BAF and FI-BAF: Efficient and Publicly Verifiable Secure Audit Logging in Distributed Systems

ATTILA ALTAY YAVUZ and PENG NING
North Carolina State University

Audit logs, providing information about the current and past states of systems, are considered one important part of modern computer systems. Protection of audit logs on an untrusted machine in a large distributed system is a challenging task, especially in the presence of active adversaries. In such a system, it is critical to have *forward security* such that when an adversary compromises a machine, she cannot modify or forge the log entries accumulated before the compromise. Unfortunately, existing secure audit logging schemes have significant limitations that make them impractical for real-life applications: Existing Public Key Cryptography (PKC) based schemes are computationally expensive for logging in task intensive or resource-constrained systems, while existing symmetric schemes are not publicly verifiable and incur significant storage and communication overheads.

In order to address these limitations, we develop a new class of forward secure and aggregate logging schemes called *Blind-Aggregate-Forward (BAF)* and *Fast-Immutable BAF (FI-BAF)* logging schemes, which are suitable for large distributed systems. BAF can produce publicly verifiable forward secure and aggregate signatures with near-zero computation, signature storage, and signature communication overheads for the loggers, without requiring any online Trusted Third Party (TTP) support. FI-BAF extends BAF by allowing selective verification of log entries without compromising the security of BAF as well as retaining its computational efficiency. We prove that both BAF and FI-BAF are secure under appropriate computational assumptions, and demonstrate that they are significantly more efficient and scalable than the previous schemes. Therefore, the proposed schemes are ideal solutions for secure logging in both task intensive and resource-constrained systems.

Categories and Subject Descriptors: K.6.5 [**Computing Milieux**]: Security and Protection; H.2.7 [**Information Systems**]: Database Administration—*Security, integrity, and protection*

General Terms: Security, Design

Additional Key Words and Phrases: Secure audit logging; digital signature; forward security; signature aggregation; digital forensics

1. INTRODUCTION

Audit logs are a fundamental digital forensic mechanism for providing security in computer systems; they are used to keep track of important events about the system activities such as program executions/crashes, data modifications, and user activities. The forensic value of audit logs makes them an attractive target for

Authors' address: Attila A. Yavuz and Peng Ning, Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206; emails: aayavuz@ncsu.edu, pning@ncsu.edu.
A preliminary version of this paper appeared in [Yavuz and Ning 2009a].

attackers, who aim to erase the traces of their malicious activities recorded by audit logs [Ma and Tsudik 2009; Schneier and Kelsey 1999].

Some naive audit log protection techniques include using a bug-free tamper-resistant hardware (to prevent the attacker from reaching audit logs), and maintaining a continuous and secure communication channel between each logger and remote trusted entity(ies) (to upload logs to a trusted entity in real-time before the attack occurs). However, as pointed out by some recent studies (e.g., [Ma and Tsudik 2009; 2008; Bellare and Yee 2003; Schneier and Kelsey 1998]), these techniques are impractical for modern computer systems. In large distributed systems (e.g., virtual computing cloud), it is impractical to assume a continuous end-to-end real-time communication between a trusted center and a logger [Fall 2003]. Similarly, assuming a tamper-resistant hardware being "bug free" and guaranteeing its presence on all types of platforms are equally impractical (e.g., logging in smart cards, implantable devices [Halperin et al. 2008] and wireless sensors [Ma and Tsudik 2007]).

To address the above problems, a set of cryptographic countermeasures have been proposed to enable secure logging on untrusted machines, without assuming a tamper-resistant hardware or continuous real-time log verifiers (e.g., [Schneier and Kelsey 1999; 1998; Ma 2008; Ma and Tsudik 2009; 2008]). In the setting where there is no tamper resistant hardware nor continuous real-time communication, the untrusted machine has to accumulate audit log entries when the log verifiers are not available. After the attacker compromises the system, no cryptographic technique can prevent her from manipulating the post-attack log entries (due to her control over the system). However, it is critical to prevent the attacker from manipulating the log entries previously accumulated before the compromise. Such a security property is referred to as *forward security* [Bellare and Yee 2003; Ma and Tsudik 2008; 2009].

One group of these schemes rely on symmetric cryptography to provide forward security in a computationally efficient way by using forward-secure Message Authentication Codes (MACs), Pseudo Random Number Generators (PRNGs) (e.g., [Bellare and Yee 1997; 2003; Schneier and Kelsey 1998]), and one-way hash chains (e.g., [Schneier and Kelsey 1998; Ma and Tsudik 2007; Schneier and Kelsey 1999]). Despite their simplicity and computational efficiency, these schemes have significant limitations: (i) Due to their symmetric nature, these schemes cannot achieve public verifiability. As a result, they either require full symmetric key distribution (e.g., FssAgg-MAC in [Ma and Tsudik 2007]) or online TTP support (e.g., [Bellare and Yee 1997; 2003; Schneier and Kelsey 1998; 1999]). While full symmetric key distribution incurs significant storage overhead to system entities, the online TTP requirement brings architectural difficulties, increases communication overhead, and makes the system vulnerable to the single point of failure problems [Ma and Tsudik 2007]. (ii) All the above schemes incur high storage and communication overheads to the loggers, since they require storing and transmitting an authentication tag for each log entry (or logging period) (e.g., [Bellare and Yee 1997; 2003; Schneier and Kelsey 1998; 1999]). (iii) Many of these schemes (e.g., [Bellare and Yee 1997; 2003]) have been shown to be vulnerable to the truncation and delayed detection attacks [Ma and Tsudik 2009; 2008][1].

Another group of schemes rely on Public Key Cryptography (PKC). 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 [Holt 2006]. However, Logcrypt incurs high storage and communication overheads due to the requirements of storing and transmitting an authentication tag for each log entry or logging period. Furthermore, it is still vulnerable to the truncation attack.

Recently, Ma and Tsudik proposed FssAgg schemes (i.e., FssAgg-MAC/BLS [Ma and Tsudik 2007], FssAgg-BM/AR [Ma 2008; Ma and Tsudik 2008] and iFssAgg variants [Ma and Tsudik 2009]) to address the above limitations. FssAgg schemes use forward secure and aggregate signatures to reduce storage and communication overheads and are also secure against the truncation attack. 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. Furthermore, since the single-final aggregate signature covers the accumulated log entries as a whole (forward-secure stream integrity [Ma 2008]), the truncation attack is no longer possible based on the unforgeability of the underlying FssAgg scheme.

Despite its advantages, the use of only a single-final aggregate signature to verify the entire set of log entries brings performance drawbacks: A verifier that is only interested in a particular log entry has to verify the entire set of log entries associated with the single-final aggregate signature. This incurs significant computational overheads to the verifiers. Therefore, in many cases, it is desirable to keep individual signatures along with the aggregate signature to enable separate verification of individual log entries. However, keeping individual signatures along with the aggregate signature allows the attacker to launch the truncation attack on FssAgg schemes. To prevent this, Ma and Tsudik [2009] offers variant FssAgg schemes, *immutable FssAgg (iFssAgg)* schemes, which allow a finer-grained verification of log entries via individual signatures while preventing the truncation attack.

Unfortunately, despite their attractive properties, all these PKC-based schemes (e.g., [Holt 2006; Ma and Tsudik 2007; Ma 2008; Ma and Tsudik 2008; 2009]) suffer from a common drawback: They are highly computationally expensive for the logger and even more for the log verifier. In addition, iFssAgg schemes [Ma and Tsudik 2009] require additional signature computation/verification operations over the original FssAgg schemes, increasing the computational overhead even further. (In other words, completely eliminating the truncation attack requires doubling the computational overhead on both the signer and verifier sides.) These high computational costs make all the aforementioned PKC-based schemes impractical for logging in task-intensive or resource constrained systems.

[1]In a truncation attack, the attacker can truncate a subset of tail-log entries from the log without being detected. In a delayed detection attack, log verifiers cannot detect whether the log entries are manipulated until an online TTP provides necessary keying information to them. Therefore, secure audit logging mechanisms that cannot achieve the immediate verification are vulnerable to this attack.

1.1 Our Contributions

The above discussion indicates that efficient audit logging mechanisms refrained from all the above limitations are solely needed. In order to fulfill this need, we propose a new class of forward secure and aggregate logging schemes for secure audit logging in distributed systems, which we call *Blind-Aggregate-Forward (BAF)* and *Fast-Immutable BAF (FI-BAF)* logging schemes. BAF and FI-BAF can address all the aforementioned limitations of existing approaches simultaneously.

We summarize the properties of our schemes as follows:

(1) *Efficient Log Generation:* In BAF, the computational cost of logging a single data item is only three cryptographic hash operations. This is as efficient as existing symmetric schemes (e.g., [Ma and Tsudik 2007; Schneier and Kelsey 1999; 1998; Bellare and Yee 1997; 2003]), and is much more efficient than all existing PKC-based schemes (e.g., [Ma and Tsudik 2007; Ma 2008; Ma and Tsudik 2009; Holt 2006; Ma and Tsudik 2008]).

(2) *Efficient Log Verification:* In BAF, the computational cost of verifying a single log entry is only a single ECC scalar multiplication, which is more efficient than existing PKC-based schemes (e.g., [Holt 2006; Ma and Tsudik 2007; Ma 2008; Ma and Tsudik 2008; 2009]).

(3) *Public Verifiability:* BAF produces publicly verifiable signatures (which implies no full symmetric key distribution), and therefore is much more scalable for distributed systems than symmetric schemes (e.g., FssAgg-MAC in [Ma and Tsudik 2007] and [Schneier and Kelsey 1998; 1999; Bellare and Yee 1997; 2003]).

(4) *Offline TTP and Immediate Verification:* Unlike some previous schemes (e.g., [Schneier and Kelsey 1998; 1999]), BAF does not need online TTP support to enable log verification. Hence, it eliminates the bandwidth overhead that stems from the frequent communication between log verifiers and the TTP. This also makes BAF more scalable and reliable due to the simple architectural design and being free of single point of failures. Last, since BAF achieves immediate verification, it is secure to the delayed detection attack [Ma and Tsudik 2009].

(5) *Small-Constant Signature Storage and Transmission:* In BAF, independent from the number of time periods or data items to be signed, a logger only needs to store/transmit a single and compact aggregate signature as the authentication tag for the entire logging process. Thus, our scheme is much more signature storage/bandwidth efficient than existing symmetric schemes (e.g., linear signature storage/transmission on the logger [Schneier and Kelsey 1999; 1998; Bellare and Yee 1997; 2003]).

(6) *Fast and Immutable Logging:* FI-BAF addresses the need of a BAF variant that 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, instead of directly adapting the umbrella signature technique [Myketun et al. 2004] as in [Ma and Tsudik 2009], uses optimization techniques that eliminate extra operations in umbrella technique not required for FI-BAF. Therefore, while enabling individual log verification and preventing the truncation attack, FI-BAF incurs near-zero computational overhead beyond the original BAF.

The above properties make BAF and BI-BAF ideal candidates for secure audit logging in large distributed systems even for highly resource constrained environments such as smart cards, implantable devices [Halperin et al. 2008] and wireless sensor networks [Ma and Tsudik 2007].

The remainder of this paper is organized as follows. Section 2 presents some preliminary definitions and assumptions. Section 3 provides the BAF syntax and security model. Sections 4 and 5 describe BAF and FI-BAF in detail, respectively. Section 6 gives the security analysis of BAF and FI-BAF. Section 7 presents performance analysis of BAF and FI-BAF and compares them with previous approaches. Section 8 briefly discusses related work, and Section 9 concludes this paper.

2. PRELIMINARIES

This section provides our notation, definitions, and complexity/system assumptions.

Notation. G is a generator of group $\mathbb{G}$ defined on an Elliptic Curve (EC) $E(F_p)$ over a prime field F_p, where p is a large prime number and q is the order of G. kG, where $k \in F_q$ is an integer, denotes a *scalar multiplication.* $x \overset{R}{\leftarrow} F_q$ denotes that variable x is selected randomly and uniformly from F_q. Operators $||$ and $|x|$ denote the concatenation operation and the bit length of variable x, respectively. $H_1/H_2/H_3/H_4$ are four distinct Full Domain Hash (FDH) functions [Bellare and Rogaway 1996], which are defined as $H_1 : \{0,1\}^{|q|} \rightarrow \{0,1\}^{|q|}$, $H_2 : \{0,1\}^* \rightarrow \{0,1\}^{|q|}$, $H_3 : \{0,1\}^{|3q|} \rightarrow \{0,1\}^{|3q|}$ and $H_4 : \{0,1\}^* \rightarrow \{0,1\}^{|p|}$, respectively.

***Definition* 2.1** Elliptic Curve Discrete Logarithm Problem (ECDLP) [Hankerson et al. 2004] 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* 2.2** The Multiplicative-HASH (MuHASH) [Bellare and Micciancio 1997] is defined as follows: Given data items $D = (D_0, \ldots, D_{L-1})$, a large prime p, and a secure FDH H_4, the MuHASH of D is

$$MuHASH(D_0, \ldots, D_{L-1}) = \prod_{j=0}^{L-1} H_4(D_j||j) \bmod p.$$

MuHASH is (τ, ϵ)-hard, if no algorithm running in time less than τ can find a collision for MuHASH with a probability more than ϵ, where ϵ is computed over the random choice of p.

Note that the collision-freeness of MuHASH means that it is computationally infeasible to find a collision such that $\exists j : D_j^* \neq D_j, 0 \leq j \leq L-1$ and $v = v^*$ where $v = MuHASH(D_0, \ldots, D_{L-1})$ and $v^* = MuHASH(D_0^*, \ldots, D_{L-1}^*)$. The intractability of MuHASH relies on the intractability of Discrete Logarithm Problem (DLP) [Bellare and Micciancio 1997; Wagner 2002] (the base problem of ECDLP [Hankerson et al. 2004]).

Assumption 2.1 *The assumptions on which BAF and FI-BAF are constructed are given below:*

(*1*) *The system owner who generates and distributes public/secret keys cannot be compromised by adversary $\mathcal{A}$.*

(*2*) *$H_1/H_2/H_3/H_4$ are ideal collision-free FDHs. The global primes (p, q), elliptic curve $E(F_p)$, and the generator G are generated by the system owner.*

(*3*) *Both BAF and FI-BAF assume that the ECDLP is (τ, ϵ)-hard [Boneh 1998]. FI-BAF additionally assumes MuHASH is (τ, ϵ)-hard [Bellare and Micciancio 1997].*

3. SYNTAX AND SECURITY MODEL

In this section, we first describe the *BAF syntax* to clarify generic BAF algorithms, key update and signer behavior models. We also briefly discuss the FI-BAF algorithms based on the BAF syntax. The BAF syntax enables us to formally define the *BAF security model.* We then give the BAF security model in which BAF is analyzed for the forward secure and aggregate existential unforgeability against adaptive chosen message attacks. In this model, we also discuss the security goals and concerns that BAF and FI-BAF aim to address such as truncation and delayed detection attacks.

The actual BAF and FI-BAF algorithms are presented in Section 4 and Section 5, respectively. The formal analysis of BAF and FI-BAF is provided in Section 6.

3.1 Syntax

BAF is an integrated scheme that achieves both forward security and sequential signature aggregation simultaneously. Hence, BAF has a *Key Update* algorithm that follows the "evolve-delete strategy" to achieve forward security similar to generic forward secure signatures (e.g., [Krawczyk 2000]). Moreover, it has *Key Generation*, *Aggregate Signature Generation* and *Aggregate Signature Verification* algorithms similar to the aggregate signatures (e.g., [Mu et al. 2007; Boneh et al. 2003; Boldyreva et al. 2007; Lysyanskaya et al. 2004; Ma and Tsudik 2007]).

***Definition* 3.1** BAF is four-tuple of algorithms $BAF = (Kg, Upd, ASig, AVer)$ that behave as follows:

—*BAF.Kg:* $BAF.Kg$ is the key generation algorithm, which takes the maximum number of key updates L and identity of signer i (ID_i) as the input and returns L public keys $(pk_0, \dots, pk_{L-1})$, initial secret key sk_0, and index $n \overset{R}{\leftarrow} F_q$ for ID_i as the output.

—*BAF.Upd:* $BAF.Upd$ is the key update algorithm, which takes the current secret key sk_j where $0 \le j \le L-1$ as the input, and returns the next secret key sk_{j+1} as the output. $BAF.Upd$ also securely erases sk_j from memory.

—*BAF.ASig:* $BAF.ASig$ is the aggregate signature generation algorithm, which takes sk_j, data item $D_j \in \{0,1\}^*$ to be signed, and an aggregate signature $\sigma_{0,j-1}$ (for previously accumulated data items) as the input. It returns an aggregate signature $\sigma_{0,j}$ by folding the individual signature of D_j (i.e., σ_j) into $\sigma_{0,j-1}$, and then securely erases $(\sigma_{0,j-1}, \sigma_j)$ from memory.

—*BAF.AVer:* $BAF.AVer$ is the aggregate signature verification algorithm, which takes $(D_0, \ldots, D_j) \in \{0,1\}^*$, its associated aggregate signature $\sigma_{0,j}$, index n and public keys $(pk_0, \ldots, pk_j)$ of ID_i as the input. If the signature is successfully verified, $BAF.AVer$ returns *success*. Otherwise, it returns *failure*. We require that any σ generated by $BAF.ASig$ is accepted by $BAF.AVer$.

FI-BAF follows the same syntax as BAF except that the signing and verifying algorithms of FI-BAF include *individual signature* computation and verification steps, respectively, in addition to the aggregate signature computation.

***Definition* 3.2** BAF and FI-BAF 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 accumulated.
(2) *Per-period model*: Sign a group of entry D'_j for each time period t_j, where D'_j denotes $\forall D_j$ accumulated in t_j.

In terms of the key evolving strategy, *per-item* and *per-period* key update models are identical. However, they allow users to set a *storage-security trade-off* [Ma 2008] that can be decided according to the requirements of audit logging applications. That is, per-item model provides forward security of each individual data item, but it demands more storage on the verifiers. In contrast, per-period model provides forward security only for across time periods, but it demands less storage on the verifiers.

BAF and FI-BAF behave according to the *same-signer-distinct-message model* similar to existing forward secure and aggregate logging schemes (e.g, [Ma and Tsudik 2007; Ma 2008; Ma and Tsudik 2008; 2009]). 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 [Mykletun et al. 2004]). This model is an ideal option for secure audit logging applications (e.g., [Schneier and Kelsey 1999; Ma and Tsudik 2008; Schneier and Kelsey 1998; Ma and Tsudik 2009; Ma 2008; Waters et al. 2004]), since each logger is only responsible for her own audit logs. Note that some aggregate signature schemes (e.g., [Boneh et al. 2003; Mu et al. 2007; Boldyreva et al. 2007]) use the *different-signer-distinct-message model* (e.g., for secure routing purposes), which is not necessary for the envisioned secure audit logging applications.

3.2 Security Model

The security of BAF is defined as the non-existence of a capable but Probabilistic Polynomial Time (PPT)-bounded adversary $\mathcal{A}$, confined with certain games, producing an existential forgery against the BAF even under the exposure of the current keying material. Since BAF aims to achieve both secure sequential signature aggregation and forward security simultaneously, we develop our security model based on the security model of aggregate signatures in [Boneh et al. 2003; Boldyreva et al. 2007; Mu et al. 2007; Lysyanskaya et al. 2004], generic forward secure signature model in [Krawczyk 2000], and hybrid security model of FssAgg schemes in [Ma and Tsudik 2007; Ma 2008]. The security model of BAF is defined below:

***Definition* 3.3** BAF is an existentially unforgeable forward secure and aggregate signature scheme against adaptive chosen message attacks, if no PPT-bounded $\mathcal{A}$ can win the following game with a non-negligible probability.

(1) *Setup.* Signing oracle $\mathcal{O}$ randomly selects (L, n), a random forgery time period $0 \leq t \leq L - 1$ and a challenge public key pk_c. $\mathcal{O}$ gives (L, n, pk_c) to $\mathcal{A}$.
(2) *Queries.* Beginning from $j = 0$, proceeding adaptively, $\mathcal{A}$ is provided with $\mathcal{O}$ under secret key sk_j once. For each query $j > 0$, $\mathcal{A}$ supplies a valid BAF signature $\sigma_{0,j-1}$ on some messages $D_0 \in \{0,1\}^*, \ldots, D_{j-1} \in \{0,1\}^*$ signed under $sk_0, \ldots, sk_{j-1}$, where both messages and keys are of her choice. $\mathcal{A}$ also queries an additional message $D_j \in \{0,1\}^*$ of her choice once, which is signed by $\mathcal{O}$ under sk_j. $\mathcal{A}$ then proceeds into the next time period and is provided with oracle $\mathcal{O}$ under sk_{j+1}. The adaptive queries continue, until $\mathcal{A}$ "breaks-in".
(3) *Break-in.* When $\mathcal{A}$ decides to break-in in time period t_T, she is allowed to access secret key sk_T.
(4) *Forgery.* Eventually, $\mathcal{A}$ halts and outputs an aggregate signature $\sigma^*_{0,T}$ on $D^*_0, \ldots, D^*_T$ under $pk^*_0, \ldots, pk^*_T$. $\mathcal{A}$ wins the game, if (i) $t = T$, (ii) $pk^*_t = pk_c$, (iii) $\sigma^*_{0,t}$ is verified by $BAF.AVer$ successfully, and (iv) $\sigma^*_{0,t}$ is non-trivial (i.e., $\mathcal{A}$ did not ask for a signature on D^*_t for time period t in the query phase).

***Definition* 3.4** The forger $\mathcal{A}$ (q_s, τ, ϵ, L)-*breaks* a BAF scheme iff: She outputs a successful forgery in the game given in Definition 3.3 with probability advantage at least ϵ, and runs in at most time τ by making at most q_s signature queries to $\mathcal{O}$. A BAF scheme is (q_s, τ, ϵ, L)-*forward secure-existentially unforgeable* against adaptive chosen-message attacks in the BAF security model if no forger $\mathcal{A}$ (q_s, τ, ϵ, L)-*breaks* it.

Basic security goals that a forward-secure aggregate signature scheme must achieve are *forward security, integrity, authentication, and unforgeability* [Yavuz and Ning 2009b; Ma and Tsudik 2007; 2007; 2009]. BAF and FI-BAF achieve these goals based on the fact that BAF is (q_s, τ, ϵ, L)-forward secure-existentially unforgeable, which is proven in Section 6 in Theorem 6.1.

In addition to the above security properties, another security concern in audit logging is *truncation* and *delayed detection* attacks identified in [Ma and Tsudik 2008; 2009]. *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. This attack can be prevented via "all-or-nothing" property [Ma and Tsudik 2007]: $\mathcal{A}$ either should retain previously accumulated data intact, or should not use them at all ($\mathcal{A}$ cannot selectively delete/modify any subset of this data [Ma and Tsudik 2009]). *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 the 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 and Tsudik [2009] shows that many existing schemes are vulnerable to these attacks (e.g., [Bellare and Yee 2003], [Bellare and Yee 1997], [Schneier and Kelsey 1999], [Schneier and Kelsey 1998]).

In section 6, we show that both BAF and FI-BAF are secure against these attacks based on the existential unforgeability of BAF.

4. BLIND-AGGREGATE-FORWARD (BAF)

In this section, we first give an overview of the proposed BAF scheme and then give the detailed description.

4.1 Overview

The objective of BAF is to achieve five seemingly conflicting goals at the same time to compute and verify forward secure and aggregate signatures for secure audit logging purposes, including high signer computational efficiency, signature storage/transmission efficiency, public verifiability, immediate verification, and high verifier computational efficiency. To demonstrate how BAF achieves these properties, we first discuss the envisioned design principles, on the basis of which BAF strategy is constructed. We then present the BAF strategy that achieves the stated goals by following these design principles.

Design Principles: BAF is based on the following design principles:

—*Avoid PKC operations for logging:* All existing PKC-based forward secure and aggregate schemes are directly derived from the existing aggregate or forward secure signature schemes. For instance, FssAgg-BLS [Ma and Tsudik 2007] is based on the aggregate signature scheme given in [Boneh et al. 2003]. Similarly, FssAgg-BM and FssAgg-AR in [Ma 2008; Ma and Tsudik 2009; 2008] are based on the forward secure signatures given in [Bellare and Miner 1999] and [Abdalla and Reyzin 2000], respectively. Hence, existing forward secure and aggregate schemes naturally inherit high computational costs of these signature primitives. To achieve efficiency, BAF restricts operations used in signature generation to basic arithmetic operations and cryptographic hash functions. This implies that BAF does not use any PKC operation for logging.

—*Avoid time factor and online TTP Support:* One possible way to avoid PKC operations for logging is to introduce asymmetry between the signer and the verifiers via the time factor (e.g., TESLA [Perrig et al. 2000]). However, such a scheme cannot achieve immediate verification at the verifier side. Moreover, it requires online TTP support to achieve forward security. (If the signer herself introduces the required asymmetry, then an active attacker compromising the signer can eventually forge the computed signatures). To achieve immediate verification and scalability, BAF uses neither the time factor nor online TTP support.

BAF Strategy: BAF uses a novel 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. To achieve this, BAF signature generation has three phases as described below:

(1) *Individual Signature Generation:* BAF computes the individual signature of each accumulated data item using 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 individual signature), which cannot be forged without knowing its associated secret blinding 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 a large prime q.

In the above construction, the individual signature computation binds a given blinding key pair to the hash of signed data item 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 secret key pair of signer i into a public key pair via an ECC scalar multiplication. Using the corresponding public keys, the verifiers follow the BAF signature verification equation by performing a scalar multiplication for each received data item. The successful verification of the aggregate signature guarantees that only the claimed signer, who possessed the correct blinding secret key pairs before their deletion, could compute such a signature (which is unforgeable after the keys were deleted).

4.2 Description of BAF

Following the syntax given in Definition 3.1, the proposed BAF scheme behaves as described below. The BAF scheme is also illustrated in Figure 1.

(1) *BAF.Kg(p, q, G, L, ID_i)*: $BAF.Kg$ generates L private/public key pairs for signer i. The parameter L determines the maximum number of key update operations that a signer can execute, which should be decided according to the application requirements. In BAF, an offline TTP executes $BAF.Kg$ before the initialization of the scheme, and then provides the required keys to system entities.

  (a) Pick two random numbers as $(a_0, b_0) \overset{R}{\leftarrow} F_q$, which are *initial blinding keys* of signer i. Also pick a random index number as $n \overset{R}{\leftarrow} F_q$, which is used to preserve the order (sequentiality) of individual signatures. Such an order enforcement is needed, since BAF signature aggregation operation is commutative.

  (b) Generate two hash chains from the initial secret blinding keys (a_0, b_0) as $a_{j+1} = H_1(a_j)$ and $b_{j+1} = H_1(b_j)$ for $j = 0, \ldots, L-1$. Also generate a public key for each element of these hash chains as ($A_j = a_j G$ and $B_j = b_j G$) for $j = 0, \ldots, L-1$.

  (c) The TTP provides required keys to signer i and verifiers as follows: $ID_i \leftarrow \{a_0, b_0, p, q, n, L\}$ and Verifiers$\leftarrow \{ID_i : A_0, B_0, \ldots, A_{L-1}, B_{L-1}, p, q, n, L, G\}$.

(2) *BAF.Upd(a_l, b_l)*: $BAF.Upd$ is the key update algorithm, which updates the given blinding keys as $a_{l+1} = H_1(a_l)$ and $b_{l+1} = H_1(b_l)$. $BAF.Upd$ then deletes (a_l, b_l) from the memory. $BAF.Upd$ is invoked after each $BAF.ASig$ operation,

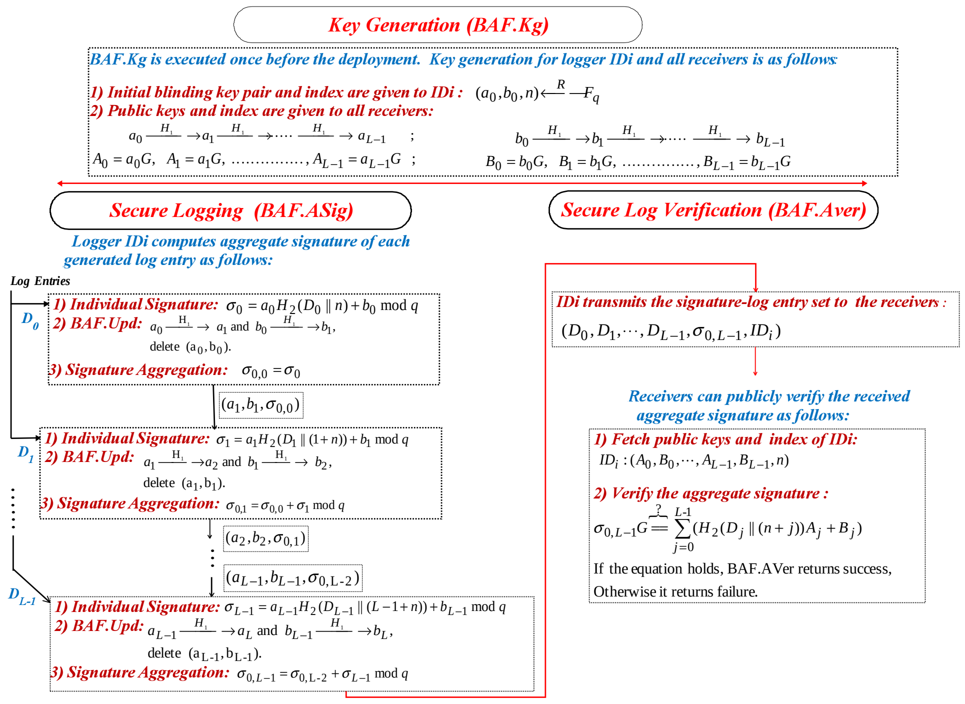
Key Generation (BAF.Kg)
BAF.Kg is executed once before the deployment. Key generation for logger IDi and all receivers is as follows:
1) Initial blinding key pair and index are given to IDi : $(a_0, b_0, n) \xleftarrow{R} F_q$
2) Public keys and index are given to all receivers:
$a_0 \xrightarrow{H_1} a_1 \xrightarrow{H_1} \cdots \xrightarrow{H_1} a_{L-1}$; $b_0 \xrightarrow{H_1} b_1 \xrightarrow{H_1} \cdots \xrightarrow{H_1} b_{L-1}$
$A_0 = a_0G,\ A_1 = a_1G, \ldots, A_{L-1} = a_{L-1}G$; $B_0 = b_0G,\ B_1 = b_1G, \ldots, B_{L-1} = b_{L-1}G$
Secure Logging (BAF.ASig)
Logger IDi computes aggregate signature of each generated log entry as follows:
Log Entries
D_0
1) Individual Signature: $\sigma_0 = a_0 H_2(D_0 \| n) + b_0 \bmod q$
2) BAF.Upd: $a_0 \xrightarrow{H_1} a_1$ and $b_0 \xrightarrow{H_1} b_1$, delete (a_0, b_0).
3) Signature Aggregation: $\sigma_{0,0} = \sigma_0$
$(a_1, b_1, \sigma_{0,0})$
D_1
1) Individual Signature: $\sigma_1 = a_1 H_2(D_1 \| (1+n)) + b_1 \bmod q$
2) BAF.Upd: $a_1 \xrightarrow{H_1} a_2$ and $b_1 \xrightarrow{H_1} b_2$, delete (a_1, b_1).
3) Signature Aggregation: $\sigma_{0,1} = \sigma_{0,0} + \sigma_1 \bmod q$
$(a_2, b_2, \sigma_{0,1})$
$(a_{L-1}, b_{L-1}, \sigma_{0,L-2})$
D_{L-1}
1) Individual Signature: $\sigma_{L-1} = a_{L-1} H_2(D_{L-1} \| (L-1+n)) + b_{L-1} \bmod q$
2) BAF.Upd: $a_{L-1} \xrightarrow{H_1} a_L$ and $b_{L-1} \xrightarrow{H_1} b_L$, delete (a_{L-1}, b_{L-1}).
3) Signature Aggregation: $\sigma_{0,L-1} = \sigma_{0,L-2} + \sigma_{L-1} \bmod q$
Secure Log Verification (BAF.Aver)
IDi transmits the signature-log entry set to the receivers : $(D_0, D_1, \cdots, D_{L-1}, \sigma_{0,L-1}, ID_i)$
Receivers can publicly verify the received aggregate signature as follows:
1) Fetch public keys and index of IDi: $ID_i : (A_0, B_0, \cdots, A_{L-1}, B_{L-1}, n)$
2) Verify the aggregate signature : $\sigma_{0,L-1}G \overset{?}{=} \sum_{j=0}^{L-1} (H_2(D_j \| (n+j))A_j + B_j)$
If the equation holds, BAF.AVer returns success, Otherwise it returns failure.

Fig. 1. BAF algorithms

whose frequency is determined according to the application requirements based on the chosen key update model given in Definition 3.2 (i.e., per-data item or per-interval key update models). Note that *BAF never re-uses* the same blinding key pair to sign two data items.

(3) *BAF.ASig*$(\sigma_{0,l-1}, D_l, a_l, b_l)$: Assume that signer i has accumulated data items $(D_0, \ldots, D_{l-1})$ and computed the aggregate signature $\sigma_{0,l-1}$. Signer i computes the aggregate signature for new data item D_l via $BAF.ASig$ as follows:
   (a) Compute the individual signature σ_l as $\sigma_l = a_l \cdot H_2(D_l||(n+l)) + b_l \bmod q$.
   (b) Fold σ_l into $\sigma_{0,l-1}$ as $\sigma_{0,l} = \sigma_{0,l-1} + \sigma_l \bmod q$, where $l > 0$ and $\sigma_{0,0} = \sigma_0$ is a known value.
   (c) Securely erase $(\sigma_{0,l-1}, \sigma_l)$ from memory and invoke $BAF.Upd(a_l, b_l)$.

(4) *BAF.AVer*$(ID_i, D_0, \ldots, D_l, A_0, \ldots, A_l, B_0, \ldots, B_l, \sigma_{0,l})$: When the verifier receives data items and their associated aggregate signature from ID_i, she first retrieves public keys (A_j, B_j) for $j = 0, \ldots, l$ and n of ID_i. She then verifies $\sigma_{0,l}$ via the BAF verification equation as follows: $\sigma_{0,l}G \overset{?}{=} \sum_{j=0}^{l}(H_2(D_j||(n+j))A_j + B_j)$. If the equation holds, $BAF.AVer$ returns *success*. Otherwise it returns *failure*.

5. FAST-IMMUTABLE BAF (FI-BAF)

In this section, we first identify the problems motivating us to develop an immutable BAF scheme. We then present our second scheme, Fast-Immutable BAF (FI-BAF), which achieves the desired security goals in a computationally efficient way.

5.1 Immutable Secure Audit Logging

In BAF, to achieve minimum signature storage/transmission overhead, only the single-final aggregate signature is kept for the entire logging process. However, despite its advantages, this strategy has important performance drawbacks [Ma and Tsudik 2009]:

—The verification of a particular log entry requires the verification of all log entries, which forces verifiers to perform a large number of expensive operations.

—If the verification of the aggregate signature fails, it is not possible to detect which log entry(ies) is (are) responsible for the failure.

The above concerns can be address by keeping the individual signature for each log entry along with the single-final aggregate signature. In this way, while a particular log entry is verified via its corresponding individual signature, the whole stream is verified via its corresponding single-final aggregate signature. However, this strategy introduces new security problems.

In contrast to the original BAF, when the individual signatures creating the single-final signature are kept instead of erased after used, $\mathcal{A}$ can truncate a set of tail log entries by using the publicly available aggregation function of BAF (*BAF.ASig step (b)*). Assume that a logger computes an aggregate signature $\sigma_{0,l}$ on $(D_0, \ldots, D_l)$ by following *BAF.ASig*, except that she keeps σ_j on D_j for $j = 0, \ldots, l$ (in contrast to *BAF.ASig step-(c)*). In this case, $\mathcal{A}$ can easily remove last m individual signature from the given aggregate signature as $\sigma^*_{0,l-m} = \sigma_{0,l} - \sum_{j=m}^{l} \sigma_j \bmod q$, and then she can truncate their corresponding log entries $D_m, \ldots, D_l$ from the entire set. Such truncation cannot be detected by the verifiers, since $(D_0, \ldots, D_{l-m}, \sigma^*_{0,l-m})$ is still valid under their corresponding public keys.

The problem of deriving "valid" aggregate signatures from existing aggregate and/or individual signatures was first addressed by Mykletun et al. [2004] 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. [2004] suggest two main types of immutable signature mechanisms: (i) *Zero-knowledge proof* techniques (one is interactive and the other is non-interactive) targeting the condensed-RSA schemes; (ii) *Umbrella signature* technique targeting the BLS-based aggregate signature schemes.

To prevent truncation attacks against FssAgg signatures [Ma and Tsudik 2007; Ma 2008], when individual signatures are kept, Ma and Tsudik [2009] adopts *umbrella signature* technique in [Mykletun et al. 2004] to their secure audit logging schemes as the immutable signature mechanism.

In the umbrella signature technique, the signer computes an additional umbrella signature for each *anchor log entry* covering at least m previously accumulated log entries by signing $D_{j-m,j} = D_{j-m}|| \ldots ||D_j$ (ideally all log entries accumulated so far). Each umbrella signature is embedded in the original aggregate signature computed so far, and then is erased from the memory while individual signatures are kept. Since it is computationally infeasible to remove the trace of a data item from its aggregate signature without knowing its corresponding individual signature (discussed in Section 6 in detail), $\mathcal{A}$ can truncate at most m log entries. By setting $m = 1$, we can eliminate the truncation attack with the price of doubling computational overhead at both the signer and verifier sides.

Note that the anchor log entry j ideally should include all the previously accumulated $j-1$ log entries up to its computation, rather than only m previously accumulated log entries as $D_{j-m,j} = (D_{j-m}, \ldots, D_j)$. Addressing this ideal case requires signing the accumulated data as a whole as $D_0, \ldots, D_j$ whenever a new D_j is received. Such an approach further increases the computational cost since it requires hashing larger data for each umbrella signature.

5.2 Description of Fast-Immutable BAF (FI-BAF)

To address the truncation problem in the presence of individual signatures, we develop Fast-Immutable BAF (FI-BAF), which prevents the truncation attacks by incurring only slightly more computational overhead over the original BAF. Instead of following the umbrella strategy [Mykletun et al. 2004] directly as in [Ma and Tsudik 2009], we develop techniques avoiding extra operations in umbrella technique that are not necessary for BAF.

In BAF, direct adaptation of the umbrella strategy for $m = 1$ would double the signing cost as $5H + H' + 2Mulq + 4Addq$ operations per log entry on the signer side, where $Mulq$, $Addq$ and H/H' denote multiplication under modulo q, addition under modulo q and hash operations, respectively. Note that the hash operation used for umbrella signature computation (denoted as H') would require hashing $j-1$ previous accumulated log entries for the jth log entry. This adaptation would also double the verification cost by requiring $2EMul + H + H'$ for per log entry, where $Emul$ denotes ECC scalar multiplication under modulo p. To minimize the computational overhead, FI-BAF uses the following strategies:

— FI-BAF never reuses the same blinding key pair to sign two distinct messages as in the original BAF. Hence, different from iFssAgg schemes [Ma and Tsudik 2009], FI-BAF uses separate key sets to compute the individual and umbrella signatures. Also, unlike iFssAgg schemes [Ma and Tsudik 2009], FI-BAF does not combine individual and umbrella signatures to form the single-final aggregate signature. (Such combination is redundant.) Therefore, the verification of aggregate signature does not require the verification of individual signatures. This reduces the verification overhead substantially.

— FI-BAF keeps the integrity tag to be signed by umbrella signatures small and constant for the entire logging process using MuHASH [Bellare and Micciancio 1997]. That is, the hash of newly received log entry $H_4(D_j||(n+j))$ is combined with the previously computed integrity tag recursively as $D_{0,j} = (D_{0,j-1} \cdot H_4(D_j||(n+j)) \bmod p) \bmod q$, which both preserves the collision-freeness and compactness of the signed integrity tag $D_{0,j}$.

— FI-BAF uses a common key update function to update the separate key sets of individual and umbrella signatures, which substantially reduces the key update cost.

Based on the above strategies, FI-BAF requires only $3(H + Addq) + 2Mulq + Mulp \cong 3(H + Mulq + Addq)$ and $EMul + H + Mulp$ operations to sign and verify each log entry, respectively. Note that the hash operation used for umbrella signature H' is now just the H operation hashing only small and constant data for each log entry.

The proposed FI-BAF scheme is presented below:

(1) *FI-BAF.Kg(p, q, G, L, ID_i)*: *FI-BAF.Kg* generates an initial private key and its corresponding $4L$ public keys for each signer i. An offline TTP executes *FI-BAF.Kg* before the initialization of the scheme as follows:
   (a) Generate the initial private key as $k_0 \overset{R}{\leftarrow} F_{3q}$ and the index number as $n \overset{R}{\leftarrow} F_q$.
   (b) Generate a hash chain from the initial private key k_0 as $k_{j+1} = H_3(k_j)$, where each k_j is split into three sub-keys with $|q|$ bit length as $k_j = (a_j||b_j||y_j)$ for $j = 0, \dots, L-1$. Corresponding public keys of (a_j, b_j, y_j) are computed as $(A_j = a_jG, V_{0,j} = (\sum_{l=0}^{j} a_jb_j)G\ , Y_j = y_jG,\ X_j = (y_ja_j)G)$ for $j = 0, \dots, L-1$.
   (c) The TTP provides required keys to signer i and verifiers as follows: $ID_i \leftarrow \{k_0, n, p, q, G\}$ and Verifiers$\leftarrow$ $\{ID_i : A_j, V_{0,j}, Y_j, X_j, n, p, q, G\}$ for $j = 0, \dots, L-1$.

(2) *FI-BAF.ASig($\sigma_{0,l-1}, D_{0,l-1}, D_l, k_l$)*: When signer i receives new data item D_l, she computes the individual signature γ_l of D_l and the aggregate signature $\sigma_{0,l}$ on $(\sigma_{0,l-1}, D_{0,l-1}, D_l)$ as follows:
   (a) Compute the individual signature as $\gamma_l = y_l \cdot (H_2(D_l||(n+l)) + a_l) \bmod q$,
   (b) Compute the aggregate signature $\sigma_{0,l}$ as $\sigma_{0,l} = \sigma_{0,l-1} + a_l \cdot (D_{0,l} + b_l) \bmod q$ and $D_{0,l} = (D_{0,l-1} \cdot H_4(D_l||(n+l)) \bmod p) \bmod q$ for $l > 0$, where $D_{0,0} = H_2(D_0||n)$ and $\sigma_{0,0} = a_0 \cdot (D_{0,0} + b_0) \bmod q$.
   (c) Securely erase $(\sigma_{0,l-1}, \sigma_l)$ from memory and invoke *FI-BAF.Upd(k_l)*.

(3) *FI-BAF.Upd(k_l)*: Update the private key as $k_{l+1} = H_3(k_l)$, and securely erase $(k_l, a_l\ , b_l, y_l)$ from memory. Similar to the BAF, FI-BAF also *never re-uses* the same key set to sign two distinct data items.

(4) *FI-BAF.AVer(.)*: When the verifier receives $(D_0, \gamma_0, \dots, D_l, \gamma_l, \sigma_{0,l}, ID_i)$ from the ID_i, she can verify an individual log entry $\exists j D_j, 0 \leq j \leq l$ as in *step (a)*, or verify the whole log entry set as in *step (b)* as follows:
   (a) To verify an individual data item $\exists j D_j, 0 \leq j \leq l$, the verifier retrieves the corresponding public key pair (Y_j, X_j) from the memory, and then verifies γ_j of D_j via the following equation: $\gamma_j G \overset{?}{==} H_2(D_j||(n+j))Y_j + X_j$. If the equation holds, $BAF.AVer$ returns *success*. Otherwise it returns *failure*.
   (b) To verify the whole data item set $(D_0, \dots, D_l)$, the verifier retrieves the corresponding public key set (A_j, B_j) for $j = 0, \dots, l$ from the memory, and then verifies $\sigma_{0,l}$ as follows:

$$\sigma_{0,l}G \overset{?}{==} \textstyle\sum_{j=0}^{l}(D_{0,j}A_j) + V_{0,l}, \quad where$$
$$D_{0,0} = H_2(D_0||n), \quad D_{0,j} = (D_{0,j-1} \cdot H_4(D_j||(n+j)) \bmod p) \bmod q, j > 0.$$

   If the equation holds, *FI-BAF.AVer* returns *success*. Otherwise it returns *failure*.

6. SECURITY ANALYSIS

We first prove that BAF is (q_s, τ, ϵ, L)-forward secure-existentially unforgeable against adaptive chosen-message attacks in Theorem 6.1 based on the BAF security model defined in Definition 3.3, as long as Assumption 2.1 holds.

Theorem 6.1 *BAF is (q_s, τ, ϵ, L)-forward secure-existentially unforgeable against adaptive chosen-message attacks in the BAF security model given in Definition 3.3 as long as ECDLP is (τ', ϵ')-hard and H_1/H_2 behave as RO, where*

$$\tau' = \tau + q_s[\tfrac{(q_s+1)}{2}(EMul + H) + H + Mulq + 2Addq] + d,$$
$$\epsilon' \geq \tfrac{\epsilon}{L},$$

$Addq, Mulq, EMul, H$ and d denote the execution times of addition modulo q, multiplication modulo q, ECC scalar multiplication modulo p, FDH operation (for both H_1 and H_2), and the cost of operations executed constant times in the Setup and Forgery phases, respectively.

Proof. Assume that forger $\mathcal{A}$ (q_s, τ, ϵ, L)-*breaks* BAF in forgery time period t_w based on the BAF security model in Definition 3.3. It is then possible to construct a cryptographic simulator $\mathcal{B}$ that solves the ECDLP by extracting the challenge private keys $(a_c, b_c) \overset{R}{\leftarrow} F_q$ from the challenge public keys $(A_c = a_cG, B_c = b_cG)$ via two linear modular equations with probability $\epsilon' \geq \epsilon/L$ in time $\tau' = \tau + q_s[\frac{(q_s+1)}{2}(EMul+H)+H+Mulq+2Addq]+d$ by using $\mathcal{A}$ as a subroutine (subscript c denotes the challenge keys). We then consider the following game, in which $\mathcal{B}$ is given a BAF signing oracle $\mathcal{O}$, and interacts with $\mathcal{A}$ as follows:

(1) *Setup.* Simulator $\mathcal{B}$ is supplied with the challenge public keys (A_c, B_c), global index n, and a forgery time period t_w selected randomly from $w \in [0, L-1]$, for which $\mathcal{A}$ is supposed to output her forgery. $\mathcal{B}$ sets the public key of t_w to the challenge public key as $(A_w = A_c, B_w = B_c)$ hoping that $\mathcal{A}$ produces a successful forgery for this time period that enables $\mathcal{B}$ to solve the *ECDLP* for (A_c, B_c). $\mathcal{B}$ is also allowed to access $\mathcal{O}$, which signs a given message D under secret key pair (a_c, b_c) (only known by oracle $\mathcal{O}$) and returns σ_c to $\mathcal{B}$. After setting the challenge public key and forgery time period, $\mathcal{B}$ computes the remaining private/public keys as follows:
   (a) $\mathcal{B}$ first generates w independent BAF public key pairs as $(A_j = a_jG, B_j = b_jG)$, where each $(a_j, b_j) \overset{R}{\leftarrow} F_q$ for $j = 0, \ldots, w-1$.
   (b) $\mathcal{B}$ then generates the initial blinding keys for t_{w+1} as $(a_{w+1}, b_{w+1}) \overset{R}{\leftarrow} F_q$, and computes the remaining $(L-w)$ public keys using hash chains $a_{j+1} = H_1(a_j)$ and $b_{j+1} = H_1(b_j)$ as $A_j = a_jG$ and $B_j = b_jG$ for $j = w+1, \ldots, L-1$ (as in *BAF.Kg* algorithm). $\mathcal{B}$ provides $\mathcal{A}$ with $(A_0, B_0, \ldots, A_{L-1}, B_{L-1}, p, q, n, G, L)$.

   Note that even though the secret keys generated by $\mathcal{B}$ are not chained with the target secret keys a_c and b_c through H_1, $\mathcal{A}$ will not be able to distinguish them, since otherwise, $\mathcal{A}$ would be able to distinguish the outputs of H_1 from the random uniform distribution (this implies H_1 is broken).

(2) *Queries.* Beginning from $j = 0$, proceeding adaptively, $\mathcal{A}$ requests a BAF signature from $\mathcal{B}$ *once* on a message D_j of her choice, which will be signed under (a_j, b_j). Note that $\mathcal{A}$ can query D_j only once for a given time period, because $BAF.ASig$ never uses the same blinding key pair to sign two distinct messages ($BAF.Upd$ immediately updates and then deletes the blind key pair after each signing operation). For each query $j > 0$, $\mathcal{A}$ also supplies $\sigma_{0,j-1}$ on $(D_0, \ldots, D_{j-1})$ under the distinct $(A'_0 = a'_0G, B'_0 = b'_0G, \ldots, A'_{j-1} = a'_{j-1}G, B'_{j-1} = b'_{j-1}G)$, where both messages and keys are of her choice. $\mathcal{B}$ then handles the query of $\mathcal{A}$ as follows:
   (a) $\mathcal{B}$ first checks whether $(A'_0, B'_0, \ldots, A'_{j-1}, B'_{j-1})$ are valid, then verifies $\sigma_{0,j-1}$ under these public keys via $BAF.AVer$ algorithm. If any of these controls fail, $\mathcal{B}$ *aborts*. Otherwise, $\mathcal{B}$ continues to the next step.
   (b) If $j \neq w$, $\mathcal{B}$ computes $\sigma_j = a_jH_2(D_j||(n + j)) + b_j \bmod q$ (since $\mathcal{B}$ knows (a_j, b_j)). Otherwise, $\mathcal{B}$ goes to $\mathcal{O}$ and requests σ_w under (a_w, b_w) (note that only $\mathcal{O}$ knows $(a_w = a_c, b_w = a_c)$).

   $\mathcal{B}$ computes $\sigma_{0,j} = \sigma_{0,j-1} + \sigma_j \bmod q$ and returns it to $\mathcal{A}$. $\mathcal{B}$ also maintains a list $\mathcal{L} =< D_j, \sigma_j >$ for each computed message-signature pair.

   If $\mathcal{A}$ completes her adaptive queries, she proceeds to the *Break-in phase*. Otherwise, she increases j by once ($j \leq L - 1$), and continues her adaptive queries as described the above. Note that after $\mathcal{A}$ queries for t_j, she cannot later query for any $t_{j'} \leq t_j$.

(3) *Break-in.* If $\mathcal{A}$ breaks-in in t_T for $0 \leq t_T \leq t_w$, then $\mathcal{B}$ *aborts*. Otherwise, $\mathcal{B}$ provides $\mathcal{A}$ with secret keys (a_T, b_T).

(4) *Forgery.* Eventually, $\mathcal{A}$ halts and outputs a forgery $\sigma^*_{0,t'}$ on $(D^*_0, \ldots, D^*_{t'})$ under distinct public keys $(A^*_0, B^*_0\ , \ldots, A^*_{t'}, B^*_{t'})$. To prove the non-triviality of her forgery, $\mathcal{A}$ additionally gives the preceding aggregate signature $\sigma^*_{0,t'-1}$ of $\sigma_{0,t'}$ to $\mathcal{B}$. The forgery of $\mathcal{A}$ is valid if $\sigma^*_{0,t'}$ is non-trivial and valid. That is,
   (a) $(t' = w) \wedge (D^*_{t'} \neq \mathcal{L}.D_w) \wedge (\sigma^*_{t'} \neq \mathcal{L}.\sigma_w) \wedge (A^*_{t'} = A_c, B^*_{t'} = B_c)$; and
   (b) $BAF.AVer(ID_{\mathcal{A}}, D^*_0, \ldots, D^*_{t'}, A^*_0, \ldots, A^*_{t'}, B^*_0, \ldots, B^*_{t'}\sigma^*_{0,t'}) = success$,
   where $\sigma^*_{t'} = \sigma_{0,t'} - \sigma^*_{0,t'-1} \bmod q$.

If both of the above conditions are satisfied, $\mathcal{B}$ proceeds to solve the $ECDLP$ for the challenge public key (A_c, B_c) as follows: Since conditions (a) and (b) are satisfied, $\sigma^*_w = \sigma^*_{t'}$ and $\sigma^*_w G = H_2(D^*_w||(n + w))A_c + B_c$ hold. $\mathcal{B}$ then fetches $\sigma_w = \sigma_c, D_w$ from $\mathcal{L}$ and finds (a_c, b_c) by solving the following modular linear equations:

$$\sigma^*_w = a_cH_2(D^*_w||(n + w)) + b_c \bmod q$$
$$\mathcal{L}.\sigma_w = a_cH_2(\mathcal{L}.D_w||(n + w)) + b_c \bmod q$$

If the above equations do not have a solution, $\mathcal{B}$ *aborts*. Otherwise, $\mathcal{B}$ returns (a_c, b_c).

The success probability and execution time analysis of $\mathcal{B}$ for the above game is summarized as follows:

—If $\mathcal{A}$ succeeds with probability ϵ in forging, then simulator $\mathcal{B}$ succeeds with probability $\sim (\epsilon/L)$. The argument is outlined as follows:

—The view of $\mathcal{A}$ that $\mathcal{B}$ produces the signatures is computationally indistinguishable from the view of $\mathcal{A}$ interacting with a real BAF signing oracle. That is, if there exists a distinguisher for these two views of $\mathcal{A}$, there exists a distinguisher for H_1.

—Conditioned on simulator $\mathcal{B}$ choosing target forgery time period t_w as the period for which $\mathcal{A}$ is supposed to output a valid forgery, the probability that $\mathcal{B}$ solves the ECDLP is the same as the probability that $\mathcal{A}$ succeeds in forgery (i.e., with probability ϵ). Since choosing the "correct" forgery time period t_w occurs with probability $1/L$, the approximate lower bound on the forging probability of $\mathcal{B}$ is $\sim (\epsilon/L)$.

—The running time of simulator $\mathcal{B}$ is that of $\mathcal{A}$ plus the overhead due to handling $\mathcal{A}$'s BAF signature queries. That is, for each query j in the *Query phase*, $\mathcal{B}$ executes $j(EMul + H) + H + Mulq + 2Addq$ operations. In addition to this, $\mathcal{B}$ performs constant number of operations for the *Setup phase* and *Forgery phase* whose costs are $L \cdot EMul + (L - w)H$ and around $(w + 1)(Emul + H) + Addq$, respectively (the cost of forgery condition checks and modular linear equation solving are negligible). Therefore, for total q_s queries $j = 1, \ldots, q_s$, the approximate running time of $\mathcal{B}$ is:

$$\tau' = \tau + q_s[\frac{(q_s + 1)}{2}(EMul + H) + H + Mulq + 2Addq] + d,$$

where $d = (L + w + 1)EMul + (L + 1)H + Addq$ is the total cost of operations that are executed constant times.

This concludes the proof. $\square$

Theorem 6.1 proves that BAF achieves all the required security objectives that a forward secure and aggregate signature scheme must satisfy [Ma and Tsudik 2007; Ma 2008; Ma and Tsudik 2009; 2008; Yavuz and Ning 2009b]: Forward security, unforgeability, integrity, authentication, and signature aggregation.

FI-BAF construction for the aggregate signature computation is equivalent to the BAF construction with the following exceptions: (i) The common key update function H_3 is used to update private keys of individual and aggregate signatures simultaneously, which is the counterpart of H_1 in BAF; (ii) FI-BAF uses MuHASH (Definition 2.2) with H_4 to minimize the size of integrity tag to be signed. Therefore, *Theorem 1 directly applies to FI-BAF as long as $H_2/H_3/H_4$ behaves as RO and MuHASH is collision-free (as assumed in Assumption 2.1).*

Based on the above facts, it is easy to show that both BAF and FI-BAF are secure against against truncation attack in Corollary 6.1.

Corollary 6.1 *In BAF and FI-BAF, for a given $(D_0, \ldots, D_l, \sigma_{0,l})$, truncation of D_l from $D_0, \ldots, D_l$ without being detected is as difficult as producing a valid existential forgery against BAF.*

PROOF. Security of the BAF against truncation attacks follows from the correctness of Theorem 6.1. That is, truncating D_l from $(D_0, \ldots, D_l)$ *without being detected* requires removing its corresponding individual signature σ_l from the given aggregate signature $\sigma_{0,l}$ *without knowing it* (individual signatures are erased from the memory immediately after their computation with step 3-(c)). Removing σ_l from $\sigma_{0,l}$ without knowing it is as difficult as producing an existential forgery against BAF. The argument is outlined as follows:

Assume that $\mathcal{O}$ supplies $\mathcal{A}$ with a valid $(D_j, A_j, B_j, \sigma_{0,l})$ for $j = 0, \ldots, l$, and $\mathcal{A}$ eventually outputs a forgery $\sigma^*_{0,l-1}$ on (D_j, A_j, B_j) truncating D_l from $(D_0, \ldots, D_l)$ where $BAF.AVer(\mathcal{ID}_{\mathcal{A}}, D_j, A_j, B_j, \sigma^*_{0,l-1}) = \mathit{success}$ for $j = 0, \ldots, l-1$. This implies that $\mathcal{A}$ produces a pair $(D^*_l \neq D_l, \sigma^{**}_l)$ on (A_l, B_l) such that $\sigma^*_{0,l-1}G + \sigma^{**}_l G = \sum_{j=0}^{l-1}[H_2(D_j||(n+j))A_j + B_j] + H_2(D^*_l||(n+l))A_l + B_l$ and $\sigma^{**}_l = \sigma_{0,l} - \sigma^*_{0,l-1} \bmod q$. Since $\mathcal{A}$ never queried (D^*_l, σ^{**}_l) to $\mathcal{A}$ before, she produces a valid forgery σ^*_l (and therefore $\sigma^*_{0,l-1}$) involving (A_l, B_l), which is proven to be as difficult as breaking the ECDLP or H_1/H_2 in Theorem 6.1.

In FI-BAF, distinct key sets are used to compute $\gamma_0, \ldots, \gamma_l$ and $\sigma_{0,l}$. Therefore, truncating D_l from $D_0, \ldots, D_l$ in FI-BAF without knowing its corresponding σ_l (erased from the memory after computed) is as difficult as breaking BAF with the condition that MuHASH [Bellare and Micciancio 1997] and H_3/H_4 are also collision-free. The argument is summarized similar to the above as follows:

Suppose that $\mathcal{A}$ is given a valid $(D_j, A_j, B_j, \sigma_{0,l})$ for $j = 0, \ldots, l$, and she eventually outputs a forgery $\sigma^*_{0,l-1}$ on $(D_j, A_j, V_{0,l-1})$ by truncating D_l from $(D_0, \ldots, D_l)$, which is successfully verified under $(A_j, V_{0,l-1})$ for $j = 0, \ldots, l-1$. This implies that $\mathcal{A}$ produces a triple $(D^*_{0,l} \neq D_{0,l}, D^*_l \neq D_l, \sigma^{**}_l)$ on $(A_l, V_{0,l})$ satisfying the following conditions:

$$\sigma^*_{0,l-1}G + \sigma^{**}_l G = \textstyle\sum_{j=0}^{l-1}(D_{0,j}A_j) + D^*_{0,l}A_l + V_{0,l-1} + V_{0,l},$$

$$\sigma^{**}_l = \sigma_{0,l} - \sigma^*_{0,l-1} \bmod q, \ \mathit{and}\ D_{0,l-1} = (\textstyle\prod_{j=0}^{l-1} H_4(D_j||(n+j)) \bmod p) \bmod q,$$

where $D^*_{0,l} = (D_{0,l-1} \cdot H_4(D^*_l||(n+l)) \bmod p) \bmod q$.

Since $\mathcal{A}$ never queried (D^*_l, σ^{**}_l) to $\mathcal{A}$ before, she produces a valid forgery σ^{**}_l (therefore $\sigma^*_{0,l-1}$) based on $(A_l, V_{0,l})$, which is shown to be as difficult as breaking BAF as the above. Note that, finding a collision such that $D^*_l \neq D_l$ but $D^*_{0,l} = D_{0,l}$ simultaneously is by definition as difficult as finding a collision for MuHASH [Bellare and Micciancio 1997] (Definition 2.2) or H_3/H_4 by contradicting Assumption 2.1. □

In addition to being secure against truncation attack, BAF and FI-BAF are also secure against the delayed detection attack: In BAF and FI-BAF, 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.

7. 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; (ii) signature/key storage and communication overheads depending on the size of signing key and the size of signature; (iii) scalability properties such as public verifiability and offline/online TTP, and (iv) security properties such as immediate verification and being resilient to the truncation and delayed detection attacks.

Table I. Notation for performance analysis and comparison

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

FssAgg-BM/AR security parameter x denotes the suggested bit length (i.e., 160 or 512 bits) for the underlying forward-secure signature scheme on which the FssAgg scheme is built [Ma 2008].

Computational, storage and communication overheads are critical to justify the practicality of these schemes for task intensive and/or resource-constrained environments. Scalability and security properties are critical to justify the applicability of these schemes in large distributed systems.

We list the notation used in our performance analysis and comparison in Table I. 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. Note that we accept the per-data item key update model (Definition 3.2) as the comparison basis in our analysis.

7.1 Computational Overhead

We first analyze the computational overhead of BAF and FI-BAF for signing a single log entry. In BAF, signature computation and key update require $(H + Mulq + 2Addq)$ and $2H$, respectively, and therefore $(3H + Mulq + 2Addq)$ in total to sign a single log entry. In FI-BAF, the signature computation (both individual and aggregate signatures) and key update require $(2(H + Mulq) + Mulp + 3Addq)$ and H, respectively, and therefore $3(H + Addq + Mulq)$ in total to sign a single log entry ($Mulp \cong Mulq$). Note that since the overhead of addition and multiplication is negligible, the total cost of signing a single log entry is dominated by $3H$.

We now analyze the signature verification overheads of BAF and FI-BAF. By following the BAF signature verification equation, verifying a single log entry requires one $EMul$, one H and one $EAdd$ (ECC addition operation). 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 $B'_j = \sum_{i=0}^{j} B_j = (\sum_{i=0}^{j} b_j)G$ instead of $B_j = b_j G$ for $j = 0, \ldots, L-1$ to speed up the signature verification. In this way, the verifiers can perform the signature verification with only one $EAdd$ (negligible cost) regardless of the value of l as $\sigma_{0,l-1} * G \overset{?}{==} \sum_{j=0}^{l-1}(H_2(D_j||(n+j)) * A_j) + B'_{l-1}$. Hence, the aggregate signature verification cost of BAF for l received log entries is $(l+1) \cdot (EMul + H)$. In FI-BAF, the aggregate signature verification cost is similar to the BAF except for an additional $Mulp$ operation per log entry due to the MuHASH computation, resulting in a total cost of $(l+1) \cdot (EMul + H + Mulp)$. The individual signature verification cost for FI-BAF is $(Emul + H + EAdd)$.

Comparison: The closest counter parts of our schemes are FssAgg schemes [Ma and Tsudik 2007; Ma 2008; Ma and Tsudik 2008; 2009]. The signature generation of FssAgg-BLS [Ma and Tsudik 2007] 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 [Ma 2008] rely on more efficient PKC operations such as Sqr and $Muln'$. However, these schemes are also computationally costly, since

Table II. Computation involved in BAF, FI-BAF and previous schemes

		Sig	**Upd**	**Ver**
PKC-based	*BAF*	$H + Mulq + 2Addq$	$2H$	$(l+1)(EMul + H)$
	FI-BAF	$3(H + Addq + Mulq)$	H	$(l+1)EMul + l(H + Mulp)$
	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·FssAgg-ASig	FssAgg-Upd	2·FssAgg-AVer
	Logcrypt	$GSig$	-	$GVer$
Symmetric		H	H	$l \cdot H$

Table III. 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	0.006	0.007	1.83/3.66	3.6/7.2	7.71/15.42	1.02	0.006
Ver	0.73	0.75	24.5/49	1.7/3.4	5.3/10.6	1.23	0.006

they require heavy use of such PKC operations. For instance, FssAgg-BM [Ma 2008] requires $x \cdot Sqr + (1 + x/2)Muln'$ (i.e., x$\hat{=}$160 [Ma 2008]) 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 [Ma and Tsudik 2009] double the signing (FssAgg.Asig) and verifying (FssAgg.Aver) costs of their base FssAgg schemes when each log entry is considered as an anchor entry to completely eliminate the truncation attack.

Logcrypt uses a digital signature scheme (e.g., DSA) to sign and verify each log entry separately without signature aggregation [Holt 2006], and thus has standard signature costs. The symmetric schemes [Ma and Tsudik 2007; Schneier and Kelsey 1998; 1999; Bellare and Yee 2003] are in general efficient, since they only need symmetric cryptographic operations.

Table II summarizes the analytical comparison of all these schemes for their computational costs using the notation given in Table I.

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 [Ma and Tsudik 2007], Logcrypt [Holt 2006] (with DSA), and the symmetric schemes [Ma and Tsudik 2007; Schneier and Kelsey 1998; 1999; Bellare and Yee 2003] were measured using implementations based on the MIRACL library [Shamus]. The execution times of FssAgg-AR/BM [Ma 2008] were computed using implementations based on the NTL library [NTL]. Table III 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 305, 600, 1,285, and 170 times faster for loggers, respectively.

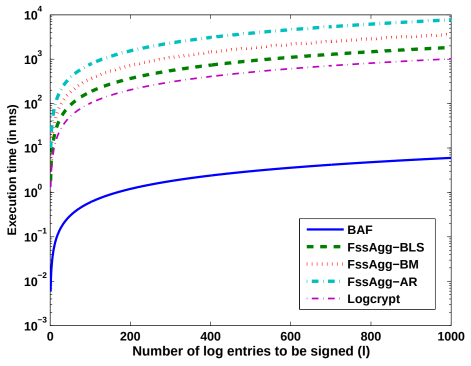

Fig. 2. Signing time comparison of BAF and its counterparts (in ms)

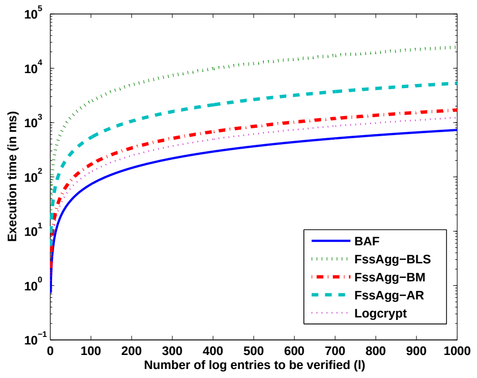

Fig. 3. Verification time comparison BAF and its counterparts (in ms)

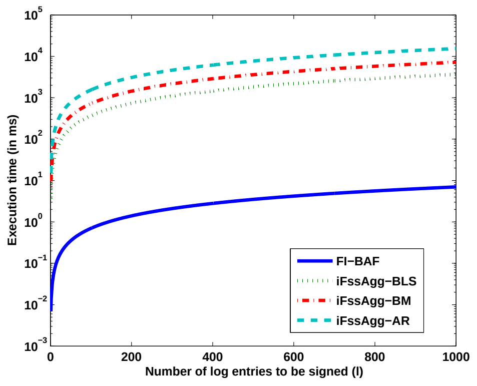

Fig. 4. Signing time comparison of FI-BAF and iFssAgg schemes (in ms)

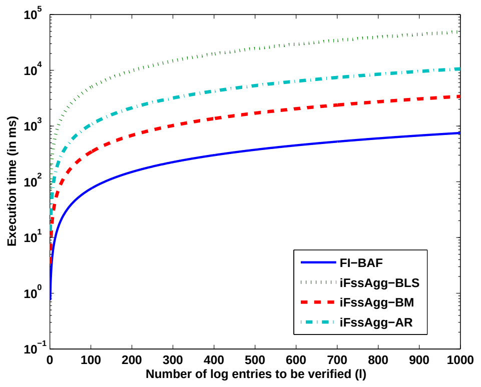

Fig. 5. Verification time comparison FI-BAF and iFssAgg schemes (in ms)

Similarly, when compared with its immutable counterparts (iFssAgg schemes), FI-BAF is also 522, 1,028, and 2,202 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, 7.2, and 1.6 times faster, respectively. Similarly, when compared with its immutable counterparts, FI-BAF is again 65, 4.5, and 14.1 times faster, respectively.

This computational efficiency makes BAF/FI-BAF the best alternative among all existing schemes for secure logging with public verifiability in task intensive and/or resource-constrained applications. Figure 2 and Figure 3 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 and Figure 5 demonstrate the comparison of FI-BAF and iFssAgg schemes [Ma and Tsudik 2009] 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 all these choices.

When compared with the signature generation of previous symmetric logging schemes (e.g., [Ma and Tsudik 2007; Schneier and Kelsey 1998; 1999; Bellare and Yee 2003; 1997]), BAF and FI-BAF signature generation is equally 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 storage/communication efficiency, public verifiability, and certain security properties (e.g., truncation and delayed detection attacks) to achieve this verifier efficiency. Overall, the advantages of BAF and FI-BAF over the symmetric logging schemes include their public verifiability, storage and communication efficiency, and scalability.

7.2 Storage and Communication Overheads

In BAF, the size of signing key is $2|q|$ (e.g., $|q|$=160), and the size of authentication tag is $|q|$. Since BAF allows signature aggregation, independent of the number of data items to be signed, the size of resulting authentication tag is always constant (i.e., $|q|$). Furthermore, BAF derives the current signing key from the previous one, and then deletes the previous signing key from the memory (i.e., evolve-delete strategy [Yavuz and Ning 2009b]). Hence, the size of signing key is also constant, which is equal to $2|q|$. Based on these parameters, both the signature storage and communication overheads of BAF are small and constant (i.e., $3|q|$ and $|q|$, respectively).

In FI-BAF, the size of signing key is $3|q|$, and the sizes of both individual signature (i.e., γ) and aggregate signature (i.e., σ) are $|q|$. Similar to BAF, FI-BAF also uses signature aggregation and evolve-delete strategy, and therefore its aggregate signature and key sizes are small constants as $|q|$ and $3|q|$, respectively. However, to enable selective verification of the individual log entries, FI-BAF keeps their corresponding individual signatures, and therefore its signature storage and communication overheads are both $O(l)|q|$ for l log entries.

Comparison: We use the signature (or key) storage and communication overheads of loggers as the comparison basis. 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 IV summarizes the comparison.

The symmetric schemes (e.g., Bellare-Yee scheme I-II [Bellare and Yee 1997; 2003], Schneier-Kelsey scheme I-II [Schneier and Kelsey 1998; 1999], and FssAgg-MAC in [Ma and Tsudik 2007]) 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). Logcrypt [Holt 2006] extends the idea given in [Schneier and Kelsey 1998; Bellare and Yee 2003] by replacing MAC with a digital signature such as DSA, where the size of signing key is $|q|$ (e.g., 160 bit) and the size of resulting signature is $2|q|$, respectively.

All these schemes incur high signature storage and communication overheads to

Table IV. 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	$2\|q\|$	$3\|q\|$	$\|q\|$	$x\|n\|$	$2\|n\|$	$\|H\|$	*FssAgg*	$\|q\|$	$\|H\|$
Sig. Size	$\|q\|$	$\|q\|$	$\|p\|$	$\|n\|$	$\|n\|$	$\|H\|$	*FssAgg*	$2\|q\|$	$\|H\|$
Storage	$3\|q\|$	$O(l)\|q\|$	$\|p\| + \|q\|$	$x\|n\|$	$3\|n\|$	$O(R)\|H\|$	$O(l) \cdot FssAgg$	$O(l)\|q\|$	$O(l)\|H\|$
Comm.	$\|q\|$	$O(l)\|q\|$	$\|p\|$	$\|n\|$	$\|n\|$	$\|H\|$	$O(l) \cdot FssAgg$	$O(2l)\|q\|$	$O(l)\|H\|$

Table V. 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

the logger. They cannot achieve signature aggregation, and therefore they require storing/transmitting an authentication tag for each log entry. Hence, the signature storage and communication overheads of these symmetric schemes [Bellare and Yee 1997; 2003; Schneier and Kelsey 1998; 1999] and Logcrypt [Holt 2006] are all linear as $O(l) * |H|$ and $O(l) * |q|$, respectively. Different from these schemes, FssAgg-MAC achieves 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 [Ma and Tsudik 2007], FssAgg-BM and FssAgg-AR [Ma 2008] achieve signature aggregation in a publicly verifiable way, and therefore their signature storage/communication overheads are constant. Table IV shows that they are efficient in terms of both the storage and communication overheads. iFssAgg schemes [Ma and Tsudik 2009] demand linear signature storage and communication when compared with their base schemes due to the need of storing and transmitting individual signatures (denoted as $O(l) \cdot FssAgg$ in Table IV).

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., [Bellare and Yee 1997; 2003; Schneier and Kelsey 1998; 1999; Ma and Tsudik 2007; Holt 2006]). BAF is also more efficient than FssAgg-AR/BM [Ma 2008] and as efficient as FssAgg-BLS [Ma and Tsudik 2007], as shown in Table IV. Similar to its immutable counterpart iFssAgg [Ma and Tsudik 2009] schemes, FI-BAF also demands linear signature storage and communication overheads.

7.3 Scalability and Security

BAF and FI-BAF can produce forward secure and aggregate signatures that are publicly verifiable via the signers' corresponding public key sets. Also, BAF and FI-BAF do not need online TTP support for the signature verification, since the verifiers can store all the required keying material without facing a key exposure risk. (Note that in the symmetric schemes such as [Bellare and Yee 1997; 2003; Schneier and Kelsey 1998; 1999], the verifiers cannot store the verification keys on their own memory, since $\mathcal{A}$ compromising a verifier can obtain all the secret keys of all signers.) Furthermore, BAF and FI-BAF do not use the time factor to be publicly verifiable, and therefore achieve immediate verification. Finally, both BAF and FI-BAF are proven to be secure against the truncation and delayed detection attacks. Hence, BAF and FI-BAF achieve all the desirable scalability and security properties simultaneously.

Comparison: Table V 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., [Bellare and Yee 1997; 2003; Schneier and Kelsey 1998; 1999]) cannot achieve public verifiability and require online TTP support to enable log verification. The lack of public verifiability and the requirement for online TTP significantly limit their applicability to large distributed systems. Furthermore, they are vulnerable to both truncation and delayed detection attacks [Ma and Tsudik 2008; 2009]. FssAgg-MAC [Ma and Tsudik 2007] does not need online TTP and is secure against the aforementioned attacks. However, FssAgg-MAC is also a symmetric scheme, which is not publicly verifiable. Hence, none of the previous symmetric schemes can fulfill the requirements of large distributed systems.

PKC-based FssAgg [Ma and Tsudik 2008; Ma 2008; Ma and Tsudik 2007], iFssAgg [Ma and Tsudik 2009] schemes and Logcrypt [Holt 2006] are publicly verifiable. They do not need online TTP support, and can achieve immediate verification. These schemes are also secure against the truncation and delayed detection attacks, with the exception of Logcrypt in [Holt 2006].

BAF and FI-BAF, achieving all the required scalability and security properties, are also much more computational and storage efficient than FssAgg [Ma and Tsudik 2008; Ma 2008; Ma and Tsudik 2007] and iFssAgg [Ma and Tsudik 2009] schemes. Hence, BAF and FI-BAF are the most efficient scheme among the existing alternatives that can achieve all the desirable secure auditing properties simultaneously.

8. RELATED WORK

The pioneering studies addressing the forward secure stream integrity for audit logging were presented in [Bellare and Yee 1997; 2003]. 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 the evolving time periods. Schneier and Kelsey [1998] proposed secure logging schemes that use one-way hash chains together with forward-secure MACs to avoid using tags and indexes. Holt [2006] extended the idea given in [Schneier and Kelsey 1998] to PKC domain by replacing MACs with digital signatures and ID-based cryptography. Finally, Ma et al. proposed a set of comprehensive secure audit logging schemes in [Ma and Tsudik 2008; 2009] based on their forward secure and aggregate signature schemes given in [Ma and Tsudik 2007; Ma 2008]. The detailed analysis and comparison of all these schemes with ours were given in Section 7.

Apart from the above schemes, Chong and Peng [2003] extended the scheme in [Schneier and Kelsey 1998] by strengthening it via tamper-resistant hardware. Moreover, Waters et al. [2004] proposed an audit log scheme that enables encrypted search on audit logs via Identity-Based Encryption (IBE) [Boneh and Franklin 2003]. These works are complementary to ours.

9. CONCLUSION

In this paper, we developed a new class of forward secure and aggregate audit logging schemes for large distributed systems, which we refer to as *Blind-Aggregate-Forward (BAF)* and *Fast-Immutable BAF (FI-BAF)* logging schemes. BAF simultaneously achieves five seemingly conflicting goals for secure audit logging, including very low logger computational overhead, near-zero signature storage/communication overheads, public verifiability (without online TTP support), immediate log verification, and high verifier efficiency. 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 original BAF. Our comparison with the previous alternative approaches demonstrate that BAF and FI-BAF are ideal choices for secure audit logging in large distributed systems, even for task intensive and/or resource constrained environments.

ACKNOWLEDGMENTS

We would like to thank Dr. Di Ma who kindly provided her implementation of FssAgg schemes [Ma and Tsudik 2007; Ma 2008; Ma and Tsudik 2009]. We also would like to thank anonymous reviewers of the preliminary version of this paper [Yavuz and Ning 2009a] for their useful comments.

This work is supported by the U.S. National Science Foundation (NSF) under grants CAREER-0447761 and 0910767, and the U.S. Army Research Office (ARO) under grant W911NF-08-1-0105 managed by NCSU Secure Open Systems Initiative (SOSI). The contents of this paper do not necessarily reflect the position or the policies of the U.S. Government.

REFERENCES

NTL: A library for doing number theory. `http://www.shoup.net/ntl/`.

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

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

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

Bellare, M. and Rogaway, P. 1996. 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)*. 399–416.

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

Bellare, M. and Yee, B. S. 2003. Forward-security in private-key cryptography. In *Proceedings of the The Cryptographers Track at the RSA Conference (CT-RSA '03)*. 1–18.

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

Boneh, D. 1998. The decision diffie-hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium, LNCS*. 48–63.

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

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

Chong, C. N. and Peng, Z. 2003. Secure audit logging with tamper-resistant hardware. In *Proceedings of the 18th IFIP International Information Security Conference*. Kluwer Academic Publishers, 73–84.

Fall, K. 2003. A delay-tolerant network architecture for challenged internets. In *Proc. of the 9th conference on Applications, technologies, architectures, and protocols for computer communications, (SIGCOMM '03)*. ACM, 27–34.

Halperin, D., Kohno, T., Heydt-Benjamin, T., Fu, K., and Maisel, W. 2008. Security and privacy for implantable medical devices. *Pervasive Computing, IEEE 7,* 1, 30–39.

Hankerson, D., Menezes, A., and Vanstone, S. 2004. *Guide to Elliptic Curve Cryptography*. Springer.

Holt, J. E. 2006. 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)*. 203–211.

Krawczyk, H. 2000. Simple forward-secure signatures from any signature scheme. In *Proceedings of the 7th ACM conference on Computer and Communications Security, (CCS '00)*. ACM, 108–115.

Lysyanskaya, A., Micali, S., Reyzin, L., and Shacham, H. 2004. Sequential aggregate signatures from trapdoor permutations. In *Proc. of the 23th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '04)*. Springer-Verlag, 74–90.

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

Ma, D. and Tsudik, G. 2007. Forward-secure sequential aggregate authentication. In *Proceedings of the 28th IEEE Symposium on Security and Privacy (SP '07)*. 86–91.

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

Ma, D. and Tsudik, G. 2009. A new approach to secure logging. *ACM Transaction on Storage (TOS) 5,* 1, 1–21.

Mu, Y., Susilo, W., and Zhu, H. 2007. Compact sequential aggregate signatures. In *Proceedings of the 22nd ACM symposium on Applied computing (SAC '07)*. ACM, 249–253.

Mykletun, E., Narasimha, M., and Tsudik, G. 2004. Signature bouquets: Immutability for aggregated/condensed signatures. In *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS '04)*. Springer-Verlag, 160–176.

Perrig, A., Canetti, R., Song, D., and Tygar, D. 2000. Efficient authentication and signing of multicast streams over lossy channels. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy.*

Schneier, B. and Kelsey, J. 1998. Cryptographic support for secure logs on untrusted machines. In *Proceedings of the 7th conference on USENIX Security Symposium*. USENIX Association.

Schneier, B. and Kelsey, J. 1999. Secure audit logs to support computer forensics. *ACM Transaction on Information System Security 2,* 2, 159–176.

SHAMUS. Multiprecision integer and rational arithmetic c/c++ library (MIRACL). `http://www.shamus.ie/`.

WAGNER, D. 2002. Generalized birthday problem. In *Advances in Cryptology (CRYPTO '02)*. Vol. 2442. Springer-Verlag, 288–303.

WATERS, B., D., DURFEE, G., AND SMETTERS, D. 2004. Building an encrypted and searchable audit log. In *Proceedings of the Network and Distributed System Security Symposium (NDSS '04)*.

YAVUZ, A. A. AND NING, P. 2009a. BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems. In *Proceedings of 25th Annual Computer Security Applications Conference (ACSAC '09)*. 219–228.

YAVUZ, A. A. AND NING, P. 2009b. 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)*.