Some Aggregate Forward-Secure Signature Schemes

N.R.Sunitha
Department of Computer Science & Engg.
Siddaganga Institute of Technology,
Tumkur, Karnataka, India.

B.B.Amberker
Department of Computer Science & Engg.
National Institute of Technology,
Warangal, Andhra Pradesh, India.

Abstract

Ordinary digital signatures have an inherent weakness: if the secret key is leaked, then all signatures, even the ones generated before the leak, are no longer trustworthy. Forward-secure digital signatures address this weakness, they ensure that the past signatures remain secure even if the current secret key is leaked.

Following the notion of aggregate signatures introduced by Boneh et al, which provides compression of signatures, we have come up with aggregate signature schemes for ElGamal/DSA/Bellare-Miner forward-secure signatures. We describe two schemes of aggregation for the Bellare-Miner Scheme. The first is a aggregate signature scheme with aggregation done separately in different time periods.The second is a aggregate signature scheme with aggregation done for a set of time periods.

Keywords : Aggregate Signature, Forward-Security, Key evolution, Hash function, Digital Signature.

1 Introduction

Aggregate signature schemes were introduced in 2003 by Boneh, Gentry, Lynn and Shacham [**?**]. Basically, an aggregate signature scheme is a digital signature that supports aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single short signature. This single signature will convince the verifier that the n users did indeed sign the n original messages (i.e., user i signed message M_i for $i = 1, \ldots, n$). The advantage of these signatures is that they provide compression of signatures.

In a general signature aggregation scheme each user i signs her message M_i to obtain a signature σ_i. Then anyone can use a public aggregation algorithm to take all n signatures $\sigma_1, \ldots, \sigma_n$ and compress them into a single signature σ. Moreover, the aggregation can be performed incrementally. Signatures σ_1, σ_2 can be aggregated into σ_{12} which can then be further aggregated with σ_3 to obtain σ_{123}, and so on. There is also an aggregate verification algorithm that takes $PK_1, \ldots, PK_n, M_1, \ldots, M_n$ and σ to decide whether the aggregate signature is valid. Thus, an aggregate signature provides non-repudiation at once on many different messages by many users. This is referred to as general aggregation since aggregation can be done by anyone and without the cooperation of the signers.

In another type of aggregation called sequential aggregation scheme, signature aggregation can only be done during the signing process. Each signer in turn sequentially adds her signature to the current aggregate. Thus, there is an explicit order imposed on the aggregate signature and the signers must communicate with each other during the aggregation process. Operationally, sequential aggregation works as follows: $User_1$ signs M_1 to obtain σ_1; $User_2$ then combines σ_1 and M_2 to obtain σ_2; and so on. The final signature σ_n binds $User_i$ to M_i for all $i = 1, \ldots, n$.

In [6], the concept of an aggregate signature, security models for such signatures, and applications for aggregate signatures are presented. They construct an efficient aggregate signature from a recent short signature scheme based on bilinear maps due to Boneh, Lynn, and Shacham [6]. In [7], the authors survey two aggregate signature schemes. The first is based on the short signature scheme of Boneh, Lynn, and Shacham and supports general aggregation. The second, based on a multisignature scheme of Micali, Ohta, and Reyzin, is built from any trapdoor permutation but only supports sequential aggregation. In [4], the authors propose sequential aggregate signatures, in which the set of signers is ordered. The aggregate signature is computed by having each signer, in turn, add his signature to it. They show how to realize this in such a way that the size of the aggregate signature is independent of the number of signatures. In [8], the authors consider FssAgg (Forward-secure signature aggregation) 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 only represents the initial investigation of Forward-Secure Aggregation as the proposed schemes are not specific or optimal. In a designated verifier aggregation scheme [3, 12], an aggregate signature is addressed to a specific verifier. And only this specific verifier needs to be convinced of the integrity and origin of the signed messages.

Ordinary digital signatures have an inherent weakness: if the secret key is leaked, then all signatures, even the ones generated before the leak, are no longer trustworthy. Forward-secure digital signatures address this weakness, they ensure that the past signatures remain secure even if the current secret key is leaked.

Following the notion of aggregate signatures introduced by Boneh et al, which provides compression of signatures, we have come up with aggregate signature schemes for ElGamal/DSA/Bellare-Miner forward-secure signatures.

The organisation of our paper is as follows: In Section 2, we describe briefly the properties of forward-secure signature schemes and in particular discuss the Forward-secure Bellare-Miner Scheme. In Section 3, we describe two schemes of aggregation for the Bellare-Miner Scheme. The first is a aggregate signature scheme with aggregation done separately in different time periods.The second is a aggregate signature scheme with aggregation done for a set of time periods. In Section 4, we discuss the Forward-Secure DSA Signature scheme and the corresponding aggregation. In Section 5, we discuss the Forward-Secure ElGamal Signature scheme and the corresponding aggregation. Lastly in Section 6, we conclude.

2 Forward Secure Signature Scheme

Digital signatures are vulnerable to leakage of secret key. If the secret key is compromised, any message can be forged. To prevent future forgery of signatures, both public key and secret key must be changed. Notice, that this will not protect previously signed messages: such messages will have to be re-signed with new pair of public key and secret key, but this is not feasible. Also changing the keys frequently is not a practical solution.

To address the above problem, the notion of forward security for digital signatures was first proposed by Anderson in [1], and carefully formalised by Bellare and Miner in [5] (see also[2, 10, 9, 11]). The basic idea is to extend a standard digital signature scheme with a key update algorithm so that the secret key can be changed frequently while the public key stays the same. Unlike a standard signature scheme, a forward secure signature scheme has its operation divided into time periods, each of which uses a different secret key to sign a message. The key update algorithm computes the secret key for the new time period based on the previous one using a one way function. Thus, given the secret key for any time period, it is hard to compute any of the previously used secret keys. (It is important for the signer to delete the old secret key as soon as the new one is generated, since otherwise an adversary breaking the system could easily get hold of these undeleted keys and forge signatures.) Therefore a receiver with a message signed before the period in which the secret key gets compromised, can still trust this signature, for it is still hard to any adversary to forge previous signatures.

To specify a forward-secure signature scheme, we need to (i) give a rule for updating the secret key (ii) specify the public key and (iii) specify the signing and the verification algorithms.

2.1 Bellare-Miner Forward-secure scheme

For the sake of completeness we describe the algorithms of the Bellare-Miner scheme.

Key generation: The signer generates the keys by running the following algorithm which takes as input the security parameter k, the number l of points in the keys and the number T of time periods over which the scheme is to operate.

Pick at random, distinct $k/2$ bit primes p, q each congruent to $3 \bmod 4$ and set $N \leftarrow pq$. N is a Blum Williams integer.

The base secret key $SK_0 = (S_{1,0}, \ldots, S_{l,0}, N, 0)$ (where $S_{i,0} \xleftarrow{R} Z_N^*$).

For verifying signatures, the verifier is given the public key PK, calculated as the value obtained on updating the base secret key $T+1$ times: $PK = (U_1, \ldots, U_l, N, T)$ where $U_i = S_{i,0}^{2^{T+1}} \bmod N, i = 1, \ldots, l$.

Key evolution: During time period j the signer signs using key SK_j. This key is generated at the start of period j by applying a key update algorithm to the key SK_{j-1}. The update algorithm squares the l points of the secret key at the previous stage to get the secret key at the next stage. Once this update is performed the signer deletes the key SK_j. Since squaring modulo N is a one way function, when the factorization of N is unknown it is computationally infeasible to recover SK_{j-1} from SK_j.

The secret key $SK_j = (S_{1,j}, \ldots, S_{l,j}, N, j)$ of the time period j is obtained from the secret key $SK_{j-1} = (S_{1,j-1}, \ldots, S_{l,j-1}, N, j-1)$ of the previous time period via the update rule: $S_{i,j} = S_{i,j-1}^2 \bmod N, i = 1, \ldots, l$.

Signature Generation: It has as input the secret key SK_j of the current period, the message M to be signed, and the value j of the period itself to return a signature $\langle j, (Y, Z) \rangle$ where Y, Z in Z_N^* are calculated as follows:

$$Y = R^{2^{(T+1-j)}} \bmod N \tag{1}$$

where $R \overset{R}{\leftarrow} Z_N^*$ and

$$Z = R \prod_{i=1}^{l} S_{i,j}^{c_i} \bmod N \tag{2}$$

with

$$c_1, \ldots, c_l = H(j, Y, M) \tag{3}$$

being the l output bits of a public hash function.

Signature Verification: A claimed signature $\langle j, (Y, Z)\rangle$ for the message M in time period j is accepted if

$$Z^{2^{(T+1-j)}} = Y \prod_{i=1}^{l} U_i^{c_i} \bmod N \tag{4}$$

where $c_1, \ldots, c_l = H(j, Y, M)$, and rejected otherwise. Notice that since

$$\begin{aligned} Z^{2^{(T+1-j)}} &= (R(\prod_{i=1}^{l} S_{i,j}^{c_i})^{2^{(T+1-j)}} \bmod N \\ &= Y.(\prod_{i=1}^{l} S_{i,0}^{2^{(T+1)} c_i}) \bmod N \\ &= Y.\prod_{i=1}^{l} U_i^{c_i} \bmod N. \end{aligned}$$

a signature by an honest signer with the secret key will be accepted.

3 Aggregate signature scheme for Forward-secure signatures with aggregation done separately in different time periods

Here we propose a forward-secure aggregate signature scheme based on Bellare-Miner Scheme in which given n signatures, $n = n_1 + n_2 + \ldots + n_T$, where n_j are the number of signatures signed by a single signer in the j^{th} period on n_j distinct messages. We aggregate the signatures in different time periods separately *i.e* each of the n_j signatures are considered for aggregation separately.

Aggregate Signature Generation: Let $\langle (M_{j,1}, j, (Y_{j,1}, Z_{j,1})), \ldots, (M_{j,nj}, j, (Y_{j,nj}, Z_{j,nj}))\rangle$ be the signatures generated as discussed in Section 3 in any j^{th} period. The aggregate signature is obtained by computing the product of the individual components of the signatures. Therefore, the aggregate signature is $\langle (j, Y_{A,j}, Z_{A,j}, M_{j,1}, \ldots, M_{j,nj})\rangle$, where

$$Y_{A,j} = Y_{j,1} \ldots Y_{j,nj} \bmod N \tag{5}$$

$$Z_{A,j} = Z_{j,1} \ldots Z_{j,nj} \bmod N. \tag{6}$$

Aggregate Signature Verification: The verification equation for time period j is given by

$$Z_{A,j}^{2^{(T+1-j)}} = Y_{A,j}. \prod_{i=1}^{l} U_i^{(c_{M_{j,1},i} + \ldots + c_{M_{j,nj},i})} \bmod N. \tag{7}$$

where $c_{M_{j,1},1}, \ldots, c_{M_{j,l},l} = H(j, Y_{j,1}, M_{j,1})$. Notice that since

$$\begin{aligned} LHS &= Z_{j,1}^{2^{(T+1-j)}} \ldots Z_{j,n_j}^{2^{(T+1-j)}} \bmod N \\ &= (R_1. \prod_{i=1}^{l} S_{i,j}^{c_{M_{j,1},i}})^{2^{(T+1-j)}} \ldots \\ &\quad (R_{nj}. \prod_{i=1}^{l} S_{i,j}^{c_{M_{j,nj},i}})^{2^{(T+1-j)}} \bmod N \\ &= (R_1 \ldots R_{n_j})^{2^{(T+1-j)}}. \\ &\quad (\prod_{i=1}^{l} S_{i,j}^{c_{M_{j,1},i} + \ldots + c_{M_{j,nj},i}})^{2^{(T+1-j)}} \bmod N \\ &= Y_{A,j}(\prod_{i=1}^{l} S_{i,j}^{c_{M_{j,1},i} + \ldots + c_{M_{j,nj},i}})^{2^{(T+1-j)}} \bmod N \\ &= Y_{A,j}.(\prod_{i=1}^{l} S_{i,0}^{2^{(T+1)}.(c_{M_{j,1},i} + \ldots + c_{M_{j,nj},i})}) \bmod N \\ &= Y_{A,j}. \prod_{i=1}^{l} U_i^{(c_{M_{j,1},i} + \ldots + c_{M_{j,nj},i})} \bmod N. \\ &= RHS, \end{aligned}$$

an aggregate signature generated by a honest signer with his secret key will be accepted.

3.1 Aggregate signature scheme for Forward-secure signatures with aggregation done for a set of time periods

We propose another aggregate signature scheme for Bellare-Miner Scheme in which given n signatures, $n = n_1 + n_2 + \ldots + n_T$, where n_j are the number of signatures signed in the j^{th} period on n_j distinct messages by a single signer. We can aggregate all the signatures occurring in any m distinct time periods, $i_1, \ldots, i_m$. Here for convenience and to reduce the complexity of equations we consider $n_1 = n_2 = \ldots = n_j = 1$.

Aggregate Signature Generation: Let $\langle (M_{i_1,1}, i_1, (Y_{i_1,1}, Z_{i_1,1})), \ldots, (M_{i_m,1}, i_m, (Y_{i_m,1}, Z_{i_m,1}))\rangle$ be the m signatures generated as discussed in Section 2 in m time periods $I = \{i_1, i_2, \ldots, i_m\}$

by a single signer. The aggregate signature is $\langle (i_1 \dots i_m, Y_A, Z_A, M_{i_1,1}, \dots, M_{i_m,1}) \rangle$, where

$$Y_A = Y_{i_1,1} \dots Y_{i_m,1} \bmod N \tag{8}$$

$$Z_A = Z_{i_1,1}^{2^{(T+1-i_1)}} \dots Z_{i_m,1}^{2^{(T+1-i_m)}} \bmod N. \tag{9}$$

Aggregate Signature Verification: The verification equation is given by

$$Z_A = Y_A . \prod_{i \in I} \prod_{j=1}^{l} U_j^{c_{M_{i,1,j}}} \bmod N. \tag{10}$$

where $c_{M_{i_k},1,1}, \dots, c_{M_{i_k},1,l} = H(i_k, Y_{i_k,1}, M_{i_k,1}),\ k = 1, 2, \dots, m$

Notice that since

$$\begin{aligned}
LHS &= Z_{i_1,1}^{2^{(T+1-i_1)}} \dots Z_{i_m,1}^{2^{(T+1-i_m)}} \bmod N \\
&= (R_{i_1}(\prod_{j=1}^{l} S_{j,i_1}^{c_{M_{i_1,1,j}}}))^{2^{(T+1-i_1)}} \dots \\
&\quad (R_{i_m}(\prod_{j=1}^{l} S_{j,i_m}^{c_{M_{i_m,1,j}}}))^{2^{(T+1-i_m)}} \bmod N \\
&= R_{i_1}^{2^{(T+1-i_1)}} \dots R_{i_m}^{2^{(T+1-i_m)}} . \\
&\quad (\prod_{j=1}^{l} S_{j,i_1}^{c_{M_{i_1,1,j}}})^{2^{(T+1-i_1)}} \\
&\quad \dots (\prod_{j=1}^{l} S_{j,i_m}^{c_{M_{i_m,1,j}}})^{2^{(T+1-i_m)}} \bmod N \\
&= Y_{i_1,1} \dots Y_{i_m,1} . (\prod_{j=1}^{l} S_{j,0}^{2^{(T+1)} . c_{M_{i_1,1,j}}}) \\
&\quad \dots (\prod_{j=1}^{l} S_{j,0}^{2^{(T+1)} c_{M_{i_m,1,j}}}) \bmod N \\
&= Y_A . \prod_{j=1}^{l} U_j^{c_{M_{i_1,1,j}}} \dots \prod_{j=1}^{l} U_j^{c_{M_{i_m,1,j}}} \bmod N. \\
&= RHS,
\end{aligned}$$

an aggregate signature generated by a honest signer with his secret key will be accepted.

4 Forward Secure DSA Signature Scheme

To specify a forward-secure signature scheme, we need to (i) give a rule for updating the secret key (ii) specify the public key and (iii) specify the signing and the verification algorithms.

In saying that our forward-secure scheme is based on a basic signature scheme, we mean that, given a message and the secret key of a time period, the signing algorithm is the same as in the basic signature scheme. The public key for the forward-secure signature scheme is the key obtained on running T times the update rule for secret keys.

Table 1: For prime p of size $|p|$ bits, $\phi^T(p)$ has a prime factor of size 160 bits.

$\lvert p \rvert$	p	T
256	23158417847463239084714197 00173758157065399693312811 28078915168015826259280709	56
256	23158417847463239084714197 00173758157065399693312811 28078915168015826259280027	56
274	60708402882054033466233184 58823496583257521372037936 0039119137804340758912662766479	77
274	6070840288205403346623318458823 49658325752137203793600391191 37804340758912662765931	73
512	268156158598851941991480499964 116922549587316411847867554471 228874435280601470939536037485 963338068553800637163729721017 07507765623893139892867298012168351	266

Now, we need to be able to write a verification equation relating the public key and the signature (and incorporating the time period of the signature) from which the claim of forward security can be deduced.

Here are the details.

1. **Secret Key Updation**
   Let p be a large prime. Let $\phi(p-1) = p_1^{r_1} \dots p_k^{r_k}$ where $p_1 < p_2 < \dots < p_k$.
   Choose g such that

   $$\gcd(g, p) = 1,\ \gcd(g, \phi(p)) = 1,\ \gcd(g, \phi^2(p)) = 1,\ \dots,\ \gcd(g, \phi^{T-1}(p)) = 1$$

   where $\phi(p)$ is the Euler totient function and $\phi^{T-i}(p) = \phi(\phi^{T-i-1}(p))$ for $1 \le i \le T-1$ with $\phi^0(p) = p$. It may be noted that a prime g chosen in the range $p_k < g < p$ satisfies the above condition. The base secret key a_0 (this is the initialisation for the secret key updation) is chosen randomly in the range $1 < a_0 < p-1$.
   The secret key a_i in any time period i is derived as a function of a_{i-1}, the secret key in the time period $i-1$, as follows:

   $$a_i = g^{a_{i-1} \bmod \phi^{T-i+1}(p)} \bmod \phi^{T-i}(p) \tag{11}$$

   for $1 \le i < T$. Once the new secret key a_i is generated for time period i, the previous secret key a_{i-1} is deleted. Thus an attacker breaking in period i will get a_i but cannot compute $a_0, \dots, a_{i-1}$, because of difficulty of computing discrete logarithms. For a given large prime p, though the value of $\phi^i(p)$ decreases exponentially over time i, we have determined experimentally (see Table 1) that for the following typical values of p, $\phi^i(p)$ factor into primes of size greater than 2^{160} for reasonable value of T. Therefore, we assume that computing discrete logarithms mod $\phi^{T-i}(p)$ is hard, for $1 \leq i < T$.

2. **Public Key Generation**
   We obtain the public key by executing the Secret Key Updation Algorithm T times as follows :
   $$\beta = g^{a_{T-1}} \mod p = a_T \mod p \tag{12}$$
3. **Signature Generation:** The signature generated in any time period i is $\langle r, s, i\rangle$. The computation of r is
   $$r = (g^k \mod p) \mod q \tag{13}$$
   where k is a random number chosen such that $0 < k < p$ and $gcd(k, (p-1)) = 1$.
   The computation of s is
   $$s = k^{-1}(SHA(m||i) + (\mathcal{A}(g, T-i-1, a_i) * r)) \mod q \tag{14}$$
   where SHA is a collision-resistant hash function. While hashing, i is concatenated with m to indicate the time period in which the message is signed.

   The notation $\mathcal{A}(\alpha, u, v) = \alpha^{\cdot^{\cdot^{\cdot^{\alpha^v}}}}$ we mean that there are u number of α 's in the tower and the topmost α is raised to v, i.e in the above equation there are $(T-i-1)$ number of α's in the tower and the topmost α is raised to a_i.
   Notice that the public key β can also be given in terms of a_i as,
   $$\beta = \mathcal{A}(g, T-i, a_i) \bmod p, \tag{15}$$
   This relation gets employed in the verification of validity of the signature.
4. **Verification:**
   $$\begin{aligned} w &= (s)^{-1} \\ u1 &= SHA(m||i) * w \\ u2 &= r * w \\ v &= g^{u1} * \beta^{u2} \end{aligned}$$
   A claimed signature $\langle r, s, i\rangle$ for the message m in time period i is accepted if
   $$v = r \tag{16}$$
   else rejected.

Recall that the claim of security of the standard DSA signature scheme is based on the difficulty of computing discrete logarithms. The same security guarantee is obtained in the Forward-secure DSA Signature Scheme.

4.1 Aggregate Signatures for Forward-Secure DSA

Let $\langle (M_{i_1,1}, i_1, (r_{i_1,1}, s_{i_1,1})) \ldots (M_{i_m,1}, i_m, (r_{i_m,1}, s_{i_m,1})\rangle$ be the m DSA forward-secure signatures generated in m time periods $I = \{i_1, \ldots, i_m\}$ by a single signer. The aggregate signature is obtained by computing the following:

$$\sigma_1 = r_{i_1,1}^{r_{i_1,1}^{-1}.s_{i_1,1}} \ldots r_{i_m,1}^{r_{i_m,1}.s_{i_m,1}} \bmod p$$

$$\sigma_2 = (SHA(M_{i_1,1})r_{i_1,1}^{-1} + \ldots + SHA(M_{i_m,1})r_{i_m,1}^{-1}).H(\sigma_1) \bmod p.$$

The verification equation is given by

$$\alpha^{\sigma_2} = ((\beta)^{-m}.\sigma_1)^{H(\sigma_1)} \bmod p$$

Since

$$\begin{aligned} RHS &= (\beta^{-m}.r_{i_1,1}^{r_{i_1,1}^{-1}.s_{i_1,1}} \ldots r_{i_m,1}^{r_{i_m,1}.s_{i_m,1}})^{H(\sigma_1)} \bmod p \\ &= (\beta^{-m}.g^{k_{i_1}.k_{i_1}^{-1}(SHA(M_{i_1,1})+\mathcal{A}(g,T-1-i_1,a_{i_1}).r_{i_1,1})r_{i_1,1}^{-1}} \\ &\quad \ldots g^{k_{i_m}.k_{i_m}^{-1}(SHA(M_{i_m,1})+\mathcal{A}(g,T-j-i_m,a_{i_m}).r_{i_m,1}).r_{i_m,1}^{-1}})^{H(\sigma_1)} \\ &= (\beta^{-m}.g^{(SHA(M_{i_1,1}).r_{i_m,1}^{-1})} \ldots \\ &\quad g^{(SHA(M_{i_m,1}).r_{i_m,1}^{-1})}.\beta^m)^{H(\sigma_1)} \bmod p \\ &= (g^{(SHA(M_{i_1,1}).r_{i_1,1}^{-1})} \ldots g^{(SHA(M_{i_m,1}).r_{i_m,1}^{-1})})^{H(\sigma_1)} \bmod p \\ &= g^{\sigma_2} \bmod p \\ &= LHS, \end{aligned}$$

a set of messages signed by a honest signer will be accepted. This can be easily extended to any number of users.

5 Forward Secure ElGamal Signature Scheme

As the Secret Key Updation Algorithm and Public Key Generation Algorithm remains the same as in Forward-Secure DSA scheme, we discuss only the Signature Generation and Signature Verification algorithms. Here are the details.

1. **Signature Generation**
   The signature generated in any time period i is $\langle y_{1,i}, y_{2,i}\rangle$. The computation of $y_{1,i}$ is
   $$y_{1,i} = \alpha^k \mod p \tag{17}$$
   where k is a random number chosen such that $0 < k < p$ and $gcd(k, (p-1)) = 1$.
   The computation of $y_{2,i}$ is
   $$y_{2,i} = (H(m||i) - (\mathcal{A}(\alpha, T-i-1, a_i).y_{1,i}))k^{-1} \mod (p-1) \tag{18}$$
   where H is a collision-resistant hash function. While hashing, i is concatenated with m to indicate the time period in which the message is signed.

   Notice that the public key β can also be given in terms of a_i as,
   $$\beta = \mathcal{A}(\alpha, T-i, a_i) \bmod p, \tag{19}$$

This relation gets employed in the verification of validity of the signature.

2. **Verification**

   A claimed signature $\langle y_{1,i}, y_{2,i} \rangle$ for the message m in time period i is accepted if

$$\alpha^{H(m||i)} = \beta^{y_{1,i}} \;\; y_{1,i}^{y_{2,i}} \bmod p \tag{20}$$

else rejected.

5.1 Aggregate Signatures for Forward-Secure Elgamal Signature Scheme

Let $\langle (M_{i_1,1}, i_1, (y_{i_1,1}, y'_{i_1,1})) \ldots (M_{i_m,1}, i_m, (y_{i_m,1}, y'_{i_m,1})\rangle$ be the m forward-secure ElGamal signatures generated in m time periods $I = \{i_1, i_2, \ldots, i_m\}$ by a single signer. The aggregate signature is obtained by computing the following:

$$\sigma_1 = y_{i_1,1}^{y_{i_1,1}^{-1} \cdot y'_{i_1,1}} \ldots y_{i_m,1}^{y_{i_m,1} \cdot y'_{i_m,1}} \bmod p$$

$$\sigma_2 = (SHA(M_{i_1,1}) y_{i_1,1}^{-1} + \ldots + SHA(M_{i_m,1}) y_{i_m,1}^{-1}).H(\sigma_1) \bmod p.$$

The verification equation is given by

$$g^{\sigma_2} = ((\beta)^m.\sigma_1)^{H(\sigma_1)} \bmod p.$$

Since,

$$\begin{aligned}
RHS &= (\beta^m . y_{i_1,1}^{y_{i_1,1}^{-1} \cdot y'_{i_1,1}} \ldots y_{i_m,1}^{y_{i_m,1} \cdot y'_{i_m,1}})^{H(\sigma_1)} \bmod p \\
&= (\beta^m . g^{k_{i_1} . k_{i_m}^{-1} (H(M_{i_1,1}) - \mathcal{A}(g, T-1-1, a_{i_1}) . y_{i_1,1}) y_{i_1,1}^{-1}} \\
&\quad \ldots g^{k_j . k_j^{-1} (H(M_{j,1}) - \mathcal{A}(g, T-j-1, a_{i_m}) . y_{i_m,1}) . y_{i_m,1}^{-1}})^{H(\sigma_1)} \\
&= (\beta^m . g^{(H(M_{i_1,1}) . y_{i_1,1}^{-1})} \ldots \\
&\quad g^{(H(M_{i_m,1}) . y_{i_m,1}^{-1})} . \beta^{-m})^{H(\sigma_1)} \bmod p \\
&= (g^{(SHA(M_{i_1,1}) . y_{i_1,1}^{-1})} \ldots g^{(H(M_{i_m,j}) . y_{i_m,1}^{-1})})^{H(\sigma_1)} \bmod p \\
&= g^{\sigma_2} \bmod p \\
&= LHS,
\end{aligned}$$

a set of messages signed by a honest signer will be accepted. This can be easily extended to any number of users.

6 Conclusion

Following the notion of aggregate signatures introduced by Boneh et al, which provides compression of signatures, we have come up with aggregate signature schemes for ElGamal/DSA/Bellare-Miner forward-secure signatures. We describe two schemes of aggregation for the Bellare-Miner Scheme. The first is a aggregate signature scheme with aggregation done separately in different time periods.The second is a aggregate signature scheme with aggregation done for a set of time periods.

References

[1] Anderson, R.: Invited Lecture, Fourth Annual Conference on Computer and Communications Security, ACM, (1997).

[2] Abdalla,M., Reyzin,L. *A New Forward-Secure Digital Signature Scheme.* In: ASIACRYPT 2000, LNCS 1976, pp. 116-129. Springer-Verlag, (2000),116-129.

[3] Akihiro Mihara, Keisuke Tanaka, *Universal Designated-Verifier Signature with Aggregation* In: Proceedings of the Third International Conference on Information Technology and Applications (ICITA05), IEEE Computer Society.

[4] Anna Lysyanskaya, Silvio Micali, Leonid Reyzin, Hovav Shacham, *Sequential Aggregate Signatures from Trapdoor Permutations*, In Proceedings of Eurocrypt 2004, pp. 74-90.

[5] Bellare, M., Miner, S. *A Forward-Secure Digital Signature Scheme.* In: Wiener, M. (eds.): Advances in Cryptology-Crypto 99 proceedings, Lecture notes in Computer Science, Vol. 1666. Springer-Verlag, (1999).

[6] 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.

[7] Dan Boneh, Craig Gentry, Ben Lynn Hovav Shacham, *A Survey of Two Signature Aggregation Techniques.* In Proc. CryptoBytes Vol.6, No.2, 2003.

[8] D. Ma, and G. Tsudik. *Forward-secure sequentical aggregate authentication* IACR ePrint 2007/052.

[9] Itkis, G., Reyzin, L. *Forward-secure signatures with optimal signing and verifying.* In: CRYPTO'01, LNCS 2139, Springer-Verlag, (2001), 332-354.

[10] Krawczyk, H. *Simple forward-secure signatures from any signature scheme.* In: Proc. of the 7th ACM Conference on Computer and Communications Security (CCS 2000), ACM, (2000), 108-115.

[11] Kozlov, A, Reyzin, L.: *Forward-Secure Signatures with Fast Key Update.* In: Security in Communication Networks (SCN 2002), LNCS 2576, Springer-Verlag, (2002), (241-256).

[12] R.Bhasker, J.Herranz, F.Laguillaumie,*Aggregate Designated Verifier Signatures and Application to Secure Routing*, In International Journal of Security and Networks, Vol-2,pp 192-201, 2007.