Yet Another Forward Secure Signature from Bilinear Pairings

Duc-Liem Vo and Kwangjo Kim

International Research center for Information Security (IRIS)
Information and Communications University (ICU)
119 Munji-ro, Yuseong-gu, Daejeon, 305-714, Korea
{vdliem, kkj}@icu.ac.kr

Abstract. In this work, we have proposed yet another forward secure signature based on bilinear pairings. Our forward secure signature requires the general security parameters only independent to the total number of time periods. The scheme can perform key evolving for unlimited time periods while maintaining sizes of keys and signature fixed. In addition, the signing algorithm is very efficient with the simple verification algorithm. We also provide a formal definition along with a detailed security proof of our signature scheme under the assumption of Computational Diffie-Hellman problem.

Key words: Forward security, pairings, key exposure, key evolution.

1 Introduction

Key Exposure Problem and Forward Secure Signatures Digital signatures are very essential components in cryptography as well as in various applications nowadays. Using a digital signature, a signer can prove whether an electronic document is produced by him/her or not. Clearly, the signing ability must be restricted to the authorized people (signers), and the signing keys (or private keys) should be kept secret. Compromise of the private keys will cause severe damage to the applications using the digital signatures. In such cases, we cannot believe in any signature that will be produced in the future by the compromised key. Moreover, the signatures generated by that key in the past are suspicious too. We can imagine how serious this kind of damage can bring to the systems that rely on digital signatures such as banking systems, certificate authorities (CAs), and so on. Once a bank's private key is compromised, no one can be sure that given money (digitally signed by the bank) is true or forged one, hence for the safety reason, the bank may revoke those signatures issued previously, making some people lose their money. Obviously, reissuing all past signatures also is a burden to the bank. The similar devastation can be occurred if the CA's root private key is compromised. All certificates issued by this CA become unverifiable until new certificates are reissued with a new root key, and clients are updated this new key. This situation is more serious since the damage

may happen widely over the Internet and it may take an unexpected time to recover from the damage completely.

To deal with the key exposure problem, there are several solutions. The first one may think is to use key revocation mechanism by certificate revocation. This method can prevent the forgery of the signatures in the future (*i.e.*, after key is compromised), however, it cannot protect for the past signatures. Time-stamp service, introduced in [14], is another way to certify the creation date of a document. Anderson [3] suggested a new method for constructing signature scheme in which the private key is updated periodically while the public key is kept unchanged. With this approach, the compromise of a private key in a certain period does not affect to the past signatures signed by the previously updated private key. From the Anderson's proposal, there are many researches on this type of signature schemes including its formalization and applications.

In [5], Bellare and Miner formalized Anderson's idea by the concepts of the forward secure signature scheme and presented their construction of this signature using a key evolution scheme. This construction is based on a binary tree model in which the total life time of the scheme is divided into a small period. The total periods equal to the number of a binary tree's leaves. In each period, a different private key is used for signing messages and deriving a new private key for the next time period. The public key remains unchanged for all time periods. An adversary, who has the private key in a time period, can produce the next time period's private key but has no way to forge any signature of the previous time periods. Therefore, the signature scheme is forward secure.

From this work, a variety of the forward secure signature schemes has been proposed with many improvements. A new forward signature scheme with shorter keys and more practical was suggested by Abdalla and Reyzin [1]. Krawczyk [19] suggested further improvement by presenting a method to construct a forward secure signature scheme from any signature scheme, such as RSA or DSA signature scheme. Itkis and Reyzin [16] proposed another forward secure signature based on Guillou-Quisquater's signature scheme. Although this scheme is efficient in signing and verifying, it increases the size of key and signature. Malkin *et al.* [20] proposed a new construction of a forward secure signature scheme based on a product and sum composition method. Utilizing this method, one can construct a new forward signature scheme with more time periods from any two forward digital signature schemes. The combining of the sum and product compositions leads to a *MMM* [20] tree construction of a forward secure signature scheme. The advantage of *MMM* tree construction is that the maximal number of time periods needs not to be fixed in advance. F. Hu *et al.* [15] suggested a new forward secure signature scheme from bilinear maps influenced by the forward secure public key encryption schemes [9, 11]; Hu's construction was based on the scheme in [13]. The scheme achieved the small size of key and signature due to the property of an elliptic curve.

Along with these contributions, many valuable researches on other aspects of the forward secure signature have been carried out. Kozlov and Reyzin [18] proposed a forward signature scheme with fast key update; Song [21] suggested

a forward secure group signature scheme; Duc *et al.* [10] proposed a forward secure blind signature scheme based on the strong RSA problem. The researches on threshold signature scheme with forward secrecy are presented in [1] and [22]. Recently, key insulated and intrusion resilience mechanisms are also explored to provide high level of security. However, these mechanisms require interaction between the devices and the server for each time period. In some cases, these methods are not suitable.

Our Contribution Clearly, the forward secure signature provides stronger security than the traditional signature. Nevertheless, the forward secure signature scheme requires additional computation as well as storage for the key updating processes. In addition, some schemes are limited in the number of time periods. When all time periods are over, the key generation process is invoked to create new keys and new time periods. The new public key needs to be republished too. To overcome this limitation, we have proposed a new forward secure signature scheme which has unlimited time periods using bilinear pairings. In fact, the proposed scheme does not utilize the total number of time periods as an input parameter. The proposed scheme also exhibits good properties: the private key has a fixed size through the key update process, and the public key is kept unchanged. Furthermore, since our signature scheme is based on bilinear pairings, which can be constructed from Weil or Tate pairings on an elliptic curve, the scheme achieves efficiency in terms of the size in key and signature.

Organization We first introduced the background about key exposure problem and forward security concept in the digital signature paradigm. In Section 2, we will provide the mathematical treatment in bilinear pairings and definitions about the forward secure signature scheme. The details of our signature scheme are presented in Section 3 and its security analysis is discussed in Section 4. The complexity comparative with other schemes is discussed in Section 5. Finally, the conclusion and suggestion for future work are given.

2 Backgrounds

Like almost other forward secure signature schemes, the definitions of our forward secure signature scheme follow the formal definition from [1, 5].

2.1 Definitions

Forward Secure Signature Scheme. The basic idea of the forward secure signature scheme is to use the key evolution technique to update the private key periodically while keeping the public key unchanged. To do so, one can divide the lifetime of the signature scheme into a small period in which a different private key is used to sign messages. The Key Generation algorithm will initialize the lifetime of the signature scheme by creating the first period key pairs. However, the public key has to be fixed for whole lifetime of the signature scheme for convenience as well as keeping the Verification algorithm simple. The Signing

algorithm has to indicate time period in which the private key is used to produce signatures. In the forward secure signature scheme, there is an additional algorithm used to update private keys of the scheme. The Key Update algorithm takes the current private key as input and generates a new private key for the next period. Of course, after generating a new private key, the old private key must be erased immediately. Forward security is ensured by the fact that the Key Update algorithm is a kind of one-way functions, therefore given the current private key, it is hard to compute any previously used private key. The detailed definition of the forward secure signature scheme is given below:

Definition 1 (Key-evolving Signature Scheme). *A key-evolving digital signature scheme is a quadruple of algorithms,* $\mathsf{FSIG} = (\mathsf{FSIG.KeyGen}; \mathsf{FSIG.KeyUp}; \mathsf{FSIG.Sign}; \mathsf{FSIG.Verify})$*, where:*

- $\mathsf{FSIG.KeyGen}$*, the Key Generation algorithm, is a probabilistic algorithm which takes as input a security parameter* $k \in N$ *(given in unary as* 1^k*) and returns a pair* $(SK_0; PK)$*, the initial secret key and the public key;*
- $\mathsf{FSIG.Sign}$*, the (possibly probabilistic) Signing algorithm, takes as input the secret key* SK_i *of the current time period* i *and a message* M*, and returns a pair* $\langle i, \sigma \rangle$*, the signature of* M *for time period* i*;*
- $\mathsf{FSIG.KeyUp}$*, the (possibly probabilistic) Secret Key Update algorithm, takes the secret key for the current period* SK_i *as input and returns the new secret key* SK_{i+1} *for the next time period;*
- $\mathsf{FSIG.Verify}$*, the (deterministic) Verification algorithm, takes the public key* PK*, a message* M*, and a candidate signature* $\langle i, \sigma \rangle$ *as input, and returns* 1 *if* σ *is a valid signature of* M *or 0, otherwise. It is required that* $\mathsf{FSIG.Verify}_{PK}(M; \mathsf{FSIG.Sign}_{SK_i}(M)) = 1$ *for every message* M *and time period* i*.*

Security Analysis Using Random Oracle Model. We analyze our signature scheme in the random oracle model [6]. The security of the signature scheme means that it is computational infeasible for any adversary to forge a signature with respect to any of the previously used secret keys even if the exposure of the current secret key happens. We use the security model introduced by Bellare and Miner [5] with some modification.

In our model, besides knowing the user's public key PK, the adversary also gets to know the current time period. The adversary runs in three phases. In the first phase, the chosen message attack phase (cma), the adversary has access to a signing oracle, which it can query to obtain signatures of messages of its choice with respect to the current secret key. At the end of each time period, the adversary can choose whether to stay in the same phase or switch to the break-in phase ($\mathsf{breakin}$). In the break-in phase, which models the possibility of a key exposure, we give the adversary the secret key SK_j for the specific time period j it decided to break in. In the last phase, the forgery phase (forge), the adversary outputs a pair signature message, that is, a forgery. The adversary is considered to be successful if it forges a signature of some new message (that is, not previously queried to the signing oracle) for some time period prior to j. In

order to capture the notion of forward security of a key-evolving signature scheme $\mathsf{FSIG} = (\mathsf{FSIG.KeyGen}; \mathsf{FSIG.KeyUp}; \mathsf{FSIG.Sign}; \mathsf{FSIG.Verify})$ more formally, let $\mathcal{F}$ be an adversary for this scheme. To assess the success probability of $\mathcal{F}$ breaking the forward security of FSIG, consider the following experiment. Throughout this paper, $k, \ldots$ indicates that the arguments of the key generation algorithm could be more than k.

Experiment F-Forge-RO(FSIG, $\mathcal{F}$)

$\texttt{Select } H : \{0,1\}^* \rightarrow \{0,1\}^l \texttt{ at random}$
$(SK_0, PK) \stackrel{R}{\leftarrow} \mathsf{FSIG.KeyGen}^H(k, \ldots)$
$i \leftarrow 0$
$\texttt{Repeat}$
  $d \leftarrow \mathcal{F}^{H, \mathsf{FSIG.Sign}^H_{SK_i}(\bullet)}(\mathsf{cma}, PK);$
  $SK_{i+1} \leftarrow \mathsf{FSIG.KeyUp}^H(SK_i); i \leftarrow i+1$
$\texttt{Until } (d = \mathsf{breakin})$
$i \leftarrow i - 1$
$(M, \langle b, \sigma \rangle) \leftarrow \mathcal{F}^H(\mathsf{forge}, SK_i)$
$\texttt{If } \mathsf{FSIG.Verify}^H_{SK_i}(M, \langle b, \sigma \rangle) = 1 \texttt{ and } 0 \leq b \leq i-1$
  $\texttt{and } M \texttt{ was not queried of } \mathsf{FSIG.Sign}^H_{SK_i} \texttt{ in period } b$
  $\texttt{then return 1 else return 0}$

With the above forger, we can define the notion of security of the forward secure signature scheme in the random oracle model.

Definition 2 (Forward-security in the Random Oracle Model). *Let* $\mathsf{FSIG} = (\mathsf{FSIG.KeyGen}; \mathsf{FSIG.KeyUp}; \mathsf{FSIG.Sign}; \mathsf{FSIG.Verify})$ *be a key-evolving signature scheme, H be a random oracle and $\mathcal{F}$ be an adversary as described above. We let* $\mathbf{Succ}^{\mathrm{fwsig}}(\mathsf{FSIG}[k, \ldots]; \mathcal{F})$ *denote the probability that the experiment* F-Forge-RO($\mathsf{FSIG}[k, \ldots]; \mathcal{F}$) *returns* 1. *Then the insecurity of* FSIG *is the function*

$$\mathbf{InSec}^{\mathrm{fwsig}}(\mathsf{FSIG}[k, \ldots]; t; q_{\mathrm{sig}}; q_{\mathrm{hash}}) = \max_{\mathcal{F}}\{\mathbf{Succ}^{\mathrm{fwsig}}(\mathsf{FSIG}[k, \ldots]; \mathcal{F})\},$$

where the maximum here is taken over all adversaries $\mathcal{F}$ making a total of at most q_{sig} queries to the signing oracles across all the stages and for which the running time of the above experiment is at most t and at most q_{hash} queries are made to the random oracle H.

2.2 Bilinear Pairings

We summarize some concepts of bilinear pairings using similar notations used by Zhang and Kim [23] which was used to design ID-based blind signature and ring signature based on pairings.

Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be additive and multiplicative groups of the same prime order q, respectively. Let P is a generator of $\mathbb{G}_1$. Assume that the discrete logarithm problems in both $\mathbb{G}_1$ and $\mathbb{G}_2$ are hard. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a pairing which satisfies the following properties:

1. *Bilinear*: $e(aP, bP') = e(P, P')^{ab}$ for all $P, P' \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}$.
2. *Non-degenerate*: If $e(P, P') = 1 \; \forall \, P' \in \mathbb{G}_1$ then $P = \mathcal{O}$.
3. *Computable*: There is an efficient algorithm such as [4] to compute $e(P, P')$ for any $P, P' \in \mathbb{G}_1$.

To construct the bilinear pairing, we can use the Weil pairing or Tate pairing associated with supersingular elliptic curves. Under such group $\mathbb{G}_1$, we can define the following hard cryptographic problems:

- *Discrete Logarithm (DL) Problem:* Given $P, P' \in \mathbb{G}_1$, find an integer n such that $P = nP'$ whenever such integer exists.
- *Computational Diffie-Hellman (CDH) Problem:* Given a triple $(P, aP, bP) \in \mathbb{G}_1$ for $a, b \in \mathbb{Z}_q^*$, find the element abP.
- *Decision Diffie-Hellman (DDH) Problem:* Given a quadruple $(P, aP, bP, cP) \in \mathbb{G}_1$ for $a, b, c \in \mathbb{Z}_q^*$, decide whether $c = ab \pmod q$ or not.

A group, where the CDH problem is hard but the DDH problem is easy, is called Gap Diffie-Hellman (*GDH*) group. Details about GDH groups can be found in [7], [8], and [17].

For the sake of comparison, we assume that, as in [15], there is a parameter generator $\mathcal{IG}$ takes input k, and outputs $\mathbb{G}_1, \mathbb{G}_2$ of order q, and pairing e. The computational complexity of $\mathcal{IG}$ is $O(k^n)$. Also the computational complexity in groups $\mathbb{G}_1, \mathbb{G}_2$, and pairings e are at most $O(k^{n_1})$, $O(k^{n_2})$, and $O(k^e)$, respectively. We have $n, n_1, n_2, e \in N$ are order of the polynomial time algorithm.

The definition of the CDH assumption which is used for our security analysis as follows:

Definition 3 (CDH Assumption). *A probabilistic algorithm $\mathcal{A}$ is said to be (t, ϵ)-break-CDH in a cyclic group $\mathbb{G}$ if $\mathcal{A}$ runs at most time t, computes the Diffie-Hellman function $DH_{P,q}(aP, bP) = abP$, with input (P, q) and (aP, bP), with a probability of at least ϵ, where the probability is over the coins of $\mathcal{A}$ and (a, b) is chosen uniformly from $\mathbb{Z}_q \times \mathbb{Z}_q$. The group $\mathbb{G}$ is a (t, ϵ)-CDH group if no algorithm (t, ϵ)-break-CDH in this group.*

3 Our Scheme

As mentioned previously, our forward secure signature scheme FSIG consists of four algorithms. Our purpose in designing this scheme need not to define the *total number of time periods* in advance, hence we can have *unlimited* time periods forward signature scheme. In other words, the key evolution process can run forever.

Key Generation Algorithm The Key Generation algorithm takes a secure parameter k and returns the initial key pair $(SK_0; PK)$. Our Key Generation algorithm uses the same strategy like [15] in order to generate system parameters: groups $\mathbb{G}_1, \mathbb{G}_2$ of the same prime order q; a generator P of $\mathbb{G}_1$; a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Let H_1 be a collision-free hash function which converts an arbitrary string $\{0, 1\}^*$ into $\mathbb{Z}_q^*$. We also assume that there is a collision-free hash

function $H_2 : \{0,1\}^* \rightarrow \mathbb{G}_1$. This hash function can be considered as a part of the Key Generation algorithm.

$\mathsf{FSIG.KeyGen}(1^k)$

`Run` $\mathcal{IG}$ `to get groups` $\mathbb{G}_1, \mathbb{G}_2$ (`prime order` q), `bilinear map` e.

`Select random generator` $P \leftarrow \mathbb{G}_1$; $s, t, r_0 \overset{R}{\leftarrow} \mathbb{Z}_q^*$.

`Compute:` $Q = sP, T = tP$

`Set` $PK \leftarrow (\mathbb{G}_1, \mathbb{G}_2, e, P, Q, T)$

`Compute:`

$s_0 = s + r_0 H_1(0); t_0 = t - r_0 H_1(0)$

$Q_0 = r_0 H_1(0) P;$

$V_0 = t_0 Q_0;$

`Erase` s, t, r_0, t_0

`Set` $SK_0 \leftarrow (s_0, V_0, Q_0)$

`Output` $(SK_0; PK)$

Key Update Algorithm The Key Update algorithm is the core part of the key evolution scheme. It refreshes the private key of the current time period to the new value corresponding to the new time period, then erases the previous private key. The Key Update algorithm is given below:

$\mathsf{FSIG.KeyUp}(i, SK_{i-1})$

`Pick a random element` $r_i \in \mathbb{Z}_q^*$;

`Parse` SK_{i-1} `as` $(s_{i-1}, V_{i-1}, Q_{i-1})$

`Compute:`

$s_i = s_{i-1} + r_i H_1(i);$

$Q_i = Q_{i-1} + r_i H_1(i) P;$

$V_i = V_{i-1} + r_i H_1(i) \left(T - Q_{i-1} - Q_i\right);$

`Erase` $s_{i-1}, r_i, V_{i-1}, Q_{i-1}$

`Set` $SK_i \leftarrow (s_i, V_i, Q_i)$

`Output` SK_i

Signing Algorithm The Signing algorithm produces a signature at the current time period using the private key of the considered time period.

$\mathsf{FSIG.Sign}(i, M, SK_i)$:

`Parse` SK_i `as` (s_i, V_i, Q_i)

`Set` $U = Q_i$;

`Compute` $\alpha = s_i Q_i + V_i$; and $\beta = s_i H_2(i, M, U)$;

`Set` $\sigma = (U, \alpha, \beta)$

`Output signature for` M `as` $\langle i, \sigma \rangle$

Verification Algorithm The Verification algorithm tests if a given signature on a message at a specific time period is valid or not. The output of the test is 1 if the signature is valid and 0 otherwise.

$\mathsf{FSIG.Verify}_{PK}(i, M, \sigma)$:

`Parse` σ `as` (U, α, β) `Verify`

$$e(\alpha, P) \overset{?}{=} e(U, T + Q) \tag{1}$$

$$e(\beta, P) \overset{?}{=} e(H_2(i, M, U), U + Q) \tag{2}$$

`Output 1 if Eqs (1) and (2) are correct, otherwise output 0.`

The first equation (1) verifies the value of U and the second one (2) verifies the signature on the message M. The time period value is embedded in the signature too.

Correctness The correctness of the proposed signature scheme comes from the correctness of Eqs (1) and (2). The correctness of Eq (1) is shown bellow:

$$
\begin{aligned}
&e(\alpha, P) = e\left(s_i Q_i + V_i, P\right) = e\left(\left[s_{i-1} + r_i H_1(i)\right] Q_i + V_i, P\right) \\
&= e\left(\left[s + \sum_0^i r_k H_1(k)\right] Q_i + V_i, P\right) = e\left(sQ_i + \sum_0^i r_k H_1(k) Q_i + V_i, P\right) \\
&= e\left(sQ_i + \sum_0^i r_k H_1(k) Q_i + V_{i-1} + r_i H_1(i)\left[T - Q_{i-1} - Q_i\right], P\right) \\
&= e\left(sQ_i + \sum_0^{i-1} r_k H_1(k) Q_i + V_{i-1} + r_i H_1(i)\left[T - Q_{i-1}\right], P\right) \\
&= e\left(sQ_i + \sum_0^{i-1} r_k H_1(k)\left[Q_{i-1} + r_i H_1(i) P\right] + V_{i-1} + r_i H_1(i)\left[T - Q_{i-1}\right], P\right) \\
&= e\left(sQ_i + \sum_0^{i-1} r_k H_1(k) Q_{i-1} + r_i H_1(i) \sum_0^{i-1} r_k H_1(k) P + V_{i-1} + r_i H_1(i)\left[T - Q_{i-1}\right], P\right) \\
&= e\left(sQ_i + \sum_0^{i-1} r_k H_1(k) Q_{i-1} + V_{i-1} + r_i H_1(i) T, P\right) \\
&\vdots \\
&= e\left(sQ_i + r_0 H_1(0) Q_0 + V_0 + r_1 H_1(1) T + ... + r_i H_1(i) T, P\right) \\
&= e\left(sQ_i + r_0 H_1(0) Q_0 + t_0 Q_0 + r_1 H_1(1) T + ... + r_i H_1(i) T, P\right) \\
&= e\left(sQ_i + tQ_0 + r_1 H_1(1) T + ... + r_i H_1(i) T, P\right) \\
&= e\left(sQ_i + r_0 H_1(0) T + r_1 H_1(1) T + ... + r_i H_1(i) T, P\right) = e\left(sQ_i + \sum_0^i r_k H_1(k) T, P\right) \\
&= e\left(sQ_i + t \sum_0^i r_k H_1(k) P, P\right) \\
&= e\left(\left[s + t\right] Q_i, P\right) = e\left(Q_i, P\right)^{s+t} = e\left(Q_i, T + Q\right)
\end{aligned}
$$

The correctness of this equation ensures that the value of U is correct. The validity of the signature is guaranteed by the correctness of Eq (2).

$$
\begin{aligned}
&e(\beta, P) = e(s_i H_2(i, M, U), P) = e\left(\left[s_{i-1} + r_i H_1(i)\right] H_2(i, M, U), P\right) \\
&\vdots \\
&= e\left(\left[s + \sum_{k=0}^i r_k H_1(k)\right] H_2(i, M, U), P\right) \\
&= e\left(sH_2(i, M, U), P\right) e\left(H_2(i, M, U), P\right)^{\sum_{k=0}^i r_k H_1(k)} \\
&= e\left(H_2(i, M, U), Q\right) e\left(H_2(i, M, U), \sum_{k=0}^i H_1(k) r_k P\right) \\
&= e\left(H_2(i, M, U), Q\right) e\left(H_2(i, M, U), Q_i\right) \\
&= e\left(H_2(i, M, U), Q\right) e\left(H_2(i, M, U), U\right) = e\left(H_2(i, M, U), Q + U\right)
\end{aligned}
$$

Efficiency Our proposed forward secure signature scheme exhibits new properties. Firstly, our scheme does not have the total number of time periods parameter. The Key Update algorithm can perform infinite and stop only when a private key in a certain time period is compromised. At that time, we need to run the Key Generation algorithm again to initialize a new key pair.

Secondly, the key pair produced by the Key Generation algorithm has a fixed length. The sizes of the private key and public key do not grow after running the Key Update algorithm. In addition, the public key remains unchanged since after produced once at the first time by the Key Generation algorithm.

The Signing and Verifying algorithms also require the fixed amount of computational time. The signing algorithm of our scheme is very unique. For a certain time period, one can compute value of α once, then stores it for future signature issuing. From the next time, the signature issuing algorithm is just to compute one point multiplication over an elliptic curve.

4 Security Analysis

We analyze the security of our forward secure signature scheme used technique like in [1, 5, 15]. In addition, we assume that, partial key exposure also leads to key exposure problem. This is obviously since we may derive the remained part from exposed part of the private key. The following theorem shows the security of our scheme.

Theorem 1. *If there exists a forger $\mathcal{F}$ that runs in time at most t, asking at most q_{hash} hash queries and q_{sig} signing queries, such that* $\mathbf{Succ}^{\text{fwsig}}(\mathsf{FSIG}[k,\ldots]; \mathcal{F}) > \epsilon$ *then there exists a adversary $\mathcal{A}$ that (t', ϵ')-break CDH in group $\mathbb{G}_1$ where:*

$$t' = t + O\left(k^{n_1}\right);\ \text{and}\ \epsilon' = \left(1 - \frac{1}{q_{\text{sig}} + 1}\right)^{q_{\text{sig}}+1} \cdot \frac{1}{q_{\text{sig}}\left(q_{\text{hash}} + q_{\text{sig}} + 1\right)} \cdot \epsilon$$

Proof (Sketch). To break CDH problem in the additive group $\mathbb{G}_1$ of the order q, an adversary $\mathcal{A}$ is given P (a random generator of $\mathbb{G}_1$), $P' = aP$, $Q' = bP$, where are randomly chosen and remain unknown to $\mathcal{A}$. The task of $\mathcal{A}$ is to derive $S' = abP$ with the help of the forger $\mathcal{F}$. $\mathcal{A}$ provides the public key to $\mathcal{F}$ and answers its hash queries, signing queries, and breakin query. First, $\mathcal{A}$ guesses a random i at which $\mathcal{F}$ will ask for the breakin query. Then $\mathcal{A}$ set the public key $PK = (\mathbb{G}_1, \mathbb{G}_2, e, P, Q, T)$, where $Q = Q'$. $\mathcal{A}$ provides PK to $\mathcal{F}$ and runs it. $\mathcal{A}$ can answer the hash queries and the signing queries since it controls the hash oracle. During execution, $\mathcal{A}$ guesses a random index g', and hopes the forgery will base on g'-th hash query. $\mathcal{A}$ makes this hash value special, *i.e.*, P'. Suppose $\mathcal{F}$ outputs a signature on message $M_{g'}$ for time period $i' < i$. From this signature $\mathcal{A}$ can derive $S' = abP$, hence solves CDH problem. The detailed proof is given in Appendix.

Theorem 2. *Let* $\mathsf{FSIG}[k;\ldots]$ *represent our key-evolving signature scheme with modulus size k. Then for any t, q_{hash} and q_{sig},*

$$\mathbf{InSec}^{\text{fwsig}}(\mathsf{FSIG}[k;\ldots]; t; q_{\text{sig}}; q_{\text{hash}}) \leq$$

$$q_{\text{sig}}(q_{\text{hash}} + q_{\text{sig}} + 1)\left(1 - \frac{1}{q_{\text{sig}} + 1}\right)^{-(q_{\text{sig}+1})} \mathbf{InSec}^{\text{cdh}}(k, t')$$

where $t' = t + O(k^{n_1})$

Proof. From Definition 2 and Theorem 1, the insecurity function is computed simply by solving function in Theorem 1 and express ϵ' in terms of ϵ we have:

$$\epsilon' = \left(1 - \frac{1}{q_{\text{sig}} + 1}\right)^{q_{\text{sig}+1}} \cdot \frac{1}{q_{\text{sig}}\left(q_{\text{hash}} + q_{\text{sig}} + 1\right)} \cdot \epsilon$$

$$\Longrightarrow q_{\text{sig}}\left(q_{\text{hash}} + q_{\text{sig}} + 1\right)\epsilon' \Big/ \left(1 - \frac{1}{q_{\text{sig}} + 1}\right)^{q_{\text{sig}+1}} = \epsilon$$

This completes the proof of Theorem 2. □

5 Evaluation

In this section, we compare our proposed signature scheme with the previous signature scheme [15] which has the same computational assumption. Computational complexity, the sizes of keys and signature are examined.

Table 1. Computational complexity of our signature scheme

Algorithm	Ours	Hu *et al.*[15]
Key generation	$O\left(k^n + k^{n_1}\right)$	$O\left(k^n + k^{n_1} + k^{n_1} \log T\right)$
Signing	$O\left(k^{n_1}\right)$	$O\left(k^{n_1}\right)$
Verification	$O\left(k^e + k^{n_1}\right)$	$O\left(k^e \log T + k^{n_1} \log T\right)$
Key Update	$O\left(k^{n_1}\right)$	$O\left(k^{n_1}\right)$
Public key size	$O\left(k\right)$	$O(k)$
Private key size	$O\left(k\right)$	$O\left(k \log T + k\right)$
Signature size	$O\left(k\right)$	$O\left(k \log T + k\right)$

Table 2. Computational complexity of other schemes

Algorithm	BM[5]	AR[1]	IR[16]	*MMM*[20]
Key generation	lk^2T	lk^2T	$k^5 + (k + l^3)lT$	k^2l^2
Signing	$(T + l)k^2$	lk^2T	k^2l	k^2l
Verification	$(T + l)k^2$	k^2l	k^2l	$k^2l + l^2 \log l$
Key Update	lk^2	lk^2	$(k^2 + l^3)lT$	$k^2l + (k + l^2) \log t$
Public key size	lk	k	k	$k + l \log l$
Private key size	lk	k	k	l
Signature size	k	k	k	$k + l \log l$

In Table 1, T is total number of time periods and in Table 2, l is a security parameter of conventional cryptographic operation as explained in [20].

As we can see from Tables 1 and 2, our signature scheme is very efficient in terms of computation as well as performance. The signature and key sizes do not depend on the total number of time periods. Moreover, comparing with the schemes [1, 5, 16], the signature size is shorter for the same security level since our scheme is operating over an elliptic curve (so in the security parameter k is different). The signing algorithm will be similar to that of [8] if we store the fixed part in the signature for later use. Although *MMM* scheme has unbounded time periods, it still depends on the current time period parameter in the key update algorithm.

The verification of the signature just requires four pairing operations. This can be considered to be the same as that of [8]. For verifying multiple signatures of the same time period, the result of verification equation (1) can be saved for later use. In this case, the verifying process remains just two pairing computations. Although pairing computation is expensive, there are many improvements in implementation of the pairings as in [4, 12]. Utilizing those good implementations, our scheme can be efficient in performance. Considering above features, our signature scheme can be applied in the application where storage and computation power are limited like mobile devices. Forward secure properties will strengthen the security of the applications.

6 Concluding Remarks

We have proposed yet another forward signature scheme using bilinear pairings. Our signature scheme has a specific property, namely unlimited time periods. Under the assumption of the hardness of Computational Diffie-Hellman problem, we have presented the security proof of the signature scheme in the random oracle model. Moreover, the proposed signature scheme is very efficient in terms the signature size as well as performance compared to the previous schemes. The scheme's public and private key sizes are unchanged through the key evolving processes. With a good pairing computation algorithm, we can have an efficient signature verifying algorithm.

For further work, we consider integration of our scheme with other cryptographic techniques to have new applications.

Acknowledgements

The authors would like to express great thank to the anonymous reviewers for useful comments.

References

1. M. Abdalla and L. Reyzin, "A New Forward-Secure Digital Signature Scheme," *Advances in Cryptology* – ASIACRYPT 2000, LNCS 1976, pp. 116–129, Springer-Verlag, Dec. 2000.

2. M. Abdalla, S. Miner, and C. Namprempre, "Forward-Secure Threshold Signature Schemes," *Topics in Cryptology* – CT-RSA 2001, LNCS 2020, pp. 441–456, Springer-Verlag, 2001.
3. R. Anderson. "Two Remarks on Public-Key Cryptology From Invited Lecture," *Fourth ACM onference on Computer and Communications Security*, April, 1997. http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-549.pdf
4. P.S.L.M. Barreto, H.Y. Kim, B. Lynn and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems", *Advances in Cryptology – Crypto'2002*, LNCS 2442, pp. 354-369, Springer-Verlag, 2002.
5. M. Bellare and S. K. Miner, "A Forward-Secure Digital Signature Scheme," *Advances in Cryptology* – CRYPTO '99, LNCS 1666, pp. 431–448, Springer-Verlag, Aug. 1999.
6. M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", ACM Conference on Computer and Communications Security, pp. 62–73, 1993.
7. D. Boneh and M. Franklin, "ID-based Encryption from the Weil Pairing," *Advances in Cryptology* - Crypto'2001, LNCS 2139, pp. 213–229, Springer-Verlag, 2001.
8. D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing", *Advances in Cryptology – Asiacrypt'2001*, LNCS 2248, pp. 514–532, Springer-Verlag, 2001.
9. R. Canetti, S. Halevi, and J. Katz, "A Forward-Secure Public-Key Encryption Scheme," *Advances in Cryptology* – EUROCRYPT'03, LNCS 2656, pp. 255–271, Springer-Verlag 2003.
10. Dang Nguyen Duc, Jung Hee Cheon, and Kwangjo Kim, "A Forward-Secure Blind Signature Scheme Based on the Strong RSA Assumption," *Information and Communications Security* – ICICS'03, LNCS 2836, pp. 11–21. Springer-Verlag, 2003.
11. Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung, "Intrusion Resilient Public-Key Encryption," *Topics in Cryptology* – CT-RSA 2003, LNCS 2612, pp. 19-32, Springer-Verlag, 2003.
12. S. Galbraith, K. Harrison and D. Soldera, "Implementing the Tate Pairing," *Algorithm Number Theory Symposium* - ANTS V, LNCS 2369, pp. 324–337, Springer-Verlag, 2003.
13. C. Gentry and A. Silverberg, "Hierarchical ID-Based Cryptography," *Advances in Cryptology* – ASIACRYPT 2002, LNCS 2501, Y. Zheng ed., pp. 548–566, Springer-Verlag, 2002.
14. S. Haber and W. Stornetta, "How to Time-Stamp a Digital Document", *Advances in Cryptology* – CRYPTO 90, LNCS 537, A. J. Menezes and S. Vanstone, ed., Springer-Verlag, 1990.
15. F. Hu, C. Wu, and J.D. Irwin, "A New Forward Secure Signature Scheme using Bilinear Maps," http://eprint.iacr.org/2003/188.pdf.
16. G. Itkis and L. Reyzin. "Forward-secure signatures with optimal signing and verifying," Advances in Cryptology – CRYPTO'01, LNCS 2139, pp. 332-354. Springer-Verlag, 2001.
17. A. Joux and K. Nguyen, "Separating Decision Diffie-Hellman from Diffie-Hellman in Cryptographic Groups," *Cryptology ePrint Archive – 2001/03.*
18. A. Kozlov and L. Reyzin. "Forward-Secure Signatures with Fast Key Update," Proc. of the 3rd International Conference on Security in Communication Networks – SCN'02, 2002.
19. H. Krawczyk. "Simple Forward-Secure Signatures from Any Signature Scheme," Proc. of Seventh ACM Conference on Computer and Communications Security, pp. 108–115, Nov. 2000.

20. T. Maklin, D. Micciancio and S. Miner, "Efficient Generic Forward-Secure Signatures with an Unbounded Number of Time Periods," *Advances in Cryptology* – EUROCRYPT 2002, LNCS 2332, L. Knudsen ed., pp. 400–417, Springer-Verlag, 2002.
21. D. X. Song. "Practical Forward Secure Group Signature Schemes," Proc. of the 8th ACM Conference on Computer and Communications Security – CCS 2001, pp. 225-234. ACM, 2001.
22. W. Tzeng and Z. Tzeng, "Robust Forward-Secure Digital Signature with Proactive Security," *Public Key Cryptography* – PKC'01, LNCS 1992, K. Kim ed., pp. 264–276, Springer-Verlag, 2001.
23. F. Zhang and K. Kim, "ID-Based Blind Signature and Ring Signature from Pairings", *Advances in Cryptology – Asiacrypt'2002*, LNCS 2501, Springer-Verlag, pp. 533–547, 2002.

A Proof of Theorem 1

Proof. Using the same technique in [15], we describe the details procedure of $\mathcal{A}$ as follows. As in [1], first we assume that if $\mathcal{F}$ outputs a forgery of $\langle i, \sigma \rangle$ for message M, then the hash oracle has been queried on (i, M, Q_i). Any adversary can be modified to do that. Because of this, the number of hash queries may increase to $q_{\text{hash}} + 1$. We also assume that if $\mathcal{F}$ asks for the signing query for some message M in some time period i, then the hash query on (i, M, Q_i) must also be request simultaneously. Any adversary can be modified to do so and therefore, the number of hash queries may increase to $q_{\text{hash}} + q_{\text{sig}} + 1$. Assume that $\mathcal{F}$ maintains all necessary bookkeeping and does not ask for the same hash query twice. Note that, the number of hash queries on the hash oracle H_1 can be included in q_{hash} without any effect since it only happens in Key Update procedure.

First of all, $\mathcal{A}$ has to guess the time period i at which $\mathcal{F}$ will ask for the breakin query. It randomly selects $i > 0$, hoping that the breakin query will occur at this time period. $\mathcal{A}$ then generates the public key $PK \leftarrow (\mathbb{G}_1, \mathbb{G}_2, e, P, Q, T)$ but $\mathcal{A}$ sets $Q = Q' = bP$ directly. $\mathcal{A}$ also randomly picks $k \in \mathbb{Z}_q^*$ then sets $T = kP - Q$. At this moment, $s = b$ is unknown to both $\mathcal{A}$ and $\mathcal{F}$. $\mathcal{A}$ gives the public key to forger $\mathcal{F}$, and runs until there is a breakin query.

To answer the hash query and the signing query of $\mathcal{F}$, $\mathcal{A}$ maintains two tables: a signature query table and a hash query table.

Signing queries are answered at random since $\mathcal{A}$ controls the hash oracle. In order to answer a signature query number n on a message M'_n during time period $j'_n < i$ (breakin happens), $\mathcal{A}$ selects randomly: $x_{j'_n}, y_{j'_n} \in_R \mathbb{Z}_q^*$. $\mathcal{A}$ then selects randomly $s_{j'_n} \in_R \mathbb{Z}_q^*$ and computes $U'_n = y_{j'_n} P$; $\alpha'_n = y_{j'_n}(Q + T)$; $\beta'_n = x_{j'_n}(U'_n + Q)$; $V'_n = \alpha'_n - s_{j'_n} P$, $h'_n = x_{j'_n} P$. $\mathcal{A}$ checks in its signature query table to see if a signature query on M'_n during time period j'_n has already been asked, and if there is, $j'_n, U'_n, \alpha'_n, \beta'_n$ used in answering. If not, $\mathcal{A}$ answers the query as $j'_n, U'_n, \alpha'_n, \beta'_n$ and also records the signature query entry as $\left(n, j'_n, x_{j'_n}, y_{j'_n}, s_{j'_n}, U'_n, \alpha'_n, \beta'_n, V'_n, h'_n, M'_n\right)$. This setting satisfies the Verify

algorithm:

$$\begin{aligned} e\left(\alpha'_n, P\right) &= e\left(y_{j'_n}(Q+T), P\right) = e\left(y_{j'_n}(s+t)P, P\right) \\ &= e\left(y_{j'_n}P, P\right)^{s+t} = e\left(y_{j'_n}P, (s+t)P\right) \\ &= e\left(U'_n, T+Q\right) \\ e\left(\beta'_n, P\right) &= e\left(x_{j'_n}(U'_n+Q), P\right) = e\left(x_{j'_n}(y_{j'_n}+s)P, P\right) \\ &= e\left(x_{j'_n}P, P\right)^{y_{j'_n}+s} = e\left(x_{j'_n}P, (y_{j'_n}+s)P\right) \\ &= e\left(h'_n, U'_n+Q\right) \end{aligned}$$

For a signing query n on a message M'_n during time period $j'_n \geq i$, $\mathcal{A}$ picks randomly $s_{j'_n}, x_{j'_n} \in_R \mathbb{Z}_q^*$ and computes $U'_n = s_{j'_n}P - Q$; $\alpha'_n = kU'_n$; $\beta'_n = s_{j'_n}x_{j'_n}P$; $V'_n = \alpha'_n - s_{j'_n}U'_n$, $h'_n = x_{j'_n}P$. Again, $\mathcal{A}$ also checks in its signature query table to see if a signature query on M'_n during time period j'_n has already been asked, and if there is, $j'_n, U'_n, \alpha'_n, \beta'_n$ used in answering. If not, $\mathcal{A}$ answers the query as $j'_n, U'_n, \alpha'_n, \beta'_n$ and also records the signature query entry as $\left(n, j'_n, x_{j'_n}, y_{j'_n}, s_{j'_n}, U'_n, \alpha'_n, \beta'_n, V'_n, h'_n, M'_n\right)$. The $y_{j'_n}$ can be set to 0 in this case. This setting also satisfies the Verify algorithm:

$$\begin{aligned} e\left(\alpha'_n, P\right) &= e\left(kU'_n, P\right) = e\left(U'_n, kP\right) \\ &= e\left(U'_n, Q+T\right) \\ e\left(\beta'_n, P\right) &= e\left(x_{j'_n}s_{j'_n}P, P\right) = e\left(x_{j'_n}P, s_{j'_n}P\right) \\ &= e\left(h'_n, U'_n+Q\right) \end{aligned}$$

The triple $(s_{j'_n}, U'_n, V'_n)$ also is valid if $\mathcal{F}$ checks it in this breakin phase.

Hash queries are answered at random. To answer the t-th hashing query for (j_t, M_t, U_t), $\mathcal{A}$ first checks the signature query table to see if there is an entry $\left(n, j'_n, x_{j'_n}, y_{j'_n}, s_{j'_n}, U'_n, \alpha'_n, \beta'_n, V'_n, h'_n, M'_n\right)$ such that $(j_t, M_t, U_t) = (j'_n, M'_n, U'_n)$. If so, it just outputs h'_n. Otherwise, $\mathcal{A}$ picks randomly $x_{j_t} \in_R \mathbb{Z}_q^*$, and set the output to $h_t = x_{j_t}P$. It also records value $(t, j_t, x_{j_t}, U_t, h_t, M_t)$. During execution, $\mathcal{A}$ has to guess a random index g'-th, with hope that forgery will happen. $\mathcal{A}$ sets it as special value $P' = aP$.

At the breakin occurs in the time period i, $\mathcal{A}$ simply outputs the secret key SK_i, which is an entry in the signing query table. The validity of SK_i is easy to check. If breakin occurs not in time period i, $\mathcal{A}$ will abort.

Suppose $\mathcal{A}$'s guesses for the time period breakin and the hash index are correct, and $\mathcal{F}$ outputs a forgery $\langle i', \sigma' \rangle$ on a message $M_{g'}$, where $\sigma' = (U', \alpha', \beta')$, and $i' < i$. If the verification holds, $\mathcal{A}$ can derive $S' = abP$ as follows:

We have:

$$\begin{aligned} e(\beta', P) &= e(H_2(i', M_{g'}, U'), U' + Q) \\ &= e(H_2(i', M_{g'}, U'), U')e(H_2(i', M_{g'}, U'), Q) \end{aligned}$$

$$\Rightarrow \frac{e(\beta', P)}{e(H_2(i', M_{g'}, U'), U')} = e(H_2(i', M_{g'}, U'), Q)$$

$\mathcal{A}$ controls the hash oracle and the forger $\mathcal{F}$ does not have ability to alter or verify the hash oracle. We may assume that U' equals to the one in time period $i' < i$, in which $\mathcal{A}$ has queried for signatures and $\mathcal{A}$ has value $U' = y'P$ in that time period, otherwise $\mathcal{A}$ fails. Then we have:

$$\Rightarrow \frac{e(\beta', P)}{e(H_2(i', M_{g'}, U'), y'P)} = e(H_2(i', M, U'), Q)$$
$$\Rightarrow \frac{e(\beta', P)}{e(y^{-1}H_2(i', M_{g'}, U'), P)} = e(H_2(i', M, U'), Q)$$
$$\Rightarrow \frac{e(\beta', P)}{e(y'^{-1}H_2(i', M_{g'}, U'), P)} = e(H_2(i', M, U'), Q)$$
$$\Rightarrow e(\beta' - y'^{-1}P', P) = e(P', Q) = e(aP, bP) = e(abP, P)$$

Therefore we can set $S' = abP = \beta' - y^{-1}P'$. We can have this since non-degenerate property of the bilinear pairings.

Run time. Suppose that bit operations in $\mathbb{G}_1$ is at most $O(k^{n_1})$ as in [15], to run $\mathcal{F}$, $\mathcal{A}$ needs to perform some key generation and some group operations. Therefore, we have the time running between $\mathcal{A}$ and $\mathcal{F}$ is different: $t' = t + O(k^{n_1})$

Probability. First, we can see that, $\mathcal{A}$ always acts as a real signer to the $\mathcal{F}$ from $\mathcal{F}$'s point of view except one case when $\mathcal{A}$ answers the hash query with other value. There are totally two guesses performed by $\mathcal{A}$.

The probability that $\mathcal{A}$ guesses the correct time period $\mathcal{F}$ sends the breakin query after q_{sig} signing queries is calculated as follows. Call the event in which breakin occurs E_b and ω is a probability constant which E_b depends on. ω will distribute over $\{0, 1\}$, where 1 is drawn with probability ω and 0 with probability $1-\omega$. We need to calculate the probability of E_b in a certain time period. Suppose that at each time period $1, 2, \ldots i$, the number of signatures has been queried is $q_1, q_2, \ldots q_i$, respectively. If E_b happens at time period i, probability of such event is calculated as $\Pr = (1-\omega)^{q_1}(1-\omega)^{q_2}\ldots(1-\omega)^{q_i}\omega$ Notice that, at the breakin period, there are total q_{sig} queries have been done, so $q_{\text{sig}} = \sum_{k=1}^{i} q_k$. And the probability of guessing break-in at time period i correctly will be $\Pr = (1-\omega)^{q_{\text{sig}}}\omega$. This value is maximized at $\omega = 1/(q_{\text{sig}} + 1)$ and we have

$$\Pr = \left(1 - \frac{1}{q_{\text{sig}} + 1}\right)^{q_{\text{sig}}} \cdot \frac{1}{q_{\text{sig}} + 1} = \frac{1}{q_{\text{sig}}} \cdot \left(1 - \frac{1}{q_{\text{sig}} + 1}\right)^{q_{\text{sig}}+1}$$

The probability to guess the correct hash query on which the forgery is based is $\Pr \geq 1/(1 + q_{\text{sig}} + q_{\text{hash}})$. Therefore the probability of $\mathcal{A}$'s success in deriving $S' = abP$ is at least:

$$\epsilon' = \frac{1}{q_{\text{sig}}} \cdot \left(1 - \frac{1}{q_{\text{sig}} + 1}\right)^{q_{\text{sig}}+1} \cdot \frac{1}{q_{hash} + q_{sig} + 1} \cdot \epsilon$$

where ϵ is the minimum probability with which $\mathcal{F}$ to successfully forge a signature. □