8-20-2013

A Forward-Secure Certificate-based Signature Scheme

Jiguo Li

Huiyun Teng

Xinyu Huang
University of South Carolina - Columbia, xyhuang@sc.edu

Yichen Zhang

Jianying Zhou

Publication Info

Published in *The Computer Journal*, Volume 58, Issue 4, 2013, pages 853-866.

The Computer Journal Advance Access published December 3, 2013

doi:10.1093/comjnl/bxt141

A Forward-Secure Certificate-Based Signature Scheme

JIGUO LI[1]*, HUIYUN TENG[1], XINYI HUANG[2], YICHEN ZHANG[1] AND JIANYING ZHOU[3]

[1]*College of Computer and Information Engineering, Hohai University, Nanjing, Jiangsu, China 210098*
[2]*School of Mathematics and Computer Science, Fujian Normal University, Fuzhou, China 350000*
[3]*Institute for Infocomm Research (I2R), A*STAR, South Tower, Singapore 138632*
**Corresponding author: ljg1688@163.com; lijiguo@hhu.edu.cn*

Cryptographic computations are often carried out on insecure devices for which the threat of key exposure raises a serious concern. In an effort to address the key exposure problem, the notion of forward security was first presented by Günther in 1990. In a forward-secure scheme, secret keys are updated at regular periods of time; exposure of the secret key corresponding to a given time period does not enable an adversary to 'break' the scheme for any prior time period. In this paper, we first introduce forward security into certificate-based cryptography and define the security model of forward-secure certificate-based signatures (CBSs). Then we propose a forward-secure CBS scheme, which is shown to be secure against adaptive chosen message attacks under the computational Diffie–Hellman assumption in the random oracle model. Our result can be viewed as the first step toward solving the key exposure problem in CBSs and thus improving the security of the whole system.

Keywords: forward security; certificate-based signature; computational Diffie–Hellman assumption; random oracle model

Received 8 January 2013; revised 20 August 2013
Handling editor: Chris Mitchell

1. INTRODUCTION

Nowadays, secret key exposure is arguably the greatest threat against the security of a digital signature scheme, due to security breaches of the underlying system or machine storing the key. To deal with this issue, a variety of methods have been proposed, including secret sharing [1], threshold cryptography [2], proactive cryptography [3], leakage-resilient cryptography [4–9] and forward security [10].

1.1. Related work

The notion of forward security was first proposed in the context of key-exchange protocols by Günther [10]. A forward-secure key-exchange protocol guarantees that exposure of long-term secret information does not compromise the security of previously generated session keys.

Anderson [11] introduced the notion of forward-secure signatures to solve some defects in general digital signatures; namely once the secret key is lost or stolen, the signatures generated by this secret key will become invalid. Therefore, forward security can reduce the influence of key exposure: a lost or stolen key at a time period T will not affect the validity of signatures produced before T. Bellare and Miner [12] gave the first formal definition of forward-secure signatures and presented a forward-secure digital signature scheme, which is inspired by the Fiat and Shamir [13] and Ong and Schnorr [14] identification and signature schemes. In their scheme, the public key is constantly unchanged and the secret key is generated by some one-way hash functions and previous time period secret key. Therefore, signatures and secret key in each time period are different. Even if the current time period secret key is exposed, it would not affect the validity of previous signatures. This is a countermeasure to alleviate the damage caused by key exposure. In 2000, Krawczyk [15] presented simple forward-secure signatures from any signature schemes. Abdalla and Reyzin [16] proposed a new forward-secure digital signature scheme with a shorter public key size. Their scheme can be viewed as an improvement of the Bellare–Miner scheme [12]. Tzeng and Tzeng [17] proposed a robust forward-secure signature scheme which enhanced the security of Abdalla and Reyzin's forward-secure signature scheme by using threshold

and proactive mechanisms. Itkis and Reyzin [18] proposed another forward-secure signature scheme, but the efficiency of key generation and update algorithm is not satisfactory. The performance of the above algorithms depends on the security parameter as well as a priori maximum number of time periods T. Therefore, setting T to an unnecessarily large number will result in a considerable efficiency loss. In order to solve this problem, Malkin *et al.* [19] constructed the first efficient forward-secure digital signature scheme where the total number of time periods for which the public key was used does not have to be fixed in advance. Their scheme is a generic construction, namely it can be realized on any underlying signature schemes, and does not rely on specific computational assumptions like discrete log or factoring. Furthermore, its forward security was proved in the standard model. Subsequently, Kang *et al.* [20] proposed two forward-secure signature schemes based on gap Diffie–Hellman groups and proved their schemes to be secure in a slightly stronger security notion than that used by Bellare and Miner [12] in the random oracle model. To reduce the risk of key exposure, forward-secure group signature was first proposed by Song [21]. To simplify the integration of these primitives into standard security architectures, Boyen *et al.* [22] introduced the concept of forward-secure signatures with untrusted updates where private keys are additionally protected by a second factor (derived from a password). Key updates can be made on encrypted version of signing keys so that passwords only come into play for signing messages. The latter works also suggested the integration of untrusted updates in the Bellare–Miner forward-secure signature [12] and left open the problem of endowing other existing forward-secure signature systems with the same second factor protection. Libert *et al.* [23] solved this problem by showing how to adapt the very efficient generic construction of [19] in untrusted update environments. Alomair *et al.* [24] proposed a generic construction method to obtain a forward-secure signature scheme that is very efficient in parameter size and computation times. Furthermore, they showed that their scheme can be easily extended to proxy signature schemes.

In Eurocrypt 2003, Gentry [25] introduced the notion of certificate-based encryption (CBE). As in the traditional public key infrastructure (PKI), each client in CBE generates its own public/private key pair and the certificate authority (CA) then generates a certificate that can guarantee the authenticity of the client's public key. In CBE, the certificate has an additional feature, namely it also acts as a partial private key. A successful decryption requires both the private key and the up-to-date certificate. This provides an implicit verification of one's certificate and eliminates third-party queries for certificate status. Since the CA does not know the client's private key, there is no key escrow problem in CBE. Certificate-based cryptography is envisioned as a promising mechanism in constructing efficient PKIs and has attracted a lot of attention since it was proposed. Analogous to CBE, Kang *et al.* [26] proposed the notion of certificate-based signatures (CBSs) inspired by the idea of CBE presented by Gentry [25]. Li *et al.* [27] first introduced key replacement attack into certificate-based system and refined the security model of the CBS. They showed that one of CBS schemes presented by Kang *et al.* [26] was insecure under key replacement attacks. Furthermore, they proposed a new secure and efficient CBS scheme, which was shown to be existentially unforgeable against adaptive chosen message attacks under the computational Diffie–Hellman (CDH) assumption in the random oracle model. A generic construction of CBSs was proposed by Wu *et al.* [28, 29]. Li *et al.* [30] presented two new CBS schemes that are secure against key replacement attacks. Compared with other designs, their first scheme enjoys shorter signature length and less operation cost. Their second scheme is the first construction of a CBS secure against key replacement attacks in the standard model. Recently, Li *et al.* [31] proposed an efficient short CBS scheme, which requires only one pairing operation in signature generation and verification. In addition, the signature size of their scheme is only one group element. Furthermore, they [32] proposed a new certificate-based signcryption scheme with enhanced security features.

1.2. Motivations and contributions

CBSs have potential applications in trusted computing. Trusted computing is undoubtedly a powerful technology, with a huge range of possible applications. However, Balfe *et al.* [33] pointed out some challenges for trusted computing, which affects its widespread deployment. The most significant challenge is the deployment and management of the PKI, which is necessary to enable the general use of security services (for example, some certificates from an endorsement CA, a platform CA and one or more conformance CAs.) supported by trusted computing. Another issue is that credential revocation within a TC-PKI may introduce further inconvenience. Given the complex dependencies between many of the TC-PKI credentials, the compromise of an individual key and the subsequent revocation of its associated public key certificate will result in a cascading revocation of all dependent TPM credentials. Solving the above problems seems to require a lot of infrastructure. Similarly, the existing property-based attestation solutions proposed by Chen *et al.* [34–36] require a trusted third party to provide a reliable link of configurations to properties, e.g. by means of certificates. A traditional PKI system requires a large amount of computing time and storage when the number of users increases rapidly. At the same time, it is difficult for certificate revocation to distribute large amounts of fresh certification information. The apparent need for this infrastructure is regarded as a major reason, which affects widespread implementation of public-key cryptography. Therefore, traditional PKI is very difficult to directly apply in a trusted computing setting. In Eurocrypt 2003, Gentry [25] introduced the notion of CBE. The main motivation of CBE/signature is to construct an efficient PKI requiring

less infrastructures, solve certificate revocation problem and eliminate third-party queries in the traditional PKI. This new cryptographic paradigm can supply practical methods for general use of the above security services supported by trusted computing. Forward-secure signature can guarantee that even if the current time period secret key is exposed, it would not affect the validity of previous signatures, which reduces the impact of key exposure. Both CBSs and forward-secure signatures can easily apply to security services required by trusted computing. However, to date, there is no concrete design of forward-secure CBSs. This paper is aimed at constructing an efficient forward-secure CBS scheme, which enables general use of security services supported by trusted computing.

In this paper, we introduce a new signature paradigm called forward-secure CBS and formally define the security model. It preserves the advantages of CBS such as implicit certificate and no private key escrow. At the same time it also inherits the properties of forward-secure signatures. We construct a forward-secure CBS scheme. The key update algorithm in our construction makes use of the pre-order traversal technique of [37]. We associate time periods with all nodes of the tree, which improves the efficiency of our key-generation and key-update algorithm.

Organization. In Section 2, we review the notions of bilinear mapping and the CDH problem, and introduce the formal definition and security model of a key-evolving signature scheme. In Section 3, we formally define the security model of forward-secure CBSs. We define two different types of adversaries: Type I adversary $\mathcal{A}_I$ and Type II adversary $\mathcal{A}_{II}$, and describe their ability, respectively. In Section 4, we review the pre-order traversal technique of binary trees and construct a forward-secure CBS scheme. In Section 5, we prove the security of our scheme in the random oracle model based on the computation Diffie–Hellman assumption. Section 6 concludes this paper.

2. PRELIMINARIES

In this section, we introduce several relevant background knowledge including the bilinear mapping and CDH problem, and review the formal definition and security model of forward-secure signatures.

2.1. Bilinear mapping

Let G_1 and G_2 be two cyclic groups of prime order q, where G_1 is an additive group and G_2 is a multiplicative group. Let P be a generator of G_1 and $e : G_1 \times G_1 \rightarrow G_2$ be a bilinear mapping with the following properties:

(1) Bilinear: For all $P, Q \in G_1$ and all $a, b \in \mathbb{Z}_q^*$, $e(aP, bQ) = e(P, Q)^{ab}$.
(2) Non-degenerate: $e(P, P) \neq 1 \in G_2$.
(3) Computable: e is efficiently computable.

2.2. The CDH problem

We assume that G_1 is an additive cyclic group with prime order q, and P is a generator of G_1. Given (P, aP, bP), where $a, b \in \mathbb{Z}_q^*$, compute abP. The advantage of an algorithm $\mathcal{A}$ in solving the CDH problem in G_1 is defined to be $\text{Succ}^{CDH}_{\mathcal{A},G_1} = \Pr[\mathcal{A}(P, aP, bP = abP | a, b \in \mathbb{Z}_q^*]$.

The CDH assumption states that, for every probabilistic polynomial-time algorithm $\mathcal{A}$, $\text{Succ}^{CDH}_{\mathcal{A},G_1}$ is negligible.

2.3. A key-evolving signature scheme

A forward-secure digital signature scheme is, first of all, a key-evolving digital signature scheme. A key-evolving signature scheme is very similar to a standard one, except that its operation is divided into time periods, each of which uses a different secret key to sign a message. The keys are updated by an algorithm that computes the secret key for the new time period based on the current secret key. Meanwhile, the public key is unchanged throughout the lifetime of the scheme. The following definition of forward security is proposed by Abdalla and Reyzin [16].

DEFINITION 1. *A key-evolving digital signature scheme is a 4-tuple algorithm, FSIG = (FSIG.key, FSIG.update, FSIG.sign, FSIG.vf), where*

(i) FSIG.key, the key generation algorithm, takes as input a security parameter $k \in \mathrm{N}$ (given in unary as 1^k) and the total number of periods N, and returns a pair (SK_0, PK), the initial secret key and the public key;
(ii) FSIG.update, the secret key update algorithm, takes as input the secret key for the current period SK_j and returns the new secret key SK_{j+1} for the next period;
(iii) FSIG.sign, the signing algorithm, takes as input the secret key SK_j for the current time period j and a message M to be signed, and returns a pair $\langle j, sign \rangle$, the signature of M for time period j;
(iv) FSIG.vf, the verification algorithm, takes as input the public key PK, a message M and a candidate signature $\langle j, sign \rangle$, and returns 1 if $\langle j, sign \rangle$ is a valid signature of M or 0, otherwise.

It is required that $\text{FSIG.vf}_{PK}\ (FISG.sign_{SK_j}(M), M) = 1$ for every message M and time period j. We assume that SK_{N+1} is an empty string and that $FSIG.update_{SK_N}$ returns SK_{N+1}.

When we work in the random oracle model, all the above-mentioned algorithms would additionally have oracle access to a public hash function H, which is assumed to be a random oracle in the security analysis. Consider that an adversary is able to obtain the secret key of some time period; following the idea of Bellare and Miner [12], Abdalla and Reyzin [16] refined the security model of forward-secure signatures. Recall that the goal is that even under exposure of the current secret key, it should be computationally infeasible for an adversary to forge a signature with respect to a previous secret key. Formally, in order to attack

forward-secure signature schemes, the adversary is modeled via the following experiment in the random oracle model. In this experiment, the adversary is denoted by F, and adversary's operation is divided into three phases: chosen message attack (**cma**) phase, break-in (**breakin**) phase and forgery (**forge**) phase.

EXPERIMENT 1. F-Forge-RO(FSIG,F)
Select $H : \{0,1\}^* \rightarrow \{0,1\}^l$ at random
$(PK, SK_0) \leftarrow \text{FSIG.key}^H(k, \ldots, N)$
$j \leftarrow 0$
repeat
$j \leftarrow j + 1$
$SK_j \leftarrow \text{FSIG.update}^H(SK_{j-1}, j)$
$d \leftarrow F^{H, \text{FSIG.sign}^H_{SK_j}(\cdot)}(\textbf{cma}, PK)$
Until $(d = \textbf{breakin})$ or $(j = N)$
If $d \neq \textbf{breakin}$ and $j = T$ then $j \leftarrow T + 1$
$(M, \langle b, sign\rangle) \leftarrow F^H(\textbf{forge}, SK_j)$
If $\text{FSIG.vf}^H_{\text{PK}}(M, \langle b, sign\rangle) = 1$ and $1 \leq b < j$
and M was not queried of $\text{FSIG.sign}^H_{SK_b}(\cdot)$ in period b
then return 1 else return 0

In this model, an adversary knows the public key PK, the total number of time periods N and the current time period j. In the *cma* phase, according to time sequence, the adversary can query signatures of messages which are chosen by himself, and the challenger responds to the adversary's queries by using $SK_0, SK_1, SK_2, \cdots$. At the end of each time period, the adversary can choose whether to stay in the same phase or switch to the **breakin** phase. It cannot query signatures under previous secret keys. The secret key will be exposed in time period j, so the adversary can obtain a user's secret key in time period j. In the **breakin** phase, once the adversary has decided to break-in in time period j, challenger will give the secret key of time period j to the adversary. In the **forge** phase, the adversary outputs a signature forgery $\langle M, \sigma\rangle$ in period b, $0 < b < j$. The adversary is said to successfully attack the scheme *FSIG* if $\text{FSIG.vf}(M, \langle b, \sigma\rangle) = 1$ holds and M was not queried to $\text{FSIG.sign}^H_{SK_b}(\cdot)$ in period b.

3. FORMAL DEFINITION AND SECURITY MODEL

Prior to describing our formal definition and security model in detail, we first give a high-level introduction of how a signature scheme works in certificate-based public key cryptography. The following is a high-level description:

(i) A CBS scheme involves three parties: the certifier (generates certificates), the signer (produces signatures) and the verifier (verifies signatures).
(ii) At the very beginning, the certifier generates the system parameter and a master private/public key pair (*msk*, *mpk*). The system parameters and the master public key *mpk* are assumed to be publicly known to all users in the system.
(iii) An entity (say, Alice) generates a private/public key pair (SK_{ID}, PK_{ID}) by taking system parameters as the input.
(iv) After that, Alice sends a certificate request to the certifier and asks the latter to issue a certificate for PK_{ID}.
(v) The certifier verifies Alice's request, and if everything is correct, he/she will generate a certificate that binds together Alice, PK_{ID} and other information. Depending on concrete situations, Alice may need to provide the proof of her knowledge of the relevant private key SK_{ID}. At the end of this phase, the certificate is sent to its owner Alice.
(vi) Alice then can produce signatures using the certificate and her private key SK_{ID}.
(vii) The signature recipient verifies the signature using the system parameters and the public keys of Alice and the certifier. In particular, there is no need to verify Alice's certificate separately.
(viii) Alice needs to contact the certifier, in a regular time period, for certificate updates.

This completes the high-level description of CBSs. The reader is referred to [25–32] for details.

Inspired by the security models in [25–30], we define the security model of forward-secure CBSs. In our security model, the type I adversary $\mathcal{A}_I$ can obtain a valid signature under the public key which may be replaced by himself, with the restriction that he can supply the corresponding secret key. The type II adversary $\mathcal{A}_{II}$, who has the master secret key, wishes to generate a valid signature under the public key without the knowledge of the corresponding secret key. The type II adversary $\mathcal{A}_{II}$ mainly simulates a malicious certifier who is able to produce certificates but is not allowed to replace the target user's public key. The security notion is defined by the game between the challenger $\mathcal{C}$ and the adversary. In this section, we give formal definitions of the forward-secure CBS, describe several oracles that would be available to the adversaries in forward-secure CBSs and define the attack power of two adversaries, respectively.

3.1. Formal definition

A forward-secure CBS is a 6-tuple algorithm, FSIG.CBS = (Setup, UserKeyGen, CertGen, KeyUpdate, Sign, Verify), where Setup and CertGen algorithms are run by the certifier. These algorithms work as follows:

(i) Setup, this algorithm takes as input a security parameter $k \in \mathbb{N}$ (given in unary as 1^k) and the total number of periods N, and returns the certifier's master secret key *msk*, master public key *mpk* and public parameter

params, where *mpk* and *params* are published in the system;

(ii) UserKeyGen, the user key generation algorithm, takes as input user *ID*, *mpk*, *params*, and returns the user's initial secret key and the public key pair (SK_0, PK_{ID}); the system public key is $PK = (mpk, PK_{ID}, params)$;

(iii) CertGen, the certificate generation algorithm, takes as input the master secret key *msk*, system parameter *params*, the identity *ID* of a user and its public key PK_{ID}. It outputs a certificate $Cert_{ID}$;

(iv) KeyUpdate, the secret key update algorithm, takes as input the secret key of the current period SK_j and returns the new secret key SK_{j+1} for the next period;

(v) Sign, the signing algorithm, takes as input the secret key SK_j of the current time period j, certificate $Cert_{ID}$, user *ID* and a message M to be signed, and returns a pair $\langle j, sign\rangle$, the signature of M for time period j;

(vi) Verify, the verification algorithm, takes as input the public key *PK*, a message M and a candidate signature $\langle j, sign\rangle$, and returns 1 if $\langle j, sign\rangle$ is a valid signature of M or 0, otherwise.

It is required that $\text{Verify}_{PK}(\text{Sign}_{SK_j, Cert_{ID}}(M), M) = 1$ for every message M and time period j. We assume that SK_{N+1} is an empty string and that $KeyUpdate_{SK_N}$ returns SK_{N+1}.

Remark 1. To eliminate the need for online certificate status checks in our scheme, which significantly reduces the workload of a CA, we enhance the security notion of forward-secure CBS that follows the idea of 'certificate updating' in [25, 26]. We can easily define it by replacing algorithm CertGen with algorithms Upd1 and Upd2 in [25, 26]. Concrete algorithms are as follows:

(i) Upd1, the CertifierUpdate algorithm, takes as input the master secret key *msk*, system parameter *params*, i, string $s \in S$, where S is a string space, the identity *ID* of a user and its public key *PK* at the start of time period i. It outputs $Cert'_i$, which is sent to the user.

(ii) Upd2, the UserUpdate algorithm, takes as input system parameter *params*, i, $Cert'_i$, the identity *ID* of a user and (optionally) $Cert_{i-1}$ at the start of time period i. It returns $Cert_i$.

In the corresponding algorithm, we replace certificate $Cert_{ID}$with the updated certificate $Cert_i$, which does not affect the security model, construction of our scheme and security proof.

Remark 2. As stated in [25], it does not necessarily have to be 'certificate updating', and it can be useful for applications other than certificate management. In particular, it may be useful in other situations where authorization or access control is an issue. Therefore, we simplify our algorithms in our definition in order to avoid complex symbols.

3.2. Adversary oracles

We first define the following oracles that can be accessed by the adversary in the forward-secure CBSs.

UserKeyGen: This oracle maintains two lists L_1 and L_2, which are initially empty and used to record the information for each user *ID*. On a **UserKeyGen** query *ID*, if *ID* has already been created, nothing is to be carried out by the challenger $\mathcal{C}$. Otherwise, $\mathcal{C}$ runs the algorithm **UserKeyGen** and obtains the initial secret key and public key pair (SK_{ID}, PK_{ID}). Then it adds (ID, SK_{ID}, PK_{ID}) into the list L_1 and adds $(ID, \overline{PK_{ID}})$ into the list L_2. Here, $\overline{PK_{ID}} = PK_{ID}$. In this case, *ID* is said to be created. In both cases, PK_{ID} is returned. It is noted that L_1 provides the information of *ID*'s secret key and the public key when it is created; L_2 provides the information of *ID*'s current public key, denoted as $\overline{PK_{ID}}$, which might not be the one generated by this oracle.

PKReplace: On a **PKReplace** query (ID, PK'_{ID}), $\mathcal{C}$ finds the user *ID* in the list L_2, sets $\overline{PK_{ID}} = PK'_{ID}$ and updates the corresponding information in the list L_2. Note that PK'_{ID} is chosen by the adversary. For a created user *ID*, the adversary can replace the public key repeatedly.

CertGen: On a **CertGen** query *ID*, $\mathcal{C}$ runs algorithm **CertGen** and returns the user's certificate corresponding to the user's public key generated by **UserKeyGen**. Note that $Cert_{ID}$ is the certificate of the pair (ID, PK_{ID}), where PK_{ID} is the public key returned from the oracle **UserKeyGen**.

Corruption: On a *Corruption* query (ID, j), where *ID* denotes the identity which has been created, $\mathcal{C}$ checks the list L_1 and returns the secret key (SK_{ID}, j) in current time period j. Note that the secret key is the one corresponding to *ID*'s original public key PK_{ID} returned by **UserKeyGen**.

Sign: On a **Sign** query (ID, m), where *ID* denotes the identity which has been created. If the user's public key has been replaced, the adversary must supply the secret key corresponding to PK'_{ID}. Otherwise, $\mathcal{C}$ runs algorithm **Sign** and returns the signature σ in the current time period.

Breakin: On a **Breakin** query (ID, T), $\mathcal{C}$ returns the user $ID's$ secret key SK_T for a given time period T to adversary $\mathcal{A}$; $\mathcal{A}$ does not need to make any **Corruption** queries.

Remark 3. Corruption oracle mainly models a user collusion attack in current time period j. Corruption oracle is not queried in the break-in time period. While the break-in oracle is only queried once before moving into the break-in phase, it models forward security of the scheme.

3.3. Security against the key replacement adversary $\mathcal{A}_I$

In this section, we will consider the type I adversary $\mathcal{A}_I$. We will describe the attack scenarios where an adversary wants to forge a valid signature under the public key PK_{ID^*} whose certificate is not known to him in time period γ for some $0 \le \gamma < T$; T is the key exposure time period. The public key PK_{ID} may be the

genuine one generated by the user *ID* or the fake one chosen by the adversary.

(1) $\mathcal{A}_I$ can obtain some message/signature pairs (M_i, σ_i) in any time periods α for some $0 \leq \alpha \leq T$ generated by user *ID* which is chosen by himself.
(2) $\mathcal{A}_I$ can replace any user *ID*'s public key with PK'_{ID} which is chosen by himself. He can dupe any other third party to verify the user *ID*'s signatures using the false public key PK'_{ID}.
(3) If $\mathcal{A}_I$ has replaced the user *ID*'s public key, he cannot obtain the certificate of the false public key from the certifier.

In this game, the adversary $\mathcal{A}_I$ knows the user's public key PK_{ID}, the total number of time periods N and the current time period j. The adversary $\mathcal{A}_I$ runs in three phases. In the first phase, the chosen message attack phase, the adversary $\mathcal{A}_I$ has access to a signing oracle defined in Section 3.2, which it can query to obtain signatures of messages of its choice with respect to the current secret key. At the end of each time period, the adversary can choose whether to stay in the same phase or switch to the break-in phase. In the break-in phase, which models the possibility of a key exposure, we give the adversary the secret key (SK_{ID}, T) in the current time period T it decided to break in. In the last phase, the forgery phase, the adversary $\mathcal{A}_I$ outputs a pair signature message, that is, a forgery. The adversary $\mathcal{A}_I$ is considered to be successful if it forges a signature of some new message (that is, not previously queried to the signing oracle) for some time period prior to T. The security of a forward-secure CBS scheme against a key replacement and adaptively chosen message attack is defined by the game between $\mathcal{A}_I$ and the challenger $\mathcal{C}$ as follows:

Chosen Message Attack Phase:

Setup: The challenger $\mathcal{C}$ runs the algorithm **Setup** and returns $(mpk, params)$ to $\mathcal{A}_I$.

Query: In polynomial time t, $\mathcal{A}_I$ can adaptively submit various queries except **Breakin** query defined in Section 3.2 and hash queries to the challenger in the current time period, where we regard the hash function as a random oracle. Note that $\mathcal{A}_I$ can also submit the **CertGen** query. On $\mathcal{A}'_I$s **CertGen** query (ID, PK_{ID}), $\mathcal{C}$ runs the algorithm **CertGen** and returns the user *ID*'s certificate $Cert_{ID}$ to $\mathcal{A}_I$.

In each time period, $\mathcal{A}_I$ can choose whether to stay in the same phase or switch to the **breakin** phase. It cannot query any oracles in previous time periods.

Break-in Phase:

Breakin query: The challenger $\mathcal{C}$ models the possibility of a key exposure and gives the user's secret key for the specific time period T to the adversary $\mathcal{A}_I$.

Forgery Phase:

At last, $\mathcal{A}_I$ outputs a forgery $< \gamma, M^*, \sigma^*, ID^*, PK_{ID^*} >$. We say that $\mathcal{A}_I$ wins if all conditions have to be fulfilled.

(1) σ^* is a valid signature on the message M^* under the public key PK_{ID^*} in the time period γ, $0 \leq \gamma < T$. Here, PK_{ID^*} is chosen by $\mathcal{A}_I$ and might not be the one returned from the oracle **UserKeyGen**.
(2) ID^* has never been submitted as one of **CertGen** queries.
(3) $< M^*, ID^* >$ has never been submitted as one of *Sign* queries in time period γ.

We define the success probability of $\mathcal{A}_I$ winning the above game as $Succ_{\mathcal{A}_I}^{cma,cida,breakin}$.

DEFINITION 2. *We say a forward-secure CBS scheme is secure against a (t, q) chosen message and chosen identity adversary $\mathcal{A}_I$ if $\mathcal{A}_I$ runs in polynomial time t, makes at most q queries and $Succ_{\mathcal{A}_I}^{cma,cida,breakin}$ is negligible.*

3.4. Security against the malicious certifier adversary $\mathcal{A}_{II}$

In this section, we will consider the type II adversary $\mathcal{A}_{II}$. Informally, we will describe the attack scenarios where the malicious certifier wants to generate a valid signature under the public key PK_{ID^*} without the knowledge of the corresponding secret key.

(i) $\mathcal{A}_{II}$ has the knowledge of the certifier's secret key *msk*.
(ii) $\mathcal{A}_{II}$ can obtain some message/signature pairs (M_i, σ_i) in any time periods α for some $0 \leq \alpha \leq T$ generated by user *ID* which is chosen by himself.
(iii) $\mathcal{A}_{II}$ cannot replace any user's public key.

The security of a forward-secure CBS scheme against a type II adversary is defined by the game between $\mathcal{A}_{II}$ and the challenger $\mathcal{C}$ as follows:

Chosen Message Attack Phase:

Setup: The challenger $\mathcal{C}$ runs the algorithm **Setup** and returns $(mpk, msk, params)$ to $\mathcal{A}_{II}$.

Query: In polynomial time t, $\mathcal{A}_{II}$ can adaptively submit **UserKeyGen**, **Corruption**, **Sign** and hash queries to the oracles in the current time period. Here $\mathcal{A}_{II}$ has obtained the master secret key, namely, he can calculate any user's certificate by himself. So $\mathcal{A}_{II}$ does not need to submit any *CertGen* queries; $\mathcal{A}_{II}$ cannot submit public key replacement query.

Break-in Phase:

Break-in query: The challenger $\mathcal{C}$ models the possibility of a key exposure and gives the user's secret key for the specific time period T to the adversary $\mathcal{A}_{II}$.

Forgery Phase:

At last, $\mathcal{A}_{II}$ outputs a forgery $< \gamma, M^*, \sigma^*, ID^* >$. We say $\mathcal{A}_{II}$ wins if all conditions have to be fulfilled.

(i) σ^* is a valid signature on the message M^* under the public key PK_{ID^*} and the system's master public key *mpk* in the time period γ, $0 \leq \gamma < T$. Not that PK_{ID^*} is the public key output from the oracle **UserKeyGen**; more explicitly, the adversary does not replace the target

user public key, i.e. he only commits to a non-target ID at some point and then a user public key for this ID is generated by the simulator.

(ii) $< M^*, ID^* >$ has never been submitted as one of **Sign** queries in time period γ.

(iii) Once the adversary switches to the **breakin** phase, ID^* has never been submitted as one of the **Corruption** queries prior to the **breakin** phase again.

We define the success probability of $\mathcal{A}_{II}$ winning the above game as $Succ^{cma,cida,breakin}_{\mathcal{A}_{II}}$.

DEFINITION 3. *We say a forward-secure CBS scheme is secure against a (t, q) chosen message and chosen identity adversary $\mathcal{A}_{II}$ if $\mathcal{A}_{II}$ runs in polynomial time t, and makes at most q queries and $Succ^{cma,cida,breakin}_{\mathcal{A}_{II}}$ is negligible.*

4. A FORWARD-SECURE CBS SCHEME

Inspired by the CBS scheme proposed by Li *et al.* [27, 30], we utilize the pre-order traversal technique of binary trees [37] to update the user's secret key in the design of a forward-secure CBS scheme. Our construction is existentially unforgeable against adaptive chosen message attacks under the CDH assumption in the random oracle model. Before introducing our concrete scheme, we first summarize the pre-order traversal technique.

4.1. Notations

The key-evolving method of our scheme employs the well-known pre-order traversal technique of binary trees [37], which is the advancement of the tree-traversal method in [38]. The pre-order traversal technique associates time periods with all nodes of the tree, while the tree-traversal method in [38] associates time periods with the leaves only. Thus, the depth of binary tree can be decreased from $\log_2(N+1)$ to $\log_2(N+1)-1$ (N is the total of time periods) and the running time of the key update algorithm can be reduced from $O(logN)$ to $O(1)$ [37].

If we use a full binary tree with depth l, then the number of time periods is $N = 2^{l+1}-1$. The root of the tree is called node ε. Denote the node (represented by a bit string) and its secret key corresponding to the time period i by ω^i and S_{ω^i}, respectively. Let $\omega^i 0(\omega^i 1)$ be the left (right) child node and let $\omega^i|k$ be a k bit-prefix of ω^i. Let $\omega|\overline{k}$ be the sibling node of $\omega|k$. Pre-order traversal can be defined as follows: $\omega^0 = \varepsilon$ is the root node, and if ω^i is an internal node, then $\omega^{i+1} = \omega^i 0$. If ω^i is a leaf node and $i < N-1$, $\omega^{i+1} = \omega' 1$. $\omega' 0$ is the longest prefix of ω^i.

The system public key PK remains fixed throughout the lifetime of the system, which includes the master public key mpk, the user public key PK_{ID} and the corresponding parameters. In the time period i, the signer generates a signature with respect to the node secret key S_{ω^i}, but the secret key SK_i contains secret keys of the right siblings of the nodes on the path from the root to ω^i and the secret node key S_{ω^i}. That is, whenever $\omega' 0$ is a prefix of ω^i, SK_i contains the secret key of node $\omega' 1$. So the secret key of time period i is expressed by $SK_i = (S_{\omega^i|\overline{1}}, S_{\omega^i|\overline{2}}, \ldots, S_{\omega^i|\overline{n}}, S_{\omega^i})$, where $\omega^i = \omega_1 \cdots \omega_n$ and $S_{\omega^i|\overline{k}} = NULL$ if the last bit of $\omega^i|k$ is 1.

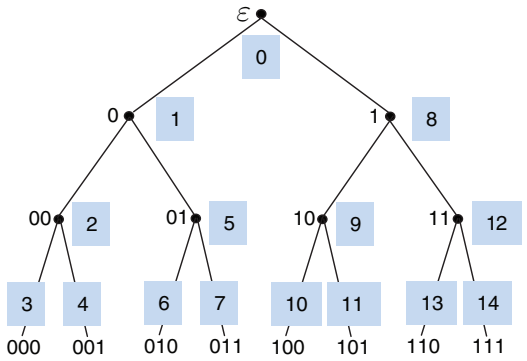

FIGURE 1. the key update algorithm based on the pre-order traversal method of binary trees
time period 0: $SK_0 = \{S_\varepsilon\}$;
time period 1: $SK_1 = \{S_1, S_0\}$;
time period 2: $SK_2 = \{S_1, S_{01}, S_{00}\}$;
time period 3: $SK_3 = \{S_1, S_{01}, S_{001}, S_{000}\}$;
time period 4: $SK_4 = \{S_1, S_{01}, S_{001}\}$;
time period 5: $SK_5 = \{S_1, S_{01}\}$;
time period 6: $SK_6 = \{S_1, S_{011}, S_{010}\}$;
time period 7: $SK_7 = \{S_1, S_{011}\}$;
.....

The process of the key update algorithm can be easily implemented via a stack. The secret key SK_i can be organized as a stack of node keys $STACK_{SK}$, with the node secret key S_{ω^i} on top. At the end of the time period i, the signer runs the key update algorithm, first pops the current node secret key S_{ω^i} off the stack.

(i) If ω^i is an internal node, $\omega^{i+1} = \omega^i 0$, then it generates secret keys $S_{\omega^i 0}$ and $S_{\omega^i 1}$ of $\omega^i 0$ and $\omega^i 1$, respectively, and pushes $S_{\omega^i 1}$ and then $S_{\omega^i 0}$ onto the stack. The new top of the stack is $S_{\omega^i 0}$.

(ii) If ω^i is a leaf, then the next key on top of the stack is $S_{\omega^{i+1}}$.

After generating the secret key of ω^{i+1}, it erases the secret node key S_{ω^i} in storage.

For example, we suppose $l = 3$ and the root secret key is $S_\varepsilon = SN_\varepsilon$; then the key update algorithm based on the pre-order traversal method of binary trees is as in Fig. 1, where the node is in black numbers, and the time period is in black numbers on a blue background. The total time period is $N = 15$, and the time period is 0–14 (represented by a bit string). In our example, node ε is the root node of the binary tree, node 0(1) is the left (right) child node of the root node, node 00(01) is the left (right) child node of the node 0, node 10(11) is the left (right) child node of the node 1 and so on (see Fig. 1).

Then, by the pre-order traversal method of binary trees, we can assign nodes to time periods. Denote the initial time period corresponding to the root node, time period 1 corresponding to node 0, time period 2 corresponding to node 00, ..., time period 14 corresponding to node 111 (see Fig. 1). By *SK* we denote the user's time period secret key and they are organized as a stack of node keys $STACK_{SK}$, and S_i is the node secret key of the binary tree. Therefore, the process of the key update algorithm as shown in Fig. 1.

4.2. Concrete scheme

We now construct a forward-secure CBS scheme **FSIG.CBS** using bilinear maps.

Setup: Given a security parameter 1^k and depth of the binary tree l (the total number of time periods is $N = 2^{l+1} - 1$), the algorithm works as follows:

(i) Let G_1, G_2 be groups of a prime order q in which there exists a bilinear map $e : G_1 \times G_1 \to G_2$.
(ii) Select a random number $s \in \mathbb{Z}_q^*$ as the master secret key *msk*, choose an arbitrary generator $P \in G_1$ and compute $mpk = sP$ as the master public key.
(iii) Choose four secure cryptographic hash functions $H_1 : \{0,1\}^* \times G_1 \to G_1$, $H_2 : \{0,1\}^* \times G_1 \to \mathbb{Z}_q^*$, $H_3 : \{0,1\}^* \times \{0,1\}^* \times G_1 \times G_1 \to G_1$ and $H_4 : \{0,1\}^* \times \{0,1\}^* \times \{0,1\}^* \times G_1 \times G_1 \to G_1$. The system parameters are $params =< G_1, G_2, e, q, P, l, H_1, H_2, H_3, H_4 >$. It is seen that *params* and *mpk* are public, and the algorithm keeps *msk* secret.

UserKeyGen: Given *params*, select a random number $x \in \mathbb{Z}_q^*$ as the user initial secret key SK_0 and compute the user public key $PK_{ID} = xP \in G_1$. Not that $SN_\varepsilon = xH_2(\varepsilon)$ is the root secret key of binary tree corresponding to time period 0, where node ε is the root node of the binary tree. The system public key is $PK = (mpk, PK_{ID}, params)$.

CertGen: Given *params*, *msk*, user public key PK_{ID} and user identity $ID \in \{0,1\}^*$, compute $Q_{ID} = H_1(ID, PK_{ID}) \in G_1$; then output the user certificate $Cert_{ID} = sQ_{ID} \in G_1$.

KeyUpdate: The input is the current time period $i \in [0, N-1)$, the user secret key of current time period $SK_i = STACK_{SK}$ and the user public key PK_{ID}. Let ω be the node corresponding to i. In general, the secret key of the node $\omega = \omega_1 \cdots \omega_n$ consists of $n+1$ group elements[1] and is denoted by $S_\omega = (R_{\omega|1}, R_{\omega|2}, \cdots, R_{\omega|n-1}, R_\omega, SN_\omega)$. For the special case of $\omega = \varepsilon$, we simply have $SN_\varepsilon = S_\varepsilon = xH_2(\varepsilon)$ and the other values are not present. It first pops the secret key S_ω off the stack $STACK_{SK}$ and then updates a secret key with respect to the position of node ω in the tree as follows:

(i) If ω is an internal node, then it chooses random numbers $\rho_{\omega 0}, \rho_{\omega 1} \in \mathbb{Z}_q^*$, and computes $R_{\omega 0} = \rho_{\omega 0} P$, $SN_{\omega 0} = SN_\omega + h_{\omega 0}\rho_{\omega 0}$, $R_{\omega 1} = \rho_{\omega 1} P$ and $SN_{\omega 1} = SN_\omega + h_{\omega 1}\rho_{\omega 1}$, where $h_{\omega 0} = H_2(\omega 0, R_{\omega 0})$ and $h_{\omega 1} = H_2(\omega 1, R_{\omega 1})$. So the left child node secret key is $S_{\omega 0} = (R_{\omega|1}, \ldots, R_{\omega|n-1}, R_\omega, R_{\omega 0}, SN_{\omega 0})$ and the right child node secret key is $S_{\omega 1} = (R_{\omega|1}, \ldots, R_{\omega|n-1}, R_\omega, R_{\omega 1}, SN_{\omega 1})$. Then pushes $S_{\omega 1}$ and $S_{\omega 0}$ in order into the stack, and erases S_ω.
(ii) If ω is a leaf, then only erases S_ω.

Sign: Take as input *params*, the user identity *ID*, the time period $i \in [0, N)$, the secret key $SK_i = STACK_{SK}$, the user certificate $Cert_{ID}$ and message $M \in \{0,1\}^*$, the signer pops the top element in the stack $STACK_{SK}$ and uses it to generate a signature. Let $\omega = \omega_1 \cdots \omega_n$. The algorithm works as follows:

(i) Choose a random number $r \in Z_q^*$ and compute $U = rP$.
(ii) Compute $V = H_3(M, i, U, PK_{ID})$, $W = H_4(M, i, ID, U, PK_{ID})$.
(iii) Compute $FS = Cert_{ID} + SN_\omega \cdot V + rW$.
(iv) The signer outputs $< i, \sigma = (U, FS, R_{\omega|\theta}) >$ where $1 \le \theta \le n$ as the signature of M.

Verify: Given the message/signature pair $(M, \sigma = (U, FS, R_{\omega|\theta}))$ where $1 \le \theta \le n$ in time period i and the system public key *PK*, this algorithm works as follows:

(i) Compute $Q_{ID} = H_1(ID, PK_{ID}) \in G_1$, $V = H_3(M, i, U, PK_{ID})$, $W = H_4(M, i, ID, U, PK_{ID})$, $h_{\omega|\theta} = H_2(\omega|\theta, R_{\omega|\theta})$ for $1 \le \theta \le n$.
(ii) If $e(P, FS) = e(mpk, Q_{ID})e(V, PK_{ID} + \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta})e(U, W)$, then $< i, \sigma = (U, FS, R_{\omega|\theta}) >$ where $1 \le \theta \le n$ is a valid signature of $< i, M >$, output *true*. Otherwise, the signature is invalid, output *false*.

Correctness

If σ is a genuine signature generated from algorithm **Sign**, then

$$
\begin{aligned}
& e(mpk, Q_{ID})e(V, PK_{ID} + \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta})e(U, W) \\
&= e(mpk, Q_{ID})e(PK_{ID}, V)e\left(V, \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta}\right)e(U, W) \\
&= e(mpk, Q_{ID})e(P, xV)e\left(V, \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta} P\right)e(U, W) \\
&= e(P, sQ_{ID})e(P, xV)e\left(P, \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta} V\right)e(P, rW) \\
&= e\left(P, sQ_{ID} + xV + \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta} V + rW\right) \\
&= e(P, Cert_{ID} + SN_\omega \cdot V + rW) \\
&= e(P, FS).
\end{aligned}
$$

[1]The lower case "*n*" is associate with the position of the node in the tree. As shown in Fig 1, for example, node 111($n = 3$), node 10($n = 2$).

5. SECURITY ANALYSIS

This section provides the security analysis on the proposed scheme. We will prove that the security of our scheme **FSIG.CBS** depends on the hardness of the CDH problem on G_1.

THEOREM 1. *If there is a (t, q_R) Type I adaptively chosen message and chosen identity adversary $\mathcal{A}_I$ which makes at most q_R queries to random oracles, and $\mathcal{A}_I$ wins the game defined in Section 3.3 with probability $Succ_{\mathcal{A}_I}^{cma,cida,breakin}$, then there exists another algorithm $\mathcal{B}$ which can solve a random instance of the CDH problem in polynomial time with success probability $Succ_{\mathcal{B},G_1}^{CDH} = \frac{1}{N(q_R+1)}\left(1+\frac{1}{q_R+1}\right) Succ_{\mathcal{A}_I}^{cma,cida,breakin}$.*

Proof. In the proof, we consider hash functions as random oracles. Let $\mathcal{A}_I$ be an adversary attacking **FSIG.CBS**; then we construct an algorithm $\mathcal{B}$ which can make use of $\mathcal{A}_I$ in solving the CDH problem. Let P be the generator of G_1. Algorithm $\mathcal{B}$ is given a challenge $(P, aP, bP) = (P, P_1, P_2)$ and its goal is to compute abP. For that purpose, algorithm $\mathcal{B}$ will simulate the oracles and interact with the adversary $\mathcal{A}_I$ as the subroutine. □

It is seen that $\mathcal{B}$ selects a total time period N and guesses the time period T, where $0 \leq T \leq N-1$. In time period T, $\mathcal{A}_I$ will ask the **breakin** query. Let $\omega^T = \omega_1\omega_2\cdots\omega_s$ be a bit string of the node corresponding to the time period T. Note that $\mathcal{B}$ chooses h_{ω^T}, ρ_{ω^T} and $h_{\omega^T|\bar{\theta}}$, $\rho_{\omega^T|\bar{\theta}}$ at random in $\mathbb{Z}_q^*$, and computes $R_{\omega^T} = \rho_{\omega^T}P$, $R_{\omega^T|\bar{\theta}} = \rho_{\omega^T|\bar{\theta}}P$, where $1 \leq \theta \leq n$ and $\omega_\theta = 0$. We see that $\mathcal{B}$ sets the master public key $mpk = aP = P_1$, where P_1 is the input of the CDH problem. Then $\mathcal{B}$ sets $H_2(\omega^T|\bar{\theta}, R_{\omega^T|\bar{\theta}}) = h_{\omega^T|\bar{\theta}}$ and $H_2(\omega^T, R_{\omega^T}) = h_{\omega^T}$. These result in generating the node secrets contained in the secret key SK_T. Then $\mathcal{B}$ gives $params = < G_1, G_2, e, q, P, l, H_1, H_2, H_3, H_4 >$, mpk and N to $\mathcal{A}_I$. It is found that $\mathcal{B}$ responds to hash queries, user key generation queries, certificate queries, public key replacement queries, corruption queries, sign queries and break-in queries from $\mathcal{A}_I$ as follows.

Chosen Message Attack Phase:

$\mathcal{B}$ initializes $\alpha = 0$. Let $\omega^\alpha = \omega_1\cdots\omega_n$ be the node corresponding to the time period α. We assume that $\mathcal{A}_I$ outputs $d = 0$ after the chosen message attack for period 0. If $d \neq$ **breakin** and $\alpha \neq N$, $\mathcal{A}_I$ continues to make chosen message attacks in the next time period.

UserKeyGen. On a new **UserKeyGen** query ID_i, $\mathcal{B}$ randomly selects $x_{ID_i} \in \mathbb{Z}_q^*$, sets $(SK_{ID_i}, PK_{ID_i}) = (x_{ID_i}, x_{ID_i}P)$, then adds $(ID_i, SK_{ID_i}, PK_{ID_i})$ into the list L_1 and adds $(ID_i, SK_{ID_i}, \overline{PK_{ID_i}})$ into the list L_2, where $\overline{PK_{ID_i}} = PK_{ID_i}$. It is noted that $\mathcal{B}$ returns PK_{ID_i} to $\mathcal{A}_I$.

KeyUpdate. Note that this procedure is done by $\mathcal{B}$ without any requests of $\mathcal{A}_I$. It is just preparing answers for queries of next time periods and **breakin** query. Given current time period α, $\mathcal{B}$ simulates the key update algorithm as follows:

(1) If ω^α is a leaf node or $\alpha = N$, $\mathcal{B}$ skips the key update procedure.
(2) If ω^α is an internal node, then $\mathcal{B}$ selects $h_{\omega^\alpha 0}$, $h_{\omega^\alpha 1}$, $\rho_{\omega^\alpha 0}, \rho_{\omega^\alpha 1} \in \mathbb{Z}_q^*$ at random, and computes $R_{\omega^\alpha 0} = \rho_{\omega^\alpha 0}P$, $R_{\omega^\alpha 1} = \rho_{\omega^\alpha 1}P$, $H_2(\omega^\alpha 0, R_{\omega^\alpha 0}) = h_{\omega^\alpha 0}$, $H_2(\omega^\alpha 1, R_{\omega^\alpha 1}) = h_{\omega^\alpha 1}$.

Then $\mathcal{B}$ can compute the user ID_i's secret key SK_{ID_i} in the next time period α, and use the new secret key SK_{ID_i} to update the list L_1. Note that these result in generating the secret key SK_T.

H_1 **queries**. On a new H_1 **query** $< ID_i, PK_{ID_i} >$, $\mathcal{B}$ chooses a random number $coin_i \in \{0, 1\}$ such that $\Pr[coin = 1] = \delta$, where the value of δ will be determined later.

(i) If $coin_i = 0$, $\mathcal{B}$ chooses a random number $c_i \in \mathbb{Z}_q^*$ and sets $H_1(ID_i, PK_{ID_i}) = c_i P$.
(ii) Else $coin_i = 1$, $\mathcal{B}$ chooses a random number $c_i \in \mathbb{Z}_q^*$ and sets $H_1(ID_i, PK_{ID_i}) = c_i P + P_2$, where P_2 is another input of the CDH problem.

In both cases, $\mathcal{B}$ will add $(< ID_i, PK_{ID_i} >, c_i, coin_i)$ into $H_1 - list$ and return $H_1(ID_i, PK_{ID_i})$ to $\mathcal{A}_I$.

H_2 **queries**. H_2 does not need to be simulated as a random oracle. We suppose that H_2 is a secure cryptographic hash function with collision resistance. The adversary can issue queries to a real hash function.

H_3 **queries**. On a new H_3 **query** $<M_i, \alpha, U_i, PK_{ID_i}>$, $\mathcal{B}$ chooses a random number $d_i \in \mathbb{Z}_q^*$ and sets $H_3(M_i, \alpha, U_i, PK_{ID_i}) = d_i P$. Then $\mathcal{B}$ adds $(<M_i, \alpha, U_i, PK_{ID_i}>, d_i)$ into H_3-list and returns $H_3(M_i, \alpha, U_i, PK_{ID_i})$ to $\mathcal{A}_I$.

H_4 **queries**. On a new H_4 **query** $<M_i, \alpha, ID_i, U_i, PK_{ID_i}>$, $\mathcal{B}$ chooses a random number $\lambda_i \in \mathbb{Z}_q^*$ and sets $H_4(M_i, \alpha, ID_i, U_i, PK_{ID_i}) = \lambda_i P$. Then $\mathcal{B}$ adds $(<M_i, \alpha, ID_i, U_i, PK_{ID_i}>, \lambda_i, \lambda_i P)$ into H_4-list and returns $H_4(M_i, \alpha, ID_i, U_i, PK_{ID_i})$ to $\mathcal{A}_I$.

PKReplace. On a public key replacement query $<ID_i, PK'_{ID_i}>$, this oracle finds the user ID_i in the list L_2, sets $\overline{PK_{ID_i}} = PK'_{ID_i}$ and updates the corresponding information as $<ID_i, PK'_{ID_i}>$.

CertGen. On a certificate query ID_i, $\mathcal{B}$ first checks the list L_1 and L_2 to obtain ID_i's original public key PK_{ID_i} and ID_i's current public key $\overline{PK_{ID_i}}$. If $PK_{ID_i} \neq \overline{PK_{ID_i}}$, it means that ID_i's public key has been replaced by the adversary. In this case, $\mathcal{B}$ rejects to respond. Otherwise, $PK_{ID_i} = \overline{PK_{ID_i}}$, the user ID_i's public key is the original public key returned by **UserKeyGen**; then $\mathcal{B}$ works as follows. First, we assume that $(<ID_i, PK_{ID_i}>, \cdot, \cdot)$ has been in $H_1 - list$. If not, $\mathcal{B}$ adds $(<ID_i, PK_{ID_i}>, c_i, coin_i)$ into $H_1 - list$ in the same way that he responds to H_1 **queries**.

(i) If $coin_i = 0$, which means $Q_{ID_i} = H_1(ID_i, PK_{ID_i}) = c_i P$, $\mathcal{B}$ returns the certificate $Cert_{ID_i} = c_i P_1$ to $\mathcal{A}_I$.
(ii) Otherwise, $\mathcal{B}$ aborts.

Corruption. On a **Corruption** query ID_i in time period α, if $0 \leq \alpha < T$, $\mathcal{B}$ first checks the list L_1 and returns SK_{ID_i} to $\mathcal{A}_I$. Otherwise, $\alpha = T$ and $\mathcal{B}$ aborts.

Sign. On a **Sign** query $< M_i, ID_j >$, $\mathcal{B}$ first checks the list L_1 and L_2 to obtain ID_j's original public key PK_{ID_j} and ID_j's current public key $\overline{PK_{ID_j}}$. If $PK_{ID_j} \neq \overline{PK_{ID_j}}$, $\mathcal{B}$ will ask the adversary to supply the current secret key $\overline{SK_{ID_j}}$ corresponding to $\overline{PK_{ID_j}}$. After that, $\mathcal{B}$ uses $\overline{SK_{ID_j}}$ and the certificate for $\overline{PK_{ID_j}}$ to generate M_i's signature σ_i. Otherwise, $PK_{ID_j} = \overline{PK_{ID_j}}$, and $\mathcal{B}$ responds to its signature query as follows:

If $\alpha \neq T$, $\mathcal{B}$ first checks the list $H_1 - list$ to obtain $(<ID_j, PK_{ID_j}>, c_j, coin_j)$. If $coin_j = 0$, $\mathcal{B}$ can generate the certificate $Cert_{ID_j}$ as he responds to the**CertGen** queries and uses $(Cert_{ID_j}, SK_{ID_j})$ to generate M_i's signature σ_i. Else, $coin_j = 1$ and $H_1(ID_j, PK_{ID_j}) = c_j P + P_2$. Then $\mathcal{B}$ chooses a random number $r_i \in \mathbb{Z}_q^*$ and sets $U_i = r_i P - P_1$.

(i) $\mathcal{B}$ checks $H_3 - list$: if $(<M_i, \alpha, U_j, PK_{ID_j}>, \cdot)$ does not exists in $H_3 - list$, $\mathcal{B}$ will add $(<M_i, \alpha, U_j, PK_{ID_j}>, d_i)$ into $H_3 - list$ in the same way that he responds to H_3 queries.

(ii) $\mathcal{B}$ checks $H_4 - list$: if $(<M_i, \alpha, ID_j, U_i, PK_{ID_j}>, \cdot, \cdot)$ exists in $H_4 - list$, a collision occurs, and $\mathcal{B}$ must reselect the number r_i. Then $\mathcal{B}$ further sets $H_4(M_i, \alpha, ID_j, U_i, PK_{ID_j}) = \lambda_i P + P_2$ and adds $(<M_i, \alpha, ID_j, U_i, PK_{ID_j}>, \lambda_i, \lambda_i P)$ into $H_4 - list$.

(iii) $\mathcal{B}$ computes $FS_i = c_j P_1 + (x_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta} \rho_{\omega|\theta}) H_3(M_i, \alpha, U_i, PK_{ID_j}) + \lambda_i U_i + r_i P_2$ and outputs $\sigma_i = (U_i, FS_i)$ as the signature in time period α.

Correctness

$$
\begin{aligned}
& e(P, FS_i) \\
&= e\left(P, c_j P_1 + \left(x_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta}\right) \times H_3(M_i, \alpha, U_i, PK_{ID_j}) + \lambda_i U_i + r_i P_2\right) \\
&= e\left(P, c_j P_1 + \left(x_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta}\right) \times H_3(M_i, \alpha, U_i, PK_{ID_j}) + \lambda_i U_i + r_i P_2 - abP + abP\right) \\
&= e\left(P, a(c_j P + P_2) + \left(x_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta}\right) \times H_3(M_i, \alpha, U_i, PK_{ID_j}) + \lambda_i(r_i P - P_1) + r_i bP - abP\right) \\
&= e\left(P, a(c_j P + P_2) + \left(x_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta}\right) \times H_3(M_i, \alpha, U_i, PK_{ID_j}) + (r_i - a)\lambda_i P + (r_i - a)P_2\right) \\
&= e\left(P, a(c_j P + P_2) + \left(x_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta}\rho_{\omega|\theta}\right) \times H_3(M_i, \alpha, U_i, PK_{ID_j}) + (r_i - a) H_4(M_i, \alpha, ID_j, U_i, PK_{ID_j})\right) \\
&= e(mpk, Q_{ID_j}) \times e\left(H_3(M_i, \alpha, U_i, PK_{ID_j}), PK_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta}\right) \times e(U_i, H_4(M_i, \alpha, ID_j, U_i, PK_{ID_j})).
\end{aligned}
$$

If $\alpha = T$, $\mathcal{B}$ randomly selects $d_i, \lambda_i, r_i \in \mathbb{Z}_q^*$ and computes $H_3(M_i, \alpha, ID_j, PK_{ID_j}) = d_i P$, $H_4(M_i, \alpha, ID_j, U_i, PK_{ID_j}) = \lambda_i P$, $U_i = r_i P$; $\mathcal{B}$ gives $<T, \sigma = (U_i, FS_i)>$ and $R_{\omega^T|\theta}$ $(1 \leq \theta \leq n)$ to $\mathcal{A}_I$, where $FS = Cert_{ID_j} + SN_{\omega^T} \cdot V + \lambda_i U_i$.

Break-in Phase:

When $\mathcal{A}_I$ outputs a decision value d, $\mathcal{B}$ simulates the **breakin** phase as follows. When $\alpha < T$ and $d = 0$, then A_I increments α and moves into the *cma* phase for period α. When $\alpha = T$ and $d =$ **breakin**, $\mathcal{B}$ returns the current secret key $SK_T = (S_{\omega^T|1}, S_{\omega^T|2}, \cdots, S_{\omega^T|n-1}, S_{\omega^T})$, where $\omega^T = \omega_1\omega_2\cdots\omega_n$, as the response of **breakin** query to $\mathcal{A}_I$. If none of the above cases occur, $\mathcal{B}$ fails and aborts. Note that if $\mathcal{A}_I$ comes into the **breakin** phase, it cannot get access to the previous oracle.

Forgery phase:

After the above attack process, $\mathcal{A}_I$ outputs a forgery $<\gamma, M^*, \sigma^* = (U^*, FS^*), ID^*, PK_{ID^*}>$ for $0 \leq \gamma < T$, $\omega^\gamma = \omega_1\omega_2\cdots\omega_n$. It is seen that PK_{ID^*} is chosen by $\mathcal{A}_I$ and might not be ID^*'s public key output from the oracle **UserKeyGen**. We assume that $(<ID^*, PK_{ID^*}>, c^*, coin^*)$, $(<M^*, \gamma, U^*, PK_{ID^*}>, d^*)$, $(<M^*, \gamma, ID^*, U^*, PK_{ID^*}>, \lambda^*, \lambda^* P)$ have been in $H_1 - list$, $H_3 - list$ and $H_4 - list$, respectively. If σ^* is a valid signature of the message M^* in time period γ, then

$$
\begin{aligned}
FS^* &= Cert_{ID^*} + SN_{\omega^\gamma} \cdot V^* + r^*\lambda^* P \\
&= a H_1(ID^*, PK_{ID^*}) + \left(x_{ID^*} + \sum_{\theta=1}^{n} h_{\omega^\gamma|\theta}\rho_{\omega^\gamma|\theta}\right) \cdot d^* P + \lambda^* U^* \\
&= a H_1(ID^*, PK_{ID^*}) + d^* PK_{ID^*} + \sum_{\theta=1}^{n} d^* h_{\omega^\gamma|\theta} R_{\omega^\gamma|\theta} + \lambda^* U^*.
\end{aligned}
$$

(1) If $coin^* = 1$ and $d =$ **breakin**, $H_1(ID^*, PK_{ID^*}) = c^* P + P_2$, $\mathcal{B}$ can compute

$$
abP = FS^* - \left(c^* P_1 + d^* PK_{ID^*} + \sum_{\theta=1}^{n} d^* h_{\omega^\gamma|\theta} R_{\omega^\gamma|\theta} + \lambda^* U^*\right).
$$

(2) Otherwise, $\mathcal{B}$ fails to solve this instance of the CDH problem.

According to the simulation, $\mathcal{B}$ can compute the value of *abP* if and only if all the following four events happen:

Event E_1: $\mathcal{B}$ does not abort during the simulation.
Event E_2: $\mathcal{A}_I$ outputs $\alpha = T$ and $d = breakin$.
Event E_3: $\mathcal{A}_I$ outputs a valid forgery in time period γ for $0 \leq \gamma < T$.
Event E_4: In the forgery output by $\mathcal{A}_I$, $coin^* = 1$.

Therefore, the probability that $\mathcal{B}$ solves this instance of the CDH problem is $Succ^{CDH}_{\mathcal{B},G_1} = \Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4] = \Pr[E_1]\Pr[E_2|E_1]\Pr[E_3|E_1 \wedge E_2]\Pr[E_4|E_1 \wedge E_2 \wedge E_3]$. All simulations can be done in polynomial time. From the simulation, we have $\Pr[E_1] \geq (1-\delta)^{q_R}$, $\Pr[E_2|E_1] \geq 1/N$, $\Pr[E_3|E_1 \wedge E_2] = Succ^{cma,cida,breakin}_{\mathcal{A}_I}$ and $\Pr[E_4|E_1 \wedge E_2 \wedge E_3] = \delta$. Thus, $Succ^{CDH}_{\mathcal{B},G_1} \geq \delta(1-\delta)^{q_R} Succ^{cma,cida,breakin}_{\mathcal{A}_I}/N$. When $\delta = 1/(q_R+1)$, this probability is maximized at

$$Succ^{CDH}_{\mathcal{B},G_1} = \frac{1}{N(q_R+1)}\left(1+\frac{1}{q_R+1}\right) Succ^{cma,cida,breakin}_{\mathcal{A}_I}.$$

THEOREM 2. *If there is a (t, q_R) Type II adaptively chosen message and chosen identity adversary $\mathcal{A}_{II}$ which makes at most q_R queries to random oracles, and $\mathcal{A}_{II}$ wins the game defined in Section 3.4 with probability $Succ^{cma,cida,breakin}_{\mathcal{A}_{II}}$, then there exists another algorithm $\mathcal{B}$ which can solve a random instance of the CDH problem in polynomial time with success probability*

$$Succ^{\mathrm{CDH}}_{\mathcal{B},G_1} \geq (1-1/q')^{q_R} Succ^{cma,cida,breakin}_{\mathcal{A}_{II}}/(q'N),$$

where $1 \neq q' \leq q_R$ denotes the number of queries submitted to the oracle **UserKeyGen**.

Proof. Like the proof of Theorem 1, we consider hash functions as random oracles. Let $\mathcal{A}_{II}$ be an adversary attacking **FSIG.CBS**; then we construct an algorithm $\mathcal{B}$ which can solve the CDH problem. Let P be the generator of G_1. Algorithm $\mathcal{B}$ is given a challenge (P, P_1, P_2), where $P_1 = aP \in G_1$, $P_2 = bP \in G_1$, and its goal is to compute *abP*. Algorithm $\mathcal{B}$ will simulate the oracles and interact with the adversary $\mathcal{A}_{II}$ as the subroutine. □

It is noted that $\mathcal{B}$ selects a total time period N and guesses the time period T, where $0 \leq T \leq N-1$; $\mathcal{A}_I$ will ask the **breakin** query in time period T. Let $\omega^{\mathrm{T}} = \omega_1\omega_2\cdots\omega_n$ be a bit string of the node corresponding to the time period T. It is seen that $\mathcal{B}$ chooses ρ_{ω^T}, $h_{\omega^{\mathrm{T}}}$ and $\rho_{\omega^{\mathrm{T}}|\bar{\theta}}$, $h_{\omega^{\mathrm{T}}|\bar{\theta}}$ at random in $\mathbb{Z}^*_q$, and computes $R_{\omega^{\mathrm{T}}} = \rho_{\omega^T}P$, $R_{\omega^{\mathrm{T}}|\bar{\theta}} = \rho_{\omega^{\mathrm{T}}|\bar{\theta}}P$, where $1 \leq \theta \leq n$ and $\omega_\theta = 0$. It is found that $\mathcal{B}$ sets $H_2(\omega^{\mathrm{T}}|\bar{\theta}, R_{\omega^{\mathrm{T}}|\bar{\theta}}) = h_{\omega^{\mathrm{T}}|\bar{\theta}}$ and $H_2(\omega^{\mathrm{T}}, R_{\omega^{\mathrm{T}}}) = h_{\omega^{\mathrm{T}}}$; $\mathcal{B}$ selects a random number $s' \in \mathbb{Z}^*_q$, and sets the master secret key $msk = s'$ and the master public key $mpk = s'P$. Then $\mathcal{B}$ gives *msk*, *mpk*, N and *params* $=< G_1, G_2, e, q, P, l, H_1, H_2, H_3, H_4 >$ to $\mathcal{A}_{II}$. It is noted that $\mathcal{B}$ simulates hash queries, user key generation queries, corruption queries, sign queries and the break-in query from $\mathcal{A}_{II}$.

Chosen Message Attack Phase:

$\mathcal{B}$ initializes $\alpha = 0$. Let $\omega^\alpha = \omega_1\cdots\omega_n$ be the node corresponding to the time period α. We assume that $\mathcal{A}_{II}$ outputs $d = 0$ after the chosen message attack for period 0. If $d \neq$ **breakin** or $\alpha \neq N$, $\mathcal{A}_{II}$ moves into the next time period and continues to make chosen message attacks.

UserKeyGen. In time period 0, $\mathcal{A}_{II}$ can submit some **UserKeyGen** queries, and $\mathcal{B}$ acts as follows. Suppose that there are up to q' **UserKeyGen** queries; then $\mathcal{B}$ will choose a random number $\pi \in \{1, 2, \cdots, q'\}$.

(i) If ID_i is the π^{th} query, $\mathcal{B}$ sets $SK_{ID_i} = \perp$. Here, $\perp$ indicates that $\mathcal{B}$ does not know the corresponding value.
(ii) Otherwise, $\mathcal{B}$ chooses a random number $SK_{ID_i} \in \mathbb{Z}^*_q$ and sets $PK_{ID_i} = SK_{ID_i}P$.

Then $\mathcal{B}$ adds $(ID_i, SK_{ID_i}, PK_{ID_i})$ into the list L_1 and returns PK_{ID_i} to $\mathcal{A}_{II}$.

KeyUpdate. Note that this procedure is done by $\mathcal{B}$ without any requests of $\mathcal{A}_{II}$. It is just preparing answers for queries of next time periods and **breakin** query. Given current time period α, $\mathcal{B}$ simulates the key update algorithm as follows.

(i) If ω^α is a leaf node or $\alpha = N$, $\mathcal{B}$ skips the key update procedure.
(ii) Otherwise, ω^α is an internal node; then $\mathcal{B}$ selects $h_{\omega^\alpha 0}, h_{\omega^\alpha 1}, \rho_{\omega^\alpha 0}, \rho_{\omega^\alpha 1} \in \mathbb{Z}^*_q$ at random and computes $R_{\omega^\alpha 0} = \rho_{\omega^\alpha 0}P$, $R_{\omega^\alpha 1} = \rho_{\omega^\alpha 1}P$, $H_2(\omega^\alpha 0, R_{\omega^\alpha 0}) = h_{\omega^\alpha 0}$, $H_2(\omega^\alpha 1, R_{\omega^\alpha 1}) = h_{\omega^\alpha 1}$.

Then $\mathcal{B}$ can compute the user ID_i's secret key SK_{ID_i} in the next time period α, and use the new secret key SK_{ID_i} to update the list L_1.

H_1 **queries**. On a new H_1 **query** $<ID_i, PK_{ID_i}>$, $\mathcal{B}$ chooses a random number $c_i \in \mathbb{Z}^*_q$ and sets $H_1(ID_i, PK_{ID_i}) = c_iP$. It is found that $\mathcal{B}$ will add $(<ID_i, PK_{ID_i}>, c_i)$ into $H_1 - list$ and return $H_1(ID_i, PK_{ID_i})$ to $\mathcal{A}_{II}$.

H_2 **queries** H_2 does not need to be simulated as a random oracle. We suppose that H_2 is a secure cryptographic hash function with collision resistance. The adversary can issue queries to a real hash function.

H_3 **queries**. On a new H_3 **query** $<M_i, \alpha, U_i, PK_{ID_i}>$, $\mathcal{B}$ chooses a random number $d_i \in \mathbb{Z}^*_q$ and sets $H_3(M_i, \alpha, U_i, PK_{ID_i}) = d_iP + P_2$. Then $\mathcal{B}$ adds $(<M_i, \alpha, U_i, PK_{ID_i}>, d_i, d_iP + P_2)$ into $H_3 - list$ and returns $H_3(M_i, \alpha, U_i, PK_{ID_i})$ to $\mathcal{A}_{II}$.

H_4 **queries**. On a new H_4 **query** $<M_i, \alpha, ID_i, U_i, PK_{ID_i}>$, $\mathcal{B}$ chooses a random number $\lambda_i \in \mathbb{Z}^*_q$ and sets $H_4(M_i, \alpha, ID_i, U_i, PK_{ID_i}) = \lambda_iP$. Then $\mathcal{B}$ adds $(<M_i, \alpha, ID_i, U_i, PK_{ID_i}>, \lambda_i, \lambda_iP)$ into $H_4 - list$ and returns $H_4(M_i, \alpha, ID_i, U_i, PK_{ID_i})$ to $\mathcal{A}_{II}$.

Corruption. On a **Corruption** query ID_i in time period α, if $0 \leq \alpha < T$, $\mathcal{B}$ first checks the list L_1 and returns SK_{ID_i} to $\mathcal{A}_I$. Otherwise, $\alpha = T$ or $SK_{ID_i} = \perp$, and $\mathcal{B}$ aborts.

Sign. On a **Sign** query $<M_i, ID_j>$, $\mathcal{B}$ responds to its query as follows:

If $\alpha \neq T$, $\mathcal{B}$ first checks the list $H_1 - list$ to obtain $(<ID_j, PK_{ID_j}>, c_j)$.

(1) If $SK_{ID_i} = \perp$, $\mathcal{B}$ will choose random numbers $U_i = r_i P \in G_1$ and $d_i \in \mathbb{Z}_q^*$. Then $\mathcal{B}$ adds $(<M_i, \alpha, U_i, PK_{ID_i}>, d_i, d_i P)$ into $H_3 - list$. If a collision occurs, $\mathcal{B}$ reselects U_i and d_i. In addition, $\mathcal{B}$ will add $(<M_i, \alpha, ID_j, U_i, PK_{ID_j}>, \lambda_i, \lambda_i P)$ into $H_4 - list$, and then respond to H_4 queries. We suppose that $(<ID_j, PK_{ID_j}>, c_j)$ has already been in $H_1 - list$. It is noted that $\mathcal{B}$ computes $FS_i = Cert_{ID_j} + d_i PK_{ID_j} + \sum_{\theta=1}^{n} d_i h_{\omega|\theta} R_{\omega|\theta} + \lambda_i U_i$, and returns signature $\sigma_i = (U_i, FS_i)$ to $\mathcal{A}_{II}$.

Correctness

$$
\begin{aligned}
&e(P, FS_i)\\
&\quad = e\left(P, Cert_{ID_j} + d_i PK_{ID_j} + \sum_{\theta=1}^{n} d_i h_{\omega|\theta} R_{\omega|\theta} + \lambda_i U_i\right)\\
&\quad = e(P, Cert_{ID_j}) e\left(P, d_i SK_{ID_j} P + \sum_{\theta=1}^{n} d_i h_{\omega|\theta} R_{\omega|\theta}\right)\\
&\qquad \times e(P, \lambda_i \cdot r_i P)\\
&\quad = e(mpk, H_1(ID_j, PK_{ID_j}))\\
&\qquad \times e\left(H_3(M_i, \alpha, U_i, PK_{ID_j}), PK_{ID_j} + \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta}\right)\\
&\qquad \times e(H_4(M_i, \alpha, ID_j, U_i, PK_{ID_j}), U_i)
\end{aligned}
$$

(2) Otherwise, $\mathcal{B}$ uses $Cert_{ID_j}$ and SK_{ID_j} to generate M_i's signature in the time period as the response of **Sign** query to $\mathcal{A}_{II}$.

If $\alpha = T$, $\mathcal{B}$ randomly selects $d_i, \lambda_i, r_i \in \mathbb{Z}_q^*$ and computes $H_3(M_i, \alpha, U_i, PK_{ID_j}) = d_i P$, $H_4(M_i, \alpha, ID_j, U_i, PK_{ID_j}) = \lambda_i P$, $U_i = r_i P$. $\mathcal{B}$ gives $<T, \sigma = (U_i, FS_i)>$ and $R_{\omega^T|\theta} (1 \leq \theta \leq n)$ to $\mathcal{A}_{II}$, where $FS = Cert_{ID_j} + SN_{\omega^T} \cdot V + \lambda_i U_i$.

Break-in Phase:

When $\mathcal{A}_{II}$ outputs a decision value d, $\mathcal{B}$ simulates **breakin** phase as follows. When $\alpha < T$ and $d = 0$, then A_{II} increments α and moves into the **cma** phase for period α. When $\alpha = T$ and $d =$ **breakin**, $\mathcal{B}$ returns the current secret key $SK_T = (S_{\omega^T|1}, S_{\omega^T|2}, \ldots, S_{\omega^T|n-1}, S_{\omega^T})$, where $\omega^{\mathrm{T}} = \omega_1 \omega_2 \cdots \omega_n$, as the response of **breakin** query to $\mathcal{A}_{II}$. If none of the above cases occur, $\mathcal{B}$ fails and aborts. Note that if $\mathcal{A}_{II}$ comes into the **breakin** phase, it cannot get access to the previous oracle.

Forgery phase:

After the above attack process, $\mathcal{A}_{II}$ outputs a forgery $<\gamma, M^*, \sigma^* = (U^*, FS^*), ID^*, PK_{ID^*}>$ for $0 \leq \gamma < T$, $\omega^{\gamma} = \omega_1 \omega_2 \cdots \omega_n$. Here, PK_{ID^*} must be the genuine public key of ID^*. We assume that $(<ID^*, PK_{ID^*}>, c^*)$, $(<M^*, \gamma, U^*, PK_{ID^*}>, d^*, d^* P)$ and $(<M^*, \gamma, ID^*, U^*, PK_{ID^*}>, \lambda^*, \lambda^* P)$ have been in $H_1 - list$, $H_3 - list$ and $H_4 - list$, respectively. If σ^* is a valid signature of the message M^* in time period γ, then $FS^* = s' \cdot c^* P + SN_{\gamma}(d^* P + P_2) + \lambda^* U^*$.

If $PK_{ID^*} = P_1$, ID^*'s initial key SK_{ε} should be a. Thus, $\mathcal{B}$ can compute

$$abP = FS^* - (s' \cdot c^* P + d^* P_1 + \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta} (d^* P + P_2) + \lambda^* U^*).$$

Correctness

$$
\begin{aligned}
FS^* &= s' \cdot c^* P + SN_{\gamma}(d^* P + P_2) + \lambda^* U^*\\
&= s' \cdot c^* P + \left(a + \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta}\right)(d^* P + P_2) + \lambda^* U^*\\
&= s' \cdot c^* P + d^* P_1 + abP\\
&\quad + \sum_{\theta=1}^{n} h_{\omega|\theta} R_{\omega|\theta} (d^* P + P_2) + \lambda^* U^*.
\end{aligned}
$$

Otherwise, $\mathcal{B}$ fails to solve this instance of the CDH problem.

According to the simulation, $\mathcal{B}$ can compute the value of abP if and only if all the following four events occur:

Event E_1: $\mathcal{B}$ does not abort during the simulation.
Event E_2: $\mathcal{A}_{II}$ outputs $\alpha = T$ and $d =$ **breakin**.
Event E_3: $\mathcal{A}_{II}$ outputs a valid forgery in time period γ for $0 \leq \gamma < T$.
Event E_4: In the forgery output by $\mathcal{A}_{II}$, $PK_{ID^*} = P_1$.

Therefore, the probability that $\mathcal{B}$ can solve this instance of the CDH problem is $Succ_{\mathcal{B},G_1}^{CDH} = \Pr[E_1 \wedge E_2 \wedge E_3 \wedge E_4] = \Pr[E_1] \Pr[E_2|E_1] \Pr[E_3|E_1 \wedge E_2] \Pr[E_4|E_1 \wedge E_2 \wedge E_3]$. All simulation can be done in polynomial time. From the simulation, we have $\Pr[E_1] \geq (1 - 1/q')^{q_R}$, $\Pr[E_2|E_1] \geq 1/N$, $\Pr[E_3|E_1 \wedge E_2] = Succ_{\mathcal{A}_{II}}^{cma,cida,breakin}$ and $\Pr[E_4|E_1 \wedge E_2 \wedge E_3] = 1/q'$. Thus, $Succ_{\mathcal{B},G_1}^{CDH} \geq (1 - 1/q')^{q_R} Succ_{\mathcal{A}_{II}}^{cma,cida,breakin}/q'N$, where $1 \neq q' \leq q_R$ denotes the number of queries submitted to the oracle **UserKeyGen**.

6. CONCLUSION

In this paper, we first introduced forward security into CBSs and defined the security models of forward-secure CBSs. We then constructed a forward-secure CBS scheme. Based on the CDH assumption, our scheme is proved existentially unforgeable against adaptive chosen message attacks in the random oracle model. Our design uses the pre-order traversal method of the binary tree to construct the forward-secure signature and improves the efficiency of signature generation and verification. However, how to make the signature generation and verification independent of binary tree hierarchy is a problem worth further investigation. Recently, Buchmann *et al.* [39, 40] proposed an efficient post-quantum forward-secure signature scheme with minimal security assumptions. Furthermore, they presented

the first implementation of a forward-secure signature scheme on a smart card, which solved the problem of on-card key generation and reduced the key generation time. Abdalla *et al.* [41] proposed a forward-secure signature scheme with tighter reductions. They showed that the tighter security reductions provided by their proof methodology could result in concrete efficiency gains in practice. Therefore, another open problem is how to construct an efficient forward-secure CBS scheme with tighter security reductions.

ACKNOWLEDGEMENTS

We would like to thank anonymous referees for their helpful comments and suggestions to improve our paper.

FUNDING

This work was supported by the National Natural Science Foundation of China (60842002, 61272542, 61202450, 61103183, 61103184); the Fundamental Research Funds for the Central Universities (B13020070, 2010B07114); China Postdoctoral Science Foundation Funded Project (20100471373); the "Six Talent Peaks Program" of Jiangsu Province of China (2009182); Distinguished Young Scholars Fund of Department of Education, Fujian Province, China (JA13062); Ph.D. Programs Foundation of Ministry of Education of China (20123503120001); and Program for New Century Excellent Talents in Hohai University.

REFERENCES

[1] Shamir, A. (1979) How to share a secret. *Comm. ACM*, **22**, 612–613.

[2] Desmedt, Y. and Frankel, Y. (1990) Threshold Cryptosystems. *Proc. CRYPTO 89*, Santa Barbara, CA, USA, August 20–21, Lecture Notes in Computer Science 435, pp. 307–315. Springer, Berlin.

[3] Ostrovsky, R. and Yung, M. (1991) How to Withstand Mobile Virus Attacks. *Proc. 10th ACM Symp. on Principles of Distributed Computing (PODC 91)*, Washington, DC, USA, October 27–31, pp. 51–59. ACM, New York, NY, USA.

[4] Dziembowski, S. and Pietrzak, K. (2008) Leakage-Resilient Cryptography. *Proc. FOCS 2008*, Philadelphia, PA, October 26–28, pp. 293–302. IEEE Computer Society Press, Los Alamitos, CA, USA.

[5] Alwen, J., Dodis, Y. and Wichs, D. (2009) Leakage-Resilient Public-Key Cryptography in the Bounded-Retrieval Model. *Proc. CRYPTO 2009*, Santa Barbara, CA, USA, August 16–20, Lecture Notes in Computer Science 5677, pp. 36–54. Springer, Berlin.

[6] Naor, M. and Segev, G. (2009) Public-Key Cryptosystems Resilient to Key Leakage. *Proc. CRYPTO 2009*, Santa Barbara, CA, USA, August 16–20, Lecture Notes in Computer Science 5677, pp. 18–35. Springer, Berlin.

[7] Alwen, J., Dodis, Y., Naor, M., Segev, G., Walfish, S. and Wichs, D. (2010) Public-Key Encryption in the Bounded-Retrieval Model. *Proc. EUROCRYPT 2010*, French Riviera, 30 May–3 June, Lecture Notes in Computer Science 6110, pp. 113–134. Springer, Berlin.

[8] Chow, S., Dodis, Y., Rouselakis, Y. and Waters, B. (2010) Practical Leakage-Resilient Identity-Based Encryption from Simple Assumptions. *Proc. 17th ACM Conf. on Computer and Communications Security (CCS 2010)*, Chicago, IL, USA, October 4–8, pp. 152–161. ACM, New York, NY, USA.

[9] Katz, J. and Vaikuntanathan, V. (2009) Signature Schemes with Bounded Leakage Resilience. *Proc. ASIACRYPT 2009*, Tokyo, Japan, December, Lecture Notes in Computer Science 5912, pp. 703–720. Springer, Berlin.

[10] Günther, G. (1990) An Identity-Based Key-Exchange Protocol. *Proc. EUROCRYPT 89*, Houthalen, Belgium, April 10–13, Lecture Notes in Computer Science 2501, pp. 29–37. Springer, Berlin.

[11] Anderson, R. (1997) Invited Lecture. *Proc. 4th Annual Conf. on Computer and Communications Security (CCS 97)*, Zurich, Switzerland, April 2–4. ACM, New York, NY, USA.

[12] Bellare, M. and Miner, S.K. (1999) A Forward-Secure Digital Signature Scheme. *Proc. CRYPTO 99*, Santa Barbara, CA, USA, August 15–19, Lecture Notes in Computer Science 1666, pp. 431–448. Springer, Berlin.

[13] Fiat, A. and Shamir, A. (1987) How to Prove Yourself: Practical Solution of Identification and Signature Problem. *Proc. CRYPTO 86*, Santa Barbara, CA, USA, August 11–15, Lecture Notes in Computer Science 263, pp. 186–196. Springer, Berlin.

[14] Ong, H. and Schnorr, C. (1991) Fast Signature Generation with a Fiat-Shamir Like Scheme. *Proc. EUROCRYPT 90*, Aarhus, Denmark, May 21–24, Lecture Notes in Computer Science 473, pp. 432–440. Springer, Berlin.

[15] Krawczyk, H. (2000) Simple Forward-Secure Signatures from Any Signature Scheme. *Proc. 7th ACM Conf. on Computer and Communications Security (CCS 2000)*, Athens, Greece, November 1–4, pp. 108–115. ACM, New York, NY, USA.

[16] Abdalla, M. and Reyzin, L. (2000) A New Forward-Secure Digital Signature Scheme. *Proc. ASIACRYPT 2000*, Kyoto, Japan, December 3–7, Lecture Notes in Computer Science 1976, pp. 116–129. Springer, Berlin.

[17] Tzeng, W.G. and Tzeng, Z.J. (2001) Robust Forward-Secure Signature Schemes with Proactive Security. *Proc. PKC 2001*, Cheju Island, Korea, February 13–15, Lecture Notes in Computer Science 1992, pp. 264–276. Springer, Berlin.

[18] Itkis, G. and Reyzin, L. (2001) Forward-Secure Signature with Optical Signing and Verifying. *Proc. CRYPTO 2001*, Santa Barbara, CA, USA, August 19–23, Lecture Notes in Computer Science 2139, pp. 332–354. Springer, Berlin.

[19] Malkin, T., Micciancio, D. and Miner, S. (2002) Efficient Generic Forward-Secure Signatures with an Unbounded Number of Time Periods. *Proc. EUROCRYPT 2002*, Amsterdam, Netherlands, 28 April–2 May Lecture Notes in Computer Science 2332, pp. 400–417. Springer, Berlin.

[20] Kang, B.G., Park, J.H., Hahn, S.G. A new forward secure signature scheme. Cryptology ePrint Archive, Report 2004/183. Available from http://eprint.iacr.org/2004/183/.

[21] Song, X.D. (2001) Practical Forward Secure Group Signature Schemes. *Proc. 8th ACM Conf. on Computer and Communications Security (CCS 2001)*, Philadelphia, PA, USA, November 6–8, pp. 225–234. ACM, New York, NY, USA.

[22] Boyen, X., Shacham, H., Shen, E. and Waters, B. (2006) Forward-Secure Signatures with Untrusted Update. *Proc. 13th ACM Conf. on Computer and Communications Security (CCS 2006)*, Alexandria, VA, USA, October 30–November 3, pp. 191–200. ACM, New York, NY, USA.

[23] Libert, B., Quisquater, J. and Yung, M. (2007) Forward-Secure Signatures in Untrusted Update Environments: Efficient and Generic Constructions. *Proc. 14th ACM Conf. on Computer and Communications Security (CCS 2007)*, Alexandria, VA, USA, 29 October–2 November, pp. 266–275. ACM, New York, NY, USA.

[24] Alomair, B., Sampigethaya, K. and Poovendran, R. (2008) Efficient Generic Forward-Secure Signatures and Proxy Signatures. *Proc. EuroPKI 2008*, Trondheim, Norway, June 16–17, Lecture Notes in Computer Science 5057, pp. 166–181. Springer, Berlin.

[25] Gentry, C. (2003) Certificate-Based Encryption and the Certificate Revocation Problem. *Proc. EUROCRYPT 2003*, Warsaw, Poland, May 4–8, Lecture Notes in Computer Science 2656, pp. 272–293. Springer, Berlin.

[26] Kang, B.G., Park, J.H. and Hahn, S.G. (2004) A Certificate-Based Signature Scheme. *Proc. CT-RSA 2004*, Moscone Center, San Francisco, USA, February 23–27, Lecture Notes in Computer Science 2964, pp. 99–111. Springer, Berlin.

[27] Li, J.G., Huang, X.Y., Mu, Y., Susilo, W. and Wu, Q.H. (2007) Certificate-Based Signature: Security Model and Efficient Construction. *Proc. EuroPKI'2007*, Palma de Mallorca, Spain, June 28–30, Lecture Notes in Computer Science 4582, pp. 110–125. Springer, Berlin.

[28] Wu, W., Mu, Y., Susilo, W. and Huang, X.Y. (2009) Certificate-Based Signatures: New Definitions and a Generic Construction from Certificateless Signatures. *Proc. WISA 2008*, Jeju Island, Korea, September 23–25, Lecture Notes in Computer Science 5379, pp. 99–114. Springer, Berlin.

[29] Wu, W., Mu, Y., Susilo, W. and Huang, X.Y. (2009) Certificate-based signatures revisited. *J. Univers. Comput. Sci.*, **15**, 1659–1684.

[30] Li, J.G., Huang, X.Y., Mu, Y., Susilo, W. and Wu, Q.H. (2010) Constructions of certificate-based signature secure against key replacement attacks. *J. Comput. Secur.*, **18**, 421–449.

[31] Li, J.G., Huang, X.Y., Zhang, Y.C., Xu, L.Z. (2012) An efficient short certificate-based signature scheme. *J. Syst. Softw.*, **85**, 314–322.

[32] Li, J.G., Huang, X.Y., Hong, M.X. and Zhang, Y.C. (2012) Certificate-based signcryption with enhanced security features. *Comput. Math. Appl.*, **64**, 1587–1601.

[33] Balfe, S., Gallery, E., Mitchell, C.J. and Paterson, K.G. (2008) Challenges for trusted computing. Technical Report, RHUL-MA-2008-14, 26 February 2008. http://www.rhul.ac.uk/mathematics/techreports.

[34] Sadeghi, A. and Stüble, C. (2004) Property-Based Attestation for Computing Platforms: Caring about Properties, Not Mechanisms. *Proc. NSPW 2004*, Nova Scotia, Canada, September 20–23, pp. 67–77. Springer, Berlin.

[35] Chen, L., Landfermann, R., Löhr, H., Rohe, M., Sadeghi, A. and Stüble, C. (2006) A Protocol for Property-Based Attestation. *Proc. 1st ACM Workshop on Scalable Trusted Computing (STC 2006)*, Alexandria, VA, USA, 30 October–3 November, pp. 7–16. ACM, New York, NY, USA.

[36] Chen, L., Löhr, H., Manulis, M. and Sadeghi, A. (2008) Property-Based Attestation without a Trusted Third Part. *Proc. ISC 2008*, Taipei, Taiwan, September 15–18, Lecture Notes in Computer Science 5222, pp. 31–46. Springer, Berlin.

[37] Canetti, R., Halevi, S. and Katz, J. (2007) A forward-secure public-key encryption scheme. *J. Cryptol.*, **20**, 265–294.

[38] Canetti, R., Halevi, S. and Katz, J. (2003) A Forward-Secure Public-Key Encryption Scheme. *Proc. EUROCRYPT 2003*, Warsaw, Poland, May 4–8, Lecture Notes in Computer Science 2656, pp. 255–271. Springer, Berlin.

[39] Buchmann, J., Dahmen, E. and Hülsing, A. (2011) XMSS-a Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. *Proc. PQCrypto 2011*, Taipei, Taiwan, 29 November–2 December, Lecture Notes in Computer Science 7071, pp. 117–129. Springer, Berlin.

[40] Hülsing, A., Busold, C. and Buchmann, J. (2013) Forward Secure Signatures on Smart Cards. *Proc. Selected Areas in Cryptography (SAC 2012)*, Windsor, ON, Canada, August 15–16, Lecture Notes in Computer Science 7707, pp. 66–80. Springer, Berlin.

[41] Abdalla, M., Hamouda, F.B. and Pointcheval, D. (2013) Tighter Reductions for Forward-Secure Signature Schemes. *Proc. 16th Int. Conf. on Practice and Theory in Public-Key Cryptography (PKC 2013)*, Nara, Japan, 26 February–1 March, Lecture Notes in Computer Science 7778, pp. 292–311. Springer, Berlin.