www.czasopisma.pan.pl PAN www.journals.pan.pl

jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2021, VOL. 67, NO. 4, PP. 717–726
Manuscript received August 30, 2021; revised November, 2021. DOI: 10.24425/ijet.2021.137868

Binary Tree Based Forward Secure Signature Scheme in the Random Oracle Model

Mariusz Jurkiewicz

***Abstract*—In this paper we construct and consider a new group-based digital signature scheme with evolving secret key, which is built using a bilinear map. This map is an asymmetric pairing of Type 3, and although, for the reason of this paper, it is treated in a completely abstract fashion it ought to be viewed as being actually defined over $E(\mathbb{F}_{q^n})[p] \times E(\mathbb{F}_{q^{nk}})[p] \rightarrow \mathbb{F}_{q^{nk}}[p]$. The crucial element of the scheme is the key updater algorithm. With the adoption of pairings and binary trees where a number of leaves is the same as a number of time periods, we are assured that an updated secret key can not be used to recover any of its predecessors. This, in consequence, means that the scheme is forward-secure. To formally justify this assertion, we conduct analysis in fu-cma security model by reducing the security of the scheme to the computational hardness of solving the Weak ℓ-th Bilinear Diffie-Hellman Inversion problem type. We define this problem and explain why it can be treated as a source of security for cryptographic schemes. As for the reduction itself, in general case, it could be possible to make only in the random oracle model.**

***Keywords*—forward secure digital signature scheme, bilinear pairing of Type 3, random-oracle model, bilinear Diffie-Hellman inversion problem**

I. Introduction

THE concept of forward-secure signature schemes refers to the security model in which leaking the private key related with a certain time frame does not essentially influence on the unforgeability of the scheme for the time periods prior to this leakage [2]. This model is strictly connected with so called signature schemes with evolving private key [2], [6]. Such schemes are characterized by the fact that, roughly speaking, the lifetime of a public key is split into a some number of subperiods with associated different secret keys. More precisely, at the beginning both a public key and an initial secret key are generated and assigned to the first period, next the initial key is updated to the next period and so on until reaching the last period. This, in turn, means that the updating mechanism is of crucial importance in this security model. Namely, besides justification of unforgeability for singular time frames, it must be proven, above all, that disclosure of a certain secret key reveals nothing about the past periods. In other words, this mechanism has to fulfill a nontrivial property which can be called a "memory loss", meaning that a secret key associated with a given time frame must store nothing but data required for making current signatures and generating a key for the next period, moreover this data must be useless with regard to the previous periods.

M. Jurkiewicz is with Faculty of Cybernetics, Military University of Technology, Warsaw, Poland (e-mail: mariusz.jurkiewicz@wat.edu.pl).

II. Contribution

In this paper we construct a signature scheme with evolving secret key, which is based on Type 3 pairing, defined in [11], and prove that the scheme is forward-secure in the random oracle model.

The security proof is conducted by reducing the whole analysis to considerations which pertain to the difficulty of solving a certain computational problem that we call $(\ell, 1)$-wBDHI$^*_3$, and define formally in Section III-A. This problem constitutes a natural generalization of Weak ℓ-th Bilinear Diffie-Hellman Inversion one, which has been defined by Boneh, Boyen and Goh in [4]. Although the cited paper is devoted to some HIBE scheme, it is well known that there is a natural correspondence between HIBE schemes and signature schemes, see for instance [5], [8], [9]. When it comes to forward-security of the constructed scheme, it is provided by a security reduction carried out in the random oracle model.

Due to using a concept and some properties of binary trees, we have been able to create an updating mechanism in such a way that it has been possible to gain all the requirements described above. Namely, a fixed positive integer ℓ induces a binary tree of height ℓ and with 2^ℓ leafs. These leaves can be numbered from 0 to $2^\ell - 1$, besides, it is well known that for every leaf there is a unique path joining the root with this leaf. If we adopt a rule that for a given node choosing its left or right child means assigning the value 0 or 1, respectively, then this path can be viewed as a binary string of the length ℓ, which is also a binary representation of the index assigned to the leaf. Obviously, the same observation can be made for every node, where we identify the index of the node with the binary representation of the unique path between this node and the root. The binary representation itself is obtained after applying the introduced rule. Such an idea allows us to equate the leaves with successive time periods and use the rule for making paths to generate a secret key associated with a leaf related to a given time frame. Furthermore, as we have already stated, secret keys must carry data needed for both making current signatures and generating a key for the next period. In our scheme these tasks are split into two separated components, namely a signing one and a stack, consisting of at most ℓ nodes so that they enable the generation of keys only for future periods. It means that none of the elements of this stack may be used to obtain any

of the previous keys. It is possible because each node from the stack lies on the right-hand side of the path joining the root with the leaf referring to the current time period. Moreover, it must be stressed that the algorithm which is intended to fill in the stack has been designed so as to guarantee its minimal content. Minimality here means that if even one element of the stack has been removed, then it is not possible to generate at least one future key.

Although the concept of forward-secure signature schemes was proposed by Anderson [1] in 1997, the formalization of this notion was made by Bellare and Miner [2]. They provided tree-based and Fiat-Shamir [10] -based constructions that meet this definition. In [16] Krawczyk put forward simple transformation of a regular signature scheme into a forward-secure signature scheme, where the parameters of resultant signatures have size independent of the number of periods, but the signer's storage grows linearly with this number. Camenisch and Koprowski [7] use construction proposed by Itkis and Reyzin [14] to achieve a forward-secure signature scheme in the standard model under the Strong-RSA assumption. Following this line of thought, Hohenberger and Waters [13] put forward a new abstraction that was called RSA sequencer and enables achieving efficient forward-secure signatures.

Thanks to exploit asymmetric pairing along with some properties of binary trees we have been able to obtain a new provable forward-secure signature scheme. To the best of our knowledge, the approach that is proposed herein is new and differs from the approach presented by other authors. Nonetheless, we want to stress that this work is ideologically connected with [4] where Boneh, Boyen and Goh used symmetric pairing in construction of a certain HIBE system.

III. Preliminaries

A. Weak Bilinear Diffie-Hellman Inversion Type Assumption

Assume that $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ are three multiplicative cyclic groups of prime order p. Let us remind that if $\mathbb{G}_1 \neq \mathbb{G}_2$ and no efficiently computable isomorphism is known between $\mathbb{G}_1$ and $\mathbb{G}_2$, in either direction, then a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is called a pairing of Type 3 if it satisfies the following properties:

1) (*bilinearity*) i.e., for all $u \in \mathbb{G}_1, v \in \mathbb{G}_2$ and $a, b \in \mathbb{F}_p$ we have
$$\hat{e}\left(u^a, v^b\right) = \hat{e}\left(u, v\right)^{ab};$$
2) (*non-degeneracy*) i.e.:
   - if for all $u \in \mathbb{G}_1$ we have $\hat{e}(u, v) = 1_{\mathbb{G}_T}$ then it is equivalent to $v = 1_{\mathbb{G}_2}$;
   - if for all $v \in \mathbb{G}_2$ we have $\hat{e}(u, v) = 1_{\mathbb{G}_T}$ then it is equivalent to $u = 1_{\mathbb{G}_1}$.

Let g_1, g_2 be generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and let $\alpha, \beta \xleftarrow{\$} \mathbb{F}_p^*$. We introduce the following problem, called $(\ell, 1)$-wBDHI$_3^*$:

$$\text{given} \quad G_i(\alpha, \beta) := \{g_i, g_i^{\alpha}, \ldots, g_i^{(\alpha^{\ell})}, g_i^{\beta}\}, \ i = 1, 2,$$
$$\text{compute} \quad \hat{e}(g_1, g_2)^{\beta(\alpha^{\ell+1})}.$$

Suppose that $\mathsf{params} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, \hat{e}) \leftarrow \mathscr{G}(1^n)$. We say that $(\ell, 1)$-wBDHI$_3^*$ *problem is hard relative to* $\mathscr{G}$ if for all PPT adversaries $\mathcal{A}$, the following probability is negligible

$$\Pr\left[\begin{array}{c} \mathsf{params} \leftarrow \mathscr{G}(1^n); \\ \alpha, \beta \xleftarrow{\$} \mathbb{F}_p^* \end{array} : \begin{array}{r} \hat{e}(g_1, g_2)^{\beta(\alpha^{\ell+1})} \leftarrow \mathcal{A}\,(1^n, \\ \mathsf{params}, G_i(\alpha, \beta)) \end{array}\right]$$

This probability is called the *advantage* of the adversary $\mathcal{A}$ in solving $(\ell, 1)$-wBDHI$_3^*$ problem, and is denoted by $\mathbf{Adv}^{(\ell,1)\text{-wBDHI}_3^*}_{\mathsf{params},n}(\mathcal{A})$.

Note that $(\ell, 1)$-wBDHI$_3^*$ is a natural generalization of so-called the Weak ℓ-th Bilinear Diffie-Hellman Inversion problem, denoted by ℓ-wBDHI* and defined in [4] for pairings of Type 1. Indeed, remind that if $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is Type 1 pairing, $\mathbb{G}, \mathbb{G}_T$ are cyclic groups of prime order p and g is a random generator of $\mathbb{G}$ and $\alpha, \beta \xleftarrow{\$} \mathbb{F}_p^*$, then ℓ-wBDHI* is as follows (obviously, since p is a prime then $h = g^{\beta}$ is a generator too):

$$\text{given } g, g^{\alpha}, \ldots, g^{(\alpha^{\ell})}, h := g^{\beta} \text{ compute } \ e(g, g)^{\beta(\alpha^{\ell+1})}.$$

Substituting $a := g^{(\alpha^{\ell})}$ and $\gamma := \alpha^{-1}$, $\delta := \beta(\alpha^{\ell})^{-1}$, we see that $a^{(\gamma^i)} = g^{(\alpha^{\ell-i})}$, $i \in [\ell]$ and $a^{\delta} = g^{\beta}$. Thus, taking $(a, b := a^{\delta}, a^{\gamma}, a^{(\gamma^2)}, \ldots, a^{(\gamma^{\ell})})$ as an input to ℓ-wBDHI*, we immediately conclude that ℓ-wBDHI* is polynomially reducible to to the following problem known as ℓ-wBDHI and introduced in [4]:

$$\text{given } g, g^{\alpha}, \ldots, g^{(\alpha^{\ell})}, h := g^{\beta} \text{ compute } \ e(g, h)^{\frac{1}{\alpha}}.$$

In the same manner as above, we show the existence of a polynomial transformation, acting in the other direction. In consequence, both problems ℓ-wBDHI* and ℓ-wBDHI are equivalent under polynomial time reductions. Furthermore, it turns out that there is a strict connection between these problems and the commonly known ℓ-BDHI (see [3], [17], for instance). Namely, D. Boneh, X. Boyen and E.-J. Goh proved in [4] that an algorithm for ℓ-wBDHI or ℓ-wBDHI* in $\mathbb{G}$ gives an algorithm for ℓ-BDHI with a tight reduction.

To sum up, all this information provided above lead us to the conclusion it is highly likely that $(\ell, 1)$-wBDHI$_3^*$ is at least as hard as ℓ-BDHI. In fact, if there was a method making possible to break $(\ell, 1)$-wBDHI$_3^*$ then this one should be able to be applied to break easier case with Type 1 pairing. This means that ℓ-wBDHI* would be broken as well, what eventually would imply weakness of ℓ-BDHI.

B. Forward-Secure Signature Schemes

A signature scheme with evolving private key is composed of five algorithms $\Pi_{\mathsf{fu}} = (\mathscr{G}, \mathsf{KGen}, \ \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vrfy})$ along with an associated message space $\mathcal{M}$, such that:

- (*System parameters generation*) $\mathscr{G}$ is a PT algorithm, which takes as an input the value 1^n of a security parameter and maximum number of time periods T. It outputs the system parameters params.

- (*Key generation*) KGen is a PPT algorithm. It takes as input the system parameters params and maximum number of time periods T and outputs a public verification key pk along with an initial secret signing key sk_0 for the time period $t = 0$.
- (*Key update*) KUpd is a PPT algorithm. It takes as input the secret key sk_t for time period $t < T - 1$ and outputs the secret key sk_{t+1} for the next time period $t + 1$.
- (*Signing*) Sign is a PPT algorithm. It takes as input the current secret key sk_t and a message $m \in \mathcal{M}$ and outputs a signature σ.
- (*Verification algorithm*) Vrfy is a DPT algorithm. It takes as input a public key pk, a message $m \in \mathcal{M}$, the proper time period t and a (purported) signature σ. It outputs a single bit b, with $b = 1$ meaning *accept* and $b = 0$ meaning *reject*.

In addition, we assume the correctness, meaning that for all messages $m \in \mathcal{M}$ and for all time periods $t \in \{0, 1, \ldots T-1\}$ it holds that $\mathsf{Vrfy}_{\mathsf{pk}}(t, m, \mathsf{Sign}_{\mathsf{sk}_t}(m)) = 1$ with probability one if $(\mathsf{sk}_0, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{params}, T)$ and $\mathsf{sk}_{i+1} \leftarrow \mathsf{KUpd}(\mathsf{sk}_i)$ for $i = 0, \ldots t - 1$.

We explain below how a well known euf-cma security model may be expanded to signature schemes with evolving private key. This idea was taken from Bellare-Miner paper [2]. Let $\mathcal{A}$ be an adversary. Consider the following experiment $\mathbf{Exp}^{\mathsf{fu\text{-}cma}}_{\mathcal{A},\Pi_{\mathsf{fu}}}$, which depends on the system parameters and a number of periods. We assume that the system parameters have been generated and they are known to the adversary.

1) Generate $\mathsf{params} \leftarrow \mathcal{G}(1^n, T)$ and $(\mathsf{sk}_0, \mathsf{pk}) \leftarrow \mathsf{KGen}(\mathsf{params}, T)$.
2) The adversary $\mathcal{A}$ is given pk and access to three oracles, signing oracle Sign, key update oracle KUpd and break in oracle Break.
3) $t \leftarrow 0$.
4) **while** $t < T$

4.1. Sign : For current secret key sk_t the adversary $\mathcal{A}$ requests signatures on as many messages as it like (analogously to euf-cma it is denoted by $\mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}_t}(\cdot)}(\mathsf{pk})$).

4.2. KUpd : If the current time period $t < T - 1$ then $\mathcal{A}$ requests update: $t \leftarrow t + 1$, $\mathsf{sk}_{t+1} \leftarrow \mathsf{KUpd}(\mathsf{sk}_t)$.

4.3 If Break then break the loop **while**;
Break : If $\mathcal{A}$ is intended to go to the forge phase then it launches Break. Then the experiment records the break-in time $\bar{t} = t$ and sends the current signing key $\mathsf{sk}_{\bar{t}}$ to $\mathcal{A}$. This oracle can only be queried once, and after it has been queried, the adversary can make no further queries to the key update or signing oracles.

5) Eventually $(t^\star, m^\star, \sigma^\star) \leftarrow \mathcal{A}(1^n, \mathsf{state})$.
6) If $t^\star < \bar{t}$ and $\mathsf{Vrfy}_{\mathsf{pk}}(t^\star, m^\star, \sigma^\star) = 1$ and the signing oracle $\mathsf{Sign}_{\mathsf{sk}_{t^\star}}$ has been never queried about $m^\star$ within the time period $t^\star$, then output 1, otherwise output 0.

We refer to such an adversary as an fu-cma-adversary. The advantage of the adversary $\mathcal{A}$ in attacking the scheme Π_{fu} is defined as

$$\mathbf{Adv}^{\mathsf{fu\text{-}cma}}_{\Pi_{\mathsf{fu}},n}(\mathcal{A}) = \Pr[\mathbf{Exp}^{\mathsf{fu\text{-}cma}}_{\mathcal{A},\Pi_{\mathsf{fu}}}(1^n, T) = 1].$$

A signature scheme with evolving private key $\Pi_{\mathsf{fu}} = (\mathcal{G}, \mathsf{KGen}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vrfy})$, is called to be *existentially forward unforgeable under a chosen-message attack* or just *forward-secure* if for all efficient probabilistic, polynomial-time adversaries $\mathcal{A}$, there is a negligible function negl such that

$$\mathbf{Adv}^{\mathsf{fu\text{-}cma}}_{\Pi_{\mathsf{fu}},n}(\mathcal{A}) = \mathsf{negl}(n).$$

In a well known and widely cited paper [12], the authors provide a classification of security strength for signature schemes. Except commonly used notion of existential forgery they also indicate some weaker notions, like the second on the top list, namely a selective forgery where a signature must be forged for a particular message chosen *a priori* by the adversary. This idea was exploited by Canetti, Halevi, Katz [8] and Boneh, Boyen [3], who define security against selective forgery for IBE and HIBE. Therefore, it is not surprising that it can be also adopted for scheme with evolving private key regarding their forward security. Formally, we start with a description of a proper experiment, which will be referred to as $\mathbf{Exp}^{\mathsf{sfu\text{-}cma}}_{\mathcal{A},\Pi_{\mathsf{fu}}}$. This experiment differs from $\mathbf{Exp}^{\mathsf{fu\text{-}cma}}_{\mathcal{A},\Pi_{\mathsf{fu}}}$ in such a way that an adversary $\mathcal{A}$ outputs a message together with the associated time parameters $(\mathbf{m}^*, t^*, \bar{t})$ that are intended to be forged, before receiving the public key. Next, the experiment is conducted in the same manner as $\mathbf{Exp}^{\mathsf{fu\text{-}cma}}_{\mathcal{A},\Pi_{\mathsf{fu}}}$, with this difference that only if the time period $\bar{t}$ is reached then the oracle Break is launched. The adversary wins if it has been able to output a valid signature σ^* for $\mathbf{m}^*$ in the period t^*. This lead us to the following definition. A signature scheme with evolving private key $\Pi_{\mathsf{fu}} = (\mathcal{G}, \mathsf{KGen}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vrfy})$, is called to be *selectively forward-secure* if for all efficient probabilistic, polynomial-time adversaries $\mathcal{A}$, $\mathbf{Adv}^{\mathsf{fu\text{-}cma}}_{\Pi_{\mathsf{fu}},n}(\mathcal{A})$ is a negligible function of n.

IV. Construction of Forward-Secure Scheme

In this section we shall show the construction of our signature scheme with evolving private key $\Pi_{\mathsf{fu}} = (\mathcal{G}, \mathsf{KGen}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vrfy})$. We will justify that the scheme is forward secure in the random oracle model with only assumed hardness of $(\ell, 1)$-wBDHI$^*_3$ problem for pairings of Type 3.

Let $\mathcal{M} = \{0,1\}^*$ be the message space associated with the scheme and $H : \mathcal{M} \to \{0,1\}^{l \cdot \ell}$ be a collision resistant hash function. We must stress that the length of the hash values is not accidental, because output hashes are split into l blocks, where a single block is a ℓ-bit string.

Before going to the formal description, we briefly explain the idea of the scheme, which is based on the geometry and some properties of binary-trees. Let us adopt a rule that for a given node, choosing its left or right child means assigning the value 0 or 1, respectively (see Fig. 1). On the other hand, it is commonly known that for every node there is a unique path joining the root with this node (in particular including

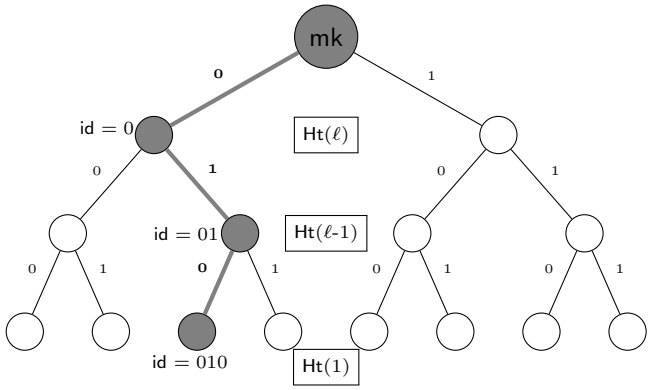

Fig. 1. Visualization of some basic ideas standing behind the scheme.

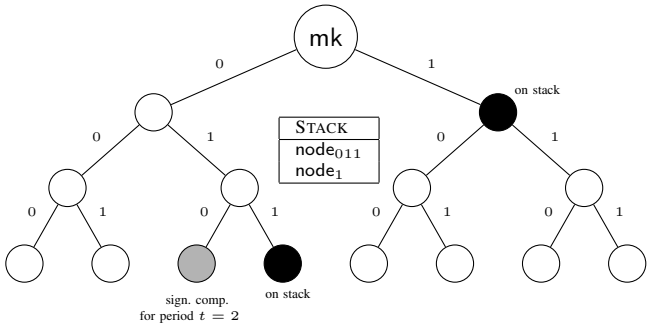

Fig. 2. Content of a secure key associated with a period t.

leaves). Therefore, when we apply the described rule, we see that this path is represented by the following bit-string $t_\ell t_{\ell-1}\cdots t_h$, $h \in [\ell]$, with h being the height of a node. Moreover, this bit-string provide the node with the unique identification, which thus may be viewed as an identifier of this node. As we mentioned in the introduction, the goal is to make a binary tree with a number of leaves being the same as the number of time periods i.e. 2^ℓ, therefore a high of this tree must be ℓ. Let us choose $u_{0,0}, u_1, \ldots, u_\ell$ from $\mathbb{G}$ and define $\mathsf{Ht}(h) := u_{0,0}\prod_{i=h}^{\ell} u_i$, for $h \in [\ell]$. Then, it is obvious that every node indexed by $\mathsf{id} = t_\ell \cdots t_h$ can be uniquely connected with $\mathsf{Ht}(h)^{\mathsf{id}} = u_{0,0}\prod_{i=h}^{\ell} u_i^{t_i}$ (see Fig. 1). Furthermore, note that if P is the path linking the root with a leaf $t_\ell \cdots t_1$ and $\mathsf{id}_1 = t_\ell \cdots t_h$, $\mathsf{id}_2 = t_\ell \cdots t_{h-1} \in P$ are indexes of two successive nodes, then there is the following relation between $\mathsf{Ht}(h)^{\mathsf{id}_1}$ and $\mathsf{Ht}(h)^{\mathsf{id}_2}$, namely $\mathsf{Ht}(h-1)^{\mathsf{id}_2} = \mathsf{Ht}(h)^{\mathsf{id}_1} \cdot u_{h-1}^{t_{h-1}}$.

Next, suppose that the root is related to mk, which should be viewed as a master-key. This key controls the entire scheme, which means that knowing mk we are able to generate a valid secret key for each period. Bear in mind that what we are trying to gain is forward-security, which is why revealing a secret key assigned to a period t must not leak anything about secret keys of prior periods. This implies, in particular, that master-key ought not to be recovered from any period's secret key or, even worse, can not be a plain part of these keys, keeping simultaneously in minds that all these keys must strictly depend on the master key. In addition, all the knowledge regarding prior secret keys has to be lost. These are the reasons why we talk about „memory loss" by the key updater. To meet these requirements, we encapsulate mk by randomization Hts, according to the following iterative method. At first, we pick r_ℓ uniformly at random from $\mathbb{F}_p$, and compute $\mathsf{mk}\cdot(\mathsf{Ht}(\ell)^{t_\ell})^{r_\ell}$; note that to keep the randomness under control we must save $u_{\ell-1}^r, \ldots, u_1^r$. We also have to keep an additional element g_2^r, which is required for verification. Taking all of these into account we obtain $\mathsf{node}_{t_\ell} = \left(\mathsf{mk}\cdot(\mathsf{Ht}(\ell)^{t_\ell})^{r_\ell}; u_{\ell-1}^{r_\ell}, \ldots, u_1^{r_\ell}; g_2^{r_\ell}\right)$. Next, if $\mathsf{node}_{t_\ell\cdots t_h} = \left(\mathsf{mk}\cdot(\mathsf{Ht}(h)^{t_\ell\cdots t_h})^{r_h}; u_{h-1}^{r_h}, \ldots, u_1^{r_h}; g_2^{r_h}\right)$ is a node at height h and $t_\ell\cdots t_{h-1}$ is the index of a successive node of height $h-1$, lying on a same path which links the root with a leaf, then to get $\mathsf{node}_{t_\ell\cdots t_{h-1}}$, we compute $\mathsf{mk}\cdot(\mathsf{Ht}(h)^{t_\ell\cdots t_h})^{r_h}\cdot(u_{h-1}^{r_h})^{t_{h-1}} = \mathsf{mk}\cdot(\mathsf{Ht}(h-1)^{t_\ell\cdots t_{h-1}})^{r_h}$, next we choose r'_{h-1} uniformly at random from $\mathbb{F}_p$ and calculate $(\mathsf{Ht}(h-1)^{t_\ell\cdots t_{h-1}})^{r'_{h-1}}$; having this, we do $\mathsf{mk}\cdot(\mathsf{Ht}(h-1)^{t_\ell\cdots t_{h-1}})^{r_h}\cdot(\mathsf{Ht}(h-1)^{t_\ell\cdots t_{h-1}})^{r'_{h-1}} = \mathsf{mk}\cdot(\mathsf{Ht}(h-1)^{t_\ell\cdots t_{h-1}})^{r_{h-1}}$, with $r_{h-1} = r_h + r'_{h-1}$; in the same way we obtain $u_i^{r_{h-1}} = u_i^{r_h}\cdot u_i^{r'_{h-1}}$, $i \in [h\text{-}2]$, and $g_2^{r_{h-1}} = g_2^{r_h}\cdot g_2^{r'_{h-1}}$; eventually $\mathsf{node}_{t_\ell\cdots t_{h-1}} = \left(\mathsf{mk}\cdot(\mathsf{Ht}(h-1)^{t_\ell\cdots t_{h-1}})^{r_{h-1}}; u_{h-2}^{r_{h-1}}, \ldots, u_1^{r_{h-1}}; g_2^{r_{h-1}}\right)$. Following this method, we get in the end to the last node on the path, namely $\mathsf{leaf}_{t_\ell\cdots t_1} = \left(\mathsf{mk}\cdot(\mathsf{Ht}(1)^{t_\ell\cdots t_1})^{r_1};\ g_2^{r_1}\right)$. It must be emphasized that even though $r_h = r_h(r_{h+1}, \ldots, r_\ell)$, all of $r_h, r_{h-1}, \ldots, r_\ell$ are equally likely. It is a consequence of an easy and well-known fact, namely if we take a probability measure $\mu(A) := \#A/p$ defined on the σ-field $2^{\mathbb{F}_p}$, then for every $A \in 2^{\mathbb{F}_p}$ and fixed $\gamma \in \mathbb{F}_p$ we have $\mu(A) = \mu(\gamma + A)$ (see [15], for instance).

These considerations lead us to the conclusion that having a node $\mathsf{node}_{\mathsf{id}}$ at height h it is easy to determine nodes at lower heights, that lie on root-to-leaves paths and pass through $\mathsf{node}_{\mathsf{id}}$. On the other hand, if DLP is hard in $\mathbb{G}$, then it is hard to figure out a form of nodes at higher heights. Furthermore, all these nodes encapsulate „master-key" mk. Finally, it is seen that to guarantee the „memory loss" property, secret keys associated with periods must consists both components needed for making signatures and nodes being roots of maximal subtrees that let to compute leaves associated with future periods. The latter is made by the function $\mathsf{StackFilling}$, defined through Algorithm 1 (see Section IV-B). More precisely, nodes required for generating secret keys for future periods are gathered on a stack, which any time consists of at most ℓ elements. As we indicated above, the content of the stack is optimal, meaning that if at least one element of the stack has been removed, then it would not be possible to generate at least one future key. Algorithm 1 is depicted in Fig. 2.

A. System parameters generation

Let n be a security parameter. An efficient and polynomial time system parameters generator $\mathscr{G}$ takes on input both a value of the security parameter 1^n and a maximum number of time periods T, to then output $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, \hat{e})$, where:

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are three cyclic groups of prime order p, where group operations can be performed efficiently and

no efficiently computable isomorphism is known between $\mathbb{G}_1$ and $\mathbb{G}_2$, in either direction .
- $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ are chosen uniformly at random from the set of all generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively.
- $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable Type 3 pairing.

B. Key generation

Before we go to the description of the initial keys generation process, we will present the other algorithm, namely StackFilling. This algorithm plays the crucial role in our construction, because it is directly responsible for „memory loss" of our keys updater, meaning in particular that it allows us to gain the desired security requirements.

For the formal reasons, let us define

$$\prod_{i=i_1}^{i_2} u_i^{\alpha_i} = \begin{cases} u_{i_1}^{\alpha_{i_1}} \cdots u_{i_2}^{\alpha_{i_2}} & \text{for} \quad i_1 \le i_2 \\ 1_{\mathbb{G}_1} & \text{for} \quad i_1 > i_2 \end{cases} .$$

Assume there is a stack STACK, that will be filled with pairs (node, h), where node and h are a node and its height on the binary tree, respectively. We stress that height h varies from 1 to $\ell+1$, where $\ell+1$ is assigned to the root. Besides, it must be kept in mind that a node, being on a height of h have the specific form, namely

$$\begin{aligned} \mathsf{node} &:= \left(\tau_1^x \cdot \left(u_{0,0} \prod_{i=h}^{\ell} u_i^{t_i}\right)^r ; u_{h-1}^r, \ldots, u_1^r; g_2^r\right) \\ &= \left(A; (b_i)_{i\in[h-1]}; C\right), \end{aligned} \tag{1}$$

where r is an element of $\mathbb{F}_p$, meaning that each element of $\mathbb{F}_p$ is equally likely.

Algorithm 1 Function StackFilling

Input: params, STACK, $t = t_\ell \cdots t_1$
Output: STACK, scp

1: $(\mathsf{node}, h) \leftarrow$ STACK.pop() ▷ $\mathsf{node} = \left(A; (b_i)_{i\in[h-1]}; C\right)$
2: $h \leftarrow h-1$
3: ▷ After reindexing, $\mathsf{node} = \left(A; (b_i)_{i\in[h]}; C\right)$
4: **while** $h > 0$ **do**
5: $r \xleftarrow{\$} \mathbb{F}_p$
6: $\mathsf{tmp} \leftarrow \left(A \cdot b_h^1 \cdot \left(\left(u_{0,0} \prod_{i=h+1}^{\ell} u_i^{t_i}\right) \cdot u_h^1\right)^r ; b_{h-1} \cdot u_{h-1}^r, \ldots, b_1 \cdot u_1^r; C \cdot g_2^r\right)$
7: ▷ $A := \mathsf{node}.A$, $b_i := \mathsf{node}.b_i$, $C := \mathsf{node}.C$, i.e. $\mathsf{tmp} = \mathsf{tmp}(\mathsf{node})$.
8: STACK.push $((\mathsf{tmp}, h))$
9: $r \xleftarrow{\$} \mathbb{F}_p$ ▷ A new randomness, i.e. independent of r picked in 5
10: $\mathsf{node} \leftarrow \left(A \cdot \left(u_{0,0} \prod_{i=h+1}^{\ell} u_i^{t_i}\right)^r ; b_{h-1} \cdot u_{h-1}^r, \ldots, b_1 \cdot u_1^r; C \cdot g_2^r\right)$
11: $h \leftarrow h-1$
12: **end while**
13: $\mathsf{scp} \leftarrow \mathsf{node}$

Now we are ready to describe the algorithm KGen, generating both an initial private key sk_0 and a long term public key

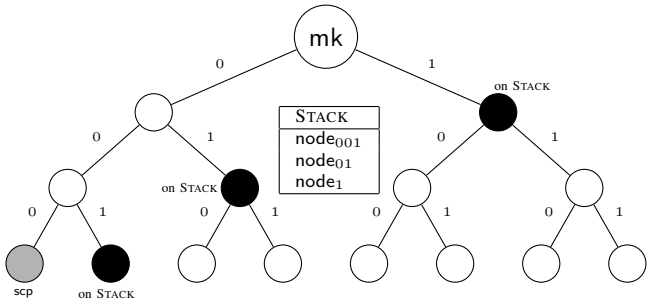

Fig. 3. The initial secret key $\mathsf{sk}_0 = (\mathsf{scp}, \text{STACK}, t = 0)$ and output from StackFilling.

pk. It depends on two variables, namely system parameters, which has been generated by $\mathscr{G}$, and a maximal number of time periods T. Let us suppose that, we have been taken $\mathsf{params} := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, \hat{e}) \leftarrow \mathscr{G}(1^n)$. The formal definition of the algorithm is as follow:

1) Choose $\tau_1 \xleftarrow{\$} \{g \in \mathbb{G}_1 \mid \langle g \rangle = \mathbb{G}_1\}/\{g_1\}$.
2) Pick $x \xleftarrow{\$} \mathbb{F}_p^*$ and set $\tau_2 := g_2^x \in \mathbb{G}_2$, which is the crucial component of pk.
3) Choose $(u_{0,i})_{i=0}^{l}, u_1, \ldots, u_\ell \xleftarrow{\$} \mathbb{G}_1$.
4) Pick $r \xleftarrow{\$} \mathbb{F}_p$ and compute $\mathsf{node} \leftarrow \left(\tau_1^x u_{0,0}^r;\ u_\ell^r, u_{\ell-1}^r, \ldots, u_1^r;\ g_2^r\right) =: \left(A; (b_i)_{i\in[\ell]}; C\right)$.
5) Initialize an empty stack STACK, next do STACK.push $((\mathsf{node}, \ell+1))$.
6) Run $(\mathsf{scp}, \text{STACK}) \leftarrow$ StackFilling $(\mathsf{params}, \text{STACK}, t = 0)$. After the function StackFilling has output the value, STACK consists of ℓ elements.
7) The initial secret key is $\mathsf{sk}_0 = (\mathsf{scp}, \text{STACK}, t = 0)$ and the public key is $\mathsf{pk} = (\tau_1, \tau_2, (u_{0,i})_{i=0}^{l}, u_1, \ldots, u_\ell)$.

Note that the signing component scp of sk_0 has the following form

$$\mathsf{scp} = \left(\tau_1^x u_{0,0}^r;\ g_2^r\right) = (A; C). \tag{2}$$

It is important to highlight that r in the above formula only express randomness and it is highly likely to be different from the other randomness r, appearing in 4. It can be confusing at the first glimpse, so we focus on this for a while. Taking into account the method of generating scp, we see from 4) and Algorithm 1 that there are $(\ell+1)$ random elements of $\mathbb{F}_p$, required to generate r appearing in (2). To be more accurate $r = r(r_0, r_1, \ldots, r_\ell)$, where for instance r_0 is the same as r appearing in 4). In fact, the relation between all these r's is linear and $r = r_0 + \ldots + r_\ell$.

C. Key update

The algorithm KUpd takes as an input the secret key, assigned to the period $t < T-1$ and updates this key for the next period $t+1$. Below we describe the successive steps of this algorithm.

1) Parse $\mathsf{sk}_t = (\mathsf{scp}, \text{STACK}, t)$. Obviously it is the current secret key, which is dedicated to the period t, and is going to be updated.

2) Update a variable carrying a time period, namely conduct $t \leftarrow t+1$. After this step the variable t stores a value of the new time period.
3) If $t \equiv 1 \pmod 2$, then the following steps are carried out.
   3.1 $(\mathsf{node}, h) \leftarrow \textsc{Stack}.\mathrm{pop}()$ and $\mathsf{scp} \leftarrow \mathsf{node}$. It is easily seen, that here the highest element is popped from the stack and passed to the signing component scp.
   3.2 The secret key for the new period has the form $\mathsf{sk}_t = (\mathsf{scp}, \textsc{Stack}, t)$.
4) If $t \equiv 0 \pmod 2$, then the following steps are conducted.
   4.1 Run $(\mathsf{scp}, \textsc{Stack}) \leftarrow \mathsf{StackFilling}(\mathsf{params}, \textsc{Stack}, t)$.
   4.2 The secret key for the new period has the form $\mathsf{sk}_t = (\mathsf{scp}, \textsc{Stack}, t)$.

This time the signing component scp of sk_t has the form

$$\mathsf{scp} = \left(\tau_1^x \cdot \left(u_{0,0}\prod_{i=1}^{\ell} u_i^{t_i}\right)^r \;;\; g_2^r\right) = (A; C). \tag{3}$$

We strongly recommend keeping in mind the remarks regarding randomness, that are pointed out after (2).

D. Signing

Here we explain how the procedure of making signature looks like. It is obvious that Sign depends on the secret key sk_t associated with a period $t < T$, meaning that for a fixed t, the signatures are made by Sign_{sk_t}, taking as an argument a message that is going to be signed, and outputting a value of the signature. As we have seen above each secret key sk_t consists of two components, namely a signing component scp and a stack $\textsc{Stack}$. Both play an important role in the scheme simultaneously, having completely different tasks. The latter is of crucial importance with regard to updating a key to the next period and is not used in the signing process, while the signing component is not needed in updating a key but it is essential in making a desired signature.

Below we describe the consecutive steps of computing a signature on a given message $\mathbf{m}$ from the message space $\mathcal{M}$.

1) Compute the hash value $m = H(\mathbf{m})$.
2) Parse $\mathsf{sk}_t = (\mathsf{scp}, \textsc{Stack})$ and next parse $\mathsf{scp} = (A; C)$.
3) Write t in the binary form $t = (t_\ell \cdots t_1)_2$. Split m into concatenation of l blocks $m_1 \| \cdots \| m_l$ and write each block m_i in the binary form $(m_{i,\ell} \cdots m_{i,1})_2$.
4) Pick $r \xleftarrow{\$} \mathbb{F}_p$ and $s_i \xleftarrow{\$} \mathbb{F}_p$, $i \in [l]$, independently and uniformly at random.
5) Let us compute

$$\sigma_1 \leftarrow A \cdot \left(u_{0,0}\cdot\prod_{i=1}^{\ell} u_i^{t_i}\right)^r \cdot \prod_{j=1}^{l}\left(u_{0,j}\cdot\prod_{i=1}^{\ell} u_i^{m_{j,i}}\right)^{s_j}. \tag{4}$$

$$\begin{aligned}\sigma_2 &\leftarrow C \cdot g_2^r \\ \sigma_{3,j} &\leftarrow g_2^{s_j},\ j = 1, \ldots, l.\end{aligned} \tag{5}$$

Output a signature $\sigma = \left(\sigma_1, \sigma_2, (\sigma_{3,j})_{j\in[l]}\right)$.

There is an interesting observation to make. Namely, let the hash m be presented as the following binary matrix

$$M := \begin{bmatrix} m_{1,\ell} & m_{1,\ell-1} & \cdots & m_{1,1} \\ m_{2,\ell} & m_{2,\ell-1} & \cdots & m_{2,1} \\ \vdots & \vdots & \ddots & \vdots \\ m_{l,\ell} & m_{l,\ell-1} & \cdots & m_{l,1} \end{bmatrix}$$

The i-th row and the i-th column of M are denoted by $M(i,\cdot) = m_i$ and $M(\cdot, j)$, respectively. Furthermore, the former can be treated as a vector in $\mathbb{F}_p^\ell$, whilst the latter as a vector in $\mathbb{F}_p^l$. Let us put $\mathbf{s} = [s_1, \ldots, s_l] \in \mathbb{F}_p^l$ and compute the following inner products in $\mathbb{F}_p^l$

$$\bar{s^i} := \langle \mathbf{s}, M(\cdot, i)\rangle = \sum_{j=1}^{l} m_{j,i}s_j, \quad \text{for } i = 1, \ldots, \ell. \tag{6}$$

Then σ_1 can be written in the form

$$\sigma_1 \leftarrow A \cdot \left(u_{0,0}\cdot\prod_{i=1}^{\ell} u_i^{t_i}\right)^r \cdot \prod_{j=1}^{l} u_{0,j}^{s_j} \cdot \prod_{i=1}^{\ell} u_i^{\bar{s^i}}$$

E. Verification

1) Compute $m = H(\mathbf{m})$ and parse pk as $(\tau_1, \tau_2, (u_{0,j})_{j=0}^{l}, u_1, \ldots, u_\ell)$.
2) Write t in the binary form $t = (t_\ell \cdots t_1)_2$ and split m into l blocks of ℓ-bits each, as described above, i.e. $m = m_1 \| m_2 \| \cdots \| m_l$, where $m_i = (m_{i,\ell} \cdots m_{i,1})_2$.
3) Output 1 if and only if the following condition holds

$$\begin{aligned}\hat{e}(\sigma_1, g_2) \stackrel{?}{=}\ & \hat{e}(\tau_1, \tau_2)\cdot\hat{e}\left(u_{0,0}\cdot\prod_{i=1}^{\ell} u_i^{t_i}, \sigma_2\right) \\ & \cdot\prod_{j=1}^{l}\hat{e}\left(u_{0,j}\cdot\prod_{i=1}^{\ell} u_i^{m_{j,i}}, \sigma_{3,j}\right).\end{aligned} \tag{7}$$

Otherwise, output 0.

To justify the correctness of the verification algorithm, we use (4). Due to the fact that both the time part and m_j-parts of σ_1 keep the same randomness as σ_2 and $\sigma_{s,j}$, respectively, we have that

$$\begin{aligned}&\hat{e}(\tau_1, \tau_2)\cdot\hat{e}\left(u_{0,0}\cdot\prod_{i=1}^{\ell} u_i^{t_i}, \sigma_2\right)\cdot\prod_{j=1}^{l}\hat{e}\left(u_{0,j}\cdot\prod_{i=1}^{\ell} u_i^{m_{j,i}}, \sigma_{3,j}\right) \\ &\quad = \hat{e}(\tau_1^x, g_2)\cdot\hat{e}\left(\left(u_{0,0}\cdot\prod_{i=1}^{\ell} u_i^{t_i}\right)^r, g_2\right) \\ &\qquad \cdot\hat{e}\left(\prod_{j=1}^{l}\left(u_{0,j}\cdot\prod_{i=1}^{\ell} u_i^{m_{j,i}}\right)^{s_j}, g_2\right) \\ &\quad = \hat{e}\left(\tau_1^x\cdot\left(u_{0,0}\cdot\prod_{i=1}^{\ell} u_i^{t_i}\right)^r\cdot\prod_{j=1}^{l}\left(u_{0,j}\cdot\prod_{i=1}^{\ell} u_i^{m_{j,i}}\right)^{s_j}, g_2\right) \\ &\quad = \hat{e}(\sigma_1, g_2).\end{aligned}$$

This yields (7).

V. Security of the Scheme

The goal of this section is to show that the presented scheme is forward-secure. The proof itself is conducted in the random oracle model, and is split into two parts. Namely, we firstly deal with the special case, when the hash function is the identity on $\{0,1\}^{l\ell}$, and we prove that the scheme is selectively forward-secure then. Having done this, we are able to consider the general case, guessing both the proper time period and the query to the random oracle, what finally provides us with the desired full fu-cma security of the scheme.

A. Selective forward security of the scheme

Let $\Pi_{\mathsf{sfu}} = (\mathscr{G}, \mathsf{KGen}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vrfy})$ *be the scheme defined above for* $T = 2^\ell$ *time periods and with the associated message space* $\mathcal{M} = \{0,1\}^{l\ell}$. *In addition, let us assume that* $H = id_{\mathcal{M}}$. *If there exists an* sfu-cma *adversary* $\mathcal{A}$ *against* Π_{sfu} *that has advantage* $\mathbf{Adv}^{\mathsf{sfu\text{-}cma}}_{\Pi_{\mathsf{sfu}},n}(\mathcal{A})$, *then there exists an algorithm* $\mathcal{B}$ *to solve* $(\ell,1)$*-wBDHI*$^*_3$ *problem with advantage*

$$\mathbf{Adv}^{(\ell,1)\text{-}wBDHI^*_3}_{\mathsf{params},n}(\mathcal{B}) \geq \frac{p-1}{p} \cdot \mathbf{Adv}^{\mathsf{sfu\text{-}cma}}_{\Pi_{\mathsf{sfu}},n}(\mathcal{A}), \tag{8}$$

and running time $\mathcal{O}(time(\mathcal{A}))$

In order to justify this assertion, we reduce the selective forward security of Π_{sfu} to the hardness of $(\ell,1)$-wBDHI$^*_3$ relative to $\mathscr{G}$.

We start with introducing an algorithm, which is connected with Algorithm 1 and outputs indexes of nodes stored in STACK for a time frame $t < T$.

Algorithm 2 Function StackFillingID

Input: STACKID, $t = (t_\ell \cdots t_1)_2$
Output: STACKID,

```
(nodeID, h) ← STACKID.pop()        ▷ nodeID = t_ℓ ··· t_h
h ← h − 1                          ▷ After reindexing, nodeID = t_ℓ ··· t_{h+1}
while h > 0 do
    tmp ← t_ℓ ··· t_{h+1} 1        ▷ i.e. t_h = 1
    STACKID.push((tmp, h))
end while
```

Suppose that $\mathcal{A}$ is an adversary which attacks the scheme. Without loss of generality, we can assume that for every time period the adversary $\mathcal{A}$ makes q queries to the signing oracle. Having this, we construct an algorithm $\mathcal{B}$ which solves the $(\ell,1)$-wBDHI$^*_3$ problem.

Algorithm $\mathcal{B}$

The algorithm is given $\mathsf{params} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, \hat{e})$ and
$G_i(\alpha,\beta) = \{g_i, g_i^\alpha, \ldots, g_i^{(\alpha^\ell)}, g_i^\beta\}$, $i = 1,2$.

1) The parameters params are sent to $\mathcal{A}$, which chooses and outputs $(m^*, t^*, \bar{t}) \in \{0,1\}^{(l\ell,\ell,\ell)}$ with $t^* < \bar{t}$; i.e $(m^*, t^*, \bar{t}) \leftarrow \mathcal{A}(\mathsf{params})$.
2) Uniformly at random select $y, y_{0,j}, y_i \xleftarrow{\$} \mathbb{F}_p$, $j = 0, \ldots, l$, $i = 1, \ldots \ell$ and put

$$x := \alpha, \quad \tau_2 \leftarrow g_2^\alpha,$$
$$\tau_1 \leftarrow g_1^y \cdot g_1^{(\alpha^\ell)}, \tag{9}$$
$$u_{0,0} \leftarrow g_1^{y_{0,0}} \cdot \prod_{i=1}^{\ell} \left(g_1^{(\alpha^i)}\right)^{-t^*_i}, \tag{10}$$
$$u_{0,j} \leftarrow g_1^{y_{0,j}} \cdot \prod_{i=1}^{\ell} \left(g_1^{(\alpha^i)}\right)^{-m^*_{j,i}}, \text{ for } j = 1, \ldots l, \tag{11}$$
$$u_i \leftarrow g_1^{y_i} \cdot g_1^{(\alpha^i)}, \text{ for } i = 1, \ldots \ell. \tag{12}$$

For $t < T$ and $m = m_1 \| m_2 \| \cdots \| m_l$, define the functions

$$Y_0(t) = y_{0,0} + \sum_{i=1}^{\ell} y_i t_i,$$
$$Y_j(m_j) = y_{0,j} + \sum_{i=1}^{\ell} y_i m_{j,i}.$$

3) If $Y_0(t^*) = 0$ in $\mathbb{F}_p$ then abort.
4) Otherwise, set $t \leftarrow 0$ and send $\mathsf{pk} = (\tau_1, \tau_2, (u_{0,j})_{j=0}^l, u_1, \ldots u_\ell)$ to $\mathcal{A}$. Assign nodeID $\leftarrow$ null, initialize STACKID.push $((\mathsf{nodeID}, \ell+1))$ and launch STACKID $\leftarrow$ StackFillingID $(\text{STACKID}, t)$ (see Algorithm 2).
5) If $\mathcal{A}$ requests to update the current key and $t < T-1$, then do $t \leftarrow t+1$ and next if $t \equiv 1 \pmod 2$ then do STACKID.pop(), else do STACKID $\leftarrow$ StackFillingID $(\text{STACKID}, t)$. Otherwise output $\perp$.
6) When $\mathcal{A}$ requests a signature on a message $m \neq m^*$, then do:

6.1. If $t \neq t^*$ then pick $\xi, \eta_1, \ldots, \eta_l \xleftarrow{\$} \mathbb{F}_p$, compute $i_0 \leftarrow \max\{i \in [\ell] \mid t_i \neq t^*_i\}$ and provide $\mathcal{A}$ with a desired signature $\sigma = (\sigma_1, \sigma_2, (\sigma_{3,j})_{j\in[l]})$, where

$$\sigma_1 = \left(g_1^{(\alpha)}\right)^y \cdot g_1^{\xi Y_0(t)} \cdot \left(\prod_{i=1}^{i_0} \left(g_1^{(\alpha^i)}\right)^{(t_i - t^*_i)}\right)^{\xi} \cdot \left(g_1^{(\alpha^{\ell+1-i_0})}\right)^{-\frac{Y_0(t)}{t_{i_0} - t^*_{i_0}}} \cdot \prod_{i=1}^{i_0-1} \left(g_1^{(\alpha^{\ell+1+i-i_0})}\right)^{-\frac{t_i - t^*_i}{t_{i_0} - t^*_{i_0}}} \cdot \prod_{j=1}^{l} \left(g_1^{Y_j(m_j)} \cdot \prod_{i=1}^{\ell} \left(g_1^{(\alpha^i)}\right)^{(m_{j,i} - m^*_{j,i})}\right)^{\eta_j},$$
$$\sigma_2 = g_2^{\xi - (t_{i_0} - t^*_{i_0})^{-1} \cdot (\alpha^{\ell+1-i_0})},$$
$$\sigma_{3,j} = g_2^{\eta_j}.$$

6.2. If $t = t^*$, then choose $\xi, \eta_1, \ldots, \eta_l \xleftarrow{\$} \mathbb{F}_p$ and return a required signature $\sigma = (\sigma_1, \sigma_2, (\sigma_{3,j})_{j\in[l]})$ to $\mathcal{A}$, where

$$\sigma_1 = (g_1^{\alpha})^{y} \cdot g_1^{\xi Y_0(t^*)} \cdot \prod_{j=1}^{l} \left(g_1^{Y_j(m_j)} \cdot \prod_{i=1}^{\ell} \left(g_1^{(\alpha^i)} \right)^{(m_{j,i}-m^*_{j,i})} \right)^{\eta_j},$$
$$\sigma_2 = g_2^{\xi - Y_0(t^*)^{-1}\cdot(\alpha^{\ell+1})},$$
$$\sigma_{3,j} = g_2^{\eta_j}.$$

7) When the break-in time $\bar{t}$ is reached then the following steps are carried out.

7.1 Set $i_0 = \max\{i \in [\ell]\} \mid \bar{t}_i \neq t^*_i\}$ and choose $\xi \xleftarrow{\$} \mathbb{F}_p$ uniformly at random. Compute $\mathsf{scp} = (A; C)$, where

$$A = (g_1^{\alpha})^{y} \cdot g_1^{\xi Y_0(\bar{t})} \cdot \left(\prod_{i=1}^{i_0} \left(g_1^{(\alpha^i)} \right)^{(\bar{t}_i - t^*_i)} \right)^{\xi} \cdot \left(g_1^{(\alpha^{\ell+1-i_0})} \right)^{-(\bar{t}_{i_0} - t^*_{i_0})^{-1} Y_0(\bar{t})} \cdot \prod_{i=1}^{i_0-1} \left(g_1^{(\alpha^{\ell+1+i-i_0})} \right)^{-(\bar{t}_{i_0} - t^*_{i_0})^{-1}\cdot(\bar{t}_i - t^*_i)};$$
$$C = g_2^{\xi} \cdot \left(g_2^{(\alpha^{\ell+1-i_0})} \right)^{-(\bar{t}_{i_0} - t^*_{i_0})^{-1}}.$$

7.3 Let $L \leftarrow$ STACKID.len() and initialize two empty stacks $\overline{\text{STACK}}$, STACK.

7.4 **while** $L > 0$ **do**

7.5 $((t_\ell, \ldots, t_h), h) \leftarrow$ STACKID.pop() .

7.6 Set $i_0 = \max\{i \in \{h, \ldots, \ell\} \mid t_i \neq t^*_i\}$ and pick $\xi \xleftarrow{\$} \mathbb{F}_p$. Compute $\mathsf{node} = (A'; b'_{h-1}, \ldots, b'_1; C')$, where

$$A' = (g_1^{\alpha})^{y} \cdot g_1^{\xi\left(y_{0,0} + \sum_{i=h}^{\ell} y_i t_i\right)} \cdot \prod_{i=1}^{h-1} \left(g_1^{(\alpha^i)} \right)^{-\xi t^*_i} \cdot \prod_{i=h}^{i_0-1} \left(g_1^{(\alpha^i)} \right)^{\xi(t_i - t^*_i)} \cdot g_1^{-\left(y_{0,0} + \sum_{i=h}^{\ell} y_i t_i\right)\cdot \frac{\alpha^{(\ell+1-i_0)}}{t_{i_0} - t^*_{i_0}}} \cdot \prod_{i=h}^{i_0-1} \left(g_1^{(\alpha^{\ell+1+i-i_0})} \right)^{-\frac{t_i - t^*_i}{t_{i_0} - t^*_{i_0}}};$$
$$b'_i = g_1^{\xi y_i} \cdot \left(g_1^{(\alpha^i)} \right)^{\xi} \cdot \left(g_1^{(\alpha^{\ell+1-i_0})} \right)^{-\frac{y_i}{t_{i_0} - t^*_{i_0}}} \cdot \left(g_1^{(\alpha^{\ell+1-i_0+i})} \right)^{-\frac{1}{t_{i_0} - t^*_{i_0}}};$$
$$C' = g_2^{\xi - (t_{i_0} - t^*_{i_0})^{-1}\cdot(\alpha^{\ell+1-i_0})}.$$

7.7 Push node onto the top of the stack $\overline{\text{STACK}}$, i.e. $\overline{\text{STACK}}$.pop(node, h).

7.8 Decrement $L \leftarrow L - 1$.

7.9 **end while**

7.10 Put $L \leftarrow \overline{\text{STACK}}$.len()

7.11 **while** $L > 0$ **do**

7.12 tmp $\leftarrow \overline{\text{STACK}}$.pop() and next STACK.push (tmp).

7.13 Decrement $L \leftarrow L - 1$

7.14 **end while**

8) Sent $\mathsf{sk}_{\bar{t}} = (\mathsf{scp}, \text{STACK})$ to $\mathcal{A}$.

9) Eventually, $\mathcal{A}$ outputs $\sigma^* = (\sigma^*_1, \sigma^*_2, (\sigma_{3,j})^*_{j\in[l]})$. If it is a valid forgery of m^* in the time period t^*, output

$$\hat{e}\left(\sigma^*_1, g_2^{\beta}\right) \cdot \left(\hat{e}\left((g_1^{\alpha})^{y}, g_2^{\beta} \right) \cdot \hat{e}\left(g_1^{\beta}, \sigma^*_2 \right)^{Y_0(t^*)} \cdot \prod_{j=1}^{l} \hat{e}\left(g_1^{\beta}, \sigma^*_{3,j} \right)^{Y_j(m^*_{j,i})} \right)^{-1}.$$

At first, note that the signatures given to $\mathcal{A}$ are correctly distributed. Indeed, according to (5) and (4) it is seen that the real signature of $m \in \{0,1\}^{l\ell}$ for a given time period $t = (t_\ell \cdots t_1)_2$ has the following form

$$\sigma_1 = \tau_1^{x} \cdot \left(u_{0,0} \cdot \prod_{i=1}^{\ell} u_i^{t_i} \right)^{r} \cdot \prod_{j=1}^{l} \left(u_{0,j} \cdot \prod_{i=1}^{\ell} u_i^{m_{j,i}} \right)^{s_j},$$
$$\sigma_2 = g_2^{r}, \quad \sigma_{3,j} = g_2^{s_j}, \ j = 1, \ldots, l,$$

where r and s_j are selected uniformly at random from $\mathbb{F}_p$. Therefore, choosing ξ and η_j uniformly at random from $\mathbb{F}_p$, next setting $r = \xi - \lambda$ and $s_j = \eta_j$, where λ is an element of $\mathbb{F}_p$, we see that ξ, η_j and r, s_j are equally likely, respectively.

Assume that $t \neq t^*$. Obviously in this case there exists at least one $i \in [\ell]$ such that $t_i \neq t^*_i$. Let $i_0 := \max\{i \in [\ell] \mid t_i \neq t^*_i\}$, then $t_i = t^*_i$ for $i > i_0$ if such exist, meaning that if $i_0 \neq \ell$. Choosing $\xi \xleftarrow{\$} \mathbb{F}_p$ uniformly at random and putting $r = \xi - (t_{i_0} - t_{i^*_0})^{-1} \cdot \alpha^{\ell+1-i_0}$, we obtain after taking (9), (10) and (12) into account, that

$$\begin{aligned}
&\tau_1^{x} \cdot \left(u_{0,0} \cdot \prod_{i=1}^{\ell} u_i^{t_i} \right)^{r} \overset{x\equiv\alpha}{=} (g_1^{\alpha})^{y} \cdot g_1^{(\alpha^{\ell+1})} \cdot \left(g_1^{Y_0(t)} \cdot \prod_{i=1}^{i_0} \left(g_1^{\alpha^i} \right)^{(t_i - t^*_i)} \right)^{r} \\
&= (g_1^{\alpha})^{y} \cdot g_1^{(\alpha^{\ell+1})} \cdot g_1^{\xi Y_0(t)} \cdot \left(\prod_{i=1}^{i_0} \left(g_1^{\alpha^i} \right)^{(t_i - t^*_i)} \right)^{\xi} \cdot \left(g_1^{(\alpha^{\ell+1-i_0})} \right)^{-\frac{Y_0(t)}{t_{i_0} - t_{i^*_0}}} \\
&\quad \cdot \prod_{i=1}^{i_0-1} \left(g_1^{(\alpha^{\ell+1+i-i_0})} \right)^{-\frac{t_i - t^*_i}{t_{i_0} - t^*_{i_0}}} \cdot g_1^{(\alpha^{\ell+1})} \qquad (13) \\
&= (g_1^{\alpha})^{y} \cdot g_1^{\xi Y_0(t)} \cdot \left(\prod_{i=1}^{i_0} \left(g_1^{\alpha^i} \right)^{(t_i - t^*_i)} \right)^{\xi} \cdot \left(g_1^{(\alpha^{\ell+1-i_0})} \right)^{-\frac{Y_0(t)}{t_{i_0} - t_{i^*_0}}} \\
&\quad \cdot \prod_{i=1}^{i_0-1} \left(g_1^{(\alpha^{\ell+1+i-i_0})} \right)^{-\frac{t_i - t^*_i}{t_{i_0} - t^*_{i_0}}}.
\end{aligned}$$

It is immediately seen that the maximal power of α in the last equality is $i_0 \leq \ell$, thus this formula can be explicitly computed, as we know $G_1(\alpha, \beta)$.

Further, selecting $\eta_j \xleftarrow{\$} \mathbb{F}_p$ and assigning $s_j = \eta_j$ and taking (11)-(12) into account, we get for every $m \neq m^*$

$$\left(u_{0,j} \cdot \prod_{i=1}^{\ell} u_i^{m_{j,i}}\right)^{s_j} = \left(g_1^{y_{0,j}} \prod_{i=1}^{\ell} \left(g_1^{(\alpha^i)}\right)^{-m^*_{j,i}} \cdot \prod_{i=1}^{\ell} \left(g_1^{y_i} g_1^{(\alpha^i)}\right)^{m_{j,i}}\right)^{\eta_j} = \left(g_1^{Y_j(m_j)} \cdot \prod_{i=1}^{\ell} \left(g_1^{\alpha^i}\right)^{(m_{j,i}-m^*_{j,i})}\right)^{\eta_j}.$$

Combining this with (13), we obtain the form of σ_1 as in 6.1 of Algorithm $\mathcal{B}$.

Let us go to the stage break in. According to the model, when the time $\bar{t}$ is reached, then the secret key $\mathsf{sk}_{\bar{t}}$ is handed to $\mathcal{A}$. In the considered scheme $\mathsf{sk}_{\bar{t}} = (\mathsf{scp}, \textsc{Stack})$, where scp serves for making signatures related to the current time period $\bar{t}$, whereas STACK keeps data needed only for updating the key.

The signing component is given by (3). Therefore, there exists $r \in \mathbb{F}_p$ such that $A = \tau_1^x \cdot \left(u_{0,0} \prod_{i=1}^{\ell} u_i^{\bar{t}_i}\right)^r$ and $C = g_2^r$. If we select $\xi \xleftarrow{\$} \mathbb{F}_p$ and put $r = \xi - (\bar{t}_{i_0} - t^*_{i_0})^{-1} \cdot \alpha^{\ell+1-i_0}$, then we obtain the proper distribution, as r and ξ are equally probable. Analogously as above, there is i_0, such that $i_0 := \max\{i \in [\ell] \mid t_i \neq t^*_i\}$ thus, carrying out the same steps as above, we obtain

$$A \overset{x=\alpha}{=} (g_1^{\alpha})^y \cdot g_1^{\xi Y_0(\bar{t})} \cdot \left(\prod_{i=1}^{i_0} \left(g_1^{\alpha^i}\right)^{(\bar{t}_i - t^*_i)}\right)^{\xi} \cdot \left(g_1^{(\alpha^{\ell+1-i_0})}\right)^{-\frac{Y_0(\bar{t})}{t_{i_0} - t^*_{i_0}}} \cdot \prod_{i=1}^{i_0-1} \left(g_1^{(\alpha^{\ell+1+i-i_0})}\right)^{-\frac{\bar{t}_i - t^*_i}{\bar{t}_{i_0} - t^*_{i_0}}},$$

and $C = g_2^{\xi} \cdot \left(g_2^{(\alpha^{\ell+1-i_0})}\right)^{-(\bar{t}_{i_0} - t^*_{i_0})^{-1}}$. This justifies 7.1 of Algorithm $\mathcal{B}$.

Now, let us examine the second component, STACK. We start with a general remark, noting that each node being on the stack sk_t.STACK associated with a time period $t < T$ has the form (node, h), where node is of the form (1) and h is its height on the binary tree. Therefore, such nodes are uniquely determined by the binary strings $t_\ell \cdots t_h$ called indexes, which in turn are contained in STACKID. Besides, a position of (node, h) on sk_t.STACK is the same as a position of its index on STACKID, for every fixed $t < T$. This means that $\mathcal{B}$, simulating behavior of Π_{sfu} regarding a time period t, still keeps knowledge about indexes of nodes, where these nodes are originally from sk_t.STACK. Further, as all nodes create a binary tree, there is a unique path between any different two of them. In particular, there exists the unique path P, joining the root and the leaf at index $t^* = (t^*_\ell \cdots t^*_1)_2$. Therefore, taking any $(\mathsf{node}, h) \in \mathsf{sk}_{\bar{t}}$.STACK, we know by Section IV-C and the assumption $t^* < \bar{t}$, that $\mathsf{node} \notin P$. It is because, in other case the index of node would be $t^*_\ell \cdots t^*_h$, what contradicts the construction of the algorithm KUpd. This implies that if $t_\ell \cdots t_h$ is the index of node, then there is $i_0 \in \{h, \ldots, \ell\}$ such that $i_0 := \max\{i \in \{h, \ldots, \ell\} \mid t_i \neq t^*_i\}$. By (1) we know that node $= \left(A'; (b'_i)_{i\in[h-1]}; C'\right)$, where $A' = \tau_1^x \cdot \left(u_{0,0} \prod_{i=h}^{\ell} u_i^{t_i}\right)^r$, $b'_i = u_i^r$ and $C' = g_2^r$. Choosing $\xi \xleftarrow{\$} \mathbb{F}_p$ uniformly at random and putting $r = \xi - (\bar{t}_{i_0} - t^*_{i_0})^{-1} \cdot \alpha^{\ell+1-i_0}$, we get after having regard to the substitutions (9), (10) and (12), that

$$\begin{aligned} A' &\overset{x=\alpha}{=} (g_1^{\alpha})^y \cdot g_1^{(\alpha^{\ell+1})} \cdot \left(g_1^{y_{0,0}} \cdot \prod_{i=1}^{\ell} \left(g_1^{(\alpha^i)}\right)^{-t_i^*} \cdot \prod_{i=h}^{\ell} g_1^{y_i t_i} \left(g_1^{(\alpha^i)}\right)^{t_i}\right)^r \\ &= (g_1^{\alpha})^y \cdot g_1^{(\alpha^{\ell+1})} \cdot \left(g_1^{y_{0,0} + \sum_{i=h}^{\ell} y_i t_i} \cdot \prod_{i=1}^{h-1} \left(g_1^{(\alpha^i)}\right)^{-t_i^*} \cdot \prod_{i=h}^{\ell} \left(g_1^{(\alpha^i)}\right)^{(t_i - t_i^*)}\right)^r \\ &= (g_1^{\alpha})^y \cdot g_1^{\xi\left(y_{0,0} + \sum_{i=h}^{\ell} y_i t_i\right)} \cdot \prod_{i=1}^{h-1} \left(g_1^{(\alpha^i)}\right)^{-\xi t_i^*} \cdot \prod_{i=h}^{\ell} \left(g_1^{(\alpha^i)}\right)^{\xi(t_i - t_i^*)} \\ &\quad \cdot g_1^{-\frac{y_{0,0} + \sum_{i=h}^{\ell} y_i t_i}{t_{i_0} - t^*_{i_0}} \cdot (\alpha^{\ell+1-i_0})} \cdot \prod_{i=h}^{i_0-1} \left(g_1^{(\alpha^{\ell+1+i-i_0})}\right)^{-\frac{t_i - t_i^*}{t_{i_0} - t^*_{i_0}}}. \end{aligned}$$

Moreover, we have

$$b'_i = \left(g_1^y \cdot g_1^{\alpha^i}\right)^r = g_1^{\xi y_i} \cdot \left(g_1^{(\alpha^i)}\right)^{\xi} \cdot \left(g_1^{(\alpha^{\ell+1-i_0})}\right)^{-\frac{y_i}{t_{i_0} - t^*_{i_0}}} \cdot \left(g_1^{(\alpha^{\ell+1-i_0+i})}\right)^{-\frac{1}{t_{i_0} - t^*_{i_0}}},$$

for $i = 1, \ldots, h-1$ and obviously $C' = g_2^{\xi - (t_{i_0} - t^*_{i_0})^{-1} \cdot (\alpha^{\ell+1-i_0})}$.

Finally, let $\sigma = \left(\sigma_1, \sigma_2, (\sigma_{3,j})_{j\in[l]}\right)$ be a forged signature of m^* in the time period t^*, then according to (7) we know that

$$\begin{aligned} \hat{e}(\sigma_1^*, g_2) &= \hat{e}\left(g_1^y g_1^{\alpha^\ell}, g_2^{\alpha}\right) \cdot \hat{e}\left(g_1^{Y_0(t^*)}, \sigma_2^*\right) \cdot \prod_{j=1}^{l} \hat{e}\left(g_1^{Y_j(m^*_{j,i})}, \sigma^*_{3,j}\right) \\ &= \hat{e}\left((g_1^{\alpha})^y, g_2\right) \cdot \hat{e}\left(g_1^{(\alpha^{\ell+1})}, g_2\right) \cdot \hat{e}\left(g_1^{Y_0(t^*)}, \sigma_2^*\right) \cdot \prod_{j=1}^{l} \hat{e}\left(g_1^{Y_j(m^*_{j,i})}, \sigma^*_{3,j}\right). \end{aligned}$$

It is easy to see that the above formula can be written in the following form

$$\hat{e}(\sigma_1^*, g_2^{\beta})^{\beta^{-1}} = \left(\hat{e}\left((g_1^{\alpha})^y, g_2^{\beta}\right) \cdot \hat{e}\left(g_1^{(\alpha^{\ell+1})}, g_2^{\beta}\right) \cdot \hat{e}\left((g_1^{\beta})^{Y_0(t^*)}, \sigma_2^*\right) \cdot \prod_{j=1}^{l} \hat{e}\left((g_1^{\beta})^{Y_j(m^*_{j,i})}, \sigma^*_{3,j}\right)\right)^{\beta^{-1}}.$$

This in conjunction with the fact that p is a prime number lead us to the conclusion that the formula in 9 of Algorithm $\mathcal{B}$ essentially equals $\hat{e}\left(g_1^{(\alpha^{\ell+1})}, g_2^{\beta}\right)$.

In conclusion, we have showed that if $\mathcal{A}$ is able to make sfu-cma forgery and independently $Y_0(t^*) \neq 0$ in $\mathbb{F}_p$, then $\mathcal{B}$ solves $(\ell, 1)$-wBDHI$^*_3$ problem. Therefore, the following estimation is satisfied

$$\mathbf{Adv}^{\mathsf{sfu\text{-}cma}}_{\Pi_{\mathsf{sfu}},n}(\mathcal{A}) \leq \frac{p}{p-1} \cdot \mathbf{Adv}^{(\ell,1)\text{-wBDHI}^*_3}_{\mathsf{params},n}(\mathcal{B}).$$

This proofs the assertion.

B. Forward security of the scheme

Now we are ready to justify the forward security of the presented scheme.

Let $\Pi_{\mathsf{fu}} = (\mathscr{G}, \mathsf{KGen}, \mathsf{KUpd}, \mathsf{Sign}, \mathsf{Vrfy})$ *be the scheme defined in Section IV with the associated message space* $\mathcal{M} = \{0,1\}^*$. *If* $(\ell, 1)$*-wBDHI*$^*_3$ *is hard relative to* $\mathscr{G}$*, then* Π_{fu} *is forward-secure in the random-oracle model .*

The proof of this assertion immediately follows from Section V-A. It is because, if $H : \mathcal{M} \to \{0,1\}^{l\ell}$ is modeled as a random oracle, then given an adversary $\mathcal{A}$, attacking fu-cma security in the random oracle model, it is possible to break sfu-cma security. To be more precise, we are able to construct an sfu-cma adversary $\mathcal{B}$, that guesses the time frame t^* and the index of $\mathcal{A}$'s random oracle query for $H(\mathbf{m}^*)$. Note that, if $\mathcal{B}$ has chosen a proper t^* then it can set $\bar{t} \leftarrow t^* + 1$ and use $\mathsf{sk}_{\bar{t}}$ to simulate oracles KUpd and Break after the time period $\bar{t}$ and up to the point when $\mathcal{A}$ requests to launch the oracle Break.

Finally, if $\mathcal{B}$ has correctly guessed the index of $H(\mathbf{m}^*)$, then a forgery output by $\mathcal{A}$ is a valid forgery for $\mathcal{B}$. This means that intersection of three independent events, namely making a forgery by $\mathcal{A}$ and the proper choice of both t^* and the random oracle query, implies the event that $\mathcal{B}$ breaks the sfu-cma security. Therefore, we get the estimation

$$\mathbf{Adv}^{\mathsf{fu\text{-}cma}}_{\Pi_{\mathsf{fu}},n}(\mathcal{A}) \leq q_H(T-1) \cdot \mathbf{Adv}^{\mathsf{sfu\text{-}cma}}_{\Pi_{\mathsf{sfu}},n}(\mathcal{B}),$$

where q_H is the number of random-oracle queries made by $\mathcal{A}$. Hence, (8) then yields $\mathbf{Adv}^{\mathsf{fu\text{-}cma}}_{\Pi_{\mathsf{fu}},n}(\mathcal{A}) = \mathsf{negl}(n)$. This completes the proof.

REFERENCES

[1] A. Anderson, Invited lecture, in *Fourth Annual Conference on Computer and Communications Security, ACM*, Am Psychiatric Assoc, 1997.

[2] M. Bellare and S. K. Miner, "A Forward-Secure Digital Signature Scheme", in *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference*, 1999, pp. 431–449, doi:10.1007/3-540-48405-1_28.

[3] D. Boneh and X. Boyen, "Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles", in *Advances in Cryptology - EUROCRYPT 2004*, C. Cachin and J.L. Camenisch, Eds. 2004, pp. 223-238.

[4] D. Boneh, X. Boyen and E.-J. Goh, "Hierarchical Identity Based Encryption with Constant Size Ciphertext", Cryptology ePrint Archive, Report 2005/015. [Online]. Available: https://eprint.iacr.org/2005/015.pdf.

[5] X. Boyen, H. Shacham, E. Shen and B. Waters, "Forward Secure Signatures with Untrusted Update", in *Proceedings of CCS 2006*, W. Rebecca Ed. 2006, pp. 191–200.

[6] J. Buchmann, E. Dahmen and A. Hülsing, "XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions", in *Post-Quantum Cryptography*, B.-Y. Yang, Ed. 2011, pp. 117–129.

[7] J. Camenisch and M. Koprowski, "Fine-grained Forward-secure Signature Schemes without Random Oracles", *Discrete Applied Mathematics*, vol. 154, no. 2, pp. 175–188, Feb. 2006, doi: 10.1016/j.dam.2005.03.028.

[8] R. Canetti, S. Halevi, J. Katz, "A Forward-Secure Public-Key Encryption Scheme", in *Advances in Cryptology - EUROCRYPT 2003*, E. Biham, Ed. 2003, pp. 255–271.

[9] Y. Cui, E. Fujisaki, G. Hanaoka, H. Imai and R. Zhang, "Formal Security Treatments for Signatures from Identity-Based Encryption", in *Provable Security*, W. Susilo, J. K. Liu, Y. Mu, Eds. 2007, pp. 218–227.

[10] A. Fiat and A. Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems", in *Conference on the theory and application of cryptographic techniques*, 1986, pp. 186–194.

[11] S. D. Galbraith, K. G. Paterson and N. P. Smart, "Pairings for Cryptographers", *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113 - 3121, Sep. 2008, doi: 10.1016/j.dam.2007.12.010.

[12] S. Goldwasser S. Micali and R. L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks", *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281–308, 1988, doi: 10.1137/0217017.

[13] S. Hohenberger and B.Waters, "New Methods and Abstractions for RSA-Based Forward Secure Signatures", in *International Conference on Applied Cryptography and Network Security*, M. Conti, J. Zhou, E. Casalicchio and Angelo Spognardi, Eds. 2020, pp. 292–312.

[14] G. Itkis, and L. Reyzin, "Forward-secure Signatures with Optimal Signing and Verifying", in *Advances in Cryptology - CRYPTO '01, 21st Annual International Cryptology Conference*, J. Kilian, Ed. 2001, pp. 332–354.

[15] M. Jurkiewicz, "Improving Security of Existentially Unforgeable Signature Schemes", *International Journal of Electronics and Telecommunications*, vol. 66, no. 3, pp. 473–480, 2020, doi: 0.24425/ijet.2020.131901.

[16] H. Krawczyk, "Simple Forward-secure Signatures from any Signature Scheme", in *Proceedings of the 7th ACM conference on Computer and Communications Security*, P. Samarati, Ed. 2000, pp. 108–115, doi: 10.1145/352600.352617.

[17] S. Mitsunari, R. Sakai and M. Kasahara, "A new traitor tracing", *IEICE transactions on fundamentals of electronics, communications and computer sciences*, vol. 85, no. 2, pp. 481–484, Feb. 2002.