adversary $\mathcal{A}$ who produces an existential forgery against HaSAFSS even under the exposure of current keying material in the duration of a designated time interval $t_{w,w'}$. This is called as* Time-valid Forward-secure Existential Unforgeability (TFEU) *property.*

Note that HaSAFSS schemes mainly rely on symmetric cryptography to achieve the above goal[1]. Indeed, in Section 5.5, we show that they achieve the highest QoF (i.e., per-item QoF) in a given time interval, and still remain much more computationally efficient than previous PKC-based schemes.

Remark: The time validity requirement in Definition 3.5 implies that a signer should transmit the forward-secure and aggregate signature computed in $t_{w,w'}$ to the verifiers, before the TTP releases the time trapdoor key associated with $t_{w'}$. Such a requirement is compatible with the periodic data collection characteristic of the envisioned UWSN applications [46, 76, 111]. Details of how our schemes handle data/time trapdoor information are given in data models and Section 3.3.

HaSAFSS schemes integrate various cryptographic primitives in a novel and efficient way to achieve their security and efficiency goals. In Section 3.4, we prove that breaking a HaSAFSS scheme is as difficult as breaking its underling primitive(s). Therefore, HaSAFSS schemes achieve *TFEU* as long as the below assumptions hold:

Assumption 3.1 *We assume that cryptographic primitives used in our schemes have all the semantic security properties [52] as follows:*

(i) H_1, H_2, ..., H_6 are secure and collision-free hash functions producing indistinguishable outputs from the random uniform distribution [13]. (ii) MAC is Existential Unforgeable Under Chosen Message Attacks (EU-CMA) [112]. (iii) Symmetric encryption function $\mathcal{E}$ is Indistinguishable under Chosen

[1]Sym-HaSAFSS relies on an omni-symmetric construction, but ECC-HaSAFSS and SU-HaSAFSS consult a few ExpOp *once* to initialize a desired time interval. However, SU-HaSAFSS still uses *only symmetric primitives* to sign/verify each data item in this time interval, and therefore it preserves the computational efficiency of HaSAFSS constructions.

Ciphertext Attacks (IND-CCA secure) [65]. (iv) ECDLP *[56] and* q-BDHI problem *[22] are* (τ, ϵ)*-hard with appropriate parameters.*

Assumption 3.2 *We assume a Trusted Third Party (TTP), which is trusted by all network entities. (i) The adversary* $\mathcal{A}$ *cannot compromise the TTP; (ii)* $\mathcal{A}$ *may jam the TTP, but if an entity continuously tries, its messages can eventually reach the TTP; (iii) The TTP releases* time trapdoor keys *(secret cryptographic keys) with which the receivers verify the forward-secure and aggregate signatures generated by the senders (the TTP acts as a Trusted Agent (TA) as in TRE schemes [41]). We assume that time trapdoor keys released by the TTP reach the receivers eventually. Details of the time trapdoor key delivery are given in the data models.*

3.2.2 Data Model

We consider three data delivery models for the envisioned UWSN applications:

(a) *Pre-determined Data Delivery Model*: This model addresses applications in which signers/verifiers and the TTP can agree on a prospective data delivery schedule so that data/trapdoor delivery can be performed based on this schedule. In this model, the TTP *passively broadcasts* time trapdoor keys based on a *pre-determined schedule*. The verifiers are assumed to be (e.g., mobile collectors) available for their signers based on this schedule.

(b) *On-demand Data Delivery Model*: This model addresses applications where the nature of application does not allow a prospective delivery schedule. In this case, the TTP provides the time trapdoor information to the verifiers on demand. A representative scenario would be a military UWSN application, in which soldiers gather information from sensors from time to time and then request time trapdoor keys from the TTP (e.g., UAV/satellite). Note that in the worst case, verifiers can obtain time trapdoor keys from a mobile TTP directly (e.g., MTC (Mobile Tactical Center) [119]). Thus, Assumption 3.2-(iii) is realistic.

(c) *Self-Decisive Data Delivery Model*: This model offers a flexible data delivery schedule for the signers. It requires neither a prospective data delivery schedule nor a collaborative request mechanism for the time trapdoor release (no interaction with the TTP). Instead, the TTP passively broadcasts time trapdoor keys *periodically* with a unit time μ (e.g., per-minute). Each signer *herself decides* how long (e.g., a time duration $\mu \cdot x$ for any desired $x \in \mathbb{N}$) she needs to accumulate, sign and seal the data (independent from the TTP and other signers). Verifiers decrypt and verify the data when its associated time trapdoor key is released.

This model enables HaSAFSS to address applications that require users to operate long times in hostile environments autonomously. One example would be mobile reconnaissance vehicles (e.g., autonomous LANdroids [46]) that gather information from a target area. Such a vehicle *autonomously* decides sense, sleep and broadcast time durations, and signs and seals the data according to the situation. No entity can modify or recover the data that the vehicle accumulated until the end of designated

time duration, even if it is captured by the enemy.

Remark: (i) Signers do not need to communicate with the TTP. Verifiers communicate with the TTP only in on-demand data delivery model, only once for each time period (the TTP can be offline most of the time). (ii) In HaSAFSS, similar to the previous forward-secure and aggregate schemes (e.g., [75, 76, 79, 120]), the signer computes aggregate signatures of distinct data items accumulated-so-far (i.e., similar to the condensed signatures notion in [91]). Cross-signer signature aggregation (e.g., [21, 26, 90]) is out of our scope.

3.3 Proposed Schemes

We now present the proposed Sym-HaSAFSS, ECC-HaSAFSS and SU-HaSAFSS schemes. Before giving the detailed description, we first present an overview of these schemes, providing instruments and strategies that are common to all HaSAFSS schemes.

3.3.1 Overview

The main goal of the HaSAFSS schemes is to create a forward-secure and aggregate signature scheme, which is as efficient as a MAC-based signature scheme and is publicly verifiable at the same time. Our schemes achieve this goal based on the following observations:

Delay is already intrinsic to the envisioned UWNS applications; such delays can be used to introduce asymmetry naturally between the signer (sender) and the verifiers (receivers) in order to bring both public verifiability and efficiency to the envisioned UWSN applications. HaSAFSS introduces this asymmetry with the aid of TRE concept, instead of offloading this task simply to the signers. Hence, even when the signers are compromised, such asymmetry can still guarantee the forward security and signature aggregation in a publicly verifiable way during the desired time interval.

The HaSAFSS schemes consist of four algorithms: Key generation, forward-secure and aggregate signature generation, time trapdoor release, and forward-secure aggregate signature verification.

HaSAFSS Instruments and Strategies: The HaSAFSS schemes rely on four main types of cryptographic keys; they use these keys in different ways to achieve different properties. There are also other types of cryptographic keys that are specific to a particular HaSAFSS scheme (e.g., public tokens in SU-HaSAFSS), whose details will be given in corresponding schemes.

• *Per-data item key*: Per-data item key is used with a MAC to generate or verify forward-secure and aggregate signatures during a given time interval (a single interval t_w in Sym-HaSAFSS/ECC-HaSAFSS and a unified interval $t_{w,w'}$ in SU-HaSAFSS). We take the unified interval as a basis in this overview. The first per-data item key of a given $t_{w,w'}$, called the *chain root* of the per-data item keys (i.e., k_0) in $t_{w,w'}$, is either derived from an auxiliary key (Sym-HaSAFSS) or randomly generated (ECC-HaSAFSS/SU-HaSAFSS) at the beginning of $t_{w,w'}$.

The signer signs each accumulated data item individually by computing its MAC using the corresponding per-data item key (derived from k_0) and updates her per-data item key with a hash operation (and deletes the previous one). The signer then folds individual signature of the newly collected data item into the existing aggregate signature by concatenating and hashing them together. This strategy provides the forward security of these data items in $t_{w,w'}$. To enable verifiers to publicly verify this signature at the end of $t_{w,w'}$ by following the same procedure as in the signature generation, an asymmetry should be introduced between the signer and verifiers (by preserving the forward security) . This is done via *encrypted chain roots*.

• *Encrypted chain root*: Such asymmetry can be introduced by two conditions. First, k_0 should remain confidential in $t_{w,w'}$. In this way, if the signer is compromised in $t_{w,w'}$, the adversary cannot obtain k_0 before the end of $t_{w,w'}$, and therefore she cannot forge signatures computed via k_0 (k_0 is also updated for each signed data item). At the same time, k_0 should be publicly available to all verifiers at the end of $t_{w,w'}$ so that any entity should be able to verify signatures computed via k_0.

The signer seals k_0 until the end of $t_{w,w'}$ by encrypting it with a *session key* $K_{w,w'}$ as $c_{w,w'} = \mathcal{E}_{K_{w,w'}}(k_0)$, and then deleting $(k_0, K_{w,w'})$.

• *Session keys and time trapdoor keys*: To enable the recovery of k_0 from $c_{w,w'}$ at the end of $t_{w,w'}$, each *session key* $K_{w,w'}$ is transformed by a *time trapdoor key* $tk_{w'}$. HaSAFSS schemes achieve their distinctive properties by following different session key and time trapdoor mechanisms:

In Sym-HaSAFSS and ECC-HaSAFSS, each time trapdoor key is constructed as an element of a hash chain to enable its ExpOp-free verification and session key recovery. These time trapdoor keys are released by following either the synchronous or asynchronous data delivery model (See Section 3.2).

Sym-HaSAFSS pre-computes each encrypted chain root with its corresponding session key and time trapdoor key, and gives them to the signers before the deployment. This omni-symmetric approach allows ExpOp-free session key computation and recovery, but it sacrifices the signer storage efficiency. In contrast, ECC-HaSAFSS allows each signer to randomly generate his own session key and therefore it achieves the signer storage efficiency. However, it requires an ExpOp per-time interval to compute this key, and also incurs a linear public key storage per-signer to the verifiers (i.e., quadratic storage overhead). Despite their computational efficiency, the above mechanisms limit the sustainability and flexibility of Sym-HaSAFSS and ECC-HaSAFSS.

SU-HaSAFSS uses a pairing based time trapdoor key structure, which is inspired by AnTRE [35]. Such a structure allows signers to compute their own session keys and encrypted chain roots without relying on pre-computed public keys. Hence, despite being slightly more costly than Sym-HaSAFSS and ECC-HaSAFSS, SU-HaSAFSS addresses their limitations and also preserves the per-data item efficiency of HaSAFSS constructions over the traditional PKC-based schemes.

After the release of the time trapdoor key, the verifiers never accept any obsolete signature associated with this time trapdoor key from any signer.

3.3.2 Sym-HaSAFSS

The four algorithms of Sym-HaSAFSS is given below:

► *Key Generation*: The TTP performs key generation as follows:

1) Choose the maximum clock synchronization error as δ_t and the trapdoor release times as $0 < T_0 < T_1 < \ldots < T_{L-1}$. Every two consecutive time points T_{i-1} and T_i form the i-th time interval t_i.

2) Randomly generate a hash chain $v_w \leftarrow H_1(v_{w-1})$ for $w = 1, ..., L-1$, whose elements will be used as the secret time trapdoor keys in the reversed order as $tk_w \leftarrow v_{L-1-w}$ for $w = 0, ..., L-1$. Each tk_w is associated with time interval t_w for $w = 0, ..., L-1$. Compute the encrypted chain roots for each signer ID_i as follows:

a) Generate the initial per-interval key $z_0 \xleftarrow{R} \{0,1\}^n$ for each ID_i. The objective of the per-interval key is to provide a fresh initialization key for each time period, from which the signer ID_i will derive the chain root (i.e., per-item key) of that time interval. That is, the chain root of ID_i for each t_w is derived as $k_0 \leftarrow H_2(z_w)$ and $z_{w+1} \leftarrow H_1(z_w)$ for $w = 0, ..., L-1$.

b) Compute the session key as $tk_w \leftarrow H_3(tk_w||ID_i)$ and the encrypted chain root of ID_i for t_w as $c_w \leftarrow \mathcal{E}_{tk_w}(k_0)$ for $w = 0, ..., L-1$.

3) Distribute required keys and the data delivery schedule to each ID_i and verifiers as ID_i : $\{z_0, c_w, T_w, \delta_t\}$ and *Verifiers* : $\{H_1(tk_0), T_w, \delta_t\}$ for $w = 0, \ldots, L-1$, respectively.

► *Time Trapdoor Release*: Time trapdoor release can be executed in two different modes:

(1) *Synchronous Mode:* According to the pre-determined delivery time schedule, at the end of each t_w, the TTP releases the secret time trapdoor key tk_w.

(2) *Asynchronous Mode:* Each verifier sends a request to the TTP for the release of tk_w, when she is done with the data accumulation (or, a mobile TTP visits and requests the data from the verifiers). When the TTP receives more than a threshold number of (authenticated) requests (e.g., $\tau = 90\%$), the TTP releases tk_w.

► *Forward-secure and Aggregate Signature Generation*:

1) At the beginning of t_w, derive the per-data item key as $k_0 \leftarrow H_2(z_w)$, update the per-interval key as $z_{w+1} \leftarrow H_1(z_w)$ and delete z_w from the memory.

2) Assume that the signer ID_i computed $\sigma_{0,l-1}$ on $D_0, \ldots, D_{l-1}$ in t_w. Compute $\sigma_{0,l}$ on new D_l as $(\sigma_l \leftarrow MAC_{k_l}(D_l), \sigma_{0,l} \leftarrow H_3(\sigma_{0,l-1}||\sigma_l))$, where $k_l \leftarrow H_1(k_{l-1})$.

In the synchronous mode, all keys and signatures associated with t_w expire at the end of t_w. Thus, signer ID_i must transmit $pkt = \{ID_i : D_0, \ldots, D_l, \sigma_{0,l}, c_w, t_w\}$ before t_w ends. In the asynchronous mode, signer ID_i can transmit pkt at any time before the TTP releases tk_w. However, if the signer transmits it too late, she may miss the opportunity to have verifiers accept it if the transmission is after the trapdoor release.

► *Forward-secure and Aggregate Signature Verification*:

1) Assume that the verifier has received pkt at time t. In the synchronous mode, the verifier checks

whether the time condition $(t + \delta_t) \leq t_w$ holds for $\sigma_{0,l}$. If yes, the verifier buffers pkt and waits for the end of t_w to obtain tk_w from the TTP. In the asynchronous mode, the verifier sends a request to the TTP to obtain tk_w. Note that due to the nature of UWSN applications, there may be a delay before this request is delivered to the TTP (or, the TTP might not be able to visit the verifiers for a long time). In this mode, the verifier can buffer pkt as long as it is received before the release of tk_w.

2) When the TTP releases tk_w, each verifier verifies tk_w by checking whether $tk_w = H_1(tk_{w-1})$ holds. If tk_w is verified, then the verifier verifies $\sigma_{0,l}$ as follows:

The verifier decrypts c_w by computing $tk_w \leftarrow H_3(tk_w||ID_i)$ and $k_0 \leftarrow \mathcal{D}_{tk_w}(c_w)$. Using the per-data item key, the verifier computes individual signatures of D_j as $\sigma'_j \leftarrow MAC_{k_j}(D_j)$ and $k_{j+1} \leftarrow H_1(k_j)$ for $j = 0, \ldots, l$. Finally, the verifier computes $\sigma'_{0,j} \leftarrow H_3(\sigma'_{0,j-1}||\sigma'_j)$ for $j = 1, \ldots, l$, where $\sigma'_{0,0} = \sigma'_0$, and checks if $\sigma'_{0,l} = \sigma_{0,l}$ holds. If they match, the verifier accepts $\sigma_{0,l}$; otherwise, reject.

As a result, only using the cryptographic hash and symmetric encryption functions, Sym-HaSAFSS generates publicly verifiable, forward-secure and aggregate signatures. Signature generation/verification cost of a single data item in Sym-HaSAFSS is *only three hash operations*, which are extremely efficient when compared with all PKC-based alternatives. This optimal computational efficiency of Sym-HaSAFSS makes it an ideal choice for resource-constrained UWSN applications.

3.3.3 ECC-HaSAFSS

In contrast to Sym-HaSAFSS, ECC-HaSAFSS addresses the applications where the signers are storage limited while the receivers can afford certain storage [76]. To achieve this, ECC-HaSAFSS uses *pre-computed public keys* instead of pre-computed encrypted chain roots, and it enables each signer to compute her own session keys after the deployment.

In ECC-HaSAFSS, the TTP generates the initial per-interval key r_0 for each signer i before the system deployment. Each signer i then updates the per-interval key at the beginning of each time interval t_w and computes the session key K_w using the per-interval key r_w with an ECC scalar multiplication. Signer i then randomly generates a per-data item key k_0 (i.e., the first per-data item key in t_w). To protect k_0 in t_w, signer i encrypts it with K_w to obtain the encrypted chain root c_w. After this stage, signer i computes the signature using the per-data item k_0 following the signature generation step 2 in Sym-HaSAFSS.

To verify the signature, a verifier first recovers K_w from the public key of signer i (i.e., V_w) using tk_w with an ECC scalar multiplication. Note that $(V_0, \ldots, V_{L-1})$ of signer i are pre-computed by the TTP before the deployment to enable such a recovery via tk_w. The verifier then decrypts the per-data item key of signer i and verifies the signature following the same steps of the signature generation.

► *Key Generation*: The TTP generates tk_w of each t_w for $w = 0, \ldots, L-1$ by following the Sym-HaSAFSS initialization steps. The TTP then generates the public key of each ID_i for each t_w as follows:

Generate the initial per-interval key as $r_0 \xleftarrow{R} \mathbb{Z}_q^*$, and compute the public key of each t_w as $V_w \leftarrow tk_w(r_wG - tk_w(\alpha_wG))$, where $(\alpha_wG \leftarrow H_4(tk_w), r_{w+1} \leftarrow H_1(r_w))$ for $w = 0, \ldots, L-1$. Give each signer ID_i her own r_0, and give V_w of each ID_i for $w = 0, \ldots, L-1$ to all verifiers.

► *Time Trapdoor Release*: Same as in Sym-HaSAFSS.

► *Forward-secure Aggregate Signature Generation*:

1) At the beginning of t_w, signer ID_i randomly generates a per-data item key k_0, and computes the session key as $K_w \leftarrow H_1(r_wG)$ and then the encrypted chain root as $c_w \leftarrow \mathcal{E}_{K_w}(k_0)$. She updates the per-interval key as $r_{w+1} \leftarrow H_1(r_w)$ and deletes (K_w, r_w) from the memory.

2) Signer ID_i computes $\sigma_{0,l}$ on $(D_0, \ldots, D_l)$ for t_w using k_0 by following step 2 in Sym-HaSAFSS signature generation, and then broadcasts $pkt = (D_0, D_1, \ldots, D_l, \sigma_{0,l}, c_w, t_w, ID_i)$ before the end of t_w.

► *Forward-secure and Aggregate Signature Verification*: When a verifier receives pkt, she first checks timing/request conditions for the received packet and verifies tk_w upon its receipt as in step 2 Sym-HaSAFSS signature verification. The verifier then recovers the session key as $K_w \leftarrow H_1(tk_w^{-1}V_w + H_4(tk_w))$ and decrypts the per-data item key as $k_0 \leftarrow \mathcal{D}_{K_w}(c_w)$. The verifier verifies $\sigma_{0,l}$ using k_0 by following step 2 in Sym-HaSAFSS signature verification.

3.3.4 SU-HaSAFSS

To explain the intuition behind SU-HaSAFSS, we first discuss the limitations of Sym-HaSAFSS and ECC-HaSAFSS.

► *Key pre-distribution and limited usage:* In Sym-HaSAFSS, encrypted chain roots are directly computed from the time trapdoor keys. Similarly, in ECC-HaSAFSS, public keys are a function of time trapdoor keys. Hence, in both schemes, these keys have to be pre-computed and distributed before the deployment.

The above requirement incurs a linear storage overhead to the signers in Sym-HaSAFSS, and a quadratic storage overhead to the verifiers in ECC-HaSAFSS. In both cases, the storage overhead *grows linearly* with the maximum number of time period (i.e., L). Furthermore, this puts a *linear bound* on the maximum number of time periods that a signer can use. Once the pre-computed values are depleted, the TTP needs to replenish them via an authenticated channel. The nature of some applications might not allow such a re-initialization, and even if possible, it incurs $O(L)$ communication overhead for each signer.

► *Inflexible data delivery schedule*: In Sym-HaSAFSS and ECC-HaSAFSS, time trapdoor keys cannot be derived from a desired time instance, and therefore have to be either released based on a pre-determined data delivery schedule, or requested collaboratively by the verifiers. In either case, signers cannot decide their own data delivery schedule independent from the TTP or the verifiers.

SU-HaSAFSS Strategy

SU-HaSAFSS enables each signer to compute her own key set *without requiring any online coordination* with the TTP or verifiers. This self-sustaining approach does not put any upper bound on the maximum number time period to be used, and therefore achieves high storage efficiency (i.e., $O(1)$ storage for the signer, and $O(S')$ storage for the verifiers). It also allows a signer to decide *her own* data delivery schedule independently. That is, a signer can sign the data items not in a pre-determined time interval t_w, but in a time interval $t_{w,w'}$ for *any* $w' \geq w$.

SU-HaSAFSS achieves these goals as follows:

► *Key Generation:* The TTP provides each signer a master public key S and a master token V, with which the signer can initialize an interval $t_{w,w'}, w' \geq w$. To do this, the signer first randomly generates a chain root k_0, which will be used to sign and encrypt data items accumulated in $t_{w,w'}$. The signer then generates a session key $K_{w,w'}$ using a random number r_w and token V, and then seals k_0 as $c_{w,w'} \leftarrow \mathcal{E}_{K_{w,w'}}(k_0)$.

To enable the recovery of k_0 at the end of $t_{w,w'}$ (with $K_{w,w'}$), the signer also computes an auxiliary token $Z_{w,w'}$ with $(r_w, T_{w'}, S)$ via two scalar multiplications. $Z_{w,w'}$ serves as the masked version of $K_{w,w'}$, and its computation does not require the knowledge of trapdoor keys. Once the signer erases $(r_w, K_{w,w'})$ from the memory, no entity including the signer herself can recover k_0 before the end of $t_{w,w'}$. In this way, SU-HaSAFSS introduces the desired asymmetry between signer and verifiers.

► *Forward-secure Aggregated Signature Generation (and Encryption):* Assume that the adversary $\mathcal{A}$ breaks-in at time t during $t_{w,w'}$. In contrast to Sym-HaSAFSS and ECC-HaSAFSS, the above self-sustaining strategy allows $\mathcal{A}$ to initialize a new key set independent from the current one (chain roots are no longer generated by the TTP). Therefore, $\mathcal{A}$ can compute a different signature on the data items accumulated in $[T_{w-1}, t]$ apart from the existing signature (note that $\mathcal{A}$ still cannot forge the existing aggregate signature computed in $[T_{w-1}, t]$).

SU-HaSAFSS prevents this by using a symmetric cipher along with the forward-secure MAC strategy (i.e., Step 2 in Sym-HaSAFSS signature generation). That is, each D_j is both signed and then encrypted with k_j, and (D_j, k_j) are deleted from the memory. Since k_0 is sealed until the end of $t_{w'}$, $\mathcal{A}$ cannot decrypt $(D_0, \ldots, D_j)$ accumulated in $[T_{w-1}, t]$, and therefore cannot compute a different signature on them.

Another advantage of this approach is that it offers forward-secure encryption and signature simultaneously via symmetric cryptography. Therefore, it is significantly more efficient than all existing forward-secure signcryption[2] schemes (e.g., [40]) with the limitation that it cannot achieve immediate verification.

► *Time Trapdoor Release and Signature Verification*: To recover $K_{w,w'}$ at the end of $t_{w,w'}$, we use a pairing-based time trapdoor key structure, which was inspired by AnTRE [35]. Such a time trapdoor

[2]Signcryption is a PKC primitive that simultaneously performs the functions of both digital signature and encryption [125].

key structure allows the derivation of all time trapdoor keys from a single master secret key s without revealing it. When $tk_{w'}$ is released at the end of $t_{w,w'}$, the verifier first removes the mask of $Z_{w,w'}$ using $tk_{w'}$ via a pairing operation (i.e., Step 2 in SU-HaSAFSS signature verification). The verifier obtains k_0 as $k_0 \leftarrow \mathcal{D}_{K_{w,w'}}(c_{w,w'})$, and then both decrypt and verify data items using k_0.

► *Efficiency*: SU-HaSAFSS preserves the computational efficiency of HaSAFSS construction over the traditional PKC-based schemes, since the per-item cost is still only three hash operations as in Sym-HaSAFSS[3].

Detailed Description

► *Key Generation*: Executed by the TTP as follows:

1) Choose δ_t and the time trapdoor release period as μ (i.e, a unit time such as one hour). Every two consecutive time points $T_{i-1} \leftarrow (i-1)\mu$ and $T_i \leftarrow i \cdot \mu$ form the i-th time interval t_i for $i > 0$, and $t_{w,w'}$ denotes a unified time interval beginning from t_w to the end of $t_{w'}$.

2) Generate a master private/public key pair and a token as $(s \xleftarrow{R} \mathbb{Z}_q^*, S = sG)$ and $V = \widehat{e}(G,G) \in \mathbb{G}_2$, respectively. Also generate a private/public key pair as $(\overline{sk}, \overline{pk}) \leftarrow SGN.Kg(1^\kappa)$ that will be used to sign or verify time trapdoor keys.

3) Generate a MMM private/public key pair for each ID_i as $(sk_0, pk) \leftarrow MMM.Kg(1^\kappa)$, and then distribute the required keys as $ID_i : \{S, V, sk_0, pk, G, \widehat{e}, q, \delta_t, \mu\}$ and $Verifiers : \{\overline{pk}, \forall i, ID_i : pk, \delta_t, \mu\}$.

► *Forward-secure Aggregated Signature Generation*:

1) The signer ID_i initializes an interval $t_{w,w'},\ w' \geq w$:

a) Compute the session key of $t_{w,w'}$ as $K_{w,w'} \leftarrow H_6(V^{r_w})$, where $r_w \xleftarrow{R} \mathbb{Z}_q^*$. Also compute the auxiliary token for $t_{w,w'}$ as $Z_{w,w'} \leftarrow r_w S + r_w H_5(T_{w'})G$, where $T_{w'} \leftarrow \mu \cdot w'$ (i.e., the end of $t_{w,w'}$).

b) Generate $k_0 \xleftarrow{R} \{0,1\}^n$ and compute $c_{w,w'} \leftarrow \mathcal{E}_{K_{w,w'}}(k_0)$ for $t_{w,w'}$, and securely erase $(r_w, K_{w,w'})$ from the memory.

c) Compute $\overline{c}_{w,w'} \leftarrow MMM.Sig_{sk_w}(c_{w,w'}||Z_{w,w'}||t_w||t_{w'}||w||ID_i)$. Update sk_w following the MMM key update procedure.

2) Given the current $(\widehat{D}_0, \ldots, \widehat{D}_{l-1}, \sigma_{0,l-1})$, compute $\widehat{D}_l$ and $\sigma_{0,l}$ on newly collected data item D_l as follows:

a) Compute $\sigma_{0,l}$ on $(D_l, \sigma_{0,l-1})$ with k_l by following step 2 in Sym-HaSAFSS signature generation.

b) Compute $\widehat{D}_l \leftarrow \mathcal{E}_{k_l}(D_l)$, update $k_{l+1} \leftarrow H_1(k_l)$, and securely erase (D_l, k_l) from the memory. Broadcast $pkt = \{ID_i : \widehat{D}_0, \ldots, \widehat{D}_l, \sigma_{0,l}, c_{w,w'}, Z_{w,w'}, t_w, t_{w'}, w, \overline{c}_{w,w'}\}$ before the end of $t_{w,w'}$.

[3]The costs of ExpOps required to initialize $t_{w,w'}$ are amortized even in a short term, since the overall cost is dominated by the per-item cost.

► *Trapdoor Release*: The TTP computes the time trapdoor key corresponding to t_w as $tk_w \leftarrow (s + H_5(T_w))^{-1}G$, where $T_w = \mu \cdot w$. The TTP then signs it as $\overline{tk_w} \leftarrow SGN.Sig_{\overline{sk}}(tk_w||w)$, and broadcasts $(tk_w, \overline{tk_w}, w)$ at the end of each time period t_w periodically[4].

► *Signature Verification and Decryption*: Assume that the verifier received pkt at time t:

1) If $(t+\delta_t) > t_{w'}$ then *abort*. Otherwise, if $\{failure\} = MMM.Ver_{pk}(\overline{c}_{w,w'}, \langle c_{w,w'}||Z_{w,w'}||t_w||t'_w||w||ID_i||\rangle)$ holds then *abort*. Otherwise, buffer pkt and wait the release of $tk_{w'}$.

After $tk_{w'}$ is received from the TTP, if $\{failure\} = SGN.Ver_{\overline{pk}}(\overline{tk_{w'}}, tk_{w'}||w)$ holds then *abort*, else continue to the next step.

2) Recover the session key as $K_{w,w'} \leftarrow H_6(\widehat{e}(Z_{w,w'}, tk_{w'}))$, and decrypt $k_0 \leftarrow \mathcal{D}_{K_{w,w'}}(c_{w,w'})$.

3) Decrypt data items as $D_j \leftarrow \mathcal{D}_{k_j}(\widehat{D}_j)$ for $j = 0, \ldots, l$, and verify $(D_0, \ldots, D_l, \sigma_{0,l})$ with k_0 by following step 2 in Sym-HaSAFSS signature verification.

3.4 Security Analysis

We prove the security of HaSAFSS schemes in three stages:

Lemma 3.1 proves that no entity, including the signer and the adversary $\mathcal{A}$ even after the break-in in $t_{w,w'}$, can decrypt $c_{w,w'}$ without knowing its corresponding time trapdoor key $tk_{w'}$. That is, no entity can obtain the chain root k_0 before the release of $tk_{w'}$. This guarantees that HaSAFSS schemes introduce the desired asymmetry between the signer and verifiers amd preserves forward security.

Based on Lemma 3.1, Lemma 3.2 proves that HaSAFSS schemes remain forward-secure and existential unforgeable during interval $t_{w,w'}$ by regularly updating per-item keys evolved from chain root k_0.

Finally, Theorem 3.1 proves that the successful verification of $\sigma_{0,l}$ via k_0 guarantees *TFEU* property (i.e., Definition 3.5) based on Lemma 3.2.

Lemma 3.1 *HaSAFSS schemes guarantee the confidentiality of k_0 in the time duration between the releases of tk_{w-1} and $tk_{w'}$ as long as Assumptions 3.1 and 3.2 hold.*

Proof: Assume that $\mathcal{A}$ breaks-in during the interval $t_{w,w'}$. Obtaining k_0 from $c_{w,w'}$ without knowing its corresponding session key $K_{w,w'}$ is as difficult as breaking $\mathcal{E}$. It is therefore sufficient to show that $K_{w,w'}$ remains confidential until the end of $t_{w,w'}$ (i.e., until its corresponding time trapdoor key $tk_{w'}$ is released):

Sym-HaSAFSS: In Sym-HaSAFSS, $w = w'$. Thus, $K_{w,w'} = K_w$ and $c_{w,w'} = c_w$. For a given tk_{w-1}, computing $K_w = H_3(tk_w||ID_i)$ without knowing tk_w is as difficult as inverting H_1 since $tk_w = H_1(tk_{w-1})$. This contradicts with Assumption 1-(i).

[4]In SU-HaSAFSS, each signer can seal its own data independent from each other for different time periods, and untimely release of a time trapdoor key might expose several signers' data before their intended time. Therefore, the asynchronous mode is not used in SU-HaSAFSS.

SU-HaSAFSS: For given $(S, V, t_{w'})$, obtaining $K_{w,w'}$ without knowing $tk_{w'}$ is as difficult as solving *q-BDHI problem*:

Assume that $\mathcal{A}$ outputs a session key K^* before the release of $tk_{w'}$ in a polynomial time τ with a non-neglible probability ϵ such that it correctly decrypts its corresponding per-data item key as $k_0 \leftarrow \mathcal{D}_{K^*}(c_{w,w'})$. This implies $K^* = K_{w,w'}$ and $K_{w,w'} = H_6(\hat{e}(Z_{w,w'}, tk^*_{w'}))$ holds for $tk^*_{w'}$. This means $\mathcal{A}$ also non-trivially computed a valid time trapdoor key $tk^*_{w'}$.

We then verify that $\widehat{e}(tk^*_{w'}, (S + H_5(T_{w'})G)) = 1$, and therefore $tk^*_{w'} = (s + H_5(T_{w'}))^{-1}G$. This implies that $\mathcal{A}$ solved *q-BDHI problem*, and this contradicts with Assumption 3.1-(iii).

ECC-HaSAFSS: For given V_w, obtaining K_w without knowing tk_w is as difficult as solving *ECDLP problem*:

Assume that $\mathcal{A}$ outputs a session key K^* before the release of tk_w in τ with ϵ such that $k_0 = \mathcal{D}_{K^*}(c_w)$ holds. This implies $K^* = K_w$, and therefore $\mathcal{A}$ non-trivially computed a valid time trapdoor key tk^*_w such that $K_w = H_1((tk^*_w)^{-1}V_w + H_4(tk^*_w))$. Hence, $tk^*_w = tk_w$ and $\mathcal{A}$ extracted tk_w from V_w. This contradicts with Assumption 3.1-(iii). □

Lemma 3.2 *Assume that $\mathcal{A}$ breaks-in at time t during interval $t_{w,w'}$, after $\sigma_{0,l}$ on $(D_0, \ldots, D_l)$ was computed. Producing an existential forgery against HaSAFSS in the time duration between the releases of tk_{w-1} and $tk_{w'}$ is as difficult as breaking either one of the cryptographic hash functions (H_1, H_2, H_3) or* MAC.

Proof: Lemma 3.1 guarantees that k_0 remains confidential until the end of $t_{w,w'}$. At the same time, the signer regularly updated k_0 for each accumulated data item until $\mathcal{A}$ breaks-in at time t:

Sym-HaSAFSS and ECC-HaSAFSS: Step 2 in Sym-HaSAFSS signature generation updated per-interval and per-item keys as $(z_{l+1} \leftarrow H_2(z_l), k_{l+1} \leftarrow H_1(k_l))$, respectively, and then deleted (z_l, k_l) from the memory. Obtaining any previous per-interval key from z_{l+1} is as difficult as breaking H_2. Similarly, obtaining any previous per-item key from k_{l+1} is as difficult as breaking H_1. Without knowing $(k_0, \ldots, k_l)$, forging $\sigma_{0,l}$ on $(D_0, \ldots, D_l)$ is as difficult as breaking *MAC* function or H_3 (selectively deleting or truncating a data item from $(D_0, \ldots, D_l)$ is subsumed in this forgery). Therefore, Sym-HaSAFSS remains forward-secure and existential unforgeable in $t_{w=w'}$. The signature generation in ECC-HaSAFSS is identical to that of Sym-HaSAFSS, and therefore this analysis also applies to it.

SU-HaSAFSS: Step 2 in SU-HaSAFSS is identical to that of Sym-HaSAFSS except that it additionally encrypts the data items as $\widehat{D}_j \leftarrow \mathcal{E}_{k_j}(D_j)$. Thus, producing a forgery against SU-HaSAFSS is as difficult as breaking either $\mathcal{E}$ or one of (MAC, H_1, H_2, H_3). Similarly, computing an independent valid signature on $(D_0, \ldots, D_l)$ apart from $\sigma_{0,l}$ is as difficult as breaking $\mathcal{E}$. Hence, SU-HaSAFSS remains forward-secure and existential unforgeable in $t_{w,w'}$. □

Theorem 3.1 *The verifier receives packet pkt in time t. The successful verification of $\sigma_{0,l}$ on $(D_0, \ldots, D_l)$ guarantees* TFEU *property (Definition 3.5) in the time duration between the releases of tk_{w-1} and $tk_{w'}$.*

Proof: The verifier should ensure the freshness and authenticity of k_0 before proceeding to the verification:

- *Freshness*: The timing condition $(t+\delta_t) < t_{w'}$ (and also the request condition of the asynchronous mode in Sym-HaSAFSS and ECC-HaSAFSS) prevents the verifier from accepting any obsolete signature associated with $tk_{w'}$. That is, if $\mathcal{A}$ breaks-in after the release of $tk_{w'}$, she cannot compute a "valid" signature on $(D_0, \ldots, D_l)$ using any key associated with $t' \leq t_{w'}$.

- *Authenticity*: k_0 is obtained from $c_{w,w'}$ via $tk_{w'}$.

- *Sym-HaSAFSS and ECC-HaSAFSS*: $\forall(w = w'), tk_w$ can easily be verified, since they are elements of a hash chain and are released in the reverse order. Since tk_w is authenticated, only an authenticated k_0 can be recovered correctly from c_w via this time trapdoor key. Therefore, the successful verification of $\sigma_{0,l}$ with k_0 also implies that only the claimed ID_i could compute such $\sigma_{0,l}$ before the release of tk_w.

- *SU-HaSAFSS*: $\forall w', tk_{w'}$ is verified with $\overline{pk}$ via SGN to ensure its origin and integrity. Similarly, $\overline{c}_{w,w'}$ is verified with pk via MMM to ensure the forward-secure integrity and origin of $(c_{w,w'}, Z_{w,w'}, t_w, t_{w'})$. That is, the verifier ensures that the claimed interval $t_{w,w'}$ is correct and $(c_{w,w'}, Z_{w,w'})$ are intact.

Based on Lemma 3.2 and the fact that k_0 is fresh and authenticated, we prove that HaSAFSS schemes achieve the *TFEU* property. □

3.4.1 Discussion

Truncation Attack: Another security property related to forward-secure and aggregate signatures is the defense against *truncation* attack identified in [78, 79]. Truncation attack is a special type of deletion attack, in which $\mathcal{A}$ deletes a continuous subset of accumulated data items. This attack can be prevented via "all-or-nothing" property [76]: $\mathcal{A}$ should either retain all previously accumulated data items, or not use them at all (i.e., $\mathcal{A}$ cannot selectively delete/modify any subset of the data [79]). Lemma 3.2 proves that HaSAFSS schemes are secure against any type of deletion attack including the truncation attack.

Lack of Immediate Verification: Despite all the advantages, introducing asymmetry between the signer and verifiers using the time factor brings a natural complication: HaSAFSS schemes cannot provide immediate verification on the verifier side. In order to verify a received signature, a verifier needs to wait for the release of the time trapdoor key corresponding to this signature. However, such a property is compatible with the non-real-time nature of the envisioned UWSN applications. Thus, HaSAFSS schemes are ideal solutions for the envisioned UWSN applications. Note, however, that while delayed detection is intrinsic for UWSNs, it might pose a treat for certain real-life applications such as secure logging [79].

Table 3.1: Notation used in the performance analysis and comparison of HaSAFSS and FssAgg schemes

Exp: Modular exponentiation mod p	Enc/Dec: Symmetric enc./dec.	L: # of time periods
$EMul$: ECC scalar multiplication over F_p	S'/V': # of senders/verifiers	w: Current time period
$Muln$: Modular multiplication mod n	ℓ: # of data items	PR: ECC pairing operation
Sqr: Squaring mod n	H: Hash operation	x: FssAgg security parameter
$\|\sigma\|, \|sk\|, \|pk\|$: Bit lengths of signature, private key and public key of the given scheme, respectively.		

Suggested bit lengths to achieve 80-bit security for the above parameters are as follows for each compared scheme: Large primes ($|p| = 512, |q| = 160$) for ECC-HaSAFSS, SU-HaSAFSS and FssAgg-BLS. Integers ($|n| = 1024, x = 160$) for FssAgg-AR and FssAgg-BM, where n is Blum-Williams integer [75].

Table 3.2: Analytical comparison of HaSAFSS and FssAgg schemes in terms of dominant cryptographic operations

	HaSAFSS			FssAgg			
	Sym	SU	ECC	BLS	AR	BM	MAC
Signer	$(3H)l$	$5EMul + (4H + Enc)l$	$EMul + (3H)l$	$(Exp + H)l$	$(3x \cdot Sqr + \frac{x}{2}Muln)l$	$(x \cdot Sqr + \frac{x}{2}Muln)l$	$(3H)l$
Verifier	$(3H)l$	$4EMul+PR+ (4H + Dec)l$	$EMul + (3H)l$	$(PR + H)l$	$x(L + l)Sqr + (l + \frac{x}{2})Muln$	$L \cdot Sqr + (2l + l \cdot x)Muln$	$(3H)l$

HaSAFSS schemes require only three hash operations per-item while FssAgg schemes require at least one ExpOp per-item (initial ExpOps to start given time interval in SU-HaSAFSS and ECC-HaSAFSS become insignificant even for small l values (e.g., $l = 10$)). Also, in HaSAFSS, both signers and verifiers equally enjoy this computational efficiency (extra $PR + EMul$ in SU-HaSAFSS in the initialization also becomes negligible asymptotically).

In HaSAFSS schemes, the TTP is assumed to be trusted (i.e., it does not act maliciously against legitimate users). Therefore, the adversary models that include "curious time server" (e.g., [35]) do not apply to HaSAFSS. This allows us to simplify the time trapdoor mechanism used in [35].

3.5 Performance Analysis

In this section, we present the performance analysis of HaSAFSS schemes and compare them with FssAgg schemes (best known alternatives) in terms of their quantitative and qualitative properties. We use the notation in Table 3.1 for our analysis and comparison. In our experimental evaluation, we use ECDSA [6] as SGN and the base signature scheme for MMM in SU-HaSAFSS.

3.5.1 Computational Overhead

In all HaSAFSS schemes, the cost of signing a single data item is only three hash operations (i.e., overall cost for l data items accumulated in $t_{w,w'}$ is $(3H)l$). While Sym-HaSAFSS does not require any ExpOp, ECC-HaSAFSS and SU-HaSAFSS need to perform $EMul$ and $5EMul$ operations, respectively, but only once at the beginning of $t_{w,w'}$ for the initialization purpose (the rest of the signature generation is

Table 3.3: Execution time (in ms) comparison of HaSAFSS and FssAgg schemes

		HaSAFSS			FssAgg			
		Sym	SU	ECC	BLS	AR	BM	MAC
Signer	$l = 10$	0.06	7.83	0.63	10.2	264	128	0.06
	$l = 10^2$	0.6	8.55	1.35	140	25.8×10^2	12.7×10^2	0.6
	$l = 10^3$	6.1	17.11	8.55	11.8×10^2	26.6×10^3	12.5×10^3	6
	$l = 10^4$	61.2	106.21	80.9	11.9×10^3	26.3×10^4	12.4×10^4	60
Verifier	$l = 10$	0.06	17.88	0.63	156	77.1×10^3	524	0.06
	$l = 10^2$	0.6	18.6	1.35	15.4×10^2	78.4×10^3	920	0.6
	$l = 10^3$	6.1	27.69	8.55	14.9×10^3	88.2×10^3	51.6×10^2	6
	$l = 10^4$	61.2	118.01	80.9	15.2×10^4	18.1×10^4	46.6×10^3	60

(i) The execution times were measured on a computer with an Intel(R) Core(TM) i7 Q720 at 1.60 GHz CPU and 2GB RAM running Ubuntu 10.10. We tested HaSAFSS schemes, FssAgg-BLS/MAC [76] using the MIRACL library [106], and FssAgg-AR/BM [75] using the NTL library [108]. Parameter sizes determining the execution times of each scheme were selected to achieve 80-bit security, whose suggested bit lengths were discussed in Table 3.1. (ii) Execution times are based on the cost of signing/verifying data items accumulated in a given interval $t_{w,w'}$ including the initialization costs.

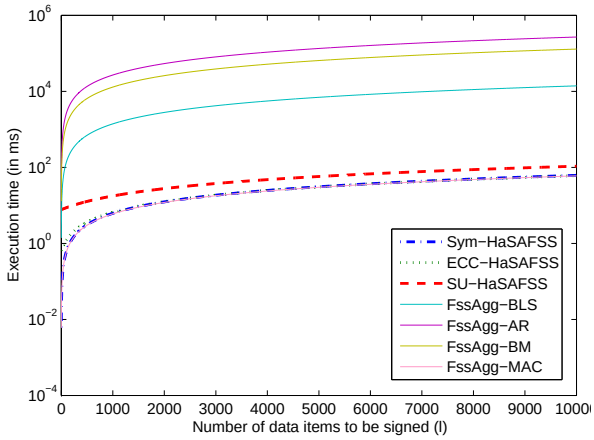

Figure 3.1: Signing time comparison of HaSAFSS schemes and their counterparts (in ms)

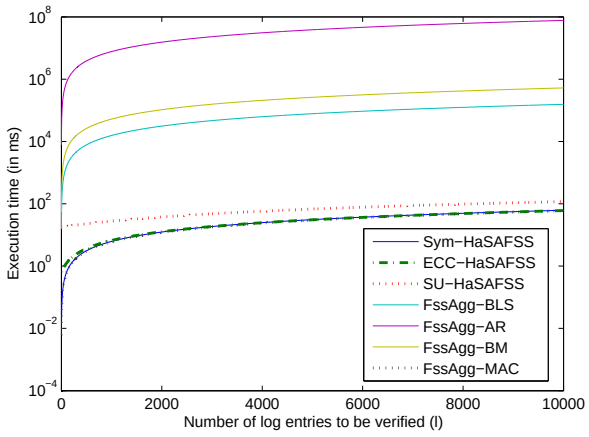

Figure 3.2: Verification time comparison of HaSAFSS schemes and their counterparts (in ms)

only hash-based). The analysis of signature verification cost is similar to the signature generation except that SU-HaSAFSS requires an additional $PR + Emul$ operation for the initialization.

Comparison: All publicly verifiable (PKC-based) FssAgg schemes require ExpOp(s) to sign or verify a data item. For example, FssAgg-BLS requires $O(l)(\ Exp{+}H)$ and $O(l)(PR{+}H)$ for the signature generation and verification, respectively. Similarly, FssAgg-AR and FssAgg-BM require $O(l)ExpOp$ for the signature generation and verification.

Table 3.2 and Table 3.3 compare the computational costs of HaSAFSS schemes with FssAgg schemes analytically and numerically, respectively.

In HaSAFSS schemes, the cost of signature generation and verification for a single data item is the

Table 3.4: Asymptotic comparison of HaSAFSS and FssAgg schemes in terms of their storage overheads

-	HaSAFSS			FssAgg			
	Sym	SU	ECC	BLS	AR	BM	MAC
Signer	$O(L-w)\|H\|$	$O(1)(\|sk\|+c\|H\|)$	$O(1)(\|H\|+\|q\|)$	$O(1)(\|sk\|+\|\sigma\|)$	$O(1)(z\|sk\|+\|\sigma\|)$		$O(V')\|H\|$
Verifier	$O(1)\|H\|$	$O(S')(pk')$	$O(L\cdot S')\|p\|$	$O(L\cdot S')\|p\|$	$O(S')\|n\|$		$O(S')\|H\|$

Table 3.5: Comparison of HaSAFSS and FssAgg schemes in terms of some important qualitative properties

-	HaSAFSS			FssAgg			
	Sym	SU	ECC	BLS	AR	BM	MAC
Public Verifiability	✓	✓	✓	✓	✓	✓	X
Unbounded Time Periods	X	✓	X	X	X	X	X
Forward-secure Confidentiality	X	✓	X	X	X	X	X
Flexible Delivery Schedule	X	✓	X	✓	✓	✓	✓
Signer Storage Efficient	X	✓	✓	✓	✓	✓	X
Verifier Storage Efficient	✓	✓	X	X	✓	✓	✓
Immediate Verification	X	X	X	✓	✓	✓	✓

same (i.e., only three hash operations). This is much more efficient than PKC-based FssAgg schemes requiring at least one ExpOp per-item and also equally efficient to the FssAgg-MAC. For instance, the signature generation for $l = 10^4$ data items with SU-HaSAFSS is 112, 2,476, and 1,167 times more efficient than FssAgg-BLS, FssAgg-AR and FssAgg-BM, respectively. Similarly, the signature verification for $l = 10^4$ data items with SU-HaSAFSS is 1,288, 1,533, and 394 times more efficient than FssAgg-BLS, FssAgg-AR and FssAgg-BM, respectively.

Note that HaSAFSS schemes are always more computationally efficient than any PKC-based scheme that requires an ExpOp per-item. Thus, by specifically comparing HaSAFSS schemes with FssAgg schemes, we can see their difference from this general class of schemes.

Sym-HaSAFSS and FssAgg-MAC are equally efficient, while ECC-HaSAFSS and SU-HaSAFSS are more costly than FssAgg-MAC due to their initialization costs. However, HaSAFSS schemes and PKC-based FssAgg schemes have the advantage of being publicly verifiable against FssAgg-MAC, which is a critical requirement for large and ubiquitous systems.

While being more costly at initialization, SU-HaSAFSS is comparable with Sym-HaSAFSS and ECC-HaSAFSS asymptotically, and it also possesses several qualitative advantages over them as we will discuss in Section 3.5.3.

Figure 3.1 and Figure 3.2 further show the comparison of HaSAFSS and FssAgg schemes in terms of signature generation and verification times as the number of data items (log entries) increases. These figures also confirm that HaSAFSS schemes are the most computationally efficient schemes among all these choices.

3.5.2 Storage and Communication Overheads

Besides their computational efficiency, HaSAFSS schemes are also storage/ bandwidth efficient and complement each other in terms of their storage overhead.

In Sym-HaSAFSS, each signer initially stores L encrypted chain roots. As the time goes from one period into the next, the signer deletes the encrypted chain root associated with the previous time period from her memory. Thus, each signer stores $(L - w)$ keys in t_w. However, each verifier always *stores only a single key* (negligible $|H|$ overhead, e.g., 160 bit). In ECC-HaSAFSS, each signer stores *only one key*, however, in order to recover session keys, each verifier stores L public keys for each signer (i.e., quadratic storage overhead as $O(L \cdot S')|p|$).

In SU-HaSAFSS, each signer is capable of computing her own key set after the deployment (independent from the TTP). Therefore, the key/signature storage of a signer is constant including the overhead due to generic signature and MMM signatures (i.e., $O(1)(|sk| + c|H|)$). Each verifier stores only one MMM public key (i.e., $|pk'|$) for each signer (and one extra public key to verify time trapdoor keys). Thus, in contrast to ECC-HaSAFSS, the storage overhead of a verifier is linear as $O(S')|pk'|$.

Comparison: Table 3.4 asymptotically compares HaSAFSS and FssAgg schemes in terms of storage overhead.

From a verifier's perspective, Sym-HaSAFSS, which requires only single key storage, is the most storage efficient scheme among all the compared schemes. SU-HaSAFSS and FssAgg schemes both require linear storage. ECC-HaSAFSS and FssAgg-BLS require quadratic storage and obey the traditional resourceful verifier assumption to address such UWSN applications (e.g., high-end mobile receivers [76]). From a signer's perspective, all compared schemes except for Sym-HaSAFSS and FssAgg-MAC require constant storage.

All compared schemes incur only a constant signature transmission overhead due to their signature aggregation property. Thus, when compared with traditional signature schemes (e.g., [6, 102, 105]), they are much more communication efficient (data items has to be transmitted in any case and therefore their overhead is not the part of comparison). Note that the signature aggregation also offers "all-or-nothing" property that provides the resilience against the truncation attacks as discussed in Section 3.4.1.

3.5.3 Sustainability, Applicability and Flexibility

In addition to the above quantitative criteria, we also analyze our schemes in terms of some important qualitative properties. Table 3.5 compares HaSAFSS schemes and FssAgg schemes in terms of the following properties:

Public Verifiability: This property is especially important for the scalability and applicability of a scheme to the large and distributed UWSNs. All compared schemes achieve public verifiability with the exception of FssAgg-MAC.

Unbounded Time Period and Flexible Data Delivery Schedule: All compared schemes with the exception of SU-HaSAFSS puts a linear bound on the number of time periods (and implicitly number of data items to be signed) that a signer can use after the system initialization. Eliminating this limitation, SU-HaSAFSS offers a unique sustainability that can be highly useful in many applications such as military UWSNs. That is, SU-HaSAFSS minimizes any risk that may stem from the requirement of replenishing cryptographic keys and re-initializing the entire system (e.g., costly and sometimes impossible re-deployment/re-programming, long-term network disconnections).

Another related property is the flexible data delivery schedule, which is only offered by SU-HaSAFSS among our schemes. This property allows a signer to decide its own data delivery schedule herself after the deployment, and therefore SU-HaSAFSS can address applications in which a pre-determined data delivery schedule cannot be decided. Note that FssAgg schemes directly achieve this property, since they do not rely on the time factor.

The above properties depend on the ability that signers can compute their own key sets after the deployment. Therefore, they are also related to the storage overhead introduced by the compared schemes, which was discussed in the previous section.

Forward-secure Confidentiality: SU-HaSAFSS can integrate forward-secure encryption and forward-secure integrity in a seamless way, since it relies on symmetric cryptography to achieve these goals. Note that to achieve the same property, FssAgg schemes have to resort to costly PKC-based forward-secure encryption schemes (e.g., [32]), which will make these schemes even more expensive.

Immediate Verification: The main drawback of HaSAFSS schemes is that they cannot achieve immediate verification. A more detailed discussion about this issue was given in Section 3.4.1. FssAgg schemes achieve immediate verification, since they do not rely on the time factor.

Overall, being equally storage efficient to FssAgg schemes but much more computationally efficient than them, and at the same same time being more sustainable and flexible than Sym-HaSAFSS and ECC-HaSAFSS, SU-HaSAFSS is an ideal choice for large scale UWSN applications with mildly resource-constrained signers. In contrast, Sym-HaSAFSS is an ideal alternative for highly computationally resource-constrained applications with mildly storage-constrained signers.

3.6 Conclusion

In this chapter, we proposed a new class of cryptographic schemes, Hash-Based Sequential Aggregate and Forward Secure Signature (HaSAFSS), which is suitable for UWSN applications. HaSAFSS schemes achieve the most desirable properties of both symmetric and PKC-based forward-secure and aggregate signature schemes at the same time. They achieve this by using already existing verification delays in the envisioned UWSN applications via three realistic data/time trapdoor delivery models.

We developed three HaSAFSS schemes, Sym-HaSAFSS, ECC-HaSAFSS and SU-HaSAFSS. All these schemes achieve high computational efficiency, public verifiability, signature aggregation and

forward-secure integrity simultaneously. They are significantly more efficient than all of their PKC-based counterparts and still remain publicly verifiable in contrast to other symmetric schemes. Sym-HaSAFSS and ECC-HaSAFSS complement each other by being a signer storage friendly and a verifier storage friendly scheme, respectively.

Preserving all other desirable properties of HaSAFSS schemes, SU-HaSAFSS is much more computationally efficient than all of the previous PKC-based counterparts, and additionally achieves unique properties such as unlimited number of time periods, forward-secure confidentiality, and flexible data delivery schedule.

While HaSAFSS schemes are ideal solutions for UWSNs, they cannot address real-time applications due to their reliance on a time factor. Furthermore, HaSAFSS schemes need a broadcast environment and a passive TTP support, which might not be available in some applications. In following chapters, we present our schemes BAF/FI-BAF [120, 123] and LogFAS [124] that address these limitations.

Chapter 4

Efficient, Compact and Compromise Resilient Cryptographic Constructions for Resource-Constrained Devices

Compromise resilient, compact and publicly verifiable logging in resource-constrained devices is a challenging task, since such devices cannot tolerate any ExpOp or heavy storage overhead. All previous publicly verifiable cryptographic secure logging schemes (e.g., FssAgg [75, 79]) require ExpOps at the signer side, and also produce large cryptographic tags (the details of these previous schemes were given in Chapter 2).

In previous chapter, we presented HaSAFSS schemes as an ideal secure logging solution for UWSNs. However, HaSAFSS schemes cannot address some important applications:

- There are applications that require real-time secure audit logging with the immediate verification property. Despite being computational efficient, HaSAFSS schemes rely on a time factor to introduce an asymmetry, and therefore they cannot achieve the immediate verification.
- HaSAFSS schemes require a passive TTP support and assume a broadcast environment to operate. However, the nature of some applications might not allow even a passive TTP support. Also, the availability of a broadcast environment cannot be guaranteed for all types of applications.

Therefore, a publicly verifiable secure logging scheme that achieves the immediate verification without requiring an online TTP support and/or a time factor is needed. Moreover, such a scheme should also achieve the signer efficiency to be practical for resource-constrained devices.

To fulfill this need, we propose a new class of forward-secure and aggregate signature schemes called *Blind-Aggregate-Forward (BAF)* and its extension *Fast-Immutable BAF (FI-BAF)*. We summarize the properties of BAF and FI-BAF as follows:

1. *Public Verification*: Unlike the symmetric schemes (e.g., [18, 76, 103, 104]), BAF can produce publicly verifiable signatures, and therefore it can protect applications requiring public auditing (e.g., e-voting, financial books) [57, 79]. Different from HaSAFSS schemes [121, 122], BAF does not rely on a time factor to be publicly verifiable, and therefore it can achieve the immediate verification.

2. *Independence of Online Trusted Server*: BAF does not require an online trusted server support to enable the log verification. Therefore, it is more reliable than the previous schemes that require such a support (e.g., [18, 103, 104, 121, 122]).

3. *Optimal Logger (signer) Efficiency:* BAF is the only scheme that achieves the signer computational, storage and communication efficiency at the same time:

   - *ExpOp-free signing and key update:* In BAF, the computational cost of logging a single data item is only a few cryptographic hash operations including the key update cost. This is as efficient as existing symmetric schemes (e.g., [18, 76]) and is much more efficient than all existing PKC-based schemes.
   - *Constant Key/Signature Sizes:* In BAF, independent from the number of time periods and data items to be signed, a logger only needs to store a single key pair, and also needs to store/transmit a single and compact aggregate signature as the authentication tag. Hence, it is more storage/bandwidth efficient than some previous schemes that require linear key and signature storage/transmission on the logger (e.g., [17, 57, 103, 104]).

4. *Computationally Efficient Log Verification*: In BAF, the computational cost of verifying a single log entry is only a single exponentiation operation, which is more efficient than previous PKC-based schemes with the exceptions of our proposed schemes HaSAFSS and LogFAS.

5. *Provable Security*: Unlike some previous schemes [18, 57, 103, 104], BAF is also secure against both the truncation and delayed detection attacks. Moreover, instead of relying on heuristic security arguments against the truncation attacks as in previous schemes, we formally prove that BAF is secure against the truncation attacks in Random Oracle Model (ROM) [13].

6. *Fast and Immutable Logging*: Our extended scheme FI-BAF addresses the need of a BAF variant that allows the verification of a particular log entry without compromising the security of BAF as well as preserving its computational efficiency.

Table 4.1 outlines the above properties and compares the proposed schemes with their counterparts. A detailed performance analysis and comparison can be found in Section 4.5.

The remainder of this chapter is organized as follows. Section 4.1 gives the preliminary definitions. Section 4.2 provides the BAF syntax and security model. Section 4.3 describes the proposed schemes in detail. Section 4.4 gives the security analysis of BAF. Section 4.5 presents performance analysis of the proposed schemes and compares them with previous approaches. Section 4.6 concludes this chapter.

Table 4.1: Comparison of BAF schemes with their counterparts in terms of asymptotic computational/storage/communication overheads and some qualitative properties

Criteria		*PKC-based*			*Sym.*
Computational		BAF/FI-BAF	FssAgg/iFssAgg BLS / BM / AR	Logcrypt	
Sig		$O(L)H$	$O(L)(ExpOp+H)$	$O(L)(ExpOp+H)$	$O(L)H$
Upd		H	$ExpOp+H$	-	H
Ver		$O(L)(ExpOp+H)$			$O(L)H$
Kg		$O(L)(ExpOp+H)$			$O(L)H$
Communication		$O(1)\|\sigma\|$	$O(1)\|\sigma\|$	$O(L)\|\sigma\|$	$O(L)\|H\|$
Storage	*Signer*	$O(1)(\|K\|+\|\sigma\|)$	$O(1)(\|K\|+\|\sigma\|)$	$O(L)(\|K\|+\|\sigma\|)$	$O(L)(\|H\|)$
	Verifier	$O(L\cdot S)\|K\|$	$O(L\cdot S)\|K\|$ / $O(S)\|K\|$	$O(L\cdot S)\|K\|$	$O(S)\|K\|$
Public Ver.		Y	Y		N
Online TTP		Y	Y		N
Immediate Ver.		Y	Y		N
Delayed Detection A.		Y	Y		N
Truncation A.		Y	Y	N	N
Security Argument		Provable	Heuristic	N	N

*Table 4.1 demonstrates the asymptotic costs of processing L data items for each compared scheme. H and S denote the cost of single hash operation and number signers in the system, respectively. $|H|$, $|K|$ and $|\sigma|$ denote the bit length of hash output, the bit length of private/public key and the bit length of signature, respectively ($|K|$ and $|\sigma|$ slightly vary for each scheme). Storage and communication costs are based on the cryptographic overhead introduced by the schemes (data overheads are the same for all compared schemes).

† BAF schemes are the only alternative that achieve the signer efficiency (i.e., ExpOp-free signing and constant storage/communication overhead), while retaining the verifier computational efficiency. At the same time, they possess all the desirable properties of PKC schemes when compared with the symmetric schemes.

4.1 Preliminaries

Notation. We use $||$, $|a|$ and $\{0,1\}^*$ to denote the concatenation operation, bit length of variable a, and the set of binary strings of any finite length, respectively. $a \xleftarrow{\$} \mathcal{S}$ denotes that the value of variable a is randomly and uniformly selected from set $\mathcal{S}$. For any integer l, $(a_0,\ldots,a_l) \xleftarrow{\$} \mathcal{S}$ means $(a_0 \xleftarrow{\$} \mathcal{S},\ldots,a_l \xleftarrow{\$} \mathcal{S})$. $\mathcal{A}^{\mathcal{O}_0,\ldots,\mathcal{O}_i}(.)$ denotes algorithm $\mathcal{A}$ is provided with oracles $\mathcal{O}_0,\ldots,\mathcal{O}_i$. For example, $\mathcal{A}^{Sch.Sig_{sk}}(.)$ denotes algorithm $\mathcal{A}$ is provided with a *signing oracle* of Sig of signature scheme Sch under private key sk.

BAF schemes rely on the intractability of *Discrete Logarithm Problem (DLP)* [16], which is defined below.

Definition 4.1 *Given a cyclic group G of prime order q and a generator α of G, let $\mathcal{A}$ be an algorithm that returns an element of Z_q^*. Consider the following experiment:*

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

$y \overset{\$}{\leftarrow} \mathbb{Z}^*_q$,

$Y \leftarrow \alpha^y$,

$y' \leftarrow \mathcal{A}(Y)$,

If $\alpha^{y'} = Y$, *return* 1, *else, return* 0.

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

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

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 maximum is over all $\mathcal{A}$ *having time complexity* t.

4.2 Models

In this section, we first briefly describe our system model that is based on the Forward-secure Stream Integrity (FSI) model. We then provide the generic model of *F*orward-secure and *A*ggregate *S*ignature *(FAS)* schemes, which is suitable for our system model. Last, we introduce our security model, in which a FAS scheme is proven to be *Forward-secure Aggregate Existential Unforgeable against Chosen Message Attack (FAEU-CMA)* and secure against the *truncation attack*.

The actual BAF schemes and their security analysis are presented in Section 4.3 and Section 4.4, respectively.

4.2.1 System Model

Before presenting our system model, we first discuss the *Forward-secure Stream Integrity (FSI)* model, which is the basis of all existing FAS constructions.

FSI Model

FSI model is the classic tamper-evident audit logging model initially introduced by Bellare and Yee [17] in the context of symmetric key cryptography, and they later formalized it in [18]. The basic FSI model includes two entities: (i) Storage-limited loggers who are honest until they are compromised. These loggers compute an authentication tag (e.g., a MAC) for each log entry in a forward-secure way and then upload these logs and MACs to the verifiers when they are available. (ii) A limited number of verifiers

who are fully trusted (e.g., they do not disclose the keying material) but not always readily available for the loggers. The basic FSI model assumes a full symmetric key distribution via an authenticated channel.

Schneier and Kelsey [104] follows a similar model in the presence of a TTP(s). That is, a TTP provides the required symmetric keying material to the verifiers accordingly (based on a request or periodically).

Logcrypt extends the basic FSI model into the PKC domain. Later, Ma and Tsudik [76] follows this PKC-based FSI model by adding the sequential signature aggregation. This provides "all-or-nothing" property and compactness. These models assume that verifiers are more resourceful than signers.

Our System Model

Our system model is based on PKC-based FSI model. There are two new entities in the system: (i) Storage/computational/bandwidth limited loggers who are honest until they are compromised. (ii) Storage resourceful verifiers who can be *any (untrusted) entity* and do *not* need an *online* TTP support for the verification.

We assume that the key generation/distribution is performed offline before deployment as in all FSI models. According to the application requirement, each signer can generate its own private/public keys and provide them to the verifiers (via a certification procedure), or optionally, a Key Generation Center (KGC) generates these keys *offline* before the deployment and then distributes them to the system entities (e.g., suitable for WSNs and RFID tags). If the latter approach is preferred, the KGC is assumed to be trusted and it cannot be compromised by the adversary. For each signer, there is a different private/public key set, and therefore the key generation algorithm is implemented for each signer in the system once.

Our constructions behave according to the *same-signer-distinct-message model* similar to the existing PKC-based FAS constructions (e.g., [75, 76, 78, 79]). In this model, the same logger computes aggregate signatures of distinct audit logs accumulated-so-far (i.e., similar to the condensed signatures notion in [91]). This model is an ideal option for secure audit logging applications (e.g., [75, 78, 79, 103, 104, 116]), since each logger is responsible for only her own audit logs.

4.2.2 Model of Forward-Secure and Aggregate Signature (FAS) Schemes

A FAS scheme is an integrated signature scheme that achieves both the forward-security and the sequential signature aggregation properties simultaneously. Hence, it has a *Key Update* algorithm that follows the "evolve-and-delete strategy" to achieve the forward security similar to the forward-secure signatures (e.g., [68]). Moreover, it has *Key Generation*, *Forward-secure and Aggregate Signature Generation* and *Forward-secure and Aggregate Signature Verification* algorithms. The signature generation algorithm performs the signature aggregation as in the aggregate signatures (e.g., [21, 26]) and then uses the key update algorithm to update the private key.

Definition 4.2 *A FAS scheme is a tuple of four algorithms* (Kg, Upd, Sig, Ver) *that behave as follows:*

1. $(sk, PK) \leftarrow FAS.Kg(1^\kappa, L)$*: The key generation algorithm takes the security parameter* 1^κ *and the maximum number of key updates* L *as the input. It returns a private/public key pair* (sk, PK) *as the output.*

2. $sk_{j+1} \leftarrow FAS.Upd(sk_j, L)$*: The key update algorithm takes the private key* sk_j, $0 \leq j < L-1$, *and* L *as the input. It returns the private key* sk_{j+1} *as the output.*

3. $\sigma_{0,l} \leftarrow FAS.Sig(sk_j, \overrightarrow{D})$*: The forward-secure and aggregate signing algorithm takes the private key* sk_j*, a message* $\overrightarrow{D} = (D_j, \ldots, D_l)$, $l \geq j$*, to be signed and an internal state* $\Psi = (\sigma_{0,j-1}, \langle D_0, \ldots, D_{j-1}\rangle)$ *as the input, where* Ψ *is an empty vector initially. It returns a forward-secure and aggregate signature* $\sigma_{0,l}$ *as the output, and then updates the internal state and the private key as* $\Psi \leftarrow (\sigma_{0,l}, \langle D_0, \ldots, D_l\rangle)$ *and* $sk_{m+1} \leftarrow FAS.Upd(sk_m, L), m = j, \ldots, l$*, respectively.*

4. $b \leftarrow FAS.Ver(PK, \overrightarrow{D}, \sigma_{0,l})$*: The forward-secure and aggregate verification algorithm takes* PK*, a message* $\overrightarrow{D} = (D_0, \ldots, D_l)$, $l \leq L$*, and* $\sigma_{0,l}$ *as the input. It returns a bit* b*, with* $b = 1$ *meaning* valid*, and* $b = 0$ *meaning* invalid*.*

In BAF, the private key sk is provided to the signer as an initial key, and it is evolved via the key update algorithm in the logging process. Therefore, the private key size is constant at the signer side. PK is a vector with $4L$ components (i.e., individual public keys), which are stored by the verifiers.

4.2.3 Threat and Security Model

Our treat model reflects how a generic FAS scheme works in our envisioned system model. That is, in a real FAS implementation, $\mathcal{A}$ can obtain a large number of forward-secure and aggregate signatures $\sigma_0, \ldots, \sigma_i$ of distinct audit log files $\overrightarrow{D}_0, \ldots, \overrightarrow{D}_i$ computed under a PK. Each vector $\overrightarrow{D}_k = \{D_{j'}, \ldots, D_j\}$, $j \geq j'$ for $k = 0, \ldots, i$ represents a separate log file that includes a set of individual logs. $\mathcal{A}$ can observe these values even before the compromise (e.g., a user can read system logs or logs/signatures are transmitted to the verifiers via an insecure channel). Once $\mathcal{A}$ compromises the signer, she also obtains private key(s) that have not been erased from the memory in the duration of logging. $\mathcal{A}$ may attempt to modify, re-order and selectively delete any of previously signed audit logs.

A FAS scheme is proven to be *ForWard-secure Aggregate Existentially Unforgeable against Chosen Message Attack (FAEU-CMA)* based on the experiment defined in Definition 4.3. Moreover, we provide a formal treatment for the truncation attacks via the *truncation experiment (TRUNC)* defined in Definition 4.4 based on the *signature extraction* argument [26, 42]. In both experiments, $\mathcal{A}$ is provided with three oracles that behave as follows:

(i) *Random Oracle*: $\mathcal{A}$ is given to access a random oracle $RO(.)$ that she can request hash of any message D of her choice up to L' messages.

(ii) *Signing Oracle*: $\mathcal{A}$ is provided with a *signing oracle* $FAS.Sig_{sk}(.)$. For each batch query i, $\mathcal{A}$ can query $FAS.Sig_{sk}(.)$ oracle on a set of message $\overrightarrow{D}_i = \{D_{j'}, \ldots, D_j\}, j \geq j'$, of her choice. $FAS.Sig_{sk}(.)$ returns a forward-secure and aggregate signature $\sigma_{0,i}$ under sk on $(\langle \overrightarrow{D}_0, \ldots, \overrightarrow{D}_{i-1} \rangle, \overrightarrow{D}_i)$ (i.e., $\sigma_{0,i}$ is on *all* previous messages that $\mathcal{A}$ queried up to now). $\mathcal{A}$ can query $FAS.Sig_{sk}(.)$ up to L individual messages in total (i.e., i batch queries in total) as described, until she decides to "break-in".

(iii) $Break\text{-}in$ *oracle*: $\mathcal{A}$ is then provided with a $Break\text{-}in$ oracle, which returns the current private key to $\mathcal{A}$. That is, if $\mathcal{A}$ queried $l \leq L$ individual messages to $FAS.Sig_{sk}(.)$, then $Break\text{-}in$ oracle returns $(l+1)$-th private key to $\mathcal{A}$ (if $l = L$ then $Break\text{-}in$ oracle rejects the query, since all private keys were used).

Definition 4.3 $FAEU\text{-}CMA$ experiment *is defined as follows:*
Experiment $Expt_{FAS}^{FAEU\text{-}CMA}(\mathcal{A})$

$(sk, PK) \leftarrow FAS.Kg(1^\kappa, L)$,
$(\overrightarrow{D}^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.), FAS.Sig_{sk}(.), Break\text{-}in}(PK)$,
If $FAS.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1$ *and* $\exists n \in \{0, \ldots, l\} : \overrightarrow{D}^*[n] \notin \overrightarrow{D}$ *holds, then return* 1*, else return* 0*. Here,* $\overrightarrow{D} = \{\overrightarrow{D}_0 || \ldots || \overrightarrow{D}_i\}$ *denotes* i *batch queries (including* $l \leq L$ *individual messages in total) asked to the* $FAS.Sig_{sk}(.)$ *oracle, each* $\overrightarrow{D}_m, 0 \leq m \leq i$*, denotes* m*-th batch query (a vector), and* $\overrightarrow{D}^*[n]$ *denotes* n*-th individual data item in the forgery data item vector* $\overrightarrow{D}^*$*.*

FAEU-CMA-advantage of $\mathcal{A}$ *is defined as*

$$Adv_{FAS}^{FAEU\text{-}CMA}(\mathcal{A}) = Pr[Expt_{FAS}^{FAEU\text{-}CMA}(\mathcal{A}) = 1]$$

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

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

where the maximum is over all $\mathcal{A}$ *having time complexity* t*, making at most* L' *queries to* $RO(.)$*, at most* L *queries to* $FAS.Sig_{sk}(.)$*, and the sum of lengths of these queries being at most* μ *and* μ'*, respectively.*

Definition 4.4 $TRUNC$ experiment *is defined as follows:*
Experiment $Expt_{FAS}^{TRUNC}(\mathcal{A})$

$(sk, PK) \leftarrow FAS.Kg(1^\kappa, L)$,
$(\overrightarrow{D}^*, \sigma^*) \leftarrow \mathcal{A}^{RO(.), FAS.Sig_{sk}(.), Break\text{-}in}(PK)$,

If $(FAS.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1) \wedge (\overrightarrow{D}^* \subset \overrightarrow{D}) \wedge (\forall I \subseteq \{0, \ldots, i\}, \overrightarrow{D}^* \neq ||_{m \in I} \overrightarrow{D}_m)$ *holds, then return* 1*, else, return* 0*, where* $\overrightarrow{D}_m$ *denotes* m*-th batch query (a vector) in* $\overrightarrow{D} = \{\overrightarrow{D}_0 || \ldots || \overrightarrow{D}_i\}$.

$TRUNC$-advantage of $\mathcal{A}$ *is defined as*

$$Adv_{FAS}^{TRUNC}(\mathcal{A}) = Pr[Expt_{FAS}^{TRUNC}(\mathcal{A}) = 1]$$

$TRUNC$-advantage of FAS *is defined as*

$$Adv_{FAS}^{TRUNC}(t, L', L, \mu', \mu) = \max_{\mathcal{A}}\{Adv_{FAS}^{TRUNC}(\mathcal{A})\}$$

where the maximum is over all $\mathcal{A}$ *having time complexity* t*, making at most* L' *queries to* $RO(.)$*, at most* L *queries to* $FAS.Sig_{sk}(.)$*, and the sum of lengths of these queries being at most* μ *and* μ'*, respectively.*

4.2.4 Discussion on Our Security Model

To justify our security model, we give a discussion on the batch queries, modeling of truncation attacks and some approaches that are alternative to ours.

• *Batch queries vs. individual queries*: The previous FAS constructions (i.e., [75,76,78,79,120]) implement the signing oracle based on individual signature queries. Such an implementation still captures a forgery on an individual data modification. However, we prefer batch queries for two reasons:

(i) Batch queries reflect the FAS mechanism better than individual queries. In all FAS construction, the aggregation function is public and easily invertible on a given aggregate signature $\sigma_{0,j}$, if its individual components are known. For example, for given $\sigma_{0,j}$ and $\sigma_{0,j'},\ j' \leq j$, it is easy to compute $\sigma_{j'+1,j}$. Hence, for a given set of messages, if "all-or-nothing" property is needed, a FAS scheme is required to delete all intermediate aggregate signatures (e.g., individual signatures) during the signing process, and only keep the final aggregate signature as the authentication tag [75, 76, 79]. Note that our batch query approach reflects this behavior while the individual query approach cannot capture it.

(ii) None of the previous FAS constructions provide a formal reduction in the case of a tail-truncation attack. Recall that the tail-truncation attack is a special type of deletion attack, in which $\mathcal{A}$ deletes a continuous subset of entries at the end of the log. Since this type of deletion does not cause an order change, plain individual query model is not sufficient to capture this case. In contrast, our batch query approach captures this attack based on the aggregate signature extraction argument.

• *Aggregate Signature Extraction and Truncation Attack*: The truncation attack can be modeled based on the *aggregate signature extraction* argument [26, 42]). The difficulty of aggregate signature extraction implies that for a given aggregate signature $\sigma_{0,k}$ computed from k individual signatures, it is difficult to extract these individual signatures $\sigma_0, \ldots, \sigma_k$ (provided that only $\sigma_{0,k}$ is known to the extractor). In fact, it should be difficult to separate any proper aggregate signature subset σ' from the given aggregate signature of $\sigma_{0,k}$.

Note that a truncation attack implies a signature extraction [76, 79]. For instance, the extraction of an individual signature σ_k from the given aggregate signature $\sigma_{0,k}$ without knowing its complementary aggregate signature $\sigma_{0,k-1}$ is equivalent to a tail-truncation attack (i.e., $\mathcal{A}$ can trivially truncate the corresponding data item D_k (i.e., the tail log entry) of σ_k without being detected, since the aggregation function is public and invertible).

In our security analysis, we prove the resilience of BAF against the truncation attacks based on the difficult of the aggregate signature extraction. In particular, we make a reduction to *DLP* for a single signature extraction case. That is, for a given random valid aggregate signature σ' on two individual public keys, if $\mathcal{A}$ can split it into two valid individual signatures on their corresponding public keys, then it is possible to break the *DLP*.

• *Alternative Approaches*: An alternative approach to avoid a truncation attack is to use auxiliary signatures on aggregate signatures or indexes. For instance, in addition to the aggregate signature, one can compute a forward-secure signature on a counter that is increased by once for each accumulated log entry. This prevents $\mathcal{A}$ to modify the number of data items in the log. However, this approach increases the computational and storage costs of FAS construction due to the use of a secondary forward-secure signature.

4.3 Blind-Aggregate-Forward (BAF) Schemes

In this section, we first present our main BAF scheme, and then its extension FI-BAF.

4.3.1 Overview

All previous PKC-based FAS constructions are directly derived from existing aggregate or forward-secure signature schemes. For instance, FssAgg-BLS [76] is derived from the aggregate signature scheme given in [26]. Similarly, FssAgg-BM and FssAgg-AR in [75, 78, 79] are derived from the forward secure signatures given in [12] and [2], respectively. Hence, they inherit the high computational/storage costs of these signature primitives as well as incurring extra overheads to achieve the additional aggregation or forward security property.

One possible way to avoid ExpOps for logging is to introduce an asymmetry between the signer and the verifiers via the time factor (e.g., TESLA [96]). However, such a scheme cannot achieve the immediate verification at the verifier side. Moreover, it requires online TTP support to achieve the forward security. (If the signer herself introduces the required asymmetry, then an active attacker compromising the signer can eventually forge the computed signatures [121]). To achieve the immediate verification and scalability, BAF uses neither the time factor nor an online TTP support.

• *BAF Strategy:* BAF uses a new strategy called *"Blind-Aggregate-Forward"*. Such a strategy enables signers to log a large number of log entries with little computational, storage, and communication

costs in a publicly verifiable way:

1. *Individual Signature Generation:* BAF computes the individual signature of each accumulated data item with a simple and efficient blinding operation. Blinding is applied to the hash of a data item via first a multiplication and then an addition operation modular a large prime q by using a pair of secret blinding keys (referred as the blinding key pair). The result of this blinding operation is a unique and random looking output (i.e., the one-time individual signature), which cannot be forged without knowing its associated private keys.

2. *Key Update:* BAF updates the blinding key pair via two hash operations after each individual signature generation, and then deletes the previous key pair from memory.

3. *Signature Aggregation:* BAF aggregates the individual signature of each accumulated data item into the existing aggregate signature with a single addition operation modular q.

In the above construction, the individual signature computation binds the hash of a signed data item to its index, a random number and the corresponding blinding key pair in a specific algebraic form. The signature aggregation maintains this form incrementally and also preserves the indistinguishability of each individual signature. Hence, the resulting aggregate signature can be verified by a set of public key securely. BAF enables this verification by embedding each blinding private key pair into a public key pair via an modular exponentiation in the key generation phase in an offline manner. Using the corresponding public keys, the verifiers follow the BAF signature verification equation by performing an modular exponentiation for each received data item.

4.3.2 Description of BAF

The proposed BAF scheme is given below.

1. $\underline{(sk, PK) \leftarrow BAF.Kg(1^{\kappa}, L)}$: Given the security parameter 1^{κ}, generate large primes (p, q) such that $p > q$ and $q|(p-1)$, where $|p|$ and $|q|$ are selected to achieve κ bit security. Also generate a generator α of the subgroup G of order q in Z_p^*. H is a cryptographic hash function [14], which is defined as $H : \{0,1\}^* \rightarrow Z_q^*$.

   (a) Generate the initial key pair as $(a_0, b_0) \stackrel{\$}{\leftarrow} Z_q^*$, and then generate two hash chains from (a_0, b_0) as $a_{j+1} \leftarrow H(a_j)$ and $b_{j+1} \leftarrow H(b_j)$ for $j = 0, \ldots, L-1$ [1].

   (b) Generate two master seeds $(x, x') \stackrel{\$}{\leftarrow} Z_q^*$. Also generate $r_j \leftarrow H(x||j)$ and $k_j \leftarrow H(x'||j)$, respectively, for $j = 0, \ldots, L-1$. Compute the corresponding tokens as $u_j \leftarrow k_j + r_j \bmod q$ for $j \leftarrow 0, \ldots, L-1$ and $u'_j \leftarrow k_{j-1} + H(k_j) \bmod q$ for $j = 1, \ldots, L-1$.

[1] Key evolve function can also be a Forward-secure Pseudo Random Number Generator (FWPRNG) as suggested in [68]. However, we prefer an ideal hash function here to provide a more succinct proof in our security analysis.

(c) Compute $\{A_j \leftarrow \alpha^{a_j} \bmod p,\ B_j \leftarrow \alpha^{b_j} \bmod p\}_{j=0}^{L-1}$.

(d) Private/public key of ID_i is as follows:
$sk \leftarrow (\langle a_0, b_0\rangle, x, x')$ and $PK \leftarrow (\{A_j, B_j, u_j\}_{j=0}^{L-1}, \{u'_j\}_{j=1}^{L-1}, p, q, \alpha, L)$.

2. $\underline{sk_{j+1} \leftarrow BAF.Upd(sk_j, L)}$: Given $sk_j = (\langle a_j, b_j\rangle, x, x')$, if $j \geq L-1$ then return $\perp$ (i.e., invalid input). Otherwise, update the private key as $sk_{j+1} \leftarrow (\langle a_{j+1}, b_{j+1}\rangle, x, x')$, where $(a_{j+1} \leftarrow H(a_j),\ b_{j+1} \leftarrow H(b_j))$, and then securely erase (a_j, b_j) from the memory[2].

3. $\underline{\sigma_{0,l} \leftarrow BAF.Sig(sk_j, \overrightarrow{D})}$: Given the private key $sk_j = (\langle a_j, b_j\rangle, x, x')$, data to be signed $\overrightarrow{D} = \langle D_j, \ldots, D_l\rangle$ and the internal state $\Psi = (\sigma_{0,j-1}, \langle D_0, \ldots, D_{j-1}\rangle)$ (initially Ψ is an empty vector), compute the forward-secure and aggregate signature $\sigma_{0,l}$ as follows:

(a) Compute $s_{j,l} \leftarrow \sum_{m=j}^{l}(a_m H(D_m||r_m||m) + b_m) \bmod q$, where $r_m = H(x||m)$ and $(\langle a_{m+1}, b_{m+1}\rangle, x, x') \leftarrow BAF.Upd((\langle a_m, b_m\rangle, x, x'), L)$ for $m = j, \ldots, l$.

(b) Fold $s_{j,l}$ into $s_{0,j-1}$ as $s_{0,l} \leftarrow s_{0,j-1} + s_{j,l} \bmod q$, where $l > 0$ and $s_{0,0} = s_0$.

(c) Erase $(\sigma_{0,j-1}, s_{j,l}, r_j, \ldots, r_l)$ from memory. Update the state as $\Psi \leftarrow (\sigma_{0,l}, \langle D_0, \ldots, D_l\rangle)$ and return the signature $\sigma_{0,l} \leftarrow \langle s_{0,l}, k_l\rangle$, where $k_l = H(x'||l)$.

4. $\underline{b \leftarrow BAF.Ver(PK, \overrightarrow{D}, \sigma_{0,l})}$: Recall that $PK = (\{A_j, B_j, u_j\}_{j=0}^{L-1}, \{u'_j\}_{j=1}^{L-1}, p, q, \alpha, L)$. Given $\overrightarrow{D} = (D_0, \ldots, D_l)$, if the below equality holds, $BAF.Ver$ returns 1, else, returns 0.

$$\alpha^{s_{0,l}} \bmod p \equiv \prod_{j=0}^{l}(A_j^{H(D_j||r_j||j)} \cdot B_j) \bmod p,$$

where $k_{j-1} \leftarrow u'_j - H(k_j) \bmod q$ for $j = l, \ldots, 1$ and $r_j \leftarrow u_j - k_j \bmod q$ for $j = l, \ldots, 0$.

Correctness: Recall that the signature is $\sigma_{0,l} = \langle s_{0,l}, k_l\rangle$, where $s_{0,l} \equiv \sum_{j=0}^{l}(a_j\ H(D_j||r_j||j) + b_j) \bmod q$ and $k_l = H(x'||l)$, which are computed via the $BAF.Sig$ algorithm. In the BAF verification equation, the verifier computes the left-side of the equation as $\alpha^{s_{0,l}} \bmod p \equiv \alpha^{\sum_{j=0}^{l}(a_j H(D_j||r_j||j)+b_j)} \bmod p$. At the right-side of the equation, the verifier checks whether the data items $D_0, \ldots, D_l$ (along with random numbers $r_0, \ldots, r_l$ and indexes), when exponentiated over the public keys $\{A_j \equiv \alpha^{a_j} \bmod p, B_j \equiv \alpha^{b_j} \bmod p\}_{j=0}^{l}$, correctly constructs $s_{0,l}$ on the exponent. The correctness of the BAF follows

[2]Note that, in contrast to the private keys (a_j, b_j), the master seeds (x, x') do not need to be forward-secure, and therefore they are not evolved (they are given as the input to $BAF.Upd$ for the completeness of the interface).

as:

$$\begin{aligned}
\alpha^{s_{0,l}} \bmod p &\equiv \alpha^{\sum_{j=0}^{l}(a_j H(D_j||r_j||j)+b_j)} \bmod p \\
&\equiv ((\alpha^{a_0})^{H(D_0||r_0||0)}\alpha^{b_0})((\alpha^{a_1})^{H(D_1||r_1||1)}\alpha^{b_1}) \cdots ((\alpha^{a_l})^{H(D_l||r_l||l)}\alpha^{b_l}) \bmod p \\
&\equiv \prod_{j=0}^{l}(A_j^{H(D_j||r_j||j)} \cdot B_j) \bmod p
\end{aligned}$$

If the above equality holds; this guarantees that the data items are intact and only the claimed signer, who possessed the correct private key pairs before their deletion, could compute such a signature (which is unforgeable after the keys were deleted as proven in Section 4.4).

Remark: Different from our preliminary version [120], the current BAF algorithm uses a random number r_j for each signature s_j computed on D_j. We introduce these random numbers to achieve a correct behavior for the simulators constructed in $FAEU\text{-}CMA$ and $TRUNC$ experiments. That is, they enable the simulator $\mathcal{F}$ (i.e., the DLP attacker) to simulate $\mathcal{A}$'s (i.e., the BAF attacker's) $RO(.)$ and $FAS.Sig_{sk}(.)$ queries without causing $\mathcal{A}$ to abort with a non-negligible probability (in terms of κ). The details are given in Section 4.4.

We use master seeds (x, x'), tokens (u, u') and masking keys k as the auxiliary components to integrate these random numbers to the BAF without degrading its optimal signer efficiency. In the key generation, each random number r_j and the previous key k_{j-1} are masked via the key k_j as $u_j \leftarrow k_j + r_j \bmod q$ for $j = 0, \ldots, L-1$ and $u'_j \leftarrow k_{j-1} + H(k_j) \bmod q$ for $j = 1, \ldots, L-1$, respectively. These tokens are given to the verifiers as a part of the public key.

Once the signer releases the signature $\sigma_{0,l} = \langle s_{0,l}, k_l \rangle$, the verifiers can recover all the previous random numbers $r_0, \ldots, r_{l-1}$ from (u_l, u'_l) via k_l in a computationally efficient way. Similarly, the signer can derive all random numbers and masking keys from the master seeds (x, x') with only two hash operations (these seeds and random numbers do not need to be forward-secure).

Notice that, instead of using the above strategies, the signer could simply generate a new random number for each data item. However, such an approach requires storing and then transmitting these random numbers to the verifiers, which destroy the constant-size key/signature storage and transmission properties (i.e., the aggregation property) at the signer side. Based on the above strategies, the signer avoids storing/transmitting a random number for each data item (i.e., linear storage/communication overhead), and she also does not perform any ExpOp as required. The verifier computational efficiency is also preserved, but the verifier storage overhead is doubled.

4.3.3 Fast-Immutable BAF (FI-BAF)

All existing FAS constructions including the BAF keep only the single-final aggregate signature for the entire signing process. There are two reasons behind this strategy: (i) The aggregation function of all PKC-based FAS constructions is public and easy to invert if its individual signature compo-

nents are known. Therefore, all individual signatures are securely erased immediately after they are aggregated in the signing process to prevent the truncation attacks. (ii) This also offers the signature storage/transmission efficiency (i.e., the constant signature size).

However, this strategy also has certain drawbacks [79]: (i) The verification of a particular log entry requires the verification of all log entries, which forces verifiers to perform a large number of ExpOps. (ii) If the verification of the aggregate signature fails, it is not possible to detect which log entry(ies) is (are) responsible for the failure.

It is therefore desirable to enable a fine-grained verification of individual data items, while still being secure against the truncation attack.

The problem of deriving "valid" aggregate signatures from existing aggregate and/or individual signatures (either via truncation or partial aggregation) was first addressed by Mykletun et al. [91] with *immutable aggregate signatures*. Immutable aggregate signatures prevent such derivations by introducing additional protection mechanisms for individual signatures according to the underlying aggregate signature scheme. Mykletun et al. [91] suggest two main types of immutable signature mechanisms: (i) *Zero-knowledge proof* techniques (one is interactive and the other is non-interactive) for condensed-RSA schemes; (ii) *Umbrella signature* technique for the BLS-based aggregate signature schemes. These constructions are designed for the generic *distinct-signer-distinct-message model*.

To prevent the truncation attacks against FssAgg signatures [75, 76], when individual signatures are kept, [79] adopts *umbrella signature* technique in [91] to their schemes as the immutable signature mechanism. Unfortunately, the direct adaptation of this technique, which is particularly useful for distinct-signer-distinct-messages model, increases the computational overhead of already costly FssAgg schemes.

Fast-Immutable BAF (FI-BAF): To address the above problem, we give a simple variant of the BAF called Fast-Immutable BAF (FI-BAF). FI-BAF leverages the fact that all existing FAS constructions behave according the *same-signer-distinct-message model* (see Section 4.2.1), and therefore the signer can easily compute two independent signature sets to achieve both the "all-or-nothing" and the individual signature verification properties. This simple strategy is more efficient than the direct use of immutable signatures [91], which are designed for the *different-signer-distinct-message model*.

In FI-BAF, "all-or-nothing" property is achieved by using BAF as a sub-routine. To enable the fine-grained verification, FI-BAF computes a second set of forward-secure signatures along with the execution of BAF. These signatures are computed as in BAF with the exception that they are kept in individual form instead of being aggregated. Furthermore, these individual signatures are bind to a random number n, which is used along with an index incrementally, and is specific to each signer.

Remark that the individual signatures and the aggregate signature are computed with distinct private key sets. Hence, these individual signatures cannot be used to launch a truncation attack (any such attempt is equivalent to produce a forgery on the aggregate signature). Similarly, individual signatures are identified with the use of a distinguishing index n, which prevents them to be used in an aggregated

form. That is, any signature computed with n is considered as an individual signature, and if it is used in an aggregate form, the verifiers will reject the associated signature. Therefore, $\mathcal{A}$ either has to remove n from the individual signature (i.e., explicitly forge it), or keep it intact in individual form.

Note that since the computation of individual signature set does not require any ExpOp as in BAF, FI-BAF is practically as computation-efficient as the original BAF at the signer side. At the same time, different from iFssAgg schemes [79], FI-BAF does not combine individual and aggregate signatures to form the single-final aggregate signature. (Such combination is redundant in the same-signer-distinct-message model.). Hence, FI-BAF is also as computation-efficient as the original BAF at the verifier side.

The storage overhead of FI-BAF is linear with the number of individual signatures as in all immutable aggregate signature schemes. Similarly, FI-BAF doubles the storage overhead of its base scheme at the verifier side. However, FI-BAF is more computation-efficient than iFssAgg schemes, which also doubles the computational overhead of their base schemes both at the signer and verifier sides.

4.4 Security Analysis

We prove that BAF is a *FAEU-CMA* signature scheme in Theorem 4.1 below.

Theorem 4.1

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

where $O(t') = O(t + L\kappa^2 + L'\kappa)$.

Proof: Let $\mathcal{A}$ be a *BAF attacker* and $(y \leftarrow Z_q^*, Y \leftarrow \alpha^y \bmod p)$. We construct a *DL attacker* $\mathcal{F}$ that uses $\mathcal{A}$ as a sub-routine on Y as follows:

Algorithm $F(Y)$

Set the target forgery/signature extraction index $w \xleftarrow{\$} [0, L-1]$,

$(sk, PK) \leftarrow BAF.Kg(1^\kappa, L)$, where $sk = (\langle a_0, b_0\rangle, x, x')$, $PK \leftarrow (\{A_j, B_j\}_{0\leq j\leq L-1, j\neq w}, \{u_j\}_{j=0}^{L-1}, \{u'_j\}_{j=1}^{L-1}, p, q, \alpha, L)$.

$(\langle a_j, b_j\rangle, x, x') \leftarrow BAF.Upd((\langle a_{j-1}, b_{j-1}\rangle, x, x'), L)$ for $j = 1, \ldots, L-1$,

Simulation: Simulate public keys (A_w, B_w), an individual signature γ, and its corresponding random oracle answer z on (A_w, B_w) as follows:

- $A_w \leftarrow Y$,
- $(z, \gamma) \xleftarrow{\$} \mathbb{Z}_q^*$,
- $B_w \leftarrow (Y^z)^{-1}\alpha^\gamma \bmod p$,

Initialize the counters as $l \leftarrow 0,\ j' \leftarrow 0,\ i \leftarrow 0,\ l' \leftarrow 0$,

Execute $\mathcal{A}^{RO(.),FAS.Sig_{sk}(.),\textit{Break-in}}(PK)$: $\mathcal{F}$ maintains three lists $\mathcal{HL}$, $\mathcal{LD}$, and $\mathcal{LS}$ to keep track the query results in the duration of the experiment. $\mathcal{HL}$ is a hash list in a form of tuples (D_j, h_j), where D_j and h_j denote a data item queried to $RO(.)$ and its corresponding $RO(.)$ answer, respectively. $\mathcal{LD}$ is a data list, in which each of its element $\mathcal{LD}[i]$ is also a data vector $\overrightarrow{D}$ (i.e., a batch query). $\mathcal{LS}$ is a signature list that is used to record answers given by $FAS.Sig_{sk}(.)$.

- Queries: $\mathcal{A}$ queries the $FAS.Sig_{sk}(.)$ oracle on up to L messages of her choice, and then queries the $\textit{Break-in}$ oracle once. $\mathcal{A}$ also queries the $RO(.)$ oracle on up to L' messages of her choice. These queries are handled as follows:

  • *How to respond queries to $RO(.)$ oracle*: $\mathcal{F}$ executes the function $H\text{-}Sim(\delta)$ that works as follows: If $\delta \in \mathcal{HL}$ then return the same result from $\mathcal{HL}$. Otherwise, return $h \xleftarrow{\$} \mathbb{Z}_q^*$ as the answer, insert the new tuple (δ, h) to $\mathcal{HL}$, and update $l' \leftarrow l' + 1$.

  • *How to respond i-th $FAS.Sig_{sk}(.)$ query*:

    - For each batch query i, $\mathcal{A}$ queries $FAS.Sig_{sk}(.)$ on $\overrightarrow{D}_i = \{D_{j'}, \ldots, D_j\},\ j \geq j'$ of her choice.
    - If $((j < w) \vee (j' > w))$ then compute $s_{j',j} \leftarrow \sum_{m=j'}^{j}(a_m h_m + b_m) \bmod q$ as in the real system, where $h_m \leftarrow H\text{-}Sim(D_m||r_m||m)$ and $r_m \leftarrow H\text{-}Sim(x||m)$ (if $j' = j$ then $s_{j',j} = s_j$) [3].

      Otherwise, if $(D_w||r_w||w) \in \mathcal{HL}$ then *abort* and return 0 (an abort probability analysis is given in the following parts). Otherwise, compute $s_{j',j} \leftarrow [\sum_{j' \leq m \leq j, m \neq w}(a_m h_m + b_m)] + \gamma \bmod q$, where $h_m \leftarrow H\text{-}Sim(D_m||\ r_m||m)$ (if $j' = j$ then $s_{j',j} = s_j$). Insert the tuple $(D_w||r_w||w, z)$ into $\mathcal{HL}$.
    - $s_{0,j} \leftarrow s_{0,j'-1} + s_{j',j} \bmod q$ (for initial $j' = 0$, $s_{0,j'-1} = 0$), and $\sigma_{0,j} \leftarrow \langle s_{0,j}, k_j \rangle$, where $k_j = H\text{-}Sim(x'||j)$.
    - Respond i-th batch query as $\sigma_{0,j}$, and then insert $\overrightarrow{D}_i$ and $\sigma_{0,j}$ into $\mathcal{LD}$ and $\mathcal{LS}$, respectively.
    - Update $j' \leftarrow j + 1,\ l \leftarrow j',\ i \leftarrow i + 1$ and continue to respond $\mathcal{A}$'s queries.

  • *How to respond queries to the $\textit{Break-in}$ oracle*: Assume that $FAS.Sig_{sk}(.)$ oracle was queried l individual messages up to now. If $l = L$ then reject the query (all private keys were used) and proceed to the *Forgery phase*. Otherwise, if $l \leq w$ then *abort* and return 0. Otherwise, give $\xi \leftarrow \langle \{a_m, b_m\}_{m=l+1}^{L-1}, x, x' \rangle)$ to $\mathcal{A}$.

- Forgery: Finally, $\mathcal{A}$ outputs a forgery as $(\overrightarrow{D}^*, \sigma^* = \langle s_{0,t}^*, k^* \rangle)$ on PK.

[3] Remind that $BAF.Kg$ algorithm already computed $\{r_m, k_m\}_{m=0}^{L-1}$ from seeds (x, x'), respectively, during the key generation and inserted these results into $\mathcal{HL}$.

By Definition 4.3, $\mathcal{A}$ wins if $BAF.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1$ and $\exists n \in \{0, \ldots, l\} : \overrightarrow{D}^*[n] \notin \{\mathcal{LD}[0]||\ldots||\mathcal{LD}[i]\}$ holds (recall that each $\mathcal{LD}[m], 0 \leq m \leq i$, is a batch query (a data vector), and $\overrightarrow{D}^*[n]$ is the n-th individual data item in the forgery data vector $\overrightarrow{D}^*$).

If $\mathcal{A}$ loses in the *FAEU-CMA* experiment, then $\mathcal{F}$ also loses in the *DL experiment*, and therefore $\mathcal{F}$ *aborts* and returns 0. Otherwise, $\mathcal{F}$ proceeds as follows:

Extraction: If $((e < w) \vee (\overrightarrow{D}^*[w] = D_w))$ then $\mathcal{F}$ *aborts* and return 0, where $e = |\overrightarrow{D}^*|$ (i.e., $\mathcal{A}$'s forgery is valid but it is not on the values (A_w, B_w)). Otherwise, $\mathcal{F}$ proceeds for the discrete log extraction as follows:

The forged aggregate signature $s^*_{0,e}$ is valid on PK, and $\mathcal{F}$ knows all the corresponding private keys of PK except $(a_w = y', b_w)$, which are included in the forged individual signature γ^*. Hence, $\mathcal{F}$ first isolates γ^* from $s^*_{0,e}$ as follows:

$$\gamma^* \leftarrow s^*_{0,e} - \sum_{0 \leq v \leq e, v \neq w} (a_v H\text{-}Sim(\overrightarrow{D}^*[v]||r_v||v) + b_v) \bmod q$$

Recall that $B_w \equiv (A^z_w)^{-1}\alpha^\gamma \bmod p$ holds due to the simulation. Moreover, since $BAF.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1$ holds, $\alpha^{\gamma^*} \equiv (A_w)^{h^*_w} B_w \bmod p$ also holds, where $(A_w, B_w) \in PK$ and $h^*_w \leftarrow H\text{-}Sim(\overrightarrow{D}^*[w]||r_w||w)$. Therefore, we write the following equations:

$$\begin{aligned} \alpha^\gamma &\equiv (\alpha^{y'})^z \alpha^{b_w} \bmod p, \\ \alpha^{\gamma^*} &\equiv (\alpha^{y'})^{h^*_w} \alpha^{b_w} \bmod p, \end{aligned}$$

$\mathcal{F}$ then extracts y' by solving the below modular linear equations (note that only unknowns are y' and b_w):

$$\begin{aligned} \gamma &\equiv y'z + b_w \bmod q, \\ \gamma^* &\equiv y'h^*_w + b_w \bmod q, \end{aligned}$$

Note that $Y \equiv \alpha^{y'} \bmod p$ holds, since $\mathcal{A}$'s forgery is valid and non-trivial on Y. Therefore, by Definition 4.1, $\mathcal{F}$ wins the *DL experiment*.

The success probability and execution time analysis of the above experiment, and the indistinguishability argument are as follows:

▶ *Success Probability Analysis*: We analyze the events that are needed for $\mathcal{F}$ to win the *DL experiment* as follows:

- $\overline{Abort1}$: $\mathcal{F}$ does not abort as a result of $\mathcal{A}$'s queries.
- $Forge$: $\mathcal{A}$ wins the *FAEU-CMA experiment*.
- $\overline{Abort2}$: $\mathcal{F}$ does not abort in the extraction.

- Win: $\mathcal{F}$ wins the ***DL experiment***

$\mathcal{F}$ succeeds if all of these events happen, and hence the probability $Adv^{DL}_{G,\alpha}(t')$ decomposes as,

$$Pr[Win] = Pr[\overline{Abort1}] \cdot Pr[Forge|\overline{Abort1}] \cdot Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge]$$

- *The probability of event $\overline{Abort1}$ occurs*: $\mathcal{F}$ may abort in the duration of $FAS.Sig_{sk}(.)$ queries, if one the below events occurs:

  i. Before obtaining k_w from $FAS.Sig_{sk}(.)$, if $\mathcal{A}$ queries the data item $D_w||r_w||w$ to the $RO(.)$ oracle and then requests its signature from the $FAS.Sig_{sk}(.)$ oracle, then $\mathcal{F}$ aborts. This occurs if $\mathcal{A}$ randomly guesses r_w or the master seeds (x, x'), from which r_w and its masking key k_w are derived. The probability that this occurs is $3/(q-1)$, which is negligible in terms of κ.

  ii. $\mathcal{A}$ queries the $FAS.Sig_{sk}(.)$ oracle on $0 \leq l \leq L-1$ data items and then queries the $Break\text{-}in$ oracle. If $l \leq w$ then $\mathcal{F}$ aborts (i.e., $\mathcal{F}$ does not know the corresponding private key of $A_w = Y$, and therefore cannot answer this query). The probability that $\mathcal{F}$ does *not* abort (i.e., the index w falls into the safe range $[0, l]$) is l/L.

  Omitting the negligible terms, the probability is $Pr[\overline{Abort1}] = (1 - 3/(q-1))(l/L) \cong \frac{l}{L}$.

- *The probability of event $Forge$ occurs*: If event $\overline{Abort1}$ occurs, then $\mathcal{A}$ also does *not* abort, since $\mathcal{A}$'s view is *statistically indistinguishable* from her view in a real-system (see the *indistinguishability argument* below). Hence, this occurs with $Pr[Forge|\overline{Abort1}] = Adv^{FAEU\text{-}CMA}_{BAF(p,q,\alpha)}(t, L', L, \mu', \mu)$.

- *The probability of event $\overline{Abort2}$ occurs*: If $\mathcal{A}$'s forgery is on (A_w, B_w) then $\mathcal{F}$ does *not* abort in the extraction. Since $w \leq |\overrightarrow{D}^*| = e \leq l$, this occurs with a probability at least $Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge] \geq 1/l$. Note that the probability that $\mathcal{A}$ wins on a data item D^*_w *without* querying it to the $RO(.)$ oracle is negligible in terms of κ, and therefore $H\text{-}Sim$ always returns an existing answer from $\mathcal{HL}$ in the extraction. Hence, after the extraction, the probability that $Y \not\equiv \alpha^{y'} \bmod p$ is also negligible.

Therefore, the upper bound on ***FAEU-CMA-advantage of BAF*** is as follows:

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

► *Execution Time Analysis*: The running time of $\mathcal{F}$ is that of $\mathcal{A}$ plus the time it takes to respond up to L' $RO(.)$ queries and L $FAS.Sig_{sk}(.)$ queries. Each new $RO(.)$ query requires drawing a random number from $\mathbb{Z}^*_q$, whose cost is denoted as $O(\kappa)$. Each $FAS.Sig_{sk}(.)$ query requires at least two modular additions and one modular multiplication, whose costs are denoted as $O(\kappa^2)$. Hence, the approximate running time of $\mathcal{F}$ is $O(t') = O(t + L\kappa^2 + L'\kappa)$.

► *Indistinguishability Argument*: The real-view of $\mathcal{A}$ is comprised of the public key $PK = (\{A_j, B_j, u_j\}_{j=0}^{L-1}, \{u'_j\}_{j=1}^{L-1}, p, q, \alpha, L)$ and the answers of $FAS.Sig_{sk}(.)$, $RO(.)$ and $Break\text{-}in$ oracles given as $\mathcal{LS}, \mathcal{HL} = \{h_m\}_{m=0}^{l'}$, and $\xi = \langle\{a_m, b_m\}_{m=l+1}^{L-1}, x, x'\rangle$, respectively. That is, $\overrightarrow{A}_{real} = \langle PK, \mathcal{LS}, \mathcal{HL}, \xi\rangle$, where all values are generated/computed by BAF algorithms as in the real system.

In $\overrightarrow{A}_{real}$, all variables in PK are computed from those values denoted as $\Upsilon = (\{a_j, b_j\}_{j=0}^{L-1}, x, x')$. Similarly, all variables in $\mathcal{LS}$ are computed from the variables in $\mathcal{HL}$ and Υ. That is, the joint probability distribution of all other variables in $\overrightarrow{A}_{real}$ are binary probabilities, which are decided by the joint probability distribution of $(\Upsilon, \mathcal{HL})$. Note that all variables in $(\Upsilon, \mathcal{HL})$ are random values in Z_q^*, where $|\mathcal{HL}| = l'$ and $|\Upsilon| = 2L + 2$. Hence, the joint probability distribution of $\overrightarrow{A}_{real}$ is,

$$\begin{aligned}
Pr[\overrightarrow{A}_{real} = \overrightarrow{a}] &= Pr[\overline{\Upsilon} = \Upsilon | \overline{\mathcal{HL}} = \mathcal{HL}] \\
Pr[\overrightarrow{A}_{real} = \overrightarrow{a}] &= Pr[\overline{x'} = x' | \overline{x} = x \wedge \overline{a}_0 = a_0 \wedge, \ldots, \wedge \overline{b}_0 = b_0 \wedge, \ldots, \wedge \overline{\mathcal{HL}} = \mathcal{HL}] \\
Pr[\overrightarrow{A}_{real} = \overrightarrow{a}] &= Pr[\overline{x'} = x' | \overline{x} = x \wedge \overline{a}_0 = a_0 \wedge, \ldots, \wedge \overline{h}_0 = h_0 \wedge, \ldots, \wedge \overline{h}_{l'} = h_{l'}] \\
&= \frac{1}{(q-1)^{2(L+1)+l'}}
\end{aligned}$$

The simulated-view of $\mathcal{A}$ is $\overrightarrow{A}_{sim}$, and it is equivalent to $\overrightarrow{A}_{real}$ except that in the simulation, the original decider variables (a_w, h_w, b_w) are replaced with the decider variables (y, z, c), where $(y, z, \gamma) \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$ and $c = \gamma - y \cdot z \bmod q$. That is, all $2(L+1) + l'$ deciders are random variables in Z_q^* as in the real system. Therefore, the joint probability distributions $Pr[\overrightarrow{A}_{real} = \overrightarrow{a}] = Pr[\overrightarrow{A}_{sim} = \overrightarrow{a}]$ (i.e., perfectly indistinguishable). □

We now prove that BAF is secure against the truncation attacks in Theorem 4.2 below.

Theorem 4.2

$$Adv^{TRUNC}_{BAF(p,q,\alpha)}(t, L', L, \mu', \mu) \leq \frac{L^2}{L-1} \cdot Adv^{DL}_{G,\alpha}(t'),$$

where $O(t') = O(t + L\kappa^2 + L'\kappa)$.

Proof: Let $\mathcal{A}$ be a *BAF attacker* and $(y \leftarrow Z_q^*, Y \leftarrow \alpha^y \bmod p)$. We construct a *DL attacker* $\mathcal{F}$ that uses $\mathcal{A}$ as a sub-routine on Y as follows:

Algorithm $F(Y)$

Set the target forgery/signature extraction index $w \stackrel{\$}{\leftarrow} [0, L-1]$,
$(sk, PK) \leftarrow BAF.Kg(1^\kappa, L)$, where $sk = (\langle a_0, b_0\rangle, x, x')$ and $PK \leftarrow (\{A_j\}_{0 \le j \le L-1, j \ne w}$,
$\{B_j\}_{0 \le j \le L-1, j \ne w+1}, \{u_j\}_{j=0}^{L-1}, \{u'_j\}_{j=1}^{L-1}, p, q, \alpha, L)$.
$(\langle a_j, b_j\rangle, x, x') \leftarrow BAF.Upd((\langle a_{j-1}, b_{j-1}\rangle, x, x'), L)$ for $j = 1, \ldots, L-1$,
Simulation: Simulate public keys (A_w, B_{w+1}), a batch signature s', and its corresponding random oracle answers (z_0, z_1) on $(\langle A_w, B_w\rangle, \langle A_{w+1}, B_{w+1}\rangle)$ as follows:

- $A_w \leftarrow Y$,
- $s' \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$,
- $(z_0, z_1) \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$,
- $B_{w+1} \leftarrow \alpha^{s'}(A_w^{z_0} B_w A_{w+1}^{z_1})^{-1} \bmod p$,

Initialize the counters as $l \leftarrow 0,\ j' \leftarrow 0,\ i \leftarrow 0,\ l' \leftarrow 0$,

Execute $\mathcal{A}^{RO(.),FAS.Sig_{sk}(.),Break\text{-}in}(PK)$: $\mathcal{F}$ maintains $\mathcal{HL}$, $\mathcal{LD}$, and $\mathcal{LS}$ to keep track the query results in the duration of the experiment as in Theorem 5.1.

- Queries: $\mathcal{F}$ handles $\mathcal{A}$'s queries as follows,

  • *How to respond queries to $RO(.)$ oracle*: $\mathcal{F}$ executes the function $H\text{-}Sim(.)$ that is defined in Theorem 4.1.

  • *How to respond i-th $FAS.Sig_{sk}(.)$ query*:

  - For each batch query i, $\mathcal{A}$ queries $FAS.Sig_{sk}(.)$ on $\overrightarrow{D}_i = \{D_{j'}, \ldots, D_j\},\ j \geq j'$ of her choice.
  - If $((j < w) \vee (w + 1 < j'))$ then compute $s_{j',j} \leftarrow \sum_{m=j'}^{j}(a_m h_m + b_m) \bmod q$ as in the real-system, where $h_m \leftarrow H\text{-}Sim(D_m||r_m||m)$ and $r_m \leftarrow H\text{-}Sim(x||m)$ (if $j' = j$ then $s_{j',j} = s_j$).

    Otherwise, if $((j = w) \vee (D_w||r_w||w \in \mathcal{HL}) \vee (D_{w+1}||r_{w+1}||(w+1)) \in \mathcal{HL})$ then *abort* and return 0 (an abort probability analysis is given in the following parts). Otherwise, compute $s_{j',j} \leftarrow [\sum_{j' \leq m \leq j, m \neq w, m \neq w+1}(a_m h_m + b_m)] + s' \bmod q$, where $h_m \leftarrow H\text{-}Sim(D_m||r_m||m)$. Insert tuples $\{(D_w||r_w||w, z_0),\ (D_{w+1}||r_{w+1}||(w+1), z_1)\}$ into $\mathcal{HL}$.
  - $s_{0,j} \leftarrow s_{0,j'-1} + s_{j',j} \bmod q$ (for initial $j' = 0$, $s_{0,j'-1} = 0$), and $\sigma_{0,j} \leftarrow \langle s_{0,j}, k_j \rangle$, where $k_j \leftarrow H\text{-}Sim(x'||j)$.
  - Respond i-th batch query as $\sigma_{0,j}$, and then insert $\overrightarrow{D}_i$ and $\sigma_{0,j}$ into $\mathcal{LD}$ and $\mathcal{LS}$, respectively.
  - Update $j' \leftarrow j + 1,\ l \leftarrow j',\ i \leftarrow i + 1$ and continue to respond her queries.

  • *How to respond queries to the Break-in oracle*: $\mathcal{A}$ queried $FAS.Sig_{sk}(.)$ oracle on l individual messages up to now. If $l = L$ then reject the query (all private keys were used) and proceed to the *Forgery phase*. Otherwise, if $l \leq w+1$ then *abort*. Otherwise, give $\xi \leftarrow \langle \{a_m, b_m\}_{m=l+1}^{L-1}, x, x' \rangle$ to $\mathcal{A}$.

- Forgery: Finally, $\mathcal{A}$ outputs a forgery as $(\overrightarrow{D}^*, \sigma^* = \langle s_{0,t}^*, k^* \rangle)$ on PK.

By Definition 4.4, $\mathcal{A}$ wins if $BAF.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1$ and $\overrightarrow{D}^* \subset \mathcal{LD}$ and $\forall I \subseteq \{0, \ldots, i\}, \overrightarrow{D}^* \neq ||_{m \in I} \mathcal{LD}[m]$ holds. Recall that i denotes the total number of batch queries $\mathcal{A}$ made to $FAS.Sig_{sk}(.)$ oracle, and $\mathcal{LD}[m], 0 \leq m \leq i$, denotes k-th batch query.

If $\mathcal{A}$ loses in the $TRUNC$ *experiment*, then $\mathcal{F}$ also loses in the *DL experiment*, and therefore $\mathcal{F}$ *aborts* and returns 0. Otherwise, $\mathcal{F}$ proceeds as follows:

Extraction: This occurs if $\mathcal{A}$ performs a tail-truncation attack on the simulated values (i.e., individual public keys) (A_w, B_{w+1}). Note that, due to the indexing mechanism, any non-tail-truncation attack (see Section 4.2.4 for a discussion on aggregate signature extraction and truncation attacks) results in a traditional forgery (i.e., $\mathcal{A}$ has to modify index and/or random seeds), which was analyzed in Theorem 4.1. Therefore, we only analyze the tail-truncation case as below:

$\mathcal{A}$ queried the $FAS.Sig_{sk}(.)$ oracle on $l \leq L-1$ data items and then queried the $Break\text{-}in$ oracle. Hence, the final signature that $\mathcal{A}$ obtained from $FAS.Sig_{sk}(.)$ is $\sigma_{0,l} = \langle s_{0,l}, k_l \rangle$. A tail-truncation attack occurs if $\mathcal{A}$ extracts a valid aggregate signature $s^*_{0,e} = s_{0,e}$ from $s_{0,l}$ without querying $s_{0,e}$ or $s_{e+1,l}$ to $FAS.Sig_{sk}(.)$, where $e = |D^*|$ and $s_{0,l} \equiv s_{0,t} + s_{t+1,l} \bmod q$ ($0 < e < l$ and for $e+1 = l$, $s_{l,l} = s_l$). Notice that, different from the traditional forgery case (analyzed in Theorem 5.1), in a tail-truncation attack, $\mathcal{A}$ does not modify the data items corresponding to the truncated signature $s^*_{0,e}$. That is, $s^*_{0,e} = s_{0,e}$ is valid on $\overrightarrow{D}^* = (D_0, \ldots, D_e) \subset \mathcal{LD}$ as denoted in the winning condition.

Recall that $\mathcal{F}$ embedded Y into A_w and then setup the simulation as $B_{w+1} \leftarrow \alpha^{s'}(A_w^{z_0} B_w A_{w+1}^{z_1})^{-1} \bmod p$, which implies the below equality holds,

$$s' \equiv y \cdot z_0 + b_w + a_{w+1} \cdot z_1 + b_{w+1} \bmod q$$

, where (y, b_{w+1}) are the unknowns.

Due to the above simulation, if $\mathcal{A}$ extracts the individual signature s'_0 from the batch signature s' (that is valid on the tokens $(\langle A_w, B_w \rangle, \langle A_{w+1}, B_{w+1} \rangle)$, then $\mathcal{F}$ achieves the discrete log extraction. That is, since s'_0 is valid on (A_w, B_w), the equation $\alpha^{s'_0} \equiv A_w^{z_0} B_w \bmod p$ also holds. Therefore, $\mathcal{F}$ can extract y by solving the modular equation $s'_0 \equiv y \cdot z_0 + b_w \bmod q$.

Based on the above argument, $\mathcal{F}$ checks if $e = w$ and $\overrightarrow{D}^*[w] = D_w$ (i.e., $\mathcal{A}$ splits the batch signature s' that $\mathcal{F}$ embedded into $\mathcal{A}$'s $FAS.Sig_{sk}(.)$ query on the values $\langle A_w, B_w \rangle, \langle A_{w+1}, B_{w+1} \rangle$). If this is not the case, $\mathcal{F}$ *aborts* and return 0. Otherwise, $\mathcal{F}$ proceeds for the discrete log extraction as follows:

The forged (extracted) aggregate signature $s^*_{0,e}$ is valid on PK, and $\mathcal{F}$ knows all the corresponding private keys of PK except $a_w = y'$, which is included in the individual signature s'^*_0. Hence, $\mathcal{F}$ first isolates s'^*_0 from $s^*_{0,e}$ as follows (note that $e = w$):

$$s'^*_0 \quad \leftarrow \quad s^*_{0,w} - \sum_{v=0}^{w-1} (a_v H\text{-}Sim(\overrightarrow{D}^*[v] || r_v || v) + b_v) \bmod q$$

Remind that since $BAF.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1$ holds, the equality $\alpha^{s'^*_0} \equiv Y^{z_0} B_w \bmod p$ holds. Hence, $\mathcal{F}$ extracts y' as,

$$y' \quad \equiv \quad (s'^*_0 - b_w) \cdot z_0^{-1} \bmod q$$

If $\mathcal{F}$ does not abort then $Y \equiv \alpha^{y'} \bmod p$ holds, since $\mathcal{A}$'s forgery is valid and non-trivial on Y. Therefore, by Definition 4.1, $\mathcal{F}$ wins the *DL experiment*.

The success probability and the execution time analysis of the above experiment, and the indistinguishability argument are as follows:

► *Success Probability Analysis*: The probability $Adv^{DL}_{G,\alpha}(t')$ is as follows:

$$Pr[Win] = Pr[\overline{Abort1}] \cdot Pr[Forge|\overline{Abort1}] \cdot Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge]$$

- *The probability of event $\overline{Abort1}$ occurs*: $\mathcal{F}$ may abort in the duration of $FAS.Sig_{sk}(.)$ queries, if one the below events occurs:

  i. $\mathcal{A}$ queries the $FAS.Sig_{sk}(.)$ oracle on $l \leq L-1$ data items and then queries the $Break\text{-}in$ oracle. If $l \leq w+1$ then $\mathcal{F}$ aborts (i.e., $\mathcal{F}$ does not know the corresponding private key of $(A_w = Y, B_{w+1})$, and therefore cannot answer this query). The probability that $\mathcal{F}$ does *not* abort (i.e., the index w falls into the safe range $[0, l]$) is l/L.

  ii. $\mathcal{A}$ makes a batch query $\overrightarrow{D}_i = \{D_{j'}, \ldots, D_j\},\ j \geq j'$ for $j = w$. $\mathcal{F}$ cannot answer this query, since $\mathcal{F}$ does not know the individual signature corresponding (A_w, B_w) that he simulated on Y. Since the target forgery index is chosen as $w \xleftarrow{\$} [0, L-1]$, this occurs with a probability $1/L$.

  iii. Before obtaining (k_{w-1}, k_w) from the $FAS.Sig_{sk}(.)$, $\mathcal{A}$ first queries data items $(D_w||r_w||w, D_{w+1}||r_{w+1}||w+1)$ to the $RO(.)$ oracle, and then request their corresponding signature from the $FAS.Sig_{sk}(.)$ oracle. This happens if $\mathcal{A}$ randomly guesses one of these values (r_w, r_{w+1}, x, x'), whose probability is $4/(q-1)$.

  Omitting the negligible terms, the probability is $Pr[\overline{Abort1}] = (1 - 1/L)(1 - 4/(q-1))(l/L) \simeq \frac{(L-1)l}{L^2}$.

- *The probability of event Forge occurs*: If event $\overline{Abort1}$ occurs, then $\mathcal{A}$ also does *not* abort, since $\mathcal{A}$'s view is *statistically indistinguishable* from her view in a real system (see the *indistinguishability argument* below). Hence, this occurs with $Pr[Forge|\overline{Abort1}] = Adv^{TRUNC}_{BAF(p,q,\alpha)}(t, L', L, \mu', \mu)$.

- *The probability of event $\overline{Abort2}$ occurs*: If $\mathcal{A}$'s forgery is on (A_w, B_w) then $\mathcal{F}$ does *not* abort in the extraction. Given that $w \leq |\overrightarrow{D}^*| = t \leq l$, this occurs with a probability at least $Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge] \geq 1/l$.

Therefore, the upper bound on *$TRUNC$-advantage of BAF* is as follows:

$$Adv^{TRUNC}_{BAF(p,q,\alpha)}(t, L', L, \mu', \mu) \quad \leq \quad \frac{L^2}{L-1} \cdot Adv^{DL}_{G,\alpha}(t').$$

► *Execution Time Analysis*: It is as in the Theorem 4.1.

► *Indistinguishability Argument*: Recall that $\overrightarrow{A}_{real} = \langle PK, \mathcal{LS}, \mathcal{HL}, \xi \rangle$ and $Pr[\overrightarrow{A}_{real} = \overrightarrow{a}] = \frac{1}{(q-1)^{2(L+1)+l'}}$ as given in Theorem 4.1.

The simulated-view of $\mathcal{A}$ is $\overrightarrow{A}_{sim}$, and it is equivalent to $\overrightarrow{A}_{real}$ except that in the simulation, the original decider variables $(a_w, h_w, h_{w+1}, b_{w+1})$ are replaced with the decider variables (y, z_0, z_1, c), where $(y, z_0, z_1, s') \xleftarrow{\$} \mathbb{Z}_q^*$ and $c \leftarrow s' - y \cdot z_0 - b_w - a_{w+1} \cdot z_1 \bmod q$ (note that $(b_w, a_{w+1}) \xleftarrow{\$} \mathbb{Z}_q^*$ as in the real system). That is, all $2(L+1)+l'$ deciders are random variables in Z_q^* as in the real system. Therefore, the joint probability distributions $Pr[\overrightarrow{A}_{real} = \overrightarrow{a}] = Pr[\overrightarrow{A}_{sim} = \overrightarrow{a}]$ (i.e., perfectly indistinguishable) as in Theorem 4.1. □

Another security concern in audit logging is the *delayed detection attack* identified in [78, 79]. *Delayed detection attack* targets the audit logging mechanisms requiring online TTP support to enable the log verification. In these mechanisms, the verifiers cannot detect whether the log entries are modified before a TTP provides required keying information. Due to the lack of immediate verification, these mechanisms cannot fulfill the requirement of applications in which the log entries should be processed in real-time. Ma et al. [79] shows that many existing schemes are vulnerable to these attacks (e.g., [18], [17], [104], [103]).

Remark that BAF schemes are also secure against the delayed detection attack: In BAF schemes, the verifiers are provided with all the required public keys before deployment. Hence, both schemes achieve the immediate verification property, and therefore are secure against the delayed detection attack.

4.5 Performance Analysis and Comparison

In this section, we present the performance analysis of our schemes. We also compare BAF and FI-BAF with the previous schemes using the following criteria: (i) The computational overhead of signature generation/verification operations (including the key update cost); (ii) signature/key storage and communication overheads depending on the size of signing key and the size of signature; (iii) desirable properties such as public verifiability, offline TTP and immediate verification, and (iv) security properties such being resilient to the truncation and delayed detection attacks and the provable security.

We list the notation used in our performance analysis and comparison in Table 4.2. Based on this notation, for each of the above category, we first provide the analysis of BAF and FI-BAF, and then present their comparison with the previous schemes both analytically and numerically.

4.5.1 Computational Overhead

We implement our schemes on an Elliptic Curve (EC) [87], which offers small key/signature sizes and high computational efficiency [56]. Let G be a generator of group $\mathbb{G}$ defined on an elliptic curve $E(F_p)$ over a prime field F_p, where p is a large prime number and q is the order of G such that $p > q$ and $q|(p-1)$. kG, where $k \in F_q$ is an integer, denotes a *scalar multiplication*.

Table 4.2: Notation for performance analysis and comparison

$Muln'$: Mul. mod $n' = p'q'$, where (p', q') are large primes		R: # of verifiers
$Mulp/Mulq$: Mul. mod p and mod q, respectively		$Addq$: Addition mod q
$EMul/EAdd$: ECC scalar mul. and addition over F_p, respectively		MtP: ECC map-to-point
Exp: Exponentiation mod p	Sqr: Squaring mod n'	PR: ECC pairing
$GSig/GVer$: Generic signing and verification, respectively		H: FDH operation
x: FssAgg-BM/AR security parameter	L: max. # of key updates	l: # of data items

Suggested bit lengths to achieve $\kappa = 80$ for the above parameters are as follows for each compared scheme: Large primes ($|p| = 512, |q| = 160$) for BAF/FI-BAF, Logcrypt and FssAgg-BLS. ($|n'| = 1024, x = 160$) for FssAgg-AR and FssAgg-BM, where n' is Blum-Williams integer [75]. Note that BAF/FI-BAF, Logcrypt and FssAgg-BLS were implemented in EC.

In BAF, signature computation and key update require $(Mulq+2(Addq+H))$ and $2H$, respectively. Hence, BAF requires $(Mulq + 4H + 2Addq)$ in total to sign a single log entry. In FI-BAF, the cost of signing a single log entry is the twice of that of BAF. Note that since the overhead of modular addition is negligible, the total cost of signing a single log entry is dominated by hash and modular multiplication operations.

By following the BAF signature verification equation, verifying a single log entry requires $EMul + EAdd + 2(H + Addq)$. Note that it is possible to avoid performing one $EAdd$ for per log entry by using an optimization: In the key generation phase, we can compute and release $B'_j = (\sum_{i=0}^{j} b_j)G$ instead of $B_j = b_jG$ for $j = 0, \ldots, L-1$ to speed up the signature verification. In this way, verifiers can perform the signature verification with only one $EAdd$ regardless of the value of l. Hence, the signature verification cost of BAF for l received log entries is $(l + 1) \cdot (EMul + 2H)$. The signature verification cost of FI-BAF is the same with that of BAF.

Comparison: The closest counter parts of our schemes are FssAgg schemes [75, 76, 78, 79]. The signature generation of FssAgg-BLS [76] is expensive due to Exp and MtP, while its signature verification is highly expensive due to PR. Different from FssAgg-BLS, FssAgg-BM and FssAgg-AR [75] rely on more efficient operations such as Sqr and $Muln'$. However, these schemes are also computationally costly, since they require heavy use of such operations. For instance, FssAgg-BM requires $x \cdot Sqr + (1 + x/2)Muln'$ (i.e., x≅160 [75]) for the signature generation (key update plus the signing cost), and it requires $L \cdot Sqr + (l + x \cdot l/2)Muln'$ for the signature verification. Similarly, FssAgg-AR requires $(3x)Sqr + (2 + x/2)Muln'$ for the signature generation, and it requires $x(L+l)Sqr + (2l + l \cdot x)Muln'$ for the signature verification. iFssAgg schemes [79] double the signing (*FssAgg.Sig*) and verifying (*FssAgg.Ver*) costs of their base FssAgg schemes to completely eliminate the truncation attack.

Logcrypt uses a digital signature scheme to sign and verify each log entry separately without signature aggregation [57], and thus has standard signature costs (e.g., we use ECDSA [6] for Logcrypt in our comparison). The symmetric schemes [18, 76, 103, 104] are in general efficient, since they only need symmetric cryptographic operations.

Table 4.3 summarizes the analytical comparison of all these schemes for their computational costs

Table 4.3: Computation involved in BAF, FI-BAF and previous schemes

		Sig	Upd	Ver
PKC-based	*BAF*	$Mulq + 2H$	$2H$	$(l+1)(EMul + 2H)$
	FI-BAF	$2 \cdot BAF.Sig$	$2 \cdot BAF.Upd$	$BAF.Ver$
	FssAgg-BLS	$MtP + Exp + Mulp$	H	$l(Mulp + H + PR)$
	FssAgg-BM	$(1 + \frac{x}{2})Muln'$	$x \cdot Sqr$	$L \cdot Sqr + (l + \frac{l \cdot x}{2})Muln'$
	FssAgg-AR	$x \cdot Sqr + (2 + \frac{x}{2})Muln'$	$(2x)Sqr$	$x(L+l)Sqr + (2l + l \cdot x)Muln'$
	iFssAgg	$2 \cdot FssAgg.Sig$	$2 \cdot FssAgg.Upd$	$2 \cdot FssAgg.Ver$
	Logcrypt	$GSig$	-	$l \cdot GVer$
Symmetric		$2H$	H	$l \cdot H$

Table 4.4: Execution times (in ms) of BAF, FI-BAF and previous schemes for a single log entry

	PKC-based						Symmetric
	BAF	FI-BAF	FssAgg Schemes			Logcrypt	
			BLS/iBLS	BM/iBM	AR/iAR		
Sig	0.01	0.02	1.83/3.66	3.6/7.2	7.71/15.42	1.02	0.006
Ver	0.74	0.76	24.5/49	1.7/3.4	5.3/10.6	1.23	0.006

using the notation given in Table 4.2.

In addition to the analytical comparison, we also measure the execution times of all the compared schemes on a computer with an Intel(R) Xeon(R)-E5450 3GHz CPU and 2GB RAM running Ubuntu 9.04. The execution times of BAF, FI-BAF, FssAgg-BLS, Logcrypt, and the symmetric schemes [18, 76, 103, 104] were measured using implementations based on the MIRACL library [106]. The execution times of FssAgg-AR/BM were computed using implementations based on the NTL library [108]. Table 4.4 shows the signing/verifying costs of a single log entry in each scheme.

When compared with PKC-based FssAgg-BLS, FssAgg-BM, FssAgg-AR and Logcrypt, BAF is 183, 360, 771, and 102 times faster for loggers, respectively. Similarly, when compared with its immutable counterparts (iFssAgg schemes), FI-BAF is also 183, 360 and 102 times faster for loggers, respectively. Note that both BAF and FI-BAF signature verifications are also more efficient than the previous schemes. When compared with FssAgg-BLS, FssAgg-BM, FssAgg-AR and Logcrypt, BAF is 33, 2.3, 7.1, and 1.6 times faster, respectively. Similarly, when compared with its immutable counterparts, FI-BAF is again 64, 4.47, and 13.9 times faster, respectively.

This computational efficiency makes BAF/FI-BAF the best alternative among existing schemes for secure logging with public verifiability in resource-constrained devices.

Figure 4.1 and Figure 4.2 further show the comparison of BAF and the previous schemes that allow public verification in terms of signature generation and verification time as the number of log entries increases. Similarly, Figure 4.3 and Figure 4.4 demonstrate the comparison of FI-BAF and iFssAgg schemes in terms of signature generation and verification time as the number of log entries increases. These figures clearly show that BAF and FI-BAF are the most computationally efficient schemes among

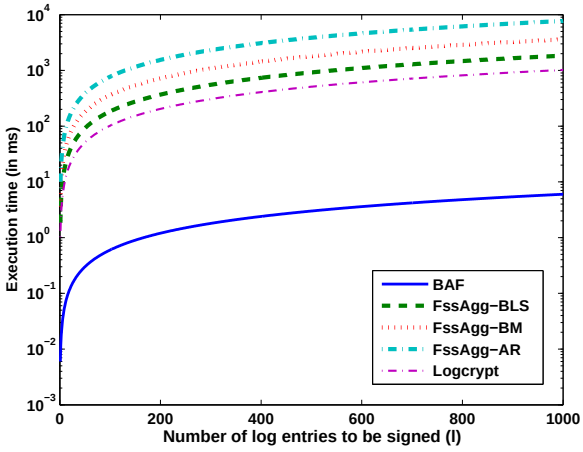

Figure 4.1: Signing time comparison of BAF and its counterparts (in ms)

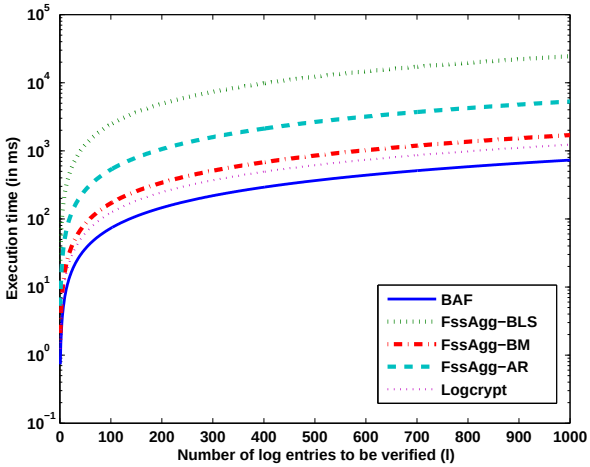

Figure 4.2: Verification time comparison BAF and its counterparts (in ms)

all these choices.

Remark that the key generation phase of all compared PKC-based schemes require $O(L)ExpOp$, where L denotes the maximum number of key updates (i.e., the maximum number data items to be signed). In BAF and FI-BAF, the key generation cost is $O(L)(Add + EMul + 2H)$ and $O(2L)(Add + EMul + 2H)$, respectively. The key generation cost is $O(L)(H + Emul)$, $O(L)(x \cdot Sqr + x/2 \cdot Muln')$, $O(2L)(x \cdot Sqr + x/2 \cdot Muln')$ and $O(L)(H + Emul)$ for FssAgg-BLS, FssAgg-BM, FssAgg-AR and Logcrypt (with ECDSA), respectively. Therefore, the key generation of BAF is more efficient than that of FssAgg-AR and FssAgg-BM, but slightly less efficient than that of FssAgg-BLS and Logcrypt. The key generation of FI-BAF is also more efficient than that of iFssAgg schemes with FssAgg-AR and FssAgg-BM, but slightly less efficient than that of FssAgg-BLS.

When compared with the signature generation of previous symmetric logging schemes (e.g., [17, 18, 76, 103, 104]), BAF and FI-BAF signature generation is comparable efficient even though they are PKC-based schemes. However, signature verification of the symmetric logging schemes is more efficient than all the existing PKC-based schemes, including BAF and FI-BAF. Note that these symmetric schemes sacrifice the public verifiability and certain security properties (e.g., truncation and delayed detection attacks) to achieve this verifier efficiency.

4.5.2 Storage and Communication Overheads

In BAF, the size of signing key is $4|q|$ (e.g., $|q|$=160), and the size of authentication tag is $2|q|$. Since BAF allows the signature aggregation, independent of the number of data items to be signed, the size of resulting authentication tag is always constant (i.e., $2|q|$). Furthermore, BAF derives the current signing key from the previous one, and then deletes the previous signing key from the memory (i.e., evolve-

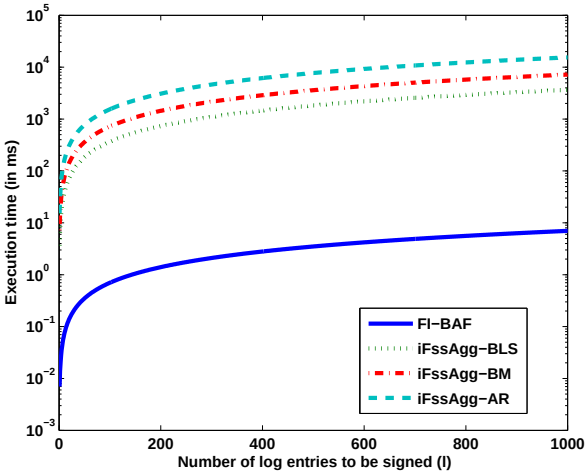

Figure 4.3: Signing time comparison of FI-BAF and iFssAgg schemes (in ms)

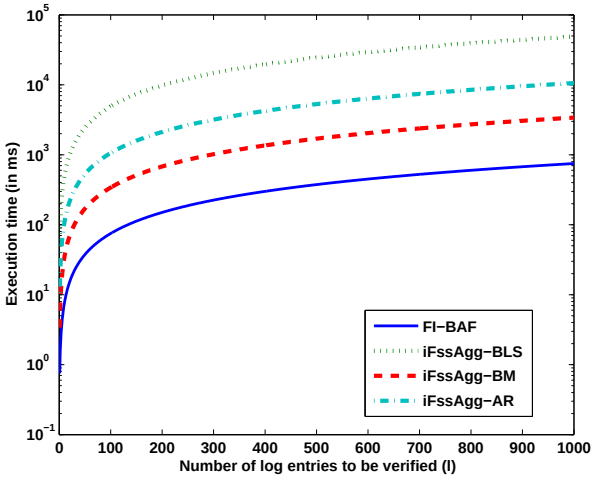

Figure 4.4: Verification time comparison FI-BAF and iFssAgg schemes (in ms)

Table 4.5: Signature/key storage and communication overheads of BAF, FI-BAF and previous schemes

Criteria	BAF	FI-BAF	FssAgg Schemes					Logcrypt	Sym.
			BLS	BM	AR	MAC	iFssAgg		
Key Size	$4\|q\|$	$2 \cdot BAF$	$\|q\|$	$x\|n\|$	$2\|n\|$	$\|H\|$	$2 \cdot FssAgg$	$\|q\|$	$\|H\|$
Sig. Size	$2\|q\|$	$2 \cdot BAF$	$\|p\|$	$\|n\|$	$2\|n\|$	$\|H\|$	$2 \cdot FssAgg$	$2\|q\|$	$\|H\|$
Storage	$6\|q\|$	$O(2l)\|q\|$	$\|p\| + \|q\|$	$x\|n\|$	$4\|n\|$	$O(R)\|H\|$	$O(2l)\cdot FssAgg$	$O(l)\|q\|$	$O(l)\|H\|$
Comm.	$2\|q\|$	$O(2l)\|q\|$	$\|p\|$	$\|n\|$	$\|n\|$	$\|H\|$	$O(2l)\cdot FssAgg$	$O(2l)\|q\|$	$O(l)\|H\|$

delete strategy [121]). That is, the size of signing key is also constant (i.e., $4|q|$). Therefore, both the signature storage and communication overheads of BAF are constant (i.e., $6|q|$ and $2|q|$, respectively).

In FI-BAF, the size of signing key is two times of that of BAF. Since it uses BAF as a sub-routine, its aggregate signature is small-constant as $2|q|$. However, to enable a fine-grained verification of log entries, FI-BAF keeps their corresponding individual signatures, and therefore its signature storage and communication overheads are both $O(2l)|q|$ for l log entries.

Comparison: The storage and communication overheads are measured according to the size of a single signing key, the size of a single authentication tag, and the growth rate of these two parameters with respect to the number of data items to be processed, that is, whether they grow linearly, or remain constant for an increasing number of data items to be processed. Table 4.5 summarizes the comparison.

The symmetric schemes (e.g., [17, 18, 103, 104], and FssAgg-MAC in [76]) all use a MAC function to compute an authentication tag for each log entry with a different key, where the sizes of the key and the resulting tag are both $|H|$ (e.g., 160 bit for SHA-1). Instead of using MACs, Logcrypt uses a digital signature such as ECDSA, where the size of signing key is $|q|$ (e.g., 160 bit) and the size of signature is $2|q|$, respectively.

Table 4.6: Scalability and security properties of BAF, FI-BAF, and previous schemes

Criteria	BAF / FI-BAF	FssAgg/iFssAgg				Logcrypt	Symmetric
		BLS	BM	AR	MAC		
Public Verifiability	Y	Y	Y	Y	N	Y	N
Offline TTP	Y	Y	Y	Y	Y	Y	N
Immediate Verification	Y	Y	Y	Y	Y	Y	N
Resilient to Delayed Detection Attack	Y	Y	Y	Y	Y	Y	N
Resilient to Truncation (Deletion) Attack	Y	Y	Y	Y	Y	N	N
Security Argument (the Truncation Attacks)	P	H	H	H	H	N	N

Y/N denotes "yes/no", and P/H denotes "provable/heuristic"

These schemes cannot achieve the signature aggregation, and therefore they require storing/transmitting an authentication tag for each log entry. That is, the signature storage and communication overheads of these symmetric schemes and Logcrypt are all linear as $O(l)|H|$ and $O(l)|q|$, respectively. Different from these schemes, FssAgg-MAC achieves the signature aggregation, and its signature communication overhead is only $|H|$. However, since FssAgg-MAC requires symmetric key distribution, its key storage overhead is also linear (i.e., $O(R)|H|$).

The PKC-based FssAgg-BLS [76], FssAgg-BM and FssAgg-AR [75] achieve the signature aggregation in a publicly verifiable way, and therefore their signature storage/communication overheads are constant. Table 4.5 shows that they are efficient in terms of both the storage and communication overheads. iFssAgg schemes [79] demand linear signature storage and communication when compared with their base schemes due to the need of storing and transmitting individual signatures (denoted as $O(2l) \cdot FssAgg$ in Table 4.5).

BAF has constant signature storage and communication overheads, and is significantly more efficient than all the schemes that incur linear signature (or key) storage and communication overheads (e.g., [17, 18, 57, 76, 103, 104]). BAF is also more efficient than FssAgg-AR/BM [75], but less efficient than FssAgg-BLS [76], as shown in Table 4.5. Similar to its immutable counterpart iFssAgg [79] schemes, FI-BAF also demands linear signature storage and communication overheads.

4.5.3 Scalability and Security

BAF and FI-BAF are publicly verifiable, and they do not need an online TTP support for the signature verification. Furthermore, BAF and FI-BAF do not rely on a time factor to be publicly verifiable, and therefore they achieve the immediate verification property (in contrast to HaSAFSS schemes [121]). Finally, they are proven to be resilient against the truncation attacks in ROM, whereas all the previous cryptographic secure audit logging schemes only rely on heuristic security arguments about the truncation attacks. Note that our schemes are also resilient against the delayed detection attacks as in all

PKC-based schemes.

Table 4.6 shows the comparison of BAF and FI-BAF with the previous schemes in terms of their scalability and security properties. The symmetric schemes (e.g., [17, 18, 103, 104]) cannot achieve the public verifiability and require online TTP support to enable the log verification. The lack of public verifiability and the requirement for an online TTP limit their applicability to large distributed systems. Furthermore, they are vulnerable to both truncation and delayed detection attacks [78, 79]. FssAgg-MAC [76] does not need an online TTP and is secure against the aforementioned attacks. However, it is not publicly verifiable.

PKC-based FssAgg schemes, iFssAgg schemes and Logcrypt are publicly verifiable. They do not need online TTP support, and can achieve the immediate verification.

BAF and FI-BAF, achieving all the required scalability and security properties, are also much more computational efficient than all these PKC-based schemes.

4.5.4 Limitation

In BAF schemes, the size of public key is linear with respect to the number time periods. This may incur high storage overhead to the verifiers. However, for our envisioned applications, the signer computational/storage/communication efficiency is more important than the verifier storage efficiency alone (as assumed in all PKC-based FSI models (see Section 4.2)). For example, the signer computational/storage efficiency is critically important for secure logging in resource-constrained devices such as RFID tags [9] and wireless sensors [76], where the verifiers (e.g., laptops) can tolerate the storage overhead.

Remark that some generic forward-secure signature constructions (e.g., the storage efficient construction in [80]) offer sub-linear public key sizes. However, such constructions also require *several online ExpOps* at the signer side, and therefore they are not practical for resource-constrained applications. They also do not provide the "all-or-nothing" property.

4.6 Conclusion

In this paper, we developed a new class of forward-secure and aggregate audit logging schemes, which we refer to as *Blind-Aggregate-Forward (BAF)* and *Fast-Immutable BAF (FI-BAF)* logging schemes. BAF simultaneously achieves several desirable properties for secure audit logging, including near-zero logger computational overhead, small-constant signature storage/communication overheads, public verifiability (without online TTP support), immediate log verification and provable security. Our extended scheme FI-BAF enables the selective verification of individual log entries via their corresponding individual signatures while preserving the security and performance advantages of the BAF. Our comparison with the previous alternatives show that our schemes are ideal choices for secure audit logging in

resource-constrained devices.

While BAF schemes are optimally efficient at the signer side, they still require an ExpOp per-data item for the signature verification. Moveover, their public key size grows linearly with respect to the number of time periods. In the next chapter, we present LogFAS [124] that achieves the verifier efficiency while preserving the immediate verification, public verifiability and provable security properties.

Chapter 5

Efficient, Compromise Resilient and Append-Only Cryptographic Constructions with Verifier Efficiency

In previous chapter, we presented BAF schemes that are ideal for real-time secure audit logging in resource-constrained devices by achieving the optimal signer efficiency. However, BAF signature verification requires a modular exponentiation for each data item to be verified. Despite being more efficient than all previous publicly verifiable schemes (with the exception of our HaSAFSS schemes that are ideal only for non-real-time applications), this still may be computationally expensive for some verification-intensive applications. Moreover, each verifier is required to store L tokens (i.e., individual public keys) for each signer (i.e., $L \cdot S$ keys in total where S denotes the number of signers in the system).

In this chapter, we present a new class of secure audit logging schemes called *Log Forward-secure and Aggregate Signature (LogFAS)* to address these limitations. LogFAS is designed to address applications, in which system auditors (i.e., verifiers) are required to verify large number of log entries in real-time simultaneously. We first develop a main LogFAS scheme, and then extend it to provide additional capabilities.

The desirable properties of LogFAS are outlined below. The first three properties show the efficiency of LogFAS compared with its PKC-based counterparts, while the other two properties demonstrate the applicability, availability and security advantages over their symmetric counterparts. Table 5.1 summarizes these properties and compares LogFAS with its counterparts.

1. *Efficient Log Verification with $O(1)$ ExpOp*: All existing PKC-based secure audit logging schemes (e.g., [57, 75, 76, 78, 79, 120, 121]) require $O(L)$ ExpOps to verify L log entries, which make them costly. LogFAS is the first PKC-based secure audit logging scheme that achieves signature verification with only a small number of ExpOps (and $O(L)$ hash operations denoted as H). Specifically,

Table 5.1: Comparison of LogFAS schemes and their counterparts for performance, applicability, availability and security parameters

Criteria		*PKC-based*					*SYM*
		LogFAS	FssAgg/iFssAgg		BAF	Logcrypt	[18, 104]
Computational			AR	BM / BLS			
On-line	*Sig&Upd (per item)*	$O(1)ExpOp$	$O(1)ExpOp$		$O(1)H$	$O(1)ExpOp$	$O(1)H$
	Ver, (L items)	$O(1)ExpOp + O(L)H$	$O(L)ExpOp + O(L)H$				$O(L)H$
	Subset ver (l')	$O(1)ExpOp + O(l')H$	$O(2l')(ExpOp + H)$		$O(l')(ExpOp + H)$	Not immutable	$O(l')H$
	Efficient Search	Available	Not Available				-
Key Generation (Offline)		$O(L)ExpOp$					$O(L)H$
Storage	*Verifier*	$O(1)\|K\|$	$O(S)\|K\|$	$O(L \cdot S)\|K\|$			$O(S)\|K\|$
	Signer	$O(L)(\|D\| + \|K\|)$	$O(L)\|D\| + O(1)\|K\|$			$O(L)\|K\|$	$O(L)\|K\|$
Communication		$O(L)\|D\|$					
Public Verifiability		Y	Y				N
Offline Server		Y	Y				N
Immediate Verification		Y	Y				N
Immediate Detection		Y	Y				N
Truncation Resilience		Y	Y			N	N
Security Argument (truncation)		Provable	Heuristic		Provable	N	N

LogFAS is the only PKC-based forward-secure and append-only scheme that can verify $O(L)$ items with $O(1)$ ExpOp; all other similar schemes require $O(L)$ExpOp. Similarly, LogFAS is the only one achieving $O(1)$ key storage on the verifier side, while all other schemes incur either linear or quadratic storage overhead ($S, |D|, |K|$ denote number of signers in the system, the approximate bit lengths of a log entry and unit keying material, respectively). At the same time, LogFAS is the only scheme that enables truncation-free subset verification and sub-linear search simultaneously.

LogFAS can verify L log entries with only $O(1)$ ExpOps regardless of the value of L. Therefore, it is much more efficient than all its PKC-based counterparts, and is also comparably efficient with symmetric schemes (e.g., [17, 18, 76, 103, 104]) at the verifier side.

2. *Efficient Fine-grained Verification and Change Detection*: LogFAS allows fine-grained verification with advantages over iFssAgg, the only previous solution for fine-grained verification:

   i. Unlike iFssAgg schemes [79], LogFAS prevents the truncation attack in the presence of individual signatures without doubling the verification cost.

   ii. LogFAS can verify any selected subset with $l' < L$ log entries with $O(1)$ ExpOps, while iFssAgg and BAF schemes require $O(2l')ExpOps$ and $O(l')ExpOps$, respectively. Note that HaSAFSS schemes do not provide this fine-grained verification property.

   iii. LogFAS can identify the corrupted log entries with a *sub-linear* number of ExpOps when most log entries are intact. In contrast, iFssAgg and BAF schemes always require a linear number of ExpOps.

3. *Verifier Storage Efficiency with $O(1)$ Overhead*: Each verifier in LogFAS only stores one public key independent of the number of loggers or the number of log entries to be verified. Therefore, it is the most verifier-storage-efficient scheme among all existing PKC-based alternatives. This enables verifiers to handle a large number of log entries and/or loggers simultaneously without facing any storage problem.

4. *Provable Security*: We prove LogFAS to be *ForWard-secure Existentially Unforgeable against Chosen Message Attack (FWEU-CMA)* in ROM [13]. Furthermore, unlike some previous symmetric schemes [18, 103, 104], LogFAS schemes are also secure against both truncation and delayed detection attacks.

5. *Public Verifiability, Offline TTP and Immediate Verification:* While achieving all the above properties, LogFAS preserves the immediate verification, public verifiability and independence from online TTP properties. Therefore, LogFAS preserves all the advantages of BAF schemes over HaSAFSS and some other previous schemes (e.g., [18, 103, 104, 121]).

The remainder of this chapter is organized as follows. Section 5.1 provides preliminary notation and definitions. Section 5.2 describes the syntax and models. Section 5.3 presents the proposed LogFAS schemes. Section 5.4 and Section 5.5 provide the security and the performance analysis of the LogFAS schemes, respectively. Section 5.6 concludes this chapter.

5.1 Preliminaries

Notation. $||$ denotes the concatenation operation. $|x|$ denotes the bit length of variable x. $x \xleftarrow{\$} \mathcal{S}$ denotes that variable x is randomly and uniformly selected from set $\mathcal{S}$. For any integer l, $(x_0, \ldots, x_l) \xleftarrow{\$} \mathcal{S}$ means $(x_0 \xleftarrow{\$} \mathcal{S}, \ldots, x_l \xleftarrow{\$} \mathcal{S})$. We denote by $\{0,1\}^*$ the set of binary strings of any finite length. H is an ideal cryptographic hash function, which is defined as $H : \{0,1\}^* \rightarrow \{0,1\}^{|H|}$; $|H|$ denotes the output bit length of H. $\mathcal{A}^{\mathcal{O}_0,\ldots,\mathcal{O}_i}(\cdot)$ denotes algorithm $\mathcal{A}$ is provided with oracles $\mathcal{O}_0, \ldots, \mathcal{O}_i$. For example, $\mathcal{A}^{Scheme.Sig_{sk}}(\cdot)$ denotes that algorithm $\mathcal{A}$ is provided with a *signing oracle* of signature scheme $Scheme$ under private key sk.

Definition 5.1 *A signature scheme SGN is a tuple of three algorithms (Kg, Sig, Ver) defined as follows:*

- $(sk, PK) \leftarrow SGN.Kg(1^\kappa)$*: Key generation algorithm takes the security parameter 1^κ as the input. It returns a private/public key pair (sk, PK) as the output.*

- $\sigma \leftarrow SGN.Sig(sk, D)$*: The signature generation algorithm takes sk and a data item D as the input. It returns a signature σ as the output (also denoted as $\sigma \leftarrow SGN.Sig_{sk}(D)$).*

- $c \leftarrow SGN.Ver(PK, D, \sigma)$*: The signature verification algorithm takes PK, D and σ as the input. It outputs a bit c, with $c = 1$ meaning* valid *and $c = 0$ meaning* invalid*.*

Definition 5.2 *Existential Unforgeability under Chosen Message Attack (EU-CMA) experiment for SGN is as follows:*
Experiment $Expt_{SGN}^{EU\text{-}CMA}(\mathcal{A})$

$(sk, PK) \leftarrow SGN.Kg(1^\kappa)$,

$(D^*, \sigma^*) \leftarrow \mathcal{A}^{SGN.Sig_{sk}(\cdot)}(PK)$,

If $SGN.Ver(PK, D^*, \sigma^*) = 1$ *and* D^* *was not queried, return* 1*, else, return* 0*.*

EU-CMA-advantage *of* $\mathcal{A}$ *is* $Adv_{SGN}^{EU\text{-}CMA}(\mathcal{A}) = Pr[Expt_{SGN}^{EU\text{-}CMA}(\mathcal{A}) = 1]$.

EU-CMA-advantage *of* SGN *is* $Adv_{SGN}^{EU\text{-}CMA}(t, L, \mu) = \max_{\mathcal{A}}\{Adv_{SGN}^{EU\text{-}CMA}(\mathcal{A})\}$*, where the maximum is over all* $\mathcal{A}$ *having time complexity* t*, making at most* L *oracle queries, and the sum of lengths of these queries being at most* μ *bits.*

LogFAS is built on the Schnorr signature scheme [105]. It also uses an Incremental Hash function $\mathcal{IH}$ [11] and a generic signature scheme SGN (e.g., Schnorr) as building blocks. Both Schnorr and $\mathcal{IH}$ require that $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$ is a random oracle.

Definition 5.3 *The Schnorr signature scheme is a tuple of three algorithms* (Kg, Sig, Ver) *behaving as follows:*

- $(y, \langle p, q, \alpha, Y\rangle) \leftarrow Sch.Kg(1^\kappa)$*: Key generation algorithm takes* 1^κ *as the input. It generates large primes* q *and* $p > q$ *such that* $q|(p-1)$*, and then generates a generator* α *of the subgroup* G *of order* q *in* $\mathbb{Z}_p^*$*. It also generates* $(y \xleftarrow{\$} \mathbb{Z}_q^*,\ Y \leftarrow \alpha^y \bmod p)$*, and returns private/public keys* $(y, \langle p, q, \alpha, Y\rangle)$ *as the output.*

- $(s, R, e) \leftarrow Sch.Sig(y, D)$*: Signature generation algorithm takes private key* y *and a data item* D *as the input. It returns a signature triplet* (s, R, e) *as follows:*

  - $R \leftarrow \alpha^r \bmod p$*,*
  - $e \leftarrow H(D||R)$*,*
  - $s \leftarrow (r - e \cdot y) \bmod q$*, where* $r \xleftarrow{\$} \mathbb{Z}_q^*$*.*

- $c \leftarrow Sch.Ver(\langle p, q, \alpha, Y\rangle, D, \langle s, R, e\rangle)$*: Signature verification algorithm takes public key* $\langle p, q, \alpha, Y\rangle$*, data item* D *and signature* $\langle s, R, e\rangle$ *as the input. It returns a bit* c*, with* $c = 1$ *meaning* valid *if* $R \equiv Y^e \alpha^s \bmod p$*, and with* $c = 0$ *otherwise.*

Definition 5.4 *Given a large random integer* q *and integer* L*, incremental hash function family* $\mathcal{IH}$ *is defined as follows: Given a random key* $z = (z_0, \ldots, z_{L-1})$*, where* $(z_0, \ldots, z_{L-1}) \xleftarrow{\$} \mathbb{Z}_q^*$ *and hash function* H*, the associated incremental hash function* $\mathcal{IH}_z^{q,L}$ *takes an arbitrary data item set* $D_0, \ldots, D_{L-1}$ *as the input. It returns an integer* $T \in Z_q$ *as the output,*

Algorithm $\mathcal{IH}_z^{q,L}(D_0, \ldots, D_{L-1})$

$T \leftarrow \sum_{j=0}^{L-1} H(D_j) z_j \bmod q$*, return* T*.*

Target Collision Resistance (TCR) [15] of $\mathcal{IH}$ relies on the intractability of *Weighted Sum of Subset (WSS) problem* [11, 61] assuming that H is a random oracle.

Definition 5.5 *Given $\mathcal{IH}_z^{q,L}$, let $\mathcal{A}_0$ be an algorithm that returns a set of target messages, and $\mathcal{A}_1$ be an algorithm that returns a bit. Consider the following experiment:*

Experiment $Expt^{TCR}_{\mathcal{IH}_z^{q,L}}(\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1))$

$(D_0, \ldots, D_{L-1}) \leftarrow \mathcal{A}_0(L)$,

$z = (z_0, \ldots, z_{L-1}) \xleftarrow{\$} \mathbb{Z}_q^*$,

$T \leftarrow \mathcal{IH}_z^{q,L}(D_0, \ldots, D_{L-1})$,

$(D_0^*, \ldots, D_{L-1}^*) \leftarrow \mathcal{A}_1(D_0, \ldots, D_{L-1}, T, \mathcal{IH}_z^{q,L})$,

If $T = \mathcal{IH}_z^{q,L}(D_0^*, \ldots, D_{L-1}^*) \wedge \exists j \in \{0, \ldots, L-1\} : D_j^* \neq D_j$, *return* 1, *else, return* 0.

TCR-advantage *of* $\mathcal{A}$ *is* $Adv^{TCR}_{\mathcal{IH}_z^{q,L}}(\mathcal{A}) = Pr[Expt^{TCR}_{\mathcal{IH}_z^{q,L}}(\mathcal{A}) = 1]$.

TCR-advantage *of* $\mathcal{IH}_z^{q,L}$ *is* $Adv^{TCR}_{\mathcal{IH}_z^{q,L}}(t) = \max_{\mathcal{A}}\{Adv^{TCR}_{\mathcal{IH}_z^{q,L}}(\mathcal{A})\}$, *where the maximum is over all* $\mathcal{A}$ *having time complexity* t.

5.2 Syntax and Models

LogFAS is a Forward-secure and Append-only Signature (FSA) scheme, which combines *key-evolve* (e.g., [8, 68]) and *signature aggregation* (e.g., [26, 90]) techniques. Specifically, LogFAS is built on the Schnorr signature scheme [100, 105], and it integrates forward-security and signature aggregation strategies in a novel and efficient way. That is, different from previous approaches (e.g., [75, 76, 78, 79, 104, 120, 121]), LogFAS introduces verification with a constant number of ExpOps, selective subset verification and sub-linear search properties via incremental hashing [11] and masked tokens (inspired from [92]) in addition to the above strategies.

Before giving more details, we briefly discuss the *append-only* signatures. A forward-secure and aggregate signature scheme is an *append-only* signature scheme if no message can be re-ordered or selectively deleted from a given stream of messages, while new messages can be appended to the stream [76, 79]. In Section 5.4, we prove that LogFAS is an append-only signature scheme. Note that, for the envisioned applications, the goal of signature aggregation is to achieve verification with a constant number of ExpOps and append-only property rather than storage efficiency.

Definition 5.6 *A FSA is comprised of a tuple of three algorithms* $(Kg, FASig, FAVer)$ *behaving as follows:*

- $(sk, PK) \leftarrow FSA.Kg(1^\kappa, L)$*: The key generation algorithm takes the security parameter* 1^κ *and the maximum number of key updates* L *as the input. It returns a private/public key pair* (sk, PK) *as the output.*

- $(sk_{j+1}, \sigma_{0,j}) \leftarrow FSA.FASig(sk_j, D_j, \sigma_{0,j-1})$*: The forward-secure and append-only signing algorithm takes the current private key* sk_j*, a new message* D_j *to be signed and the append-only signature* $\sigma_{0,j-1}$ *on the previously signed messages* $(D_0, \ldots, D_{j-1})$ *as the input. It computes an append-only*

signature $\sigma_{0,j}$ on $(D_0, \ldots, D_j)$, evolves (updates) sk_j to sk_{j+1}, and returns $(sk_{j+1}, \sigma_{0,j})$ as the output.

- $c \leftarrow FSA.FAVer(PK, \langle D_0, \ldots, D_j \rangle, \sigma_{0,j})$*: The forward-secure and append-only verification algorithm takes PK, $\langle D_0, \ldots, D_j \rangle$ and their corresponding $\sigma_{0,j}$ as the input. It returns a bit c, with $c = 1$ meaning* valid*, and $c = 0$ otherwise.*

In LogFAS, private key sk is a vector, whose elements are comprised of specially constructed Schnorr private keys and a set of tokens. These tokens later become the part of append-only signature σ accordingly. The public key PK is a system-wide public key that is shared by all verifiers, and is comprised of two long-term public keys. Details are given in Section 5.3.

5.2.1 System Model

LogFAS system model is comprised of a *Key Generation Center (KGC)* and multiple signers (i.e., logging machines that could be compromised) and verifiers. As in forward-secure stream integrity model (e.g., [18, 75, 76]), signers honestly execute the scheme until they are compromised by the adversary. Verifiers may be *untrusted*.

The KGC executes $LogFAS.Kg$ once offline before the deployment, and distributes a distinct private key/token set (auxiliary signature) to each signer, and two long-term public keys to all verifiers. After the deployment, a signer computes the forward-secure and append-only signature of log entries with $LogFAS.FASig$, and verifiers can verify the signature of any signer with $LogFAS.FAVer$ via two public keys without communicating with KGC (constant storage overhead at the verifier side).

In LogFAS, the same logger computes the append-only signature of her own log entries. Note that this form of signature computation is ideal for the envisioned secure audit logging applications, since each logger is only responsible for her own log entries [78, 79, 120, 121]. Signature schemes where the signatures of different signers are aggregated (e.g., [26, 90]) is out of the scope of this dissertation.

5.2.2 Security Model

A FSA scheme is proven to be *ForWard-secure Existentially Unforgeable against Chosen Message Attack (FWEU-CMA)* based on the experiment defined in Definition 5.7. In this experiment, $\mathcal{A}$ is provided with two types of oracles that she can query up to L messages in total as follows:

$\mathcal{A}$ is first provided with a *batch signing oracle* $FAS.Sig_{sk}(.)$. For each batch query j, $\mathcal{A}$ queries $FAS.Sig_{sk}(.)$ on a set of message $\overrightarrow{D}_j$ of her choice once. $FAS.Sig_{sk}(.)$ returns a forward-secure and append-only signature $\sigma_{0,j}$ under sk by aggregating σ_j (i.e., the current append-only signature) on $\overrightarrow{D}_j$ with the previous signature $\sigma_{0,j-1}$ on $\overrightarrow{D}_0, \ldots, \overrightarrow{D}_{j-1}$ that $\mathcal{A}$ queried. Assume that $\mathcal{A}$ makes i batch queries (with $0 \leq l \leq L$ individual messages) as described the above until she decides to "break-in".

$\mathcal{A}$ then queries the $Break\text{-}in$ oracle, which returns the remaining $L - l$ private keys to $\mathcal{A}$ (if $l = L$ $Break\text{-}in$ rejects the query).

Definition 5.7 FWEU-CMA experiment *is defined as follows:*
Experiment $Expt_{FSA}^{FWEU\text{-}CMA}(\mathcal{A})$
$(sk, PK) \leftarrow FSA.Kg(1^{\kappa}, L)$,
$(\overrightarrow{D}^*, \sigma^*) \leftarrow \mathcal{A}^{FAS.Sig_{sk}(.), Break\text{-}in}(PK)$,
If $FSA.FAVer(PK, \overrightarrow{D}^*, \sigma^*) = 1 \wedge \forall I \subseteq \{0, \ldots, l\}, \overrightarrow{D}^* \neq ||_{k \in I} \overrightarrow{D}_k$, *return* 1, *else, return* 0.

FWEU-CMA-advantage of $\mathcal{A}$ *is* $Adv_{FSA}^{FWEU\text{-}CMA}(\mathcal{A}) = Pr[Expt_{FSA}^{FWEU\text{-}CMA}(\mathcal{A}) = 1]$.
FWEU-CMA-advantage of FSA *is* $Adv_{FSA}^{FWEU\text{-}CMA}(t, L, \mu) = \max_{\mathcal{A}}\{Adv_{FSA}^{FWEU\text{-}CMA}(\mathcal{A})\}$, *where the maximum is over all* $\mathcal{A}$ *having time complexity* t, *making at most* L *oracle queries, and the sum of lengths of these queries being at most* μ *bits.*

Note that the winning condition of $\mathcal{A}$ also subsumes the truncation attack in addition to data modification. That is, $\mathcal{A}$ wins the experiment when she either modifies a data item or keeps data items intact but outputs a valid signature on a subset of a given batch query (i.e., she splits an append-only signature without knowing its individual signatures).

The above experiment does not implement a random oracle for $\mathcal{A}$ explicitly. However, we still assume the ROM [13], since Schnorr signature scheme [105] on which LogFAS is built requires the ROM.

Remark 5.1 LogFAS uses SGN to *prevent truncation attacks* by computing signatures of counter values. Resilience against the traditional data forgery (without truncation) relies on EU-CMA property of Sch and target collision-freeness of $\mathcal{IH}$. In Theorem 5.1, we prove that a successful truncation attack against LogFAS is equivalent to break SGN, and a successful data modification (including re-ordering) against LogFAS is equivalent to break Sch and/or $\mathcal{IH}$.

5.3 LogFAS Schemes

In this section, we first present the intuition and detailed description of LogFAS, and then describe a LogFAS variation that has additional capabilities.

5.3.1 LogFAS Scheme

All the previous forward-secure and aggregate (append-only) constructions [75, 76, 78, 79, 120] rely on a direct combination of an aggregate signature (e.g., [26]) and a forward-secure signature (e.g., [2, 68]) (with the exception of our proposed schemes HaSAFSS and BAF presented in Chapter 3 and Chapter 4). Therefore, the resulting constructions simultaneously inherit all overheads of their base primitives: (i) Forward-secure signatures on individual data items, which are done separately from the append-only

design, force verifiers to perform $O(l)$ ExpOps. (ii) These schemes either eliminate ExpOps from the logging phase with pre-computation but incur quadratic storage overhead to the verifiers (e.g., [120]), or require ExpOps in the logging phase for each log entry and incur linear storage overhead to the verifiers (e.g., [57, 75, 79]).

The above observations inspired us to design cryptographic mechanisms that can verify *the integrity of entire log entry set once directly* (preserving forward-security), instead of checking the integrity of each data item individually, though the signing operations have to be performed on individual data items. That is, instead of verifying each item one-by-one with the corresponding public key(s), verify all of them via a *single set of aggregated cryptographic components* (e.g., tokens as auxiliary signatures). These mechanisms also achieve constant storage overhead at the verifier side.

We achieve these goals with a provable security by using Schnorr signature and incremental hash $\mathcal{IH}$ as follows:

a) To compute a forward-secure and append-only Schnorr signature, we aggregate each individual signature s_l on D_l with the previous aggregate signature as $s_{0,l} \leftarrow s_{0,l-1} + s_l \bmod q$, $(0 < l \leq L-1, s_{0,0} = s_0)$. This is done by using a distinct private key pair (r_j, y_j) for $j = 0, \ldots, L-1$ on each data item.

b) Despite being forward-secure, the above construction still requires an ExpOp for each data item. To verify the signature on $D_0, \ldots, D_l$ with only $O(1)$ ExpOp, we introduce the notion of *token*.

In LogFAS, each Schnorr private y_j is comprised of a random key pair (a_j, d_j) for $j = 0, \ldots, L-1$. Random key a_j is mutually blinded with another random factor x_j and also a long-term private key b for $j = 0, \ldots, L-1$. The result of these blinding operations is called *auxiliary signature* (token) z_j, which can be kept publicly without revealing information about (a_j, x_j) and also can be authenticated with the long-term public key B by all verifiers. Furthermore, these masked tokens $z = z_0, \ldots, z_l$ also serve as a one-time initialization key for the incremental hash as $\mathcal{IH}_z^{q,l}$ (Definition 5.4), which enable verifiers to reduce the integrity of each D_j into the integrity of a final tag $z_{0,l}$. This operation preserves the integrity of each D_j and verifiability of each z_j (via public key B) without ExpOps.

c) To verify $(s_{0,l}, z_{0,l})$ via B in an aggregate form, verifiers also aggregate tokens R_j as $R_{0,l} \leftarrow \prod_{j=0}^{l} R_j \bmod p$, where p a large prime on which the group was constructed. However, initially, $(s_{0,l}, R_{0,l}, z_{0,l})$ cannot be verified directly via B, since the reduction operations introduce some extra verification information. LogFAS handles this via *auxiliary signature* (token) $M'_{0,l}$ that bridges $(s_{0,l}, R_{0,l}, z_{0,l})$ to B. That is, the signer computes an aggregate token $M'_{0,l} \leftarrow M'_{0,l-1} M_l^{e_j} \bmod p$, where $0 < l \leq L-1$ and $M_{0,0} = M_0$), along with $s_{0,l}$ in the signing process. During verification, this aggregate token eliminates the extra terms and bridges $(s_{0,l}, R_{0,l}, z_{0,l})$ with B.

This approach allows LogFAS to compute publicly verifiable signatures with only one ExpOp per-item, and this signature can be verified with only $O(1)$ ExpOps by storing only two public keys at the verifier side (regardless of the number of signers). This is much more efficient than all of its PKC-based counterparts, and also is as efficient as the symmetric schemes at the verifier side.

The detailed description of LogFAS algorithms is given below:

1) *LogFAS.Kg*$(1^\kappa, L)$: Given 1^κ, generate primes q and $p > q$ such that $q|(p-1)$, and then generate a generator α of the subgroup G of order q in $\mathbb{Z}_p^*$.

a) Generate $(b \xleftarrow{\$} \mathbb{Z}_q^*, B \leftarrow \alpha^{b^{-1}} \bmod p)$ and $(\widehat{sk}, \widehat{PK}) \leftarrow SGN.Kg(1^\kappa)$. *System-wide private key* of KGC is $\overline{sk} \leftarrow (b, \widehat{sk})$. *System-wide public key* of all verifiers is $PK \leftarrow \{p, q, \alpha, B, \widehat{PK}, L\}$.

b) Generate $(r_j, a_j, d_j, x_j) \xleftarrow{\$} \mathbb{Z}_q^*$ for $j = 0, \ldots, L-1$. The private key of signer ID_i is $sk \leftarrow \{r_j, y_j, z_j, M_j, R_j, \beta_j\}_{j=0}^{L-1}$, where

- $y_j \leftarrow a_j - d_j \bmod q$, $z_j \leftarrow (a_j - x_j)b \bmod q$,
- $R_j \leftarrow \alpha^{r_j} \bmod p$, $M_j \leftarrow \alpha^{x_j - d_j} \bmod p$,
- $\beta_j \leftarrow SGN.Sig(\widehat{sk}, H(ID_i||j))$. Remark that each β_j is initially used kept secret initially, and then then released as a part of signature publicly when they are needed.

2) *LogFAS.FASig*$(\langle r_l, y_l, z_l, M_l, R_l, \beta_l \rangle, D_l, \sigma_{0,l-1})$: Given $\sigma_{0,l-1}$ on $D_0, \ldots, D_{l-1}$, compute $\sigma_{0,l}$ on $D_0, \ldots, D_l$ as follows,

a) $e_l \leftarrow H(D_l||l||z_l||R_l)$, $M_l' \leftarrow M_l^{e_l} \bmod p$, $s_l \leftarrow r_l - e_l y_l \bmod q$,

b) $s_{0,l} \leftarrow s_{0,l-1} + s_l \bmod q$, $(0 < l \leq L-1,\ s_{0,0} = s_0)$,

c) $M'_{0,l} \leftarrow M'_{0,l-1} M_l' \bmod p$, $(0 < l \leq L-1,\ M'_{0,0} = M_0)$,

d) $\sigma_{0,l} \leftarrow \{s_{0,l}, M'_{0,l}, \beta_l, R_j, e_j, z_j\}_{j=0}^{l}$ and erase $(r_l, y_l, s_{0,l-1}, s_l, \beta_{l-1})$.

3) *LogFAS.FAVer*$(PK, \langle D_0, \ldots, D_l \rangle, \sigma_{0,l})$:

a) If $SGN.Ver(\widehat{PK}, H(ID_i||l), \beta_l) = 0$ then return 0, else continue,

b) If $\prod_{j=0}^{l} R_j \bmod p \equiv M'_{0,l} \cdot B^{z_{0,l}} \cdot \alpha^{s_{0,l}} \bmod p$ holds return 1, else return 0, where $z_{0,l} = \mathcal{IH}_{z_0,\ldots,z_l}^{q,l}(D_0 ||w||z_0||R_0, \ldots, D_l||w||z_l||R_l)$.

5.3.2 Selective Verification with LogFAS

All the previous forward-secure and aggregate constructions (e.g., [75, 76, 78, 121]) verify the set of log entries via only the final aggregate signature to prevent the truncation attack and save the storage. However, this approach causes performance drawbacks: (i) The verification of any subset of log entries requires the verification of the entire set of log entries (i.e., always $O(L)$ ExpOps for the subset verification). (ii) The failure of signature verification does not give any information about which log entries were corrupted.

Ma et al. proposed immutable-FssAgg (iFssAgg) schemes in [79] to allow fine-grained verification without being vulnerable to truncation attacks. However, iFssAgg schemes double the signing/verifying costs of their base schemes. In addition, even if the signature verification fails due to only a few corrupted log entries (i.e., accidentally damaged entry(ies)), detecting which log entry(ies) is (are) responsible for the failure requires verifying each individual signature.

LogFAS can address the above problems via a simple variation without incurring any additional costs: The signer keeps *all* signatures and tokens in their individual forms (including s_j for $j = 0, \ldots, l$) without aggregation. The verifiers can aggregate them according to their needs by preserving the security and verifiability. This offers performance advantages over iFssAgg schemes [79]:

(i) LogFAS protects the number of log entries via pre-computed tokens $\beta_0, \ldots, \beta_l$, and therefore individual signatures can be kept without a truncation risk. This eliminates the necessity of costly immutability strategies used in iFssAgg schemes [79]. Furthermore, a verifier can selectively aggregate any subset of $l' < l$ log entries and verify them by performing only $O(1)$ ExpOps as in the original LogFAS. This is much more efficient than the iFssAgg schemes, which require $O(2l')$ ExpOps.

(ii) LogFAS can use a recursive subset search strategy to identify corrupted log entries causing the verification failure faster than linear search[1]. That is, the log entry set is divided into subsets along with their corresponding individual signatures. Each subset is then independently verified by $LogFAS.AVer$ via its corresponding aggregate signature, which is efficiently computed from individual signatures. Subsets returning 1 are eliminated from the search, while each subset returning 0 is again divided into subsets and verified by $LogFAS.AVer$ as described. This subset search continues recursively until all the corrupted log entries are identified.

The above strategy can quickly identify the corrupted entries when most log entries are intact. For instance, if only one entry is corrupted, it can identify the corrupted entry by performing $(2\log_2 l)$ ExpOps + $O(l)$ hash operations. This is much faster than linear search used in the previous PKC-based schemes, which always requires $O(l)$ ExpOps + $O(l)$ hash operations.

Recursive subset strategy remains more efficient than linear search as long as the number of corrupted entries c satisfies $c \leq \frac{l}{2\log_2 l}$. When $c > \frac{l}{2\log_2 l}$, depending on c and the distribution of corrupted entries, recursive subset search might be more costly than linear search. To minimize the performance loss in such an inefficient case, the verifier can switch from recursive subset search to the linear search if the recursive division and search step continuously returns 0 for each verified subset. The verifier can ensure that the performance loss due to an inefficient case does not exceed the average gain of an efficient case by setting the maximum number of recursive steps to be executed to $l'/2 - \log_2 l'$ for each subset with l' entries.

5.4 Security Analysis

We prove that LogFAS is a *FWEU-CMA* signature scheme in Theorem 5.1 below.

Theorem 5.1 *$Adv^{FWEU\text{-}CMA}_{LogFAS}(t, L, \mu)$ is bounded as follows,*

$$Adv^{FWEU\text{-}CMA}_{LogFAS}(t, L, \mu) \leq L \cdot Adv^{EU\text{-}CMA}_{Sch}(t, L, \mu) + Adv^{EU\text{-}CMA}_{SGN}(t'', L, \mu'') + Adv^{TCR}_{\mathcal{IH}^{q,L}_z}(t'''),$$

[1]Note that the previous PKC-based audit logging schemes *cannot* use such a recursive subset search strategy to identify corrupted log entries with a sub-linear number ExpOps, since they always require linear number of ExpOps to verify a given subset from the entire log entry set (in contrast to LogFAS that requires $O(1)$ExpOp to verify a given subset).

where $t' = O(t) + L \cdot O(\kappa^3)$ *and* $\mu' = \mu/L$.

Proof: Let $\mathcal{A}$ be a LogFAS attacker. We construct a *Schnorr* attacker $\mathcal{F}$ that uses $\mathcal{A}$ as a sub-routine as follows:

Algorithm $F^{Sch.Sig_y(\cdot)}(Y)$

Set the target forgery index $w \xleftarrow{\$} [0, L-1]$,

$(sk, PK) \leftarrow LogFAS.Kg(1^\kappa, L)$, where $sk = \{\langle b, \overline{sk}\rangle, \langle ID_i : r_j, y_j, z_j, M_j, R_j, \beta_j\rangle\}_{j=0}^{L-1}$ and $PK = (p, q, \alpha, B, \widehat{PK}, L)$,

To embed Schnorr public key Y into token M_w, simulate tokens (M_w, z_w) as follows:

-$(\gamma, \gamma') \xleftarrow{\$} \mathbb{Z}_q^*,\ M_w \leftarrow Y \cdot \alpha^{(-\gamma+\gamma' b^{-1})} \bmod p,\ z_w \leftarrow \gamma \cdot b - \gamma' \bmod q$,

Execute $\mathcal{A}^{FAS.Sig_{sk}(.), Break\text{-}in}(PK)$ as follows:

- $l \leftarrow 0,\ j' \leftarrow 0,\ i \leftarrow 0$,
- Queries: $\mathcal{A}$ first queries $FAS.Sig_{sk}(.)$ and then $Break\text{-}in$ oracles up to L messages of her choice in total:
  - How to respond i-th $FAS.Sig_{sk}(.)$ query:
    - For each query i, $\mathcal{A}$ queries $FAS.Sig_{sk}(.)$ on $\overrightarrow{D}_i = \{D_{j'}, \ldots, D_j\},\ j > j'$ of her choice. If $j + 1 > L$ then reject the query and proceed to the *Forgery phase* ($\mathcal{A}$ exceeds her query limit). Otherwise, continue to the next step,
    - If $j' \leq w \leq j$ then $FAS.Sig_{sk}(.)$ goes to the Schnorr oracle on D_w as $(s_w, R_w, e_w) \leftarrow Sch.Sig_y(D_w||w||z_w)$. $FAS.Sig_{sk}(.)$ then computes $s_{j',j} \leftarrow \sum_{j' \leq k \leq j, k \neq w} (r_k - e_k y_k) + s_w \bmod q$, where $e_k \leftarrow H(D_k||k||z_k||R_k)$ for $k = j', \ldots, j$. Also set variable $D' \leftarrow D_w$. Otherwise, compute $s_{j',j} \leftarrow \sum_{k=j'}^{j} (r_k - e_k y_k) \bmod q$, where $e_k \leftarrow H(D_k||k||z_k||R_k)$ for $k = j', \ldots, j$,
    - $M'_{j',j} \leftarrow \prod_{k=j'}^{j} M_k^{e_k} \bmod p$,
    - $s_{0,j} \leftarrow s_{0,j'-1} + s_{j',j} \bmod q$ and $M'_{0,j} \leftarrow M'_{0,j'-1} M'_{j',j} \bmod p$, where $(s_{0,j'-1}, M'_{0,j'-1})$ were computed on $\mathcal{A}$'s previous queries $D_0, \ldots, D_{j'-1}$, (for initial $j' = 0$, $s_{0,j'-1} = 0, M_{0,j'-1} = 1$),
    - Response i-th query of $\mathcal{A}$ as $\sigma_{0,j} \leftarrow \{s_{0,j}, M'_{0,j}, \beta_j, R_k, e_k, z_k\}_{k=j'}^{j}$,
    - $\mathcal{F}$ maintains four lists and a variable z' in addition to D' for bookkeeping purposes. Insert i-th query $\overrightarrow{D}_i$ into data list $\mathcal{LD}[i]$. Insert signature results $(s_{0,j}, M_{0,j}, \beta_j)$ into the lists $(\mathcal{LS}1, \mathcal{LS}2, \mathcal{LS}3)$, respectively. Also update variable z' as $z' \leftarrow \mathcal{IH}_z^{q,j}(e_0, \ldots, e_j)$ for $z = z_0, \ldots, z_j$,
    - If $\mathcal{A}$ decides to the "break-in", proceed to the next step. Otherwise, update $j' \leftarrow j+1,\ l \leftarrow j',\ i \leftarrow i+1$ and continue to respond her queries,

• How to respond queries to the *Break-in* oracle: $\mathcal{A}$ queried l individual messages to $FAS.Sig_{sk}(.)$ oracle up to now. If $l = L$ then reject the query (all private keys were used and erased) and proceed to the next step. Otherwise, if $l < w$ then *abort* and return 0 ($\mathcal{F}$ does not know the Schnorr private key corresponding index w). Otherwise, supply $\mathcal{A}$ with $\{r_j, y_j, z_j, M_j, R_j, \beta_j\}_{j=l+1}^{L-1}$.

- Forgery: Finally, $\mathcal{A}$ outputs a forgery as $(\langle D_0^*, \ldots, D_k^* \rangle, \sigma^*)$, where $\sigma^* = \{s_{0,k}^*, M_{0,k}^*, \beta^*, R_j^*, e_j^*, z_j^*\}_{j=0}^{k}$.

By Definition 5.7, $\mathcal{A}$ wins if the below condition holds:

1. $LogFAS.FAVer(PK, \langle D_0^*, \ldots, D_k^* \rangle, \sigma^*) = 1$

2. $\forall I \subseteq \{0, \ldots, i\}, \{D_0^*, \ldots, D_k^*\} \neq ||_{m \in I} \mathcal{LD}[m]$,

If one of the above conditions fails, $\mathcal{A}$ loses in *FWEU-CMA* experiment, and therefore $\mathcal{F}$ *aborts* and returns 0. Otherwise, $\mathcal{F}$ proceeds according to one of the below cases, which are implied by condition 2 as follows:

a) $\exists j \in \{0, \ldots, k\} : (D_j^* \notin \{\mathcal{LD}[0], \ldots, \mathcal{LD}[i]\} \wedge k = L - 1)$

b) $\exists j \in \{0, \ldots, i\} : (\{D_0^*, \ldots, D_k^*\} \subset \mathcal{LD}[j] \ \wedge \ \beta^* \notin \mathcal{LS}3)$

c) $\exists j \in \{0, \ldots, k\} : (D_j^* \notin \{\mathcal{LD}[0], \ldots, \mathcal{LD}[i]\} \ \wedge \ z' = \mathcal{IH}_z^{q,k}(D_0^*, \ldots, D_k^*))$

Case a): This case implies $\mathcal{A}$ modifies at least one data item (without truncation). $\mathcal{F}$ checks if $D_w^* \neq D'$ (i.e., whether one of $\mathcal{A}$'s forgery is on D', whose corresponding token includes Schnorr public key Y that $\mathcal{F}$ embedded). If it fails, $\mathcal{F}$ *aborts* and return 0. Otherwise, by Definition 5.2, $\mathcal{F}$ wins the *EU-CMA* experiment and returns 1, since the below conditions hold:

$Sch.Ver(\langle p, q, \alpha, Y \rangle, D_w^*, \langle s_w^*, R_w^*, e_w^* \rangle) = 1$ and $\mathcal{F}$ did not ask D_w^* to the Schnorr oracle, where $s_w^* \leftarrow s_{0,L-1}^* - \sum_{0 \leq m \leq L-1, m \neq w}^{L-1} (r_m - e_m^* y_m) \bmod q$ and $e_m^* \leftarrow H(D_m^* || m || z_m^* || R_m^*)$ for $m = 0, \ldots, L-1$.

Since the target forgery index is randomly chosen as $w \xleftarrow{\$} [0, L-1]$, if $\mathcal{A}$ wins the experiment based on this case with the probability $Adv_{LogFAS}^{FWEU\text{-}CMA}(t, L, \mu)$, then $\mathcal{F}$ wins with the probability $Adv_{LogFAS}^{FWEU\text{-}CMA}(t, L, \mu)/L$.

The running time of $\mathcal{F}$ is that of $\mathcal{A}$ plus the overhead due to handling $\mathcal{A}$'s queries as $t' = O(t) + L \cdot O(\kappa^3)$, where $O(\kappa^3)$ denotes the execution time of modular exponentiation operation in Z_p^* for given κ.

Case b): This case implies a successful tail-truncation attack. If it holds, then by Definition 5.2, $\mathcal{A}$ breaks SGN since β^* is valid and it was not queried. This happens with probability $Adv_{SGN}^{EU\text{-}CMA}(t'', L, \mu'')$.

Case c): If this case holds, then by Definition 5.5, $\mathcal{A}$ breaks $\mathcal{IH}$ by finding a target collision. This happens with probability $Adv_{\mathcal{IH}_z^{q,L}}^{TCR}(t''')$.

Remark that in the above experiment, the simulated view of $\mathcal{A}$ is *perfectly indistinguishable* from the real view of $\mathcal{A}$: The real view of $\mathcal{A}$ after L queries ($0 \leq l \leq L$ queries to the $FAS.Sig_{sk}(.)$ oracle and $L - l$ queries to the *Break-in* oracle) is $\overrightarrow{A}_{Real} = \{PK, \mathcal{LS}1, \mathcal{LS}2, \mathcal{LS}3, e_i, R_j, z_j, M_{l+1},$

$\ldots, M_{L-1}\}_{i=0,j=0}^{l,L-1}$, where all keys/tokens/signatures are computed/generated via original LogFAS algorithms. The simulated view of $\mathcal{A}$ after L queries is equivalent to $\overrightarrow{A}_{Real}$ except that (M_w, z_w) are simulated as described. One may verify that the joint probability distribution of these views are identical as $Pr[\overrightarrow{A}_{Real} = \overrightarrow{a}] = Pr[\overrightarrow{A}_{Sim} = \overrightarrow{a}]$. □

Remark 5.2 Another security concern in audit logging is *delayed detection* identified in [78]. In delayed detection, log verifiers cannot detect whether the log entries are modified until an online TTP provides auxiliary keying information to them (the details of this attack were also discussed in Chapter 4). LogFAS does not rely on an online TTP support or time factor to achieve the signature verification, and therefore it is not prone to delayed detection.

5.5 Performance Analysis and Comparison

In this section, we present the performance analysis of LogFAS and compare it with previous schemes. We follow the notation in Table 5.2 in our analysis and comparison.

Computational Overhead: In LogFAS, the costs of signing a single item is $Exp + Mul + H$ including the key update cost. The cost of verifying l items is $2Exp + O(l)(3Mul + H)$. The key generation cost for L items is $O(L)(2Exp + Mul)$.

Table 5.3 and Table 5.4 compare the computational cost of LogFAS with its counterparts analytically and numerically, respectively.

From a verifier's perspective, LogFAS requires only a small and constant number of Exp operations regardless of the number of log entries to be verified. Therefore, it is much more efficient than all PKC-based schemes, which require one ExpOp per log entry. Besides, it does not double the verification cost to prevent the truncation attacks, providing further efficiency over iFssAgg schemes [79]. For instance, the verification of 10,000 log entries with LogFAS is 2650, 479, 1937, 1427 and 208 times faster than that of FssAgg-BLS, FssAgg-BM, FssAgg-AR, Logcrypt, and BAF, respectively. The verification of subsets from these entries with LogFAS is also much more efficient than all of its counterparts as shown in Table 5.4. The execution time differences with LogFAS and its PKC-based counterparts grow linearly with respect to the number of log entries to be verified. Initially, the symmetric schemes are more efficient than all PKC-based schemes, including ours. However, since the verification operations of LogFAS are dominated by H, their efficiency become comparable with symmetric schemes as the number of log entries increases (e.g., $l = 10^4$). From a logger's perspective, LogFAS is also more efficient than its PKC-based counterparts with the exception of BAF.

Figure 5.1 and Figure 5.2 further show the comparison of LogFAS and the previous schemes that allow public verification in terms of signature generation and verification time as the number of log entries increases. Similarly, Figure 5.3 and Figure 5.4 demonstrate the comparison of LogFAS with selective verification and iFssAgg schemes in terms of signature generation and verification time as the number

Table 5.2: Notation used in performance analysis and comparison of LogFAS and its counterparts

$GKg/GSig/GVer$: Generic key gen/sig/verifying ops.	Sqr: Squaring mod n'	H: Hashing
$Mul/Mulq'/Mulp'/Muln'$: Mul. mod p, q', p', n'	$Exp/Expp'$: Exp. mod p and p'	L: Max. # of key upd.
w and l:# of data items processed and will be processed	l': # data items to be processed in a subset	S: # of signers
$Add/Addq'$: Add. mod q and q', resp.	PR/Mtp: ECC map-to-point/pairing	z: FssAgg sec. param.

Suggested bit lengths to achieve 80-bit security for the above parameters are as follows for each compared scheme: Large primes ($|p| = 2048, |q| = 1600$) for LogFAS and Logcrypt, primes ($|p'| = 512, |q'| = 160$) for BAF and FssAgg-BLS, ($|n'| = 1024, z = 160$) for FssAgg-AR and FssAgg-BM, where n' is Blum-Williams integer [75].

Table 5.3: Computation involved in LogFAS and its counterparts

		Online		**Offline**
		ASig & Upd (per item)	**AVer** (l or l' entries)	**KG** (max. L)
PKC-based	*LogFAS*	$Exp + Mul + H$	$2Exp + O(l)(3Mul + H)$	$O(L)(2Exp + Mul)$
	FssAgg-BLS	$MtP + Expp' + Mulp'$	$l(Mulp' + H + PR)$	$O(L)(H + Expp')$
	FssAgg-BM	$\frac{z}{2}Muln' + z \cdot Sqr$	$O(L) \cdot Sqr + \frac{l \cdot z}{2}Muln'$	$O(L)(z \cdot Sqr + \frac{z}{2}Muln')$
	FssAgg-AR	$3z \cdot Sqr + \frac{z}{2}Muln'$	$z(L + l)Sqr + (2l + l \cdot z)Muln'$	$O(2L)(z \cdot Sqr + \frac{z}{2}Muln')$
	iFssAgg	$2 \cdot ASig + Upd$	$2 \cdot AVer(L, l')$	Kg
	Logcrypt	$GSig$	$O(l)GVer$	$O(L)GKg$
	BAF	$3H + Mulq' + 2Addq'$	$O(2l)(EMul + H)$	$O(2L)(H + EMul)$
Symmetric		$O(1)H$	$O(l)H$	$O(L)H$

LogFAS is the only scheme achieving verification with $O(1)ExpOp$ regardless of the value of (l, l'), while their counterparts require either $O(l)ExpOp$ (FssAgg, Logcrpyt and BAF) or $O(l')ExpOp$ (iFssAgg schemes). At the same time, LogFAS is as efficient as their counterparts at the signer side except the BAF.

of log entries increases. These figures demonstrate that LogFAS is the most verifier computationally efficient scheme among all these choices.

All PKC-based schemes require $O(L)$ ExpOps in the key generation phase.

Signature/Key/Data Storage and Transmission Overheads: LogFAS is a verifier storage friendly scheme; it requires each verifier to store only two public keys and an index along with system-wide parameters (e.g., $|q| + |4p|$), regardless of the number of signers or the number of log entries to be verified.

In LogFAS, the append-only signature size is $|q|$. The key/token and data storage overheads on the logger side are linear as (i.e., $O(L)(5|q|+2|p|)+O(l)|D|$) (assuming SGN is chosen as Schnorr [105]). LogFAS transmits a token set along with each data item requiring $O(l)(|q| + |p| + |D|)$ transmission in total. The fine-grain verification introduces $O(l')$ extra storage/communication overhead due to the individual signatures.

From a verifier's perspective, LogFAS is much more storage efficient than all existing schemes, which require either $O(L \cdot S)$ (e.g., FssAgg-BLS [76] and BAF [120]), or $O(S)$ (e.g., [17, 18, 57, 75, 79, 103, 104]) storage. From a logger's perspective, all the compared schemes both accumulate (stores) and transmit linear number of data items (i.e., $O(l)D$) until their verifiers become available to them. This dominates the main storage and communication overhead for these schemes. In addition to this, LogFAS

Table 5.4: Execution time (in ms) comparison of LogFAS and its counterparts

Criteria			*PKC-based*						*Sym.*
			LogFAS	FssAgg (l) / iFssAgg (l')			Logcrypt	BAF	
			($l = 10^4, l' < l$)	BLS / i	BM / i	AR / i			
Off.	*Kg,* $L = 10^4$		5.06×10^4	3.3×10^3	8.8×10^4	1.7×10^5	2.6×10^4	4×10^4	20
Onl.	*Sig&Upd (1)*		1.2	1.8 / 3.6	13.1 / 26.2	28 / 56	2.05	0.007	0.004
	Ver.	$l' = 10^2$	72.87	4.8×10^3	1.8×10^3	1.6×10^5	1.4×10^3	0.2×10^3	0.2
		$l' = 10^3$	75.2	4.8×10^4	1×10^4	1.8×10^5	1.5×10^4	2.05×10^3	2
		$l = 10^4$	98.12	2.6×10^5	4.7×10^4	1.9×10^5	1.4×10^5	2.04×10^4	19.9

(i) The execution times were measured on a computer with an Intel(R) Xeon(R)-E5450 3GHz CPU and 4GB RAM running Ubuntu 9.04. We tested LogFAS, BAF [120], FssAgg-BLS [76], Logcrypt (with DSA), and the symmetric schemes (e.g., [18, 76, 104]) using the MIRACL library [106], and FssAgg-AR/BM using the NTL library [108]. Parameter sizes determining the execution times of each scheme were selected s.t. $\kappa = 80$ (parameter sizes were discussed in Table 5.2).

(ii) The execution time of single exponentiation performed in LogFAS verification is more computationally expensive than that of LogFAS signature generation. This stems from the fact that the size of exponent in signature verification is larger than that of used in signature computation (e.g., 1600 bits vs. 160 bits). Note that despite the LogFAS uses larger exponents (and also performs a single SGN verification operation), it is significantly more efficient than all existing schemes at the verifier side (i.e., verification with $O(1)ExpOp$ property).

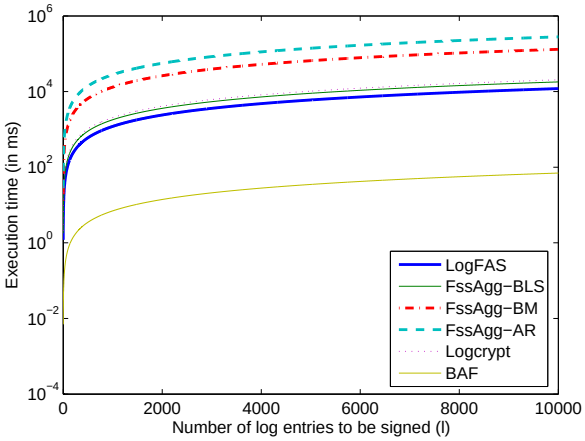

Figure 5.1: Signing time comparison of LogFAS and its counterparts (in ms)

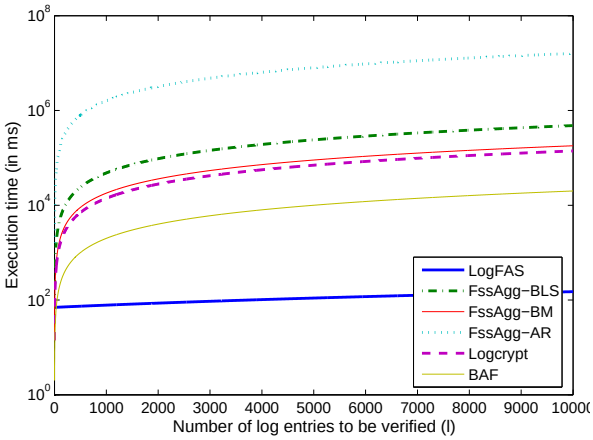

Figure 5.2: Verification time comparison of LogFAS and its counterparts (in ms)

requires linear key storage overhead at the logger side, which is slightly less efficient than [75, 76, 120]. LogFAS with fine-grained verification and its counterpart iFssAgg schemes [79] both require linear key/signature/data storage/transmission overhead.

Availability, Applicability and Security: The symmetric schemes [17, 18, 103, 104] are not publicly verifiable and also require online server support to verify log entries. Furthermore, they are vulnerable to both truncation and delayed detection attacks [78, 79] with the exception of FssAgg-MAC [76]. In contrast, PKC-based schemes [57, 75, 76, 78, 79] are publicly verifiable without requiring online server support, and they are secure against the truncation and delayed detection attacks, with the exception of Logcrypt [57].

LogFAS achieves all the desirable availability/applicability and security properties as well as being

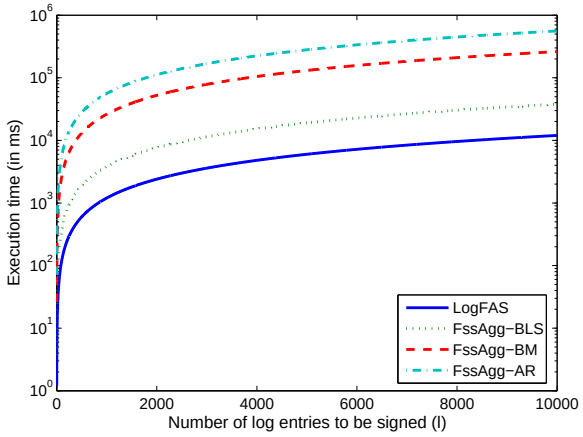

Figure 5.3: Signing time comparison of selective verification with LogFAS and iFssAgg (in ms)

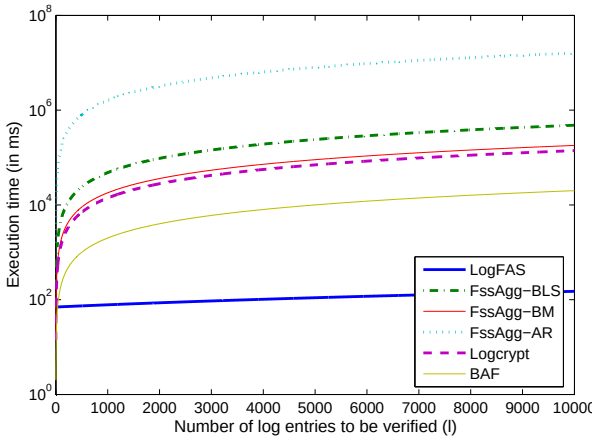

Figure 5.4: Verification time comparison of selective verification with LogFAS and iFssAgg (in ms)

significantly more efficient than PKC-based schemes.

5.6 Conclusion

In this chapter, we developed a new forward-secure and append-only audit logging scheme called LogFAS. LogFAS achieves the public verifiability without requiring any online trusted server support, and is secure against truncation and delayed detection attacks. LogFAS is much more computationally efficient than all existing PKC-based alternatives, with a performance comparable to symmetric schemes at the verifier side. LogFAS is also the most verifier storage efficient scheme among all existing alternatives. Last, a variation of LogFAS enables selective subset verification and efficient search of corrupted log entries. Overall, our comparison with the existing schemes show that LogFAS is an ideal choice for secure audit logging in verification intensive systems, where system auditors need to verify large number of log entries simultaneously produced by several loggers.

Chapter 6

Conclusion and Future Work

In this chapter, we conclude our dissertation and then briefly discuss our future work.

6.1 Conclusion

In current resource-constrained and/or task-intensive systems, protecting audit logs is a challenging task, especially in the presence of active adversaries. To protect audit logs effectively, an ideal cryptographic log protection mechanism must satisfy several properties simultaneously: (i) computational efficiency, (ii) applicability and flexibility (e.g., public verifiability), (iii) forward security (i.e., compromise resiliency), (iv) compactness (i.e., storage/communication efficiency), (v) provable security. Unfortunately, previous cryptographic secure logging solutions fail to meet most of these requirements.

In this dissertation, we developed a series of forward-secure and aggregate (append-only) cryptographic constructions that achieve all of the above desirable properties. We summarize our contributions as follows:

- *Efficient Secure Logging for UWSNs*: The lack of real-time communication, resource-constraints and active adversaries make secure logging a difficult task in UWSNs. We developed a series of cryptographic protocols called *Hash-Based Sequential Aggregate and Forward-Secure Signatures (HaSAFSS)* [121, 122] to address these challenges. HaSAFSS schemes utilize already existing verification delays in UWSNs to introduce an asymmetry between signers and verifiers via the TRE concept. HaSAFSS schemes and their properties are outlined below:
  - HaSAFSS schemes are the only alternative in which both signers and verifiers get equal benefits of high computational efficiency, while existing schemes incur heavy overhead on either the signer or the verifier side.
  - *Symmetric HaSAFSS (Sym-HaSAFSS*) is the most computation-efficient forward-secure and aggregate signature scheme of all existing alternatives. It is as efficient as symmetric schemes

both at the signer and the verifier sides and is also publicly verifiable. It also achieves the high verifier storage efficiency by requiring only small-constant storage at the verifier side. However, Sym-HaSAFSS requires a linear storage at the signer side.

- *Elliptic Curve Cryptography HaSAFSS (ECC-HaSAFSS)* requires only a small-constant storage for signers, while preserving the per-item computational efficiency of Sym-HaSAFSS. Therefore, ECC-HasAFSS offers a signer storage friendly alternative to Sym-HaSAFSS. However, ECC-HaSAFSS incurs quadratic storage overhead to the verifiers.
- Sym-HaSAFSS and ECC-HaSAFSS have two main limitations: (i) They do not allow signers to decide their own data delivery schedule after the deployment. (ii) A signer can only use a limited (pre-determined) number of time periods. This creates a heavy storage overhead either the signer side or the verifier side. It also requires the re-keying of sensor nodes when they deplete their keying material.

  Our extended scheme *Self-SUstaining HaSAFSS* (*SU-HaSAFSS*) in [122] addresses these limitations by introducing only a little more computational overhead. Note that SU-HaSAFSS preserves the per-data item computational efficiency of HaSAFSS schemes over the previous PKC-based alternatives. SU-HaSAFSS also achieves a flexible data delivery schedule, and it is also storage efficient both at the signer and at the verifier sides.

HaSAFSS schemes are ideal solutions for UWSNs; however, they cannot achieve the immediate verification property and also require a passive TTP support.

- *Efficient Secure Logging in Resource-Constrained Devices with Immediate Verification*: Publicly verifiable, compact and forward-secure audit logging in resource-constrained devices is a challenging task, since such devices cannot tolerate any ExpOp or heavy storage overhead. All existing PKC-based solutions (e.g., FssAgg schemes [75, 76, 79], LogFAS [124]) require ExpOps at the signer side, and also incur heavy storage overhead (e.g., Logcrypt [57]). This makes them impractical for secure logging in resource-constrained devices. While HaSAFSS schemes are computation-efficient, they cannot address real-time applications. Moreover, HaSAFSS schemes require passive TTP support from time to time, which might not be available for certain applications.

  To address these limitations, we developed a new class of forward-secure and aggregate cryptographic constructions called *Blind Aggregate Forward (BAF)* and *Fast-Immutable BAF*. BAF relies on simple but efficient algebraic blinding operations to compute forward-secure and aggregate signatures, which offer the following desirable properties:

  - *High Logger Computational Efficiency with Immediate Verification*: The BAF scheme is the only secure logging scheme that is as efficient as a symmetric scheme but yet publicly verifiable without online TTP support or time factor. This makes BAF a magnitude of times

more computational-efficient than all existing PKC-based schemes at the signer side with the exception of HaSAFSS schemes. When compared with HaSAFSS, BAF achieves immediate verification and it does not rely on online TTP support.

- *Small Key/Signature Sizes:* BAF has the smallest signature/key size among all of its counterparts.
- *Provable Security*: Previous secure logging schemes give only heuristic security arguments against the truncation attacks. However, BAF is proven to be secure against the truncation attack in ROM.
- *Fine-Grained Verification*: An extension of BAF, *Fast-Immutable BAF (FI-BAF)*, provides a finer-grained verification of log entries by preserving the computational efficiency and security of BAF.

All of the above properties make BAF an ideal alternative for secure logging in resource-constrained devices.

- *Efficient Secure Logging for Verification-Intensive Applications with Immediate Verification*: All existing PKC-based secure logging solutions require an ExpOp per-item at the verifier side, which make them costly for verification-intensive applications. We address this problem by developing a new forward-secure and append-only signature scheme called LogFAS [124]. We summarize the desirable properties of LogFAS below:
  - *High Verifier Efficiency*: LogFAS is the only PKC-based scheme that can verify L items with only a small-constant number of ExpOps. Hence, it is significantly more efficient than all previous schemes that require a linear number of ExpOps. Furthermore, each verifier stores only a small and constant number of public keys independent from the number of loggers or the number of log entries to be verified. Therefore, it is also more storage efficient than all previous alternatives.
  - *Selective Log Verification with a Sub-linear Number of ExpOps*: LogFAS can identify corrupted log entries with a *sub-linear* number of ExpOps when most log entries are intact. However, all other schemes always require a linear number of ExpOps.
  - LogFAS achieves public verifiability and provable security without requiring online TTP support or time factor.

LogFAS is an ideal alternative for task-intensive applications, in which log verifiers need to verify a large number of log entries simultaneously.

6.2 Future Work

In this chapter, we identify some open problems that can be addressed upon further research. We first discuss some prospective improvements on LogFAS, which aims to increase the computational efficiency of LogFAS while preserving its security. We then discuss a new research direction that is complementary to our current research by offering additional security services.

6.2.1 Improvements on LogFAS

We proved that BAF is resilient to the truncation attacks in ROM [13] without requiring any external signature support. However, LogFAS still relies on a standard signature scheme SGN (e.g., [93]) to prevent the truncation attacks by computing signatures of counter values. It is desirable to eliminate the necessity of such external signature support. This reduces the computation cost of LogFAS signature generation by removing $SGN.Sig$ operation from $LogFAS.FASig$. Similarly, it reduces the LogFAS signature verification cost by eliminating the verification of counter signature β.

To address this problem, we envision a cryptographic simulation for LogFAS, which follows a "batch signature splitting" strategy as in BAF (see Section 4.2 and Section 4.4 for details). However, different from BAF that directly relies on *DLP*, LogFAS relies on the $EU\text{-}CMA$ property of Schnorr signature scheme [105]. Therefore, to prove that LogFAS is "truncation-free" without any external signature support, we first need to ensure that *it is computationally infeasible to split an aggregate Schnorr signature without knowing its individual components* (i.e., the signature extraction is computationally infeasible). That is, let $\sigma_{0,l} \leftarrow (s_{0,l} \equiv \sum_{j=0}^{l} s_j \bmod q, \langle \{R_j, e_j\}_{j=0}^{l} \rangle)$ be a valid aggregate Schnorr signature on data items $(D_0, \ldots, D_l)$ and public keys $(Y_0, \ldots, Y_l)$. Only given the aggregate signature $s_{0,l}$, if no *PPT* adversary can extract an individual signature $s_j, 0 \leq j \leq l$ valid on public key Y_j from $s_{0,l}$, then the signature extraction is computationally infeasible for the aggregate Schnorr (see a detailed discussion on the relationship between the signature extraction argument [26] and truncation attacks [75] in Section 4.2.4).

If our investigation confirms that the above argument holds, then we plan to construct a cryptographic simulation for LogFAS, in which the simulator $\mathcal{F}$ (i.e., the Schnorr attacker) embeds a batch Schnorr signature $s \equiv s_0 + s_1 \bmod q$ into her answers for $\mathcal{A}$'s signature queries. $\mathcal{F}$ then expects the simulator $\mathcal{A}$ splits s into two valid individual Schnorr signatures s_0 and s_1 on their corresponding public keys. If this occurs then simulator $\mathcal{F}$ can extract a forged Schnorr signature from $\mathcal{A}$'s forgery output. That is, if $\mathcal{A}$ performs a valid signature extraction on LogFAS (i.e., a successful truncation attack), then simulator $\mathcal{F}$ breaks the Schnorr signature scheme with a non-negligible probability.

6.2.2 Encrypted Searches on Audit Logs

In this dissertation, we focused on authentication and integrity services for secure audit logging. Another useful security service is searchable encryption [116], which offers the privacy and the usability of audit logs simultaneously. In our future work, we plan to develop efficient and provable secure searchable encryption schemes and then integrate them into our integrity and authentication mechanisms.

Waters et al. proposed the first searchable encryption in [116], which relies on the Boneh-Franklin IBE scheme [25]. Boneh et al. [24] independently proposed a PKC-based searchable encryption scheme (PEKS) motivated by urgent email classification for routers (without decrypting the emails). This scheme is also based on Boneh-Franklin IBE and it has some similarities with the scheme in [116]. However, different from [116], Boneh et al.'s PEKS is a generic PKC-based searchable scheme with formal security models and proofs.

Following Boneh et al.'s PEKS, various PEKS schemes achieving additional properties were proposed. Shi et al. proposed encrypted search schemes for multi-dimensional data in [107]. Bringer et al. proposed encrypted search schemes for biometric data and fuzzy data in [30] and [29], respectively. Wang et al. proposed ranked multi-keyword encrypted search schemes for computing clouds in [114]. Abdalla et al. gave new security definitions and refinements on the encrypted search schemes in [1].

All of the above (PKC-based) schemes are computationally costly on the encryption side and highly computationally costly on the decryption side (i.e., the searcher side). In our future research, we plan to develop novel PEKS schemes that achieve computational efficiency and provable security at the same time. We envision two types of PEKS schemes, each addressing different application requirements.

Searchable Encryption on Audit Logs in Resource-Constrained Devices

Efficient searchable encryption is a need for recently emerging technologies such as biometric devices. For instance, mobile biometric devices [110] are becoming essential tools for extracting and accumulating noisy biometric templates in various scenarios like custom border protection and forensic investigations. Searchable encryption is an ideal tool to provide the privacy and usability of audit logs for these devices. However, all existing PEKS schemes incur prohibitive computational/storage costs for such resource-constrained devices.

We will address this problem by developing PEKS schemes that achieve high computational/storage efficiency on the encrypter side. To achieve computational efficiency, Offline-Online Identity-Based Encryption (OOIBE) schemes [55] offer fast encryption solutions. However, OOIBE schemes do not satisfy the security conditions required for PEKS such as key privacy (i.e., anonymity) [1], and also incur an intolerable storage overhead due to their one-time nature.

We plan to eliminate the one-time key requirement by designing an IND-CCA secure [16,65] stateful IBE [97] that also preserves the OOIBE properties. We will then instantiate this new construction with an anonymous IBE (e.g., Boneh-Boyen [23]) to fulfill the key privacy requirement. Lastly, a security

model capturing the required properties and formal proofs based on this security model will be provided.

Search Efficient PEKS for Fast Searches on Encrypted Audit Logs

Another limitation of PEKS schemes is their immense computational costs in their search phase. The inefficiency stems from the decryption algorithms of PEKSs, which require several expensive operations (e.g., pairing). The mitigation of this inefficiency plays a key role towards making these primitives available for practical use.

We will investigate the feasibility of anonymous IBE constructions that do not rely on pairings or quadratic residues (e.g., [48]). This will lead to *search efficient PEKS schemes*. First, we plan to develop constructions that leverage the naturally anonymous and decryption efficient primitives such as ElGamal encryption [87] with a mediated entity support. Second, we will consider recent results (e.g., [89]) that have a potential to yield an efficient IBE from traditional PKC schemes without requiring a mediated entity.

REFERENCES

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *Journal of Cryptology*, 21:350–391, March 2008.

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

[3] R. Accorsi. Log data as digital evidence: What secure logging protocols have to offer? In *International Computer Software and Applications Conference (COMPSAC '09)*, pages 398–403, 2009.

[4] R. Accorsi. Safe-keeping digital evidence with secure logging protocols: State of the art and challenges. In *Proc. of 5th International Conference on IT Security Incident Management and Forensics (IMF '09)*, pages 94 –110, 2009.

[5] B. Alomair, K. Sampigethaya, and R. Poovendran. Efficient generic forward-secure signatures and proxy signatures. In *Proceedings of the 5th European PKI Workshop on Public Key Infrastructure: Theory and Practice (EuroPKI '08)*, pages 166–181, Berlin, Heidelberg, 2008. Springer-Verlag.

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

[7] A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. In *Proc. of Information Security Conference (ISC '01)*, pages 379–393. Springer-Verlag, 2001.

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

[9] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede. Public-key cryptography for RFID-Tags. In *Proceedings of the Fifth IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMM '07)*, pages 217–222, Washington, DC, USA, 2007. IEEE Computer Society.

[10] M. Bellare and S. Micali. How to sign given any trapdoor permutation. *Journal of ACM*, 39:214–233, January 1992.

[11] M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Proc. of the 16th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '97)*, pages 163–192. Springer-Verlag, 1997.

[12] M. Bellare and S. Miner. A forward-secure digital signature scheme. In *Advances in Crpytology (CRYPTO '99)*, pages 431–448. Springer-Verlag, 1999.

[13] 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)*, pages 62–73, NY, USA, 1993. ACM.

[14] M. Bellare and P. Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Proceedings of the 15th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '96)*, pages 399–416. Springer-Verlag, 1996.

[15] M. Bellare and P. Rogaway. Collision-resistant hashing: Towards making UOWHFs practical. In *Proceedings of Advances in Cryptology (CRYPTO '97)*, pages 470–484, London, UK, 1997. Springer-Verlag.

[16] M. Bellare and P. Rogaway. *Introduction to modern cryptography*. 1st edition, 2005.

[17] M. Bellare and B. S. Yee. Forward integrity for secure audit logs, 1997.

[18] 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)*, pages 1–18, 2003.

[19] F. Bergadano, D. Cavagnino, and B. Crispo. Chained stream authentication. In *Proceedings of the 7th Annual International Workshop on Selected Areas in Cryptography (SAC '00)*, pages 144–157, London, UK, 2000. Springer-Verlag.

[20] J. Bethencourt, D. Boneh, and B. Waters. Cryptographic methods for storing ballots on a voting machine. In *Proc. of the Network and Distributed System Security Symposium (NDSS 07')*, 2007.

[21] A. Boldyreva, C. Gentry, A. O'Neill, and D. Yum. Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In *Proceedings of the 14th ACM Conference on Computer and Communications Security, (CCS '07)*, pages 276–285. ACM, 2007.

[22] D. Boneh and X. Boyen. Efficient Selective-ID secure identity-based encryption without random oracles. In *Proc. of the 23th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '04)*, pages 223–238, 2004.

[23] D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In *Proceedings on Advances in Cryptology (CRYPTO '04)*, pages 443–459, 2004.

[24] D. Boneh, G. D. Crescenzo, Rafail Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Proc. of the 23th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '04)*, pages 506–522, 2004.

[25] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. *SIAM Journal on Computing*, 32:586–615, 2003.

[26] 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)*, pages 416–432. Springer-Verlag, 2003.

[27] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of Cryptology*, 14(4):297–319, 2004.

[28] X. Boyen, H. Shacham, E. Shen, and B. Waters. Forward-secure signatures with untrusted update. In *Proceedings of the 13th ACM conference on Computer and Communications Security (CCS '06)*, pages 191–200, New York, NY, USA, 2006. ACM.

[29] J. Bringer, H. Chabanne, and B. Kindarji. Error-tolerant searchable encryption. In *IEEE International Conference on Communications (ICC '09)*, pages 1–6, June 2009.

[30] J. Bringer, H. Chabanne, and B. Kindarji. Identification with encrypted biometric data. *Security and Communication Networks*, 4(5):548–562, 2011.

[31] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. *Journal of ACM*, 51, July 2004.

[32] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. *Journal of Cryptology*, 20(3):265–294, 2007.

[33] J. Cathalo, B. Libert, and J. Quisquater. Efficient and non-interactive timed-release encryption. In *Proc. of the 6th International Conference on Information and Communications Security (ICICS '05)*, pages 291–303, 2005.

[34] EMC Corp. EMC Centera. `http://www.emc.com/products/family/emc-centera-family.htm`.

[35] K. Chalkias, D. Hristu-Varsakelis, and G. Stephanides. Improved anonymous timed-release encryption. In *12th European Symposium on Research in Computer Security, (ESORICS '07)*, pages 311–326, 2007.

[36] A. Chan and I. F. Blake. Scalable, server-passive, user-anonymous timed release cryptography. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS '05)*, pages 504–513. IEEE Computer Society, 2005.

[37] J. H. Cheon, N. Hopper, Y. Kim, and I. Osipkov. Provably secure timed-release public key encryption. *ACM Transaction of Information System Security*, 11:4:1–4:44, May 2008.

[38] J. Y. Choi, P. Golle, and M. Jakobsson. Tamper-evident digital signature protecting certification authorities against malware. In *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing (DASC '06)*, pages 37–44, 2006.

[39] C. N. Chong and Z. Peng. Secure audit logging with tamper-resistant hardware. In *Proceedings of the 18th IFIP International Information Security Conference*, pages 73–84. Kluwer Academic Publishers, 2003.

[40] S. M. Chow, L. C. Hui, S. Yiu, and K. P. Chow. Forward-secure multisignature and blind signature schemes. *Applied Mathematics and Computation*, 168(2):895–908, 2005.

[41] S. S. Chow and S. M. Yiu. Timed-release encryption revisited. In *Proceedings of the 2nd International Conference on Provable Security (ProvSec '08)*, pages 38–51. Springer-Verlag, 2008.

[42] J. Coron and D. Naccache. Boneh et al.'s k-element aggregate extraction assumption is equivalent to the diffie-hellman assumption. In *Proceedings of the 9th International Conference on the Theory and Application of Cryptology (ASIACRYPT 03')*, pages 392–397, 2003.

[43] 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.

[44] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security (CCS '06)*, pages 79–88. ACM, 2006.

[45] R. D. Crescenzo, Ostrovsky and S. Rajagopalan. Conditional oblivious transfer and timed-release encryption. In *Proceedings of the 18th International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT '99)*, pages 74–89, Berlin, Heidelberg, 1999. Springer-Verlag.

[46] Information Processing Technology Office (IPTO) Defense Advanced Research Projects Agency (DARPA). BBA 07-46 LANdroids broad agency announcement, 2007. `http://www.darpa.mil/ipto/solicit/baa/BAA-07-46_PIP.pdf`.

[47] D. Davis, F. Monrose, and M. Reiter. Time-scoped searching of encrypted audit logs. In *Proc. of the 6th International Conference on Information and Communications Security (ICICS '04)*, pages 532–545, 2004.

[48] X. Ding and G. Tsudik. Simple identity-based cryptography with mediated rsa. In *Proceedings of the RSA conference on the cryptographers' track (CT-RSA '03)*, pages 193–210. Springer-Verlag, 2003.

[49] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In *Proceedings of the 21th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '02)*, pages 65–82, 2002.

[50] K. Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 9th conference on Applications, technologies, architectures, and protocols for computer communications, (SIGCOMM '03)*, pages 27–34. ACM, 2003.

[51] S. Farrell and V. Cahill. Security considerations in space and delay tolerant networks. In *Proc. of the 2nd IEEE Conference on Space Mission Challenges for Information Technology (SMC-IT '06)*, July 2006.

[52] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[53] Delay Tolerant Networking Research Group. Delay tolerant networks. `http://www.dtnrg.org/wiki`.

[54] L. C. Guillou and J. J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In *Proceedings on Advances in Cryptology (CRYPTO '88)*, pages 216–231. Springer-Verlag, 1988.

[55] F. Guo, Y. Mu, and Z. Chen. Identity-based online/offline encryption. In *Financial Cryptography and Data Security (FC '08)*, pages 247–261. Springer-Verlag, 2008.

[56] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[57] J. E. Holt. Logcrypt: Forward security and public verification for secure audit logs. In *Proc. of the 4th Australasian workshops on Grid computing and e-research (ACSW '06)*, pages 203–211, 2006.

[58] Windsor W. Hsu and Shauchi Ong. Technical forum: WORM storage is not enough. *IBM System Journal*, 46(2):363–369, 2007.

[59] J. Y. Hwang, D. H. Lee, and M. Yung. Universal forgery of the identity-based sequential aggregate signature scheme. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security (ASIACCS '09)*, pages 157–160, New York, NY, USA, 2009. ACM.

[60] Y. Hwang, D. Yum, and P. Lee. Timed-release encryption with pre-open capability and its application to certified e-mail system. In *Proceedings of the 8th International Conference on Information Security, (ISC '05)*, pages 77–82. Springer-Verlag, 2005.

[61] R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 236–241, Washington, DC, USA, 1989. IEEE Computer Society.

[62] G. Itkis. Cryptographic tamper evidence. In *Proc. of the 10th ACM conference on Computer and communications security (CCS '03)*, pages 355–364, New York, NY, USA, 2003. ACM.

[63] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In *Advances in Cryptology (CRYPTO '01)*, pages 332–354. Springer-Verlag, 2001.

[64] A. Joux and K. Nguyen. Separating decision diffie-hellman from computational diffie-hellman in cryptographic groups. *Journal of Cryptology*, 16(4):239–247, 2003.

[65] J. Katz and Y. Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[66] M. Kim and K. Kim. A new identification scheme based on the bilinear diffie-hellman problem. In *Proceedings of the 7th Australian Conference on Information Security and Privacy*, ACISP '02, pages 362–378, London, UK, 2002. Springer-Verlag.

[67] A. Kozlov and L. Reyzin. Forward-secure signatures with fast key update. In *Proc. of the 3rd International Conference on Security in Communication Networks (SCN '02)*, 2002.

[68] H. Krawczyk. Simple forward-secure signatures from any signature scheme. In *Proceedings of the 7th ACM conference on Computer and Communications Security, (CCS '00)*, pages 108–115. ACM, 2000.

[69] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. IETF RFC 2104, February 1997.

[70] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[71] B. Libert, J. Quisquater, and M. Yung. Forward-secure signatures in untrusted update environments: Efficient and generic constructions. In *Proceedings of the 14th ACM conference on Computer and communications security (CCS '07)*, pages 266–275. ACM.

[72] 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)*, pages 465–485. Springer-Verlag, 2006.

[73] A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. Sequential aggregate signatures from trapdoor permutations. In *Proc. of the 23th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '04)*, pages 74–90. Springer-Verlag, 2004.

[74] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf. Pseudonym systems. In *Proceedings of the 6th Annual International Workshop on Selected Areas in Cryptography*, SAC '99, pages 184–199, London, UK, 2000. Springer-Verlag.

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

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

[77] D. Ma and G. Tsudik. DISH: Distributed self-healing. In *Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS '08)*, pages 47–62. Springer-Verlag, 2008.

[78] 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)*, pages 48–63, 2008.

[79] D. Ma and G. Tsudik. A new approach to secure logging. *ACM Transaction on Storage (TOS)*, 5(1):1–21, 2009.

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

[81] P. Maniatis and M. Baker. Authenticated append-only skip lists. *Acta Mathematica*, 137:151–169, 2003.

[82] M. Mass. Pairing-based cryptography. Master's thesis, Technische Universiteit Eindhoven, 2004.

[83] T. Matsuda, Y. Nakai, and K. Matsuura. Efficient generic constructions of timed-release encryption with pre-open capability. In *Proceedings of the 4th international conference on Pairing-based cryptography*, Pairing '10, pages 225–245, Berlin, Heidelberg, 2010. Springer-Verlag.

[84] S. Mauw, I. Vessem, and B. Bos. Forward secure communication in wireless sensor networks. In *Proceedings of the 3rd International Conference on Security in Pervasive Computing (SPC '06)*. Springer-Verlag, 2006.

[85] T. May. Time release crypto. Technical report, February 1993.

[86] J. C. McEachen and J. Casias. Performance of a wireless unattended sensor network in a freshwater environment. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS '08)*, Washington, DC, USA, 2008. IEEE.

[87] A.J. Menezes, P. C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. ISBN: 0-8493-8523-7.

[88] R. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, Apr 1980.

[89] C. J. Mitchell, F. C. Piper, and P. R. Wild. ID-based cryptography using symmetric primitives. *Desing, Codes and Cryptography*, 44:249–262, September 2007.

[90] Y. Mu, W. Susilo, and H. Zhu. Compact sequential aggregate signatures. In *Proceedings of the 22nd ACM symposium on Applied computing (SAC '07)*, pages 249–253. ACM, 2007.

[91] E. Mykletun, M. Narasimha, and G. Tsudik. Signature bouquets: Immutability for aggregated/condensed signatures. In *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS '04)*, pages 160–176. Springer-Verlag, September 2004.

[92] D. Naccache, D. M'Raïhi, S. Vaudenay, and D. Raphaeli. Can D.S.A. be improved? complexity trade-offs with the digital signature standard. In *Proc. of the 13th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '94)*, pages 77–85, 1994.

[93] National Institute of Standards and Technology. Federal information processing standard 186: Digital signature standard. `http://csrc.nist.gov/publications/`, 1993.

[94] A. Oprea and K. D. Bowers. Authentic time-stamps for archival storage. In *14th European Symposium on Research in Computer Security, (ESORICS '09)*, pages 136–151, Berlin, Heidelberg, 2009. Springer-Verlag.

[95] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *Proc. of the 15th ACM conference on Computer and Communications Security (CCS 2008)*, pages 437–448, New York, NY, USA, 2008. ACM.

[96] A. Perrig, R. Canetti, D. Song, and D. Tygar. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2000.

[97] L. T. Phong, H. Matsuoka, and W. Ogata. Stateful identity-based encryption scheme: Faster encryption and decryption. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security (ASIACCS '08)*, pages 381–388. ACM, 2008.

[98] D. Pietro, L.V. Mancini, C. L.V., A. Spognardi, and G. Tsudik. Catch me (if you can): Data survival in unattended sensor networks. *Proceedings of the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom '08)*, pages 185–194, March 2008.

[99] R. Di Pietro, D. Ma, C. Soriente, and G. Tsudik. Posh: Proactive co-operative self-healing in unattended wireless sensor networks. *IEEE Symposium on Reliable Distributed Systems (SRDS '08)*, pages 185–194, October 2008.

[100] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Proc. of the 15th International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '96)*, pages 387–398. Springer-Verlag, 1996.

[101] R. Rivest, A. Shamir, and D. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Cambridge, MA, USA, 1996.

[102] R.L. Rivest, A. Shamir, and L.A. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[103] B. Schneier and J. Kelsey. Cryptographic support for secure logs on untrusted machines. In *Proc. of the 7th conference on USENIX Security Symposium*. USENIX Association, 1998.

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

[105] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[106] Shamus. Multiprecision integer and rational arithmetic c/c++ library (MIRACL). `http://www.shamus.ie/`.

[107] E. Shi, J. Bethencourt, H. T.-H. Chan, D. X. Song, and A. Perrig. Multi-dimensional range query over encrypted data. In *IEEE Symposium on Security and Privacy*, pages 350–364, 2007.

[108] V. Shoup. NTL: A library for doing number theory. `http://www.shoup.net/ntl/`.

[109] D. Stinson. *Cryptography: Theory and Practice,Second Edition*. CRC/C&H, 2002.

[110] Crossmatch Technologies. Mobile biometric devices, 2011. `http://www.mobioproject.org/`.

[111] Trident Systems. Trident's family of unattended ground sensors. `http://www.tridsys.com/white-unattended-ground-sensors.htm`.

[112] U.S. National Institute of Standards and Technology. DES modes of operation. Federal Information Processing Standards Publication 81 (FIPS PUB 4-3), December 1980.

[113] H. Varsakelis, K. Chalkias, and G. Stephanides. Low-cost anonymous timed-release encryption. In *Proceedings of the 3rd International Symposium on Information Assurance and Security (IAS '07)*, pages 77–82. IEEE Computer Society, 2007.

[114] C. Wang, N. Cao, J. Li, K. Ren, and W. Lou. Secure ranked keyword search over encrypted cloud data. In *Proc. of 30th International Conference on Distributed Computing Systems (ICDCS '10)*, pages 253–262, 2010.

[115] Y. Wang and Y. Zheng. Fast and secure magnetic worm storage systems. In *Proc. of the 2nd IEEE International Security in Storage Workshop (SISW '03)*, pages 11–25, October 2003.

[116] B. Waters, D., G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '04)*, 2004.

[117] Wikipedia. Elliptic curve cryptography. `http://en.wikipedia.org/wiki/Elliptic_curve_cryptography`. Visited on August 9th, 2010.

[118] W. Xu, D. Chadwick, J. Li, and S. Otenko. A PKI-based secure audit web service. In *Proc. IASTED Int. Conf. on Communication, Network, and Information Security (CNIS '05)*, 2005.

[119] A. A. Yavuz, F. Alagöz, and E. Anarim. HIMUTSIS: Hierarchical multi-tier adaptive ad-hoc network security protocol based on signcryption type key exchange schemes. In *Proceedings of the 21th International Symposium Computer and Information Sciences (ISCIS '06)*, volume 4263 of *Lecture Notes in Computer Science*, pages 434–444. Springer-Verlag, 2006.

[120] 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)*, pages 219–228, 2009.

[121] A. A. Yavuz and P. Ning. Hash-based sequential aggregate and forward secure signature for unattended wireless sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous '09)*, July 2009.

[122] A. A. Yavuz and P. Ning. Self-sustaining, efficient and forward-secure cryptographic constructions for unattended wireless sensor networks. Submitted to Ad Hoc Networks, Elsevier, 2011.

[123] A. A. Yavuz, P. Ning, and M. K. Reiter. BAF and FI-BAF: Efficient and publicly verifiable secure audit logging for resource-constrained devices, 2011.

[124] A. A. Yavuz, P. Ning, and M. K. Reiter. Efficient, compromise resilient and append-only cryptographic constructions for digital forensics. Submitted to *ACSAC* 2011, 2011.

[125] Y. Zheng. Digital signcryption or how to achieve cost(signature & encryption) $<<$ cost(signature) + cost(encryption). In *Proceedings of Advances in Cryptology (CRYPTO '97)*, pages 165–179, 1997.

[126] H. Zhu, F. Bao, and T. Chigan. Compact routing discovery protocol with lower communication complexity. In *IEEE Wireless Communications and Networking Conference (WNCN '06)*, 2006.

[127] H. Zhu and J. Zhou. Finding compact reliable broadcast in unknown fixed-identity networks (short paper). In *Proc. of the 6th International Conference on Information and Communications Security (ICICS '06)*, pages 72–81, 2006.