Forward-Secure Sequential Aggregate Authentication

Di Ma, Gene Tsudik

University of California, Irvine
{dma1,gts}@ics.uci.edu

Abstract. Wireless sensors are employed in a wide range of applications. One common feature of most sensor settings is the need to communicate sensed data to some collection point or sink. This communication can be direct (to a mobile collector) or indirect – via other sensors towards a remote sink. In either case, a sensor might not be able to communicate to a sink at will. Instead it collects data and waits (for a potentially long time) for a signal to upload accumulated data directly.
In a hostile setting, a sensor may be compromised and its post-compromise data can be manipulated. One important issue is *forward security* – how to ensure that pre-compromise data cannot be manipulated? Since a typical sensor is limited in storage and communication facilities, another issue is how to minimize resource consumption due to accumulated data. It turns out that current techniques are insufficient to address both challenges. To this end, we explore the notion of *Forward-Secure Sequential Aggregate* (*FssAgg*) authentication Schemes. We consider *FssAgg* authentication schemes in the contexts of both conventional and public key cryptography and construct a *FssAgg* MAC scheme and a *FssAgg* signature scheme, each suitable under different assumptions. This work represents the initial investigation of Forward-Secure Aggregation and, although the proposed schemes are not optimal, it opens a new direction for follow-on research.
KEYWORDS: sensors, signature schemes, authentication schemes, key compromise, forward security, aggregate signatures.

1 Introduction

Wireless sensors can enable large-scale data collection in many different settings, scenarios and applications. Examples abound in all kinds of tracking and monitoring applications in both civilian and military domains. A Wireless Sensor Network (WSN) might contain hundreds or thousands of low-cost sensors and one or more sinks or data collectors. Individual sensors obtain measurements from the environment and (periodically or upon request) forward the accumulated data to the sink. A sink might be a gateway to another network, a powerful data processing or storage center, or an access point for human interface. (Some WSNs support user-driven data queries and commands through the sink.)

In this paper, we are motivated by two types of envisaged sensor scenarios:

A Sensors do not communicate with each other, i.e., there is *no sensor network* as such. Instead, a mobile device that we call a *collector*.[1] A collector might not be fully trusted; it might be nothing more than an intermediary between sensors and an off-line (trusted) sink.

B Sensors communicate but they do not actually "network", i.e., communication is restricted to mere forwarding of information from other sensors towards a sink or sinks. In this context, a sink is a fully trusted entity.

In either case, a sensor might not be able to communicate to a sink at will. Instead, it collects data and waits (potentially, for a long while) either for a signal – or some pre-determined time – to upload accumulated data to a collector or a sink. Put another way, there is no real-time reporting of sensed information between sensors and a collector or a sink.

Data integrity and (sensor) authentication are essential security services required in most sensor applications [19] since sensors are often used in unattended and adversarial environments. They interact closely

[1] We use the terms "collector" and "sink" to distinguish between entities that gather data in the two scenarios.

with the physical environment and with people, thus being subject to a wide range of security risks. An attacker may inject its own data as well as modify and delete data produced by sensors. As a result, sensor data must be authenticated before being processed and used for whatever purposes. Particularly in critical settings (e.g., radiation, seismic or intrusion monitoring) strong data integrity and authenticity guarantees are needed. Standard textbook techniques, such as MACs (Message Authentication Codes) or digital signatures, can be used in applications where data integrity/authenticity is required. However, several obstacles hinder straight-forward usage of these standard techniques.

One important issue is the threat of **sensor compromise** and the consequent exposure of secret keys used for MACs or signatures.[2] Key exposure makes it easy for the adversary to produce fraudulent data ostensibly sensed after the compromise. Moreover, it also allows the adversary to produce fraudulent data *before the compromise*, assuming it has not been reported to a sink or a collector. This is clearly undesirable. Fortunately, there are so-called *forward-secure* cryptographic techniques that allow the signer (sensor, in our case) to periodically evolve its secret key such that compromise of a current secret key cannot lead to compromise of secret key(s) used in past periods. It is therefore possible to mitigate the effects of sensor compromise by using a sense-and-sign approach. In other words, a sensor does not wait to sign (or MAC) ALL sensed data until it has to send it, since doing that would open **all collected data** to attack. Instead, it signs data as soon as it is sensed and evolves the signing key.

Another important issue is **storage and communication overheads**. Clearly, on-board storage is a limited commodity in most sensor settings and it is natural to minimize its size and consumption. In both scenarios A and B outlined above, a sensor gradually accumulates data (readings, measurements), stores it locally and – at some later time – sends it to a sink. We are not concerned in minimizing storage consumed by the actual data; that is an interesting topic in its own right. Instead, we are interested in minimizing storage due to authentication tags (i.e., MACs or signatures) since they represent pure overhead. If key compromise and forward security were not an issue, minimizing storage overhead would be trivial – a sensor simply signs or MACs all accumulated data once, before forwarding it to the sink. At the same time, forward security forces us to compute authentication tags per sensed unit of data, which we refer to as a *message* from now on.[3] Therefore, a sensor accumulates as many authentication tags as messages while it waits for a time or a signal to off-load the data. This is problematic since even the size of a MAC (and certainly of a signature) can easily exceed the size of actual data, i.e., messages. At the minimum, each 128 bits per MAC or 160 bits per signature would need to be allocated.

Communication overhead is a related, though perhaps not as critical, matter. In scenario A, a sensor uploads accumulated messages directly to the collector. Thus, the communication overhead due to sending multiple authentication tags is less problematic than in Scenario B where the same overhead affects all sensors that forward information from other sensors towards the sink. (We refer to the oft-cited folklore in [3] which claims that wireless transmission of a single bit can consume over $1,000$ times of the energy of a single 32-bit computation.)

Reconciling the need to minimize storage (and communication) overhead with the need to mitigate potential key compromise (i.e., obtain forward security) is precisely the topic of this paper.

Contributions: We explore Forward Secure Sequential Aggregate (*FssAgg*) authentication schemes that simultaneously mitigate the threat of key compromise and achieve optimal storage and communication efficiency. An *FssAgg* scheme allows a signer to combine multiple authentication tags generated in different key/time periods into a single constant-size tag. Compromise of the current key does not allow the attacker

[2] Building an inexpensive tamper-proof, or even tamper-resistant, sensor is a much greater challenge.

[3] Note that the duration of the key evolvement period in a forward-secure scheme does not have to match the time between successive sensor readings; however, to simplify the discussion, we assume that it does.

to forge any aggregate authentication tag containing elements pre-dating the compromise. Any insertion of new messages, modification and deletion (including truncation) of existing messages makes the aggregate tag demonstrably invalid. We consider this topic in both conventional and public key cryptographic settings and construct two practical schemes: an *FssAgg* MAC scheme as well as an *FssAgg* signature scheme.

Applications: Despite the fact that the main motivation comes from certain sensor scenarios, we note that forward-secure sequential aggregation is also germane to more traditional applications, such as secure logging systems [5, 20]. In a secure logging system, the signer (log server) does not suffer from storage or bandwidth limitations. However, resistance to so-called *truncation attacks* is important. (Briefly, a truncation attack occurs when the adversary succeeds in deleting a contiguous subset of tail-end messages.) Schemes such as [5] and [20] are vulnerable to truncation attacks. A secure *FssAgg* signature scheme is well-suited for secure logging applications – it resists truncation attacks due to all-or-nothing (aggregate and forward-secure) signature verification.

Organization: After a brief overview of related work in Section 2, we introduce the model and security requirements in Section 3. Next, we present an *FssAgg* MAC scheme in Section 4 and an *FssAgg* signature scheme in 5. Section 6 concludes the main body of the paper. Appendix A presents a brief performance evaluation of the *FssAgg* signature scheme, followed by appendices B and C that contain, respectively, the security model and a proof sketch for the same scheme.

2 Related Work

NOTE: this section is kept brief due to dire space limitations.

The topic of this paper is quite distinct from data aggregation in sensor networks [8, 11, 12, 21, 22]. In an *FssAgg* authentication scheme, authentication objects are aggregate while data records (messages) are kept intact. In a data aggregation scheme, individual data information is lost and the aggregate value is used to provide or derive statistical information, such as mean, median or max/min. Data aggregation schemes are very useful, but unsuitable for applications, where the availability of individual sensed data records is required (e.g., temperature pattern sensing in a nuclear reactor).

The notion of forward security was introduced in the context of key-exchange protocols [10] and lagter adapted to signature schemes. Forward-secure signatures were first proposed by Anderson in [2] and subsequently formalized by Bellare and Miner in [4]. The main challenge is efficiency: an ideal scheme must have constant (public and secret) key size, constant signature size as well as constant signing, verification, and (public and secret) key update operations. Several schemes proposed in the literature satisfy some or most of these requirements [1, 4, 13–15]. Also, in [5], Bellare and Yee examine forward security in the context of conventional cryptography.

Several aggregate signature schemes have been proposed in the literature, starting with the initial seminal result by Boneh, et al. [6, 16, 17]. An aggregate signature scheme combines k signatures generated by n signers ($k \geq n$) into a single and compact aggregate signature that, if verified, simultaneously verifies every component signature. Interestingly, our goal is to aggregate signatures by the *same* signer (e.g., a sensor), however, these signatures are computed in different periods, and with different keys. Thus, our goals impose no additional restrictions on existing definitions of aggregate signatures. Also, our envisaged schemes do not require simultaneous aggregaqtion of multiple signatures as in [6]; instead, we need sequential (incremental) aggregation as in [17] or [16].

3 Definitions and Properties

In this section we present some informal definitions and properties.[4] An *FssAgg* signature scheme is composed of the following algorithms. They are quite similar to those in sequential aggregated signature schemes, notably, the recent scheme of Lu, et al. [16].

The key generation algorithm *FssAgg.Kg* is used to generate public/private key-pairs. Unlike the one used in [16], it also takes as input T – the maximum number of time periods (key evolvements).

The sign-and-aggregate algorithm *FssAgg.Asig* takes as input a private key, a message to be signed and a signature-so-far (an aggregated signature computed up to this point). It computes a new signature on the input message and combines it with the signature-so-far to produce a new aggregated signature. As the final step of *FssAgg.Asig*, it runs a key update subroutine *FssAgg.Upd* which takes as input the signing key for the current period and returns the new signing key for the next period (not exceeding T.) We make key update part of the sign-and-aggregate algorithm in order to obtain stronger security guarantees (see below).

The verify algorithm *FssAgg.Aver*, on input of a putative aggregate signature, a set of presumably signed distinct messages and a public key, outputs whether the aggregate is valid. (The distinction from non-forward-secure schemes is that we use a single public key, as there is only one signer.)

The key update algorithm *FssAgg.Upd* takes as input the signing key for the current period and returns the new signing key for the next period (provided that the current period does not exceed $T - 1$.)

A secure *FssAgg* scheme must satisfy the following properties:

1. *Correctness:* Any aggregated signature produced with $\mathbf{FssAgg}.\mathbf{Asig}$ must be accepted by $\mathbf{FssAgg}.\mathbf{Aver}$.
2. *Unforgeability:* Without the knowledge of any signing keys (for any period), no adversary can compute an aggregate signature on any message or set of messages.
3. *Forward-security:* No adversary who compromises the signer's i-th signing key can generate a valid aggregate signature containing a signed message – for any period $j < i$ – except the aggregate-so-far signature generated by the signer before the compromise, i.e., the aggregated signature the adversary finds upon compromise.

Note that the last property subsumes security against truncation or deletion attacks. An adversary who compromises a signer has two choices: either it includes the intact aggregate-so-far signature in future aggregated signatures, or it ignores the aggregate-so-far signature completely and start a brand new aggregated signature. What it cannot do is selectively delete components of an already-generated aggregate signature.

4 A Forward-Secure Sequential Aggregate MAC Scheme

We now present a trivial *FssAgg* MAC scheme. It can be used to authenticate multiple messages when public (transferrable) verification is not required. As such, it is well-suited for scenario B in Section 1 where a sensor communicates (via other sensors) to the sink. We first present the scheme and then show how to apply it to the envisaged sensor environment.

The scheme uses the following cryptographic primitives:

- $\mathcal{H}$: a collision resistant one-way hash function with domain restricted to k-bit strings: $\mathcal{H} : \{0,1\}^k \rightarrow \{0,1\}^k$.
- $\mathcal{H}_a$: a collision resistent one-way hash function with arbitrary length input: $\mathcal{H}_a : \{0,1\}^* \rightarrow \{0,1\}^k$.
- h: a secure MAC scheme $h : \{0,1\}^k \times \{0,1\}^* \rightarrow \{0,1\}^t$ that, on input of a k-bit key x and an arbitrary message m outputs a t-bit MAC $h_x(m)$.

[4] Our presentation is informal to conserve very limited space.

FssAgg.Kg. Any symmetric key generation algorithm can be used to generate an initial k-bit secret key s. We set $sk_0 = vk = s$.

FssAgg.Asig. At time period i, the signer is given a message M_i to be signed and an aggregate-so-far MAC $\sigma_{1,i-1}$ on messages $M_1, \cdots, M_{i-1}$. The signer first generates a MAC σ_i on M_i with h using sk_i: $\sigma_i = h_{sk_i}(M_i)$. It then computes $\sigma_{1,i}$ by folding σ_i onto $\sigma_{1,i-1}$ through $\mathcal{H}_a$: $\sigma_{1,i} = \mathcal{H}_a(\sigma_{1,i-1}||\sigma_i)$. $\mathcal{H}_a$ acts as the aggregation function. Alternatively we can compute $\sigma_{1,i}$ as follows:[5]

$$\sigma_{1,i} = \mathcal{H}_a(\mathcal{H}_a(\cdots\mathcal{H}_a(\mathcal{H}_a(\sigma_1||\sigma_2)||\sigma_3))||\cdots)||\sigma_i)\text{where}\sigma_j = h_{sk_j}(M_j)\forall j = 1, \cdots, i \quad (1)$$

Finally, the signer executes the key update subroutine defined as:

FssAgg.Upd. We define the i-th signing key sk_i as the image under $\mathcal{H}$ of the previous key sk_{i-1}: $sk_i = \mathcal{H}(sk_{i-1})$, $i > 0$. (This part is the same as the forward-secure MAC scheme in [19].)

FssAgg.Aver. To verify a candidate $\sigma_{1,i}$ over messages $M_1, \cdots, M_i$, the verifier (who has the verifying key vk which is the same as the initial signing key sk_0) computes keys $sk_1, \cdots, sk_i$ through the public key update function. It then mimics the signing process and re-computes $\sigma^c_{1,i}$ and compares it with $\sigma_{1,i}$. If the two values match, it outputs valid. Otherwise it outputs invalid.

5 A Forward-Secure Sequential Aggregate Signature Scheme

If public (transferrable) verification is required we need a *FssAgg* signature scheme to check the authenticity of data records. Trivially, all aggregate signature schemes [6, 16, 17] can be used as a *FssAgg* signature scheme if we treat the key of signer i as the key used (by the same signer) in the time period i. However a trivial construction is useless for our purposes since a signer (e.g., a sensor) would need $O(T)$ storage to store its secret keys.

The overall efficiency of a *FssAgg* signature scheme depends on the following metrics: 1) size of the aggregate signature; 2) size of the signing key; 3) complexity of key update; 4) complexity of aggregate signing; 5) size of verification key; 6) complexity of aggregate verifying. The first four represent *signer efficiency* and the last two represent *verifier efficiency*; the size parameters (aggregate signature, signing key and verification key) represent *space efficiency* and the complexity parameters (sign, verify and key update) represent *time efficiency*. In our envisaged sensor scenarios, signer efficiency is much more important than verifier efficiency and space efficiency more important than time efficiency.

Focusing on the signer and space efficiency, we propose a *FssAgg* signature scheme based on the BLS signature scheme [6]. BLS signatures can be aggregated through EC multiplication by anyone [7]. We first introduce the BLS scheme and then show how to modify it to be a *FssAgg* signature scheme.

The BLS scheme works in groups with bilinear maps. A bilinear map is a map $e : G_1 \times G_2 \rightarrow G_T$, where: (a) G_1 and G_2 are two (multiplicative) cyclic groups of prime order q; (b) $|G_1| = |G_2| = |G_T|$; (c) g_1 is a generator of G_1 and g_2 is a generator of G_2. The bilinear map $e : G_1 \times G_2 \rightarrow G_T$ satisfies the following properties:

1. Bilinear: for all $x \in G_1$, $y \in G_2$ and $a, b \in \mathbb{Z}$, $e(x^a, y^b) = e(x, y)^{ab}$;

[5] Note that hash functions are generally designed as an iterative process [18]. That is, a hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^k$ with arbitrarily long finite input is executed by iteratively invoking an internal (per block) function $f : \{0,1\}^{r+k} \rightarrow \{0,1\}^k$ ($r > k$ as a hash function compresses its input) with fixed-size input. A hash input x of arbitrary finite length is divided into fixed-length r-bit blocks x_i. In each iteration, f takes on the current input block x_i and the intermediate result H_{i-1} produced by f in the previous iteration. We can thus modify the aggregation function as follows: form an input block with several MACs and then fold the block into the aggregate in one round. This way, $\sigma_{1,i}$ can be represented as: $\sigma_{1,i} = \mathcal{H}_a(\sigma_1||\sigma_2||\cdots||\sigma_i)$. Compared with 1, this aggregation function in is more efficient.

2. Non-degenerate: $e(g_1, g_2) \neq 1$

The BLS scheme uses a full-domain hash function $\mathcal{H}_1(\cdot)$: $\{0,1\}^* \rightarrow G_1$. Key generation involves picking a random $x \in \mathbb{Z}_q$ for each signer, and computing $v = g_2^x$. The signer's public key is $v \in G_2$ and her secret key is x. Signing a message M involves computing the message hash $h = \mathcal{H}_1(M)$ and then the signature $\sigma = h^x$. To verify a signature one computes $h = \mathcal{H}_1(M)$ and checks that $e(\sigma, g_2) = e(h, v)$. The verification costs amount to 2 bilinear mappings.

To aggregate n BLS signatures, one computes the product of individual signatures as follows:

$$\sigma_{1,n} = \prod_{i=1}^{n} \sigma_i \tag{2}$$

where σ_i corresponds to the signature on message M_i. The aggregate signature $\sigma_{1,n}$ is of the same size as an individual BLS signature and aggregation can be performed incrementally and by anyone.

Verification of an aggregate BLS signature $\sigma_{1,n}$ includes computing the product of all message hashes and verifying the following match:

$$e(\sigma_{1,n}) \stackrel{?}{=} \prod_{i=1}^{n} e(h_i, v_i) \tag{3}$$

where v_i is the public key of the signer who generates σ_i on message M_i.

FssAgg.Kg The signer picks a random $x_0 \in \mathbb{Z}_p$ and computes a pair (x_i, v_i) $(i = 1, \cdots, T)$ as: $x_i = \mathcal{H}(x_{i-1})$, $v_i = g_2^{x_i}$. The initial signing key is x_0 and the public key is: $(v_1, \cdots, v_T) = (g_2^{x1}, \cdots, g_2^{xT})$. Note that, in our sensor scenarios, a sensor (signer) would not generate its own keys. Instead, the sink (or some other trusted party) would generate all public and secret keys for all sensors. The collector, however, would be given the public keys only.

FssAgg.Asig With inputs of message M_i to be signed, an aggregate-so-far signature $\sigma_{1,i-1}$ over messages $M_1, \cdots, M_{i-1}$ and the current signing key x_i, the signer first computes a BLS signature on M_i using x_i: $\sigma_i = \mathcal{H}^{x_i}(index||M_i)$ where $index$ denotes the position of M_i in the storage. The purpose of this index is to provide message ordering, since the original BGLS aggregation function does not impose any order on aggregate elements. Next, the signer aggregates σ_i onto $\sigma_{1,i-1}$ through multiplication: $\sigma_{1,i} = \sigma_{1,i-1} \cdot \sigma_i$. Finally, the signer updates the key.

FssAgg.Upd A signer evolves its secret signing key through the hash function $\mathcal{H}$: $x_i = \mathcal{H}(x_{i-1})$.

FssAgg.Aver The verifier uses Equation 3 and the public key pk to verify an aggregate signature $\sigma_{1,i}$

The security of our *FssAgg* signature scheme is based on the underlying BLS scheme and no other assumptions is needed. The following theorem summarizes the security of our *FssAgg* signature scheme and is strait-forward to prove. For completeness, a formal description of the security model and the proof of the theorem can be found in Appendix B and C.

Theorem 1. *If BLS is a $(t', q_H, q'_S, \epsilon)$-secure signature scheme, our construction above is a $(t, q_H, q_S, T, \epsilon)$-secure* FssAgg *signature scheme where $t' = t + O(q_H + q_S)$, $\epsilon' = \epsilon/T$, and $q'_S = q_S/T$.*

A proof sketch for this theorem is presented in Appendix C. (Appendix B contains the security model).

See Appendix A for some performance results.

6 Summary and Future Work

In this paper we motivated the need for Forward-Secure Sequential Authentication to address both key exposure and storage efficiency issues. We constructed two sample *FssAgg* schemes (one MAC-based and one signature-based). While our trivial MAC-based scheme is near-optimal in terms of efficiency, the signature-based scheme is not. Although it is both signer- and space-efficient, it is not verifier-friendly as the verifier needs $O(T)$ space to store the public key and the verification is fairly expensive because of bilinear map operations. Constructing a more efficient scheme – with either (or both) compact public keys or lower verification complexity – is a challenge for future work. And, a more careful formal treatment of Forward-Secure Sequential Authentication is certainly needed.

References

1. M. Abdalla, and L. Reyzin. "A new forward-secure digital signature scheme." In *ASIACRYPT 2000*, pp. 116-129, 2000.
2. R. Anderson. "Two remarks on public-key cryptology - Invited Lecture". *Fourth ACM Conference on Computer and Communications Security*, Apr. 1997.
3. K. Barr, and K. Asanovic. "Energy aware lossless data compression." In *Proc. of MobiSys'03*. San Francisco, CA, May 2003.
4. M. Bellare, and S. K. Miner. "A forward-secure digital signature scheme". In *Proc. of Adances in Cryptology - Crypto 99*, LNCS Vol 1666:431-448, Aug. 1999.
5. M. Bellare, and B. Yee. "Forward-Security in Private-Key Cryptography". In Proceedings of *CT-RSA'03*, LNCS Vol. 2612, M. Joye ed, Springer-Verlag, 2003.
6. D. Boneh, B. Lynn, and H. Shacham. "Short signatures from the Weil pairing". *J. Cryptology*, 17(4):297-319, Sept. 2004. Extended abstract in *Proceedings of Asiacrypt 2001*.
7. D. Boneh, C. Gentry, B. Lynn, and H. Shacham. "Aggregate and verifiably encrypted signatures from bilinear maps". In *Proc. of Eurocrypt 2003*, LNCS 2656:416-432, May 2003.
8. C. Castelluccia, E. Mykletun, and G. Tsudik. "Efficient aggregation of encrypted data in wireless networks". In *Mobile and Ubiquitous Systems: Networking and Services MobiQuitous 2005*. July 2005.
9. Y. Frankel, P. Gemmell, P.D. MacKenzie, and M. Yung. "Optimal resilience proactive public-key cryptosystems". In *FOCS*, 1997.
10. C. G. Gunther. "An identity-based key-exchange protocol." *Advances in Cryptology - EuroCrypt'89*. LNCS 434, pp. 29-37, 1990.
11. L. Hu, and D. Evans. "Secure aggregation for wireless networks." In *Workshop on Security and Assurance in Ad Hoc Networks*, 2003.
12. C. Intanagonwiwat, D. Estrin, R. Govindan, and J. Heidemann. "Impact of network density on data aggregation in wireless sensor networks". In *ICDCS'02*, pp. 457-458. 2002.
13. G. Itkis, and L. Reyzin. "Forward-secure signatures with optimal signing and verifying". In *Proc. of Advances in Cryptology - Crypto'01*, LNCS 2139:332-354, Aug. 2001.
14. A. Kozlov, and L. Reyzin. "Forward-secure signatures with fast key update". In *Prof. of the 3rd International Conference on Security in Communication Networks (SCN'02)*, 2002.
15. H. Krawczyk. "Simple forward-secure signatures from any signature scheme". In *Proc. 7th ACM Conference on Computer and Communication Security (CCS)*, pp. 108-115, Nov. 2000.
16. S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, and B. Waters. "Sequential aggregate signatures and multisignatures without random oracles". In *Prof. of Eurocrypt 2006*, May 2006.
17. A. Lysyanskaya, S. Micali, L. Reyzin, and H. Shacham. "Sequential aggregate signatures from trapdoor permutations". In *Proc. of Eurocrypt 2004*, LNCS 3027:245-254, Nov. 2001.
18. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. "Handbook of applied cryptography". *CRC Press*, 1997. ISBN 0-8493-8523-7.
19. A. Perrig, J. Stankovic, and D. Wagner. "Security in wireless sensor networks". *ACM Commun.*, 47(6):53-57, 2004.
20. B. Schneier, and J. Kelsey. "Cryptographic support for securelogs on untrusted machines". *ACM TISSEC*, Vol. 2, 1999.
21. D. Wagner. "Resilient aggregation in sensor networks". In *Workshops on Security of Ad Hoc and Sensor Networks.* 2004.
22. Y. Yang, X. Wang, S. Zhu, and G. Cao. "SDAP: a secure hop-by-hop data aggregation protocol for sensor networks". In *ACM MOBIHOC'06*. May 2006.

A Performance

In this section, we evaluate the performance of the proposed BLS-based *FssAgg* signature scheme. We begin by accessing the cost in terms of basic cryptographic operations(e.g, multiplications, exponentiation, etc). Then we show the actual overhead incurred through experiment.

We use the notation in Table 1. We consider the generation and verification of a *FssAgg* signature $\sigma_{1,k*t}$ where t denotes the number of periods occupied by $\sigma_{1,k*t}$ and k denotes the number of signatures generated per time period. Table 2 illustrates the overhead (computation, storage and bandwidth) associated with the scheme in terms of cryptographic operations.

Table 1. Notations.

$MtP^t(\mathcal{H}_1(\cdot))$	t map-to-point operations
$SclMult^t_m(l)$	t scalar multiplications with modulus of size m and exponent of size l
$SclAdd^t_m$	t scalar additions with modulus of size m
$BM(t)$	t bilinear mappings
$Hash^t(l)(\mathcal{H}(\cdot))$	t hash operations with input size of l

Table 2. Operation Cost in Terms of Cryptographic Operations.

Parameters	Cost	Complexity
Aggregate Signature Size	$\lvert p\rvert$	$O(1)$
Secret Key Size	$\lvert q\rvert$	$O(1)$
Key Update Time	$Hash(\lvert q\rvert)$	$O(1)$
Aggregate Signing Time	$MtP^1 + Exp^1_{\lvert p\rvert}(\lvert q\rvert)$ $+Mult^1(\lvert p\rvert)$	$O(1)$
Public Key Size	$T * \lvert q\rvert$	$O(T)$
Aggregate Verifying Time	$BM(t+1)+$ $+Mult^{k*t-1}(\lvert p\rvert)$	$O(t)$

Table 3. Operation Cost in msecs.

		BLS Sign		Aggregation
Asig	1 signature	MtP^1	$SclMul^1_p(q)$	$SclAdd^1_p$
		1.5	6.14	0.05
Aver	1 signature	53.62		
	k=1000, t=1	54.40		
	k=100, t=10	295.71		
	k=10, t=100	2708.79		

We used a field F_p where $|p| = 512$ and we choose the size of group order as $|q| = 160$. We test our scheme on a Pentium 1.86GHz machine with 512M memory. The experiment result is listed in Table 3. Signature generation is quite efficient and it costs an average 7.64ms to generate a BLS signature (1.5ms on the map-to-point operation and 6.14ms on the scalar multiplication operation) and another 0.05ms to fold it into the aggregate. Aggregation imposes little overhead on the overall time for *Asig*. Verification cost is quite expensive because of the involvement of pairing operations. When the number of time periods increases to 100, it takes the verifier more than 2 seconds to verify. The verification cost might impose an upper ceiling on the total number of time periods T.

B Security Model

The security of a *FssAgg* signature scheme is defined as the nonexistence of an adversary, capable, within the confines of a certain game, of existentially forging a *FssAgg* signature even in the event of exposure of the current secret key. Because a *FssAgg* signature scheme combines security properties from both a aggregate signature scheme and a forward-secure signature scheme, we describe a security model for it that is a hybrid of the aggregate chosen key model for aggregate signatures [7, 17] and the break-in model for forward-secure signatures [4].

This new security model reflects the way a *FssAgg* scheme is used. In this model, the adversary $\mathcal{A}$ first conducts an adaptive chosen message attack, requesting signatures on messages of its choice for as many time periods as it desires. Whenever it chooses, it "breaks in" and is given the secret key sk_b for the current time period b. Its goal is the existential forgery of a *FssAgg* signature pertaining to any past time periods before the break-in time period. A forgery $\sigma_{1,t}$ over messages $m_1, \cdots, m_t$ under keys $sk_i, \cdots, sk_t$

is considered as a valid forgery if at least one message m_i ($1 \leq i \leq t \leq b$) is not queried by $\mathcal{A}$ during the chosen message attack phase. To make explanation easy, we set $i = t$ and the attackers's goal is to forge a signature $\sigma_{1,t}$ such that m_t is not queried in the chosen message attack phase. The advantage of $\mathcal{A}$ is defined to be its probability of success in the following game.

- $\mathbf{Setup}$. The *FssAgg* forger $\mathcal{A}$ is provided with the public key pk and T.
- $\mathbf{Queries}$. The initial time period is $i = 1$. Proceeding adaptively, at time period i, $\mathcal{A}$ gets access to a signing oracle $\mathcal{O}_i$ under the current secret key sk_i. For each query, it also supplies a *FssAgg* signature σ_{i-1} on messages $m_1, \cdots, m_{i-1}$ signed by secret keys $sk_1, \cdots, sk_{i-1}$, and an additional message m_i to be signed by the oracle under key sk_i. $\mathcal{A}$ queries this as often as it wants until it indicates it is done for the current time period. Then $\mathcal{A}$ moves into the next time period $i + 1$ and it is provided with a signing oracle $\mathcal{O}_{i+1}$ under the secret key sk_{i+1}. The query process repeats until $\mathcal{A}$ chooses to break in.
- $\mathbf{Break-in}$. At time period b, $\mathcal{A}$ chooses to break in and is given the break-in privilege, the current secret key sk_b.
- $\mathbf{Response}$. Finally, $\mathcal{A}$ outputs a *FssAgg* signature $\sigma_{1,t}$ on messages $m_1, \cdots, m_t$ under keys $sk_1, \cdots, sk_t$. The forger wins if (1) $t < b$; (2) the *FssAgg* signature $\sigma_{1,t}$ is a valid *FssAgg* signature on messages $m_1, \cdots, m_t$ under keys $sk_1, \cdots, sk_t$, and (3) $\sigma_{1,t}$ is nontrivial, i.e., $\mathcal{A}$ did not ask $\mathcal{O}_t$ for signature query on message m_i at time i. The probability is over the coin tosses of the key-generation algorithm and of $\mathcal{A}$.

Definition 1. *A* FssAgg *forger $\mathcal{A}$ $(t, q_H, q_S, T, \epsilon)$-breaks a T-time-period* FssAgg *signature scheme in the* FssAgg *break-in model if: $\mathcal{A}$ runs in time at most t; $\mathcal{A}$ makes at most q_H queries to the hash function and at most q_S queries to the signing oracle; the advantage of $\mathcal{A}$ is at least ϵ; and the forged signature is taken over at most T time periods. A* FssAgg *scheme is $(t, q_H, q_S, T, \epsilon)$-secure against existential forgery in the* FssAgg *break-in model if no forger $(t, q_H, q_S, T, \epsilon)$-breaks it.*

C Proof of Theorem 1

Our proof is similar to the proof in [15].

Proof. Suppose there exist a forger $\mathcal{A}$ against the BLS-based *FssAgg* signature scheme that succeeds with ϵ. We build a simulator $\mathcal{B}$ to play the forgeability game against the BLS scheme. Given the chosen challenging public key cpk, forger $\mathcal{B}$ interacts with $\mathcal{A}$ as follows.

- **Setup**. Forger $\mathcal{B}$ first selects a random time period t between 1 and T, hoping $\mathcal{A}$'s eventual forgery will be for the t-th time period. $\mathcal{B}$ sets pk as the public key of time period t: $pk_t = cpk$. It is given an oracle O_{cpk} that given a message returns a signature on that message under the public key pk. Forger $\mathcal{B}$ generates information corresponding to the other time periods as follows: (1) $\mathcal{B}$ generates independently $t-1$ pairs of BLS public/secret key pairs: (pk_i, sk_i), $i = 1, \cdots, t-1$ for time period 1 to $t-1$; (2) $\mathcal{B}$ generates a random BLS public/private key pair and sets the pair as the key pair for time period $t + 1$. It uses the key update function to generate out of it $T - t - 1$ pairs of public/private keys. These pairs are set as keys for periods $t + 1, \cdots, T$. $\mathcal{B}$ provides $\mathcal{A}$ with $pk = \{pk_1, \cdots, pk_T\}$ and T .
- **Forward-secure Aggregate Signature Queries**. $\mathcal{A}$ is allowed to query for any *FssAgg* signature corresponding to any period of its choice except for the following restriction. Whenever $\mathcal{A}$ asks for a *FssAgg* signature corresponding to a period i, it cannot later ask for a signature corresponding to a previous period.

  At time period i, when $\mathcal{A}$ requests a *FssAgg* signature, it supplies a message m_i of its choice and an aggregate-so-far signature $\sigma_{1,i-1}$ on messages $m_1, \cdots, m_{i-1}$. If i is different than t then $\mathcal{B}$ generates a BLS signature σ_i on m_i with its knowledge of the signature keys for those periods (these keys were chosen by $\mathcal{B}$). If $i = t$, $\mathcal{B}$ goes to its oracle O_{cpk} to get the corresponding signature σ_i on m_i. Finally $\mathcal{B}$ returns $\sigma_{1,i} = \sigma_{1,i-1} \times \sigma_i$ to $\mathcal{A}$ as the *FssAgg* signature at time period i.

Break-in. When $\mathcal{A}$ decides to break in and query the secret information for some b-th period then $\mathcal{B}$ does the following. If $b \leq t$ then it aborts its run (i.e., in this case $\mathcal{B}$ fails to forge). If $b > t$ then $\mathcal{B}$ provides $\mathcal{A}$ the secret information for that period ($\mathcal{B}$ knows it).

Output. Finally $\mathcal{A}$ outputs a forgery $\sigma_{1,t'}$ over messages $m_1, \cdots, m'_t$ under keys $sk_1, \cdots, sk_{t'}$. $\mathcal{B}$ acts as follows. If $t' \neq t$, $\mathcal{B}$ aborts its run failing to forge. Otherwise if $t' = t$, $\mathcal{B}$ outputs a forgery $\sigma_t = \sigma_{1,t} \cdot \prod_{i=1}^{t} \sigma_i^{-1}$ where σ_i is a BLS signature over message m_i under key sk_i, $1 \leq i < t$ ($\mathcal{B}$ knows sk_i and so it can generate σ_i). Then

$$e(\sigma_t, g_2) = e(\sigma_{1,t} \cdot \prod_{i=1}^{t-1} \sigma_i^{-1}, g_2) = e(\sigma_{1,t}, g_2) \cdot e(\prod_{i=1}^{t-1} \sigma_i^{-1}, g_2) = \prod_{i=1}^{t} e(h_i, pk_i) \cdot e(\prod_{i=1}^{t-1} \sigma_i^{-1}, g_2) = e(h_t, pk_t) \tag{4}$$

That $\sigma_{1,t}$ is a valid *FssAgg* forgery means m_t is not queried by $\mathcal{A}$ during time period t, so in particular $\mathcal{B}$ did not ask for that signature from O_{cpk}. Hence σ_t is a valid forgery for $\mathcal{B}$.

Algorithm $\mathcal{B}$ makes as many as hash queries as $\mathcal{A}$ makes. Algorithm $\mathcal{B}$ makes at most $1/T$ signature queries as $\mathcal{A}$ makes. Algorithm $\mathcal{B}$'s running time is that of $\mathcal{A}$, plus the overhead in handling $\mathcal{A}$'s hash and signature queries.

If $\mathcal{A}$ succeeds with probability of ϵ in forging, $\mathcal{B}$ succeeds at least with probability roughly ϵ/T. The argument is outlined as follows. First, the view of $\mathcal{A}$ that $\mathcal{B}$ produces is computationally indistinguishable from the view of $\mathcal{A}$ interacting with a real *FssAgg* signing oracle (where all secret keys are produced out of a single initial seed from the forward-secure hash function $\mathcal{H}$). Indeed, if a distinguisher exists for these two views of $\mathcal{A}$ then, we can construct a distinguisher for $\mathcal{H}$. Next, conditioned on $\mathcal{B}$ choosing the value of t as the period for which $\mathcal{A}$ eventually output a forgery, we have the probability that $\mathcal{B}$ outputs a forgery against the choosing public key pk is the same probability that $\mathcal{A}$ succeeds in forging, i.e., probability ϵ. Since choosing the "right" t happens with probability $1/T$ we get that ϵ/T is an approximate lower bound on the forging probability of $\mathcal{B}$.