A New Forward Secure Signature Scheme using Bilinear Maps*

Fei Hu[1] Chwan-Hwa Wu[2] J. D. Irwin[3]
September 8, 2003

Abstract

Forward-secure signatures are used to defeat signature forgeries in cases of key exposure. In this model, the signature key evolves with time and it is computationally infeasible for an adversary to forge a signature for some time-period prior to the key's exposure.

In this paper a new forward-secure digital signature scheme is presented, which is based on the use of *bilinear maps* recently advocated by Boneh and Franklin [9]. This scheme is efficiently constructed and can be used with a large number of time periods with a log magnitude complexity. The signing and key-update operations are very efficient when compared with other previously available schemes. A formal definition, as well as a detailed analysis of the security performance or this scheme, is presented. The security proof for this scheme is based on the Computational Diffie-Hellman assumption, which leads to a unique approach to proving security in the random oracle model. Furthermore, within the proof both the hash oracle and the signing oracle are constructed in an innovative manner.

1 Introduction

1.1 The Key Exposure Problem

The Key exposure problem has been classified as one of the biggest problems for a security system. In a conventional system, the system security is completely compromised once the key is exposed. Together with the increasing use of small, easy-to-lose, handheld devices, the key exposure problem is rapidly becoming more prevalent. Since users are employing their portable devices for a variety of transactions, cryptographic computations are inevitably required. While exposure of the key is certainly of great concern to the user, corporations and services providers must also employ stronger security measures in their systems in order to protect themselves from key exposure resulting from either malicious break-in or unintentional leakage.

To address this problem, several different approaches have been suggested. Many of them try to minimize exposure of the secret by splitting it and storing the parts in different places, usually via secret sharing [8, 30]. There are many follow-ups to this idea, including a threshold signature [13] and proactive secret sharing [22, 23]. However, as indicated in [5], distribution can be quite costly and not a viable option for the average user. The highly

* This is the full version of our paper.

[1] Department of Electrical & Computer Engineering, Auburn University, Auburn, AL 36849, USA. Email: feihu@eng.auburn.edu. Supported by Department of ECE, Auburn university.

[2] Department of Electrical & Computer Engineering, Auburn University, Auburn, AL 36849, USA. Email: wu@eng.auburn.edu

[3] Department of Electrical & Computer Engineering, Auburn University, Auburn, AL 36849, USA. Email: jdirwin@eng.auburn.edu

cooperative and interactive nature of the threshold schemes also makes the communication and computation burden prohibitively high for wireless devices.

1.2 The Forward-secure Signature

The term (perfect) "*forward secrecy*" was first used by Günther [21] within the context of session key exchange protocols, and again later in [14]. The basic idea is simply that compromising a long-term key does not compromise past session keys; therefore, action in the past is in some way protected against a loss of the current key. Anderson suggested this paradigm and the idea of a digital scheme with forward security in an invited lecture presented at the year 1997 ACM CCS conference [3]. It was first formalized by Bellare and Miner in the context of a *forward-secure signature scheme* [5].

The basic idea presented in [5] is the use of a *key-evolving signature scheme* whose operation is divided into time-periods, with a different secret key being used for each time-period. Each secret key is used to sign the message in the current time-period and derive the secret key for the next time-period. Like the ordinary signature scheme, the public key is constant for all time-periods. A verification scheme checks both the signature's validity and time-period. The signature scheme is forward-secure because it is impossible for an adversary to forge a signature for a previous time-period even if it obtains the current secret key.

Following the initial work by [5], a sequence of other deviations of the forward-secure signature [1, 27, 24, 28] was suggested. In [1], an improved forward-secure signature scheme with much shorter keys than those outlined in [5] was proposed. Krawczyk [27] suggested a method for constructing a forward-secure scheme from any signature scheme, and thus made the forward security of standard signature schemes (RSA, DSS) possible. Itkis and Reyzin [24] proposed another forward-secure signature scheme based on the Guillou-Quisquater signature. It provides efficient signing, verifying and storage, but the price tag for these features is a longer running time for the key generation and update routine. Maklin [28] proposed a generic construction that can be based on any underlying signature with an unlimited total number of time-periods.

Integrating forward-secure signature with threshold techniques has also been investigated [2, 31]. The forward-secure encryption scheme considered in [7] and [12] focused on private and public key cryptography respectively. Key insulated cryptography [16] and intrusion resilient cryptography [25, 17] were recently introduced to achieve a higher level of security. However, these two approaches require time synchronization and interaction between the device and server for each time-period. Therefore, they may not be applicable in some scenarios.

1.3 Our Contribution

The scheme outlined in this paper extends the work by Gentry-Silverberg [20], Canetti-Halevi-Katz [12] and Dodis etc. [17], and is the first construction of forward-secure signatures based on bilinear maps. Although the underlying signature scheme (HIDS) for this approach was initially considered by Gentry and Silverberg [20], their scheme works in a different context and is *not* forward-secure. Furthermore, the authors have not provided any security proof for their scheme (HIDS). Inspired by the works in [12, 17], we construct a forward-secure signature scheme based on the scheme outlined in [20]. The security of this new approach is based on the Computational Diffie-Hellman assumption, and leads to a unique way of proving security in the random oracle model. Furthermore, within our proof, the hash oracle and the signing oracle are constructed in an innovative manner in order to accommodate the bilinear map.

Our scheme is very efficient in terms of the total number of time periods, T, and all of our parameters have a complexity no larger than $O(\log T)$. Therefore, when compared with other forward secure signature schemes [5, 1, 24], our scheme is especially useful in scenarios where frequent key updates or long operational system times, are required. The signing and key update operations in our scheme are very simple, and only require one addition and one

multiplication on the additive group $\mathbb{G}_1$ – on average for key update. Verification only requires $O(\log T)$ mapping operations and multiplications. Because our scheme is based on bilinear maps that can be constructed from Weil and Tate pairings on an elliptic curve, we are able to work on a much smaller finite field and hence achieve both smaller key sizes and signature sizes, when T is relatively small. Since complexity is related to $\log T$, increases in size are very slow as T increases.

1.4 Outline of the Paper

Our approach is first to provide precise definitions for the key-evolving signature scheme and its forward security. Then some mathematical background in bilinear maps is provided. In section 3, a detailed description of the scheme and its security analysis is given. Some features of the scheme are given in section 4. Finally, the appendix provides a detailed security proof.

2 Definitions

In this section, we define the key-evolving signature scheme and the formal notion of forward security in the random oracle model. All these definitions closely follow those given in [1], which, in turn, are based on the first formal definition of forward-secure signature proposed by Bellare and Miner [5]. We also present the definition of the Computational Diffie-Hellman assumption and review the preliminaries of bilinear maps, since our scheme is based on them.

2.1 The Forward-secure Digital Signature Scheme

The approach employed by forward-secure digital signature schemes involves updating the secret key periodically. Therefore, a forward-secure signature scheme is, first, a key-evolving signature scheme. The operational time of a forward-secure signature scheme is divided into time-periods. As in standard signature schemes, it consists of a key generation algorithm, a signing algorithm and a verification algorithm. In addition, a key update algorithm is needed to update the secret key from one time-period to another, based on the secret key used in the current time-period. Forward security results from the fact that the update algorithm is a one-way function and it is very difficult for an adversary to recover previous secret keys even if the secret key in the current time-period is known [4].

The signing algorithm signs a message with the secret key for that specific time-period. However, the public key remains constant for all time-periods. The verification algorithm verifies not only the validity of the signature but also the time-period in which the message is signed.

Definition 2.1 [Key-evolving Signature Scheme]. A *key-evolving digital signature scheme* is an algorithm with the quadruple, $\mathsf{KE\text{-}SIG} = (\mathsf{Gen}, \mathsf{Upd}, \mathsf{Sig}, \mathsf{Ver})$, such that:

1. Gen, the *key generation* algorithm, is a probabilistic algorithm that inputs a security parameter $k \in N$ (given in unary as 1^k) and the total number of time-periods T, and returns a public key PK and the initial secret key SK_0.
2. Upd, the *secret key update* algorithm, accepts as input the secret key SK_i for the current time-period, and returns the new secret key SK_{i+1} for the next time-period.

[4] Obviously, it is important for the signer, i.e. key owner, to properly destroy the “old" secret key. Otherwise, an adversary who obtains the “old" secret key can easily forge the signature for previous time-periods. As pointed out by Itkis and Reyzin [24], secure deletion of the secret key is not a trivial task, and it is not unreasonable to insist that the signer guarantee the deletion operation rather then guaranteeing their placement in some “safe" place.

This algorithm is usually deterministic.

3. Sig, the *signing* algorithm, accepts as input the secret key SK_i in the current time-period and a message M. It returns a pair $\langle i, sign \rangle$, that represents the signature of M and time-period index i. This algorithm may be probabilistic.
4. Ver, the *verification* algorithm, accepts as input the public key PK, a message M and a candidate signature $\langle i, sign \rangle$, and returns 1 if the signature of M is valid or 0, otherwise. This algorithm is typically deterministic.

The verification algorithm is required to verify that a signature of M, generated via $\mathsf{Sig}_{SK_i}(M)$, is valid for period i. For convenience, it is also assumed that the secret key SK_i for time-period $i \in \{0,...,T-1\}$, always contains both the value i and the total number of periods T. For the last time period $T-1$, $\mathsf{Upd}(SK_{T-1})$ returns the empty string SK_T.

Since we work in the random oracle model, all algorithms in $\mathsf{KE\text{-}SIG}$ will also have oracle access to a public hash function H, which is assumed *random* in our security analysis. The reason for this stipulation will now be addressed.

SECURITY ANALYSIS USING THE RANDOM ORACLE MODEL. Our work is performed in the random oracle (RO) model, which is widely used and was explicitly formulated by Bellare and Rogaway [6]. It assumes that all parties, including adversaries, have oracle access to a *truly random function*. In implementing an ideal system, the random oracle must be replaced with some "cryptographic hash function", e.g., MD5 or SHA, since a truly random function does not exist in the real world. The RO model, i.e. its methodology, provides a practical and efficient way to design cryptographic protocols and schemes.

The security of the signature scheme should guarantee that it is computationally infeasible for any adversary to forge a signature message pair without knowledge of the secret key for that time-period. The possibility of key exposure, caused by adversarial break-in, must also be considered. Since we consider the time period starting at zero, we need modify slightly the security model suggested by Abdalla and Reyzin [1] to suit the needs.[5]

In this model, an adversary knows the public key PK, the total number of time-periods T and the current time-period. For the key-evolving signature scheme $\mathsf{KE\text{-}SIG} = (\mathsf{Gen}, \mathsf{Upd}, \mathsf{Sig}, \mathsf{Ver})$, the adversary F is functioning in three stages: the chosen-message attack (cma) phase, the break-in (breakin) phase, and the forgery (forge) phase. In the cma phase, F has access to the signing oracle, and can obtain the signature of any message it selects under the current secret key. The breakin phase is used to model the possible key exposure caused by an adversary F break-in. In such a case, F is given the current secret key SK_i. In the final forge phase, F outputs its forgery, i.e. a signature message pair. The adversary F is said to be successful if it forges the signature of some "new" message for some time-period prior to the break-in. Here, the term "new" message is used to indicate some message that has never been queried for the signature by the adversary. We use the following experiment to evaluate the success probability of breaking the forward-security of the signature scheme. The use of $k,\cdots,T$ indicates that the arguments of the key generation algorithm could be more than just k and T.

Experiment F-Forge-RO($\mathsf{KE\text{-}SIG}$, F)

Select $H : \{0,1\}^* \rightarrow \{0,1\}^l$ at random
$(PK, SK_0) \xleftarrow{R} \mathsf{Gen}^H(k, \cdots, T)$
$i \leftarrow 0$
`Repeat`

[5]The security model in [1] was modified using the security model by Bellare and Miner in [5] to work in the random oracle model.

$d \leftarrow F^{H, \mathrm{Sig}^H_{SK_i}(\bullet)}(\mathsf{cma}, PK)$; $SK_{i+1} \leftarrow \mathsf{Upd}^H(SK_i)$; $i \leftarrow i+1$;
`Until` $(d = \mathsf{breakin})$ `or` $(i = T)$
`If` $d \neq \mathsf{breakin}$ `and` $i = T$ `then` $i \leftarrow T+1$
$i \leftarrow i-1$
$(M, \langle b, sign \rangle) \leftarrow F^H(\mathsf{forge}, SK_i)$
`If` $\mathsf{Ver}^H_{PK}(M, \langle b, sign \rangle) = 1$ `and` $0 \le b < i$ `and` M was not queried of $\mathsf{Sig}^H_{SK_b}(\bullet)$ in period b
`Then return 1 else return 0`

In this formulation, it is understood that the state of F is preserved across its various invocations once we first pick and fix coin for it [5]. In this foregoing experiment, a random oracle H, i.e.the public hash function which is assumed to be random, is selected first; and then the key generation algorithm (Gen), with access to H-oracle[6], will generate both the public key and the secret key for time-period 0. In the chosen-message attack phase (cma), F queries the signing oracle ($\mathsf{Sig}^H_{SK_i}(\bullet)$) and the H -oracle as many times as it wants, and then outputs some value d to indicate it is finished. As long as d is not breakin , it proceeds to the next time-period. At some time i, F decides to break-in; it is then given the current secret key SK_i. If F does not break-in by the last time-period, it is given SK_T, which by definition is the empty string. F will then try to forge a signature for some new message M in time-period $b < i$. Depending upon the signature verification results, the experiment will return a 1 or 0 to indicate the success or failure of the adversary F.

Definition 2.2 [Forward-security in the Random Oracle Model]. Let KE-SIG = (Gen, Upd, Sig, Ver) be a key-evolving signature scheme, H a random oracle and F an adversary as described earlier. Let $\mathbf{Succ}^{fwsig}(\mathsf{KE\text{-}SIG}[k, \cdots, T], F)$ denote the probability that the experiment F-Forge-RO ($\mathsf{KE\text{-}SIG}[k, \cdots, T], F$) returns 1. Then the insecurity of KE-SIG is the function

$$\mathbf{Insec}^{fwsig}(\mathsf{KE\text{-}SIG}[k, \cdots, T]; t, q_{sig}, q_{hash}) = \max_F \{\mathbf{Succ}^{fwsig}(\mathsf{KE\text{-}SIG}[k, \cdots, T], F)\},$$

where the maximum is taken over all adversaries F for which the following condition holds: the execution time for the above experiment is at most t; F makes at most q_{sig} signing queries to the signing oracle and q_{hash} hash queries to the H-oracle .

2.2 Cryptographic Preliminaries and Assumptions

Let q be some large prime, and let g be a generator of some cyclic group $\mathbb{G}$ with prime order q. The security of our signature scheme is based upon the difficulties encountered in solving the CDH problem.

Definition 2.3 [CDH Assumption]. A probabilistic algorithm A is said to be (t, ε) *-break CDH* in a cyclic group $\mathbb{G}$ if A runs at most time t, computes the Diffie-Hellman function $DH_{g,q}(g^a, g^b) = g^{ab}$ with input (g, q) and (g^a, g^b) with a probability of at least ε, where the probability is over the coins of 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) *-breaks* CDH in this group.

BILINEAR MAPS. Our signature scheme is constructed using the bilinear map suggested by Boneh and Franklin [9]. Definitions of the terms employed in the use of bilinear maps are

[6] We adopt the convention to denote the access of the hash oracle and signing oracle as the superscript in expressions.

briefly outlined as follows. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of order q for some large prime q. A bilinear map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ is a one-way mapping between these two groups that has following properties:

1. The map is *bilinear*; that is, for all $P, Q \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}_q$, there is $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$.
2. The map is *non-degenerate*, i.e. the map does not send all pairs in $\mathbb{G}_1 \times \mathbb{G}_1$ to the identity in $\mathbb{G}_2$. In particular, because $\mathbb{G}_1$ and $\mathbb{G}_2$ are in prime order q, if P is a generator of $\mathbb{G}_1$ then $\hat{e}(P, P)$ is a generator of $\mathbb{G}_2$.
3. There is an efficient algorithm to compute the map $\hat{e}(P, Q)$ for any $P, Q \in \mathbb{G}_1$.

A *parameter generator* $\mathcal{IG}$ is defined as a randomized algorithm that takes a security parameter $k \in N$ (denoted as unary 1^k), runs in polynomial time k, and outputs the description of the two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ (of prime order q) and the bilinear maps $\hat{e}$, having the above properties. The description of $\mathbb{G}_1$, $\mathbb{G}_2$ contains the polynomial time (in k) algorithm for computing the group operation. k is used to determine the bit length of q.

To simplify the efficiency analysis with input security parameter k, we assume the computation for $\mathcal{IG}$ is a $O(k^n)$ bit operation, a group operation on $\mathbb{G}_1$ is at most a $O(k^{n_1})$ bit operation, a group operation on $\mathbb{G}_2$ is at most a $O(k^{n_2})$ bit operation and one mapping operation is a $O(k^e)$ bit operation. In this case, $n, n_1, n_2, e \in N$ are orders of the polynomial time algorithm and are determined by the constructing of the bilinear map.

Note that the parameter generator, that generates $\mathbb{G}_1$ and $\mathbb{G}_2$, which are (t, ε)-*CDH* groups, can be constructed from Weil and Tate pairings on elliptic curves or abelian varieties [9, 18, 26, 29, 32]. However, our scheme is independent of any specific construction.

3 The Proposed Forward-secure Signature Scheme

3.1 Scheme Overview

Our scheme employs the binary tree structure, which has been widely used by many researchers in cryptographic design. In [5], Bellare and Miner first suggested the possibility of using such a structure to form a forward secure signature scheme. It was also adopted in [20, 12, 17].

Following the approach in [12, 17], we define the total number of time-periods T to be a power of 2 ($T = 2^l$). Therefore, each time-period i ($0 \le i < 2^l - 1$) can be represented using a binary representation in l bits, that is, $\langle i \rangle = i_1 i_2 i_3 \cdots i_l$. We construct a full binary tree of height l to cover all time-periods. The *root* of the tree is called node ε. Each *leaf* of the tree is labeled with its binary representation $\langle i \rangle$ and denotes the time-period i. The time-period evolves from the leftmost leaf to the rightmost leaf with the first leaf representing time-period 0, the second leaf representing time-period 1, and so forth, culminating in the last time-period $2^l - 1$. All other *internal nodes*, i..e. a node that is not a root or leaf of the tree, are labeled with the binary representation of its position in j bits, where j is the depth of the node ($1 \le j < l$). We denote each of these internal nodes as ζ in short representation and $\zeta = \zeta_1 \zeta_2 \cdots \zeta_j$ in binary representation.

The root node ε contains the "root secret" s_ε and a "root verification point" Q. For each internal node ζ, we denote its left and right child node as $\zeta 0$ and $\zeta 1$ individually. Each of the nodes ζ also contain a "node secret" s_ζ, a "local secret" $S_\zeta \in \mathbb{G}_1$, and a "verification

point" Q_ζ. The "local secret key" for node ζ is $sk_\zeta = (S_\zeta, \mathcal{Q}_\zeta)$, where $\mathcal{Q}_\zeta = (Q_{\zeta_1}, Q_{\zeta_1\zeta_2}, \cdots, Q_{\zeta_1\zeta_2\cdots\zeta_{j-1}})$. Each leaf contains a "leaf secret" $s_{\langle i\rangle}$, a "local secret" $S_{\langle i\rangle} \in \mathbb{G}_1$ and a "verification point" $Q_{\langle i\rangle} \in \mathbb{G}_1$. The "local secret key" for a leaf is then $sk_{\langle i\rangle} = (S_{\langle i\rangle}, \mathcal{Q}_{\langle i\rangle})$, where $\mathcal{Q}_{\langle i\rangle} = (Q_{i_1}, Q_{i_1 i_2}, \cdots, Q_{i_1 i_2 \cdots i_{l-1}}, Q_{\langle i\rangle})$. Note that $Q_{\langle i\rangle}$ is also included in $\mathcal{Q}_{\langle i\rangle}$. All those "verification points" are necessary to verify the signature and are included in the signature when the user signs a message. However, they are not really "secrets" and can be revealed to the public. The secret keys have the following properties:

1. To sign a message during time-period i, one needs only $s_{\langle i\rangle}$ and $sk_{\langle i\rangle}$.
2. Given sk_ζ, secrets $sk_{\zeta 0}$ and $sk_{\zeta 1}$ can be efficiently derived.
3. If one has no knowledge of sk_ζ for any prefixes ζ of $\langle i\rangle$, then given PK and time-period i, it is infeasible to derive $sk_{\langle i\rangle}$.

The "global secret key" SK_i for time-period i consists of (1) $s_{\langle i\rangle}$, (2) $sk_{\langle i\rangle}$, and (3) $\{sk_{i_0 i_1 i_2 \cdots i_{j-1}1}\}$, for each $i_j = 0$, $1 \le j \le l$. The first two parts are the "leaf secret" and the "local secret key" for the leaf. Given the binary representation $\langle i\rangle = i_0 i_1 \cdots i_l$ (where i_0 is the root), for each index j $(1 \le j \le l)$ that provides $i_j = 0$, node $\zeta 0 = i_0 i_1 i_2 \cdots i_j$ has a right sibling node $\zeta 1 = i_0 i_1 i_2 \cdots i_{j-1} 1$. These right sibling nodes are important for key updates. Thus, the last part of SK_i is a set of "local secret keys" for all right sibling nodes for time period i. $\psi(i)$ is used to denote all these right sibling nodes for time period i.

To reduce the storage requirement, we utilize the same simplification techniques outlined in [17] and eliminate some redundant storage of "verification points" in $sk_{\langle i\rangle}$ and $\{sk_{i_0 i_1 i_2 \cdots i_{j-1}1}\}$. We decompose $sk_{\langle i\rangle}$, $\{sk_{i_0 i_1 i_2 \cdots i_{j-1}1}\}$ and represent SK_i as $\{s_{\langle i\rangle}, S_{\langle i\rangle}, \mathcal{S}_{\langle i\rangle}, \mathcal{Q}_{\langle i\rangle}\}$. Here $S_{\langle i\rangle}$ and $\mathcal{Q}_{\langle i\rangle}$ are decomposed from $sk_{\langle i\rangle}$; $\mathcal{S}_{\langle i\rangle} = \{S_\zeta \mid \zeta \in \psi(i)\}$ represents all the "local secrets" of those right sibling nodes for time period i. Other redundant storage of $\mathcal{Q}_\zeta$ is eliminated. Note that the "node secret" (including root node) s_ζ (or s_ε) is randomly selected and will be deleted as soon as the derivation of $S_{\zeta 0}, S_{\zeta 1}$ and Q_ζ (or S_0, S_1 and Q) is performed. Therefore, s_ζ is not a part of SK_i.

We can see from the foregoing description that it is easy to derive $sk_{\langle i"\rangle}$ for a later time-period $i" > i$ given the "global secret key" SK_i. However, by properly destroying the "old secrets", we can make the derivation of the previous secret key difficult for everyone, even the signer. In our scheme, $\mathcal{S}_{\langle i\rangle}$ is stored for key update purposes only. Given $\mathcal{S}_{\langle i\rangle}$, it is easy to update the current $S_{\langle i\rangle}$ to $S_{\langle i+1\rangle}$. Update of $\mathcal{Q}_{\langle i\rangle}$ and $\mathcal{S}_{\langle i\rangle}$ (to $\mathcal{Q}_{\langle i+1\rangle}$ and $\mathcal{S}_{\langle i+1\rangle}$) accompanies the update of $S_{\langle i\rangle}$ (to $S_{\langle i+1\rangle}$) naturally, where $s_{\langle i+1\rangle}$ is selected at random. The forward security of our signature scheme is based on the intractability of $S_{\langle i\rangle}$ after time-period i. Assuming the safe deletion of $S_{\langle i\rangle}$ at the end of time-period i, the one-way property of the key update algorithm guarantees the forward security of our scheme. Figure 1 shows the overall structure of the binary tree and an example for time period $i = 5$. Detailed algorithms will be presented in the following section.

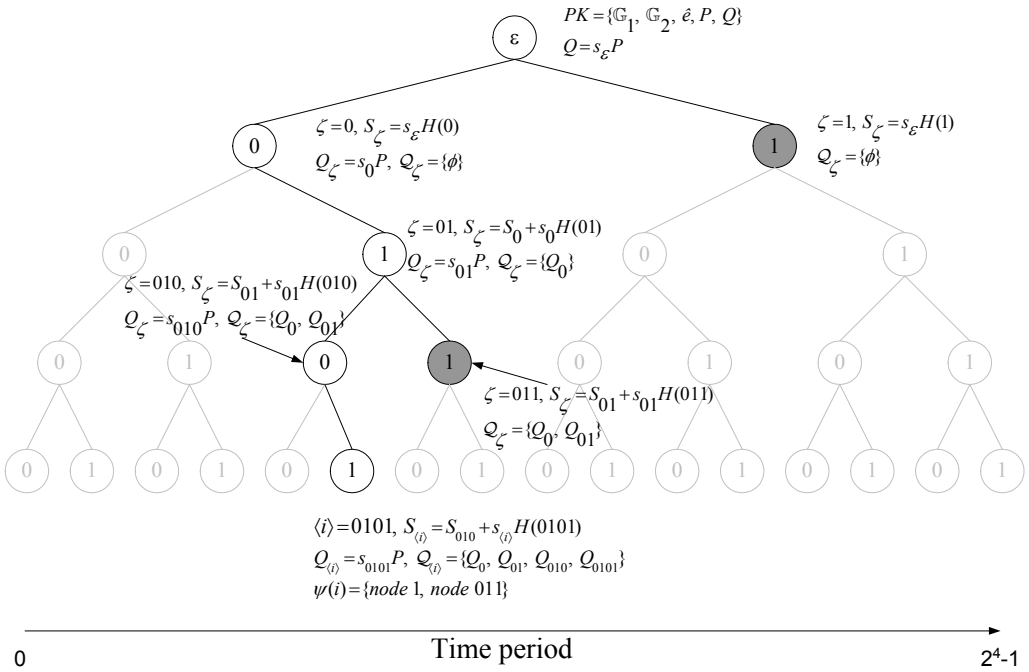

Figure 1. The overall structure of the binary tree and an example for time period $i=5$. The binary representation is $\langle i\rangle=0101$. Nodes in dark gray are right sibling nodes that are important for key update purposes. Other nodes in light gray are irrelevant at time period i.

3.2 Scheme Details

Our construction is based on the HIDS (hierarchical identity-based signature) in [20] and the ke-PKE in [12]. We emphasize that there is no security proof for HIDS in [20] and the proof in [12] does not fit our context directly, because our scheme is a signature, rather than an encryption, scheme.

We assume the hash function $H:\{0,1\}^*\rightarrow\mathbb{G}_1$ is defined either by Gen or other specifications of the scheme. H is public to all parties and assumed to be random. We use ϕ to denote an empty string and 0^j to denote j zeros. The details of our scheme are:

algorithm Gen(1^k, $T=2^l$)
  run $\mathcal{IG}(1^k)$ to generate $\mathbb{G}_1, \mathbb{G}_2$ (of prime order q) and $\hat{e}$
  select a random generator $P\leftarrow\mathbb{G}_1$; $s_\varepsilon\overset{R}{\leftarrow}\mathbb{Z}_q$; set $Q=s_\varepsilon P$
  public key $PK=\{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q\}$
  $S_0=s_\varepsilon H(0)$, $S_1=s_\varepsilon H(1)$; set $sk_0=(S_0, \phi)$, $sk_1=(S_1, \phi)$, $\mathcal{S}_{0^l}=\{S_1\}$
  For $j=1$ to $l-1$
    let $\zeta=0^j$; $(sk_{\zeta 0}, sk_{\zeta 1})=$ Extract(sk_ζ, ζ)
    parse $sk_{\zeta 0}=\{S_{\zeta 0}, \mathcal{Q}_{\zeta 0}\}$, $sk_{\zeta 1}=\{S_{\zeta 1}, \mathcal{Q}_{\zeta 1}\}$, $\mathcal{S}_{0^l}=\mathcal{S}_{0^l}\cup\{S_{\zeta 1}\}$
  select $s_{0^l}\overset{R}{\leftarrow}\mathbb{Z}_q$; set $Q_{0^l}=s_{0^l}P$; set $SK_0=\{s_{0^l}, S_{\zeta 0}, \mathcal{S}_{0^l}, \mathcal{Q}_{\zeta 0}\cup\{Q_{0^l}\}\}$ $(\zeta=0^{l-1})$
  delete all other intermediate values
return (PK, SK_0)

Subroutine Extract(sk_ζ, ζ)

parse $\zeta = \zeta_1\zeta_2\cdots\zeta_j$, where $|\zeta| = j$; parse $sk_\zeta = \{S_\zeta, \mathcal{Q}_\zeta\}$

$s_\zeta \xleftarrow{R} \mathbb{Z}_q$, $Q_\zeta = s_\zeta P$

$S_{\zeta 0} = S_\zeta + s_\zeta H(\zeta_1\zeta_2\cdots\zeta_j 0)$, $S_{\zeta 1} = S_\zeta + s_\zeta H(\zeta_1\zeta_2\cdots\zeta_j 1)$

set $sk_{\zeta 0} = \{S_{\zeta 0}, \mathcal{Q}_{\zeta 0}\}$, $sk_{\zeta 1} = \{S_{\zeta 1}, \mathcal{Q}_{\zeta 1}\}$, where $\mathcal{Q}_{\zeta 0} = \mathcal{Q}_{\zeta 1} = \mathcal{Q}_\zeta \cup \{Q_\zeta\}$

return ($sk_{\zeta 0}$, $sk_{\zeta 1}$)

The subroutine Extract is used to derive the "local secret key" for child nodes. Given the input sk_ζ and the node ζ, it returns the "local secret key" $sk_{\zeta 0}$, $sk_{\zeta 1}$. The Upd algorithm also uses this subroutine.

algorithm Upd(SK_i)

If $i = T-1$

$SK_{i+1} = \phi$; delete SK_i

else

let $\langle i\rangle = i_0 i_1 i_2 \cdots i_l$ where $i_0 = \varepsilon$ for convenience, parse SK_i as $\{s_{\langle i\rangle}, S_{\langle i\rangle}, \mathcal{S}_{\langle i\rangle}, \mathcal{Q}_{\langle i\rangle}\}$

If $i_l = 0$

get $S_{\langle i+1\rangle}$ from $\mathcal{S}_{\langle i\rangle}$; set $\mathcal{S}_{\langle i+1\rangle} = \mathcal{S}_{\langle i\rangle} - \{S_{\langle i+1\rangle}\}$ (remove $S_{\langle i+1\rangle}$ from $\mathcal{S}_{\langle i\rangle}$)

select $s_{\langle i+1\rangle} \xleftarrow{R} \mathbb{Z}_q$; set $Q_{\langle i+1\rangle} = s_{\langle i+1\rangle} P$; set $\mathcal{Q}_{\langle i+1\rangle} = (\mathcal{Q}_{\langle i\rangle} - \{Q_{\langle i\rangle}\}) \cup \{Q_{\langle i+1\rangle}\}$

set $SK_{i+1} = \{s_{\langle i+1\rangle}, S_{\langle i+1\rangle}, \mathcal{S}_{\langle i+1\rangle}, \mathcal{Q}_{\langle i+1\rangle}\}$; delete SK_i

else

find the largest j $(1 \le j < l)$ that gives $i_j = 0$

let $\eta = i_0 i_1 i_2 \cdots i_{j-1} 1$ where $i_0 = \varepsilon$ for convenience, we have $\langle i+1\rangle = \eta 0^{l-j}$

get S_η from $\mathcal{S}_{\langle i\rangle}$; set $\mathcal{S}_{\langle i+1\rangle} = \mathcal{S}_{\langle i\rangle} - \{S_\eta\}$ (remove S_η from $\mathcal{S}_{\langle i\rangle}$)

For $m = 1$ to $l - j$

let $\zeta = \eta 0^{m-1}$, ($sk_{\zeta 0}$, $sk_{\zeta 1}$) =Extract(sk_ζ, ζ),

parse $sk_{\zeta 0} = \{S_{\zeta 0}, \mathcal{Q}_{\zeta 0}\}$, $sk_{\zeta 1} = \{S_{\zeta 1}, \mathcal{Q}_{\zeta 1}\}$; $\mathcal{S}_{\langle i+1\rangle} = \mathcal{S}_{\langle i+1\rangle} \cup \{S_{\zeta 1}\}$

select $s_{\langle i+1\rangle} \xleftarrow{R} \mathbb{Z}_q$; set $Q_{\langle i+1\rangle} = s_{\langle i+1\rangle} P$

set $SK_{i+1} = \{s_{\langle i+1\rangle}, S_{\zeta 0}, \mathcal{S}_{\langle i+1\rangle}, \mathcal{Q}_{\zeta 0} \cup \{Q_{\langle i+1\rangle}\}\}$ ($\zeta = \eta 0^{l-j-1}$); delete SK_i

delete all other intermediate values

return SK_{i+1}

Depending on the input time period, the key update algorithm returns empty string (for the last time period) or proceed to calculate the secret key for the next time period. The calculation algorithm has two cases. If the l-th bit of $\langle i\rangle$ equals zero, it simply randomly selects $s_{\langle i+1\rangle}$, gets the local secret $S_{\langle i+1\rangle}$ from $\mathcal{S}_{\langle i\rangle}$, deletes $S_{\langle i+1\rangle}$ from $\mathcal{S}_{\langle i\rangle}$ to avoid redundant storage and updates $\mathcal{Q}_{\langle i\rangle}$ to $\mathcal{Q}_{\langle i+1\rangle}$ by replacing the "verification point" for the leaf. If the l-th bit of $\langle i\rangle$ is not equal to zero, it finds the largest index j that produces $i_j = 0$, the corresponding right sibling node η, and derives $S_{\langle i+1\rangle}$ from S_η. The update of $\mathcal{S}_{\langle i\rangle}$ and $\mathcal{Q}_{\langle i\rangle}$ follows the derivation of $S_{\langle i+1\rangle}$, and $s_{\langle i+1\rangle}$ is selected at random.

algorithm $\mathsf{Sig}_{SK_i}(M)$

parse $\langle i\rangle = i_1 i_2 \cdots i_l$, $SK_i = \{s_{\langle i\rangle}, S_{\langle i\rangle}, \mathcal{S}_{\langle i\rangle}, \mathcal{Q}_{\langle i\rangle}\}$

let $V = S_{\langle i\rangle} + s_{\langle i\rangle} H(i_1 i_2 \cdots i_l M)$ and $sign = \{V, \mathcal{Q}_{\langle i\rangle}\}$

return $\langle i, sign\rangle$

algorithm $\mathsf{Ver}_{PK}(M, \langle i, sign\rangle)$

parse $\langle i\rangle = i_1 i_2 \cdots i_l$, $sign = \{V, \mathcal{Q}_{\langle i\rangle}\}$ and $\mathcal{Q}_{\langle i\rangle} = \{Q_{i_1}, Q_{i_1 i_2}, \cdots, Q_{i_1 i_2 \cdots i_{l-1}}, Q_{\langle i\rangle}\}$

let

$$X = \frac{\hat{e}(P, V)}{\prod_{j=2}^{l} \hat{e}(Q_{i_1 i_2 \cdots i_{j-1}}, H(i_1 i_2 ... i_j))}; \quad Y = \hat{e}(Q, H(i_1)) \cdot \hat{e}(Q_{\langle i\rangle}, H(i_1 i_2 \cdots i_l M))$$

If $X = Y$ then return 1 (true)

else return 0 (false)

The following analysis shows how to verify if $sign$ is a valid signature for time-period i and message M. If the signature is valid, we have

$$X = \frac{\hat{e}(P,V)}{\prod_{j=2}^{l} \hat{e}(Q_{i_1 i_2 \cdots i_{j-1}}, H(i_1 i_2 ... i_j))} = \frac{\hat{e}(P, s_\varepsilon H(i_1) + \sum_{j=2}^{l} s_{i_1 i_2 \cdots i_{j-1}} H(i_1 i_2 \cdots i_j) + s_{\langle i\rangle} H(i_1 i_2 \cdots i_l M))}{\prod_{j=2}^{l} \hat{e}(P, H(i_1 i_2 \cdots i_j))^{s_{i_1 i_2 \cdots i_{j-1}}}}$$

$$= \frac{\hat{e}(P, s_\varepsilon H(i_1)) \cdot \prod_{j=2}^{l} \hat{e}(P, s_{i_1 i_2 \cdots i_{j-1}} H(i_1 i_2 \cdots i_j)) \cdot \hat{e}(P, s_{\langle i\rangle} H(i_1 i_2 \cdots i_l M))}{\prod_{j=2}^{l} \hat{e}(P, H(i_1 i_2 \cdots i_j))^{s_{i_1 i_2 \cdots i_{j-1}}}}$$

$$= \frac{\hat{e}(P, s_\varepsilon H(i_1)) \cdot \prod_{j=2}^{l} \hat{e}(P, H(i_1 i_2 \cdots i_j))^{s_{i_1 i_2 \cdots i_{j-1}}} \cdot \hat{e}(P, s_{\langle i\rangle} H(i_1 i_2 \cdots i_l M))}{\prod_{j=2}^{l} \hat{e}(P, H(i_1 i_2 \cdots i_j))^{s_{i_1 i_2 \cdots i_{j-1}}}}$$

$$= \hat{e}(P, s_\varepsilon H(i_1)) \cdot \hat{e}(P, s_{\langle i\rangle} H(i_1 i_2 \cdots i_l M)) = \hat{e}(Q, H(i_1)) \cdot \hat{e}(Q_{\langle i\rangle}, H(i_1 i_2 \cdots i_l M)) = Y$$

Thus, the verification succeeds (algorithm $\mathsf{Ver}_{PK}(M, \langle i, sign\rangle)$ returns 1).

3.3 Efficiency Analysis

SIGNING AND VERIFYING. The signing operation in our scheme requires only one multiplication and addition on $\mathbb{G}_1$, which uses $O(k^{n_1})$ bit operations. The total signing computation is independent of the total number of the time periods, T. The verification requires more computation because it involves the pairing computation. To verify a signature one must do $O(\log T)$ pairing operations and $O(\log T)$ multiplications or division on $\mathbb{G}_2$. Therefore, the total computation for verification is $O(k^e \log T + k^{n_2} \log T)$ bit operations, and the complexity is $O(\log T)$ solely in terms of T.

KEY GENERATION. The key generation only requires operations on $\mathbb{G}_1$. It requires 1 multiplication to generate Q in the public key ($O(k^{n_1})$ bit operations), $\log_2 T$ multiplications

and $\log_2 T - 1$ additions to generate S_{0^l} ($O(k^{n_1} \log T)$ bit operations), another $\log_2 T$ multiplications and $\log_2 T - 1$ additions to generate $\mathcal{S}_{0^l}$ ($O(k^{n_1} \log T)$ bit operations), and an additional $\log_2 T$ multiplications to generate $\mathcal{Q}_{0^l}$ ($O(k^{n_1} \log T)$ bit operations). In addition, the parameter generator $\mathcal{IG}$ runs in polynomial time in $O(k^n)$. Therefore, the total computation is $O(k^{n_1} + k^{n_1} \log T + k^n)$ bit operations. The total computation complexity in terms of T is $O(\log T)$.

KEY UPDATE. Key generation only requires operations on $\mathbb{G}_1$. In the worst case, the key update takes up to $\log_2 T$ multiplications to generate $\mathcal{Q}_{\langle i+1 \rangle}$, $\log_2 T - 1$ multiplications and $\log_2 T - 1$ additions to generate $S_{\langle i+1 \rangle}$, another $\log_2 T - 1$ multiplications and $\log_2 T - 1$ additions to generate $\mathcal{S}_{\langle i+1 \rangle}$. The computation is $O(k^{n_1} \log T)$ bit operations. Considering all time periods, the average computation is reduced to $O(k^{n_1})$ and the complexity, in terms of T, is reduced to $O(1)$.

SIZES. Assuming an element in either the public or private key, with the exception of the description for groups and the map $\hat{e}$, is represented in $O(k)$ bits, the public key size is then $O(k)$. The complexity in terms of T is then $O(1)$. The private key contains a $O(k)$ bits $s_{\langle i \rangle}$, a $O(k)$ bits $S_{\langle i \rangle}$, a $O(k) \log_2 T$ bits $\mathcal{Q}_{\langle i \rangle}$, and an average $O(k) \log_2 T / 2$ bits of $\mathcal{S}_{\langle i \rangle}$. Therefore, it has the size of $O(k \log T + k)$, which is $O(\log T)$ in terms of T. The signature contains a $O(k)$ bits V and all the $\mathcal{Q}_\zeta$. The size is $O(k \log T + k)$, which is $O(\log T)$ in terms of T.

Table 3-1 summarizes our results.

Key generation time	$O(k^{n_1} + k^{n_1} \log T + k^n)$	$O(\log T)$
Signing time	$O(k^{n_1})$	$O(1)$
Verification time	$O(k^e \log T + k^{n_2} \log T)$	$O(\log T)$
Key update time	$O(k^{n_1})$	$O(1)$
Public key size	$O(k)$	$O(1)$
Private key size	$O(k \log T + k)$	$O(\log T)$
Signature size	$O(k \log T + k)$	$O(\log T)$

Table 3-1 A Complexity Summary of our signature scheme**.**

3.4 Security Analysis

The following theorem provides an upper bound on the insecurity of our signature scheme. Since the security of our scheme is based on a different cryptographic assumption, i.e. the CDH assumption, than previous works, we have a unique method for proving security in the random oracle model. Both the hash oracle and the signing oracle are constructed in an innovative manner, and the technique employed in the proof uses ideas from [1, 5, 12].

Theorem 3.1. If there exists a forger F that runs in time at most t, asking at most q_{hash} hash queries and q_{sig} signing queries, such that $\mathbf{Succ}^{fwsig}(\mathsf{KE}\text{-}\mathsf{SIG}[k, \cdots, T], F) > \varepsilon$ then there exists a adversary A that (t', ε')*-break CDH* in group $\mathbb{G}_1$ where

$$t' = t + O(k^{n_1} + k^2 \log T)$$
$$\varepsilon' = 1/T \cdot 1/(q_{hash} + q_{sig} + 1) \cdot \varepsilon$$

Proof idea: To break CDH in the additive group $\mathbb{G}_1$ with the order of q, A is given P (a random generator of $\mathbb{G}_1$), $P' = aP$ and $Q' = bP$, where $a, b \in \mathbb{Z}_q$ is randomly chosen and remains unknown to A. The target of A is to derive $S' = abP$ with the help of the forger F. To derive S', A runs F as the subroutine. A provides F the public key and answers its hash queries, signing queries and breakin query. Note that F has no other means of obtaining answers to these queries and cannot verify that the hash answers are the same as those given by the public hash function H. A appears to be the real signer to F, as long as the signatures and the secret key, answered by A, are verifiable using the related hash answers from A. A embeds the CDH problem into the public key and then answers the foregoing queries. After F forges a signature successfully, A is able to derive the answer to the CDH problem using the forged signature.

The procedure for A is briefly described as follows. First of all, A guesses a random index i ($0 \le i < T$), hoping the forger F will ask for the breakin query in time period i. A then generates public key $PK = \{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q\}$ with $Q = Q'$. A gives the public key to F and runs from the time period 0 to $T-1$ (as in the Experiment F-Forge-RO(KE-SIG, F)). A can easily answer the hash queries and signing queries from F because A completely controls the hash oracle. A embeds P' into some hash entries and provides them to F in its queries. During the execution, A must make a second guess of the random index g', hoping the final forgery is based on the g'-th hash query. A sets this hash entry specifically. Suppose all the guessing is correct and forger F finally outputs a signature on message $M_{g'}$ for some time period $i' < i$. A is then able to derive $S' = abP$ from this signature utilizing the non-degenerate property of the bilinear map $\hat{e}$, the "verification points" in time period i' and related hash entries. The lower bound probability to solve the CDH problem is derived from the lower bound probability of a successful forgery.

Due to space limitations, a detailed proof is included in Appendix A.

Theorem 3.2 Let KE-SIG[k, T] represent our key evolving signature scheme with modulus size k and number of time periods T. Then for any t, q_{sig} and q_{hash},

$$\mathbf{Insec}^{fwsig}(\text{KE-SIG}[k, T]; t, q_{sig}, q_{hash}) \le T(q_{hash} + q_{sig} + 1)\mathbf{Insec}^{cdh}(k, t')$$

where $t' = t + O(k^{n_1} + k^2 \log T)$.

Proof: The insecurity function is computed simply by solving for the function in Theorem 3.1 and then represent ε as ε'

$$\varepsilon' = 1/T \cdot 1/(q_{hash} + q_{sig} + 1) \cdot \varepsilon \Rightarrow$$
$$T(q_{hash} + q_{sig} + 1)\varepsilon' = \varepsilon$$

Theorem 3.2 follows.

4 Discussions

The scheme proposed in this paper can be readily incorporated with PKI. The system parameters $\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P$ can be published, and Q is bound to the signer by a certificate. In comparison to previous forward secure signature schemes [5, 1, 24], our scheme has the advantage that no parameter has a complexity value of more than $\log T$, where T is the total number of time periods. In [5, 1], the signing and verification times are both linear in T. For the scheme in [24], the signing and verification are fast and independent of T, but their key

generation and key update times are still linear in T. This advantage ensures that our scheme is particularly useful for systems requiring frequent key updates or a long operating time. Furthermore, in some cases, the signature size of our scheme can even be reduced to $O(k)$. For example, for some verifier who has previously received a signature for the same time period, the transmission of "verification points" Q can be neglected, and the verifier can simply use those "verification points" stored for the previous signature.

The efficiency of our scheme is balanced across all its aspects. The efficient signing and verification (in terms of T) comes at the cost of signature and private key sizes. All these sizes are not independent to T (has a $\log T$ complexity). However, a distinct feature of our scheme is that it is based on bilinear maps that can be constructed from Weil or Tate pairings on an elliptic curve. Our scheme can work on a smaller finite field and, as a result, achieves a shorter signature size than other known forward secure signature schemes [5, 1, 24] when the total number of time period is small (e.g. $T \leq 64$). These other schemes, which are based on the integer factoring problem or a strong RSA assumption, normally have a 1024-bit group size. Using the same bilinear map construction in the BLS signature scheme [11], we can achieve a 171 bit signature with 1024-bits security in RSA based approach, when T=1. Furthermore, the $\log T$ complexity also guarantees that a size increase is very slow when T increases.

The signing and key update operations in our scheme require only one addition and one multiplication (on average), which is very efficient when compared with previous schemes. The efficiency of our signing and key update operations is comparable to a signing operation in the BLS signature scheme [11] if the bilinear map is constructed in the same manner. The BLS signing operation requires only one multiplication on the elliptic curve. However, an addition operation on the elliptic curve is much faster than a multiplication operation and can be neglected in this approximation. From the implementation results in [4], we find that a BLS signature generation is faster than a RSA signature generation. Therefore, we can be assured that our scheme is able to achieve more efficient signing and key updates when compared with other known forward secure signature schemes [5, 1, 24], whose signature generation or key update requires more computation than that involved in one RSA signature generation.

Signature verification in our scheme is still limited by the efficient computation involved in pairing operations. However, the results shown in [4, 19] have demonstrated a lot of progress in this area. From [4], the computation time for a Tate pairing with the prime field size of 512 bits (with preprocessing) is now comparable to one RSA signing operation with a 1024 bits modular and a 1007 bits exponent. We believe our scheme has a great potential and should aid the progress of research in this field.

With its inherent features, our scheme is applicable in a number of areas. For example, for mobile device authentication, the signature verification is done at the server side, which has more computation power to do the pairing operations. Signature generation can be done efficiently on the mobile device. The forward secure nature of the scheme provides a stronger security guarantee and the $\mathrm{Log}T$ complexity also enables frequent key updates or longer system reset times in scenarios where they are needed.

5 Conclusions

In this paper, we have proposed the first forward secure signature scheme based on bilinear maps. Our scheme is efficiently constructed with a complexity of no more than $O(\log T)$, with flexibility based on the underlying bilinear map. We also provide a detailed performance analysis based on the complexity assumptions of the underlying construction. The security of our scheme is based on the Computational Diffie-Hellman assumption and is unique in comparison to other approaches. We provide the formal definitions and the security proofs for our scheme in the random oracle model, in which the hash oracle and signing oracle are constructed in an innovative manner that accommodates the bilinear map. Our scheme offers

an efficient signing and key update operation when compared to previous schemes. Although signature verification is still limited by the efficiency of the paring operations, our signature scheme is useful in many applications due to its enhanced scalability from the standpoint of time periods, efficient signing and key updates.

References

1. M. Abdalla and L. Reyzin, "A new forward-secure digital signature scheme", *Advance in Cryptology – ASIACRYPT 2000*, Vol. 1976 of Lecture Notes in Computer Science, T. Okamoto ed., pp. 116–129, Springer-Verlag, 2000.
2. M. Abdalla, S. Miner, and C. Namprempre, "Forward-secure threshold signature schemes", *Topics in Cryptology – CT-RSA 2001,* Vol. 2020 of Lecture Notes in Computer Science, D. Naccache ed., pp. 441–456, Springer-Verlag, 2001.
3. R. Anderson, "Two remarks on public-key cryptology", Invited lecture, CCCS'97, Available at `http://www.cl.cam.ac.uk/users/rja14/.`
4. P. Barreto, H. Kim, B. Lynn, and M. Scott, "Efficient algorithms for pairing-based cryptosystems", *Advance in Cryptology – CRYPTO 02*, Vol. 2442 of Lecture Notes in Computer Science, pp. 354–368, Springer-Verlag, 2002.
5. M. Bellare and S. Miner, "A forward-secure digital signature scheme", *Advance in Cryptology – CRYPTO 99 proceedings*, Vol. 1666 of Lecture Notes in Computer Science, M. Wiener ed., pp. 431–448. Springer-Verlag, 15-19 August 1999.
6. M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm in designing efficient protocols", In *First Annual Conference on Computer and Communications Security, ACM*, pp. 62–73, 1993.
7. M. Bellare and B. Yee, "Forward-security in private-key cryptography", *Topics in Cryptology – CT-RSA 2003*, vol. 2612 of Lecture Notes in Computer Science, M. Joye ed, pp. 1–18, Springer-Verlag, 2003.
8. G. R. Blakley, "Safeguarding cryptographic keys", *Proceedings of AFIPS 1979 National Computer Conference*, Vol. 48, pp. 313–317, 1979.
9. D. Boneh and M. Franklin, "Identity-based encryption from the Weil pairing", *SIAM Journal on Computing*, Vol. 32, No. 3, pp. 586-615, 2003.
10. D. Boneh, "The decision Diffie-Hellman problem", *Proceedings of the Third Algorithmic Number Theory Symposium*, Vol. 1423 of Lecture Notes in Computer Science, Joe P. Buhler ed., pp 48–63, Springer Verlag, 1998.
11. D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing", In *Advances in Cryptology - ASIACRYPT 2001*, LNCS Vol. 2248, pp. 514-532, Springer-Verlag, 2001.
12. R. Canetti, S. Halevi, and J. Katz, "A forward-secure public-key encryption scheme", To appear in *Advances in Cryptology – Eurocrypt '03*, At `http://eprint.icar.org/2003/083/.`
13. Y. Desmedt and Y. Frankel, "Threshold cryptosystems", *Advances in Cryptology – CRYPTO 89 proceedings*, Vol. 435 of Lecture Notes in Computer Science, G. Brassard ed., pp. 307–315, Springer-Verlag, 1989.
14. W. Diffie, P. van Oorschot, and M. Wiener, "Authentication and authenticated key exchange", *Designs, Code, and Cryptography*, Vol. 2, pp. 107–125, 1992.
15. W. Diffie and M. Hellman, "New directions in cryptography", *IEEE Transactions on Information Theory*, 22(6), pp. 644–654, November 1976.
16. Y. Dodis, J. Katz, S. Xu, and M. Yung, "Key-insulated public key cryptosystems", *Advances in Cryptology – Eurocrypt 2002*, Vol. 2332 of Lecture Notes in Computer Science, L. Knudsen ed., pp. 65–82, Springer Verlag, 2002.
17. Y. Dodis, M. Franklin, J. Katz, A. Miyaji, and M. Yung, "Intrusion resilient public-key encryption", *Topics in Cryptology – CT-RSA 2003*, Vol. 2612 of Lecture Notes in Computer Science, M. Joye ed, pp. 19–32, Springer-Verlag, 2003.
18. S. Galbraith, "Supersingular curves in cryptography", *Advances in Cryptology – Asiacrypt 2001*, Vol. 2248 of Lecture Notes in Computer Science, pp. 495–513, Springer-Verlag, 2001.
19. S. Galbraith, K. Harrison and D. Soldera, "Implementing the tate pairing", *Algorithm Number Theory Symposium – ANTS V*, Vol 2369 of Lecture Notes in Computer Science, pp 324–337, Springer-Verlag, 2003.
20. C. Gentry and A. Silverberg, "Hierarchical ID-Based Cryptography", *Advances in Cryptology – Asiacrypt 2002*, Vol. 2501 of Lecture Notes in Computer Science, Y. Zheng ed., pp. 548–566, Springer-Verlag, 2002.

21. C. Günther, "An identity-based key-exchange protocol", *Advances in Cryptology – Eurocrypt'89*, Vol. 434 of Lecture Notes in Computer Science, J-J. Quisquater and J. Vandewille ed., pp. 29–37, Springer Verlag, 1989.
22. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive secret sharing or how to cope with perpetual leakage," *Advances in Cryptology - CRYPTO '95* Vol. 963 of Lecture Notes in Computer Science, D. Coppersmith ed., pp. 339–352, Springer, 1995.
23. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive public key and signature scheme", *Proceedings of the 4th ACM Conference on Computer and Communications Security*, pp. 100–110, ACM press, 1997.
24. G. Itkis and L. Reyzin, "Forward-secure signatures with optimal signing and verifying", *Advances in Cryptology-CRYPTO 2001*, Vol. 2139 of Lecture Notes in Computer Science, J. Kilian, ed., pp. 332–354, Springer-Verlag, 2001.
25. G. Itkis and L. Reyzin, "SiBIR: Signer-base intrusion-resilient signatures", *Advances in Cryptology – CRYPTO 2002*, Vol. 2442 of Lecture Notes in Computer Science, M. Yung ed., pp 499–514, Springer-Verlag, 2002.
26. A. Joux, "The Weil and Tate pairing as building blocks for public key cryptosystems", in *Proceeding of Fifth Algorithmic Number Theory Symposium*, Vol. 2369 of Lecture Notes in Computer Science, pp. 22–32, Springer-Verlag, 2002.
27. H. Krawczyk, "Simple forward-secure signatures for any signature scheme", *Proceedings of the 7th ACM Conference on Computer and Communications Security*, pp. 108–115, ACM Press 2000.
28. T. Maklin, D. Micciancio and S. Miner, "Efficient generic forward-secure signatures with an unbounded number of time periods", *Advances in Cryptology – Eurocrypt 2002*, Vol. 2332 of Lecture Notes in Computer Science, L. Knudsen ed., pp. 400–417, Springer-Verlag, 2002.
29. K. Rubin, A. Silverberg, "Supersingular abelian varieties in cryptography", *Advances in cryptology – crypto 2002*, Vol. 2442 of Lecture Notes in Computer Science, pp. 336–353, Springer-Verlag, 2002.
30. A. Shamir, "How to share a secret", *Communications of ACM*, 22(11), pp. 612–613, 1979.
31. W. Tzeng and Z. Tzeng, "Robust forward-secure digital signature with proactive security", *Public Key Cryptography – Proceedings of PKC'01*, Vol. 1992 of Lecture Notes in Computer Science, K. Kim ed., pp. 264 –276, Springer-Verlag, 2001.
32. E. Verheul, "Evidence that XTR is more secure than supersingular elliptic curve cryptosystems", *Advances in cryptology – Eurocrypt 2001*, Vol. 2045 of Lecture Notes in Computer Science, pp. 195–210, Springer-Verlag, 2001.

A. Appendix --Proof of Theorem 3.1

Our proof technique is a combination of those given in [5, 1, 12]. However, some details can be simplified because we are working from a different context. The structure of the proof is very similar to those contained in [1], but the method is different because this scheme is based on different mathematical assumptions. Although the results in [20, 12] help illustrate how the tree-based scheme works, they do not result in our proofs because they concentrate on encryption schemes. The detailed proof to our forward secure signature scheme proceeds as follows.

As discussed in [1], we first assume that if F outputs a forgery $\langle j, sign \rangle$ for message M' and time period *j,* then the hash oracle has been queried on $(j_1 j_2 \cdots j_l M')$, where $j_1 j_2 \cdots j_l$ is the binary representation of *j*. This assumption is reasonable because any forger can be modified to do so. In addition, this may increase the number of hash queries to $q_{hash}+1$, because this hash query may not be part of the q_{hash} hash queries performed earlier by F. We further assume that if F asks for the signing query for some message M in some time period *j,* then the hash query $(j_1 j_2 \cdots j_l M)$ must also be requested simultaneously. Similar to our first assumption, any forger can be modified to do so and this may further increase the number of hash queries up to $q_{hash}+q_{sig}+1$. We also assume that F maintains all necessary bookkeeping and does not ask for the same hash query twice.

First of all, A has to guess the time period *i* at which F will ask for the breakin query. It

randomly selects i, $0 < i \le T$, hoping that the breakin query will occur at this time period. A then generates the public key $PK = \{\mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q\}$ in a manner similar to a real signer, but A sets $Q = Q' = bP$ directly. Note, that in this case, $s_\varepsilon = b$ is unknown to both A and F. A gives the public key to forger F, and runs from time period 0 to $T-1$ while maintaining all the "node secrets".

To answer the hash query and the signing query of F, A maintains two tables, a hash query table and a signing query table.

Hash queries are answered at random. Each entry in the hash table is a tuple $(\mathrm{I}, x, y, \Delta)$, in which I is the input value, (representing a node index when the number of bits of I is less than l), x is the random "exponent" for P, y is a partial "node/leave secret" (if necessary) and Δ is the output value. We use Δ_1 to denote the output value corresponding to the input I and $\Delta = H(\mathrm{I})$ to denote the relationship between the input and output of the hash table.

The hash table is initiated from time period i. A first randomly selects $x_{i_1 i_2}, x_{i_1 i_2 i_3}, \cdots, x_{i_1 i_2 \cdots i_l} \in \mathbb{Z}_q$, and $y_{i_1 i_2 \cdots i_{l-1}} \in \mathbb{Z}_q^*$ for time period i, and sets hash entry (i_1, ϕ, ϕ, P'), $(i_1 i_2, x_{i_1 i_2}, \phi, x_{i_1 i_2} P)$, $(i_1 i_2 i_3, x_{i_1 i_2 i_3}, \phi, x_{i_1 i_2 i_3} P) \ldots$, $(i_1 i_2 \cdots i_{l-1}, x_{i_1 i_2 \cdots i_{l-1}}, \phi, x_{i_1 i_2 \cdots i_{l-1}} P)$ and $(i_1 i_2 \cdots i_l, x_{i_1 i_2 \cdots i_l}, y_{i_1 i_2 \cdots i_{l-1}}, x_{i_1 i_2 \cdots i_l} P - y_{i_1 i_2 \cdots i_{l-1}}^{-1} P')$, where $P' = aP$ i.e. a part of the CDH problem that was given to A. Then, A finds all indexes j $(1 \le j \le l)$ such that $i_j = 0$, sets entry $(1, x_1, \phi, x_1 P)$ if $i_1 = 0$, and sets other hash entries (when $2 \le j \le l$ and $i_j = 0$) as $(i_1 i_2 \cdots i_{j-1} 1, x_{i_1 i_2 \cdots i_{j-1} 1}, y_{i_1 i_2 \cdots i_{j-1}}, x_{i_1 i_2 \cdots i_{j-1} 1} P - y_{i_1 i_2 \cdots i_{j-1}}^{-1} P')$. $x_{i_1 i_2 \cdots i_{j-1} 1} \in \mathbb{Z}_q$, and $y_{i_1 i_2 \cdots i_{j-1}} \in \mathbb{Z}_q^*$ are selected at random. Finally, in case of $i_1 = 1$, A sets entry $(0, x_0, \phi, x_0 P')$ with $x_0 \in \mathbb{Z}_q^*$ selected at random. Note that all possible hash queries for $\mathrm{I} = i_1, i_1$ =0 and 1, are defined as Δ_0 and Δ_1 in the hash table.

After the hash table is initiated, A answers hash queries according to the input. To answer a hash query with an input of k $(1 \le k \le l)$ bits, let $w_1 w_2 \cdots w_k$ be the binary representation of input w, A first checks to see if w is within the hash table. If the entry already exists, A outputs the corresponding O. If not, A randomly selects $x_{w_1 w_2 \cdots w_k} \in \mathbb{Z}_q$, stores the entry $(w_1 w_2 \cdots w_k, x_{w_1 w_2 \cdots w_k}, \phi, x_{w_1 w_2 \cdots w_k} P)$ and outputs $x_{w_1 w_2 \cdots w_k} P$. To answer a hash query with an input of l bits or more, input w is parsed as $w_1 w_2 \cdots w_l M_g$, where M_g could be a chosen message by F. Let $\langle v \rangle = w_1 w_2 \cdots w_l$ $(0 \le v < T)$ represent the first l bits of w, M_g summarizes the remaining bits. The index g $(1 \le g \le q_{hash} + q_{sig} + 1)$ denotes the g-th hash query with an input longer than l bits. A checks to see if the entry exists in the hash table. If not, A answers the query depending on the value of v. For $v < i$, A randomly selects $x_{w_1 w_2 \cdots w_l M_g} \in \mathbb{Z}_q$, $y_{\langle v \rangle} \in \mathbb{Z}_q^*$ (if $y_{\langle v \rangle}$ was not define before), stores the entry $(w_1 w_2 \cdots w_l M_g, x_{w_1 w_2 \cdots w_l M_g}, y_{\langle v \rangle}, x_{w_1 w_2 \cdots w_l M_g} P - y_{\langle v \rangle}^{-1} \Delta_{w_1})$ and outputs the Δ value $(\Delta = H(w_1 w_2 \cdots w_l M_g))$; for $v \ge i$, A randomly selects $x_{w_1 w_2 \cdots w_l M_g} \in \mathbb{Z}_q$, stores the entry $(w_1 w_2 \cdots w_l M_g, x_{w_1 w_2 \cdots w_l M_g}, \phi, x_{w_1 w_2 \cdots w_l M_g} P)$ and outputs the Δ value. During execution, A must guess a random index g' $(1 \le g' \le q_{hash} + q_{sig} + 1)$, hoping the forgery is based on the g'-th hash query; set the entry to $(w_1 w_2 \cdots w_l M_{g'}, x_{w_1 w_2 \cdots w_l M_{g'}}, \phi, x_{w_1 w_2 \cdots w_l M_{g'}} P)$ if the input is more than l bits, and return the Δ value. In the case that the input of this hash query is less than l bits, A fails and aborts.

The signing query is answered at random with two cases. For time period $i' < i$, A chooses "node secret" $s_{i'_1 i'_2 \cdots i'_{j-1}} \in \mathbb{Z}_q$ $(2 \le j \le l)$ at random and sets the "verification point" for leaf $Q_{\langle i' \rangle} = y_{\langle i' \rangle} Q$ with $y_{\langle i' \rangle} \in \mathbb{Z}_q^*$ chosen at random. Here, "leaf secret" $s_{\langle i' \rangle} = y_{\langle i' \rangle} \cdot s_\varepsilon$

although s_ε is unknown to *A*. Let $\langle i' \rangle = i'_1 i'_2 \cdots i'_l$. Then A returns $sign = \{V, Q_{\langle i' \rangle}\}$, where $V = \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} x_{i'_1 i'_2 \cdots i'_l M_g} y_{\langle i' \rangle} Q$ and $Q_{\langle i' \rangle}$ consisting of $s_{i'_1 i'_2 \cdots i'_{j-1}} P$ $(2 \le j \le l)$ and $Q_{\langle i' \rangle}$. To answer a signing query in time period i, let $\langle i \rangle = i_1 i_2 \cdots i_l$. Then A chooses $y_{i_1 i_2 \cdots i_{j-1}} \in \mathbb{Z}_q^*$ (for $2 \le j \le l-1$, and $i_j \ne 0$) and "leaf secret" $s_{\langle i \rangle} \in \mathbb{Z}_q$ at random, sets $Q_{i_1 i_2 \cdots i_{l-1}} = y_{i_1 i_2 \cdots i_{l-1}} Q$ ($y_{i_1 i_2 \cdots i_{l-1}}$ was selected during the initiation of hash table), and returns $sign = \{V,\ Q_{\langle i' \rangle}\}$, where $V = \sum_{j=2}^{l-1} x_{i_1 i_2 \cdots i_j} y_{i_1 i_2 \cdots i_{j-1}} Q + x_{i_1 i_2 \cdots i_l} y_{i_1 i_2 \cdots i_{l-1}} Q + s_{\langle i \rangle} H(i_1 i_2 \cdots i_l M_g)$ and $Q_{\langle i' \rangle}$ consists of $y_{i_1 i_2 \cdots i_{j-1}} Q$ $(2 \le j \le l-1)$, $Q_{i_1 i_2 \cdots i_{l-1}}$, and $s_{\langle i \rangle} P$. In summary, the two cases described above are now listed to illustrate that the above equations are correct.

1. For time period $i' < i$, let $y_{\langle i' \rangle} s_\varepsilon = s_{\langle i' \rangle}$,

$$
\begin{aligned}
V &= \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} x_{i'_1 i'_2 \cdots i'_l M_g} y_{\langle i' \rangle} Q \\
&= \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} x_{i'_1 i'_2 \cdots i'_l M_g} y_{\langle i' \rangle} s_\varepsilon P - s_\varepsilon H(i'_1) + s_\varepsilon H(i'_1) \\
&= s_\varepsilon H(i'_1) + \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} H(i'_1 i'_2 \cdots i'_j) + y_{\langle i' \rangle} s_\varepsilon (x_{i'_1 i'_2 \cdots i'_l M_g} P - y_{\langle i' \rangle}^{-1} H(i'_1)) \\
&= s_\varepsilon H(i'_1) + \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} H(i'_1 i'_2 \cdots i'_j) + y_{\langle i' \rangle} s_\varepsilon H(i'_1 i'_2 \cdots i'_l M_g) \\
&= s_\varepsilon H(i'_1) + \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} H(i'_1 i'_2 \cdots i'_j) + s_{\langle i' \rangle} H(i'_1 i'_2 \cdots i'_l M_g)
\end{aligned}
$$

   and we have $Q_{\langle i' \rangle} = \{s_{i'_1} P, s_{i'_1 i'_2} P, \cdots, s_{i'_1 i'_2 \cdots i'_{l-1}} P, Q_{\langle i' \rangle}\}$, where $Q_{\langle i' \rangle} = s_{\langle i' \rangle} P = y_{\langle i' \rangle} s_\varepsilon P = y_{\langle i' \rangle} Q$.

2. For time period i, let $y_{i_1 i_2 \cdots i_{j-1}} s_\varepsilon = s_{i_1 i_2 \cdots i_{j-1}}$,

$$
\begin{aligned}
V &= \sum_{j=2}^{l-1} x_{i_1 i_2 \cdots i_j} y_{i_1 i_2 \cdots i_{j-1}} Q + x_{i_1 i_2 \cdots i_l} y_{i_1 i_2 \cdots i_{l-1}} Q + s_{\langle i \rangle} H(i_1 i_2 \cdots i_l M_g) \\
&= \sum_{j=2}^{l-1} y_{i_1 i_2 \cdots i_{j-1}} s_\varepsilon x_{i_1 i_2 \cdots i_j} P + x_{i_1 i_2 \cdots i_l} y_{i_1 i_2 \cdots i_{l-1}} Q + s_{\langle i \rangle} H(i_1 i_2 \cdots i_l M_g) + s_\varepsilon P' - s_\varepsilon P' \\
&= s_\varepsilon P' + \sum_{j=2}^{l-1} s_{i_1 i_2 \cdots i_{j-1}} H(i_1 i_2 \cdots i_j) + x_{i_1 i_2 \cdots i_l} y_{i_1 i_2 \cdots i_{l-1}} s_\varepsilon P - s_\varepsilon P' + s_{\langle i \rangle} H(i_1 i_2 \cdots i_l M_g) \\
&= s_\varepsilon H(i_1) + \sum_{j=2}^{l-1} s_{i_1 i_2 \cdots i_{j-1}} H(i_1 i_2 \cdots i_j) + y_{i_1 i_2 \cdots i_{l-1}} s_\varepsilon (x_{i_1 i_2 \cdots i_l} P - y_{i_1 i_2 \cdots i_{l-1}}^{-1} P') + s_{\langle i \rangle} H(i_1 i_2 \cdots i_l M_g) \\
&= s_\varepsilon H(i_1) + \sum_{j=2}^{l-1} s_{i_1 i_2 \cdots i_{j-1}} H(i_1 i_2 \cdots i_j) + y_{i_1 i_2 \cdots i_{l-1}} s_\varepsilon H(i_1 i_2 \cdots i_l) + s_{\langle i \rangle} H(i_1 i_2 \cdots i_l M_g) \\
&= s_\varepsilon H(i_1) + \sum_{j=2}^{l-1} s_{i_1 i_2 \cdots i_{j-1}} H(i_1 i_2 \cdots i_j) + s_{i_1 i_2 \cdots i_{l-1}} H(i_1 i_2 \cdots i_l) + s_{\langle i \rangle} H(i_1 i_2 \cdots i_l M_g)
\end{aligned}
$$

   and we have $Q_{\langle i' \rangle} = \{y_{i_1} Q, y_{i_1 i_2} Q, \cdots, y_{i_1 i_2 \cdots i_{l-1}} Q, s_{\langle i \rangle} P\}$, where $Q_{i_1 i_2 \cdots i_{j-1}} = s_{i_1 i_2 \cdots i_{j-1}} P = y_{i_1 i_2 \cdots i_{j-1}} s_\varepsilon P = y_{i_1 i_2 \cdots i_{j-1}} Q$ $(2 \le j \le l)$.

If the breakin query occurs in time period i, A simply outputs the current secret key

$SK_{\langle i\rangle} = \{s_{\langle i\rangle}, S_{\langle i\rangle}, \mathcal{S}_{\langle i\rangle}, \mathcal{Q}_{\langle i\rangle}\}$. Already known are $s_{\langle i\rangle}$, $S_{\langle i\rangle} = \sum_{j=2}^{l-1} x_{i_1 i_2 \cdots i_j} y_{i_1 i_2 \cdots i_{j-1}} Q + x_{i_1 i_2 \cdots i_l} y_{i_1 i_2 \cdots i_{l-1}} Q$ and $\mathcal{Q}_{\langle i\rangle} = \{y_{i_1} Q, y_{i_1 i_2} Q, \cdots, y_{i_1 i_2 \cdots i_{l-1}} Q, s_{\langle i\rangle} P\}$. For elements in $\mathcal{S}_{\langle i\rangle}$, A calculates $S_{i_1 i_2 \cdots i_{j-1} 1} = \sum_{m=2}^{j-1} x_{i_1 i_2 \cdots i_m} y_{i_1 i_2 \cdots i_{m-1}} Q + x_{i_1 i_2 \cdots i_{j-1} 1}\, y_{i_1 i_2 \cdots i_{j-1}} Q$ for all indexes j ($2 \le j \le l$) such that $i_j = 0$. The validity of this equation is illustrated as follows:

$$\begin{aligned} S_{i_1 i_2 \cdots i_{j-1} 1} &= \sum_{m=2}^{j-1} x_{i_1 i_2 \cdots i_m} y_{i_1 i_2 \cdots i_{m-1}} Q + x_{i_1 i_2 \cdots i_{j-1} 1} y_{i_1 i_2 \cdots i_{j-1}} Q + s_\varepsilon P' - s_\varepsilon P' \\ &= s_\varepsilon H(i_1) + \sum_{m=2}^{j-1} s_{i_1 i_2 \cdots i_{m-1}} H(i_1 i_2 \cdots i_m) + y_{i_1 i_2 \cdots i_{j-1}} s_\varepsilon (x_{i_1 i_2 \cdots i_{j-1} 1} P - y_{i_1 i_2 \cdots i_{j-1}}^{-1} P') \\ &= s_\varepsilon H(i_1) + \sum_{m=2}^{j-1} s_{i_1 i_2 \cdots i_{m-1}} H(i_1 i_2 \cdots i_m) + s_{i_1 i_2 \cdots i_{j-1}} H(i_1 i_2 \cdots i_{j-1} 1) \end{aligned}$$

In the case where $i_1 = 0$, A sets $S_1 = s_\varepsilon H(i_1) = s_\varepsilon x_1 P = x_1 Q$. Forger F can then derive the secret key $SK_{i''}$ for $i'' > i$. A aborts if the breakin query is not in time period i.

Suppose that the guess for the break-in time and the index of hash for the signature forgery are correct, and forger F finally outputs a signature $\langle i', sign\rangle$ on message $M_{g'}$ for time period $i' < i$ and $i'_1 = i_1$. If the verification condition holds, A can derive $S' = abP$ utilizing the properties of the bilinear map $\hat{e}$:

$$\frac{\hat{e}(P, V)}{\prod_{j=2}^{l} \hat{e}(Q_{i'_1 i'_2 \ldots i'_{j-1}}, H(i'_1 i'_2 \ldots i'_j))} = \hat{e}(Q, H(i'_1)) \cdot \hat{e}(Q_{i'_1 i'_2 \ldots i'_j}, H(i'_1 i'_2 \cdots i'_l M_{g'})) \quad \overset{(1)}{\Rightarrow}$$

$$\frac{\hat{e}(P, V)}{\hat{e}(Q_{i'_1 i'_2 \ldots i'_l}, H(i'_1 i'_2 \cdots i'_l M_{g'})) \cdot \prod_{j=2}^{l} \hat{e}(Q_{i'_1 i'_2 \cdots i'_{j-1}}, H(i'_1 i'_2 \ldots i'_j))} = \hat{e}(Q, H(i'_1)) \quad \overset{(2)}{\Rightarrow}$$

$$\hat{e}(P, V - \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} H(i'_1 i'_2 \ldots i'_j) - s_{i'_1 i'_2 \cdots i'_l} H(i'_1 i'_2 \cdots i'_l M_{g'})) = \hat{e}(P,\ s_\varepsilon H(i'_1)) \quad \overset{(3)}{\Rightarrow}$$

$$V - \sum_{j=2}^{l} s_{i'_1 i'_2 \cdots i'_{j-1}} H(i'_1 i'_2 \ldots i'_j) - s_{i'_1 i'_2 \cdots i'_l} H(i'_1 i'_2 \cdots i'_l M_{g'}) = s_\varepsilon H(i'_1) \quad \overset{(4)}{\Rightarrow}$$

$$V - \sum_{j=2}^{l} x_{i'_1 i'_2 \ldots i'_j} Q_{i'_1 i'_2 \cdots i'_{j-1}} - x_{i'_1 i'_2 \ldots i'_l M_g} Q_{i'_1 i'_2 \ldots i'_l} = abP \quad \overset{(5)}{\Rightarrow}$$

$$S' = abP = V - \sum_{j=2}^{l} x_{i'_1 i'_2 \ldots i'_j} Q_{i'_1 i'_2 \cdots i'_{j-1}} - x_{i'_1 i'_2 \ldots i'_l M_g} Q_{i'_1 i'_2 \ldots i'_l}$$

Similarly, A can derive $S' = abP = x_0^{-1} \cdot (V - \sum_{j=2}^{l} x_{i'_1 i'_2 \ldots i'_j} Q_{i'_1 i'_2 \cdots i'_{j-1}} - x_{i'_1 i'_2 \ldots i'_l M_g} Q_{i'_1 i'_2 \ldots i'_l})$ when $i'_1 \neq i_1$. For $i' < i$ and $i'_1 \neq i_1$, the only possible case is the one in which $i'_1 = 0$, $i_1 = 1$, and hence $s_\varepsilon H(i'_1) = x_0 abP$. The above derivations utilized the non-degenerate property of the bilinear map in step (3). Note that $\mathbb{G}_1$ and $\mathbb{G}_2$ are in prime order q, which implies that if P is a generator of $\mathbb{G}_1$, then $\hat{e}(P, P)$ is a generator of $\mathbb{G}_2$. It also means that there does not exist $P_1, P_2 \in \mathbb{G}_1$ ($P_1 \neq P_2$) that yields $\hat{e}(P, P_1) = \hat{e}(P, P_2)$. Therefore, P_1 must be the same as P_2.

RUNNING TIME ANALYSIS. A runs F from time period 0 to $T-1$. To answer the hash query, ignoring the table look up time, requires some multiplications and additions on $\mathbb{G}_1$, i.e.

$O(k^{n_1})$ bit operations. It also requires some inverse operations on $\mathbb{Z}_q^*$ – $O(k^2 \log T)$ bit operations for time period i. To answer a signing query, A may require some additional multiplications on $\mathbb{Z}_q^*$ ($O(k^2 \log T)$) and one less multiplication on $\mathbb{G}_1$ ($O(k^{n_1})$) than the real user. To compute the final $S' = abP$, it takes some multiplication and addition operations on $\mathbb{G}_1$ ($O(k^{n_1})$ bit operations). Therefore, A exceeds the running time of F by $O(k^{n_1} + k^2 \log T)$.

PROBABILITY ANALYSIS. We first compare A's answer to the signing oracle with that of the real signer. From F's point of view, A always acts like a real signer, except in one case, i.e. when A answers the signing query for some time period $i' < i$, and the hash entry that A wants to use has already been defined to $(w_1 w_2 \cdots w_l M_g, x_{w_1 w_2 \cdots w_l M_g}, \phi, x_{w_1 w_2 \cdots w_l M_g} P)$ by the previous hash query of F. This is the only case in which A fails to answer the signing query and the reason cause this is A's wrong guessing of the hash index g'. In other words, A always acts like a real signer, as long as the guessing of g' is correct.

There are totally two guesses performed by A. The probability to guess the correct time period F sends the breakin query is exactly $1/T$ and the probability to guess the correct hash query on which the forgery is based is $\Pr \geq 1/(q_{hash} + q_{sig} + 1)$. Therefore, the probability of A's success in deriving $S' = abP$ is at least

$$\varepsilon' = 1/T \cdot 1/(q_{hash} + q_{sig} + 1) \cdot \varepsilon$$

where ε is the minimum probability for F to successfully forge a signature. Theorem 3.1 follows.