Bilateral-secure Signature by Key Evolving

Tao Xiang
College of Computer Science
Chongqing University
Chongqing 400044, China
txiang@cqu.edu.cn

Xiaoguo Li
College of Computer Science
Chongqing University
Chongqing 400044, China
xguoli@cqu.edu.cn

Fei Chen*
College of Computer Science
and Engineering
Shenzhen University
Shenzhen 518060, China
fchen@szu.edu.cn

Yi Mu
School of Computing and
Information Technology
University of Wollongong
Wollongong, NSW 2522,
Australia
ymu@uow.edu.au

ABSTRACT

In practice, the greatest threat against the security of a digital signature scheme is the exposure of signing key, since the forward security of past signatures and the backward security of future signatures could be compromised. There are some attempts in the literature, addressing forward-secure signature for preventing forgeries of signatures in the past time; however, few studies addressed the backward-security of signatures, which prevents forgeries in the future time. In this paper, we introduce the concept of key-evolving signature with bilateral security, i.e., both forward security and backward security. We first define the bilateral security formally for preventing the adversaries from forging a valid signature of the past and the future time periods in the case of key exposure. We then provide a novel construction based on hub-and-spoke updating structure and the random oracle model, and show that the construction achieves bilateral security and unbounded number of time periods. Finally, we compare our scheme with the existing work by rigorous analysis and experimental evaluation, and demonstrate that our construction is more secure and efficient for practical applications.

Keywords

Digital Signature; Bilateral Security; Key Evolving; Forward Security; Backward Security

*The corresponding author is Fei Chen (fchen@szu.edu.cn).

ASIA CCS '16, May 30-June 03, 2016, Xi'an, China
© 2016 ACM. ISBN 978-1-4503-4233-9/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2897845.2897864

1. INTRODUCTION

Digital signature is one of the most fundamental and useful components in modern cryptography. It proves the authenticity of a message by providing a way for users to sign the message so that the signature can be verified. *Key exposure* is a major threat in practice for a digital signature scheme because the exposure of signing key typically implies that all security guarantees are lost [5]. It not only compromises the security and validity of any signature issued after the exposure, but also compromises all past signatures [21, 6], namely the attacker can always construct a valid signature once the key exposure happens.

Many security properties are proposed or considered in the design of signature schemes to solve the key exposure problem, including proactive security [11, 17], key-insulated security [12, 13], intrusion-resilient[19], and forward security [4, 5, 21]. However, many of these solutions are costly in various ways and are unsuitable for practical deployment. A simpler and more practical approach to solve the problem was suggested by Anderson [4] who suggested to update the key periodically such that attackers could not forge a signature for past time periods. Such a signature scheme against key exposure is referred to as *forward-secure signature*.

Forward-secure signature, as the most practical way to solve the key exposure problem, divides the lifetime of a signature scheme into discrete periods (e.g. days, weeks, years) and updates the key at each new time period. The recipient of the signature can verify two aspects: the correctness and the correspondence to a particular period time [21]. By this way, the attacker cannot forge a valid signature of previous time periods after key exposure. The forward-secure signature was firstly formalized in [5], in which two constructions were proposed. The first one is a generic method with logarithmic complexity in the number of time periods; the second scheme is based on Fiat-Shamir signature and has constant-size signatures, but it has a linear cost in signature generation and verification. In [18], the authors proposed a scheme with highly efficient signing and verification based on the work in [15], however their basic technique requires an expensive update. In [26], the authors constructed generic forward-secure signatures with an unbounded number of time periods. In [21], an extremely simple construction of forward-secure signatures based on any regular signature scheme was presented.

Backward security is another important security property. It would be desirable for a digital signature scheme to capture this property, as it can prevent attackers from forging a valid signature of *future* time periods after the key exposure. However, backward security has only been scarcely studied; for example, none of the previously-mentioned forward-secure signature schemes is backward-secure. In [8], the notion of forward-secure signature with untrusted update was proposed, which meets the backward security in some extent. However, the lifetime of their signature system is bounded by a predefined parameter. Similar work can be found in [22] and [14], but they either suffer from attacks or need substantial improvement on performance.

In this paper, we introduce *bilateral security*, a new security property that satisfies forward security and backward security at the same time, to address the key-exposure problem of digital signatures. Based on the CDH assumption, we propose a novel bilateral-secure signature scheme using bilinear map. In our construction, we employ a hub-and-spoke structure to evolve the key. Specifically, the signer generates a root key at the beginning of the key generation. Then he chooses two secrets that satisfy a specified relation with the root key, and stores the two secrets separately. After updating the key at each time period, the specified relation always holds. In the verify phase, the verifier not only checks the correctness of the signature but also validate whether the specified relation holds.

1.1 Contributions

Our contributions are summarized as follows:

- We introduce the concept of key-evolving signature with bilateral security (i.e. forward security and backward security) that allows to update the secret keys after each time period. We also define the bilateral security of a key-evolving signature scheme, which prevents attackers from forging a valid signature of the past and future time periods when the key is exposed.
- We propose a novel forward secure and backward secure signature scheme (FBSS), the scheme dismisses the threat of key expose problem. Furthermore, to the best of our knowledge, our work is the first to achieve unbounded number of time periods by a specific construction with hub-and-spoke updating structure for prolonging the lifetime of signature system.
- Finally, we prove that our scheme achieves the bilateral-security under the random oracle model and the CDH assumption. We compare our work with existing forward-secure schemes with untrusted update (FSSUU). Experimental results of key generation, key update, message signing and verifying show that our scheme is more efficient than existing FSSUU schemes.

1.2 Organization

The organization of this paper is as follows. We begin by discussing related work in Section 2. Next, we give the preliminaries and define the secure model of our work in Section 3. Then we present a novel construction in Section 4 and show the correctness analysis and security proof in Section 5. Section 6 describes our experimental results. Finally we conclude this paper in Section 7.

2. RELATED WORK

We briefly review forward security and backward security schemes in this section.

2.1 Forward Security

Forward secure protocol design is an important approach to the key exposure problem. Originally, forward security was introduced for key exchange protocols [16]. The notion forward secure signature was first formalized by Bellare and Minner [5], building on the earlier ideas of Anderson [4]. Bellare and Miner in [5] further proposed practical schemes and formalized the definitions of forward-secure signature. Subsequently, a large number of papers about forward security have been published. The existing work can be roughly classified into two categories.

One category is generic constructions that do not necessarily require random oracles [4, 5, 21, 26, 10]. The first example is the tree construction in [5], which builds a binary tree from chains of certificates where leaves correspond to time periods. In [26], the authors constructed generic forward-secure signatures with an unbounded number of time periods. In [21], an extremely simple construction of forward-secure signature based on any regular signature scheme is presented. In [10], an evaluation of the practical performance of these schemes is provided and the authors also build an open-source forward-secure signature library.

The other category is the specific schemes based on random oracle [5, 3, 18, 20]. In [5], for the first time, the authors achieve short signatures with fast key update based on the Ong-Schnorr scheme [27]; however, the complexity of verification is linear with the period time T. In [18], the authors proposed a scheme with highly efficient signing and verification based on [15]. Although their basic technique requires an expensive update, they show how to apply certain pebbling techniques to achieve constant update time. Furthermore, forward security has been introduced to other cryptographic primitives, such as forward secure encryption [9], forward secure aggregate signatures [25], and forward secure group signatures [23].

However, all the above schemes cannot prevent attackers from forging a fake signature of some message in future time periods.

2.2 Backward Security

For solving the key exposure problem exhaustively, we also expect a forward-secure signature scheme to be backward-secure, which prevents the adversaries from forging fakes in future time periods. There are several preliminary attempts on backward security, but the study in this field is scarce and a formal security definition about backward security is still absent.

In [8], the notion of forward-secure signature with untrusted update was proposed, which meets the backward security to a certain extent. Their scheme split the secret information into two parts: one (encrypted key, storing at machines) is for evolving keys in the binary tree and the other (password, remembering in the user) is for encrypting the former. They assume that password is secure and cannot be comprised. The signer has to use the password to sign a message at each signing stage, which ensures that the adversaries cannot forge a signature after key exposure. In [22], a generic forward-secure signature with untrusted update was proposed from any traditional signature scheme by applying the similar approach in [26]. However the efficiency of generic construction is low. In [14] and [24], one-way hash chain was employed to the forward-secure signature for achieving the backward detection, but it was vulnerable to attacks [28].

Compared with the previous work, our scheme not only achieves bilateral security, but also has a better performance and achieves unbounded number of time periods, as shown in the following sections.

3. PRELIMINARIES AND DEFINITIONS

3.1 Preliminaries

Bilinear map [7] is a cryptographic tool, which has been used for a number of cryptographic constructions. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two multiplicative cyclic groups of prime order q and g_1 and g_2 be two generators of $\mathbb{G}_1$. A bilinear map e is a map $e : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ with the following properties:

- *Computability*: There exists a polynomial time algorithm for computing map e efficiently.
- *Bilinearity*: For all $u, v \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_q$, $e(u^a, v^b) = e(u, v)^{ab}$.
- *Non-degeneracy*: $e(g_1, g_2) \neq 1$.

A bilinear pairing parameter generator is defined as a polynomial-time algorithm $\mathcal{IG}$, which takes as input a security parameter κ and outputs a representation $(e, \mathbb{G}_1, \mathbb{G}_2, q)$ of the bilinear map parameters that satisfy the above properties, where q is a κ-bit prime.

ASSUMPTION 1. *(CDH assumption). Given a finite cyclic group $\mathbb{G} = \langle g \rangle$ of prime order q with a generator g, for a given triple (g, g^a, h) where $a \in \mathbb{Z}_q$, the CDH problem is to compute a group element $w \in \mathbb{G}$ such that $w = h^a$. An algorithm $\mathcal{A}$ is said to have ϵ-advantage in solving the CDH problem if*

$$Pr[\mathcal{A}(\mathbb{G}, q, g, g^a, h) = h^a] > \epsilon \tag{1}$$

The CDH assumption holds in $\mathbb{G}$ if no probabilistic polynomial time algorithm has at least ϵ-advantage in solving the CDH problem in $\mathbb{G}$.

3.2 Definitions

DEFINITION 1. *A key-evolving signature scheme with bilateral security (i.e. forward-security and backward-security) has five polynomial algorithms, FBSS = (*KeyGen, CheckKey, Update, Sign, Verify*), where*

- *FBSS.*KeyGen*: The key generation algorithm takes as input a security parameter $1^\kappa, \kappa \in N$ and the total number of periods T, returns the initial secret key SK_0 and the public key PK.*
- *FBSS.*CheckKey*: The check key algorithm takes as input the time period j, the corresponding secret key SK_j and the public key PK, returns $\perp$ if the SK_j is not a valid key.*
- *FBSS.*Update*: The secret key update algorithm takes as input the time period j and the corresponding secret key SK_j, returns the new secret key SK_{j+1} of the time period $j + 1$.*
- *FBSS.*Sign*: The signing algorithm takes as input the secret key SK_j for the current time period j and the message M to be signed, returns a pair $\langle j, \sigma \rangle$, where σ is the signature of M at period j.*
- *FBSS.*Verify*: The verification algorithm takes as input the public key PK, a message M and a candidate signature $\langle j, \sigma \rangle$, returns 1 if $\langle j, \sigma \rangle$ is a valid signature of M or 0, otherwise.*

It is required that $Verify_{PK}(M,\ Sign_{SK_j}(M)) = 1$ for every message M and time period j.

We aim to have an in-depth understanding of the security of the key-evolving signature scheme against existential forgery under adaptive chosen-message attacks at each time period in this work. We now define the bilateral security in terms of a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. The game proceeds in three phases.

- Setup phase. The challenger $\mathcal{C}$ runs the KeyGen algorithm and gives the adversary $\mathcal{A}$ the public key PK. The current time period is set to 0.
- Query phase. In this phase $\mathcal{A}$ can issue three types of requests in an adaptive, interactive manner ($\mathcal{A}$ can repeatedly make Sign and Update queries):
  - Sign: The adversary can request the challenger to sign a message M on the current time period j; the challenger then returns a signature $\langle j, \sigma \rangle$ of M.
  - Update: The adversary $\mathcal{A}$ can request the challenger to execute the Update algorithm, in which case the time period is updated to the next period.
  - Break-in: The adversary $\mathcal{A}$ can request the challenger to hand out all the key at the current time period. (Unlike the definition in [5], which requires the game to move to the next phase once $\mathcal{A}$ makes a Break-in query. In our definition, we allow $\mathcal{A}$ continuously to issue Sign query and Update query, but the Break-in query is allowed only once.)
- Forge phase. Let j be the time period at which the adversary breaks in. The adversary produces a forgery, consisting of a time, message, signature tuple (j^*, M^*, σ^*). The adversary wins if the forged signature passes the Verify algorithm and the adversary has not queried for signature on M^* at the exact time j^*.

When the forge phase stops and the adversary issues a valid signature at time period j^*, one of the following three cases may occur. Case i): If $j^* < j$, the adversary generates a valid signature of past time periods, which breaks the forward-secure. Case ii): If $j^* > j$, the adversary generates a valid signature of future time periods, which breaks the backward-secure. Case iii): If $j^* = j$, the adversary always can generate a valid signature of time period j, because at the break-in phase, the adversary gets the responsive secret key at the time period j. Thus, in the definition of bilateral security, we only take case i) and ii) into consideration and ignore the last situation.

Formally, we give the following definitions:

Definition 2. *The key-evolving signature scheme $FBSS$ is (t, q_s, q_u, ε) forward-secure if no probabilistic polynomial time adversary has ε-advantage to forge a valid signature at past time period j^* ($j^* < j$) within running in time t after q_s signing queries and q_u update queries.*

Definition 3. *The key-evolving signature scheme $FBSS$ is (t, q_s, q_u, ε) backward-secure if no probabilistic polynomial time adversary has ε-advantage to forge a valid signature at future time period j^* ($j^* > j$) within running in time t after q_s signing queries and q_u update queries.*

A key evolving signature scheme is called bilateral-secure if it is both forward-secure and backward-secure. Formally,

Definition 4. *Let $\mathrm{AdvFB}_{\mathcal{A}}$ denote the advantage of an algorithm $\mathcal{A}$. We say that a key evolving signature scheme $FBSS$ is (t, q_s, q_u, ε) bilateral-secure if no probabilistic polynomial time adversary has advantage ε to forge a valid signature at any time period j^* except the time period of breaking in within running in time t after q_s signing queries and q_u update queries.*

4. CONSTRUCTION

4.1 Basic Idea

It is instructive to first understand the intuition behind our construction of forward secure and backward secure signature scheme (FBSS). To achieve forward security, we evolve the secret keys in a probabilistic manner. When the current key is exposed, the attacker cannot obtain any useful information of the past secret keys and forge a fake signature of the past time. To achieve backward security, we split the secret parameter into two secret keys and keep them in different ways. As long as the attacker cannot comprise these two secret keys at the same time, then he cannot update the two secret keys to that of the next time period and forge a fake signature of the future time, which ensure the backward security.

Furthermore, to achieve unbounded time periods, we use a novel key-evolving structure to support unbounded key updating. Existing forward-secure signature schemes usually use linear structure [5] or binary tree structure [9] to evolve the key, and the number of time periods is thus controlled by the predefined parameter T (the number of time periods) or l (the hight of the tree). In our construction, we employ a hub-and-spoke structure to evolve the key. Specifically, the signer generates a root key at the beginning of the key generation. And then chooses two secrets that satisfy a specified relation between the root key and stores the two secrets separately. After updating the key at each time period, the specified relation always holds and in the verify phase, the verifier not only check the correctness of the signature but also validate the specified relation is or is not holds. In this manner, the number of time periods in our construction is not limited by a predefined parameters and it can be unbounded large.

4.2 Detailed Scheme

We identify three roles in our construction: Alice for signing message, Bob for verifying the validity of signatures, and the $k+1$ players P_i, $i = 1, 2, \cdots, k+1$, for maintaining one part of the secret key in a distributed manner and updating the signing key at each time period. The notations used in the following construction are given in Table 1 and the flow chart of our design is illustrated in Figure 1. Roughly speaking, the proposed signature scheme splits the secret parameters into two parts: one x_1^j for signing message at time period j stored on the signer's side and the other one x_2^j is distributed to $k+1$ players using a secret sharing mechanism for updating the key x_1^j periodically after each time period. We assume that all shares of $k+1$ players will not be comprised at the same time. Anytime for signing a message, the signer first runs CheckKey to verify the validity of the key and then signs the message using a bilinear map. In the verification stage, the verifier validates the signature is signed by the effective signing key x_1^j at time period j by the public key.

Table 1: Notations used in our scheme.

Notations	Definitions
$\mathbb{G}_1$, $\mathbb{G}_2$, q	two groups for bilinear map of prime order q.
g	the generator of group $\mathbb{G}_1$.
$H_1(\cdot)$	a map-to-point hash function, which maps a message from message space to a element in group $\mathbb{G}_1$.
$H_2(\cdot)$	a map-to-point hash function, which maps a integer to a element in $\mathbb{G}_1$.
x_1^j	the first part secret information serving as the signing key.
x_2^j	the second part secret information serving as the updating key and stored in a distributed manner.
g_j	defined as $g_j = H_2(j)$ at each time period j, for $j = 0, 1, \cdots$.
Y_j	defined as $Y_j = g_j^{x_2^j}$ at each time period j for $j = 0, 1, \cdots$, and it will not reveal information about x_2^j.
SK_j	the secret key at time period j and stored at local.
PK_j	the public key at time period j and stored at verifier's side.
$f_j(i)$	the share of the i-th player p_i at time period j.

We now describe the detailed construction of our bilateral-secure signature scheme FBSS. Specifically, it works as follows:

- FBSS.KeyGen(1^κ):
  1. Generate groups $\mathbb{G}_1$, $\mathbb{G}_2$ of some prime order q and an admissible pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ by running $\mathcal{IG}(1^\kappa)$. Let g be the generator of $\mathbb{G}_1$.
  2. Chooses two cryptographic hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{Z}_q \to \mathbb{G}_1$.
  3. Select $x, x_1^0 \leftarrow \mathbb{Z}_q^* \setminus \{1\}$ randomly and set $x_2^0 = x \cdot (x_1^0)^{-1}$, $g_0 = H_2(0)$.

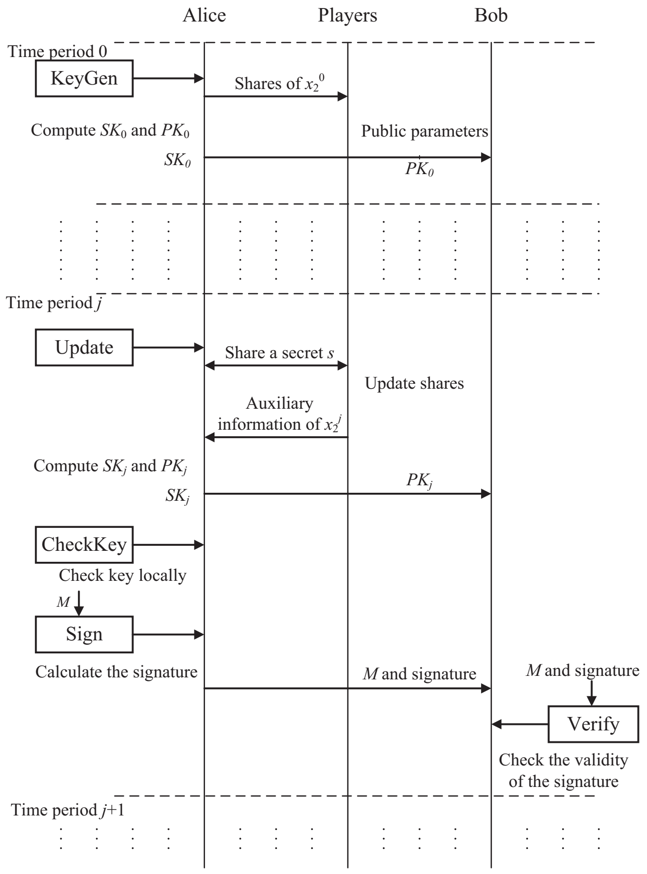

Figure 1: The flow chart of our scheme FBSS.

4. Construct a degree-k polynomial
$$f_0(t) = a_k t^k + \cdots + a_1 t + x_2^0$$
and then send $f_0(i)$ to i-th player P_i, for $i = 1, 2, \cdots, k+1$.

5. Compute $Y_0 = \prod_{i=1}^{k+1} Y_{0,i}$ [1], where
$$Y_{0,i} = (g_0^{f_0(i)})^{(-1)^{i-1}\binom{k+1}{i}}$$
for $i = 1, 2, \cdots, k+1$.

6. Set public key as $PK_0 = \langle g^x, g_0^x \rangle$ and $SK_0 = \langle x_1^0, Y_0 \rangle$. Finally, delete x and x_2^0 immediately.

- FBSS.CheckKey(j, SK_j, PK_j):
  1. Parse SK_j as (x_1^j, Y_j) and compute the hash value $g_j = H_2(j)$.
  2. Check whether $e(g^{x_1^j}, Y_j) = e(g, g_j^x)$ holds. Return true if the equation holds, otherwise return false.

- FBSS.Update(j, SK_j):
  1. Parse SK_j as (x_1^j, Y_j).
  2. Share a secret $s \in \mathbb{Z}_q^* \setminus \{1\}$ with all $k+1$ players.
  3. Compute $x_1^{j+1} = x_1^j \cdot s$. Each player P_i updates the share with $f_{j+1}(i) = f_j(i) \cdot s^{-1}$ and sends the new $Y_{j+1,i} = (g_{j+1}^{f_{j+1}(i)})^{(-1)^{i-1}\binom{k+1}{i}}$, where $g_{j+1} = H_2(j+1)$.
  4. Calculate $Y_{j+1} = \prod_{i=1}^{k+1} Y_{j+1,i}$ and sets $SK_{j+1} = \langle x_1^{j+1}, Y_{j+1} \rangle$, $PK_{j+1} = \langle g^x, Y_{j+1}^{x_1^{j+1}} \rangle$. Finally, delete s immediately.

In fact, by the correctness analysis in Theorem 1 we have that $PK_{j+1} = g_{j+1}^{x_1^{j+1} \cdot x_2^{j+1}} = g_{j+1}^x$, where
$$x_2^{j+1} = \sum_{i=1}^{k+1} (-1)^{i-1} \binom{k+1}{i} f_{j+1}(i).$$

- FBSS.Sign(j, SK_j, M):
  1. Check whether the secret key SK_j is the valid key for time period j. If this test fails, output $\perp$ and halt. Otherwise continue.
  2. Calculate the hash $m \leftarrow H_1(M)$, $m \in \mathbb{G}_1$.
  3. The signature of M at time period j is $\langle j, \sigma \rangle$, where $\sigma = \langle m^{x_1^j}, Y_j \rangle$.

- FBSS.Verify$(PK_j, M, \langle j, \sigma \rangle)$:.

[1]From Lagrange interpolating formula, we have that
$$x_2^0 = f_0(0) = \sum_{i=1}^{k+1} (-1)^{i-1} \binom{k+1}{i} f_0(i)$$
and thus we obtain that $Y_0 = g_0^{x_2^0}$.

1. Parse σ as $\langle m^{x_1^j}, Y_j \rangle$, calculate the hash $m \leftarrow H_1(M)$, and ensure that the following two equations holds, where g_j^x is calculated from public key PK_j.

$$e(m^{x_1^j}, Y_j) \stackrel{?}{=} e(m, g_j^x) \tag{2}$$

and

$$e(g, g_j^x) \stackrel{?}{=} e(g^x, g_j) \tag{3}$$

2. Return 1 if Equation (2) and Equation (3) both hold, otherwise return 0.

5. CORRECTNESS AND SECURITY ANALYSIS

5.1 Correctness

The correctness of the above scheme is justified by the following theorem.

Theorem 1. *The proposed FBSS signature scheme is correct.*

Proof. Our proof proceeds in two steps: the first step shows that at each time period j, equation

$$x_1^j \cdot x_2^j = x \tag{4}$$

always holds, where x_2^j is defined as following

$$x_2^j = \sum_{i=1}^{k+1} (-1)^{i-1} \binom{k+1}{i} f_j(i)$$

and $f_j(i)$ denotes the share of the i-th player at time period j; the second argues that the verification procedure is correct.

Step-1: We show the Equation (4) holds at each time period j for $j = 0, 1, \cdots$ by mathematical induction.

When $j = 0$, the following two equation hold clearly from the key generation algorithm.

$$x_1^0 \cdot x_2^0 = x$$

and

$$x_2^0 = \sum_{i=1}^{k+1} (-1)^{i-1} \binom{k+1}{i} f_0(i)$$

Suppose the Equation (4) holds at time period j, then after running the key update algorithm, we have that

$$\begin{aligned} x_1^{j+1} \cdot x_2^{j+1} &= x_1^{j+1} \cdot (\sum_{i=1}^{k+1} (-1)^{i-1} \binom{k+1}{i} f_{j+1}(i)) \\ &= (x_1^j \cdot s) \cdot (\sum_{i=1}^{k+1} (-1)^{i-1} \binom{k+1}{i} f_j(i)) \cdot s^{-1} \\ &= x_1^j \cdot (\sum_{i=1}^{k+1} (-1)^{i-1} \binom{k+1}{i} f_j(i)) \\ &= x_1^j \cdot x_2^j \\ &= x \end{aligned}$$

where the second '=' holds since that $f_{j+1}(i) = f_j(i) \cdot s^{-1}$ from the update procedure. Consequently, the equation $x_1^j \cdot x_2^j = x$ always holds at any time period.

Step-2: We now show the correctness of the verification procedure.

$$\begin{aligned} e(m^{x_1^j}, Y_j) &= e(m^{x_1^j}, g_j^{x_2^j}) \\ &= e(m, g_j^{x_1^j \cdot x_2^j}) \\ &= e(m, g_j^x) \end{aligned}$$

Combining steps 1 and 2, our proposed FBSS signature scheme is correct. $\square$

5.2 Security

The following theorem shows that the signature scheme is bilateral-secure. Security of the scheme follows from the hardness of the CDH problem.

Theorem 2. *The proposed key-evolving signature scheme is bilateral-secure under the random oracle model for any probabilistic polynomial time adversary if the CDH assumption holds.*

Proof. We prove it by contradiction. Suppose there is a probabilistic polynomial time adversary $\mathcal{A}$ $(t, q_s, q_u, \varepsilon)$ that breaks the signature scheme, namely breaks the forward-security or backward-security with ε-advantage. Then we build a t'-time algorithm $\mathcal{B}$ that solves the CDH problem with advantage at least ϵ', where

$$t' \le t + c_q(q_{H_1} + 2q_s + 2q_u) \tag{5}$$

c_q denotes the maximum time of one time query, q_{H_1} denotes the number of queries to the hash function H_1 and

$$\epsilon' \ge \frac{\epsilon}{2e(1+q_s)}. \tag{6}$$

which yields a algorithm that solves the CDH problem in group $\mathbb{G}_1$ with at least $\frac{\epsilon}{2e(1+q_s)}$-advantage.

Let $(g, u = g^a, h)$ be $\mathcal{B}$'s challenge in group $\mathbb{G}_1$. Recall that $\mathcal{B}$'s goal is to break the CDH assumption which asks $\mathcal{B}$ to find $v \in \mathbb{G}_1$ such that $v = h^a$. $\mathcal{B}$ interacts with $\mathcal{A}$ in the following three phases:

- Setup phase: Algorithm $\mathcal{B}$ starts by selecting two random integers $y_1, y_2 \in \mathbb{Z}_q$, then $\mathcal{B}$ calculates $u^{y_2} g^{y_1 y_2} = g^{y_2(a+y_1)}$ as the public key at time periods 0, which implies that $x = y_2(a + y_1)$, $x_1 = a + y_1$ and $Y_0 = g^{y_2}$. $\mathcal{B}$'s goal is to compute h^a. In the process of attack, our algorithm $\mathcal{B}$ records all information (previous public keys or signatures) of different time period into a table, which provide facilities for $\mathcal{B}$ to query the information of previous time periods while calculating h^a in the future. The current time period is set to 0. Algorithm $\mathcal{B}$ maintains a list of tuples $(j, s_j, x_1^j, \pi_j, Y_j)$ for key updating and this list is initially empty. We call this list as the updated-list. Finally, $\mathcal{B}$ adds the tuple $(0, 1, \perp, 0, Y_0)$ to the updated-list, where $\perp$ denotes the default values.

- Query phase: In the query phase $\mathcal{A}$ can issue several types of requests in an adaptive, interactive manner. ($\mathcal{A}$ can also repeatedly make random oracle queries to H_1 and H_2 just as the Sign and Update queries).

  - H_2 queries: At any time period j, $\mathcal{B}$ allows $\mathcal{A}$ to query the random oracle H_2. In order to respond to these queries, $\mathcal{B}$ maintains a list of tuples

(j, g_j, c_j) as explained below. We refer to this list as the H_2-list. This list is initially empty. When $\mathcal{A}$ queries the oracle H_2 at time period j, algorithm $\mathcal{B}$ responds as follows:

1. If the query at time period j already has been issued on the H_2-list in a tuple (j, g_j, c_j), then the algorithm $\mathcal{B}$ responds with $H_2(j) = g_j \in \mathbb{G}_1$.
2. Otherwise, $\mathcal{B}$ chooses a random $c_j \in \mathbb{Z}_q$ and computes $g_j = g^{c_j} \in \mathbb{G}_1$.
3. Algorithm $\mathcal{B}$ adds the tuple (j, g_j, c_j) to the H_2-list and responds to $\mathcal{A}$ by $H_2(j) = g_j$.

In fact, at each time period j, we allow algorithm $\mathcal{A}$ to query H_2 many times. However, no matter how many times $\mathcal{A}$ queries, the query results are the same. Thus, $\mathcal{A}$ queries to H_2 only once in the description of algorithm $\mathcal{B}$.

– H_1 queries: At any time period j, $\mathcal{B}$ allows $\mathcal{A}$ to query the random oracle H_1. For responding to these queries, $\mathcal{B}$ maintains a list of tuples (M_i, w_i, b_i, τ_i) as explained below. We refer to this list as the H_1-list. This list is initially empty. When $\mathcal{A}$ queries the oracle H_2 at a point $M_i \in \{0, 1\}^*$, algorithm $\mathcal{B}$ responds as follows:

1. If the query of M_i already has been issued on the H_1-list in a tuple (M_i, w_i, b_i, τ_i), then algorithm $\mathcal{B}$ responds with $H_1(M_i) = w_i \in \mathbb{G}_1$.
2. Otherwise, $\mathcal{B}$ generates a random coin $\tau_i \in \{0, 1\}$ such that $\Pr[\tau_i = 0] = 1/(q_s + 1)$.
3. Algorithm $\mathcal{B}$ also runs the above algorithm for responding to H_2-list and obtains a $g_j \in \mathbb{G}_1$ such that $H_2(j) = g_j$.
4. Algorithm $\mathcal{B}$ picks a random $b_i \in \mathbb{Z}_q$ and computes $w_i = h^{1-\tau_i} g_j^{b_i} \in \mathbb{G}_1$.
5. Algorithm $\mathcal{B}$ adds the tuple (M_i, w_i, b_i, τ_i) to the H_1-list and responds to $\mathcal{A}$ by $H_1(M_i) = w_i$.

Note that either way w_i is uniform in $\mathbb{G}_1$ and is independent of $\mathcal{A}$' current view as required.

– Update: At any time period $j - 1$, algorithm is allowed to issue the key update queries. Firstly, $\mathcal{B}$ picks a random coin $\pi_j \in \{0, 1\}$ such that $\Pr[\pi_j = 0] = 1/2$. If $\pi_j = 0$, then $\mathcal{B}$ updates the key to the next time period j as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding H_2-list to obtain a $g_j \in \mathbb{G}_1$ such that $H_2(j) = g_j$.
2. Algorithm $\mathcal{B}$ selects a random $s_j \in \mathbb{Z}_q$ and computes $Y_j = g^{y_2 s_j^{-1}}$ and $PK_j = (ug^{y_1})^{c_j y_2}$, where $u = g^a$.
3. Algorithm $\mathcal{B}$ adds the tuple $(j, s_j, \perp, \pi_j, Y_j)$ to the updated-list, here $\pi_j = 0$.

If $\pi_j = 1$, then $\mathcal{B}$ updates the key to the next time period j as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding H_2-list to obtain a $g_j \in \mathbb{G}_1$ such that $H_2(j) = g_j$.
2. Algorithm $\mathcal{B}$ selects a random $x_1^j \in \mathbb{Z}_q$ and computes $Y_j = (g_j)^{y_2(a+y_1)(x_1^j)^{-1}}$ and $PK_j = (g_j)^{y_2(a+y_1)}$. One may worry that a is unknown and in fact we can calculate the PK_j by
$$\begin{aligned} PK_j &= (g_j)^{y_2(a+y_1)} \\ &= (g^{c_j})^{y_2(a+y_1)} \\ &= (ug^{y_1})^{y_2 c_j} \end{aligned}$$
and thus Y_j.
3. Algorithm $\mathcal{B}$ adds the tuple $(j, \perp, x_1^j, \pi_j, Y_j)$ to the updated-list, here $\pi_j = 1$.

– Sign: Let M_i be a signature query at time period j issued by $\mathcal{A}$. If $\pi_j = 0$, then we obtain the tuple $(j, s_j, \perp, 0, Y_j)$ from the updated-list. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding H_1-list to obtain a $w_i \in \mathbb{G}_1$ such that $H_1(M_i) = w_i$. Let (M_i, w_i, b_i, τ_i) be the corresponding tuple in the H_1-list. If $\tau_i = 0$, then $\mathcal{B}$ reports failure and terminates.
2. Otherwise, we know $\tau_i = 1$ and hence $w_i = (g_j)^{b_i}$ where $g_j = g^{c_j}$. Then $\mathcal{B}$ responds to $\mathcal{A}$ with $\langle j, \sigma_i = \langle \sigma_{ij}, Y_j \rangle \rangle$, where
$$\sigma_{ij} = (ug^{y_1})^{c_j b_i s_j}$$
and Y_j is obtained from the updated-list. Observe that $\sigma_{ij} = w_i^{(a+y_1)s_j}$ and the underlying $x_1^j \cdot x_2^j = (a + y_1)s_j \cdot y_2 s_j^{-1} = y_2(a + y_1)$, therefore σ_i is a valid signature on M_i at time period j.

If $\pi_j = 1$, then we obtain the tuple $(j, \perp, x_1^j, 1, Y_j)$ from the updated-list. Algorithm $\mathcal{B}$ responds to this query as follows:

1. Algorithm $\mathcal{B}$ runs the above algorithm for responding H_1-list to obtain a $w_i \in \mathbb{G}_1$ such that $H_1(M_i) = w_i$.
2. Then $\mathcal{B}$ responds to $\mathcal{A}$ with $\langle j, \sigma_i = \langle \sigma_{ij}, Y_j \rangle \rangle$, where $\sigma_{ij} = w_i^{x_1^j}$ and Y_j is obtained from the updated-list. Observe that the underlying $x_1^j \cdot x_2^j = x_1^j \cdot y_2(a+y_1)(x_1^j)^{-1} = y_2(a+y_1)$, therefore σ_i is a valid signature on M_i at time period j.

– Break-in: The algorithm $\mathcal{A}$ is allowed to request the sign key at some time periods. For responding the query at time period j, algorithm $\mathcal{B}$ does the following:

1. Algorithm $\mathcal{B}$ gets the π_j from the updated-list. If $\pi_j = 0$, then $\mathcal{B}$ reports failure and terminates.
2. Otherwise, $\mathcal{B}$ obtains the x_1^j and Y_j from the updated-list and sends $\langle x_1^j, Y_j \rangle$ as the signing key to $\mathcal{A}$.

- Forge phase: Eventually algorithm $\mathcal{A}$ produces a message-signature pair (j^*, M_f, σ_f) of time period j^* such that no signature query was issued for M_f and j^* is not the break-in time period. If there is no tuple on the H_1-list containing M_f, then $\mathcal{B}$ issues a query itself for

$H_1(M_f)$ to ensure that such a tuple exists. We assume σ_f is a valid signature on M_f at time period j^*, where $\sigma_f = \langle \sigma_{fj}, Y_{j^*} \rangle$. If it is not, $\mathcal{B}$ reports failure and terminates. Then we computes h^a as follows:

1. Algorithm $\mathcal{B}$ gets the π_{j^*} from the updated-list. If $\pi_{j^*} = 1$, then $\mathcal{B}$ reports failure and terminates.
2. Otherwise, $\mathcal{B}$ finds the tuple (M_f, w, b, τ) on the H_1-list. If $\tau = 1$, $\mathcal{B}$ reports failure and terminates. Otherwise, $\tau = 0$ and therefore $H_1(M_f) = w = h \cdot g^{bc_{j^*}}$. Hence we have

$$\begin{aligned}\sigma_{fj} &= h^{(a+y_1)s_{j^*}} \cdot (g^{bc_{j^*}})^{(a+y_1)s_{j^*}} \\ &= (h^a \cdot h^{y_1} \cdot (ug^{y_1})^{bc_j})^{s_{j^*}}\end{aligned}$$

and thus we compute h^a by

$$h^a = \frac{(\sigma_{fj})^{s_{j^*}^{-1}}}{h^{y_1} \cdot (ug^{y_1})^{bc_{j^*}}}$$

This completes the description of algorithm $\mathcal{B}$. It remains to show that $\mathcal{B}$ solves the CDH problem in $\mathbb{G}_1$ with probability at least ϵ'. To do so, we analyze four events needed for $\mathcal{B}$ to succeed:

- $\mathcal{E}_1$: $\mathcal{B}$ does not abort as a result any of $\mathcal{A}$'s signature queries.
- $\mathcal{E}_2$: $\mathcal{B}$ does not abort at the break-in phase.
- $\mathcal{E}_3$: $\mathcal{A}$ generates a valid message-signature pair M_f, σ_f.
- $\mathcal{E}_4$: $\pi_{j^*} = 0$ for the tuple on the updated-list and $\tau = 0$ for the tuple on the tuple containing M_f on the H_1-list.

From the descrition of algorithm $\mathcal{B}$, the parameters π_{j^*} and τ in event $\mathcal{E}_4$ are chosen randomly and thus $\mathcal{E}_4$ is independent of $\mathcal{E}_3$, and the probability of succeeding is

$$\begin{aligned}\Pr[\mathcal{B}\text{ succeeds}] &= \Pr[\mathcal{E}_3]\Pr[\mathcal{E}_4] \\ &= \Pr[\mathcal{E}_3|\mathcal{E}_1 \wedge \mathcal{E}_2]\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2]\Pr[\mathcal{E}_4]\end{aligned}$$

Since that $\mathcal{E}_1$ and $\mathcal{E}_2$ are independent of each other, we have that $\Pr[\mathcal{E}_1 \wedge \mathcal{E}_2] = \Pr[\mathcal{E}_1]\Pr[\mathcal{E}_2]$. Consequently, we obtain that

$$\Pr[\mathcal{B}\text{ succeeds}] = \Pr[\mathcal{E}_3|\mathcal{E}_1 \wedge \mathcal{E}_2]\Pr[\mathcal{E}_1]\Pr[\mathcal{E}_2]\Pr[\mathcal{E}_4] \quad (7)$$

First, the sign queries were issued at most q_s times and the the probability that algorithm $\mathcal{B}$ does not abort because of sign queries is at least

$$\Pr[\mathcal{E}_1] = (1 - \frac{1}{1+q_s})^{q_s} \geq \frac{1}{e} \quad (8)$$

where e is the base of the natural logarithm.

Second, the break-in queries and forge phase both happen at most only once and the coin π in the updated-list and the coin τ in the H_1-list are both independent, thus the probability of $\mathcal{E}_2$ and $\mathcal{E}_4$ is

$$\Pr[\mathcal{E}_2] = \frac{1}{2} \quad (9)$$

and

$$\Pr[\mathcal{E}_4] = \frac{1}{2} \cdot \frac{1}{1+q_s} \quad (10)$$

Third, if algorithm $\mathcal{B}$ does not abort in the signature queries and break-in query, then algorithm $\mathcal{A}$'s view is identical to its view in the real attack. Thus the probability of $\mathcal{A}$ generating a valid message-signature pair M_f, σ_f is

$$\Pr[\mathcal{E}_3|\mathcal{E}_1 \wedge \mathcal{E}_2] = \epsilon \quad (11)$$

Consequently, substituting Equations (8), (9), (11) and (10) into Equation (7), we obtain the probability of $\mathcal{B}$ succeeding is

$$\epsilon' = \Pr[\mathcal{B}\text{ succeeds}] = \Pr[\mathcal{E}_3]\Pr[\mathcal{E}_4] \geq \frac{\epsilon}{2e(1+q_s)}$$

Algorithm $\mathcal{B}$'s running time is the same as $\mathcal{A}$'s running time plus the time it takes for $q_{H_1} + q_s$ H_1 queries, q_u H_2 queries, q_s signature queries, q_u key update queries, and one break-in query. Suppose each of these queries takes at most c_q time. Hence the total running time is at most

$$t' \leq t + c_q(q_{H_1} + 2q_s + 2q_u)$$

This completes the proof of Theorem 2. □

6. PERFORMANCE EVALUATION

In this section, we show the theoretical and experimental performance of our scheme respectively. We also compare our scheme FBSS with the scheme FSSUU in [8], which is the most related state-of-the-art work. First, we compare FBSS and FSSUU from the perspectives of time complexity and space complexity theoretically. Second, we implement FBSS and FSSUU in C++ programming language to evaluate their experimental performance. Experimental results demonstrate that FBSS is more efficient than FSSUU, except for the key update algorithm; FBSS is also more space-saving than FSSUU because of the trade off between the time complexity and space complexity.

6.1 Theoretical Evaluation

We first compare our signature scheme FBSS and the scheme FSSUU theoretically, which illustrates that our F-BSS outperforms FSSUU in terms of effectiveness and efficiency.

Let T_p be the time of one pairing operation, T_e be the time of one exponentiation operation, T_m be the time of one multiplication operation, and $\mathbb{G}_1$ be the length of the elements in the group $\mathbb{G}_1$ (such as 128, 256, 512 bits). Usually, we have that $T_p > T_e > T_m$, where T_p is about 8 ms, T_e is about 1.8 ms and T_m is about 0.013 ms. Table 2 presents the comparison between our FBSS and the FSSUU, where l is a parameter capturing the total time periods in FSSUU, m is the binary representation length of message in FSSUU, and k is the parameter relevant to the number of players in FBSS.

In the key generation algorithm, our FBSS requires $k+1$ exponentiation operations and $k(k+1)$ multiplication operations; the costs in the FSSUU mainly center on the $\frac{l(l+1)}{2}$ exponentiation operations. In reality, to maintain a signature system to be used for a longer time, l usually is larger than the parameter k, for example $l = 10, 20, 30, \cdots$ and $k = 3, 4, 5, \cdots$. Thus FSSUU requires more times to generate their keys. $\frac{l(l-1)}{2}$ pairing operations are required to check whether a key is valid at some period in FSSUU, which is increasing with the parameter l. Clearly, the cost is much lager than the two pairing operations in FBSS. To compare

Table 2: Theoretical comparison of FBSS and FSSUU

Algorithms	FSSUU	FBSS
KeyGen	$2T_p + \frac{l(l+1)}{2}T_e$	$(k+1)T_e + k(k+1)T_m$
CheckKey	$\frac{l(l-1)}{2}T_p + (\frac{l}{2}+4)T_m$	$2T_p + T_e$
Update	$T_e + \frac{l^2}{4}T_m$	$(k+1)T_e + k^2T_m$
Sign	$(\frac{l+m}{2}+10)T_e$	$2T_p + 2T_e$
Verify	$3T_p + (\frac{l+m}{2}+5)T_m$	$2T_p$
Secret storage size	$\frac{l(l+3)}{4}\lvert\mathbb{G}_1\rvert$	$2\lvert\mathbb{G}_1\rvert$
Forward secure	Yes	Yes
Bilateral secure	At some extent	Yes
Existence Forgery	Yes	Yes
Time periods	$T = 2^l$	Unbounded

the costs for signing a message, our FBSS only needs two paring operations and two exponentiation operations; however FSSUU requires $\frac{l+m}{2} + 10$ exponentiation operations, which is also much lager than the costs in FBSS because the message in FSSUU is denoted as a binary representation, which is about hundreds of bits. In the verification phase, the costs in FBSS have one time pairing operation advantage than FSSUU, which demonstrates that our FBSS is more efficient than FSSUU. For updating the secret keys from one time period to the next time period, our FBSS requires $(k + 1)$ exponentiation operations; the FSSUU only needs one time exponentiation operation, which is more cost-effective. However, this (time complexity) advantage for FSSUU is achieved by sacrificing more space complexity, namely secret storage size. The secret storage size in FSSUU is about $\frac{l(l+3)}{4}$ elements in group $\mathbb{G}_1$, which depends on the parameter l; our FBSS only needs two elements in the group. This formulates the trade off between the time efficiency for updating the secret parameters and the space size of storing secret parameters.

From the security perspective, both schemes are forward secure, but the work FSUU does not achieve the bilateral security. This is because the key update algorithm of FSSUU only depends on the inputs secret key of the current time period, the ID of new time period, and the public key. Once the secret key was comprised, the attacker can update the key to any time periods in the future. However, in our FBSS, we utilize the distributed key to update the current secret key. Although the attacker may compromise the current time period, the attacker cannot produce a secret key of future time periods.

Both scheme are based on the idea of splitting the secret information into two parts. In the construction of FSSUU, one piece secret information is stored in the signer's machine and the other piece secret, password, is memorized by the signer. To ensure against the existence forgery, the signer signs a message using the password, which is assumed not to be compromised. In the case when signing messages is more frequent than updating key, the scheme FSSUU does not work because the signer has to input the password for each signing. In our construction of FBSS, we also split the secret information, the one for updating secret key is distributed to other parties and the other one for signing message is stored locally. To protect against the existence forgery, we only use the signing key to sign a message, which is very suitable for the case when signing messages is more frequent than updating keys.

Furthermore, our work achieves the unbounded number of time periods, which allows the signer to update the secret key arbitrarily with unbound times. However, in the construction of FSSUU, the signer has to prolong the lifetime of the signature system by increasing the parameter l, which leads to much more time costs of algorithms KeyGen, CheckKey, Sign and verify and more space costs for storing secret parameters. Oppositely, for reducing these costs from both time and space, the signer has to sacrifice the lifetime of the signature system as a cost. Thus, our proposed scheme is more practical for real-world applications.

6.2 Experimental Evaluation

We now evaluate the performance of the proposed FBSS and FSSUU experimentally. We first introduce the methodology of our experiments when implementing both schemes, then we present the experimental results and the analysis.

6.2.1 Methodology and Configuration

All the following experiments are based on C++ (Visual Studio 2013) and are conducted on an Intel-based i5-2320 personal computer with 3GHz processor and 4GB RAM. In our experiments, we utilize the GNU Multiple Precision Arithmetic (GMP) library [1] and Pairing Based Cryptography (PBC) library version 0.5.14 [2] to implement both signature schemes, where type A parameter (a.param) is used to do the paring operations. All experimental results represent the mean of 10 trials.

We implements both schemes by building two classes for every signature scheme. Each class contains five public interfaces (KeyGen, CheckKey, Update, Sign and verify) and the data parameters (system parameters, public parameters and secret parameters). For both implementations, we generate the system parameters by invoking an auxiliary protected method Initialize, which allows us to get a more precise time cost evaluation of the key generation procedure. In the implementation of FBSS, the parameter k is set to $k = 5$ and in the implementation of FSSUU the parameter l is set to $l = 10, 15, \cdots$. For the hash functions in both scheme, we use the ready-made method *element_from_hash* in the PBC library for convenience. In FSSUU, the messages to be signed must be fixed-length binary strings of m-bits, which is set to $m = 160$ in our implementation and all messages

Table 3: Experimental results of FBSS and FSSUU

Algorithms	FSSUU							FBSS
	$l = 10$	$l = 15$	$l = 20$	$l = 25$	$l = 30$	$l = 35$	$l = 40$	
KeyGen(ms)	973.707	1159.69	1522.45	1986.74	2527.84	3157.06	3856.21	35.0392
CheckKey(ms)	397.753	895.707	1592.14	2486.87	3585.52	4853.7	6329.3	11.299
Update(ms)	10.899	11.0372	11.2081	11.3047	11.5657	11.7709	12.3309	29.8116
Sign(ms)	399.61	918.902	1629.51	2507.68	3588.05	4875.89	6352.02	22.7591
Verify(ms)	11.6304	11.7059	12.9753	11.4605	11.8231	11.9514	11.6626	8.13898

are generated from $\{0,1\}^m$ randomly. To compare fairly, the same configuration is used in the implementation of F-BSS. Besides above, we use the method *sizeof* to compute the storage size of secret parameters.

6.2.2 Experimental Results

The experimental results include two aspects: the time cost in milliseconds of every operation and the space cost in kilobytes of secret parameters. The results and their analysis are detailed in the following paragraphs.

First, we estimate the time cost of every operation, which has been presented in Table 3. For algorithm KeyGen, the time cost of FSSUU requires 970+ ms when $l = 10$; with the parameter l increasing, the time cost also grows fast. However, our FBSS only consumes 35.0352 ms, which improves by more than 27 times. For algorithm CheckKey, the time cost of FSSUU needs 397+ ms when $l = 10$; with the parameter l increasing, the time cost also grows much faster than algorithm KeyGen. However, our FBSS only consumes 11.299ms and when $l = 40$; our FBSS improves about 600 times. The cost of signing message is close to the cost for checking key in FSSUU, but our FBSS's time cost for signing is only 22.7591ms, which is more efficient than that of FSSUU. To verify whether a message-signature pair is valid, $11 \sim 12$ ms is required in FSSUU while our FBSS only needs 8.13898ms, which has slight improvement. The FSSUU has its superiority over FBSS in updating the secret key to a next time period, but this has little impact on the performance of FBSS in practice because the key update is not very frequent in reality. For example, the signer may update the secret key once a day, a week or a month, the time cost of updating secret parameter in FBSS is negligible since it only takes less than one second.

Second, we compare the space cost of secret parameters, which has been presented in Fig. 2. While our FBSS signature scheme only needs 0.25 KB for storing the secret key, the FSSUU requires 1.625 KB if we set $l = 5$. If the signer updates the key once a day using the FSSUU signature scheme, the signer has to restart the signature system after a month later, because $T = 2^5$. For prolonging the lifetime of the signature system, the signer has to increase the parameter l. However, with the increasing of l, the storage size also grows in a quadratic speed. When l increases to 40, 106.625 KB is required to store the secret parameters, which is indeed costly than ours. Consequently, the experimental results validate our theoretical analysis, and our signature scheme is more suitable for practical application.

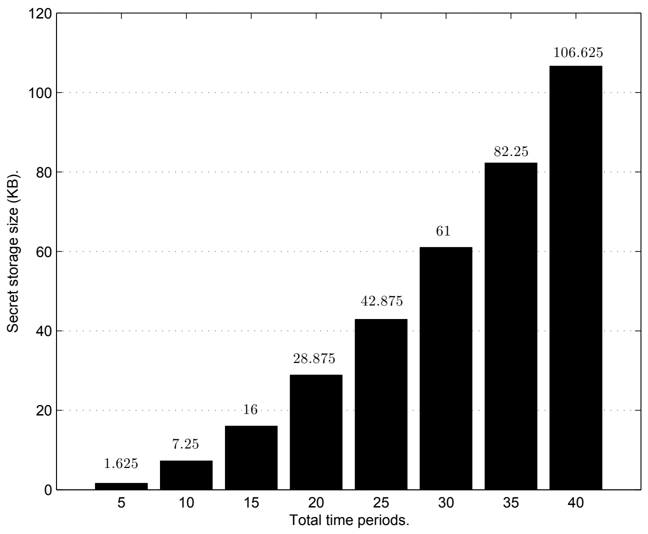

Figure 2: The relation between the parameter l and secret storage size in FSSUU. The secret storage size is 0.25KB when $k = 5$ in the FBSS.

7. CONCLUSIONS

In this paper, we introduced the concept of key-evolving signature with bilateral security (i.e. forward-security and backward-security) to deal with the key exposure problem in digital signature. This new concept advances the current research on forward-secure digital signature by considering the signature forgeries of future time period once the key of current time period is leaked. First, we defined the bilateral security of key-evolving signature scheme, which prevents the attacker from forging a valid signature of the past and future time periods when the key is exposed. Next, we presented a novel and specific construction satisfying the above requirements. Our construction splits secret parameters into two parts: the first part is used to update the keys and stored in a distributed way; the second part is employed for signing message and maintained by the singer. We employed a hub-and-spoke structure to evolve the key for supporting unbounded number of key updating. Then, we proved that our construction is correct and bilateral-secure under the random oracle model and the CDH assumption. Finally, we demonstrated that our construction outperforms pervious work by theoretical and experimental evaluation.

8. ACKNOWLEDGMENTS

The work in this paper was supported by the Fundamental Research Funds for the Central Universities (No. 106112015CDJXY180003), the National Natural Science Foundation of China (No. 61502314), and the Fundamental Research Funds for the Shenzhen University (No. 201533).

9. REFERENCES

[1] The GNU multiple precision arthmetic library. https://gmplib.org/.

[2] PBC: the pairing-based cryptography library. http://crypto.stanford.edu/pbc/.

[3] M. Abdalla and L. Reyzin. A new forward-secure digital signature scheme. In *ASIACRYPT*, pages 116–129. 2000.

[4] R. Anderson. Two remarks on public key cryptography. In *Invited Talk, ACM Conference on Computer and Communications Security (CCS)*, 1997.

[5] M. Bellare and S. K. Miner. A forward-secure digital signature scheme. In *CRYPTO*, pages 431–448, 1999.

[6] P. Błaśkiewicz, P. Kubiak, and M. Kutyłowski. Digital signatures for e-government - a long-term security architecture. In *Forensics in Telecommunications, Information, and Multimedia*, pages 256–270, 2010.

[7] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRYPT*, pages 416–432, 2003.

[8] X. Boyen, H. Shacham, E. Shen, and B. Waters. Forward-secure signatures with untrusted update. In *ACM Conference on Computer and Communications Security (CCS)*, pages 191–200, 2006.

[9] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[10] E. Cronin, S. J. h, T. Malkin, and P. McDaniel. On the performance, feasibility, and use of forward-secure signatures. In *ACM Conference on Computer and Communications Security (CCS)*, pages 131–144, 2003.

[11] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO*, pages 307–315, 1990.

[12] Y. Dodis, J. Katz, S. Xu, and M. Yung. Key-insulated public key cryptosystems. In *EUROCRYPT*, pages 65–82, 2002.

[13] Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. In *International Conference on Practice and Theory of Public-Key Cryptography (PKC)*, pages 130–144, 2002.

[14] D. Guan, D.-R. Lin, and C.-I. Wang. A forward-secure signature with backward-secure detection. In *International Conference on Information Security and Assurance (ISA)*, pages 106–110, 2008.

[15] L. C. Guillou and J.-J. Quisquater. A "paradoxical" identity-based signature scheme resulting from zero-knowledge. In *CRYPTO*, pages 216–231, 1990.

[16] C. G. Günther. An identity-based key-exchange protocol. In *EUROCRYPT*, pages 29–37, 1990.

[17] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *ACM Conference on Computer and Communications Security (CCS)*, pages 100–110, 1997.

[18] G. Itkis and L. Reyzin. Forward-secure signatures with optimal signing and verifying. In *CRYPTO*, pages 332–354, 2001.

[19] G. Itkis and L. Reyzin. SiBIR: Signer-base intrusion-resilient signatures. In *CRYPTO*, pages 499–514, 2002.

[20] A. Kozlov and L. Reyzin. Forward-secure signatures with fast key update. In *International Conference on Security in Communication Networks (SCN)*, pages 241–256, 2003.

[21] H. Krawczyk. Simple forward-secure signatures from any signature scheme. In *ACM Conference on Computer and Communications Security (CCS)*, pages 108–115, 2000.

[22] B. Libert, J.-J. Quisquater, and M. Yung. Forward-secure signatures in untrusted update environments: efficient and generic constructions. In *ACM Conference on Computer and Communications Security (CCS)*, pages 266–275, 2007.

[23] B. Libert and M. Yung. Dynamic fully forward-secure group signatures. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 70–81, 2010.

[24] G. Lize, W. Feng, Z. Yousheng, and Z. Shi-hui. A bilateral secure threshold signature scheme with distinguished signing authorities. *International Journal of Advancements in Computing Technology*, 4(8), 2012.

[25] D. Ma. Practical forward secure sequential aggregate signatures. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 341–352, 2008.

[26] T. Malkin, D. Micciancio, and S. Miner. Efficient generic forward-secure signatures with an unbounded number of time periods. In *EUROCRYPT*, pages 400–417, 2002.

[27] H. Ong and C.-P. Schnorr. Fast signature generation with a fiat shamir-like scheme. In *EUROCRYPT*, pages 432–440, 1991.

[28] L. Wang, K. Chen, X. Mao, and Y. Wang. On the security of a forward-backward secure signature scheme. *International Journal of Network Security*, 17(3):307–310, 2015.