Hash-Based Sequential Aggregate and Forward Secure Signature for Unattended Wireless Sensor Networks

Attila Altay Yavuz and Peng Ning
Department of Computer Science
North Carolina State University
Raleigh, NC 27695, USA
Email: {aayavuz, pning}@ncsu.edu

***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 between senders (sensors) and receivers (e.g., mobile data collectors, static sinks). The lack of real-time communication forces sensors to accumulate the sensed data possibly for long time periods, along with the corresponding signatures for authentication purposes. Moreover, non-real-time characteristic of UWSNs makes sensors vulnerable especially to *active adversaries*, which compromise sensors and extract all data stored in them. Hence, it is critical to have *forward security* property such that even if the adversary can compromise the current keying materials, she cannot modify or forge authenticated data generated before the node compromise. Forward secure and aggregate signatures are cryptographic primitives developed to address these issues. Unfortunately, existing forward secure and aggregate signature schemes either impose substantial computation and storage overhead, or do not allow public verifiability, thereby impractical for resource-constrained UWSNs.**

In order to address these problems, we propose a new class of signature schemes, which we refer to as *Hash-Based Sequential Aggregate and Forward Secure Signature (HaSAFSS)*. Such a scheme allows a signer to sequentially generate a compact, fixed-size, and publicly verifiable signature at a nearly optimal computational cost. We propose two HaSAFSS schemes, Symmetric HaSAFSS (Sym-HaSAFSS) and Elliptic Curve Cryptography (ECC) based HaSAFSS (ECC-HaSAFSS). Both schemes integrate the efficiency of MAC-based aggregate signatures and the public verifiability of bilinear map based signatures by preserving forward security via Timed-Release Encryption (TRE). We demonstrate that our schemes are secure under appropriate computational assumptions. We also show that our schemes are significantly more efficient in terms of both computational and storage overheads than previous schemes, and therefore quite practical for even highly resource-constrained UWSN applications.

***Index Terms*—Unattended Wireless Sensor Networks (UWSNs), security and privacy, digital signatures, forward security, signature aggregation.**

I. 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. In these time periods, sensors accumulate the sensed data and then transmit it to the receivers, whenever they become available to sensors. Receivers can collect the sensed data from sensors via, for example, mobile collectors visiting the network periodically [2], [5].

Examples of UWSNs can be found in military WSN applications (e.g., [1], [6]), where sensors are deployed to an adversarial and unattended environment to gather information about enemy activities (e.g., underground, underwater and airborne UWSNs to detect enemy vehicles as well as nuclear/chemical activities). One illustrative example is LANdroids [7], a recent U.S. Defense Advanced Research Projects Agency (DARPA) research project, which designs smart robotic radio relay nodes for the battlefield deployment. These nodes are deployed in hostile environments, gather military information and then uploads it to ally vehicles (e.g., UAV, tank or soldier) upon their arrival.

The lack of real-time communication and the resource constraints of the UWSNs bring several security and performance challenges, especially when an UWSN is deployed in a hostile environment as described above. In particular, inability to offload 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 sender digitally signs each data item as soon as it is accumulated. The sender then evolves its secret key (which implies the deletion of the previous keys) and uses the new key to sign the next data item. Consecutively signing accumulated data items also brings significant storage and communication overheads because of the accumulation of the signatures of individual data items. Aggregate signature schemes [9] were developed to address this issue by aggregating the signatures of different data items into a single small-size signature.

All 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, almost all existing forward secure and/or aggregate signature schemes (e.g., [5], [8], [9]) impose extreme computational, storage, and communication overheads on the network entities, which are intolerable for resource-constrained UWSN applications. The only exception is FssAgg-MAC [5], which achieves computational efficiency through hash chains and symmetric key distribution. Unfortunately, despite its computational efficiency, FssAgg-MAC has high storage overhead and does not allow the signatures to be publicly verifiable, limiting its applicability. Thus, it is necessary to seek more flexible and efficient forward secure and aggregate signatures for UWSN applications.

In this paper, we propose a new class of digital signature schemes for UWSN applications, which we call *H̲ash-Based S̲equential A̲ggregate and F̲orward S̲ecure S̲ignatures (HaSAFSS, pronounced "Hasafass")*. We develop two specific HaSAFSS schemes, a *symmetric HaSAFSS* scheme (called *Sym-HaSAFSS*) and an *ECC-based HaSAFSS* scheme (called *ECC-HaSAFSS*). A nice property of these schemes is that they achieve three seemingly conflicting goals, computational efficiency, public verifiability and forward security, 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) [10]. Using this asymmetry, our schemes achieve high efficiency by avoiding costly Public Key Cryptographic (PKC) operations, while still remaining publicly verifiable and forward secure.

We summarize the properties of our schemes as follows:

- 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., [9], FssAgg-BLS in [5]), with the exception of FssAgg-MAC in [5]. When compared with FssAgg-MAC [5], our schemes achieve public verifiability by eliminating symmetric key distribution, and therefore are much more applicable for ubiquitous systems. Note that our schemes preserve the computational efficiency of FssAgg-MAC [5], while achieving these goals.
- In our schemes, both senders and receivers get equal benefits of computational efficiency, while most existing schemes incur extremely heavy computational overhead on the receiver side. This property is especially useful for the UWSN applications in which the receivers need to verify large amounts of data efficiently.
- Besides the computational efficiency, our schemes are also storage and bandwidth efficient: (i) Since HaSAFSS schemes achieve the signature aggregation, a sender always stores and transmits only a single compact signature, regardless of the number of time periods or data items to be signed. This property provides an advantage for the bandwidth limited systems. (ii) ECC-HaSAFSS is a sender friendly scheme that requires storing only one key per sender. In contrast, Sym-HaSAFSS requires storing one key for each receiver by offering an alternative receiver friendly scheme (the storage overhead on the sender side is still plausible). Hence, ECC-HaSAFSS and Sym-HaSAFSS complement each other in terms of storage overhead.

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/bandwidth efficiency are more important than immediate verification.

The remainder of this paper is organized as follows. Section II briefly presents related work. Section III discusses notation, security and data models. Section IV describes the proposed schemes in detail. Section V provides the security analysis of the proposed schemes. Section VI gives performance analysis and compares the proposed schemes with previous approaches. Section VII concludes this paper.

II. Related Work

In this section, we first give an overview of the existing forward secure and aggregate signature schemes and then review the notion of Timed-Release Encryption (TRE) used in our schemes. We then briefly compare our schemes with TESLA [11], which also uses the time factor to achieve efficient source authentication. Last, we give a short review of self-healing schemes proposed for the UWSNs.

Forward Secure and Aggregate Signatures: A forward secure signature aims to minimize the effect of key compromises. The first forward secure signature scheme was proposed in [12]. 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], [13], [14]). However, all existing forward secure signatures are either computationally expensive or introduce high storage overhead.

Another important digital signature primitive is aggregate signature, which aggregates n individual signatures associated with n different data items into a single, compact signature. The first aggregate signature scheme was proposed in [9], which utilizes the BLS (Boneh-Lynn-Shacham) signatures [15]. Various new aggregate signature schemes have been developed to offer different properties such as sequentiality (order preserving) [16], low storage overhead [17], and forward security [5]. Despite their attractive properties, all these schemes are also computationally expensive due to the heavy use of BLS operations.

Recently, Ma et al. [5] proposed the first signature schemes that achieve signature aggregation and forward security simultaneously, motivating them for the efficient data integrity and authentication in UWSNs: FssAgg-BLS and FssAgg-MAC.

FssAgg-BLS uses hash chains and BLS based signatures [9] to compute and verify aggregate signatures, thereby extremely costly and impractical for the envisioned UWSN applications. In contrast, FssAgg-MAC archives high computational efficiency by utilizing hash chains and MAC functions to compute and verify signatures via symmetric key pre-distribution. Despite its effectiveness, FssAgg-MAC does not allow the signatures to be publicly verifiable and incurs significant storage overhead. Inspired by FssAgg-MAC scheme [5], our approach achieves low storage overhead, public verifiability as well as computational efficiency.

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 predefined 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 [10]. Hence, users can decrypt the ciphertext when its related trapdoor information is released by the TA. Most of the recent TRE schemes are based on Identity-Based Encryption (IBE) [18] and bilinear map (e.g. [19]). In order to achieve efficient solutions, we avoid using expensive IBE-TRE schemes. Instead, we only use the basic TRE concept to fulfill our requirements.

TESLA: TESLA [11] is an efficient broadcast authentication protocol that also uses delayed disclosure of the keying material, assuming that senders and receivers are loosely synchronized. However, our schemes provide important properties that are not available in TESLA. For example, TESLA cannot achieve forward security and signature aggregation, thereby cannot address requirements of the aforementioned UWSN applications. Also, TESLA cannot be used for UWSN applications when loose time synchronization cannot be guaranteed for the network entities.

Self-Healing Techniques: Recently, a series of studies [1]–[3] 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 [2], [3] 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/analytical results for them.

Note that the adversary models and security goals in [1]–[3] are different from ours. In our schemes (similar to FssAgg [5]), the goal of the adversary is to forge data and/or destroy the authentication. However, the goal of the adversary in [1]–[3] is to prevent the data from reaching the sink (not modifying or forging it).

III. Notation, Security and Data Models

Notation: G is a generator on an Elliptic Curve *(EC)* over a prime field 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.* 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., $H_4 : \{0,1\}^n \rightarrow \alpha G$. We also use a secure MAC to compute individual signatures of data items, defined as $MAC_{sk} : \{0,1\}^n \times \{0,1\}^* \rightarrow \{0,1\}^{|H|}$. Last, we use E, D, $||$, and $|x|$ to denote symmetric encryption function, symmetric decryption function, concatenation operation, and the bit length of variable x, respectively.

Security Model: The security model of our schemes based on the following assumptions and definitions:

Definition 1: Adversary $\mathcal{A}$ is a resourceful but Probabilistic Polynomial Time (PPT) bounded adversary having the following abilities: (i) passive attacks against output of cryptographic operations, (ii) active attacks including packet interception/drop/modification, and (iii) physically compromising senders/receivers (called as *"break-in"*) and extracting the cryptographic keys from the compromised nodes. $\mathcal{A}$ aims to forge the extracted data using cryptographic keys obtained from all senders and receivers.

Assumption 1: $H_1, ..., H_4$ are secure, strong collision-free hash functions producing indistinguishable outputs from the random uniform distribution. MAC cannot be forged by $\mathcal{A}$ without knowing sk. Symmetric encryption function E is an ideal cipher, which cannot be decrypted by $\mathcal{A}$ without knowing the secret key. We also assume that the *Elliptic Curve Discrete Logarithm Problem (ECDLP)* [20], [21] is intractable with appropriate parameters.

Assumption 2: We assume a Trusted Third Party (TTP), which is trusted by all network entities. (i) $\mathcal{A}$ 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. We assume that time trapdoor keys released by the TTP reach the receivers eventually. Details of the time trapdoor key delivery are given in data models.

Definition 2: The forward security objective of our schemes is to achieve the *per-data item forward security*. That is, in a given time interval t_w, senders sign each collected data item *as soon as it is received* and updates the signing key. This strategy provides forward security of each individual data item collected in a given time interval t_w (not across intervals), which is different from the *per-time period forward security* objective of FssAgg schemes [5] [1]. Detailed security analysis of this approach is given in Section V.

Remark 1: The per-data item forward security objective requires that senders should transmit the aggregate signature computed in t_w to the receivers, before the TTP releases the

[1] In FssAgg [5] schemes, the signing key is updated once for each time period, not for the each collected data item. Thus, if $\mathcal{A}$ breaks-in in t_w, then she can forge all data items accumulated-so-far from the beginning of t_w. However, $\mathcal{A}$ cannot forge data items accumulated in $t < t_w$.

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]. Details of how our schemes handle data/time trapdoor information are given in data models and Section IV-B.

Data Models: We consider two data delivery models for our envisioned UWSN applications:

(a) *Synchronous Data Delivery Model:* This model addresses applications in which senders/receivers and the TTP can agree on a prospective data delivery schedule so that data/trapdoor delivery can be performed based on this predetermined schedule. In this model, the TTP *passively broadcasts* time trapdoor keys periodically, and it is assumed that the receivers (e.g., mobile collectors) are able to visit sensors based on this pre-determined schedule.

(b) *Asynchronous 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 receivers 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, receivers can obtain time trapdoor keys from a high-end and mobile TTP directly (e.g., MTC (Mobile Tactical Center) [22]). Thus, Assumption 2-(iii) is realistic.

Remark 2: Senders do not need to communicate with the TTP. Receivers communicate with the TTP only in asynchronous data delivery model, only once for each time period (the TTP is offline majority of the operation time).

IV. Proposed Schemes

In this section, we present the proposed schemes, Sym-HaSAFSS and ECC-HaSAFSS. Before giving the detailed description, we first give an overview of these schemes.

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

The HaSAFSS schemes consist of four phases: Initialization, signature generation, time trapdoor release, and signature verification.

Initialization Phase: In this phase, the TTP prepares and distributes necessary cryptographic keys to the senders and the receivers. In HaSAFSS, the TTP divides the time-line into multiple time periods, each of which is associated with four types of cryptographic keys: *time trapdoor keys, per-interval keys, per-data item keys* and *session keys*.

(i) *Time trapdoor keys:* Time trapdoor keys are the essential instrument to introduce the desired asymmetry, utilizing the basic TRE concept. Each time trapdoor key tk_w is associated with the time period t_w and is released according to the application requirements. Time trapdoor keys introduce desired asymmetry via per-data item, per-interval, and session keys.

(ii) *Per-interval keys:* Each sender uses a per-interval key for each time period. The first per-interval key is given to the sender by the TTP. The sender then "evolves" (with a hash operation) the per-interval key as the time goes from one period into the next. The objective of the per-interval key is to provide a fresh initialization key for each time period, from which the other required keys are derived for that time interval.

(iii) *Per-data item keys:* Each sender uses a per-data item key for each data item. The first per-data item key in each time interval, called the *chain root* of the per-data item keys in that time interval, is either derived from the per-interval key (Sym-HaSAFSS) or randomly generated (ECC-HaSAFSS) at the beginning of the time interval. The per-data item key also "evolves" through hash operations on a per-data item basis. Each sender computes individual signatures of the accumulated data items with a MAC operation using the per-data item keys, which provide forward security of these data items for the given time period.

Sym-HaSAFSS and ECC-HaSAFSS complement each other in terms of storage overhead by computing these keys in different ways. In Sym-HaSAFSS, the TTP directly provides the per-data item keys to the senders in the encrypted form. In ECC-HaSAFSS, the senders can compute their own session and per-data item keys themselves. Figure 1 illustrates the computation of these keys in HaSAFSS schemes.

Time Trapdoor Release: Receivers need the time trapdoor keys to verify aggregate signatures. The TTP releases time trapdoor keys in two modes following the two data delivery models, whose details are given in Section III: (i) In *the synchronous mode*, the TTP can release the time trapdoor keys based on a pre-determined data delivery schedule periodically, without requiring an interaction with the receivers. (ii) In *the asynchronous mode*, the TTP releases the time trapdoor key if and only if it is requested by a sufficient number of valid receivers.

Signature Generation Phase: HaSAFSS schemes achieve computational efficiency, since the aggregate signature generation/verification steps rely on symmetric primitives only (signature generation step 2). Figure 2 summarizes signature generation/verification phases.

At the beginning of each time period t_w, the sender updates her per-interval key and either derives the first per-data item key (i.e., the chain root) from the per-interval key (Sym-HaSAFSS) or generates it randomly (ECC-HaSAFSS). Using this per-data item key, the sender computes aggregate signatures of individual data items as follows: The sender signs each accumulated data item individually by computing its MAC using the corresponding per-data item key and updates her per-data item key with a hash operation (and deletes the

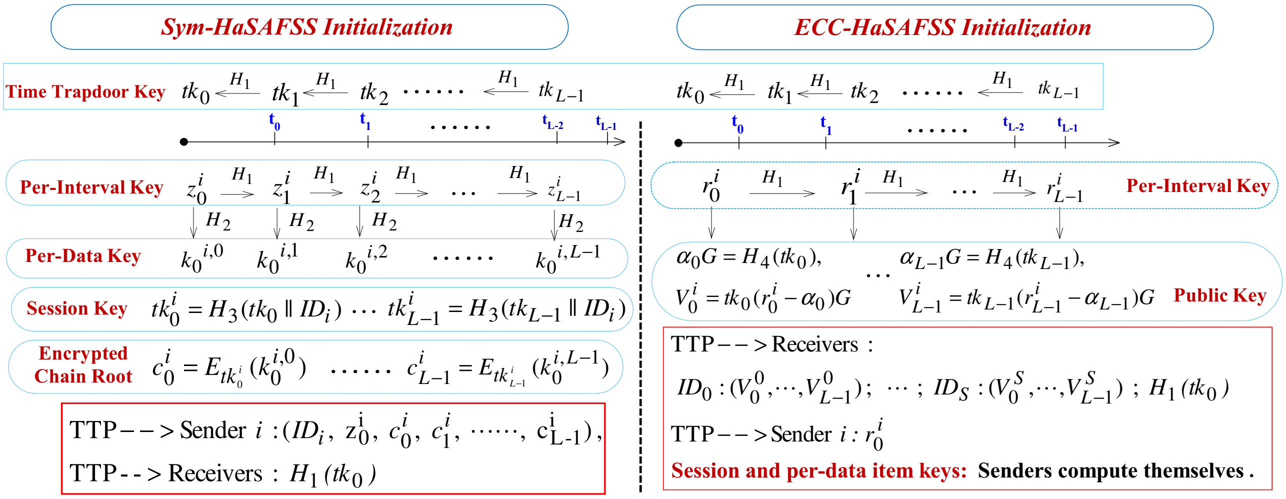
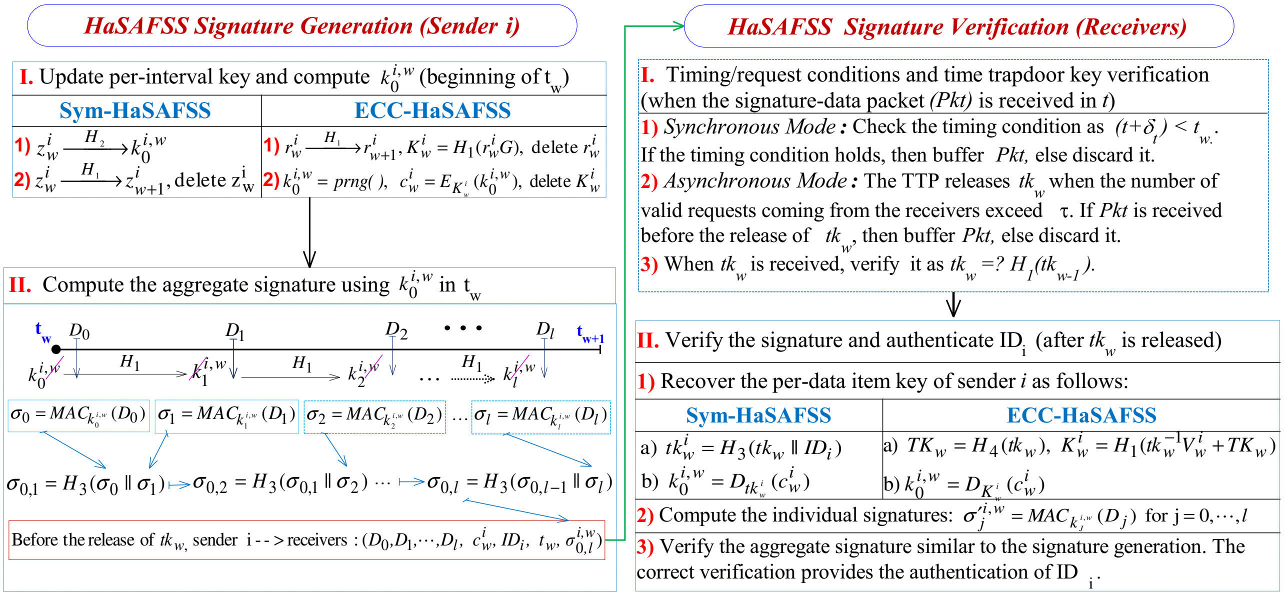

Sym-HaSAFSS Initialization
ECC-HaSAFSS Initialization
Time Trapdoor Key
$tk_0 \xleftarrow{H_1} tk_1 \xleftarrow{H_1} tk_2 \cdots\cdots \xleftarrow{H_1} tk_{L-1}$
t_0 t_1 t_{L-2} t_{L-1}
Per-Interval Key
$z_0^i \xrightarrow{H_1} z_1^i \xrightarrow{H_1} z_2^i \xrightarrow{H_1} \cdots \xrightarrow{H_1} z_{L-1}^i$
H_2
Per-Data Key
$k_0^{i,0}$ $k_0^{i,1}$ $k_0^{i,2}$ $\cdots\cdots$ $k_0^{i,L-1}$
Session Key
$tk_0^i = H_3(tk_0 \parallel ID_i) \cdots tk_{L-1}^i = H_3(tk_{L-1} \parallel ID_i)$
Encrypted Chain Root
$c_0^i = E_{tk_0^i}(k_0^{i,0}) \cdots\cdots c_{L-1}^i = E_{tk_{L-1}^i}(k_0^{i,L-1})$
TTP-- > Sender i : $(ID_i,\ z_0^i,\ c_0^i,\ c_1^i,\ \cdots\cdots,\ c_{L-1}^i)$,
TTP-- > Receivers : $H_1(tk_0)$
$r_0^i \xrightarrow{H_1} r_1^i \xrightarrow{H_1} \cdots \xrightarrow{H_1} r_{L-1}^i$
Per-Interval Key
$\alpha_0 G = H_4(tk_0)$, $V_0^i = tk_0(r_0^i - \alpha_0)G$
$\cdots$
$\alpha_{L-1}G = H_4(tk_{L-1})$, $V_{L-1}^i = tk_{L-1}(r_{L-1}^i - \alpha_{L-1})G$
Public Key
TTP-- > Receivers :
$ID_0 : (V_0^0, \cdots, V_{L-1}^0)$; $\cdots$; $ID_S : (V_0^S, \cdots, V_{L-1}^S)$; $H_1(tk_0)$
TTP-- > Sender i : r_0^i
Session and per-data item keys: Senders compute themselves .

Fig. 1. Initialization phase of Sym-HaSAFSS and ECC-HaSAFSS


HaSAFSS Signature Generation (Sender i)
HaSAFSS Signature Verification (Receivers)
t_w D_0 D_1 D_2 D_l t_{w+1}
$k_0^{i,w} \xrightarrow{H_1} k_1^{i,w} \xrightarrow{H_1} k_2^{i,w} \cdots \xrightarrow{H_1} k_l^{i,w}$

I. Update per-interval key and compute $k_0^{i,w}$ (beginning of t_w)

Sym-HaSAFSS	ECC-HaSAFSS
1) $z_w^i \xrightarrow{H_2} k_0^{i,w}$	**1)** $r_w^i \xrightarrow{H_1} r_{w+1}^i$, $K_w^i = H_1(r_w^i G)$, delete r_w^i
2) $z_w^i \xrightarrow{H_1} z_{w+1}^i$, delete z_w^i	**2)** $k_0^{i,w} = prng(\,)$, $c_w^i = E_{K_w^i}(k_0^{i,w})$, delete K_w^i

II. Compute the aggregate signature using $k_0^{i,w}$ in t_w

$\sigma_0 = MAC_{k_0^{i,w}}(D_0)$ $\sigma_1 = MAC_{k_1^{i,w}}(D_1)$ $\sigma_2 = MAC_{k_2^{i,w}}(D_2)$ $\ldots$ $\sigma_l = MAC_{k_l^{i,w}}(D_l)$

$\sigma_{0,1} = H_3(\sigma_0 \parallel \sigma_1) \mapsto \sigma_{0,2} = H_3(\sigma_{0,1} \parallel \sigma_2) \cdots \mapsto \sigma_{0,l} = H_3(\sigma_{0,l-1} \parallel \sigma_l)$

Before the release of tk_w, sender i -- > receivers : $(D_0, D_1, \cdots, D_l,\ c_w^i,\ ID_i,\ t_w,\ \sigma_{0,l}^{i,w})$

I. Timing/request conditions and time trapdoor key verification (when the signature-data packet *(Pkt)* is received in *t*)

1) *Synchronous Mode* **:** Check the timing condition as $(t+\delta_t) < t_{w}$. If the timing condition holds, then buffer *Pkt*, else discard it.

2) *Asynchronous Mode* **:** The TTP releases tk_w when the number of valid requests coming from the receivers exceed τ. If *Pkt* is received before the release of tk_w, then buffer *Pkt*, else discard it.

3) When tk_w is received, verify it as $tk_w =? H_1(tk_{w-1})$.

II. Verify the signature and authenticate ID_i (after tk_w is released)

1) Recover the per-data item key of sender *i* as follows:

Sym-HaSAFSS	ECC-HaSAFSS
a) $tk_w^i = H_3(tk_w \parallel ID_i)$	a) $TK_w = H_4(tk_w)$, $K_w^i = H_1(tk_w^{-1}V_w^i + TK_w)$
b) $k_0^{i,w} = D_{tk_w^i}(c_w^i)$	b) $k_0^{i,w} = D_{K_w^i}(c_w^i)$

2) Compute the individual signatures: $\sigma_j'^{i,w} = MAC_{k_j^{i,w}}(D_j)$ for $j = 0, \cdots, l$

3) Verify the aggregate signature similar to the signature generation. The correct verification provides the authentication of ID_i.

Fig. 2. Overview of the signature generation/verification phases of HaSAFSS schemes

previous one). The sender folds individual signature of the newly collected data item into the existing aggregate signature by concatenating and hashing them together. This procedure is shown in Figure 2 signature generation step 2.

Signature Verification Phase: In the signature verification phase, the receivers obtain and verify the time trapdoor key from the TTP. Using the identity of the sender and the time trapdoor key, each receiver computes the session key and decrypts the encrypted chain root of the sender for the given time interval. At this stage, the receivers possess per-data item keys of the sender for the time interval, which were used to compute the aggregate signature. Thus, they can verify the aggregate signature by following the same procedure as in the signature generation phase. This allows both senders and receivers to equally benefit from the computational efficiency.

Remark 3: After the release of the time trapdoor key, the receivers never accept any signature associated with this time trapdoor key from any sender. This condition (the timing condition in the synchronous mode and the request condition in the asynchronous mode) prevents $\mathcal{A}$ from forging data items accumulated in the previous time intervals. At the same time, regularly updating per-data item keys prevent her from forging data items accumulated in the given time interval. Detailed security analysis of this approach is given in Theorem 1.

B. Sym-HaSAFSS

The motivation behind Sym-HaSAFSS is to achieve three seemingly conflicting goals at the same time *without* using any PKC operations: Public verifiability, high efficiency, and forward security. To achieve this, Sym-HaSAFSS protects per-data item keys via *encrypted chain roots*, which are decrypted and publicly verified by the receivers with the release of time trapdoor keys.

The TTP generates the initial per-interval key z_0^i for sender i. The per-interval key is updated when the time goes into a new time interval. In each time interval t_w, the per-interval key is used to derive the chain root $k_0^{i,w}$ (i.e., the first per-data item key in t_w), which is then used to compute the aggregate signatures in t_w. The per-data item key is refreshed for each new data item. In order to protect $k_0^{i,w}$, the TTP encrypts each $k_0^{i,w}$ with tk_w^i to obtain the *encrypted chain root* c_w^i.

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. Near-optimal computational efficiency of Sym-HaSAFSS makes it an ideal choice for resource-constrained UWSN applications.

Besides its computational efficiency, Sym-HaSAFSS is also a storage efficient scheme. Different from the PKC-based approaches, in which receivers are required to store a large number of public key sets, the storage requirement of Sym-HaSAFSS for each receiver is *only one key*. The storage requirement of Sym-HaSAFSS for each sender is also plausible for the envisioned UWSN applications ($(L-w)$ keys in time period t_w).

We present the detailed description below:

Initialization Phase:

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

2) The TTP randomly generates a hash chain $v_w = H_1(v_{w-1})$ for $w = 1, ..., L-1$, whose elements will be used as the secret time trapdoor keys in the reversed order as $tk_w = v_{L-1-w}$ for $w = 0, ..., L-1$. Each tk_w is associated with time interval t_w for $w = 0, ..., L-1$. The TTP computes the encrypted chain roots for each sender i as follows:

a) Generate the n-bit initial per-interval key z_0^i for each sender i. The chain root for each time interval t_w is derived as $k_0^{i,w} = H_2(z_w^i)$ and $z_{w+1}^i = H_1(z_w^i)$ for $w = 0, ..., L-1$.

b) Compute the session key as $tk_w^i = H_3(tk_w || ID_i)$ and the encrypted chain root of sender i for time period t_w as $c_w^i = E_{tk_w^i}(k_0^{i,w})$, where $w = 0, ..., L-1$.

3) The TTP gives the commitment and maximum clock synchronization error $(H_1(tk_0), \delta_t)$ to all receivers, and the per-interval key, encrypted chain roots and maximum clock synchronization error $(ID_i, z_0^i, c_0^i\ , c_1^i, ..., c_{L-1}^i, \delta_t)$ to each sender i.

Time Trapdoor Release Phase: This phase can be executed in two different modes:

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

(2) *Asynchronous Mode:* Each receiver sends a request to the TTP for the release of the secret time trapdoor key, when she is ready to transmit the data (or, a mobile TTP visits and requests the data from the receivers). When the TTP receives more than a threshold number of (authenticated) requests (e.g., $\tau = 90\%$), the TTP releases the secret time trapdoor key.

Signature Generation Phase:

1) At the beginning of time interval t_w, sender i derives the per-data item key as $k_0^{i,w} = H_2(z_w^i)$, updates the per-interval key as $z_{w+1}^i = H_1(z_w^i)$ and deletes z_w^i from memory.

2) Assume that sender i has accumulated data items $D_0, D_1, \ldots, D_{l-1}$ and computed the aggregate signature $\sigma_{0,l-1}^{i,w}$ in t_w. When sender i collects D_l, she first computes the individual signature of D_l as $\sigma_l^{i,w} = MAC_{k_l^{i,w}}(D_l)$, where $k_l^{i,w} = H_1(k_{l-1}^{i,w})$, and then folds $\sigma_l^{i,w}$ into $\sigma_{0,l-1}^{i,w}$ as $\sigma_{0,l}^{i,w} = H_3(\sigma_{0,l-1}^{i,w} || \sigma_l^{i,w})$. In the synchronous mode, all keys and signatures associated with t_w expire at the end of t_w. Thus, sender i must transmit $(D_0, D_1, \ldots, D_l, \sigma_{0,l}^{i,w}, c_w^i\ , t_w, ID_i)$ before t_w ends. In the asynchronous mode, senders can transmit it any time before the TTP releases tk_w (However, the sender transmits it too late, she may miss the opportunity to have receivers accept it if the transmission is after the trapdoor release.).

Signature Verification Phase:

1) Assume that a receiver has received $(D_0, D_1, \ldots, D_l, t_w, \sigma_{0,l}^{i,w}, c_w^i, ID_i)$ at time t. In the synchronous mode, the receiver checks whether the timing condition $(t + \delta_t) \le t_w$ holds for $\sigma_{0,l}^{i,w}$. If yes, the receiver buffers the aggregate signature and data set and waits for the end of t_w to obtain tk_w from the TTP. In the asynchronous mode, the receiver 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 receivers for a long time). In this mode, the receiver can buffer the received data and signature as long as they are received before the release of tk_w.

2) When the TTP releases tk_w, each receiver verifies tk_w by checking whether $tk_w \overset{?}{=} H_1(tk_{w-1})$. If tk_w is verified, then the receiver verifies $\sigma_{0,l}^{i,w}$ as follows: The receiver decrypts c_w^i by computing $tk_w^i = H_3(tk_w || ID_i)$ and $k_0^{i,w} = D_{tk_w^i}(c_w^i)$. Using the per-data item key, the receiver computes individual signatures of D_j as $\sigma_j'^{i,w} = MAC_{k_j^{i,w}}(D_j)$ and updates $k_{j+1}^{i,w} = H_1(k_j^{i,w})$ for $j = 0, \ldots, l$. Finally, the receiver computes $\sigma_{0,j}'^{i,w} = H_3(\sigma_{0,j-1}'^{i,w} || \sigma_j'^{i,w})$ for $j = 1, \ldots, l$, where $\sigma_{0,0}'^{i,w} = \sigma_0'^{i,w}$, and checks $\sigma_{0,l}'^{i,w} \overset{?}{=} \sigma_{0,l}^{i,w}$. If they match, the receiver accepts $\sigma_{0,l}^{i,w}$; otherwise, reject.

C. ECC-HaSAFSS

ECC-HaSAFSS addresses the applications where the senders are highly storage constrained, while the receivers can afford certain storage [5]. In this context, ECC-HaSAFSS offers a sender friendly scheme, in which each sender stores *only one key*, while preserving the computational efficiency of Sym-HaSAFSS. To achieve this, ECC-HaSAFSS uses a simple ECC-based approach, which requires only a single ECC scalar multiplication for each time interval t_w. The key idea behind this approach is to give the ability to compute her own session and per-data item keys to each sender after the deployment. Thus, instead of storing a pre-computed encrypted chain root for each time interval like Sym-HaSAFSS, each sender can compute her own encrypted chain roots using her per-interval key (single key storage).

In ECC-HaSAFSS, the TTP generates the initial per-interval key r_0^i for each sender i before the deployment. Each sender i updates the per-interval key at the beginning of each time interval t_w and computes the session key K_w^i using the per-interval key r_w^i with an ECC scalar multiplication. Sender i then randomly generates a per-data item key $k_0^{i,w}$ (i.e., the first per-data item key in t_w). In order to protect $k_0^{i,w}$ in t_w, sender i encrypts it with K_w^i to obtain the encrypted chain root c_w^i. After this stage, sender i computes the aggregate signature using the per-data item $k_0^{i,w}$ following the signature generation step 2 in Sym-HaSAFSS. In order to verify the aggregate signature, a receiver first recovers K_w^i from the public key of sender i using tk_w with an ECC scalar multiplication. The receiver then decrypts the per-data item key of sender i and verifies the aggregate signature following the same steps of the signature generation.

Note that ECC-HaSAFSS preserves the computational efficiency of Sym-HaSAFSS, since the signature generation and verification costs of a single data item in ECC-HaSAFSS are similar to that of Sym-HaSAFSS (only three hash operations). We present the detailed description of ECC-HaSAFSS below:

Initialization and Time Trapdoor Release Phases:

1) The TTP generates the time trapdoor keys tk_w associated with the time period t_w for $w = 0, \ldots, L-1$ following the Sym-HaSAFSS initialization steps. The TTP then generates the public key of each sender i in each time interval t_w as follows: The TTP randomly generates the n-bit per-interval key r_0^i and computes the public key for each t_w as $V_w^i = tk_w(r_w^i - \alpha_w)G$, where $\alpha_w G = H_4(tk_w)$ for $w = 0, \ldots, L-1$. The per-interval key is then updated as $r_{w+1}^i = H_1(r_w^i)$ after each interval t_w. The TTP provides r_0^i to each sender i and gives the public keys V_w^i associated with each sender i for $w = 0, \ldots, L-1$ to all receivers.

2) Time trapdoor release in ECC-HaSAFSS is the same as in Sym-HaSAFSS.

Signature Generation Phase:

1) At the beginning of t_w, sender i randomly generates the n-bit per-data item key $k_0^{i,w}$, and computes the session key as $K_w^i = H_1(r_w^i G)$ and the encrypted chain root as $c_w^i = E_{K_w^i}(k_0^{i,w})$. She updates the per-interval key as $r_{w+1}^i = H_1(r_w^i)$ and deletes (K_w^i, r_w^i) from memory.

2) Sender i computes the aggregate signature $\sigma_{0,l}^{i,w}$ for $(D_0, D_1, \ldots, D_l)$ using the per-data item key $k_0^{i,w}$ and transmits it to the receivers following step 2 in Sym-HaSAFSS signature generation.

Signature Verification Phase: When a receiver receives $(D_0, D_1, \ldots, D_l, \sigma_{0,l}^{i,w}, c_w^i\ \ , t_w, ID_i)$, she first checks timing/request conditions for the received packet and verifies tk_w upon its receipt as in Sym-HaSAFSS signature verification (step 2). The receiver recovers the session key as $K_w^i = H_1(tk_w^{-1} V_w^i + H_4(tk_w))$ and decrypts the per-data item key as $k_0^{i,w} = D_{K_w^i}(c_w^i)$. The receiver verifies $\sigma_{0,l}^{i,w}$ using $k_0^{i,w}$ following step 2 in Sym-HaSAFSS signature verification.

V. Security Analysis

HaSAFSS schemes achieve secure signature aggregation and forward security simultaneously. We follow the example of [5] to analyze our schemes (i.e, authentication, integrity, unforgeability [9], [16] and forward security properties [14]).

First, we prove that HaSAFSS schemes guarantee the confidentiality of per-data item keys and authenticity of the aggregate signature in a given time interval t_w. They achieve this goal by reducing the above security properties to the secrecy and authenticity of tk_w. That is, HaSAFSS schemes compute the encrypted chain root c_w^i using tk_w in such a way that, no entity including sender i can decrypt it before the release of tk_w. Furthermore, the successful verification of the aggregate signature $\sigma_{0,l}^{i,w}$ with $(k_0^{i,w}, tk_w, ID_i)$ directly guarantees the authenticity of $\sigma_{0,l}^{i,w}$. Hence, the receiver can trust $\sigma_{0,l}^{i,w}$, which was generated by sender i using $k_0^{i,w}$ before the release of tk_w. Sym-HaSAFSS and ECC-HaSAFSS follow different strategies to achieve this goal, which have different storage demands on senders and receivers.

Second, we show that HaSAFSS schemes achieve forward security, integrity and unforgeability of the accumulated data items in a given time interval t_w. HaSAFSS schemes achieve these goals by regularly updating per-data item and per-interval keys and obeying the timing/request conditions of the time trapdoor key release modes.

Lemma 1: HaSAFSS schemes guarantee the confidentiality of $k_0^{i,w}$ and authenticity of $\sigma_{0,l}^{i,w}$ in the time duration between the releases of tk_{w-1} and tk_w as long as Assumption 1 and Assumption 2 hold.

Proof: First, in both HaSAFSS schemes, each tk_w can be securely verified by all receivers, since they are elements of a hash chain and are released in the reverse order.

Sym-HaSAFSS: Sym-HaSAFSS reduces the confidentiality of $k_0^{i,w}$ and authenticity of $\sigma_{0,l}^{i,w}$ to the secrecy and authenticity of tk_w, without consulting a PKC primitive. Before the deployment, the TTP derives the session key from the secret time trapdoor key as $tk_w^i = H_3(tk_w || ID_i)$ and encrypts $k_0^{i,w}$ with the session key as $c_w^i = E_{tk_w^i}(k_0^{i,w})$ for $w = 0, \ldots, L-1$.

Assume that the receiver received $(D_0, D_1, \ldots, D_l, \sigma_{0,l}^{i,w}, c_w^i, t_w)$, before the release of tk_w. At this stage, all tk_j for $j = w, \ldots, L-1$ are only known by the TTP. Thus, no entity can decrypt or generate a valid c_w^i without knowing tk_w, including the sender (even compromised by $\mathcal{A}$). After the release of tk_w, the successful verification of $\sigma_{0,l}^{i,w}$ with $(k_0^{i,w}, tk_w)$ guarantees that only sender i who knows $k_0^{i,w}$ before the release of tk_w could have computed this signature. Hence, public verification and authentication of $\sigma_{0,l}^{i,w}$ are achieved.

ECC-HaSAFSS: We analyze ECC-HaSAFSS in two stages. First, we show that ECC-HaSAFSS guarantees the confidentiality of $k_0^{i,w}$ and authenticity of $\sigma_{0,l}^{i,w}$ based on the secrecy and authenticity of tk_w. Second, we show that ECC-HaSAFSS is key compromise resistant. That is, $\mathcal{A}$ compromising a sender and a receiver cannot use the compromised keys to extract the time trapdoor keys of the TTP or the secret key of a non-compromised sender. This property guarantees the secrecy of tk_w before its release.

(1) Confidentiality of $k_0^{i,w}$ and authenticity of $\sigma_{0,l}^{i,w}$: In ECC-HaSAFSS, each sender i computes her own session key using her per-interval key as $K_w^i = H_1(r_w^i G)$, with which the

encrypted chain root is computed for each t_w. In order to make K_w^i recoverable for the receiver after the release of tk_w, the TTP computes a set of public key for each sender i. The TTP embeds session keys into the public keys V_w^i by blinding them with tk_w as $V_w^i = tk_w(r_w^i - \alpha_w)G$ where $\alpha_w G = H_4(tk_w)$ for $w = 0, \ldots, L-1$. The algebraic structure of the public keys guarantees that the session key K_w^i cannot be recovered from V_w^i without knowing tk_w. That is, $r_w^i G$ cannot be isolated from V_w^i as $K_w^i = tk_w^{-1} V_w^i + H_4(tk_w)$ without knowing tk_w.

Note that sender i immediately deleted (K_w^i, r_w^i) from the memory after computing $c_w^i = E_{K_w^i}(k_0^{i,w})$ at the beginning of t_w. Hence, before the release of tk_w, no entity can decrypt or generate a valid c_w^i without knowing tk_w. Similarly, only sender i could compute a valid c_w^i, which can be correctly decrypted by using tk_w and V_w^i. Thus, correct verification of the aggregate signature guarantees its authenticity.

(2) Key compromise resistance: Assume that $\mathcal{A}$ compromises a sender (sender i) and a receiver in time interval t_w. $\mathcal{A}$ extracts r_{w+1}^i from sender i and V_j^i for $j = 0, \ldots, L-1$ from the receiver. $\mathcal{A}$ has also accumulated pre-released tk_j for $j = 0, \ldots, w-1$ up to now. The objective of $\mathcal{A}$ is to recover the future secret time trapdoor keys of the TTP or the secret per-interval key of a non-compromised sender using the compromised keys as follows:

(a) $\mathcal{A}$ attempts to use the per-interval key r_{w+1}^i of sender i and the public keys V_j^i for $j = w+1, \ldots, L-1$ to extract any future time trapdoor key tk_j for $j = w+1, \ldots, L-1$. However, recovering such a tk_j is as difficult as solving the ECDLP problem, since it requires to compute $x_j = tk_j(r_j^i - \alpha_j)$ mod q from V_j^i. Note that even if such a x_j would be found, both (tk_j, α_j) are unknown to $\mathcal{A}$. Thus, she cannot isolate tk_j from x_j for $j = w+1, \ldots, L-1$.

(b) $\mathcal{A}$ attempts to use the pre-released tk_j and the public keys V_j^m, $j = 0, \ldots, w-1$, to recover the secret per-interval key r_j^m of a non-compromised sender m for any $0 \leq j \leq w-1$. ($\mathcal{A}$ can then easily compute the future per-interval keys of sender m using the recovered r_j^m, since they are elements of a hash chain.) However, recovering such a r_j^m as r_j^m mod q from $r_j^m G$ is as difficult as solving the ECDLP problem where $r_j^m G = tk_j^{-1} V_j^m + H_4(tk_j)$ for $j = 0, \ldots, w-1$. ■

Theorem 1: HaSAFSS schemes achieve forward security, integrity and unforgeability of $(D_0, D_1, \ldots, D_l, \sigma_{0,l}^{i,w})$ in the time duration between the releases of tk_{w-1} and tk_w as long as Lemma 1 holds.

Proof: At the beginning of each t_w, signature generation step 1 in both Sym-HaSAFSS and ECC-HaSAFSS guarantees that per-interval keys are evolved (by deleting the previous one) and per-data item keys are derived. After the initial updates, both HaSAFSS schemes regularly update their per-data item keys for each accumulated data item (Sym-HaSAFSS signature generation step 2). Hence, if $\mathcal{A}$ breaks-in in the duration of t_w, she always faces the updated per-interval and per-data item keys, which guarantee the forward security of the data accumulated in the time duration between the releases of tk_{w-1} and tk_w.

Both HaSAFSS schemes prevent $\mathcal{A}$ from forging the data accumulated in the previous time periods by obeying the timing/request rules of the time trapdoor release mode. The main condition for the receiver to accept $(D_0, D_1, \ldots, D_l, \sigma_{0,l}^{i,w})$ is that it should be received before the release of tk_w. After the release of tk_w, the timing condition of the synchronous mode and the request condition of the asynchronous mode prevent receivers from accepting any signature associated with tk_w. Thus, if $\mathcal{A}$ breaks-in after the release of tk_w, she cannot use any key associated with $t \leq t_w$, thereby cannot forge any data item accumulated in these time periods.

HaSAFSS schemes achieve unforgeability and integrity in a given t_w by using per-data item keys to individually compute MAC of each accumulated data item and fold them into the previous aggregate signature by hashing (H_3) them as shown in Sym-HaSAFSS signature generation step 2. Lemma 1 guarantees confidentiality of $k_0^{i,w}$ and authenticity of $\sigma_{0,l}^{i,w}$, while the forward security is achieved in the time duration between the releases of tk_{w-1} and tk_w as shown above. Hence, $\sigma_{0,l}^{i,w}$ computed with $k_0^{i,w}$ is unforgeable and guarantees the integrity of $(D_0, D_1, \ldots, D_l)$ in a given time interval. ■

VI. Performance Analysis

In this section, we present the performance analysis of HaSAFSS schemes. We analyze and compare HaSAFSS

TABLE I
NOTATION USED IN THE PERFORMANCE COMPARISON OF HASAFSS AND FSSAGG SCHEMES

Exp: Modular exponentiation with modulus p, where $\|p\| = 512$ bit		L: # of time periods
$ECCMul$: ECC scalar multiplication over F_p	S: # of senders	w: Current time period
Mul: Scalar multiplication with modulus p	ℓ: #of data items	PR: ECC pairing operation
MtP: Map-to-point operation	H: Hash operation	R: # of receivers

TABLE II
COMPARISON OF HASAFSS SCHEMES TO FSSAGG SCHEMES

Criteria vs Scheme		**Sym-HaSAFSS**	**ECC-HaSAFSS**	**FssAgg MAC** [5]	**FssAgg BLS** [5] & [9]
Computational Overhead	Sender	$(3H)\ell$	$ECCMul + (3H)\ell$	$(3H)\ell$	$(MtP + Exp + Mul + H)\ell$
	Receiver	$(3H)\ell$	$ECCMul + (3H)\ell$	$(3H)\ell$	$(Mul + PR)\ell$
Storage Overhead	Sender	$\|H\|(L-w)$	$\|H\|$	$\|H\|R$	$\|H\|$
	Receiver	$\|H\|$	$\|q\|(L-w)S$	$\|H\|S$	$\|q\|(L \cdot S)$
Signature Size		$\|H\|$	$\|H\|$	$\|H\|$	$\|p\|$
Key Size		n	n	n	$\|q\|$
Public Verifiability		Y	Y	N	Y
Immediate Verification		N	N	Y	Y

schemes to previous schemes in terms of six essential criteria for forward secure and aggregate signatures: computational and storage overhead of senders and receivers, respectively, the size of aggregate signature, and the size of signing key. We present our analysis for these criteria under two main subsections: computational overhead and storage overhead.

We first give notation in Table I and then provide Table II. In Table II, we compare HaSAFSS schemes with FssAgg schemes (the best known alternatives). Note that HaSAFSS schemes are always computationally more efficient than the existing aggregate signature schemes that utilize pairing operations (e.g., [9], [16]). Thus, by specifically comparing HaSAFSS schemes with FssAgg-BLS, we can see their difference from this general class of schemes.

A. Computational Overhead

We first analyze the signature generation cost of HaSAFSS schemes, using the notation introduced in Table I. In both HaSAFSS schemes, at the beginning of each t_w, each sender computes the per-data item and per-interval keys with $2H$ computational cost (i.e., two hash operations). This introduces negligible cost, since this operation is performed once for each t_w. In ECC-HaSAFSS, in addition to $2H$, a single $ECCMul$ is also performed for each t_w. After initialization, the signature generation cost of the sender for a single data item is $3H$ in both Sym-HaSAFSS and ECC-HaSAFSS. Thus, the cost of the signature generation for ℓ data items in t_w is $(3H)\ell$ in Sym-HaSAFSS, and $ECCMul + (3H)\ell$ in ECC-HaSAFSS. The analysis of the signature verification cost is similar to the signature generation. Note that the cost of a single H (verification of tk) and E/D operation (encrypt/decrypt $k_0^{i,w}$) is negligible, since they are executed only once for each t_w. Thus, signature verification costs of Sym-HaSAFSS and ECC-HaSAFSS are $(3H)\ell$ and $ECCMul + (3H)\ell$, respectively.

TABLE III
EXECUTION TIME OF THE BASIC OPERATIONS IN COMPARED SCHEMES (IN MS)

Sign/Verify cost of a data item (Executed ℓ times)		
FssAgg-MAC [5] HaSAFSS schemes	Sign/Verify	0.06
FssAgg-BLS [5]	Sign	7.69
	Verify	53.52

Initialization cost for each t_w (Executed only once)		
Sym-HaSAFSS	Sign/Verify	0.08
ECC-HaSAFSS	Sign/Verify	1.58

Comparison: We first compare HaSAFSS schemes to FssAgg-MAC [5]. FssAgg-MAC and Sym-HaSAFSS have similar computational cost as $(3H)\ell$. ECC-HaSAFSS requires one extra $ECCMul$ for each t_w as $ECCMul + (3H)\ell$, which is slightly more costly than FssAgg-MAC. Note that the essential superiority of HaSAFSS schemes over FssAgg-MAC is their "public verifiability", whose benefits can be observed in the storage overhead analysis.

Second, we compare HaSAFSS schemes with their best known PKC-based counterpart scheme, FssAgg-BLS [5]. Based on PKC primitives, FssAgg-BLS has all advantages of the public verifiability. However, it incurs very high computational overhead due to the heavy use of PKC operations. FssAgg-BLS signature generation is costly due to Exp, while its signature verification is extremely costly due to PR. Table III shows the execution time difference between signature generation and verification costs of HaSAFSS and FssAgg-BLS. Signature generation and verification costs of a single data item in FssAgg-BLS are 7.69 ms and 53.52 ms [5], while they are both 0.06 ms in HaSAFSS schemes. This prohibitive computational cost of FssAgg-BLS makes it impractical for the envisioned UWSN applications. Note that these expensive operations are executed for each data item to be signed/verified[2], which increases the execution time difference between HaSAFSS and FssAgg-BLS schemes as shown in Figure 3.

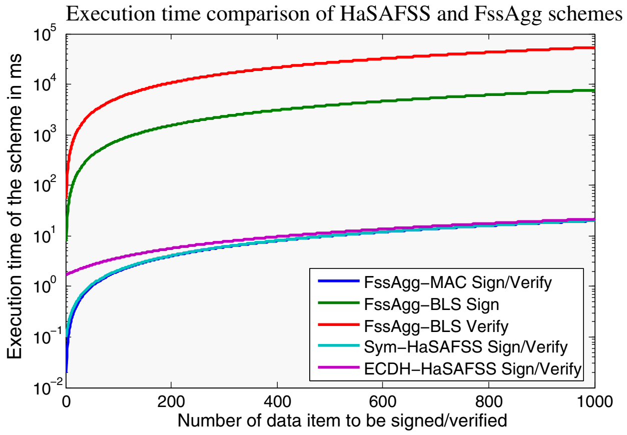


Fig. 3. Execution time comparison of HaSAFSS and FssAgg [5]

We see that both HaSAFSS schemes are much more efficient than FssAgg-BLS. Furthermore, in HaSAFSS schemes, both senders and receivers equally get benefits of this computational efficiency, while the classical aggregate signature schemes incur more computational cost to the receivers.

B. Storage Overhead

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

In Sym-HaSAFSS, each sender initially stores L encrypted chain roots. As the time goes from one period into the next, the sender deletes the encrypted chain root associated with the previous time period from her memory. Thus, each sender stores $(L-w)$ keys in t_w. However, each receiver *only stores a single key* (negligible $|H|$ overhead, e.g., 160 bit). In ECC-HaSAFSS, each sender stores *only one key*, since she can compute her own session keys after the deployment. In order to recover session keys, each receiver stores L public keys for each sender initially, and as the time goes from one period into the next, she deletes the public keys associated with the previous time period from her memory. Thus, storage overhead of each receiver is $|q|(L-w)S$ in t_w (e.g., $|q| = 160$ bits).

Comparison: Sym-HaSAFSS is the most storage efficient scheme among all the compared schemes from the *receiver's*

[2] In our computational cost analysis, we analyze FssAgg-BLS parallel to HaSAFSS schemes by updating the signing key for each data item.

perspective. Since Sym-HaSAFSS requires storing only single key, it incurs small $|H|$ bit overhead for each receiver. However, FssAgg-BLS and FssAgg-MAC incur $|q|(L \cdot S)$ and $|H|S$ bit overheads, respectively. This advantage of Sym-HaSAFSS is also valid with respect to other aggregate signature schemes that incur higher storage overhead on the receiver side. Sym-HaSAFSS is also efficient in the sender perspective by requiring only linear storage overhead to each sender. Thus, Sym-HaSAFSS is especially useful for the applications in which receivers are also storage constrained, while FssAgg schemes cannot address this type of applications.

In contrast to Sym-HaSAFSS, ECC-HaSAFSS obeys the traditional resourceful receiver assumption to address such UWSN applications (e.g., high-end mobile receivers [5]). Since the storage overhead of ECC-HaSAFSS reduces as the time advances (i.e., $|q|(L-w)S$ bit overhead in t_w), it is more efficient than FssAgg-BLS whose overhead is always constant as $|q|(L \cdot S)$ bit. When compared to FssAgg-MAC, ECC-HaSAFSS is more efficient for the sender side (i.e., $|H|$ vs. $|H|R$ bits), and FssAgg-MAC is more efficient for the receiver side (i.e., $|H|(L-w)S$ vs. $|H|S$ bits).

Remark 4: Despite all the advantages, introducing asymmetry between the senders and receivers using the time factor brings a natural complication: HaSAFSS schemes cannot provide immediate verification on the receiver side. In order to verify a received signature, the receiver needs to wait for the release of tk_w 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.

VII. Conclusion

In this paper, we proposed a new class of digital signature schemes, Hash-Based Sequential Aggregate and Forward Secure Signature (HaSAFSS), which is especially suitable for the UWSN applications. HaSAFSS schemes achieve near-optimal computational efficiency, low storage overhead, public verifiability, signature aggregation and forward security simultaneously. HaSAFSS schemes achieve these goals by using the already existing verification delays in the envisioned UWSN applications via two realistic data/time trapdoor delivery models.

We proposed two specific HaSAFSS schemes, Sym-HaSAFSS and ECC-HaSAFSS, in this paper. Sym-HaSAFSS completely relies on symmetric encryption and cryptographic hash functions. Thus, Sym-HaSAFSS is extremely efficient in terms of computational cost, when compared with the existing PKC-based aggregate signature schemes. Furthermore, Sym-HaSAFSS is also a storage efficient scheme, which requires only one key stored by each receiver, while storage requirement of each sender is linear with the total number of time periods. Our second scheme ECC-HaSAFSS preserves the computational efficiency of Sym-HaSAFSS for signature generation and verification, but offers a different storage alternative. In ECC-HaSAFSS, each sender stores only one key, while each receiver stores a set of public keys for each sender. Thus, Sym-HaSAFSS and ECC-HaSAFSS complement each other in terms of storage overhead.

In our future work, we will consider different cryptographic mechanisms that allow us to further reduce the storage requirement in HaSAFSS schemes.

References

[1] D. Pietro, R., L. Mancini, C. Soriente, A. Spognardi, and G. Tsudik, "Catch me (if you can): Data survival in unattended sensor networks," *Pervasive Computing and Communications, 2008. PerCom 2008. Sixth Annual IEEE International Conference on*, pp. 185–194, March 2008.

[2] D. Ma and G. Tsudik, "Dish: Distributed self-healing," in *SSS '08: Proceedings of the 10th International Symposium on Stabilization, Safety, and Security of Distributed Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 47–62.

[3] R. Di Pietro, D. Ma, C. Soriente, and G. Tsudik, "Posh: Proactive co-operative self-healing in unattended wireless sensor networks," *Reliable Distributed Systems, 2008. SRDS '08. IEEE Symposium on*, pp. 185–194, Oct. 2008.

[4] J. C. McEachen and J. Casias, "Performance of a wireless unattended sensor network in a freshwater environment," in *HICSS '08: Proceedings of the 41st Annual Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE, 2008, p. 496.

[5] D. Ma and G. Tsudik, "Forward-secure sequential aggregate authentication," in *Security and Privacy, SP '07. IEEE Symposium on Security and Privacy*, 20-23 May 2007, pp. 86–91.

[6] Trident Systems, "Trident's family of unattended ground sensors," http://www.tridsys.com/white-unattended-ground-sensors.htm.

[7] Information Processing Technology Office (IPTO) Defense Advanced Research Projects Agency (DARPA), "Bba 07-46 landroids broad agency announcement, 2007," http://www.darpa.mil/ipto/solicit/baa/BAA-07-46_PIP.pdf.

[8] M. Bellare and S. Miner, "A forward-secure digital signature scheme," in *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology, CRYPTO '99*.

[9] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proceedings of Advances in Cryptology (EUROCRPYT 2003)*, 2004, pp. 416–432.

[10] R. Rivest, A. Shamir, and D. Wagner, "Time-lock puzzles and timed-release crypto," Cambridge, MA, USA, Tech. Rep., 1996.

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

[12] R. Anderson, "Invited lecture," 4th ACM Computer and Communications Security, 1997.

[13] B. Libert, J. Quisquater, and M. Yung, "Forward-secure signatures in untrusted update environments: efficient and generic constructions," in *Proceedings of the 14th ACM conference on Computer and communications security, CCS '07*. NY, USA: ACM, pp. 266–275.

[14] H. Krawczyk, "Simple forward-secure signatures from any signature scheme," in *Proceedings of the 7th ACM conference on Computer and Communications Security, CCS '00*. NY, USA: ACM, pp. 108–115.

[15] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," *Journal of Cryptology*, vol. 14, no. 4, pp. 297–319, 2004.

[16] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters, "Sequential aggregate signatures and multisignatures without random oracles," in *Advances in Cryptology, EUROCRYPT '06*.

[17] A. Boldyreva, C. Gentry, A. O'Neill, and D. Yum, "Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing," in *Proceedings of the 14th ACM conference on Computer and communications security, CCS '07*. New York, NY, USA: ACM, 2007, pp. 276–285.

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

[19] H. Varsakelis, K. Chalkias, and G. Stephanides, "Low-cost anonymous timed-release encryption," *Information Assurance and Security, IAS 2007*, pp. 77–82, 29-31 Aug. 2007.

[20] D. Boneh, "The decision diffie-hellman problem," in *Proceedings of the Third Algorithmic Number Theory Symposium, LNCS*, 1998, pp. 48–63.

[21] D. Stinson, *Cryptography: Theory and Practice,Second Edition*. CRC/C&H, 2002.

[22] A. A. Yavuz, F. Alagöz, and E. Anarim, "Himutsis: Hierarchical multi-tier adaptive ad-hoc network security protocol based on signcryption type key exchange schemes," in *ISCIS*, 2006, pp. 434–444.