Ad Hoc Networks 10 (2012) 1204–1220

Contents lists available at SciVerse ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

Self-sustaining, efficient and forward-secure cryptographic constructions for Unattended Wireless Sensor Networks ☆

Attila Altay Yavuz *, Peng Ning

Cyber Defense Laboratory, Department of Computer Science, North Carolina State University, Raleigh, NC 27695, USA

ARTICLE INFO

Article history:
Received 21 February 2011
Received in revised form 17 February 2012
Accepted 26 March 2012
Available online 12 April 2012

Keywords:
Applied cryptography
Unattended Wireless Sensor Networks (UWSNs)
Digital signatures
Forward security
Aggregate signatures

ABSTRACT

Unattended Wireless Sensor Networks (UWSNs) operating in hostile environments face great security and performance challenges due to the lack of continuous real-time communication with the final data receivers (e.g., mobile data collectors). The lack of real-time communication forces sensors to accumulate sensed data possibly for long time periods, along with the corresponding authentication tags. It also makes UWSNs vulnerable to *active adversaries*, which can compromise sensors and manipulate the collected data. Hence, it is critical to have *forward security* property such that even if the adversary can compromise the current keying materials, she cannot forge authentication tags generated before the compromise. Forward secure and aggregate signature schemes are developed to address these issues. Unfortunately, existing schemes either impose substantial overhead, or do not allow public verifiability, thereby impractical for resource-constrained UWSNs.

In this paper, we propose a new class of cryptographic schemes, referred to as *Hash-Based Sequential Aggregate and Forward Secure Signature (HaSAFSS)*, which allows a signer to sequentially generate a compact, fixed-size, and publicly verifiable signature efficiently. We develop three HaSAFSS schemes, Symmetric HaSAFSS (Sym-HaSAFSS), Elliptic Curve Cryptography (ECC) based HaSAFSS (ECC-HaSAFSS) and self-SUstaining HaSAFSS (SU-HaSAFSS). These schemes integrate the efficiency of MAC-based aggregate signatures and the public verifiability of Public Key Cryptography (PKC)-based signatures by preserving forward security via Timed-Release Encryption (TRE). We demonstrate that our schemes are secure and also significantly more efficient than previous approaches.

1. Introduction

An Unattended Wireless Sensor Network (UWSN) [1–5] is a Wireless Sensor Network (WSN) in which continuous end-to-end real-time communication is not possible for sensors (senders) and their receivers (e.g., mobile collectors, static sinks). In other words, receivers might not be available for sensors from time to time, sometimes for long time periods. Sensors accumulate the sensed data in these time periods, and transmit it to the receivers whenever they become available (e.g., visits of mobile collectors [2,5]).

Examples of UWSNs can be found in military WSN applications (e.g., [6,1]), where sensors are deployed to an adversarial and unattended environment to gather information about enemy activities. One illustrative example is LANdroids [7], a recent US Defense Advanced Research Projects Agency (DARPA) research project, which designs smart robotic radio relay nodes for the battlefield deployment. These nodes are expected to be deployed in hostile environments to gather military information and upload to ally vehicles (e.g., UAV, soldier) upon their arrivals.

☆ A preliminary version of this paper appeared in *Sixth Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2009)*, pp. 1–10, July 13–16, 2009.

* Corresponding author.
E-mail addresses: aayavuz@ncsu.edu (A.A. Yavuz), pning@ncsu.edu (P. Ning).

1570-8705/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.adhoc.2012.03.006

The lack of real-time communication and the resource constraints of UWSNs bring several security and performance challenges, especially when an UWSN is deployed in a hostile environment as described above. In particular, inability to off-load the sensed data forces sensors to accumulate a large amount of data along with their authentication information. More importantly, unattended settings make the UWSN highly vulnerable to active [5] and/or mobile adversaries [1,2]. Such an adversary can physically compromise sensors and gain access to the accumulated data as well as the existing cryptographic keys. When a sensor is compromised, the adversary can always use the cryptographic keys learned from the sender to generate forged messages after the attack. However, it is critical to prevent the adversary from modifying the data accumulated before the adversary takes control of the sender [5]. Such a security property is referred to as *forward security* [8].

Forward secure signatures have been proposed to provide forward security for pre-accumulated data [8]. In a forward secure signature, a signer evolves her private key periodically (either for each signed data item or for each time period) in a one-way manner, and erases the previous private key. While mitigating the effects of key exposure, this strategy also brings significant storage and communication overheads because of the accumulation of the signatures of individual data items. Recently, forward-secure and aggregate signatures [5,9,10] were developed to address this issue by integrating aggregate signatures (e.g., [11,12]) with forward-secure signatures. This approach allows a signer to reduce linear signature size to a small and constant size.

The above properties of forward secure and aggregate signatures make them ideal cryptographic tools for achieving data integrity and authentication for UWSN applications in the presence of active adversaries [5]. However, all existing PKC-based forward secure and aggregate signature schemes (e.g., [11,5,9]) impose extreme computational overheads on the network entities, which are intolerable for resource-constrained UWSN applications. Another alternative is to rely on symmetric key cryptography via hash chains and Message Authentication Codes (MACs) as in FssAgg-MAC [5]. However, such an approach requires full symmetric key distribution and does not allow signatures to be publicly verifiable. This makes it unscalable and impractical for large distributed UWSN applications. Thus, it is necessary to seek highly efficient and flexible forward secure and aggregate signatures for UWSN applications.

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

A nice property of these schemes is that they achieve five seemingly conflicting goals, computational efficiency, public verifiability, forward security, flexibility and scalability at the same time. To achieve this, HaSAFSS schemes introduce asymmetry between the senders and receivers using the time factor via Timed-Release Encryption (TRE) [13]. Using this asymmetry, our schemes achieve high efficiency by minimizing costly Expensive Operations (ExpOps),[1] while still remaining publicly verifiable and forward secure.

We summarize the properties of our schemes as follows:

(1) Our schemes achieve near-optimal computational efficiency and public verifiability at the same time. They achieve the computational efficiency by adopting cryptographic hash functions to compute aggregate and forward secure signatures, and thus are much more efficient than all the existing schemes (e.g., [11], FssAgg-BLS in [5], FssAgg-AR/BM [9]), with the exception of FssAgg-MAC in [5]. When compared with FssAgg-MAC [5], our schemes further achieve public verifiability by eliminating symmetric key distribution.
(2) In our schemes, both signers (senders) and verifiers (receivers) get equal benefits of computational efficiency, while most existing schemes incur heavy computational overhead on either the signer or verifier side. This property is especially useful for UWSN applications in which both the signers and verifiers need to process large amounts of data efficiently.
(3) Since our schemes achieve signature aggregation, a signer always stores and transmits only a single compact signature, regardless of the number of time periods or data items to be signed. This offers bandwidth efficiency.
(4) Based on an omni-symmetric design that provides public verifiability, Sym-HaSAFSS is the most computationally efficient scheme among of all its counterparts. It is also the most verifier storage friendly scheme by requiring only a small and constant storage for the verifiers. However, it requires a linear storage at the signer side.
(5) ECC-HaSAFSS requires storing one key for each signer by offering a signer storage friendly scheme. However, it requires an ExpOp to initialize each time interval (but still requires only three hash operations to sign/verify per-item), and also demands quadratic storage overhead at the verifier side.
(6) Despite their simplicity and efficiency, Sym-HaSAFSS and ECC-HaSAFSS put a linear bound on the maximum number of time periods that a signer can use. Moreover, they require a pre-determined and fixed data delivery schedule that all signers have to agree upon before deployment. SU-HaSAFSS addresses these limitations by offering the following properties:
   – SU-HaSAFSS enables a signer to use (practically) unbounded number of time periods (this implies the ability of generating unbounded number of signatures) without requiring any re-keying after the deployment. This allows a signer to operate

[1] For brevity, in this paper, we refer to an expensive cryptographic operation, i.e., a modular exponentiation or a pairing operation, as an ExpOp.

in a hostile environment for a long time without costly (sometimes impossible) re-deployment/re-keying support. This also offers only a constant key storage at the signer side and a linear key storage at the verifier side.
- SU-HaSAFSS enables each sender to decide her own data delivery schedule dynamically (after the deployment) without requiring any (online) communication with other signers or a trusted third party. Therefore, SU-HaSAFSS can support applications in which a pre-determined data delivery schedule cannot be decided.
- To achieve these properties, SU-HaSAFSS requires a few ExpOps per interval (for the initialization purpose), and therefore is more computationally costly than Sym-HaSAFSS and ECC-HaSAFSS. However, SU-HaSAFSS is significantly more efficient than all other PKC-based schemes (e.g., FssAgg) that require an ExpOp per data item (SU-HasAFSS requires only three hash operations per data item to sign or verify in given time interval).

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

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 provides the preliminaries. Section 4 presents our assumptions as well as the security and data models. Section 5 describes the proposed schemes in detail. Section 6 provides the security analysis of the proposed schemes. Section 7 gives performance analysis and compares the proposed schemes with previous approaches. Section 8 concludes this paper.

2. Related work

2.1. Forward secure and/or aggregate signatures

A forward secure signature aims to minimize the effect of key compromises. The first forward secure signature scheme was proposed in [14]. In this scheme, each signature is associated with a time period in addition to the signed data item. After each time period, the secret key of the signer is changed and cannot be used for previous time periods. Several new schemes were later proposed to improve storage requirement, signature size, and computational cost (e.g., [8,15–20]). Our paper is also related to aggregate signatures, which aggregate n individual signatures associated with n different users into a single, compact signature. The first aggregate signature scheme was proposed in [11], and then several new schemes achieving more advanced properties were developed (e.g., sequentiality [21], low storage overhead [22]).

The most closely related schemes to ours are forward-secure and aggregate signature schemes. The first forward-secure and aggregate schemes were proposed by Ma and Tsudik in [5] (i.e., FssAgg-MAC and FssAgg-BLS). Later, Ma et al. developed FssAgg-AR and FssAgg-BM in [9,10] that were more computational and storage efficient than FssAgg-BLS. However, all these schemes are still not efficient enough for UWSNs.

In the preliminary version of this work [23], we developed Sym-HaSAFSS and ECC-HaSAFSS that achieve the near optimal computational efficiency and public verifiability at the same time by leveraging the already existing delays in UWSNs. These schemes are significantly more efficient than FssAgg schemes, but they are still not flexible enough as we discussed in the introduction.

In this paper, we add the new SU-HaSAFSS scheme, which addresses the limitation of our previous schemes by introducing slightly more computational overhead. SU-HaSAFSS is still more computationally efficient than all other PKC-based alternatives (e.g., FssAgg schemes), as we will demonstrate in Section 7.

2.2. Timed-Release Encryption (TRE)

The purpose of TRE is to encrypt a message in such a way that no entity, including the intended receivers, can decrypt it until a pre-defined future time. The majority of modern TRE schemes are based on Trusted Agent (TA), in which a time server provides universally accepted time reference and trapdoor information to users [24,13]. Hence, users can decrypt the ciphertext when its related trapdoor information is released by the TA. Most of the recent TRE schemes (e.g., [25–28]) are based on Identity-Based (IB) cryptography [29].

Sym-HaSAFSS relies on only the basic TRE concept to achieve a ExpOp-free construction, while SU-HaSAFSS was inspired by the anonymous TRE mechanism [26] to meet its requirements.

2.3. TESLA

TESLA [30] is an efficient broadcast authentication protocol that also uses delayed disclosure of the keying material, assuming that signers and verifiers are loosely synchronized. However, our schemes provide important properties that are not available in TESLA.

First, in TESLA, an attacker $\mathcal{A}$ that compromises a signer can easily forge all previously accumulated data items in a given time interval, since asymmetry is directly introduced by the signer. However, HaSAFSS schemes protect previously accumulated data items via mechanisms that achieve forward security. Moreover, TESLA generates an individual signature for each individual data item; however, HaSAFSS schemes compute an aggregate signature for multiple data items, which reduces the storage and communication overhead. Finally, HaSAFSS schemes also achieves the "all-or-nothing" property (i.e., resistance against the data truncation) due to the signature aggregation, which is not available in TESLA.

2.4. Self-healing techniques

Recently, a series of studies [2,3,1] based on self-healing techniques have been proposed to achieve data survival in

UWSNs. They first propose mobile adversary models, in which the adversary compromises the sensors and deletes the data accumulated in them. To confront such an adversary, they propose collaborative techniques, in which non-compromised sensors collectively attempt to recover a compromised sensor [3,2] by introducing local randomness (with a PRNG) to their neighborhood. DISH [2] assumes a read-only adversary and targets the data secrecy. POSH [3] allows constrained write-only adversaries and targets the data survival. Pietro et al. [1] elaborates the adversary models given in [2,3] and provides experimental and analytical results for them.

Note that the adversary models and security goals in [2,3,1] are different from ours. In our schemes, the goal of the adversary is to forge data and/or destroy the authentication. However, the goal of the adversary in [2,3,1] is to prevent the data from reaching the sink.

3. Preliminaries

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

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

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

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

SGN denotes a standard digital signature scheme (e.g., Schnorr [31], DSA [32]) and *MMM* denotes a Malkin Micciancio Miner (*MMM*) generic forward-secure signature construction [20] instantiated from an appropriate base scheme (e.g, [17,33]). $(\overline{sk}, \overline{pk}) = SGN.Kg(1^\kappa)$, $\sigma = SGN.Sig_{\overline{sk}}(m)$ and $\{success, failure\} = SGN.Ver_{\overline{pk}}(\sigma, m)$ denote the key generation for security parameter κ, signature generation on message m with private $\overline{sk}$ and verification of σ on m with public key $\overline{pk}$, respectively.

MMM signature scheme, in addition to the above standard algorithms, also has a key update algorithm, denoted $MMM.Upd(sk_w, w)$. That is, given the current private key sk_w, the update algorithm generates one or several new key instances to be used in the future. This is done by using a sum composition and a product composition iteratively based on a special tree structure. The details of this update procedure can be found in [20].

Definition 1. $\tilde{e}$ is a bilinear pairing $\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$, i.e., an admissible map with the following properties:

1. *Bilinearity*: $\tilde{e}(aP, bQ) = \tilde{e}(bP, aQ) = \tilde{e}(abP, Q) = \tilde{e}(P, abQ) = \tilde{e}(P, Q)^{ab}$, $\forall P, Q \in \mathbb{G}_1$ and $\forall a, b \in Z_q^*$.
2. *Non-degeneracy*: $\exists P, Q \in \mathbb{G}_1$ such that $\tilde{e}(P, Q) \neq 1$. In our settings, we select prime order groups in which $\forall P, Q \in \mathbb{G}_1, \tilde{e}(P, Q) \neq 1$, and therefore $\tilde{e}(G, G)$ (G is the generator of $\mathbb{G}_1$) is a generator of $\mathbb{G}_2$.
3. *Efficiency*: There exists an efficient algorithm to compute $\tilde{e}(P, Q), \forall P, Q \in \mathbb{G}_1$.

Definition 2. Elliptic Curve Discrete Logarithm Problem (ECDLP) [34] is defined as follows: Given a prime p, a generator $G \in \mathbb{G}$, and a random point $Q \in E(F_p)$, find the integer k, $0 \leqslant k \leqslant p-2$, such that $Q \equiv kG$. ECDLP is (τ, ϵ)-hard, if no algorithm running in time less than τ can solve the ECDLP with a probability more than ϵ. where ϵ is computed over the random choices of (G, k).

Definition 3. q-Bilinear Diffie–Hellman Inversion (q-BDHI) problem ? [26] is defined as follows: Given $(q+1)$-tuple $(G, aG, a^2G, \ldots, a^qG) \in \mathbb{G}_1^{q+1}$ for some $a, q \in Z_p^*$, compute $\tilde{e}(G, G)^{a^{-1}} \in \mathbb{G}_2$. q-BDHI is (τ, ϵ)-hard, if no algorithm running in time less than τ can solve the q-BDHI with a probability more than ϵ, where ϵ is computed over the random choices of G.

4. Models

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

4.1. Threat model and security model

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

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

Before giving the HaSAFSS security model, we review the *Quality of Forward Security* (*QoF*) concept [9]:

Definition 4. *QoF* is a performance-forward security quality trade-off, which is decided according to the following two key update methods:

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

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

4.1.1. HaSAFSS security objectives

The goal of HaSAFSS schemes is to achieve secure signature aggregation and forward security simultaneously. We follow the example of previous forward-secure and aggregate signature schemes (i.e., [5,9]), which focus on forward-security, existential unforgeability and authentication properties, to analyze our schemes in Section 6.

Remark that HaSAFSS schemes exploit the already existing delays (i.e., time factor) in UWSNs to achieve its desirable properties. Thus, the forward-security objective of HaSAFSS schemes is slightly different than that of previous forward-secure and aggregate schemes (i.e., [5,9,10]). That is, HaSAFSS aims to achieve a *time-valid forward-security* instead of a *permanent* forward-security.

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

Definition 5. The security of a HaSAFSS scheme is defined as the non-existence of a PPT bounded adversary $\mathcal{A}$ who produces an existential forgery against HaSAFSS even under the exposure of current keying material in the duration of a designated time interval $t_{w,w'}$. This is called as *Time-valid Forward-secure Existential Unforgeability (TFEU)* property.

Note that HaSAFSS schemes mainly rely on symmetric cryptography to achieve the above goal.[2] Indeed, in Section 7, we show that they achieve the highest QoF (i.e., per-item QoF) in a given time interval, and still remain much more computationally efficient than previous PKC-based schemes.

Remark. The time validity requirement in Definition 5 implies that a signer should transmit the forward-secure and aggregate signature computed in $t_{w,w'}$ to the verifiers, before the TTP releases the time trapdoor key associated with $t_{w'}$. Such a requirement is compatible with the periodic data collection characteristic of the envisioned UWSN applications [5,7,6]. Details of how our schemes handle data/time trapdoor information are given in data models and Section 5.

HaSAFSS schemes integrate various cryptographic primitives in a novel and efficient way to achieve their security and efficiency goals. In Section 6, we prove that breaking a HaSAFSS scheme is as difficult as breaking its underling primitive(s). Therefore, HaSAFSS schemes achieve *TFEU* as long as the below assumptions hold:

Assumption 1. We assume that cryptographic primitives used in our schemes have all the semantic security properties [35] as follows:

(i) H_1, H_2, ... , H_6 are secure and collision-free hash functions producing indistinguishable outputs from the random uniform distribution [36]. (ii) *MAC* is Existential Unforgeable Under Chosen Message Attacks (EU-CMA) [37]. (iii) Symmetric encryption function $\mathcal{E}$ is Indistinguishable under Chosen Ciphertext Attacks (IND-CCA secure) [38]. (iv) *ECDLP* [34] and *q-BDHI problem* [39] are (τ, ϵ)-hard with appropriate parameters.

Assumption 2. We assume a Trusted Third Party (TTP), which is trusted by all network entities. (i) The adversary $\mathcal{A}$ cannot compromise the TTP; (ii) $\mathcal{A}$ may jam the TTP, but if an entity continuously tries, its messages can eventually reach the TTP; (iii) the TTP releases *time trapdoor keys* (secret cryptographic keys) with which the receivers verify the forward secure and aggregate signatures generated by the senders (the TTP acts as a Trusted Agent (TA) as in TRE schemes [27]). We assume that time trapdoor keys released by the TTP reach the receivers eventually. Details of the time trapdoor key delivery are given in the data models.

4.2. Data model

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

(a) *Pre-determined data delivery model*: This model addresses applications in which signers/verifiers and the TTP can agree on a prospective data delivery schedule so that data/trapdoor delivery can be performed based on this schedule. In this model, the TTP *passively broadcasts* time trapdoor keys based on a *pre-determined schedule*, and it is assumed that the verifiers (e.g., mobile collectors) will be available for their signers based on this schedule. This data delivery model is realized via *synchronous time trapdoor release mode* in Sym-HaSAFSS and ECC-HaSAFSS. The TTP passively broadcasts time trapdoor keys according to a pre-determined data delivery schedule, which is followed by both the signers and verifiers.

(b) *On-demand data delivery model*: This model addresses applications where the nature of application does not allow a prospective delivery schedule. In this case, the TTP provides the time trapdoor information to the verifiers on demand. A representative scenario would be a military UWSN application, in which soldiers gather information from sensors from time to time and then request time trapdoor keys from the TTP (e.g., UAV/satellite). Note that in the worst case, verifiers can obtain time trapdoor keys from a mobile TTP directly (e.g., MTC (Mobile Tactical Center) [40]). Thus, Assumption 2-(iii)

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

is realistic. On-demand data delivery model is realized via *asynchronous time trapdoor release mode* in Sym-HaSAFSS and ECC-HaSAFSS. The TTP broadcasts time trapdoor keys when she receives more than a threshold number of (authenticated) requests (e.g., $\tau = 90\%$) from the verifiers. Note that in this model, the collaborative requests of verifiers might cause the release of time trapdoor key before all senders (signers) upload their data to the verifiers. In this case, verifiers reject the data received after the release of the time trapdoor key. However, this is not critical for many applications using on-demand data delivery model. Some examples of such applications are as follows: (i) receivers (verifiers) visit senders to collect the data and then demand the time trapdoor key from the TTP. (ii) The application demands a rapid decision based on the available data collected by the receivers. In this case, minimizing the verification delay is more important than collecting the data from all signers in the system. For instance, if majority of the verifiers receive the same data from a group of sender at an early stages of a given time interval, they do not have to wait until the end of this interval for the verification. Remark that in this model, time trapdoor keys are computed in a slightly different way than that of the pre-determined data delivery model. That is, the TTP imposes an order on the elements of reversed hash chain (i.e., time trapdoor keys), which allows verifiers to trace the time trapdoor keys in an asynchronous setting.

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

This model enables HaSAFSS to address applications that require users to operate long times in hostile environments autonomously. One example would be mobile reconnaissance vehicles (e.g., autonomous LANdroids [7]) that gather information from a target area. Such a vehicle *autonomously* decides sense, sleep and broadcast time durations, and signs and seals the data according to the situation. No entity can modify or recover the data that the vehicle accumulated until the end of designated time duration, even if it is captured by the enemy.

Self-decisive data delivery model is realized via the *periodic time trapdoor release mode* in SU-HaSAFSS.

Remark. (i) Signers do not need to communicate with the TTP. Verifiers communicate with the TTP only in on-demand data delivery model, only once for each time period (the TTP can be offline most of the time). (ii) In HaSAFSS, similar to the previous forward-secure and aggregate schemes (e.g., [5,9,10,41]), the signer computes aggregate signatures of distinct data items accumulated-so-far (i.e., similar to the condensed signatures notion in [42]). Cross-signer signature aggregation (e.g., [11,12,22]) is out of our scope. (iii) Introducing an asymmetry with a time factor requires that the data to be verified must be received by the verifiers before its associated keying material is released. Receivers reject any data item/signature received after the release of its associated time trapdoor key.

5. Proposed schemes

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

5.1. Overview

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

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

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

5.1.1. HaSAFSS instruments and strategies

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

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

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

- *Encrypted chain root*: Such asymmetry can be introduced by two conditions. First, k_0 should remain confidential in $t_{w,w'}$. In this way, if the signer is compromised in $t_{w,w'}$, the adversary cannot obtain k_0 before the end of $t_{w,w'}$, and therefore she cannot forge signatures computed via k_0 (k_0 is also updated for each signed data item). At the same time, k_0 should be publicly available to all verifiers at the end of $t_{w,w'}$ so that any entity should be able to verify signatures computed via k_0. The signer seals k_0 until the end of $t_{w,w'}$ by encrypting it with a *session key* $K_{w,w'}$ as $c_{w,w'} = \mathcal{E}_{K_{w,w'}}(k_0)$, and then deleting $(k_0, K_{w,w'})$.
- *Session keys and time trapdoor keys*: To enable the recovery of k_0 from $c_{w,w'}$ at the end of $t_{w,w'}$, each *session key* $K_{w,w'}$ is controlled with a *time trapdoor key* $tk_{w'}$. HaSAFSS schemes achieve their distinctive properties by following different session key and time trapdoor key computation mechanisms:

In Sym-HaSAFSS and ECC-HaSAFSS, each time trapdoor key is constructed as an element of a hash chain to enable its ExpOp-free verification and session key recovery. These time trapdoor keys are released by following either the synchronous or asynchronous data delivery model (see Section 4).

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

SU-HaSAFSS uses a pairing based time trapdoor key structure, which is inspired by AnTRE [26]. Such a structure allows signers to compute their own session keys and encrypted chain roots without relying on pre-computed public keys. Hence, despite being slightly more costly than Sym-HaSAFSS and ECC-HaSAFSS, SU-HaSAFSS addresses their limitations and also preserves the per-data item efficiency of HaSAFSS constructions over the traditional PKC-based schemes.

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

5.2. Sym-HaSAFSS

The four algorithms of Sym-HaSAFSS are given below:

- *Key generation*: The TTP performs key generation as follows:
  (1) Choose the maximum clock synchronization error as δ_t and the trapdoor release times as $0 < T_0 < T_1 < \ldots < T_{L-1}$. Every two consecutive time points T_{i-1} and T_i form the ith time interval t_i. The TTP can update δ_t dynamically when needed by broadcasting new δ_t along with a time trapdoor keys. The new δ_t must be digitally signed with its associated time trapdoor key and must be broadcasted sufficiently earlier than its actual use. This ensures that all entities are aware of this update before new (authentic) δ_t is used.
  (2) Randomly generate a hash chain $v_w = H_1(v_{w-1})$ for $w = 1, \ldots, L-1$, whose elements will be used as the secret time trapdoor keys in the reversed order as $tk_w = v_{L-1-w}$ for $w = 0, \ldots, L-1$. Each tk_w is associated with time interval t_w for $w = 0, \ldots, L-1$. Compute the encrypted chain roots for each signer ID_i as follows:
     (a) Generate the initial per-interval key $z_0 \xleftarrow{R} \{0,1\}^n$ for each ID_i. The objective of the per-interval key is to provide a fresh initialization key for each time period, from which the signer ID_i will derive the chain root (i.e., per-item key) of that time interval. That is, the chain root of ID_i for each t_w is derived as $k_0 = H_2(z_w)$ and $z_{w+1} = H_1(z_w)$ for $w = 0, \ldots, L-1$.
     (b) Compute the session key as $K_w = H_3(tk_w \| ID_i)$ and the encrypted chain root of ID_i for t_w as $c_w = \mathcal{E}_{tk_w}(k_0)$ for $w = 0, \ldots, L-1$.

  3) Distribute required keys and the data delivery schedule to each ID_i and verifiers as ID_i:$\{z_0, c_w, T_w, \delta_t\}$ and *Verifiers*:$\{tk' = H_1(tk_0), T_w, \delta_t\}$ for $w = 0, \ldots, L-1$, respectively.
- *Time trapdoor release*: Time trapdoor release can be executed in two different modes:
  (1) *Synchronous mode*: According to the pre-determined delivery time schedule, at the end of each t_w, the TTP releases the secret time trapdoor key tk_w.
  (2) *Asynchronous mode*: In this mode, step 2 of the key generation phase is executed in a slightly different than that of in the synchronous mode. That is, the hash chain is generated as $v_w = H_1(v_{w-1} \| w-1)$ for $w = 1, \ldots, L-1$, whose elements are used as the secret time trapdoor keys in the reversed order as $(tk_w, w) = (v_{L-1-w}, L-1-w)$ for $w = 0, \ldots, L-1$. Each tk_w is associated with time interval t_w along with an index w for $w = 0, \ldots, L-1$.

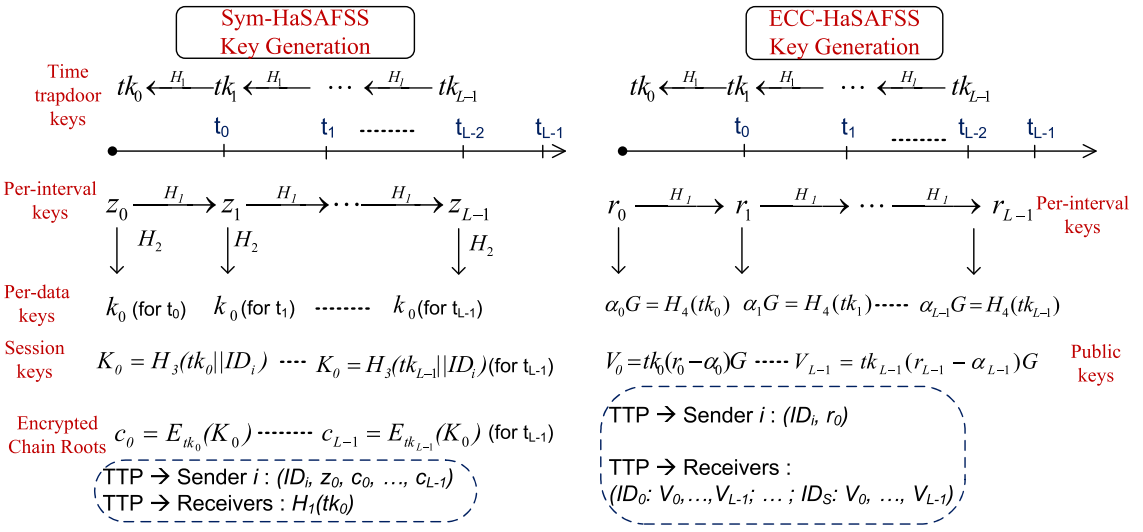

Fig. 1. Sym-HaSAFSS and ECC-HaSAFSS Key generation.

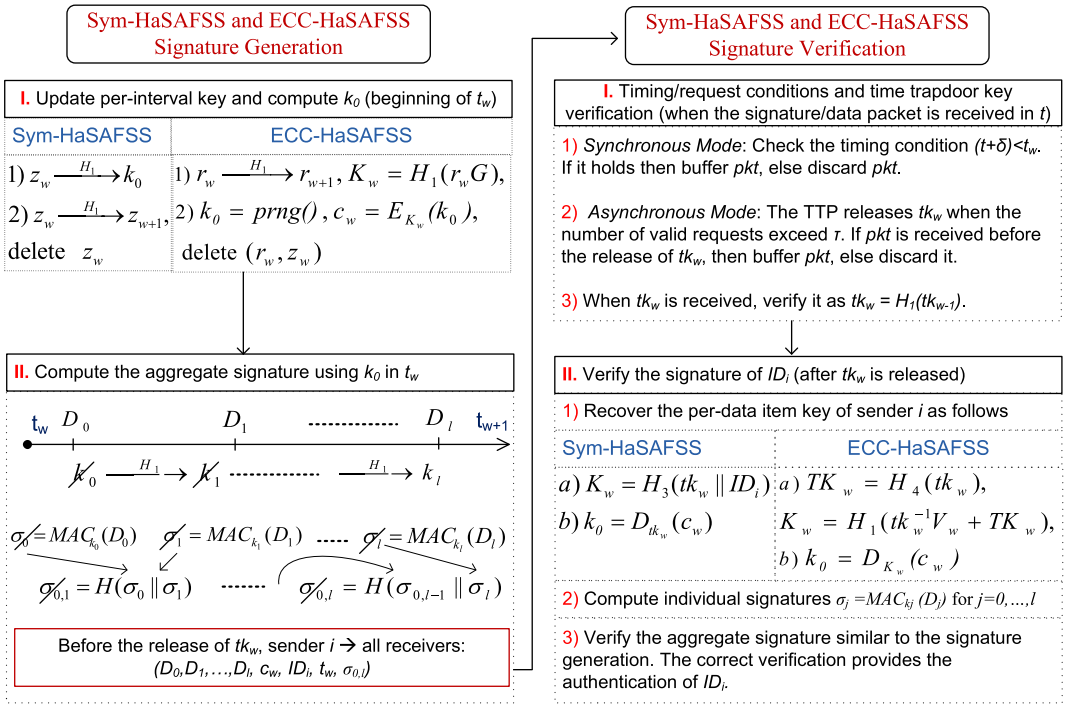

Fig. 2. Sym-HaSAFSS and ECC-HaSAFSS signature generation/verification.

Each verifier sends a request to the TTP for the release of (tk_w, w), when she is done with the data accumulation (or, a mobile TTP visits and requests the data from the verifiers). When the TTP receives more than a threshold number of (authenticated) requests (e.g., $\tau = 90\%$), the TTP releases (tk_w, w). Note that each verifier keeps the index of the last time trapdoor she received a state information (include this index in her trapdoor request as well). This ensures that verifiers and the TTP are synchronized based on the index number despite the mode itself asynchronous (with respect to the time).

- *Forward-secure and aggregate signature generation*:
  (1) At the beginning of t_w, derive the per-data item key as $k_0 = H_2(z_w)$, update the per-interval key as $z_{w+1} = H_1(z_w)$ and delete z_w from the memory.
  (2) Assume that the signer ID_i computed $\sigma_{0,l-1}$ on $D_0, \ldots, D_{l-1}$ in t_w. Compute $\sigma_{0,l}$ on new D_l as $(\sigma_l = MAC_{k_l}(D_l), \sigma_{0,l} = H_3(\sigma_{0,l-1} \| \sigma_l))$, where $k_l = H_1(k_{l-1})$.

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

- *Forward-secure and aggregate signature verification*:
  (1) Assume that the verifier has received pkt at time t. In the synchronous mode, the verifier checks whether the time condition $(t + \delta_t) \leqslant t_w$ holds for $\sigma_{0,l}$. If yes, the verifier buffers pkt and waits for the end of t_w to obtain tk_w from the TTP. In the asynchronous mode, the verifier sends a request to the TTP to obtain tk_w. Note that due to the nature of UWSN applications, there may be a delay before this request is delivered to the TTP (or, the TTP might not be able to visit the verifiers for a long time). In this mode, the verifier can buffer pkt as long as it is received before the release of tk_w.
  (2) When the TTP releases tk_w, each verifier verifies tk_w by checking whether $tk_w \stackrel{?}{=} H_1(tk_{w-1}), w > 0 (w = 0, H(tk_0) = tk')$. If tk_w is verified, then the verifier verifies $\sigma_{0,l}$ as follows: The verifier decrypts c_w by computing $K_w = H_3(tk_w \| ID_i)$ and $k_0 = \mathcal{D}_{K_w}(c_w)$. Using the per-data item key, the verifier computes individual signatures of D_j as $\sigma'_j = MAC_{k_j}(D_j)$ and $k_{j+1} = H_1(k_j)$ for $j = 0, \ldots, l$. Finally, the verifier computes $\sigma'_{0,j} = H_3(\sigma'_{0,j-1} \| \sigma'_j)$ for $j = 1, \ldots, l$, where $\sigma'_{0,0} = \sigma'_0$, and checks $\sigma'_{0,l} \stackrel{?}{=} \sigma_{0,l}$. If they match, the verifier accepts $\sigma_{0,l}$; otherwise, reject.

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

5.3. ECC-HaSAFSS

In contrast to Sym-HaSAFSS, ECC-HaSAFSS addresses the applications where the signers are storage limited while the receivers can afford certain storage [5]. To achieve this, ECC-HaSAFSS uses *pre-computed public keys* instead of pre-computed encrypted chain roots, and at the same it enables each signer to compute her own session keys after the deployment.

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

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

- *Key generation*: The TTP generates tk_w of each t_w for $w = 0, \ldots, L-1$ by following the Sym-HaSAFSS initialization steps. The TTP then generates the public key of each ID_i for each t_w as follows: Generate the initial per-interval key as $r_0 \stackrel{R}{\leftarrow} \mathbb{Z}_q^*$, and compute the public key of each t_w as $V_w = (tk_w \cdot r_w)G - tk_w(\alpha_w)G$, where $(\alpha_w G = H_4(tk_w), r_{w+1} = H_1(r_w))$ for $w = 0, \ldots, L-1$. Give each signer ID_i her own r_0, and give V_w of each ID_i for $w = 0, \ldots, L-1$ to all verifiers.
- *Time trapdoor release*: Same as in Sym-HaSAFSS.
- *Forward-secure aggregate signature generation*:
  (1) At the beginning of t_w, signer ID_i randomly generates a per-data item key k_0, and computes the session key as $K_w = H_1(r_w G)$ and then the encrypted chain root as $c_w = \mathcal{E}_{K_w}(k_0)$. She updates the per-interval key as $r_{w+1} = H_1(r_w)$ and deletes (K_w, r_w) from the memory.
  (2) Signer ID_i computes $\sigma_{0,l}$ on $(D_0, \ldots, D_l)$ for t_w using k_0 by following step 2 in Sym-HaSAFSS signature generation, and then broadcasts $pkt = (D_0, D_1, \ldots, D_l, \sigma_{0,l}, c_w, t_w, ID_i)$ before the end of t_w.
- *Forward-secure and aggregate signature verification*: When a verifier receives pkt, she first checks timing/request conditions for the received packet and verifies tk_w upon its receipt as in step 2 Sym-HaSAFSS signature verification. The verifier then recovers the session key as $K_w = H_1(tk_w^{-1} V_w + H_4(tk_w))$ and decrypts the per-data item key as $k_0 = \mathcal{D}_{K_w}(c_w)$. The verifier verifies $\sigma_{0,l}$ using k_0 by following step 2 in Sym-HaSAFSS signature verification.

Figs. 1 and 2 summarize Sym-HaSAFSS and ECC-HaSAFSS key generation and signature generation/verification steps, respectively.

5.4. SU-HaSAFSS

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

- *Key pre-distribution and limited usage*: In Sym-HaSAFSS, encrypted chain roots are directly computed from the time trapdoor keys. Similarly, in ECC-HaSAFSS, public keys are a function of time trapdoor keys. Hence, in both schemes, these keys have to be pre-computed and distributed before the deployment. The above requirement incurs a linear storage overhead to the signers in Sym-HaSAFSS, and a quadratic storage overhead to the verifiers in ECC-HaSAFSS. In both cases, the storage overhead *grows linearly* with the maximum

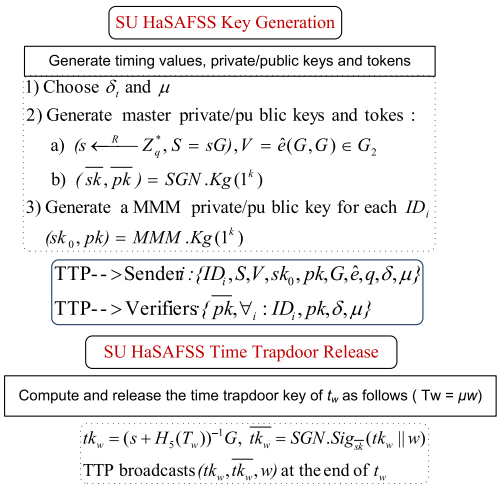

Fig. 3. SU-HaSAFSS Key generation and time trapdoor release.

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

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

5.4.1. SU-HaSAFSS strategy

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

SU-HaSAFSS achieves these goals as follows:

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

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

- *Forward-secure aggregated signature generation (and encryption)*: Assume that the adversary $\mathcal{A}$ breaks-in at time t during $t_{w,w'}$. In contrast to Sym-HaSAFSS and ECC-HaSAFSS, the above self-sustaining strategy allows $\mathcal{A}$ to initialize a new key set independent from the current one (chain roots are no longer generated by the TTP). Therefore, $\mathcal{A}$ can compute a different signature on the data items accumulated in $[t_w,t]$ apart from the existing signature (note that $\mathcal{A}$ still cannot forge the existing aggregate signature computed in $[t_w,t]$).
  SU-HaSAFSS prevents this by using a symmetric cipher along with the forward-secure MAC strategy (i.e., Step 2 in Sym-HaSAFSS signature generation). That is, each D_j is both signed and then encrypted with k_j, and (D_j,k_j) are deleted from the memory. Since k_0 is sealed until the end of $t_{w'}$, $\mathcal{A}$ cannot decrypt $(D_0,\ldots,D_j)$ accumulated in $[t_w, t]$, and therefore cannot compute a different signature on them.
  Another advantage of this approach is that it offers forward-secure encryption and signature simultaneously via symmetric cryptography. Therefore, it is significantly more efficient than all existing forward-secure signcryption[3] schemes (e.g., [44]) with the limitation that it cannot achieve immediate verification.
- *Time trapdoor release and signature verification*: To recover $K_{w,w'}$ at the end of $t_{w,w'}$, we use a pairing-based time trapdoor key structure, which was inspired by AnTRE [26]. Such a time trapdoor key structure allows the derivation of all time trapdoor keys from a single master secret key s without revealing it. When $tk_{w'}$ is released at the end of $t_{w,w'}$, the verifier first removes the mask of $Z_{w,w'}$ using $tk_{w'}$ via a pairing operation (i.e., Step 2 in SU-HaSAFSS signature verification). The verifier obtains k_0 as $k_0 = \mathcal{D}_{K_{w,w'}}(c_{w,w'})$, and then both decrypt and verify data items using k_0.
- *Efficiency*: SU-HaSAFSS preserves the computational efficiency of HaSA-FSS construction over the traditional PKC-based schemes, since the per-item cost is still only three hash operations as in Sym-HaSAFSS[4].

5.4.2. Detailed description

- *Key generation*: Executed by the TTP as follows:
  (1) Choose δ_t and the time trapdoor release period as μ (i.e, a unit time such as 1 h). Every two consecutive time points $T_{i-1} = (i-1)\mu$ and $T_i = i \cdot \mu$ form the ith time interval t_i for $i > 0$, and $t_{w,w'}$ denotes a unified time interval beginning from t_w to the end of $t_{w'}$.

[3] Signcryption is a PKC primitive that simultaneously performs the functions of both digital signature and encryption [43].

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

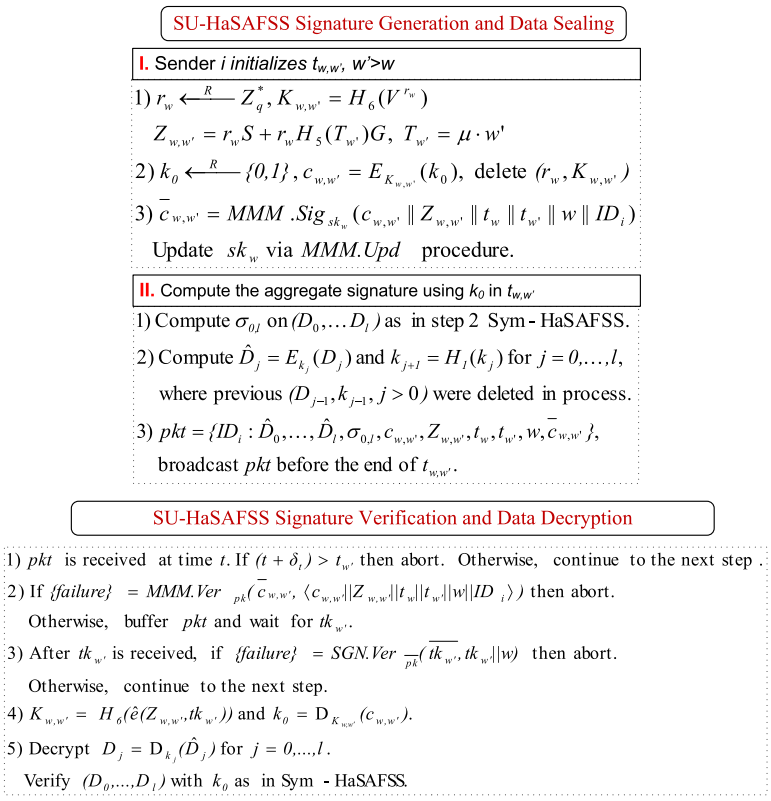

Fig. 4. SU-HaSAFSS signature generation and verification.

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

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

Fig. 3 summarizes key generation and time trapdoor release phases.

- *Forward-secure aggregated signature generation*:
  (1) The signer ID_i initializes an interval $t_{w,w'}, w' > w$:
    (a) Compute the session key of $t_{w,w'}$ as $K_{w,w'} = H_6(V^{r_w})$, where $r_w \xleftarrow{R} \mathbb{Z}_q^*$. Also compute the auxiliary token for $t_{w,w'}$ as $Z_{w,w'} = r_w S + r_w H_5(T_{w'})G$, where $T_{w'} = \mu \cdot w'$ (i.e., the end of $t_{w,w'}$).
    (b) Generate $k_0 \xleftarrow{R} \{0,1\}^n$ and compute $c_{w,w'} = \mathcal{E}_{K_{w,w'}}(k_0)$ for $t_{w,w'}$, and securely erase $(r_w, K_{w,w'})$ from the memory.
    (c) Compute $\bar{c}_{w,w'} = MMM.Sig_{sk_w}(c_{w,w'} \| Z_{w,w'} \| t_w \| t_{w'} \| w \| ID_i)$. Update sk_w following the *MMM* key update procedure.

  (2) Given the current $(\hat{D}_0, \ldots, \hat{D}_{l-1}, \sigma_{0,l-1})$, compute $\hat{D}_l$ and $\sigma_{0,l}$ on newly collected data item D_l as follows:
    (a) Compute $\sigma_{0,l}$ on $(D_l, \sigma_{0,l-1})$ with k_l by following step 2 in Sym-HaSAFSS signature generation.
    (b) Compute $\hat{D}_l = \mathcal{E}_{k_l}(D_l)$, update $k_{l+1} = H_1(k_l)$, and securely erase (D_l, k_l) from the memory. Broadcast $pkt = \{ID_i : \hat{D}_0, \ldots, \hat{D}_l, \sigma_{0,l}, c_{w,w'}, Z_{w,w'}, t_w, t_{w'}, w, \bar{c}_{w,w'}\}$ before the end of $t_{w,w'}$.

- *Trapdoor release*: The TTP computes the time trapdoor key corresponding to t_w as $tk_w = (s + H_5(T_w))^{-1}G$, where $T_w = \mu \cdot w$. The TTP then signs it as $\overline{tk_w} = SGN.Sig_{\overline{sk}}(tk_w \| w)$, and broadcasts $(tk_w, \overline{tk_w}, w)$ at the end of each time period t_w periodically.[5]
- *Signature verification and decryption*: Assume that the verifier received *pkt* at time *t*:
  (1) If $(t + \delta_t) > t_{w'}$ then *abort*. Otherwise, if $\{failure\} = MMM.Ver_{pk}(\bar{c}_{w,w'}, \langle c_{w,w'} \| Z_{w,w'} \| t_w \| t'_w \| w \| ID_i \| \rangle)$ then *abort*. Otherwise, buffer *pkt* and wait the release of $tk_{w'}$. After $tk_{w'}$ is received from the TTP, if $\{failure\} = SGN.Ver_{\overline{pk}}(\overline{tk_{w'}}, tk_{w'} \| w)$ then *abort*, else continue to the next step.
  (2) Recover the session key as $K_{w,w'} = H_6(\tilde{e}(Z_{w,w'}, tk_{w'}))$, and decrypt $k_0 = \mathcal{D}_{K_{w,w'}}(c_{w,w'})$.

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

(3) Decrypt data items as $D_j = \mathcal{D}_{k_j}(\widehat{D}_j)$ for $j = 0,\ldots,l$, and verify $(D_0,\ldots,D_l, \sigma_{0,l})$ with k_0 by following step 2 in Sym-HaSAFSS signature verification.Fig. 4 summarizes SU HaSAFSS signature generation/verification steps.

5.5. Fine-grained verification with HaSAFSS

Forward-secure and aggregate signatures (e.g., [5,9,45,23,41]) verify the data stream via only its final aggregate signature. This prevents the truncation attack (see Section 6.1) and save the storage. However, this approach also causes certain drawbacks:

(i) The verification of any subset of data items requires the verification of the entire data steam. That is, verifiers need to receive the entire data stream to verify an individual data item in this data stream. This may cause verification failures if senders cannot transmit the entire data stream before the release of time trap door key. (ii) The failure of signature verification does not give any information about which data item(s) were corrupted/forged.

Ma et al. developed immutable-FssAgg (iFssAgg) schemes [10] to enable the selective verification of individual data items without being vulnerable to truncation attacks. However, publicly verifiable iFssAgg schemes double the signing/verifying costs of their base schemes. Immutable version of FssAgg-MAC is efficient but it is not publicly verifiable as its base scheme.

A very simple variant of HaSAFSS schemes can achieve the selective verification of data items in a given data stream without being vulnerable to truncation attacks. Note that different than all previous publicly verifiable forward-secure and aggregate signature schemes, the aggregation operation of HaSAFSS schemes is a one-way hash function. Therefore, the signer can just keep partial aggregate signatures $\sigma'_{j,j'}, l > j' > j$ on $(D_j,\ldots,D_{j'})$ and also compute an aggregate signature $\sigma_{0,l}$ on the entire stream $(D_0,\ldots,D_l)$ as usual. Partial aggregate signatures cannot be used to launch a truncation attack against $\sigma_{0,l}$ as they are independent cryptographic hash outputs of different data item subsets.

This allows a storage-selective verification trade-off that can be decided according to the requirement of application.

6. Security analysis

We prove the security of HaSAFSS schemes in three stages:

Lemma 1 proves that no entity, including the signer and the adversary $\mathcal{A}$ even after the break-in in $t_{w,w'}$, can decrypt $c_{w,w'}$ without knowing its corresponding time trapdoor key $tk_{w'}$. That is, no entity can obtain the chain root k_0 before the release of $tk_{w'}$. This guarantees that HaSAFSS schemes introduce the desired asymmetry between the signer and verifiers amd preserves forward security.

Based on Lemmas 1, 2 proves that HaSAFSS schemes remain forward-secure and existential unforgeable during interval $t_{w,w'}$ by regularly updating per-item keys evolved from chain root k_0.

Finally, Theorem 1 proves that the successful verification of $\sigma_{0,l}$ via k_0 guarantees *TFEU* property (i.e., Definition 5) based on Lemma 2.

Lemma 1. *HaSAFSS schemes guarantee the confidentiality of k_0 in the time duration between the releases of tk_{w-1} and $tk_{w'}$ as long as Assumptions 1 and 2 hold.*

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

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

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

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

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

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

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

Lemma 2. *Assume that $\mathcal{A}$ breaks-in at time t during interval $t_{w,w'}$, after $\sigma_{0,l}$ on $(D_0,\ldots,D_l)$ was computed. Producing an existential forgery against HaSAFSS in the time duration between the releases of tk_{w-1} and $tk_{w'}$ is as difficult as breaking either one of the cryptographic hash functions (H_1, H_2, H_3) or MAC.*

Proof. Lemma 1 guarantees that k_0 remains confidential until the end of $t_{w,w'}$. At the same time, the signer regularly updated k_0 for each accumulated data item until $\mathcal{A}$ breaks-in at time t:

Sym-HaSAFSS and ECC-HaSAFSS: Step 2 in Sym-HaSAFSS signature generation updated per-interval and per-item keys as $(z_{l+1} = H_2(z_l), k_{l+1} = H_1(k_l))$, respectively, and then deleted (z_l, k_l) from the memory. Obtaining any previous per-interval key from z_{l+1} is as difficult as breaking H_2. Similarly, obtaining any previous per-item key from k_{l+1} is as difficult as breaking H_1. Without knowing $(k_0,\ldots,k_l)$,

forging $\sigma_{0,l}$ on $(D_0,\ldots,D_l)$ is as difficult as breaking *MAC* function or H_3 (selectively deleting or truncating a data item from $(D_0,\ldots,D_l)$ is subsumed in this forgery). Therefore, Sym-HaSAFSS remains forward-secure and existential unforgeable in $t_{w=w'}$. The signature generation in ECC-HaSAFSS is identical to that of Sym-HaSAFSS, and therefore this analysis also applies to it.

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

Theorem 1. *The verifier receives packet pkt in time t. The successful verification of $\sigma_{0,l}$ on $(D_0,\ldots,D_l)$ guarantees TFEU property (Definition 5) in the time duration between the releases of tk_{w-1} and $tk_{w'}$.*

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

- *Freshness*: The timing condition $(t+\delta_t) < t_{w'}$ (and also the request condition of the asynchronous mode in Sym-HaSAFSS and ECC-HaSAFSS) prevents the verifier from accepting any obsolete signature associated with $tk_{w'}$. That is, if $\mathcal{A}$ breaks-in after the release of $tk_{w'}$, she cannot compute a "valid" signature on $(D_0,\ldots,D_l)$ using any key associated with $t' \leqslant t_{w'}$.
- *Authenticity*: k_0 is obtained from $c_{w,w'}$ via $tk_{w'}$.
  - *Sym-HaSAFSS and ECC-HaSAFSS*: $\forall(w=w')$,tk_w can easily be verified, since they are elements of a hash chain and are released in the reverse order. Since tk_w is authenticated, only an authenticated k_0 can be recovered correctly from c_w via this time trapdoor key. Therefore, the successful verification of $\sigma_{0,l}$ with k_0 also implies that only the claimed ID_i could compute such $\sigma_{0,l}$ before the release of tk_w.
  - *SU-HaSAFSS*: $\forall w', tk_{w'}$ is verified with $\overline{pk}$ via *SGN* to ensure its origin and integrity. Similarly, $\bar{c}_{w,w'}$ is verified with *pk* via *MMM* to ensure the forward-secure integrity and origin of $(c_{w,w'}, Z_{w,w'}, t_w, t_{w'})$. That is, the verifier ensures that the claimed interval $t_{w,w'}$ is correct and $(c_{w,w'}, Z_{w,w'})$ are intact.

Based on Lemma 2 and the fact that k_0 is fresh and authenticated, we prove that HaSAFSS schemes achieve the *TFEU* property. □

6.1. Discussion

6.1.1. Truncation attack

Another security property related to forward-secure and aggregate signatures is the defense against *truncation* attack identified in [45,10]. Truncation attack is a special type of deletion attack, in which $\mathcal{A}$ deletes a continuous subset of accumulated data items. This attack can be prevented via "all-or-nothing" property [5]: $\mathcal{A}$ should either retain all previously accumulated data items, or not use them at all (i.e., $\mathcal{A}$ cannot selectively delete/modify any subset of the data [10]). Lemma 2 proves that HaSAFSS schemes are secure against any type of deletion attack including the truncation attack.

6.1.2. Lack of immediate verification

Despite all the advantages, introducing asymmetry between the signer and verifiers using the time factor brings a natural complication: HaSAFSS schemes cannot provide immediate verification on the verifier side. In order to verify a received signature, a verifier needs to wait for the release of the time trapdoor key corresponding to this signature. However, such a property is compatible with the non-real-time nature of the envisioned UWSN applications. Thus, HaSAFSS schemes are ideal solutions for the envisioned UWSN applications. Note, however, that while delayed detection is intrinsic for UWSNs, it might pose a treat for certain real-life applications such as secure logging [10].

In HaSAFSS schemes, the TTP is assumed to be trusted (i.e., it does not act maliciously against legitimate users). Therefore, the adversary models that include "curious time server" (e.g., [26]) do not apply to HaSAFSS. This allows us to simplify the time trapdoor mechanism used in [26].

7. Performance analysis

In this section, we present the performance analysis of HaSAFSS schemes and compare them with FssAgg schemes (best known alternatives) in terms of their quantitative and qualitative properties. We use the notation in Table 1 for our analysis and comparison. In our experimental evaluation, we use ECDSA [33] as *SGN* and the base signature scheme for *MMM* in SU-HaSAFSS.

7.1. Computational overhead

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

7.1.1. Comparison

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

Tables 2 and 3 compare the computational costs of HaSAFSS schemes with FssAgg schemes analytically and numerically, respectively.

Table 1
Notation used in the performance analysis and comparison of HaSAFSS and FssAgg schemes.

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

Suggested bit lengths to achieve 80-bit security for the above parameters are as follows for each compared scheme: large primes ($|p| = 512$, $|q| = 160$) for ECC-HaSAFSS, SU-HaSAFSS and FssAgg-BLS. Integers ($|n| = 1024$, $x = 160$) for FssAgg-AR and FssAgg-BM, where n is Blum–Williams integer [9].

Table 2
Analytical comparison of HaSAFSS and FssAgg schemes in terms of dominant cryptographic operations.

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

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

In HaSAFSS schemes, the cost of signature generation and verification for a single data item is the same (i.e., only three hash operations). This is much more efficient than PKC-based FssAgg schemes requiring at least one ExpOp per-item and also equally efficient to the FssAgg-MAC. For instance, the signature generation for $l = 10^4$ data items with SU-HaSAFSS is 135, 2996, and 1412 times more efficient than FssAgg-BLS, FssAgg-AR and FssAgg-BM, respectively. Similarly, the signature verification for $l = 10^4$ data items with SU-HaSAFSS is 1554, 1850, and 476 times more efficient than FssAgg-BLS, FssAgg-AR and FssAgg-BM, respectively.

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

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

While being more costly at initialization, SU-HaSAFSS is comparable with Sym-HaSAFSS and ECC-HaSAFSS asymptotically, and it also possesses several qualitative advantages over them as we will discuss in Section 7.3.

7.2. Storage and communication overhead

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

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

Table 3
Execution time (in ms) Comparison of HaSAFSS and FssAgg schemes.

		HaSAFSS			FssAgg			
		Sym	SU	ECC	BLS	AR	BM	MAC
Signer	$l = 10$	0.06	7.83	0.63	10.2	264	128	0.06
	$l = 10^2$	0.6	8.55	1.35	140	25.8×10^2	12.7×10^2	0.6
	$l = 10^3$	6.1	15.75	8.55	11.8×10^2	26.6×10^3	12.5×10^3	6
	$l = 10^4$	61.2	87.77	80.9	11.9×10^3	26.3×10^4	12.4×10^4	60
Verifier	$l = 10$	0.06	17.88	0.63	156	77.1×10^3	524	0.06
	$l = 10^2$	0.6	18.6	1.35	15.4×10^2	78.4×10^3	920	0.6
	$l = 10^3$	6.1	25.8	8.55	14.9×10^3	88.2×10^3	51.6×10^2	6
	$l = 10^4$	61.2	97.8	80.9	15.2×10^4	18.1×10^4	46.6×10^3	60

(i) The execution times were measured on a computer with an Intel(R) Core(TM) i7 Q720 at 1.60 GHz CPU and 2GB RAM running Ubuntu 10.10. We tested HaSAFSS schemes, FssAgg-BLS/MAC [5] using the MIRACL library [46], and FssAgg-AR/BM [9] using the NTL library [47]. Parameter sizes determining the execution times of each scheme were selected to achieve 80-bit security, whose suggested bit lengths were discussed in Table 1. (ii) Execution times are based on the cost of signing/verifying data items accumulated in a given interval $t_{w,w'}$ including the initialization costs.

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

7.2.1. Comparison

Table 4 asymptotically compares HaSAFSS and FssAgg schemes in terms of storage overhead.

From a verifier's perspective, Sym-HaSAFSS, which requires only single key storage, is the most storage efficient scheme among all the compared schemes. SU-HaSAFSS and FssAgg schemes both require linear storage. ECC-HaSAFSS and FssAgg-BLS require quadratic storage and obey the traditional resourceful verifier assumption to address such UWSN applications (e.g., high-end mobile receivers [5]). From a signer's perspective, all compared schemes except for Sym-HaSAFSS and FssAgg-MAC require constant storage.

All compared schemes incur only a constant signature transmission overhead due to their signature aggregation property. Thus, when compared with traditional signature schemes (e.g., [33,48,31]), they are much more communication efficient (data items have to be transmitted in any case and therefore their overhead is not the part of comparison). Note that the signature aggregation also offers "all-or-nothing" property that provides the resilience against the truncation attacks as discussed in Section 6.1.

The communication overhead required to transmit a time trapdoor key is small and constant, since it is broadcasted only once for each time period. In Sym-HaSAFSS and ECC-HaSAFSS, the TTP broadcasts tk_w (i.e., the wth element of the reversed hash chain) at the end of t_w, which incurs $|H|$ (e.g., 160 bits) transmission overhead. In SU-HaSAFSS, the TTP broadcasts the time trapdoor key as $(tk_w, \overline{tk}_w, w)$, which incurs $|q| + |\sigma| + |index|$ (e.g., 160 + 320 + 16 = 500 bits) transmission overhead.

The pre-determined data delivery model and the self-decisive data delivery model do not require a collaborative time trapdoor request mechanism, as the TTP only releases a time trapdoor key passively once for the each time period. Hence, the time trapdoor communication overhead of these models is negligible. However, in the on-demand delivery model, requesting time trapdoor keys authentically from the TTP might incur a non-negligible communication overhead. That is, assuming each verifier sends a request via a pre-computed ECDSA token (which does not require an expensive operation), the total communication overhead of the TTP is $(\tau \cdot S')|\sigma|'$, where $|\sigma|'$ denotes the bit length of ECDSA token signature (e.g., 320 bits).

7.3. Sustainability, applicability and flexibility

In addition to the above quantitative criteria, we also analyze our schemes in terms of some important qualitative properties. Table 5 compares HaSAFSS schemes and FssAgg schemes in terms of the following properties:

7.3.1. Public verifiability

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

7.3.2. Unbounded time period and flexible data delivery schedule

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

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

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

7.3.3. Forward-secure confidentiality

SU-HaSAFSS can integrate forward-secure encryption and forward-secure integrity in a seamless way, since it relies on symmetric cryptography to achieve these goals. Note that to achieve the same property, FssAgg schemes have to resort to costly PKC-based forward-secure encryption schemes (e.g., [49]), which will make these schemes even more expensive.

Table 4
Asymptotic comparison of HaSAFSS and FssAgg schemes in terms of their storage overheads.

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

Table 5
Comparison of HaSAFSS and FssAgg schemes in terms of some important qualitative properties.

	HaSAFSS			FssAgg			
	Sym	SU	ECC	BLS	AR	BM	MAC
Public verifiability	✓	✓	✓	✓	✓	✓	X
Unbounded time periods	X	✓	X	X	X	X	X
Forward-secure confidentiality	X	✓	X	X	X	X	X
Flexible delivery schedule	X	✓	X	✓	✓	✓	✓
Signer storage efficient	X	✓	✓	✓	✓	✓	X
Verifier storage efficient	✓	✓	X	X	✓	✓	✓
Immediate verification	X	X	X	✓	✓	✓	✓

7.3.4. Immediate verification

The main drawback of HaSAFSS schemes is that they cannot achieve immediate verification. A more detailed discussion about this issue was given in Section 6.1. FssAgg schemes achieve immediate verification, since they do not rely on the time factor.

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

8. Conclusion

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

We proposed three HaSAFSS schemes, Sym-HaSAFSS, ECC-HaSAFSS and SU-HaSAFSS in this paper. All these schemes achieve high computational efficiency, low storage and communication overhead, public verifiability, signature aggregation and forward-secure integrity simultaneously. They are significantly more efficient than all of their PKC-based counterparts and still remain publicly verifiable in contrast to other symmetric schemes. Sym-HaSAFSS and ECC-HaSAFSS complement each other by being a signer storage friendly and a verifier storage friendly scheme, respectively. Preserving all other desirable properties of HaSAFSS schemes, SU-HaSAFSS is much more computationally efficient than all of the previous PKC-based counterparts, and additionally achieves unique properties such as unlimited number of time periods, forward-secure confidentiality, and flexible data delivery schedule.

Acknowledgements

We would like to thank Dr. Di Ma who kindly provided her implementation of FssAgg schemes [5,9,10]. We also would like to thank anonymous reviewers of the preliminary version of this paper [23] for their useful comments.

References

[1] D. Pietro, L. Mancini, C. Soriente, A. Spognardi, G. Tsudik, Catch me (if you can): data survival in unattended sensor networks, in: Proceedings of the 6th IEEE International Conference on Pervasive Computing and Communications (PerCom '08), 2008, pp. 185–194.
[2] D. Ma, G. Tsudik, DISH: Distributed self-healing, in: Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS '08), Springer-Verlag, 2008, pp. 47–62.
[3] R.D. Pietro, D. Ma, C. Soriente, G. Tsudik, Posh: proactive co-operative self-healing in unattended wireless sensor networks, in: IEEE Symposium on Reliable Distributed Systems (SRDS '08), 2008, pp. 185–194.
[4] J.C. McEachen, J. Casias, Performance of a wireless unattended sensor network in a freshwater environment, in: Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS '08), IEE, Washington, DC, USA, 2008.
[5] D. Ma, G. Tsudik, Forward-secure sequential aggregate authentication, in: Proceedings of the 28th IEEE Symposium on Security and Privacy (S& P '07), 2007, pp. 86–91.
[6] Trident Systems, Trident's Family of Unattended Ground Sensors, <http://www.tridsys.com/white-unattended-ground-sensors.htm>.
[7] I.P.T.O.I.D.A.R.P.A. (DARPA), BBA 07-46 LANdroids Broad Agency Announcement, 2007, <http://www.darpa.mil/ipto/solicit/baa/BAA-07-46_PIP.pdf>.
[8] M. Bellare, S. Miner, A forward-secure digital signature scheme, in: Advances in Crpytology (CRYPTO '99), Springer-Verlag, 1999, pp. 431–448.
[9] D. Ma, Practical forward secure sequential aggregate signatures, in: Proceedings of the 3rd ACM symposium on Information, Computer and Communications Security (ASIACCS '08), ACM, NY, USA, 2008, pp. 341–352.
[10] D. Ma, G. Tsudik, A new approach to secure logging, ACM Transaction on Storage (TOS) 5 (1) (2009) 1–21.
[11] D. Boneh, C. Gentry, B. Lynn, H. Shacham, Aggregate and verifiably encrypted signatures from bilinear maps, in: Proc. of the 22th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '03), Springer-Verlag, 2003, pp. 416–432.
[12] Y. Mu, W. Susilo, H. Zhu, Compact sequential aggregate signatures, in: Proceedings of the 22nd ACM symposium on Applied computing (SAC '07), ACM, 2007, pp. 249–253.
[13] R. Rivest, A. Shamir, D. Wagner, Time-lock Puzzles and Timed-release Crypto, Tech. Rep., Cambridge, MA, USA, 1996.
[14] R. Anderson, Two remarks on public-key cryptology, invited lecture, in: Proceedings of the 4th ACM conference on Computer and Communications Security (CCS '97), 1997.
[15] M. Abdalla, L. Reyzin, A new forward-secure digital signature scheme, in: Advances in Crpytology (ASIACRYPT '00), Springer-Verlag, 2000, pp. 116–129.
[16] B. Libert, J. Quisquater, M. Yung, Forward-secure signatures in untrusted update environments: Efficient and generic constructions, in: Proceedings of the 14th ACM conference on Computer and communications security (CCS '07), ACM, 2007, pp. 266–275.
[17] G. Itkis, L. Reyzin, Forward-secure signatures with optimal signing and verifying, in: Advances in Cryptology (CRYPTO '01), Springer-Verlag, 2001, pp. 332–354.
[18] H. Krawczyk, Simple forward-secure signatures from any signature scheme, in: Proceedings of the 7th ACM Conference on Computer

and Communications Security, (CCS '00), ACM, pp. 108–115.
[19] A. Kozlov, L. Reyzin, Forward-secure signatures with fast key update, in: Proc. of the 3rd International Conference on Security in Communication Networks (SCN '02), 2002.
[20] T. Malkin, D. Micciancio, S.K. Miner, Efficient generic forward-secure signatures with an unbounded number of time periods, in: Proc. of the 21th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '02), Springer-Verlag, 2002, pp. 400–417.
[21] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, B. Waters, Sequential aggregate signatures and multisignatures without random oracles, in: Proc. of the 25th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '06), Springer-Verlag, 2006, pp. 465–485.
[22] A. Boldyreva, C. Gentry, A. O'Neill, D. Yum, Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing, in: Proceedings of the 14th ACM Conference on Computer and Communications Security, (CCS '07), ACM, 2007, pp. 276–285.
[23] A.A. Yavuz, P. Ning, Hash-based sequential aggregate and forward secure signature for unattended wireless sensor networks, in: Proceedings of the 6th Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous '09), 2009.
[24] T. May, Time Release Crypto, Tech. Rep., February 1993.
[25] H. Varsakelis, K. Chalkias, G. Stephanides, Low-cost anonymous timed-release encryption, in: Proceedings of the 3rd International Symposium on Information Assurance and Security (IAS '07), IEEE Computer Society, 2007, pp. 77–82.
[26] K. Chalkias, D. Hristu-Varsakelis, G. Stephanides, Improved anonymous timed-release encryption, in: 12th European Symposium on Research in Computer Security (ESORICS '07), 2007, pp. 311–326.
[27] S.S. Chow, S.M. Yiu, Timed-release encryption revisited, in: Proceedings of the 2nd International Conference on Provable Security (ProvSec '08), Springer-Verlag, 2008, pp. 38–51.
[28] Y. Hwang, D. Yum, P. Lee, Timed-release encryption with pre-open capability and its application to certified e-mail system, in: Proceedings of the 8th International Conference on Information Security, (ISC '05), Springer-Verlag, 2005, pp. 77–82.
[29] D. Boneh, M. Franklin, Identity-based encryption from the weil pairing, SIAM Journal on Computing 32 (2003) 586–615.
[30] A. Perrig, R. Canetti, D. Song, D. Tygar, Efficient authentication and signing of multicast streams over lossy channels, in: Proceedings of the IEEE Symposium on Security and Privacy, 2000.
[31] C. Schnorr, Efficient signature generation by smart cards, Journal of Cryptology 4 (3) (1991) 161–174.
[32] A. Menezes, P.C. van Oorschot, S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996. ISBN: 0-8493-8523-7.
[33] American Bankers Association, ANSI X9.62-1998: Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1999.
[34] D. Hankerson, A. Menezes, S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, 2004.
[35] O. Goldreich, Foundations of Cryptography, Cambridge University Press, 2001.
[36] M. Bellare, P. Rogaway, Random oracles are practical: a paradigm for designing efficient protocols, in: Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93), ACM, NY, USA, 1993, pp. 62–73.
[37] US National Institute of Standards and Technology, DES modes of operation, Federal Information Processing Standards Publication 81 (FIPS PUB 4-3), December 1980.
[38] J. Katz, Y. Lindell, Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series), Chapman & Hall/CRC, 2007.
[39] D. Boneh, X. Boyen, Efficient Selective-ID secure identity-based encryption without random oracles, in: Proc. of the 23th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '04), 2004, pp. 223–238.
[40] A.A. Yavuz, F. Alagöz, E. Anarim, HIMUTSIS: Hierarchical multi-tier adaptive ad-hoc network security protocol based on signcryption type key exchange schemes, in: Proceedings of the 21th International Symposium Computer and Information Sciences (ISCIS '06), Lecture Notes in Computer Science, vol. 4263, Springer-Verlag, 2006, pp. 434–444.
[41] A.A. Yavuz, P. Ning, BAF: An efficient publicly verifiable secure audit logging scheme for distributed systems, in: Proceedings of 25th Annual Computer Security Applications Conference (ACSAC '09), 2009, pp. 219–228.
[42] E. Mykletun, M. Narasimha, G. Tsudik, Signature bouquets: immutability for aggregated/condensed signatures, in: Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS '04), Springer-Verlag, 2004, pp. 160–176.
[43] Y. Zheng, Digital signcryption or how to achieve cost(signature & encryption) << cost(signature) + cost(encryption), in: Proceedings of Advances in Cryptology (CRYPTO '97), 1997, pp. 165–179.
[44] S.M. Chow, L.C. Hui, S. Yiu, K.P. Chow, Forward-secure multisignature and blind signature schemes, Applied Mathematics and Computation 168 (2) (2005) 895–908.
[45] D. Ma, G. Tsudik, A new approach to secure logging, in: Proc. of the 22nd Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSEC '08), 2008, pp. 48–63.
[46] Shamus, Multiprecision Integer and Rational Arithmetic c/c++ Library (MIRACL), <http://www.shamus.ie/>.
[47] V. Shoup, NTL: A Library for Doing Number Theory, <http://www.shoup.net/ntl/>.
[48] R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.
[49] R. Canetti, S. Halevi, J. Katz, A forward-secure public-key encryption scheme, Journal of Cryptology 20 (3) (2007) 265–294.

Dr. Attila Altay Yavuz graduated from North Carolina State University (NCSU) with a PhD degree in Computer Science. He received a BS degree in Computer Engineering from Yildiz Technical University in 2004 and a MS degree in Computer Science from Bogazici University in 2006, both in Turkey.

Dr. Attila A. Yavuz is interested in design, analysis and application of cryptographic primitives and protocols to enhance the security of computer networks and systems. His current research is focus on the developing efficient cryptographic primitives to provide the security in Virtual Computing Clouds (VCC), wireless networks, cyber-physical systems, and digital forensics.

Dr. Peng Ning is a Professor of Computer Science at NC State University, located in Raleigh, NC, USA, where he also serves as the Technical Director for Secure Open Systems Initiative (SOSI) in College of Engineering at NC State University. He joined NC State University in August 2001 after he graduated from George Mason University with a PhD degree in Information Technology. Peng Ning received a BS degree in Information Science and an ME degree in Communication and Electronic System in 1994 and 1997, respectively, both from University of Science and Technology of China.

Peng Ning's research interests are mainly in computer and network security. He is a recipient of NSF CAREER award. His research has been supported by the National Science Foundation (NSF), the Army Research Office (ARO), the Advanced Research and Development Activity (ARDA), IBM Open Collaboration Research (OCR) program, SRI International, and the NCSU/Duke Center for Advanced Computing and Communication (CACC). He was elected the Secretary/Treasurer of the ACM Special Interest Group on Security, Auditing and Control (SIGSAC) in 2009. He served/or is serving on the editorial boards of ACM Transactions on Sensor Networks, Journal of Computer Security, Ad-Hoc Networks, Ad-Hoc & Sensor Networks: an International Journal, International Journal of Security and Networks, and IET Proceedings Information Security. Peng Ning served as the Program Chairs or Co-Chairs of ESORICS '09, ACM SASN '05 and ICICS '06, the General Chair of ACM CCS '07 and CCS '08, and Program Vice Chair for ICDCS '09 & '10-Security and Privacy Track. He is a Steering Committee member of ACM CCS and a founding Steering Committee member of ACM WiSec. He has served on the organizing committees or program committees for over fifty technical conferences or workshops related to computer and network security. Peng Ning is a senior member of the ACM, the ACM SIGSAC, and a member of the IEEE and the IEEE Computer Society.