BAF and FI-BAF: Efficient and Publicly Verifiable Cryptographic Schemes for Secure Logging in Resource-Constrained Systems

ATTILA A. YAVUZ and PENG NING, North Carolina State University
MICHAEL K. REITER, University of North Carolina at Chapel Hill

Audit logs are an integral part of modern computer systems due to their forensic value. Protecting audit logs on a physically unprotected machine in hostile environments is a challenging task, especially in the presence of active adversaries. It is critical for such a system to have *forward security* and *append-only* properties such that when an adversary compromises a logging machine, she cannot forge or selectively delete the log entries accumulated before the compromise. Existing public-key-based secure logging schemes are computationally costly. Existing symmetric secure logging schemes are not publicly verifiable and open to certain attacks.

In this article, we develop a new forward-secure and aggregate signature scheme called *Blind-Aggregate-Forward (BAF)*, which is suitable for secure logging in resource-constrained systems. BAF is the only cryptographic secure logging scheme that can produce publicly verifiable, forward-secure and aggregate signatures with *low computation, key/signature storage, and signature communication overheads for the loggers, without requiring any online trusted third party support*. A simple variant of BAF also allows a fine-grained verification of log entries without compromising the security or computational efficiency of BAF. We prove that our schemes are secure in Random Oracle Model (ROM). We also show that they are significantly more efficient than all the previous publicly verifiable cryptographic secure logging schemes.

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

General Terms: Security, Design

Additional Key Words and Phrases: Applied cryptography, digital signature, secure audit logging, forward security, signature aggregation

ACM Reference Format:
Yavuz, A. A., Ning, P., and Reiter, M. K. 2012. BAF and FI-BAF: Efficient and publicly verifiable cryptographic schemes for secure logging in resource-constrained systems. ACM Trans. Inf. Syst. Secur. 15, 2, Article 9 (July 2012), 28 pages.
DOI = 10.1145/2240276.2240280 http://doi.acm.org/10.1145/2240276.2240280

A preliminary version of this article appeared in ACSAC'09 as Yavuz and Ning [2009a].
This work was 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 for this article do not necessarily reflect the position or the policies of the U.S. Government.
Authors' addresses: A. A. Yavuz and P. Ning, Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206, email: {aayavuz, pning}@ncsu.edu; M. K. Reiter, Department of Computer Science, University of North Carolina at Chapel Hill, FB350 F. P. Brooks Building Campus Box 3175 Chapel Hill, NC 27599; email: reiter@cs.unc.edu.

DOI 10.1145/2240276.2240280 http://doi.acm.org/10.1145/2240276.2240280

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 system activities such as program executions/crashes and data modifications. Providing information about the current and past states of systems, audit logs are invaluable parts of system security. The forensic value of audit logs makes them an attractive target for attackers. For instance, an active attacker compromising a logging machine can modify log entries related to the past, erasing records of the attacker's previous break-in attempts. It is therefore vital for any modern computer system to protect the integrity of audit logs in the presence of active attackers.

Some naive audit-log protection techniques include using 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 a remote trusted entity (to which to upload logs in real-time before the attack occurs). However, as pointed out by some studies (e.g., Ma and Tsudik [2008, 2009] and Crosby and Wallach [2009]), these techniques are impractical for modern computer systems.

First, assuming the presence of tamper-resistant hardware on all platforms is impractical (e.g., wireless sensors [Ma and Tsudik 2007], commercial-off-the-shelf systems). Second, it is not always possible to guarantee the "bug-freeness" of such hardware. For instance, recently introduced Write-Once-Read-Many (WORM) drives[1] were rapidly adopted for secure auditing purposes [Wang and Zheng 2003]; however, some vulnerabilities of WORM drives were later identified [Hsu and Ong 2007; Oprea and Bowers 2009].

Similarly, it is impractical to assume a reliable end-to-end real-time communication channel between a trusted verifier and a logger in all applications (e.g., delay tolerant networks [Fall 2003] and non-real-time applications such as Unattended Wireless Sensor Networks (UWSN)[2] [Yavuz and Ning 2009b]).

The above problems motivate cryptographic mechanisms that can protect audit logs without relying on such assumptions. In the setting where there is neither tamper-resistant hardware nor continuous real-time communication, the untrusted logging machine has to accumulate log entries when the log verifiers are not available. 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).[3] However, the integrity of log entries accumulated before the attack should be protected [Bellare and Yee 2003; Crosby and Wallach 2009; Holt 2006; Ma 2008; Ma and Tsudik 2008, 2009; Yavuz and Ning 2009a] (i.e., forward security property [Abdalla and Reyzin 2000]). Note that this protection should not only guarantee the integrity of individual log entries but also the integrity of the 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) [Ma 2008; Ma and Tsudik 2007, 2009] in addition to forward security.

There are two groups of forward-secure and/or aggregate logging schemes.

— *Symmetric Cryptography-Based Secure Logging Schemes*. These schemes rely on forward-secure Message Authentication Codes (MACs), Pseudo Random Number Generators (PRNGs) (e.g., Bellare and Yee [1997, 2003], Schneier and Kelsey

[1]http://www.emc.com/products/family/emc-centera-family.htm
[2]http://www.darpa.mil/ipto/solicit/baa/BAA-07-46_PIP.pdf
[3]Post-compromise data can only be recovered/controlled if a remote online Trusted Third Party (TTP) or a locally trusted intrusion resilient hardware periodically checks the untrusted machine (e.g., key insulated schemes [Dodis et al. 2002]) [Ma and Tsudik 2009].

[1998]), and one-way hash chains (e.g., Schneier and Kelsey [1998, 1999] and Ma and Tsudik [2007]). Despite their simplicity and computational efficiency, these schemes have limitations.

(i) Due to their symmetric nature, these schemes cannot achieve public verifiability. As a result, they either require a full symmetric key distribution (e.g., FssAgg-MAC [Ma and Tsudik 2007]) or online TTP support. In either case, these schemes cannot address applications that require public auditing and non-repudiation (e.g., logging on electronic voting machines and financial bookkeeping of public companies [Holt 2006; Ma and Tsudik 2008, 2009] in which the signing key must not be revealed). In addition, the online TTP requirement brings architectural difficulties, increases communication overhead, and makes the system vulnerable to single-point-of-failure problems.

(ii) Many of the above schemes incur high storage and communication overheads for the loggers, since they require storing and transmitting an authentication tag or a cryptographic key for each log entry (or logging period) (e.g., Bellare and Yee [1997, 2003] and Schneier and Kelsey [1998, 1999]).

(iii) Many of these schemes (e.g., Bellare and Yee [1997, 2003])[4] have been shown to be vulnerable to the truncation and delayed detection attacks [Ma and Tsudik 2008, 2009] (The details of these attacks are discussed in Section 3 and Section 5.)

— *Public Key Cryptography (PKC)-based Secure Logging Schemes.* Another group of schemes rely on PKC to address these problems. 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/communication overhead and also it is still vulnerable to the truncation attack. Ma and Tsudik proposed FssAgg schemes [Ma 2008; Ma and Tsudik 2007, 2008, 2009]) which use forward-secure signatures and aggregate signatures in an integrated way. These schemes require only a single 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 the "all-or-nothing" property [Ma 2008].

Despite its advantages, the use of only a single aggregate signature to verify the entire set of log entries brings some 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 aggregate signature. This incurs significant computational overheads for the verifiers. Therefore, in certain cases, it is desirable to keep individual signatures along with the aggregate signature to enable the separate verification of individual log entries. However, keeping individual signatures along with the aggregate signature allows the attacker to launch a truncation attack (due to the public and invertible aggregation function). To prevent this, Ma and Tsudik [2009] developed immutable FssAgg (iFssAgg) schemes, which allow a finer-grained verification of log entries via individual signatures while preventing the truncation attack.

All these PKC-based schemes (e.g., Holt [2006], Ma and Tsudik [2007, 2008, 2009], and Ma [2008]) suffer from a common drawback: They require Expensive

[4]These schemes rely on a forward-secure hash chain mechanism to compute the individual MAC of each log entry. They prevent the adversary from truncating non-tail log entries (i.e., beginning and middle entries) from the log stream, since such a truncation requires breaking the hash chain. However, truncating tail-log entries does not require breaking the hash chain, and therefore the adversary can truncate tail-log entries from the log stream without being detected.

Operations (ExpOps)[5] to compute and verify the signatures, which make them computationally costly. Hence, these schemes are impractical for secure logging in resource-constrained devices.

1.1. Our Contribution

In this article, we develop a new class of forward-secure and aggregate secure audit logging schemes for resource-constrained systems, which we call *Blind-Aggregate-Forward (BAF)* and *Fast-Immutable BAF (FI-BAF)* logging schemes.

We summarize the properties of our schemes as follows:

(1) *High Logger (Signer) Efficiency*. BAF is the only PKC based scheme among its counterparts that achieve the high (i.e., ExpOp-free) signer efficiency:
  - *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 and is much more efficient than all existing PKC-based secure logging 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. Thus, BAF is more signature storage/bandwidth efficient than some existing schemes that require linear key and signature storage/transmission.

(2) *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 existing PKC-based secure logging schemes.

(3) *Offline TTP and Immediate Verification*. Unlike some previous schemes (e.g., Schneier and Kelsey [1998, 1999] and Yavuz and Ning [2009b]), BAF does not need online TTP support to enable log verification. Hence, it eliminates the 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 points of failure. Moreover, BAF achieves immediate verification, and therefore is secure to the delayed detection attack.(Secure audit logging schemes that cannot achieve immediate verification property are vulnerable to delayed detection attacks. Details of this attack are discussed in Section 5.)

(4) *Fast and Immutable Logging*. Our extended scheme FI-BAF addresses the need for a BAF variant that allows the verification of a particular log entry without compromising the security and computational efficiency of the original BAF.

(5) *Public Verifiability*. The BAF schemes produce publicly verifiable signatures, and therefore, in contrast to the symmetric schemes, are suitable for applications requiring public auditing and non-repudiation.

(6) *Provable Security*. Unlike some previous schemes [Bellare and Yee 2003; Holt 2006; Schneier and Kelsey 1998, 1999], our schemes are secure against both truncation and delayed detection attacks. Moreover, instead of relying on heuristic security arguments against truncation attacks (e.g., Ma and Tsudik [2007] and Ma [2008] and the preliminary version of this article [Yavuz and Ning 2009a]), we formally prove that our schemes are secure against truncation attacks in Random Oracle Model (ROM) [Bellare and Rogaway 1993].

[5]For brevity, in this article, we refer to an expensive cryptographic operation such as modular exponentiation [Stinson 2002] and pairing [Mass 2004] as an ExpOp.

Table I. Comparison of BAF Schemes with Their Counterparts in Terms of Asymptotic Computational/Storage/Communication Overheads and Some Qualitative Properties

Criteria	*PKC-based*				*Symmetric*
	BAF/FI-BAF	FssAgg/iFssAgg		Logcrypt	
Computational		BLS	BM and AR		
Sig	H	$ExpOp + H$		$ExpOp + H$	H
Upd	H	$ExpOp + H$		-	H
Ver	$O(l \cdot (ExpOp + H))$				$O(l \cdot H)$
Kg	$O(L \cdot (ExpOp + H))$				$O(L \cdot H)$
Online Comm.	$\lvert\sigma\rvert$	$\lvert\sigma\rvert$		$O(l \cdot \lvert\sigma\rvert)$	$O(l \cdot \lvert H\rvert)$
Offline Comm.	$O(L \cdot \lvert K\rvert)$	$O(L \cdot \lvert K\rvert)$	$\lvert K\rvert$	$O(L \cdot \lvert K\rvert)$	$\lvert K\rvert$
Size of PK	$O(L)$	$O(L)$	$O(1)$	$O(L)$	-
Storage *Signer*	$\lvert K\rvert + \lvert\sigma\rvert$	$\lvert K\rvert + \lvert\sigma\rvert$		$O(L \cdot (\lvert K\rvert + \lvert\sigma\rvert))$	$O(L \cdot \lvert H\rvert)$
Storage *Verifier*	$O(L \cdot \lvert K\rvert)$	$O(L \cdot \lvert K\rvert)$	$O(\lvert K\rvert)$	$O(L \cdot \lvert K\rvert)$	$O(\lvert K\rvert)$
Public Ver.	Y	Y			N
Online TTP	Y	Y			N
Immediate Ver.	Y	Y			N
Delayed Det. A.	Y	Y			N
Truncation A.	Y	Y		N	N
Security	Provable	Heuristic (informal)		N	N

*Table I demonstrates the asymptotic costs of processing data items for each compared scheme. H, $|H|$, $|K|$ and $|\sigma|$ denote the cost of single hash operation, the bit length of hash output, the bit length of private/public key and the signature bit length, respectively ($|K|$ and $|\sigma|$ slightly vary for each scheme). Signature generation and key update costs are given for a single data item. Key generation cost is given for the maximum number of data items (i.e., L). Signature verification cost is given for $0 < l < L$ data items. Storage and communication costs are based on the cryptographic overhead introduced by the schemes (data overheads are linear and the same for all compared schemes). Offline communication overhead is the transmission overhead of keying material from key generation center to the verifiers. This transmission occurs only once before the deployment in an offline manner. Online communication overhead denotes the communication overhead that stems from the transmission of signatures, which occurs online after the deployment.

**BLS, BM and AR abbreviate FssAgg-BLS [Ma and Tsudik 2007], FssAgg-BM and FssAgg-AR in [Ma 2008], respectively. The column "Symmetric" refers symmetric cryptography based counterparts of BAF (e.g., Schneier and Kelsey [1998, 1999] and Bellare and Yee [2003]).

†BAF schemes are the only PKC-based alternative that achieve the high signer efficiency (i.e., ExpOp-free signing and constant signature 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.

Table I outlines these properties and compares the proposed schemes with their counterparts.

This article, in addition to introducing the extended scheme FI-BAF, also gives a new security model that formally captures the truncation attacks, an improved security analysis with tighter bounds and probability analysis, and a more comprehensive performance analysis than its preliminary version that appeared in Yavuz and Ning [2009a].

The remainder of this article is organized as follows. Section 2 presents our notation and preliminary definitions. Section 3 provides the models used in this paper. Section 4 describe the proposed schemes in detail. Section 5 gives the security analysis of BAF. Section 6 presents performance analysis of our schemes and compares them with previous approaches. Section 7 briefly discusses related work, and Section 8 concludes this article.

2. PRELIMINARIES

This section provides our notation and basic definitions.

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. $a \overset{\$}{\leftarrow} \mathcal{S}$ denotes that the value of variable a is randomly and uniformly selected from set $\mathcal{S}$. For any integer l, $(a_0,\ldots,a_l) \overset{\$}{\leftarrow} \mathcal{S}$ means $(a_0 \overset{\$}{\leftarrow} \mathcal{S},\ldots,a_l \overset{\$}{\leftarrow} \mathcal{S})$. $\mathcal{A}^{\mathcal{O}_0,\ldots,\mathcal{O}_i}(.)$ 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)* [Bellare and Rogaway 2005], which is defined here.

Definition 2.1. Given a cyclic group G of order prime q and a generator α of G, let $\mathcal{A}$ be an algorithm that returns an element of Z_q^*.

Experiment $Expt^{DL}_{G,\alpha}(\mathcal{A})$
$y \overset{\$}{\leftarrow} \mathbb{Z}_q^*$, $Y \leftarrow \alpha^y \bmod p$,
$y' \leftarrow \mathcal{A}(Y)$,
If $\alpha^{y'} \bmod p = Y$, return 1, else, return 0.

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

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

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

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

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

3. MODELS

In this section, we first 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*.

3.1. 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.

3.1.1. FSI Model. FSI model is the classic tamper-evident audit logging model initially introduced by Bellare and Yee [1997] in the context of symmetric key cryptography, which was later formalized in Bellare and Yee [2003]. 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 [1999] followed 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 extended the basic FSI model into the PKC domain. Later, Ma and Tsudik [2007] followed 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.

3.1.2. 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., Ma and Tsudik [2007, 2008, 2009] and Ma [2008]). 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, since each logger is responsible for only her own audit logs.

3.2. 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., Krawczyk [2000]). 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 aggregate signatures (e.g., Boneh et al. [2003] and Boldyreva et al. [2007]) and then uses the key update algorithm to update the private key.

Definition 3.1. 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, 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.

3.3. Threat and Security Model

Our threat 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 3.2. Moreover, we provide a formal treatment for the truncation attacks via the *truncation experiment (TRUNC)* defined in Definition 3.3 based on the *signature extraction* argument [Boneh et al. 2003; Coron and Naccache 2003]. 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 $RO(.)$ from which she can request the hash of any message D of her choice up to L' messages. Note that in our proofs (see Section 5), cryptographic hash function H used in our schemes is modeled as a random oracle [Bellare and Rogaway 1993] via $RO(.)$.

(ii) *Signing Oracle*. $\mathcal{A}$ is provided with a *signing oracle* $FAS.Sig_{sk}(.)$. For each batch query i, $\mathcal{A}$ can query the $FAS.Sig_{sk}(.)$ oracle on a set of messages $\overrightarrow{D}_i = (D_{j'}|| \ldots ||D_j)$, $j \geq j'$, of her choice. $FAS.Sig_{sk}(.)$returns a forward-secure and aggregate signature $\sigma_{0,i}$ under sk on $(\overrightarrow{D}_0|| \ldots ||\overrightarrow{D}_i)$ (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 as described, until she decides to "break-in".

(iii) *Break-In Oracle*. $\mathcal{A}$ can invoke the *Break-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-in* oracle returns $(l + 1)$-th private key to $\mathcal{A}$ (if $l = L$, then *Break-in* oracle rejects the query, since all private keys were used).

Definition 3.2. *FAEU-CMAexperiment* 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 mth batch query (a vector), and $\overrightarrow{D}^*[n]$ denotes the nth 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

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

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

Definition 3.3. *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 $(FAS.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1) \wedge \neg(\exists I \subseteq \{0, \ldots, i\} : \overrightarrow{D}^* = ||_{m \in I} \overrightarrow{D}_m)$ holds, then return 1, else, return 0, where $\overrightarrow{D}_m$ denotes mth batch query (a vector) in $\overrightarrow{D} = (\overrightarrow{D}_0 || \cdots || \overrightarrow{D}_i)$.

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

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

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

Note that the nontriviality condition of the above experiment captures the truncation attacks. The nontriviality condition requires that $\mathcal{A}$ must output a forgery that is not comprised of a combination of batch queries she made to the signature oracle. That is, in order to win the truncation experiment, $\mathcal{A}$ must split an aggregate signature that she queried to the signature oracle without querying its subaggregate (complementary) signatures. We explain this condition based on the *signature extraction argument* in following discussion.

3.4. 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., Ma and Tsudik [2007, 2008, 2009], Ma [2008], and Yavuz and Ning [2009a]) 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 an "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 [Ma 2008; Ma and Tsudik 2007, 2009]. 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 [Boneh et al. 2003; Coron and Naccache 2003]). 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 [Ma and Tsudik 2007, 2009]. 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 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 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. BLIND-AGGREGATE-FORWARD (BAF) SCHEMES

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

4.1. Overview

All previous PKC-based FAS constructions are directly derived from existing aggregate or forward-secure signature schemes. For instance, FssAgg-BLS [Ma and Tsudik 2007] is derived from the aggregate signature scheme given in Boneh et al. [2003]. Similarly, FssAgg-BM and FssAgg-AR in Ma [2008] and Ma and Tsudik [2008, 2009] are derived from the forward secure signatures given in Bellare and Miner [1999] and Abdalla and Reyzin [2000], 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.

— *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 modulo 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 random 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 modulo q.

In this 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 a 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 a modular exponentiation for each received data item.

4.2. Description of BAF

The proposed BAF scheme is.

(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 $q|(p-1)$. Also generate a generator α of the subgroup G of order q in Z_p^*. H is a cryptographic hash function, that is, $H : \{0,1\}^* \rightarrow Z_q^*$.

  (a) Generate the initial key pair as $(a_0, b_0) \overset{\$}{\leftarrow} Z_q^*$, 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 $j = 0, \ldots, L-1$.

  (b) Generate two master seeds $(x, x') \overset{\$}{\leftarrow} Z_q^*$. 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 $j = 0, \ldots, L-1$ and $u_j{}' \leftarrow k_{j-1} + H(k_j) \bmod q$ for $j = 1, \ldots, L-1$.

(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 the signer is as follows:
$sk \leftarrow (0, \langle a_0, b_0\rangle, x, x')$ 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 $sk_j = (j, \langle a_j, b_j\rangle, x, x')$, if $j \geq L-1$, then return $\perp$ (i.e., invalid input). Otherwise, update the private key as $sk_{j+1} \leftarrow (j+1, \langle a_{j+1}, b_{j+1}\rangle, x, x')$, 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.[6]

(3) $\underline{\sigma_{0,l} \leftarrow BAF.Sig(sk_j, \overrightarrow{D})}$. Given the private key $sk_j = (j, \langle a_j, b_j\rangle, x, x')$, 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 empty), 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 $(m+1, \langle a_{m+1}, b_{m+1}\rangle, x, x') \leftarrow BAF.Upd(\langle m, \langle a_m, b_m\rangle, x, x'\rangle, L)$ 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 (note that (a_j, b_j) were erased by *BAF.Upd* algorithm). 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) $\underline{b \leftarrow BAF.Ver\left(PK, \overrightarrow{D}, \sigma_{0,l}\right)}$. Recall that $PK = \left(\{A_j, B_j, u_j\}_{j=0}^{L-1}, \{u'_j\}_{j=1}^{L-1}, p, q, \alpha, L\right)$. Given $\overrightarrow{D} = (D_0, \ldots, D_l)$, if this 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 $j = l, \ldots, 1$, $r_j \leftarrow u_j - k_j \bmod q$ for $j = l, \ldots, 0$ and $\sigma_{0,l} = \langle s_{0,l}, k_l\rangle$.

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 via $s_{0,l}$ as $\alpha^{s_{0,l}} \bmod p$. 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 values $\{A_j = \alpha^{a_j} \bmod p, B_j = \alpha^{b_j} \bmod p\}_{j=0}^{l}$, correctly constructs $s_{0,l}$ on the exponent. The correctness of the BAF follows as:

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

Remark. Different from our preliminary version [Yavuz and Ning 2009a], 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

[6]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).

constructed in *FAEU-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 5.

We use master seeds (x, x'), tokens (u, u') and masking keys k_j 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 $\sigma_{0,l} = \langle s_{0,l}, k_l \rangle$, verifiers can recover all the previous random numbers $r_0, \ldots, r_l$ from (u_l, u'_l) via k_l in a computationally efficient way. Note that the release of these random numbers does not hurt the security, since they are required to be a part of private key only before the signature generation. After being signed, random numbers just become a part of the signature and therefore can be publicized. 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. Hence, seed and token mechanisms offload the storage of random numbers to the verifiers while keeping them secret until they need to be publicized. Based on the above strategies, the signer avoids storing/transmitting a random number for each data item (i.e., linear storage), 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. Fast-Immutable BAF (FI-BAF)

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

However, this strategy also has certain drawbacks [Ma and Tsudik 2009]: (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. [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) for condensed-RSA schemes; (ii) *Umbrella signature* technique for the BLS-based aggregate signature schemes (e.g., Boneh et al. [2003]). These constructions are designed for the generic *distinct-signer-distinct-message model*.[7]

To prevent the truncation attacks against FssAgg signatures [Ma 2008; Ma and Tsudik 2007], when individual signatures are kept, Ma and Tsudik [2009] adopt the *umbrella signature* technique in Mykletun et al. [2004] in 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.

4.3.1. Fast-Immutable BAF (FI-BAF). To address the above problem, we give a simple variant of 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 3.1.2), 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 [Mykletun et al. 2004], which are designed for the *distinct-signer-distinct-message model*.

In FI-BAF, the "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 accompanied with a random number n, which is signed along with an index j incrementally, and is specific to each signer.

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 second (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 [Ma and Tsudik 2009], 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.

[7]In distinct-signer-distinct-message model, each signer in the system computes a signature on a (distinct) message, and these signatures can be aggregated by any entity (e.g., secure routing applications [Boldyreva et al. 2007; Boneh et al. 2003; Mu et al. 2007]). However, all existing forward-secure and aggregate secure logging schemes follow the same-signer-distinct-message model, in which 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]).

5. SECURITY ANALYSIS

In the random oracle model [Bellare and Rogaway 1993], we prove that BAF is a *FAEU-CMA* signature scheme in Theorem 5.1. Note that in our proofs, we ignore terms that are negligible in terms of κ.

THEOREM 5.1.

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

where $O(t') = O(t + L \cdot \kappa^2 + L' \cdot RNG)$ *(in the random oracle model).*

PROOF. Let $\mathcal{A}$ be a *BAF attacker* and recall that ($y \overset{\$}{\leftarrow} \mathbb{Z}_q^*, Y \leftarrow \alpha^y \bmod p$) as defined in *DL-experiment* (i.e., Definition 2.1). We construct a *DL-attacker* $\mathcal{F}$ that uses $\mathcal{A}$ as a sub-routine as follows:

<u>*Algorithm F*(Y)</u>

<u>*Setup*</u>. $\mathcal{F}$ randomly chooses a target forgery/signature extraction index $w \overset{\$}{\leftarrow} [0, L-1]$, for which $\mathcal{A}$ is supposed to output her forgery. $\mathcal{F}$ generates BAF private/public keys, and then embeds the target discrete-log value Y into the w-th BAF public values (i.e., (A_w, B_w)) via a simulation, hoping that $\mathcal{A}$ produces a successful forgery on them. If this occurs then $\mathcal{F}$ extracts y from Y with a non-negligible probability.

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

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

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

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

<u>Execute $\mathcal{A}^{RO(.), FAS.Sig_{sk}(.), Break\text{-}in}(PK)$</u>. $\mathcal{F}$ maintains three lists $\mathcal{HL}$, $\mathcal{LD}$, and $\mathcal{LS}$ to keep track the query results during the experiment, all initially empty. $\mathcal{HL}$ is a hash list in a form of tuples $(\overline{D}_j, h_j)$, where $\overline{D}_j$ and h_j denote the jth data item queried to $RO(.)$ and its corresponding $RO(.)$ answer, respectively. $\mathcal{HL}[j, 0]$ and $\mathcal{HL}[j, 1]$ denote the access to the element $\overline{D}_j$ and h_j, respectively. $\mathcal{LD}$ is a data list, in which each of its elements $\mathcal{LD}[i]$ 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 $FAS.Sig_{sk}(.)$.

— <u>Queries</u>. $\mathcal{A}$ queries the $FAS.Sig_{sk}(.)$ oracle on up to L messages of her choice, and then queries the *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 to Queries to RO(.) Oracle*. $\mathcal{F}$ executes the function $H\text{-}Sim(\delta)$ that works as follows: If $\exists k : \delta \in \mathcal{HL}[k, 0]$ then return $\mathcal{HL}[k, 1]$. Otherwise, return $h \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ as the answer, insert the new tuple (δ, h) to $\mathcal{HL}$ as $(\mathcal{HL}[l', 0] \leftarrow \delta, \mathcal{HL}[l', 1] \leftarrow h)$, and then update $l' \leftarrow l' + 1$. That is, cryptographic hash function H used in BAF is modeled a random oracle.

— *How to Respond to ith FAS.Sig*$(D_{j'}, \ldots, D_j)$ *Query*.
  — 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 $r_m \leftarrow H\text{-}Sim(x||m)$.[8] 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 $s_{j',j} \leftarrow [\sum_{j' \le m \le j, m \ne w}(a_m h_m + b_m)] + \gamma \bmod q$, where $h_m \leftarrow H\text{-}Sim(D_m||r_m||m)$. Insert $(D_w||r_w||w, z)$ 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 ith 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 to Queries to the 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 $l \le w$, then *abort* and return 0. Otherwise, give $\xi \leftarrow \langle \{m, a_m, b_m\}_{m=l+1}^{L-1}, x, x' \rangle$ to $\mathcal{A}$.

— <u>Forgery</u>. Finally, $\mathcal{A}$ outputs a forgery for PK as $(\overrightarrow{D}^*, \sigma^* = \langle s^*_{0,e}, k^* \rangle)$.

By Definition 3.2, $\mathcal{A}$ wins if $BAF.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1$ and $\exists n \in \{0, \cdots, l\} : \overrightarrow{D}^*[n] \notin (\mathcal{LD}[0]||\cdots||\mathcal{LD}[i])$ holds (recall that each $\mathcal{LD}[m], 0 \le m \le i$, is a batch query (a data vector), and $\overrightarrow{D}^*[n]$ is the nth 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:

<u>*Extraction*</u>. 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 $\gamma^* \leftarrow s^*_{0,e} - \sum_{0 \le v \le e, v \ne w}(a_v H\text{-}Sim(\overrightarrow{D}^*[v]||r_v||v) + b_v) \bmod q$.

Recall that $B_w \equiv (A_w^z)^{-1}\alpha^\gamma \bmod p$ 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 these modular linear equations (note that only unknowns are y' and b_w):

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

Note that $Y \equiv \alpha^y \bmod p$ holds, since $\mathcal{A}$'s forgery is valid and nontrivial on Y. Therefore, by Definition 2.1, $\mathcal{F}$ wins the *DL-experiment*.

[8]Recall 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 $\mathcal{HL}$.

The success probability and execution time analysis of this 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 $Adv_{G,\alpha}^{DL}(t')$ 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 following 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 $0 \le l \le L-1$ data items and then queries the *Break-in* oracle. If $l \le w$, 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*). Hence, this occurs with $Pr[Forge|\overline{Abort1}] = Adv_{BAF(p,q,\alpha)}^{FAEU\text{-}CMA}(t, L', L)$.
— *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 $w \le |\overrightarrow{D}^*| = e \le l$, this occurs with a probability at least $Pr[\overline{Abort2}|\overline{Abort1} \wedge Forge] \ge 1/l$. 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:

$$Adv_{BAF(p,q,\alpha)}^{FAEU\text{-}CMA}(t, L', L) \le L \cdot Adv_{G,\alpha}^{DL}(t')$$

— *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 RNG. 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 $O(t') = O(t + L \cdot \kappa^2 + L' \cdot RNG)$.
— *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-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 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,

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

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 $(y, z, \gamma) \overset{\$}{\leftarrow} \mathbb{Z}_q^*$ 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. Furthermore, (y, z, c) are used identically as (a_w, h_w, b_w) are used in the real-system to compute required values (i.e., $\gamma = s_w$). Therefore, the joint probability distributions $Pr[\overrightarrow{A}_{real} = \overrightarrow{a}\,] = Pr[\overrightarrow{A}_{sim} = \overrightarrow{a}\,]$ (i.e., perfectly indistinguishable). □

We prove that BAF is secure against the truncation attacks in Theorem 5.2 (in the random oracle model).

THEOREM 5.2.

$$Adv_{BAF(p,q,\alpha)}^{TRUNC}(t, L', L) \le \frac{L^2}{L-1} \cdot Adv_{G,\alpha}^{DL}(t'),$$

where $O(t') = O(t + L\kappa^2 + L' \cdot RNG)$ *(in the random oracle model).*

PROOF. Let $\mathcal{A}$ be a *BAF attacker* and recall that $(y \overset{\$}{\leftarrow} \mathbb{Z}_q^*, Y \leftarrow \alpha^y \bmod p)$ as defined in *DL-experiment* (i.e., Definition 2.1). We construct a *DL attacker* $\mathcal{F}$ that uses $\mathcal{A}$ as a sub-routine as follows:

Algorithm $F(Y)$

Setup. $\mathcal{F}$ randomly chooses a target forgery/signature extraction index $w \overset{\$}{\leftarrow} [0, L-1]$, for which $\mathcal{A}$ is supposed to output her forgery. $\mathcal{F}$ generates BAF public private/public keys, and then embeds the target discrete-log value Y into the wth and $(w + 1)$-th BAF public values (i.e., (A_w, B_{w+1})) via a simulation using a batch (aggregate) signature s', hoping that $\mathcal{A}$ produces a successful forgery on them. That is, if $\mathcal{A}$ splits s' into two valid individual signatures without querying these individual signatures to the signature oracle, then $\mathcal{F}$ can extract y from Y with a nonnegligible probability.

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

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

—*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:

—$A_w \leftarrow Y,\ s' \xleftarrow{\$} \mathbb{Z}_q^*$,

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

—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 to Queries to RO(.) Oracle*. $\mathcal{F}$ executes the function *H-Sim*(.) that is defined in Theorem 5.1.

—*How to Respond to ith FAS.Sig*$(D_{j'}, \ldots, D_j)$ *Query*.

—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 $r_m \leftarrow H\text{-}Sim(x||m)$. 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 $s_{j',j} \leftarrow [\sum_{j'\le m\le j, m\ne w, m\ne w+1}(a_m h_m + b_m)] + s' \bmod q$, 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 with $\sigma_{0,j}$, 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 to 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 $l \le w+1$, 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 for PK as $\left(\overrightarrow{D}^*, \sigma^* = \langle s_{0,e}^*, k^*\rangle\right)$.

By Definition 3.3, $\mathcal{A}$ wins if $(FAS.Ver(PK, \overrightarrow{D}^*, \sigma^*) = 1) \wedge \neg(\exists I \subseteq \{0, \ldots, i\} : \overrightarrow{D}^* = ||_{m\in I}\mathcal{LD}[m])$ holds. Recall that i denotes the total number of batch queries $\mathcal{A}$ made to $FAS.Sig_{sk}(.)$ oracle, and $\mathcal{LD}[m]$, $0 \le m \le i$, denotes kth 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 3.4 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 5.1. Therefore, we only analyze the tail-truncation case as follows:

$\mathcal{A}$ queried the $FAS.Sig_{sk}(.)$ oracle on $l \le L-1$ data items and then queried the *Break-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 $s_{0,l} \equiv s_{0,e} + s_{e+1,l} \bmod q$ $(0 < e < l$ 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 $\overrightarrow{D}^* = (D_0, \ldots, D_e)$ 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,

$$s' \equiv \underbrace{y \cdot z_0 + b_w}_{s'_0} + \underbrace{a_{w+1} \cdot z_1 + b_{w+1}}_{s'_1} \bmod q,$$

where (y, b_{w+1}) are the unknowns (and so the individual signatures s'_0 and s'_1 are also unknown). Therefore, $\mathcal{F}$ checks if $e = w$ and $\overrightarrow{D}^*[w] = D_w$. That is, whether $\mathcal{A}$ 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:

Due to the validity condition, the forged (extracted) aggregate signature $s^*_{0,w}$ 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 as shown in this equation. Hence, $\mathcal{F}$ first isolates s'_0 from $s^*_{0,w}$ as $s'_0 \leftarrow s^*_{0,w} - \sum_{v=0}^{w-1}(a_v H\text{-}Sim(\overrightarrow{D}^*[v]||r_v||v) + b_v) \bmod q$ (note that $e = w$). Note that since $s^*_{0,w}$ is valid on *PK*, s'_0 is also valid on (A_w, B_w). Hence, the equation $\alpha^{s'_0} \equiv A_w^{z_0} B_w \bmod p$ holds. $\mathcal{F}$ knows (z_0, b_w) and therefore he can extract y by solving the following equation,

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

If $\mathcal{F}$ does not abort, then $Y \equiv \alpha^y \bmod p$ holds, since $\mathcal{A}$'s forgery is valid and nontrivial on Y. Therefore, by Definition 2.1, $\mathcal{F}$ wins the *DL-experiment*.

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

— *Success Probability Analysis*. The probability $Adv^{DL}_{G,\alpha}(t')$ is as follows:
  - — $\overline{Abort1}$. $\mathcal{F}$ does not abort as a result of $\mathcal{A}$'s queries.
  - — *Forge*. $\mathcal{A}$ wins the *TRUNC 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 following events occurs:
  (i) $\mathcal{A}$ queries the $FAS.Sig_{sk}(.)$ oracle on $l \leq L - 1$ data items and then queries the *Break-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 requests 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* that follows). Hence, this occurs with $Pr[Forge|\overline{Abort1}] = Adv^{TRUNC}_{BAF(p,q,\alpha)}(t, L', L)$.

— *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:

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

— <u>*Execution Time Analysis*</u>. It is as in the Theorem 5.1.

— <u>*Indistinguishability Argument*</u>. 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 5.1.

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. Furthermore, (y, z_0, z_1, c) are used identically as $(a_w, h_w, h_{w+1}, b_{w+1})$ are used in the real-system to compute required values (i.e., $s' = s_{w,w+1}$). Therefore, the joint probability distributions $Pr[\overrightarrow{A}_{real} = \overrightarrow{a}] = Pr[\overrightarrow{A}_{sim} = \overrightarrow{a}]$ (i.e., perfectly indistinguishable) as in Theorem 5.1. □

Another security concern in audit logging is the *delayed detection attack* identified in Ma and Tsudik [2008, 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 a TTP provides the 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 [1997, 2003] and Schneier and Kelsey [1998, 1999]).

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.

Table II. Notation for Performance Analysis and Comparison

$Muln$: Mul. mod n, where n is a Blum-Williams integer [Ma 2008]		R: # of verifiers
$Mulp$/$Mulq$: Mul. mod p and mod q, respectively		$Addq$: Addition mod q
Exp: Exponentiation mod p	Sqr: Squaring mod n	PR: Pairing operation
$GSig$/$GVer$: Generic signing and verification, respectively		H: Hash operation
x: FssAgg-BM/AR security parameter	L: max. # of key updates	l: # of data items

Suggested bit lengths to achieve a 80-bit security for the above parameters are as follows for each compared scheme (based on the parameter sizes suggested in Ma and Tsudik [2007] and Ma [2008]): Large primes ($|p| = 512$, $|q| = 160$) for BAF/FI-BAF, Logcrypt and FssAgg-BLS, all were implemented in EC. ($|n| = 1024$, $x = 160$) for FssAgg-AR and FssAgg-BM.

6. 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 provable security.

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

6.1. Computational Overhead

We implement our schemes on an Elliptic Curve (EC) [Menezes et al. 1996], which offers small key/signature sizes and computational efficiency [Hankerson et al. 2004].

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

Comparison. The closest counterparts of our schemes are FssAgg schemes [Ma 2008; Ma and Tsudik 2007, 2008, 2009]. The signature generation of FssAgg-BLS [Ma and Tsudik 2007] is expensive due to Exp, 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 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 $x \cdot Sqr + (1 + x/2)Muln$ (e.g., x = 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

Table III. Computation Involved in BAF, FI-BAF, and Previous Schemes

		Sig	**Upd**	**Ver**
PKC-based	*BAF*	$Mulq + 2H$	$2H$	$(l + 1)(Exp + 2H)$
	FI-BAF	$2 \cdot BAF.Sig$	$2 \cdot BAF.Upd$	$BAF.Ver$
	FssAgg-BLS	$H + Exp + Mulp$	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	$l \cdot H$

Signature generation and key update costs are given for a single data item. Signature verification cost is given for $0 < l < L$ data items.

Table IV. Average Execution Times (in μs) of BAF, FI-BAF and Previous Schemes for a Single Log Entry (Sampled over l = 10000 Entries)

	PKC-based						Symmetric
	BAF	FI-BAF	FssAgg Schemes			Logcrypt	
			BLS/iBLS	BM/iBM	AR/iAR		
Sig	10	20	1830/3660	3600/7200	7710/15420	1020	6
Ver	740	760	24500/49000	1700/3400	5300/10600	1230	6

$(3x)Sqr + (2 + x/2)Muln$ 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 ($FssAgg.Sig$) 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 [Holt 2006], and thus has standard signature costs (e.g., we use ECDSA [American Bankers Association 1999] for Logcrypt in our comparison). The symmetric schemes [Bellare and Yee 2003; Ma and Tsudik 2007; Schneier and Kelsey 1998, 1999] are in general efficient, since they only need symmetric cryptographic operations.

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

In addition to the analytical comparison, we also measure the execution times (in μs) 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 [Ma and Tsudik 2007; Schneier and Kelsey 1998, 1999; Bellare and Yee 2003] were measured using implementations based on the MIRACL library.[9] The execution times of FssAgg-AR/BM were computed using implementations based on the NTL library.[10] Table IV shows the signing/verifying costs of a single log entry in each scheme.

Both BAF and FI-BAF are *at least hundred times faster* than their PKC-based counterparts. Similarly, both BAF and FI-BAF signature verifications are also more efficient than other PKC based schemes (from 1.6 times up to 33 times faster). This computational efficiency makes BAF/FI-BAF the best alternative among existing schemes for secure logging with public verifiability in resource-constrained devices. (See Table IV.)

[9]`http://www.shamus.ie/`
[10]`http://www.shoup.net/ntl/`

Table V. 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.	$2\|q\|$	$O(2l\|q\|)$	$\|p\|$	$\|n\|$	$\|n\|$	$\|H\|$	$O(2l \cdot FssAgg)$	$O(2l\|q\|)$	$O(l\|H\|)$

When compared with the signature generation of previous symmetric logging schemes (e.g., Ma and Tsudik [2007], Schneier and Kelsey [1998, 1999], and Bellare and Yee [1997, 2003]), 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.

6.2. 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-delete strategy [Yavuz and Ning 2009b]). 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) at the signer side.

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 V summarizes the comparison. Note that storage and communication overheads that stem from the data items are independent from cryptographic technique and are the same for all compared schemes (i.e., linear). They are therefore omitted in the comparison.

The symmetric schemes (e.g., Bellare and Yee [1997, 2003], and 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). 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.

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 [Ma and Tsudik 2007], FssAgg-BM and FssAgg-AR [Ma 2008] achieve the signature aggregation in a publicly verifiable way, and therefore their signature storage/communication overheads are constant. Table V 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 $O(2l \cdot \mathit{FssAgg})$ in Table V).

BAF has constant signature storage and communication overheads (with respect to the number of data items to be signed), and is 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], and Holt [2006]). BAF is also more efficient than FssAgg-AR/BM [Ma 2008], but less efficient than FssAgg-BLS [Ma and Tsudik 2007], as shown in Table V. Similar to its immutable counterpart iFssAgg [Ma and Tsudik 2009] schemes, FI-BAF also demands linear signature storage and communication overheads.

Note that achieving constant signature storage/communication overhead is especially important for the resource constrained devices. In these type of devices, the size audit logs are generally smaller, which makes the benefit of this property more apparent. However, if the size of individual log entries are very large (e.g., order of MBs), due to high linear storage and communication overhead of log entries, the benefit of signature aggregation becomes less important.

6.3. 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 [Yavuz and Ning 2009b]). 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.

The bottom part of Table I summarizes the comparison of BAF and FI-BAF with the previous schemes in terms of their scalability and security properties. The symmetric schemes 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. FssAgg-MAC [Ma and Tsudik 2007] 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.

6.4. Limitations

In BAF schemes, the size of public key is linear with respect to the number time periods. This may incur high storage overhead for 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 3)). For example, the signer computational/storage efficiency is critically important for secure logging in resource-constrained devices such as RFID tags [Batina et al. 2007] and wireless sensors [Ma and Tsudik 2007], whereas the verifiers (e.g., laptops) can tolerate the storage overhead.

Some generic forward-secure signature constructions (e.g., the storage efficient construction in Malkin et al. [2002]) 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.

7. 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 along with an online TTP support, and also consults symmetric encryption to achieve confidentiality. Holt [2006] extended the idea given in Schneier and Kelsey [1998] to the PKC domain by replacing MACs with digital signatures and ID-based cryptography. Finally, Ma and Tsudik 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] and Ma [2008]. The detailed analysis and comparison of all these schemes with ours were given in Section 6.

Apart from these schemes, there are complementary works to ours: Chong and Peng [2003] extended the scheme in Schneier and Kelsey [1998] by strengthening it via tamper-resistant hardware. Waters et al. [2004] proposed an audit logging scheme that enables encrypted search on audit logs via Identity-Based Encryption (IBE) [Boneh and Franklin 2003]. Davis et al. [2004] proposed time-scoped search techniques on encrypted audit logs. These schemes can be coupled with BAF to provide confidentiality. Yavuz and Ning [2009b] proposed a hash-based FAS construction for Unattended Wireless Sensor Networks. Moreover, there is a line of work that rely on authenticated data structures to protect audit logs in distributed systems [Anagnostopoulos et al. 2001; Maniatis and Baker 2003; Papamanthou et al. 2008]. While being computationally efficient, these approaches do not provide forward-security. Furthermore, any authenticated data structure can be strengthened with a forward-secure signature [Crosby and Wallach 2009]. Therefore, our schemes can serve these authenticated data structures as an ideal forward-secure and aggregate digital signature primitive.

8. 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 minimal logger computational overhead, small-constant signature storage/communication overheads, public verifiability (without online TTP support), immediate log verification, and high verifier computational 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 alternatives show that our schemes are ideal choices for secure audit logging in resource-constrained devices.

ACKNOWLEDGMENTS

We would like to thank Dr. Di Ma who kindly provided her implementation of FssAgg schemes [Ma 2008; Ma and Tsudik 2007, 2009]. We also would like to thank anonymous reviewers for their useful comments.

REFERENCES

ABDALLA, M. AND REYZIN, L. 2000. A new forward-secure digital signature scheme. In *Proceedings of the Advances in Crpytology (ASIACRYPT'00)*. Springer-Verlag, 116–129.

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

ANAGNOSTOPOULOS, A., GOODRICH, M. T., AND TAMASSIA, R. 2001. Persistent authenticated dictionaries and their applications. In *Proceedings of the Information Security Conference (ISC'01)*. Springer-Verlag, 379–393.

BATINA, L., GUAJARDO, J., KERINS, T., MENTENS, N., TUYLS, P., AND VERBAUWHEDE, I. 2007. Public-key cryptography for RFID-Tags. In *Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMM'07)*. IEEE Computer Society, 217–222.

BELLARE, M. AND MINER, S. 1999. A forward-secure digital signature scheme. In *Proceedings of the Advances in Crpytology (CRYPTO'99)*. Springer-Verlag, 431–448.

BELLARE, M. AND ROGAWAY, P. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS'93)*. ACM, 62–73.

BELLARE, M. AND ROGAWAY, P. 2005. *Introduction to Modern Cryptography* 1st Ed.

BELLARE, M. AND YEE, B. S. 1997. Forward integrity for secure audit logs. Tech. rep., San Diego, CA, USA.

BELLARE, M. AND YEE, B. S. 2003. Forward-security in private-key cryptography. In *Proceedings of 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 *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07)*. ACM, 276–285.

BONEH, D. AND FRANKLIN, M. 2003. Identity-based encryption from the weil pairing. *SIAM J. Comput. 32*, 586–615.

BONEH, D., GENTRY, C., LYNN, B., AND SHACHAM, H. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings 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.

CORON, J. AND NACCACHE, D. 2003. 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)*. 392–397.

CROSBY, S. AND WALLACH, D. S. 2009. Efficient data structures for tamper evident logging. In *Proceedings of the 18th Conference on USENIX Security Symposium*.

DAVIS, D., MONROSE, F., AND REITER, M. 2004. Time-scoped searching of encrypted audit logs. In *Proceedings of the 6th International Conference on Information and Communications Security (ICICS'04)*. 532–545.

DODIS, Y., KATZ, J., XU, S., AND YUNG, M. 2002. Key-insulated public key cryptosystems. In *Proceedings of the 21th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'02)*. 65–82.

FALL, K. 2003. 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)*. ACM, 27–34.

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 *Proceedings of the 4th Australasian Workshops on Grid Computing and e-Research (ACSW'06)*. 203–211.

HSU, W. W. AND ONG, S. 2007. Technical forum: WORM storage is not enough. *IBM Syst. J. 46*, 2, 363–369.

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.

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, 341–352.

MA, D. AND TSUDIK, G. 2007. Forward-secure sequential aggregate authentication. In *Proceedings of the 28th IEEE Symposium on Security and Privacy (S&P'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 Trans. Storage (TOS) 5*, 1, 1–21.

MALKIN, T., MICCIANCIO, D., AND MINER, S. K. 2002. Efficient generic forward-secure signatures with an unbounded number of time periods. In *Proceedings of the 21th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'02)*. Springer-Verlag, 400–417.

MANIATIS, P. AND BAKER, M. 2003. Authenticated append-only skip lists. *Acta Math. 137*, 151–169.

MASS, M. 2004. Pairing-based cryptography. M.S. dissertation. Technische Universiteit Eindhoven.

MENEZES, A., VAN OORSCHOT, P. C., AND VANSTONE, S. 1996. *Handbook of Applied Cryptography*. CRC Press. ISBN: 0-8493-8523-7.

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.

OPREA, A. AND BOWERS, K. D. 2009. Authentic time-stamps for archival storage. In *Proceedings of the 14th European Symposium on Research in Computer Security (ESORICS'09)*. Springer-Verlag, Berlin, Heidelberg, 136–151.

PAPAMANTHOU, C., TAMASSIA, R., AND TRIANDOPOULOS, N. 2008. Authenticated hash tables. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM, New York, 437–448.

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 Trans. Inf. Syst. Secur. 2*, 2, 159–176.

STINSON, D. 2002. *Cryptography: Theory and Practice* 2nd Ed. CRC/C&H.

WANG, Y. AND ZHENG, Y. 2003. Fast and secure magnetic worm storage systems. In *Proceedings of the 2nd IEEE International Security in Storage Workshop (SISW'03)*. 11–25.

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)*.

Received November 2011; accepted February 2012