Constructing a pairing-free certificateless proxy signature scheme from ECDSA

Cholun Kim [1]

[1] Faculty of Mathematics, Kim Il Sung University, Pyongyang, Democratic People's Republic of Korea

E-mail address: cu.kim1019@ryongnamsan.edu.kp

Abstract

Proxy signature is a kind of digital signature, in which a user called original signer can delegate his signing rights to another user called proxy signer and the proxy signer can sign messages on behalf of the original signer. Certificateless proxy signature (CLPS) means proxy signature in the certificateless setting in which there exists neither the certificate management issue as in traditional PKI nor private key escrow problem as in Identity-based setting. Up to now, a number of CLPS schemes have been proposed, but some of those schemes either lack formal security analysis or turn out to be insecure and others are less efficient because of using costly operations including bilinear pairings and map-to-point hashing on elliptic curve groups.

In this paper, we formalize the definition and security model of CLPS schemes. We then construct a pairing-free CLPS scheme from the standard ECDSA and prove its security in the random oracle model under the discrete semi-logarithm problem's hardness assumption as in the provable security result of ECDSA.

Keywords: proxy signature; certificateless cryptography; random oracle model; provable security; elliptic curve digital signature algorithm.

1. Introduction

Proxy signature is a kind of digital signature, in which a user called original signer can delegate his signing rights to another user called proxy signer and the proxy signer can sign messages on behalf of the original signer, in case of temporal absence of him, lack of his time, etc. Since the first scheme of the proxy signature was introduced by Mambo et al. [17] in 1996, it has found a lot of practical applications in distributed systems where delegation of signing rights is quite common, including cloud computing, global distribution networks, and electronic commerce.

Mambo et al. [17] classified proxy signature schemes into three types, based on the delegation mode: *full delegation*, *partial delegation* and *delegation by warrant*. In the *full delegation*, the original signer entirely grants his private key (signing key) to the proxy signer. Therefore, the proxy signer has all signing right of the original signer and such schemes have no non-repudiation. In the *partial delegation*, a proxy signer obtains a new key, called *proxy signing key*, which is derived from the private key (signing key) of the original signer and the public or private key of the proxy signer. Obviously, proxy signatures generated by using proxy signing key are distinguished from the original signer's signatures, but the proxy signature with partial delegation still suffers from the problem that the proxy signer has no limit on which messages and when he can sign. In the *delegation by warrant*, this problem can be solved. Warrants are usually written by the original signers and can include identities of the original signer and the proxy signer, the delegation period, the associated public keys, and other information. Actually, it is possible to contain any type of security policy that specifies the restrictions under which the delegation is valid, such as the formats of messages to be signed, the number of signatures to be generated by proxy signer, etc. The original signers prepare the warrants at their will and sign them to certify the legitimacy of the proxy signers and be able to delegate the limited signing right to the proxy signers. Therefore, most of works in the literature focus on the proxy signature schemes with the delegation by warrant. The warrant signed by the original signer is called the *delegation*.

In general, a secure proxy signature scheme should satisfy the following security requirements which is introduced by Mambo et al. [17]: (1) *Strong Unforgeability*: Only the designated proxy signer can create a valid proxy signature and even the original signer cannot. (2) *Verifiability*: Anyone can verify the proxy signature and its conformance to the original signer's agreement or delegation. (3) *Strong Identifiability*: From the proxy signature, anyone can determine the identity of corresponding proxy signer. (4) *Strong Undeniability*: A proxy signer cannot deny his or her signature ever generated against anyone. (5) *Distinguishability*: Anyone can distinguish the proxy signature from the original signer's normal signature. (6) *Prevention of misuse*: The proxy signer cannot sign messages that have not been authorized by the original signer.

Among the above security requirements for a secure proxy signature scheme, the strong unforgeability is the most important and difficult one to satisfy. In most of proposed schemes, the other security properties are easily derived from the correctness and the strong

unforgeability of the schemes.

Like other types of digital signatures, the proxy signature is a cryptographic primitive in public key cryptography (PKC), so that it is an important issue to guarantee the authenticity of public keys which match with private keys (signing keys) of signers. In traditional public key setting such as PKI, *certificates* generated by a *trusted third part* (TTP) are used to mitigate this issue, but it is costly to use and manage them, and brings about new problems. In order to bypass these problems, Shamir [19] introduced the concept of *Identity-based Signature* (IBS) along with Identity-based encryption (IBE). In IBS, the public key of a user is his identity information (e.g., his email address) which is approved publicly and doesn't need to be certified by any TTP, but all private keys are only generated by a TTP called private key generator (PKG). Therefore, the PKG can freely generate any user's signature on any message and this results in the *private key escrow problem* which is inherent in the identity-based setting. In 2003, Al-Riyami and Paterson [1] introduced *certificateless* public-key cryptography (CL-PKC) and the first *certificateless signature* (CLS) scheme to solve the private key escrow problem in the identity-based setting and eliminate the need of certificates in the conventional PKI.

In the certificateless setting, a TTP named a Key Generation Center (KGC) is also required for a user to generate his private key, but unlike a PKG in IBS schemes, the KGC produces a partial private key for any user by using his identity and a master key, and transfers it the user securely. Then the user generates his own full private key by combining the partial private key generated by KGC with a secret value chosen by himself. As a result, the KGC cannot know the full private key of any user and the key escrow problem in the identity-based setting can be solved. Meanwhile, the public key for each user in CLS schemes is no longer his identity, but it can be certified by the structure of the CL-PKC without any certificate.

1.1 Motivation

It is natural to combine the concepts of proxy signature with CL-PKC. The proxy signature in CL-PKC is called the *certificateless proxy signature* (CLPS), introduced by Li et al. [13] in 2005.

Li et al. [13] proposed the first scheme of the CLPS. Later, Yap et al. [25] and Lu et al. [14] found that Li et al.'s scheme is insecure. Furthermore, Lu et al. [14] improved Li et al.'s CLPS scheme, but they, just as Li et al. [13], did not present any formal security proof for the scheme. Then, the CLPS and its some extensions (e.g. *certificateless multi-proxy signature*,

certificateless proxy multi-signature, etc) have been discussed in many literatures [3, 4, 7, 11, 15, 18, 20, 21, 22, 23, 24, 26].

For the construction of a provably secure scheme, a number of security models for the CLPS schemes were proposed by Wan et al. [22] firstly, Chen et al. [3], Chen et al. [4], Jin and Wen [11], Zhang et al. [26], Xu et al. [24], Du and Wen [7], Padhye and Tiwari [18], etc. In the above security models, the power of adversaries and the security of a scheme have been defined. However, there are subtle differences in those definitions between the security models. Furthermore, even the definitions of a CLPS scheme lack consistency between many literatures.

The CLPS schemes with formal security proofs were proposed by Chen et al. [4], Jin and Wen [11], Tian et al. [21], Xu et al. [24], Zhang et al. [26], Du and Wen [7], Padhye and Tiwari [18], Lu and Li [15], etc. All the above schemes are based on bilinear pairing operations, except for the one in Padhye and Tiwari [18]. It is well known that the cost of paring operation is very high, and schemes without pairings would be more appealing in terms of efficiency. Padhye and Tiwari [18] proposed an elliptic curve discrete log problem (ECDLP)-based CLPS scheme without pairings. However, Shi et al. [20] showed their scheme is not secure for practical applications. Therefore, to the best of our knowledge, there is no provably secure CLPS scheme without pairings up to now.

Recently, Cheng and Chen [5] and Karati et al. [12] proposed some provably secure certificateless signature (CLS) schemes without pairings. Especially, Cheng and Chen [5] presented the generic approach to construct CLS schemes from standard algorithms such as ECDSA and EC-FSDSA [10].

Motivated from above reasons, we focus on the security model for certificateless proxy signature (CLPS) schemes and provably secure schemes without pairings.

1.2 Our contribution

The main contributions of this paper can be summarized as follows:

(1) Firstly, we formalize the definition of CLPS scheme and the security model for CLPS schemes. We carefully select algorithms which compose a CLPS scheme and specify their inputs/outputs to support good schemes proposed in the literature. We also redefine oracles which specify the capabilities of adversaries against a CLPS scheme, and present more clear definition of the security of a CLPS scheme to eliminate issues of previous security models as much as possible.

(2) Secondly, we propose an efficient CLPS scheme without bilinear pairings. The idea of this scheme is inspired by Cheng and Chen's technique [5] used to construct some CLS schemes from standard ECDSA (Elliptic Curve Digital Signature Algorithm).

(3) Finally, we show our scheme is provably secure in the random oracle model under the assumption that the semi-logarithm problem [2] is intractable.

Our scheme has the merits of CL-PKC. Besides, it is derived from the standard Elliptic Curve Digital Signature Algorithm (ECDSA) widely used as an international standard and is the provably secure CLPS scheme without bilinear pairings. Therefore, it can be easily implemented using existing security elements that support ECDSA, and is particularly useful in the Internet of Things (IoT) because of high efficiency of time and space.

1.3 Organization

The remainder of this paper is organized as follows. In Section 2, some preliminaries are given. In Section 3, we present the definition of a CLPS scheme and formalize the security model for CLPS schemes. Then, a concrete CLPS scheme without pairings is proposed in Section 4. We prove the security of the proposed scheme in Section 5. Finally, we conclude the paper in Section 6.

2. Notations and Preliminaries

Throughout this paper, we will use following notations:

- $\lambda \in \mathbb{N}$: the security parameter;
- 1^{λ}: the string consisting of λ ones;
- $s \leftarrow_{\mathrm{R}} S$: the operation of picking an element s uniformly at random from a finite set S;
- $r \leftarrow\!\!\!\shortmid \mathcal{A}(i_1, \cdots, i_m)$: the operation of running a probabilistic (randomized) algorithm $\mathcal{A}$ with input $i_1, \cdots, i_m$ and assigning the result to r;
- $r \leftarrow \mathcal{A}(i_1, \cdots, i_m)$: the operation of running a deterministic algorithm $\mathcal{A}$ with input $i_1, \cdots, i_m$ and assigning the result to r;
- $V_1 \,/\, V_2$: one of two values V_1 and V_2;
- $V_1 \| V_2$: the concatenation of the (binary) string representations of two values V_1 and V_2;

- $L := R$: the assignment of the value of R to L.
- $\perp$: the null symbol indicating a failure or a false value;
- $\emptyset$: the empty set or the empty string.
- Let $\mathbb{G}$ be a cyclic additive group of prime order q with the addition "+". If $P \in \mathbb{G}$ and $k \in \mathbb{Z}_q$, we denote $\underbrace{P + \cdots + P}_{k \text{ times}} \in \mathbb{G}$ as $[k]P$.
- PPT stands for *probabilistic polynomial-time*.

A function $f : \mathbb{N} \to \mathbb{R}^+$ is said to be *negligible* if, for all polynomial p, there exists an integer $k_p \in \mathbb{N}$ such that $f(n) < 1/|p(n)|$ for all $n > k_p$, or equivalently, if for all $c \in \mathbb{N}$ there exists an integer $k_c \in \mathbb{N}$ such that $f(n) < n^{-c}$ for all $n > k_c$.

Definition 1. Discrete Logarithm Problem (DLP). Let $\mathbb{G}$ be a cyclic additive group of prime order q and G is a generator of $\mathbb{G}$. The *discrete logarithm problem* (DLP) is given a random $P \in \mathbb{G}$ to find $k \in \mathbb{Z}_q^*$ such that $[k]G = P$. k is called the *discrete logarithm* of P to the base G.

Definition 2. Semi-Logarithm Problem (SLP). Let $\mathbb{G}$ be a cyclic additive group of prime order q, G is a generator of $\mathbb{G}$ and $f : \mathbb{G} \to \mathbb{Z}_q$ is a map (called a *conversion function*). The *semi-logarithm problem* (SLP) is given a random $P \in \mathbb{G}$ to find $(u, v) \in \mathbb{Z}_q \times \mathbb{Z}_q^*$ such that $u = f\left([v^{-1}](G + [u]P)\right)$. The pair of integers (u, v) is called the (*discrete*) *semi-logarithm* of P to the base G for the conversion function f and the group $\mathbb{G}$.

D. Brown [2] said "Finding the discrete logarithm of $P \in \mathbb{G}$ allows one to find its semi-logarithm." That is, if the SLP is intractable, the DLP is also intractable. But it is not obvious whether the SLP is equivalent to or easier than the DLP. The intractability of SLP is a necessary condition for the security of ECDSA. For ECDSA, $\mathbb{G}$ is a cyclic subgroup on an elliptic curve E over some finite field $\mathbb{F}$ and the conversion function f is taken by the function $\Pi_{\mathbf{x}}$ which maps the point $P \in \mathbb{G}$ to its x-coordinate $\mathrm{x}(P) \pmod q$ interpreted as an integer when performing the reduction modulo q [2].

In CLPS, more generally, in CL-PKC, two types of adversaries with different capabilities, Type I Adversaries and Type II Adversaries, are considered [1, 5, 7, 18];

- A Type I Adversary $\mathcal{A}_{\mathrm{I}}$ acts as a dishonest user, and can replace the public key of any user with his choice but cannot access to the master secret key.
- A Type II Adversary $\mathcal{A}_{\mathrm{II}}$ acts as a malicious KGC, and knows the master secret key (so can compute any user's partial private key for itself) but cannot replace any user's public key.

3. Definition and Security Model for CLPS

In this paper, we focus on CLPS schemes based on the delegation with warrant as in many works, since it can provide more security and flexibility.

3.1 Definition of a CLPS scheme

In this section, we first define a certificateless proxy signature (CLPS) scheme based on the delegation with warrant, and then consider some differences between the new definition and the previous ones.

We seek to make the definition more versatile by combining the merits of previous works, so that it will give us more flexibility and simplicity in formalizing old schemes and designing new schemes without weakening the security requirements or losing any of nice features specific to CLPS.

Definition 3. Let $\lambda \in \mathbb{N}$ be the security parameter. A *certificateless proxy signature (CLPS)* (*with appendix*) *scheme* $\mathbf{\Sigma}(\lambda)$ comprises entities of four types: the KGC, the original signer, the proxy signer and the verifier. It consists of the following seven polynomial-time algorithms:

(1) $(Params, s) \longleftarrow \mathsf{Setup}(1^{\lambda})$: As a probabilistic (randomized) algorithm run by the KGC, it outputs a list of system parameters $Params$ and a *master secret key* s.

(2) $(\mathrm{P}_U, \mathrm{s}_U) \longleftarrow \mathsf{GenerateUserKeys}(Params, \mathrm{ID}_U)$: This is a probabilistic algorithm run by any entity except the KGC. When a user U run, it takes as input $Params$ and his identity ID_U, and outputs a *public*/*secret key* pair $(\mathrm{P}_U, \mathrm{s}_U)$ for the user U.

(3) $(\mathrm{K}_U, \mathrm{c}_U) \longleftarrow \mathsf{GeneratePartialKeys}(Params, s, \mathrm{ID}_U, \mathrm{P}_U)$: As a probabilistic algorithm run by the KGC, it takes as input $Params$, the master secret key s, the identity

ID_U of a user U and his public key P_U, and outputs a pair of *partial public / private keys* $(\mathrm{K}_U, \mathrm{c}_U)$ for the user U. $(\mathrm{P}_U, \mathrm{K}_U)/(\mathrm{s}_U, \mathrm{c}_U)$ is the pair of *full public / private keys* for the user U.

(4) $\delta_O \leftarrow$ GenerateDelegation(*Params*, ID_O, P_O, K_O, c_O, s_O, w_O): This is a probabilistic algorithm run by original signers. When an original signer O run, it takes as input *Params*, the identity ID_O and the full public/private key pair $(\mathrm{P}_O, \mathrm{K}_O)/(\mathrm{s}_O, \mathrm{c}_O)$ of O, and a warrant w_O issued by O, and then outputs the *delegation* δ_O certified by O. δ_O includes w_O.

(5) **True**/**False** $\leftarrow$ VerifyDelegation(*Params*, ID_O, P_O, K_O, δ_O): This is a deterministic algorithm run by any user. It takes as input *Params*, the identity ID_O and the full public key $(\mathrm{P}_O, \mathrm{K}_O)$ of an original signer O, and the delegation δ_O issued by O. It outputs **True** if the delegation δ_O is valid, or **False** otherwise.

(6) $\sigma_m \leftarrow$ GenerateProxySign(*Params*, $\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \delta_O, \mathrm{ID}_P, \mathrm{P}_P, \mathrm{K}_P, \mathrm{s}_P, \mathrm{c}_P, m$): This is a probabilistic algorithm run by proxy signers. It takes as input *Params*, the identity ID_O and the full public key $(\mathrm{P}_O, \mathrm{K}_O)$ of an original signer O, the delegation δ_O issued by O, the identity ID_P and the full public/private key pair $(\mathrm{P}_P, \mathrm{K}_P)/(\mathrm{s}_P, \mathrm{c}_P)$ of the proxy signer P specified by the delegation δ_O, and a message m. It outputs a signature σ_m of P on the message m.

(7) **True**/**False** $\leftarrow$ VerifyProxySign(*Params*, ID_O, P_O, K_O, δ_O, ID_P, P_P, $\mathrm{K}_P, m, \sigma_m$): This is a deterministic algorithm run by any user (verifier). It takes as input *Params*, the identity ID_O and the full public key $(\mathrm{P}_O, \mathrm{K}_O)$ of an original signer O, and the delegation δ_O issued by O, the identity ID_P and the full public key $(\mathrm{P}_P, \mathrm{K}_P)$ of a proxy signer P, a message m, and a proxy signature σ_m. It outputs **True** if the delegation δ_O is valid, the proxy signer P is specified in the delegation δ_O, the message m conforms to the delegation δ_O and the proxy signature σ_m is valid, or **False** otherwise.

We can find some differences between the above definition and the previous definitions

for CLPS schemes, which was first introduced in [13], later proposed by Wan et al. [22], Chen et al. [4], Xu et al. [24], Zhang et al. [26], Padhye and Tiwari [18], Du and Wen [7], etc.

Main difference is in the 3rd algorithm, GeneratePartialKeys, generating partial keys of any user. In other definitions, it is usually named as Partial-Private-Key-Extract, takes system parameters, the master secret key and the identity of a user as input, and outputs only a partial private key of the user. However, in our definition, following what Cheng and Chen [5] has done for the definition of CL-PKC, the public key P_U of the user U is added as its input, and the public value K_U is also outputted from it. It is possible that the input parameter P_U are ignored within the algorithm and the public value K_U of the output is the empty string. Hence, any partial key generation algorithms following the other definitions can be covered by our definition. On the other hand, by adding the public key P_U selected by a user as input of GeneratePartialKeys, our definition can capture CLPS schemes achieving Girault's trust level 3 [8], such as Zhang et al.'s scheme [26]. This technique has been already discussed in [1]. By including the partial public key K_U into output of GeneratePartialKeys, our definition can explicitly depict the schemes in which a user's partial private key has a public part, such as Padhye-Tiwari scheme [18].

Another difference is that some algorithms in the traditional definitions of CLPS (more generally, CL-PKC) are eliminated in our definition. Firstly, in the same approach as taken by Wan et al. [22], Xu et al. [24], Zhang et al. [26], Du and Wen [7], etc., we eliminate the algorithm Set-Secret-Value generating a user secret value, the algorithm Set-Private-Key generating a user private key, and the algorithm Set-Public-Key generating a user public key, by including the function of Set-Private-Key into other algorithms and combining the functions of Set-Secret-Value and Set-Public-Key with GenerateUserKeys. The detailed advantages of this approach can be referred to [9]. Secondly, we eliminate the algorithm generating a proxy signing key and include the function of the algorithm into the proxy signing algorithm GenerateProxySign. This makes the our definition more versatile as it is possible that the partial private key c_P and the secret key s_P of the proxy signer, the delegation δ_O, and the others are 'mixed' together in some randomized way during proxy signing.

Final and minor difference is in specifying the input/output of algorithms according to

the above modifications on the old definitions. At least, in our definition, partial public keys and identities of users are explicitly added to input of such algorithms as GenerateDelegation, VerifyDelegation, GenerateProxySign, VerifyProxySign, etc.

3.2 Security Model for CLPS schemes

As described in Section 2, there are two types of adversaries, types I and II, against CLPS as a cryptographic primitive of CL-PKC. In view of proxy signature, the final goal of Type I/II adversaries against CLPS is to forge a valid proxy signature on behalf of their target proxy signer or a valid delegation on behalf of their target original signer without the knowledge of the full private key of the target user.

The adversary against a CLPS scheme of Definition 3 is defined as follows.

Definition 4. Let $\lambda \in \mathbb{N}$ be the security parameter. An *adversary* $\mathcal{A}(\lambda)$ against a CLPS scheme $\mathbf{\Sigma}(\lambda)$ is a probabilistic algorithm which takes as input 1^{λ} and can access to some of the following oracles (as well as the random oracles if there exists):

(1) $(\mathrm{P}_U, \mathrm{K}_U) \leftarrow\!\!\!\shortmid \mathcal{O}_{\mathrm{CreateUser}}(\mathrm{ID}_U)$: On input an identity ID_U, it creates a new user U whose identity is ID_U if the user has not yet been created. At the same time, a partial key pair and a pair of public/secret keys of the user U are generated or updated and the full public key $(\mathrm{P}_U, \mathrm{K}_U)$ is returned.

(2) $\mathrm{s}_U / \perp \leftarrow \mathcal{O}_{\mathrm{GetSecretKey}}(\mathrm{ID}_U, \mathrm{P}_U)$: On input the identity ID_U and the public key P_U of a user U, it records $(\mathrm{ID}_U, \mathrm{P}_U)$ in the query list $\mathbf{S}$ and outputs the user's secret key s_U pairing with P_U, if the user U has been created. It outputs $\perp$ otherwise.

(3) $\mathrm{c}_U / \perp \leftarrow \mathcal{O}_{\mathrm{GetPartialKeys}}(\mathrm{ID}_U, \mathrm{K}_U)$: On input the identity ID_U and the partial public key K_U of a user U, it records $(\mathrm{ID}_U, \mathrm{K}_U)$ in the query list $\mathbf{P}$ and outputs user's partial private key c_U pairing with K_U, if there exists the user U. It outputs $\perp$ otherwise. When K_U is the empty string, c_U is the current partial private key of the user U.

(4) $(\mathrm{P}_U, \mathrm{K}_U) / \perp \leftarrow \mathcal{O}_{\mathrm{GetPublicKey}}(\mathrm{ID}_U)$: On input the identity ID_U of a user U, it records $(\mathrm{ID}_U, \mathrm{K}_U)$ in the query list $\mathbf{P}$ and outputs the full public key $(\mathrm{P}_U, \mathrm{K}_U)$ currently associated with ID_U, if there exists the user U. It outputs $\perp$ otherwise.

(5) "OK"/$\perp \leftarrow \mathcal{O}_{\text{ReplaceUserKeys}}(\mathrm{ID}_U, \mathrm{P}'_U, \mathrm{s}'_U)$: On input the identity ID_U, a new public key P'_U and a new secret key s'_U of a user U, it records $(\mathrm{ID}_U, \mathrm{P}'_U)$ in the query list $\mathbf{R}$, raplaces the current pair of partial public / private keys of the user U with the new pair $(\mathrm{P}'_U, \mathrm{s}'_U)$ and outputs "OK", if there exists the user U. It outputs $\perp$ otherwise. s'_U may be the empty string.

(6) $\delta_O / \perp \leftarrow \mathcal{O}_{\text{GetDelegation}}(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{w}_O)$: On input the identity ID_O of an original signer O, a full public key $(\mathrm{P}_O, \mathrm{K}_O)$ and a warrant w_O, it outputs the valid delegation δ_O (created newly if there does not exist) on the warrant w_O and records the tuple $(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{w}_O, \delta_O)$ in the query-answer list $\mathbf{D}$, if there exists the user O and $(\mathrm{P}_O, \mathrm{K}_O)$ is the (current or past) full public key of O. It outputs $\perp$ otherwise.

(7) $\sigma_m / \perp \leftarrow \mathcal{O}_{\text{GetProxySign}}(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \delta_O, \mathrm{ID}_P, \mathrm{P}_P, \mathrm{K}_P, m)$: On input the identity ID_O and a full public key $(\mathrm{P}_O, \mathrm{K}_O)$ of an original signer O, a delegation δ_O, the identity ID_P and a full public key $(\mathrm{P}_P, \mathrm{K}_P)$ of a proxy signer P, and a message m, it outputs the valid proxy signature σ_m on the message m. If there does not exist one of O and P, δ_O is not valid, or ID_P and m do not conform to δ_O, it returns $\perp$.

An adversary is called the *Type I adversary* and denoted by $\mathcal{A}_{\mathrm{I}}(\lambda)$ if it can access all the above oracles but cannot access to the master secret key. An adversary is called the *Type II adversary* and denoted by $\mathcal{A}_{\mathrm{II}}(\lambda)$ if it knows the master secret key but can access only 5 oracles of the above oracles except $\mathcal{O}_{\text{GetPartialKeys}}$ and $\mathcal{O}_{\text{ReplaceUserKeys}}$.

Our definition of oracles for adversaries is similar to the one in [26] but has some differences from others including it. Firstly, the oracle $\mathcal{O}_{\text{CreateUser}}$ can not only create a new user but also update all keys for any existing user. By querying to this oracle, adversaries can trigger the legitimate renewal of all keys for any user. This makes our definition of adversary more practical. Secondly, a user's public key P_U and partial public key K_U are explicitly added to input of $\mathcal{O}_{\text{GetSecretKey}}$ and $\mathcal{O}_{\text{GetPartialKeys}}$, respectively. By this, our definition can capture the actions of adversaries trying to find something from the system's history in the certificateless settings with freely revocable keys. Thirdly, as a consequence of our definition

of CLPS scheme excluding the algorithm generating a proxy signing key, we eliminate the oracle returning a proxy signing key unlike other models. Actually, it is not necessary for adversaries with having access to $\mathcal{O}_{\text{GetSecretKey}}$ and $\mathcal{O}_{\text{GetPartialKeys}}$. Finally, the specification of the input/output of oracles is more clear and compact.

Now, we define the interactive security game between an adversary $\mathcal{A}(\lambda) \in \{\mathcal{A}_{\text{I}}(\lambda), \mathcal{A}_{\text{II}}(\lambda)\}$ against a CLPS scheme $\mathbf{\Sigma}(\lambda)$ and a challenger $\mathcal{C}$, where $\lambda \in \mathbb{N}$ is the security parameter. The game consists of three stages: Setup, Attack and Forgery.

EUF-CMA Game:

Setup: $\mathcal{C}$ runs Setup(1^{λ}) to obtain the system parameter list *Params* and the master secret key s. Then $\mathcal{C}$ sends *Params* to the adversary $\mathcal{A}(\lambda)$. If $\mathcal{A}(\lambda)$ is $\mathcal{A}_{\text{II}}(\lambda)$, $\mathcal{C}$ also passes s to the adversary $\mathcal{A}(\lambda)$. Otherwise, keeps the master secret key s secret. Finally, $\mathcal{C}$ initializes the lists $\mathbf{P}$, $\mathbf{S}$, $\mathbf{R}$ and $\mathbf{D}$ with empty list ($\emptyset$) respectively.

Attack: $\mathcal{A}(\lambda)$ gathers information by adaptively querying oracles allowed to him and. $\mathcal{C}$ correctly simulates the oracles called by $\mathcal{A}(\lambda)$ and returns proper values.

Forgery: Finally, $\mathcal{A}(\lambda)$ outputs a tuple $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*, \sigma_{m^*})$.

We say that $\mathcal{A}(\lambda)$ *wins* the game or *succeeds* in the game if and only if the following conditions are satisfied:

(1) VerifyProxySign(*Params*, $\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*, \sigma_{m^*}$)=**True**.

(2) The tuple $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*)$ has not been submitted to the oracle $\mathcal{O}_{\text{GetProxySign}}$ in the *Attack* stage.

(3) When $\mathcal{A}(\lambda)$ is $\mathcal{A}_{\text{I}}(\lambda)$, the following boolean expression is true:

$$(((\mathrm{ID}_O^*, \mathrm{K}_O^*) \notin \mathbf{P} \vee (\mathrm{ID}_O^*, \mathrm{P}_O^*) \notin \mathbf{S} \cup \mathbf{R}) \wedge (\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \mathrm{w}_O^*, *) \notin \mathbf{D}) \vee$$

$$((\mathrm{ID}_P^*, \mathrm{K}_P^*) \notin \mathbf{P} \vee (\mathrm{ID}_P^*, \mathrm{P}_P^*) \notin \mathbf{S} \cup \mathbf{R}),$$

where w_O^* is the warrant in the delegation δ_O^* and * means for an arbitrary value.

When $\mathcal{A}(\lambda)$ is $\mathcal{A}_{\text{II}}(\lambda)$, the following boolean expression is true:

$$((\mathrm{ID}_O^*, \mathrm{P}_O^*) \notin \mathbf{S} \wedge (\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \mathrm{w}_O^*, *) \notin \mathbf{D}) \vee ((\mathrm{ID}_P^*, \mathrm{P}_P^*) \notin \mathbf{S}).$$

The security of a CLPS scheme is defined as follows.

Definition 5. A CLPS scheme is said to be existentially unforgeable or secure against an adaptively chosen-message attack and an adaptively chosen-warrant attack if for the security parameter $\lambda \in \mathbb{N}$, no PPT adversary $\mathcal{A}(\lambda)$ against the scheme has a non-negligible probability of success in the above game.

An adversary against a CLPS scheme can win the above game by forging a valid proxy signature or a valid delegation. But in the security models proposed by Chen et al. [4], Xu et al. [24], Padhye and Tiwari [18], Du and Wen [7], etc., an adversary can win only when he is able to forge a valid signature. Only Zhang et al.'s model [26] takes forgery of a delegation into account. Comparing Zhang et al.'s model [26], our definition of security is more concise and easy to use.

4. A CLPS scheme without bilinear pairings

In this section, we propose a pairing-free CLPS scheme in conformity with Definition 3, and consider the correctness and the efficiency of it.

Our scheme is based on ECDSA [10] and consists of the following seven algorithms:

(1) $(Params, s) \longleftarrow \mathsf{Setup}(1^{\lambda})$: KGC takes the security parameter $\lambda \in \mathbb{N}$ as input and returns $Params$ and a master secret key s as follows:

a. Chooses a λ-bit prime p, and determines a finite field $\mathbb{F}_p$ of order p, an elliptic curve $E/\mathbb{F}_p$ defined by $E: Y^2 = X^3 + aX + b$ over $\mathbb{F}_p$, a cyclic additive subgroup $\mathbb{G}$ of prime order q on $E/\mathbb{F}_p$ and a generator G of $\mathbb{G}$.

b. $s \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $P_{\mathrm{KGC}} \leftarrow [s]G$.

c. Chooses an integer $n > 0$, and cryptographic hash functions $\mathrm{H}_1 : \{0,1\}^* \to \{0,1\}^n$, $\mathrm{H}_2 : \{0,1\}^* \to \mathbb{Z}_q^*$, $\mathrm{H}_3 : \{0,1\}^* \to \mathbb{Z}_q^*$ and $\mathrm{H}_4 : \{0,1\}^* \to \mathbb{Z}_q^*$.

d. $B_{Params} := a \,\|\, b \,\|\, G \,\|\, P_{\mathrm{KGC}}$. (We assume that the binary representaions of a, b, G and P_{KGC} are obtained in a trivial way.)

e. Publishes $Params := \{a, b, p, q, G, P_{\mathrm{KGC}}, \mathrm{H}_1, \mathrm{H}_2, \mathrm{H}_3, \mathrm{H}_4\}$ and keeps s secret

(2) $(\mathrm{P}_U, \mathrm{s}_U) \longleftarrow \mathsf{GenerateUserKeys}(Params, \mathrm{ID}_U)$: The user U with the identity ID_U sets his public key P_U and secret key s_U as follows:

a. $s_U \leftarrow_R \mathbb{Z}_q^*$; $P_U \leftarrow [s_U]G$.

(3) $(K_U, c_U) \leftarrow$ GeneratePartialKeys($Params, s, ID_U, P_U$): On input $Params$, the master secret key s, a user's identity ID_U and public key P_U, KGC returns a pair of partial public / private keys (K_U, c_U) for the user U with the identity ID_U as follows:

a. $h_1 \leftarrow H_1(B_{Params} \| ID_U)$.

b. $k_U \leftarrow_R \mathbb{Z}_q^*$; $K_U \leftarrow [k_U]G$.

c. $h_2' \leftarrow H_2(K_U \| h_1)$. // (We assume that the binary representaion of K_U is obtained in a trivial way.)

d. $c_U \leftarrow (k_U + s \cdot h_2') \bmod q$.

e. Publishes K_U and sends c_U to U via a secure channel.

// U can validate its partial keys (K_U, c_U) by checking the equation

$$[c_U]G \stackrel{?}{=} K_U + [H_2(K_U \| H_1(B_{Params} \| ID_U))]P_{KGC}$$

(4) $\delta_O \leftarrow$ GenerateDelegation($Params, ID_O, P_O, K_O, s_O, c_O, w_O$): The original signer O outputs the delegation δ_O on the warrant w_O as follows:

a. $h_1 \leftarrow H_1(B_{Params} \| ID_O)$; $h_2 \leftarrow H_2(P_O \| K_O \| h_1)$.

b. $h_3 \leftarrow H_3(w_O \| h_2)$.

c. $r_O \leftarrow_R \mathbb{Z}_q^*$; $R_O \leftarrow [r_O]G$.

d. **if** $\Pi_x(R_O)=0$ **then goto** c.

e. $\sigma_O \leftarrow r_O^{-1}(\Pi_x(R_O) \cdot (s_O + c_O) + h_3) \bmod q$.

f. **if** $\sigma_O=0$ **then goto** c.

g. $\delta_O := (w_O, R_O, \sigma_O)$; **return** δ_O.

(5) **True**/**False** $\leftarrow$ VerifyDelegation($Params, ID_O, P_O, K_O, \delta_O$): Any verifier, including the proxy signer specified by the delegation δ_O, can verify δ_O as follows:

a. $(w_O, R_O, \sigma_O) := \delta_O$. // δ_O is parsed as (w_O, R_O, σ_O).

b. **if** $\Pi_x(R_O) \notin \mathbb{Z}_q^* \vee \sigma_O \notin \mathbb{Z}_q^*$ **then return False**.

c. $h_1 \leftarrow \mathrm{H}_1(B_{Params} \| \mathrm{ID}_O)$; $h_2' \leftarrow \mathrm{H}_2(\mathrm{K}_O \| h_1)$.

d. $O_O \leftarrow \mathrm{P}_O + \mathrm{K}_O + [h_2'] P_{\mathrm{KGC}}$.

d. $h_2 \leftarrow \mathrm{H}_2(\mathrm{P}_O \| \mathrm{K}_O \| h_1)$; $h_3 \leftarrow \mathrm{H}_3(\mathrm{w}_O \| h_2)$.

e. $u_1 \leftarrow (h_3 / \sigma_O) \bmod q$; $u_2 \leftarrow (\mathbf{\Pi}_{\mathbf{x}}(R_O) / \sigma_O) \bmod q$.

f. **return** ($R_O \stackrel{?}{=} [u_1] G + [u_2] O_O$).

// A verifier accepts δ_O if VerifyDelegation returns **True**, or rejects otherwise.

(6) $\sigma_m \leftarrow\!\!\!\shortmid$ GenerateProxySign($Params, \mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \delta_O, \mathrm{ID}_P, \mathrm{P}_P, \mathrm{K}_P, \mathrm{s}_P, \mathrm{c}_P, m$): If the proxy signer P specified by the delegation δ_O accepts the delegation δ_O, he returns a proxy signature σ_m on the message m as follows:

a. $(\mathrm{w}_O, R_O, \sigma_O) := \delta_O$. // δ_O is parsed as $(\mathrm{w}_O, R_O, \sigma_O)$.

b. $h_1 \leftarrow \mathrm{H}_1(B_{Params} \| \mathrm{ID}_P)$; $h_2 \leftarrow \mathrm{H}_2(\mathrm{P}_O \| \mathrm{K}_O \| h_1)$; $h_3 \leftarrow \mathrm{H}_3(\mathrm{w}_O \| h_2)$.

c. $t_P \leftarrow (\sigma_O + (\mathrm{s}_P + \mathrm{c}_P) \cdot h_3) \bmod q$. // This is just a proxy signing key.

d. $h_4 \leftarrow \mathrm{H}_4(m \| \mathrm{w}_O \| h_2)$

e. $r_P \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $R_P \leftarrow [r_P] G$.

f. $u \leftarrow \mathbf{\Pi}_{\mathbf{x}}(R_P)$; $v \leftarrow r_P^{-1}(u \cdot t_P + h_4) \bmod q$.

g. $\sigma_m := (u, v)$; **return** σ_m.

(7) **True**/**False** $\leftarrow$ VerifyProxySign($Params, \mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \delta_O, \mathrm{ID}_P, \mathrm{P}_P, \mathrm{K}_P, m, \sigma_m$): Having all inputs, any verifier verifies the proxy signature σ_m on the message m as follows:

a. **if** VerifyDelegation($Params, \mathrm{ID}_O, \mathrm{K}_O, \mathrm{P}_O, \delta_O$)=**False then return False**.

b. $(\mathrm{w}_O, R_O, \sigma_O) := \delta_O$.

c. $h_1 \leftarrow \mathrm{H}_1(B_{Params} \| \mathrm{ID}_P)$; $h_2 \leftarrow \mathrm{H}_2(\mathrm{P}_P \| \mathrm{K}_P \| h_1)$.

d. $h_3 \leftarrow \mathrm{H}_3(\mathrm{w}_O \| h_2)$; $h_4 \leftarrow \mathrm{H}_4(m \| \mathrm{w}_O \| h_2)$; $h_2' \leftarrow \mathrm{H}_2(\mathrm{K}_P \| h_1)$.

e. $O_P \leftarrow [\sigma_O] G + [h_3](\mathrm{P}_P + \mathrm{K}_P + [h_2'] P_{\mathrm{KGC}})$.

f. $(u, v) := \sigma_m$. // σ_m is parsed as (u, v).

g. $v_1 \leftarrow (v^{-1} \cdot h_4) \bmod q$; $v_2 \leftarrow (v^{-1} \cdot u) \bmod q$.

h. $R'_P \leftarrow [v_1]G + [v_2]O_P$; $u' \leftarrow \Pi_{\mathbf{x}}(R'_P)$.

i. **return** $(u \stackrel{?}{=} u')$

// A verifier accepts σ_m if VerifyProxySign returns **True**, or rejects otherwise.

Now, we consider the correctness of the proposed scheme.

The correctness of the algorithm VerifyDelegation verifying a delegation is proved as follows.

$$
\begin{aligned}
&[u_1]G + [u_2]O_O \\
&= [(h_3/\sigma_O) \bmod q]G + [(\Pi_{\mathbf{x}}(R_O)/\sigma_O) \bmod q](\mathrm{P}_O + \mathrm{K}_O + [h'_2]P_{\mathrm{KGC}}) \\
&= [\sigma_O^{-1}(h_3 + \Pi_{\mathbf{x}}(R_O))(\mathrm{s}_O + k_O + h'_2 \cdot s)) \bmod q]G \\
&= [\sigma_O^{-1}(h_3 + \Pi_{\mathbf{x}}(R_O))(\mathrm{s}_O + \mathrm{c}_O)) \bmod q]G \\
&= [r_O]G = R_O .
\end{aligned}
$$

The correctness of the algorithm VerifyProxySign verifying a proxy signature is proved as follows.

$$
\begin{aligned}
R'_P &= [v_1]G + [v_2]O_P \\
&= [(v^{-1} \cdot h_4) \bmod q]G + [(v^{-1} \cdot u) \bmod q]([\sigma_O]G + [h_3](\mathrm{P}_P + \mathrm{K}_P + [h'_2]P_{\mathrm{KGC}}) \\
&= [v^{-1}(h_4 + u(\sigma_O + h_3(\mathrm{s}_P + k_P + h'_2 \cdot s))) \bmod q]G \\
&= [r_P \frac{h_4 + u(\sigma_O + h_3(\mathrm{s}_P + k_P + h'_2 \cdot s))}{h_4 + u(\sigma_O + (\mathrm{s}_P + \mathrm{c}_P) \cdot h_3)} \bmod q]G \\
&= [r_P \frac{h_4 + u(\sigma_O + h_3(\mathrm{s}_P + k_P + h'_2 \cdot s))}{h_4 + u(\sigma_O + (\mathrm{s}_P + k_P + s \cdot h'_2) \cdot h_3)} \bmod q]G \\
&= [r_P]G = R_P .
\end{aligned}
$$

Next, we compare the efficiency of our scheme with Zhang et al.'s scheme [26] in Table 1, though it is based on bilinear pairing. As mentioned in Section 1.1, there is no provably secure CLPS scheme without pairings in the literature.

In Table 1, $H_{\mathbf{T}}, P_{\mathbf{T}}, S_{\mathbf{T}}$ and $A_{\mathbf{T}}$ stand for the time of a map-to-point hash operation, a bilinear pairing operation, an elliptic curve scalar multiplication, and an elliptic curve point addition. According to [6, 12], $H_{\mathbf{T}}$ is rather logner than $P_{\mathbf{T}}$ by a narrow margin, $P_{\mathbf{T}}$ is

several times longer than $S_{\mathbf{T}}$, and $\mathbf{S}_{\mathbf{T}}$ is several tens to hundreds times longer than $A_{\mathbf{T}}$. We ignore costless operations such as one-way hash operation and operations on integers, which are much cheaper than the point addition. G_ℓ denotes the bit length of a point in the elliptic curve group $\mathbb{G}$ and Z_ℓ means the length of the integer q, i.e. $\log q$. G_ℓ is several times longer than Z_ℓ.

Table 1. The comparison of efficiency

	[26]	Our scheme
Generation of keys	$1H_{\mathbf{T}}+2S_{\mathbf{T}}$	$2S_{\mathbf{T}}$
Generation of delegation	$2H_{\mathbf{T}}+3S_{\mathbf{T}}+2A_{\mathbf{T}}$	$1S_{\mathbf{T}}$
Verification of delegation	$3H_{\mathbf{T}}+4P_{\mathbf{T}}$	$3S_{\mathbf{T}}+3A_{\mathbf{T}}$
Signing	$3H_{\mathbf{T}}+4S_{\mathbf{T}}+3A_{\mathbf{T}}$	$1S_{\mathbf{T}}$
Verification of signature	$5H_{\mathbf{T}}+5P_{\mathbf{T}}+2A_{\mathbf{T}}$	$5S_{\mathbf{T}}+4A_{\mathbf{T}}$
Key size	$2G_\ell+1Z_\ell$	$2G_\ell+2Z_\ell$
Signature size	$3G_\ell$	$2Z_\ell$

Note that using the binding technique in [1], we can easily convert our scheme to the one which achieves Girault's trust level 3 [8], by replacing $h_2' \leftarrow \mathrm{H}_2(K_U \parallel h_1)$ with $h_2' \leftarrow \mathrm{H}_2(\mathrm{P}_U \parallel K_U \parallel h_1)$ in the step (c) of GeneratePartialKeys, the step (c) of VerifyDelegation and the step (d) of VerifyProxySign.

5. Security Proofs

In this section, we prove the security of our scheme in the random oracle model (ROM), under the assumption that the elliptic curve semi-logarithm problem (ECSLP) is hard. we use the notations and symbols in Section 4 as it is.

Theorem. In the random oracle model, if there exists a PPT adversary $\mathcal{A}(\lambda)$ that wins EUF-CMA Game against our CLPS scheme with a non-negligible probability of success (in λ), then the semi-logarithm problem in the group $\mathbb{G}$ can be solved in polynomial time (in λ) with a non-negligible probability (in λ).

Proof. Suppose that $\mathcal{A}(\lambda)$ succeeds in EUF-CMA Game with a non-negligible probability $\varepsilon(\lambda)$ in time $t(\lambda)$ which is polynomial in λ, and in EUF-CMA Game, $\mathcal{A}(\lambda)$ respectively makes $N_{\mathrm{H}_1}, N_{\mathrm{H}_2}, N_{\mathrm{H}_3}, N_{\mathrm{H}_4}$ queries to H_1, H_2, H_3, H_4 modeled as random

oracles. Let N_U and N_{UKey} be respectively the number of queries to $\mathcal{O}_{\text{CreateUser}}$ that $\mathcal{A}(\lambda)$ makes in EUF-CMA Game and the maximum number of same queries on a fixed user identity among them, i.e. the maximum number of key pairs of a fixed user. These are all functions of λ and do not exceed $t(\lambda)$.

From this point, we considering Type I and Type II adversaries separately.

Firstly, when $\mathcal{A}(\lambda)$ is Type I adversary $\mathcal{A}_{\mathrm{I}}(\lambda)$, we construct an algorithm $\mathcal{C}_{\mathrm{I}}$ that uses $\mathcal{A}_{\mathrm{I}}(\lambda)$ to solve the semi-logarithm problem in the group $\mathbb{G}$ as follows:

Algorithm $\mathcal{C}_{\mathrm{I}}$:

Input: An instance of the semi-logarithm problem given by $q, \mathbb{G}, G, \mathbf{\Pi}_{\mathbf{x}}: \mathbb{G} \to \mathbb{Z}_q$ and $P := [\alpha]G$ where α is unknown.

Output: The solution (u, v) of the above instance such that $u = \mathbf{\Pi}_{\mathbf{x}}([v^{-1}](G + [u]\ P))$.

1. Select seven indices $0 < \mathcal{I}_O, \mathcal{I}_P \le N_U$, $0 < \mathcal{J}_O, \mathcal{J}_P \le N_{UKey}$, $0 < \mathcal{K}_O, \mathcal{K}_P \le N_{\mathrm{H}_3}$, $0 < \mathcal{L} \le N_{\mathrm{H}_4}$ and a boolean value $\mathcal{B} \in \{0,1\}$ randomly.

2. Simulate a challenger in EUF-CMA Game as follows:

Setup: Set $P_{\mathrm{KGC}} := P = [\alpha]G$, $B_{Params} := a \,\|\, b \,\|\, G \,\|\, P_{\mathrm{KGC}}$, $Params := \{a, b, p, q, G, P_{\mathrm{KGC}}\}$ and send $Params$ to $\mathcal{A}_{\mathrm{I}}(\lambda)$. Initialize the lists $\mathbf{P}$, $\mathbf{S}$, $\mathbf{R}$ and $\mathbf{D}$ with empty list ($\emptyset$) respectively, and prepare lists $\mathbf{H}_1$ and $\mathbf{H}_2$ of 2-tuples, lists $\mathbf{H}_3$ and $\mathbf{H}_4$ of 4-tuples and a list $\mathbf{L}$ of tuples in form of $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \mathrm{s}_U, \mathrm{c}_U)$. Choose nine values $h_{1O}^*, h_{1P}^* \in \{0,1\}^n$ and $h_{2O}^*, h_{2P}^*, h_{2O}'^*, h_{2P}'^*, h_3^*, h_4^*, \sigma^* \in \mathbb{Z}_q^*$ randomly such that $h_{2O}^*, h_{2P}^*, h_{2O}'^*, h_{2P}'^*$ are all different and if $\mathcal{I}_O \neq \mathcal{I}_P$ then $h_{1O}^* \neq h_{1P}^*$.

Attack: Answer $\mathcal{A}_{\mathrm{I}}(\lambda)$'s queries to oracles as follows:

- $\mathrm{H}_1(\mathrm{x})$: If there exists $h_1 \in \{0,1\}^n$ such that $(\mathrm{x}, h_1) \in \mathbf{H}_1$ then return h_1. If $|\mathbf{H}_1| = \mathcal{I}_O - 1$ then put (x, h_{1O}^*) in $\mathbf{H_1}$ and return h_{1O}^*. If $|\mathbf{H}_1| = \mathcal{I}_P - 1$ then put (x, h_{1P}^*) in $\mathbf{H_1}$ and return h_{1P}^*. Otherwise, choose $h_1 \in \{0,1\}^n$ randomly. If $h_1 = h_{1O}^*$ or $h_1 = h_{1P}^*$ then terminate the game. (This case is called *Event* E_1.) Otherwise, put

(x, h_1) in $\mathbf{H_1}$ and return h_1.

- H_2(x): If there exists $h_2 \in \mathbb{Z}_q^*$ such that (x, h_2) $\in \mathbf{H}_2$ then return h_2. Otherwise, choose $h_2 \in \mathbb{Z}_q^*$ randomly. If $h_2 \in \{ h_{2O}^*, h_{2P}^*, h_{2O}'^*, h_{2P}'^* \}$ then terminate the game. (This case is called *Event* E_2.) Otherwise, put (x, h_2) in $\mathbf{H}_2$ and return h_2.
- H_3(w ‖ h): If there exists $h_3 \in \mathbb{Z}_q^*$ such that (w ‖ h, h_3,*,*) $\in \mathbf{H}_3$ then return h_3 (where * means for an arbitrary value). Otherwise, respond differently according to the following cases:

  – If $\mathcal{B}$ =0 and $h = h_{2O}^*$, then consider the following cases:

  (1) If $|\mathbf{H}_3| = \mathcal{K}_O - 1$ then put (w ‖ h, h_3^*, ⊥, ⊥) in $\mathbf{H_1}$ and return h_3^*.

  (2) Else, follow the next steps:

  a. $(u, v) \leftarrow_{\mathrm{R}} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.

  b. $R \leftarrow [u]G + [v][h_3^*]P_{\mathrm{KGC}}$.

  c. $\sigma \leftarrow (\mathbf{\Pi}_{\mathbf{x}}(R)/v) \bmod q$; $h_3 \leftarrow (u \cdot \mathbf{\Pi}_{\mathbf{x}}(R)/v) \bmod q$.

  d. put (w ‖ h, h_3, R, σ) in $\mathbf{H}_3$ and return h_3.

  – If $\mathcal{B}$ =1 and $h = h_{2P}^*$, then consider the following cases:

  (1) If $|\mathbf{H}_3| = \mathcal{K}_P - 1$, then put (w ‖ h_{2P}^*, h_3^*, ⊥, ⊥) in $\mathbf{H_1}$ and return h_3^*.

  (2) Else, follow the next steps:

  a. $(u, v) \leftarrow_{\mathrm{R}} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.

  b. $R \leftarrow [(u - v \cdot \sigma^* / h_3^*) \bmod q]G + [v][h_4^* / h_3^*]P_{\mathrm{KGC}}$.

  c. $\sigma \leftarrow (\mathbf{\Pi}_{\mathbf{x}}(R)/v) \bmod q$; $h_3 \leftarrow (u \cdot \mathbf{\Pi}_{\mathbf{x}}(R)/v) \bmod q$.

  d. put (w ‖ h, h_3, R, σ) in $\mathbf{H}_3$ and return h_3.

  – If $\mathcal{B}$ =1, $h = h_{2O}^*$ and $|\mathbf{H}_3| = \mathcal{K}_O - 1$, then follow the next steps:

  a. Search $\mathbf{H}_2$ for the item (P_U ‖ K_U ‖ h_{1O}^*, h_{2O}^*) indexed by h_{2O}^* (it must be existed uniquely!) to get P_U and K_U.

b. Search $\mathbf{H_1}$ for the item ($B_{Params} \| \mathrm{ID}_U, h_{1O}^*$) indexed by h_{1O}^* to get ID_U.

c. Search $\mathbf{L}$ for the item ($\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \mathrm{s}_U, \mathrm{c}_U$) indexed by ($\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U$) to get s_U and c_U.

d. $r_U \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $R_U \leftarrow [r_U]G$; $h_3 \leftarrow (\sigma^* \cdot r_U - \mathbf{\Pi}_{\mathbf{x}}(R_U) \cdot (\mathrm{s}_U + \mathrm{c}_U)) \bmod q$.

e. Put ($\mathrm{w} \| h, h_3, R_U, \sigma^*$) in $\mathbf{H}_3$ and return h_3.

– Else, choose $h_3 \in \mathbb{Z}_q^*$ randomly. If $h_3 = h_3^*$ then terminate the game. (This case is called *Event* E_3.) Otherwise, put ($\mathrm{w} \| h$, h_3, $\perp$, $\perp$) in $\mathbf{H}_3$ and return h_3.

- $\mathrm{H}_4(m \| \mathrm{w} \| h)$: If there exists $h_4 \in \mathbb{Z}_q^*$ such that ($m \| \mathrm{w} \| h, h_4, *, *) \in \mathbf{H}_4$ then return h_4. Otherwise, respond differently according to the following cases:

  – If $\mathcal{B}$=0, or $\mathcal{B}$=1 and $h \neq h_{2P}^*$, then choose $h_4 \in \mathbb{Z}_q^*$ randomly. If $h_4 = h_4^*$ then terminate the game. (This case is called *Event* E_4.) Otherwise, put ($m \| \mathrm{w} \| h$, h_4, $\perp$, $\perp$) in $\mathbf{H}_4$ and return h_4.

  – If $\mathcal{B}$=1, $h = h_{2P}^*$ and $|\mathbf{H}_4| = \mathcal{L} - 1$, then put ($m \| \mathrm{w} \| h, h_4^*, \perp, \perp$) in $\mathbf{H}_4$ and return h_4^*.

  – Else, follow the next steps:

    a. $(u, v) \leftarrow_{\mathrm{R}} \mathbb{Z}_q^* \times \mathbb{Z}_q^*$.

    b. $u' \leftarrow \mathbf{\Pi}_{\mathbf{x}}([u]G + [v][h_4^*]P_{\mathrm{KGC}})$; $v' \leftarrow (u'/v) \bmod q$.

    c. $h_4 \leftarrow (u \cdot u'/v) \bmod q$.

    d. Put ($m \| \mathrm{w} \| h, h_4, u', v'$) in $\mathbf{H}_4$ and return h_4.

- $\mathcal{O}_{\mathrm{CreateUser}}(\mathrm{ID}_U)$: Let $h_1 \leftarrow \mathrm{H}_1(B_{Params} \| \mathrm{ID}_U)$ and respond differently in the following cases:

  – If $\mathcal{B}$=0, $h_1 = h_{1O}^*$ and the current query is the $\mathcal{J}_O$-th one on ID_U, then follow the next steps:

  a. If $\mathcal{J}_O$=1 then choose $\mathrm{c}_U \in \mathbb{Z}_q^*$ randomly, let $\mathrm{K}_U \leftarrow [\mathrm{c}_U]G - [h_{2O}'^*]P_{\mathrm{KGC}}$

and put $(\mathrm{K}_U \| h_1, h_{2O}'^*)$ in $\mathbf{H}_2$. Otherwise, search $\mathbf{L}$ for the latest item $(\mathrm{ID}_U$, $*, \mathrm{K}_U, *, \mathrm{c}_U)$ indexed by ID_U to get K_U and c_U.

b. $\mathrm{P}_U \leftarrow [(h_3^* - h_{2O}'^*) \bmod q] P_{\mathrm{KGC}} - \mathrm{K}_U$.

c. Put $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \perp, \mathrm{c}_U)$ in $\mathbf{L}$, put $(\mathrm{P}_U \| \mathrm{K}_U \| h_1, h_{2O}^*)$ in $\mathbf{H}_2$ and return $(\mathrm{P}_U, \mathrm{K}_U)$.

– If $\mathcal{B}=1$, $h_1 = h_{1P}^*$ and the current query is the $\mathcal{J}_P$-th one on ID_U, then follow the next steps:

a. If $\mathcal{J}_P = 1$ then choose $\mathrm{c}_U \in \mathbb{Z}_q^*$ randomly, let $\mathrm{K}_U \leftarrow [\mathrm{c}_U] G - [h_{2P}'^*] P_{\mathrm{KGC}}$ and put $(\mathrm{K}_U \| h_1, h_{2P}'^*)$ in $\mathbf{H}_2$. Otherwise, search $\mathbf{L}$ for the latest item $(\mathrm{ID}_U$, $*, \mathrm{K}_U, *, \mathrm{c}_U)$ indexed by ID_U to get K_U and c_U.

b. $\mathrm{P}_U \leftarrow [(h_4^* / h_3^* - h_{2P}'^*) \bmod q] P_{\mathrm{KGC}} - [(\sigma^* / h_3^*) \bmod q] G - \mathrm{K}_U$.

c. Put $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \perp, \mathrm{c}_U)$ in $\mathbf{L}$, put $(\mathrm{P}_U \| \mathrm{K}_U \| h_1, h_{2P}^*)$ in $\mathbf{H}_2$ and return $(\mathrm{P}_U, \mathrm{K}_U)$.

– Else, follow the next steps:

a. $\mathrm{s}_U \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $\mathrm{P}_U \leftarrow [\mathrm{s}_U] G$.

b. $\mathrm{c}_U \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $h_2' \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $\mathrm{K}_U \leftarrow [\mathrm{c}_U] G - [h_2'] P_{\mathrm{KGC}}$.

c. If there exists $h \in \mathbb{Z}_q^*$ such that $(\mathrm{K}_U \| h_1, h) \in \mathbf{H}_2$ and $h \neq h_2'$ then terminate the game. (This case is called *Event* E_{21}.)

d. If $(K_U \| h_1, h_2') \notin \mathbf{H}_2$ then check whether h_2' is in $\{h_{2O}^*, h_{2P}^*\}$. If it is, terminate the game. (This case is called *Event* E_{22}.) Otherwise, put $(K_U \| h_1, h_2')$ in $\mathbf{H}_2$.

e. Put $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \mathrm{s}_U, \mathrm{c}_U)$ in $\mathbf{L}$ and return $(\mathrm{P}_U, \mathrm{K}_U)$.

- $\mathcal{O}_{\mathrm{GetSecretKey}}(\mathrm{ID}_U, \mathrm{P}_U)$: Search $\mathbf{L}$ for the item $(\mathrm{ID}_U, \mathrm{P}_U, *, \mathrm{s}_U, *)$ indexed by $(\mathrm{ID}_U, \mathrm{P}_U)$ to get s_U. If the search succeeds, put $(\mathrm{ID}_U, \mathrm{P}_U)$ in $\mathbf{S}$ and return s_U.

Otherwise, return $\perp$.

- $\mathcal{O}_{\text{GetPartialKey}}(\mathrm{ID}_U, \mathrm{K}_U)$: Search $\mathbf{L}$ for the item $(\mathrm{ID}_U, *, \mathrm{K}_U, *, \mathrm{c}_U)$ indexed by $(\mathrm{ID}_U, \mathrm{K}_U)$ to get c_U. If the search succeeds, put $(\mathrm{ID}_U, \mathrm{K}_U)$ in $\mathbf{P}$ and return c_U. Otherwise, return $\perp$.
- $\mathcal{O}_{\text{GetPublicKey}}(\mathrm{ID}_U)$: Search $\mathbf{L}$ for the latest item $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, *, *)$ indexed by ID_U to get P_U and K_U. If the search succeeds, return $(\mathrm{P}_U, \mathrm{K}_U)$. Otherwise, return $\perp$.
- $\mathcal{O}_{\text{ReplaceUserKeys}}(\mathrm{ID}_U, \mathrm{P}'_U, \mathrm{s}'_U)$: Search $\mathbf{L}$ for the latest item $(\mathrm{ID}_U, *, \mathrm{K}_U, *, \mathrm{c}_U)$ indexed by ID_U to get K_U and c_U. If the search succeeds, put $(\mathrm{ID}_U, \mathrm{P}'_U, \mathrm{K}_U, \mathrm{s}'_U, \mathrm{c}_U)$ in $\mathbf{L}$, put $(\mathrm{ID}_U, \mathrm{P}'_U)$ in $\mathbf{R}$ and return "OK". Otherwise, return $\perp$.
- $\mathcal{O}_{\text{GetDelegation}}(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{w}_O)$: Search $\mathbf{D}$ for the item $(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{w}_O, \delta_O)$ indexed by $(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{w}_O)$ to get δ_O. If the search succeeds, return δ_O. Otherwise, follow the next steps:
  a. Search $\mathbf{H_1}$ for the item $(B_{Params} \| \mathrm{ID}_O, h_1)$ indexed by $B_{Params} \| \mathrm{ID}_O$ to get h_1. If the search does not succeed, return $\perp$.
  b. Search $\mathbf{H}_2$ for the item $(\mathrm{P}_O \| \mathrm{K}_O \| h_1, h_2)$ indexed by $\mathrm{P}_O \| \mathrm{K}_O \| h_1$ to get h_2. If the search does not succeed, return $\perp$.
  c. $h_O \leftarrow \mathrm{H}_3(\mathrm{w}_O \| h_2)$.
  d. Search $\mathbf{H}_3$ for the item $(\mathrm{w}_O \| h_2, h_O, R_O, \sigma_O)$ indexed by $(\mathrm{w}_O \| h_2, h_O)$ to get R_O and σ_O. If $R_O \neq \perp$ and $\sigma_O \neq \perp$, then let $\delta_O := (\mathrm{w}_O, R_O, \sigma_O)$, put $(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{w}_O, \delta_O)$ in $\mathbf{D}$ and return δ_O. Otherwise, continue.
  e. Search $\mathbf{L}$ for the item $(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{s}_O, \mathrm{c}_O)$ indexed by $(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O)$ to get s_O and c_O. If the search succeeds, $\mathrm{s}_O \neq \perp$ and $\mathrm{c}_O \neq \perp$, then continue. Otherwise, return $\perp$.
  f. $r_O \leftarrow_R \mathbb{Z}_q^*$; $R_O \leftarrow [r_O]G$; $\sigma_O \leftarrow r_O^{-1}(\Pi_{\mathbf{x}}(R_O) \cdot (\mathrm{s}_O + \mathrm{c}_O) + h_O) \bmod q$.
  g. Let $\delta_O := (\mathrm{w}_O, R_O, \sigma_O)$, put $(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \mathrm{w}_O, \delta_O)$ in $\mathbf{D}$ and return δ_O.

- $\mathcal{O}_{\text{GetProxySign}}(\mathrm{ID}_O, \mathrm{P}_O, \mathrm{K}_O, \delta_O, \mathrm{ID}_P, \mathrm{P}_P, \mathrm{K}_P, m)$: Parse δ_O as $(\mathrm{w}_O, R_O, \sigma_O)$ and check whether the delegation δ_O is valid and all input parameters comform to w_O. If not, return $\perp$. If the checks are passed, follow the next steps:
  a. Search $\mathbf{L}$ for the item $(\mathrm{ID}_P, \mathrm{P}_P, \mathrm{K}_P, \mathrm{s}_P, \mathrm{c}_P)$ indexed by $(\mathrm{ID}_P, \mathrm{P}_P, \mathrm{K}_P)$ to get s_P and c_P. If the search does not succeed, return $\perp$.
  b. Search $\mathbf{H_1}$ for the item $(B_{Params} \| \mathrm{ID}_P, h_1)$ indexed by $B_{Params} \| \mathrm{ID}_P$ to get h_1. If the search does not succeed, return $\perp$.
  c. $h_2 \leftarrow \mathrm{H}_2(\mathrm{P}_P \| \mathrm{K}_P \| h_1)$; $h_3 \leftarrow \mathrm{H}_3(\mathrm{w}_O \| h_2)$; $h_4 \leftarrow \mathrm{H}_4(m \| \mathrm{w}_O \| h_2)$.
  d. Consider the following cases:
  - If $x_P = \perp$, then search $\mathbf{H}_4$ for the item $(m \| \mathrm{w}_O \| h_2, h_4, u, v)$ indexed by $(m \| \mathrm{w}_O \| h_2, h_4)$ to get u and v. If $u \neq \perp$ and $v \neq \perp$, return (u, v). Otherwise, terminate the game. (This case is called *Event* S.)
  - Else (i.e. if $x_P \in \mathbb{Z}_q^*$), let $t_P \leftarrow (\sigma_O + (\mathrm{s}_P + \mathrm{c}_P) \cdot h_3) \bmod q$, $r_P \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$, $R_P \leftarrow [r_P]G$, $u \leftarrow \mathbf{\Pi}_{\mathbf{x}}(R_P)$, $v \leftarrow r_P^{-1}(u \cdot t_P + h_4) \bmod q$, and return (u, v).

Forgery: Wait until $\mathcal{A}_1(\lambda)$ outputs a tuple $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*, \sigma_{m^*})$ as the final result of the game.

3. Check $\mathcal{A}_1(\lambda)$'s final result and output the solution of the above instance (i.e. the input) as follows:
   (1) If VerifyProxySign($Params, \mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*, \sigma_{m^*}$)=**False,** then reject the final result and output $\perp$.
   (2) Parse δ_O^* as $(\mathrm{w}_O^*, R_O^*, \sigma_O^*)$. If the following boolean expression is true, then reject the final result and output $\perp$.

$$(((\mathrm{ID}_O^*, \mathrm{K}_O^*) \in \mathbf{P} \wedge (\mathrm{ID}_O^*, \mathrm{P}_O^*) \in \mathbf{S} \cup \mathbf{R}) \vee (\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \mathrm{w}_O^*, \delta_O^*) \in \mathbf{D})$$
$$\wedge (\mathrm{ID}_P^*, \mathrm{K}_P^*) \in \mathbf{P} \wedge (\mathrm{ID}_P^*, \mathrm{P}_P^*) \in \mathbf{S} \cup \mathbf{R}.$$

   (3) If the following boolean expression is true, then reject the final result and return $\perp$.

$$h_{1O}^* \neq \mathrm{H}_1(B_{Params} \| \mathrm{ID}_O^*) \vee h_{1P}^* \neq \mathrm{H}_1(B_{Params} \| \mathrm{ID}_P^*) \vee$$

$$h_{2O}^* \neq \mathrm{H}_2(\mathrm{P}_O^* \| \mathrm{K}_O^* \| h_{1O}^*) \vee h_{2P}^* \neq \mathrm{H}_2(\mathrm{P}_P^* \| \mathrm{K}_P^* \| h_{1P}^*) \vee$$

$$(h_3^* \neq \mathrm{H}_3(\mathrm{w}_O^* \| h_{2O}^*) \wedge h_3^* \neq \mathrm{H}_3(\mathrm{w}_O^* \| h_{2P}^*)) \vee h_4^* \neq \mathrm{H}_4(m^* \| \mathrm{w}_O^* \| h_{2P}^*).$$

(4) If $\mathcal{B}$=0, then let $u^* \leftarrow \mathbf{\Pi}_{\mathbf{x}}(R_O^*)$, $v^* \leftarrow (\sigma_O^* / h_3^*) \bmod q$. Otherwise (i.e. if $\mathcal{B}$=1), Parse σ_{m^*} as (u, v) and let $u^* \leftarrow u$, $v^* \leftarrow (v / h_4^*) \bmod q$.

(5) Output (u^*, v^*). (The end of the algoritm $\mathcal{C}_{\mathrm{I}}$)

Now, we analyze the time complexity of $\mathcal{C}_{\mathrm{I}}$. The time complexity of $\mathcal{C}_{\mathrm{I}}$ is dominated by the scalar multiplications and additions in $\mathbb{G}$, operations in $\mathbb{Z}_q^*$ and searches on lists which are performed in the simulations of oracles. All operands for the above operations in $\mathbb{G}$ and $\mathbb{Z}_q^*$ have at most a polynomial size in λ and we can assume that the computational cost of any operation in $\mathbb{G}$ and $\mathbb{Z}_q^*$ is less than some polynomial $p(\lambda)$. On the other hand, the sizes of all lists in oracle simulations of $\mathcal{C}_{\mathrm{I}}$ do not go beyond the number of all queries by $\mathcal{A}_{\mathrm{I}}(\lambda)$, which is not larger than $t(\lambda)$. It results that the time of a search on any list in $\mathcal{C}_{\mathrm{I}}$ is also not longer than $t(\lambda)$. Therefore, the cost of answering an oracle query in $\mathcal{C}_{\mathrm{I}}$ is also less than $O(p(\lambda)+t(\lambda))$, and we conclude that the total time complexity of $\mathcal{C}_{\mathrm{I}}$ is $O(t(\lambda)p(\lambda)+t^2(\lambda))$ which is also polynomial in λ.

Next, we show that if $\mathcal{C}_{\mathrm{I}}$ does not terminate the game incompletely, then $\mathcal{A}_{\mathrm{I}}(\lambda)$'s view in the simulated game with $\mathcal{C}_{\mathrm{I}}$ is indistinguishable from its view in the real attack and when $\mathcal{C}_{\mathrm{I}}$'s output is not $\perp$, it is the solution for the SLP instance given as input of $\mathcal{C}_{\mathrm{I}}$.

In the random oracle model, the outputs of H_1, H_2, H_3 and H_4 are all sampled uniformly at random. In the simulation of $\mathcal{O}_{\mathrm{CreateUser}}$, a partial key pair $(\mathrm{K}_U, \mathrm{c}_U)$ of U is created from uniformly distributed random values c_U and h_2' to satisfy the equations $\mathrm{K}_U = [\mathrm{c}_U]G - [h_2']P_{\mathrm{KGC}}$ and $h_2' = \mathrm{H}_2(\mathrm{K}_U \| \mathrm{H}_1(B_{Params} \| \mathrm{ID}_U))$ which are equivalent to the verification equation for validity of a partial key pair. From the true randomness of hash values, the distribution of $(\mathrm{K}_U, \mathrm{c}_U)$ in the simulation is identical to the one of values taken

by $\mathcal{A}_{\mathrm{I}}(\lambda)$ in the real attack. If $\mathrm{ID}_U \notin \{\mathrm{ID}_O^*, \mathrm{ID}_P^*\}$ or $\mathrm{P}_U \notin \{\mathrm{P}_O^*, \mathrm{P}_P^*\}$, U 's public key P_U and secret key s_U are generated randomly and satisfy the equation $\mathrm{P}_U = [\mathrm{s}_U]G$. Hence, the key pair is valid and the full item $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \mathrm{s}_U, \mathrm{c}_U)$ is put in $\mathbf{L}$. In this case, the simulations of other oracles follow to the definition of our scheme.

If $\mathcal{B}$=0, then the full public key $(\mathrm{P}_O^*, \mathrm{K}_O^*)$ given by $\mathcal{C}_{\mathrm{I}}$ to $\mathcal{A}_{\mathrm{I}}(\lambda)$ for the targeted origi nal signer ID_O^* and $\mathcal{A}_{\mathrm{I}}(\lambda)$'s final result satisfy the following equations:

$$\mathrm{P}_O^* + \mathrm{K}_O^* = [(h_3^* - h_{2O}'^*) \bmod q] P_{\mathrm{KGC}},$$

$$\mathsf{VerifyDelegation}(Params, \mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*) = \mathbf{True},$$

$$h_{1O}^* = \mathrm{H}_1(B_{Params} \| \mathrm{ID}_O^*),\ h_{2O}'^* = \mathrm{H}_2(\mathrm{K}_O^* \| h_{1O}^*),$$

$$h_{2O}^* = \mathrm{H}_2(\mathrm{P}_O^* \| \mathrm{K}_O^* \| h_{1O}^*),\ h_3^* = \mathrm{H}_3(\mathrm{w}_O^* \| h_{2O}^*).$$

From the definition of $\mathsf{VerifyDelegation}$ and the above equations, we have

$$O_O = \mathrm{P}_O^* + \mathrm{K}_O^* + [h_{2O}'^*] P_{\mathrm{KGC}} = [h_3^*] P_{\mathrm{KGC}},$$

$$u_1 = (h_3^* / \sigma_O^*) \bmod q,\ u_2 = (\mathbf{\Pi}_{\mathbf{x}}(R_O^*) / \sigma_O^*) \bmod q,$$

$$\mathbf{\Pi}_{\mathbf{x}}(R_O^*) = \mathbf{\Pi}_{\mathbf{x}}([u_1]G + [u_2]O_O) = \mathbf{\Pi}_{\mathbf{x}}([u_1]G + [u_2 \cdot h_3^*] P_{\mathrm{KGC}}).$$

Since $\mathcal{C}_{\mathrm{I}}$'s output is $u^* = \mathbf{\Pi}_{\mathbf{x}}(R_O^*),\ v^* = (\sigma_O^* / h_3^*) \bmod q$ in this case, we have

$$u^* = \mathbf{\Pi}_{\mathbf{x}}([(v^*)^{-1}](G + [u^*] P_{\mathrm{KGC}})).$$

That is, (u^*, v^*) is the solution for the SLP instance given as input. In this case, the item $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \perp, \mathrm{c}_O^*)$ is put in $\mathbf{L}$. When $\mathrm{w}_O \neq \mathrm{w}_O^*$, $\mathcal{C}_{\mathrm{I}}$'s response $(\mathrm{w}_O, R, \sigma)$ to the oracle query $\mathcal{O}_{\mathrm{GetDelegation}}(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{w}_O)$ is derived from the item $(\mathrm{w}_O \| h_{2O}^*, h_3, R, \sigma)$ in $\mathbf{H}_3$ and satisfies the following equations.

$$R = [u]G + [v][h_3] P_{\mathrm{KGC}},$$

$$\sigma = (\mathbf{\Pi}_{\mathbf{x}}(R) / v) \bmod q,$$

$$h_3 = (u \cdot \mathbf{\Pi}_{\mathbf{x}}(R) / v) = \mathrm{H}_3(\mathrm{w}_O \| h_{2O}^*),$$

where $u, v \in \mathbb{Z}_q^*$ are choosed randomly. From these, it follows that $(\mathrm{w}_O, R, \sigma)$ is valid and

the distribution of $(\mathrm{w}_O, R, \sigma)$ in the simulation is identical to the one of values taken by $\mathcal{A}_{\mathrm{I}}(\lambda)$ in the real attack.

If $\mathcal{B}$=1, then the full public key $(\mathrm{P}_P^*, \mathrm{K}_P^*)$ given by $\mathcal{C}_{\mathrm{I}}$ to $\mathcal{A}_{\mathrm{I}}(\lambda)$ for the targeted proxy signer ID_P^* and $\mathcal{A}_{\mathrm{I}}(\lambda)$'s final result satisfy the following equations:

$$\mathrm{P}_O^* + \mathrm{K}_O^* = [(h_4^* / h_3^* - h_{2P}'^*) \bmod q] P_{\mathrm{KGC}} - [(\sigma^* / h_3^*) \bmod q] G,$$

$$\mathsf{VerifyProxySign}(Params, \mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*, \sigma_{m^*}) = \textbf{True},$$

$$h_{1P}^* = \mathrm{H}_1(B_{Params} \| \mathrm{ID}_P^*),\ h_{2P}'^* = \mathrm{H}_2(K_P^* \| h_{1P}^*),\ h_{2P}^* = \mathrm{H}_2(\mathrm{P}_P^* \| K_P^* \| h_{1P}^*),$$

$$h_3^* = \mathrm{H}_3(\mathrm{w}_O^* \| h_{2P}^*), h_4^* = \mathrm{H}_4(m^* \| \mathrm{w}_O^* \| h_{2P}^*),$$

$$(\mathrm{w}_O^*, R_O^*, \sigma^*) = \mathcal{O}_{\mathrm{GetDelegation}}(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{w}_O^*),$$

From the definition of $\mathsf{VerifyProxySign}$ and the above equations, we have

$$\begin{aligned} O_P &= [\sigma^*] G + [h_3^*](\mathrm{P}_P^* + K_P^* + [h_{2P}'^*] P_{\mathrm{KGC}}) \\ &= [\sigma^*] G + [h_3^*]((h_4^* / h_3^*) \bmod q] P_{\mathrm{KGC}} - [(\sigma^* / h_3^*) \bmod q] G) \\ &= [h_4^*] P_{\mathrm{KGC}}, \end{aligned}$$

$$\sigma_{m^*} = (u, v),$$

$$v_1 = (v^{-1} \cdot h_4^*) \bmod q,\ v_2 = (v^{-1} \cdot u) \bmod q,$$

$$u = \Pi_{\mathbf{x}}([v_1] G + [v_2] O_P) = \Pi_{\mathbf{x}}([v_1] G + [v_2 \cdot h_4^*] P_{\mathrm{KGC}}).$$

Since $\mathcal{C}_{\mathrm{I}}$'s output is $u^* = u$, $v^* = (v / h_4^*) \bmod q$ in this case, we have

$$u^* = \Pi_{\mathbf{x}}([(v^*)^{-1}](G + [u^*] P_{\mathrm{KGC}})).$$

That is, (u^*, v^*) is the solution for the SLP instance given as input. In this case, the item $(\mathrm{ID}_P^*, \mathrm{P}_P^*, K_P^*, \perp, \mathrm{c}_P^*)$ is put in $\mathbf{L}$. When $\mathrm{w}_O \neq \mathrm{w}_O^*$, $\mathcal{C}_{\mathrm{I}}$'s response $(\mathrm{w}_O, R, \sigma)$ to the oracle query $\mathcal{O}_{\mathrm{GetDelegation}}(\mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{w}_O)$ is derived from the item $(\mathrm{w}_O \| h_{2P}^*, h_3, R, \sigma)$ in $\mathbf{H}_3$ and satisfies the following equations.

$$R = [(u - v \cdot \sigma^* / h_3^*) \bmod q] G + [v][h_4^* / h_3^*] P_{\mathrm{KGC}},$$

$$\sigma = (\Pi_{\mathbf{x}}(R) / v) \bmod q,$$

$$h_3 = (u \cdot \mathbf{P}_x(R)/v) = \mathrm{H}_3(\mathrm{w}_O \| h_{2P}^*),$$

where $u, v \in \mathbb{Z}_q^*$ are choosed randomly. From these, it follows that $(\mathrm{w}_O, R, \sigma)$ is valid and the distribution of $(\mathrm{w}_O, R, \sigma)$ in the simulation is identical to the one of values taken by $\mathcal{A}_{\mathrm{I}}(\lambda)$ in the real attack.

Now, we analyze the probability for $\mathcal{C}_{\mathrm{I}}$ to output the solution of the given instance of SLP.

We know that $\mathcal{C}_{\mathrm{I}}$ never rejects the final result of $\mathcal{A}_{\mathrm{I}}(\lambda)$ in its step 3 (1) and 3 (2) if $\mathcal{A}_{\mathrm{I}}(\lambda)$ succeeds the simulated game, i.e. $\mathcal{A}_{\mathrm{I}}(\lambda)$ forges a valid delegation or proxy signature. And if $\mathcal{C}_{\mathrm{I}}$'s selections of seven indices $\mathcal{I}_O, \mathcal{I}_P, \mathcal{J}_O, \mathcal{J}_P, \mathcal{K}_O, \mathcal{K}_P, \mathcal{L}$ and a Boolean value $\mathcal{B}$ are correct, i.e. for the final result $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*, \sigma_{m^*})$ of $\mathcal{A}_{\mathrm{I}}(\lambda)$, the target original signer with ID_O^* is the $\mathcal{I}_O$-th user, the target proxy signer with ID_P^* is the $\mathcal{I}_P$-th user, $(\mathrm{P}_O^*, \mathrm{K}_O^*)$ is the $\mathcal{J}_O$-th full public key of the target original signer, $(\mathrm{P}_P^*, \mathrm{K}_P^*)$ is the $\mathcal{J}_P$-th full public key of the target proxy signer, $\mathrm{H}_3(\mathrm{w}_O^* \| h_{2O}^*)$ (where w_O^* is the warrant in the delegation δ_O^* and $h_{2O}^* = \mathrm{H}_2(\mathrm{P}_O^* \| \mathrm{K}_O^* \| \mathrm{H}_1(B_{Params} \| \mathrm{ID}_O^*))$) is the $\mathcal{K}_O$-th new query to H_3, $\mathrm{H}_3(\mathrm{w}_O^* \| h_{2P}^*)$ (where $h_{2P}^* = \mathrm{H}_2(\mathrm{P}_P^* \| \mathrm{K}_P^* \| \mathrm{H}_1(B_{Params} \| \mathrm{ID}_P^*))$) is the $\mathcal{K}_P$-th new query to H_3, $\mathrm{H}_4(m^* \| \mathrm{w}_O^* \| h_{2P}^*)$ is the $\mathcal{L}$-th new query to H_4, $\mathcal{B}$ =0 in the case $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \mathrm{w}_O^*, \delta_O^*) \notin \mathbf{D}$ and $\mathcal{B}$ =1 in the case $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \mathrm{w}_O^*, \delta_O^*) \in \mathbf{D}$, $\mathcal{C}_{\mathrm{I}}$ never rejects the final result of $\mathcal{A}_{\mathrm{I}}(\lambda)$ in its step 3 (3) and Event S never occurs. The probability that $\mathcal{A}_{\mathrm{I}}(\lambda)$ succeeds the simulated game and $\mathcal{C}_{\mathrm{I}}$'s selections of $\mathcal{I}_O, \mathcal{I}_P, \mathcal{J}_O, \mathcal{J}_P, \mathcal{K}_O, \mathcal{K}_P, \mathcal{L}$ and $\mathcal{B}$ are correct is at least $\varepsilon(\lambda)/(N_U^2 \cdot N_{UKey}^2 \cdot N_{\mathrm{H}_3}^2 \cdot N_{\mathrm{H}_4}^2 \cdot 2)$.

The probabilities of Events $\mathrm{E}_1, \mathrm{E}_2$ and E_3 in the running of $\mathcal{C}_{\mathrm{I}}$ is at most $2N_{\mathrm{H}_1}/2^n$, $4N_{\mathrm{H}_2}/q$ and N_{H_3}/q, respectively. Since Event E_4 occurs only when $\mathcal{B}$ =1, the probability of E_4 is at most $N_{\mathrm{H}_4}/(2q)$. Since Events E_{21} and E_{22} are mutual exclusive, the probability of Event $\mathrm{E}_{21} \bigcup \mathrm{E}_{22}$ is at most $N_U \, N_{UKey}(1/q^2 + 2/q)$.

Therefore, the probability that $\mathcal{C}_{\mathrm{I}}$ outputs the solution of the given instance of SLP is at

least the following expression:

$$\varepsilon(\lambda)/(N_U^2 \cdot N_{UKey}^2 \cdot N_{\mathrm{H}_3}^2 \cdot N_{\mathrm{H}_4}^2 \cdot 2)(1-2N_{\mathrm{H}_1}/2^n)(1-4N_{\mathrm{H}_2}/q)(1-N_{\mathrm{H}_3}/q)$$
$$(1-N_{\mathrm{H}_4}/(2q))(1-N_U \cdot N_{UKey}(1/q^2+2/q)).$$

Since $N_{\mathrm{H}_1}, N_{\mathrm{H}_2}, N_{\mathrm{H}_3}, N_{\mathrm{H}_4}, N_U, N_{UKey}$ and n are polynomial in λ, q is exponential in λ, $\varepsilon(\lambda)$ is non-negligible, the above expression is also non-negligible.

Secondly, we construct an algorithm $\mathcal{C}_{\mathrm{II}}$ that uses $\mathcal{A}_{\mathrm{II}}(\lambda)$ to solve the semi-logarithm problem in the group $\mathbb{G}$ as follows:

Algorithm $\mathcal{C}_{\mathrm{II}}$:

Input/*Output*: (Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$.)

1. (Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$.)

2. Simulate a challenger in EUF-CMA Game as follows:

Setup: Choose $s \in \mathbb{Z}_q^*$ randomly, set $P_{\mathrm{KGC}} \leftarrow [s]G$, $B_{Params} := a \parallel b \parallel G \parallel P_{\mathrm{KGC}}$, $Params := \{a, b, p, q, G, P_{\mathrm{KGC}}\}$, and send $Params$ and s to $\mathcal{A}_{\mathrm{II}}(\lambda)$. Initialize the lists $\mathbf{S}$ and $\mathbf{D}$ with empty list ($\emptyset$) respectively. As in Algorithm $\mathcal{C}_{\mathrm{I}}$, prepare lists $\mathbf{H}_1, \mathbf{H}_2, \mathbf{H}_3, \mathbf{H}_4, \mathbf{L}$ and choose nine values $h_{1O}^*, h_{1P}^* \in \{0,1\}^n$ and $h_{2O}^*, h_{2P}^*, h_{2O}'^*, h_{2P}'^*$, h_3^*, $h_4^*, \sigma^* \in \mathbb{Z}_q^*$.

Attack: Answer $\mathcal{A}_{\mathrm{II}}(\lambda)$'s queries to oracles as follows:

- $\mathrm{H}_1(\mathrm{x})$, $\mathrm{H}_2(\mathrm{x})$: (Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$.)
- $\mathrm{H}_3(\mathrm{w} \parallel h)$:(Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$, except that two P_{KGC}s in Algorithm $\mathcal{C}_{\mathrm{I}}$ are replaced by Ps)
- $\mathrm{H}_4(m \parallel \mathrm{w} \parallel h)$: (Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$, except that P_{KGC} in Algorithm $\mathcal{C}_{\mathrm{I}}$ is replaced by P)
- $\mathcal{O}_{\mathrm{CreateUser}}(\mathrm{ID}_U)$: Let $h_1 \leftarrow \mathrm{H}_1(B_{Params} \parallel \mathrm{ID}_U)$ and respond differently in the following cases:
  - If $\mathcal{B}$=0, $h_1 = h_{1O}^*$ and the current query is the $\mathcal{J}_O$-th one on ID_U, then follow

the next steps:

a. If $\mathcal{J}_O$ =1 then choose $\mathrm{k}_U \in \mathbb{Z}_q^*$ randomly, let $\mathrm{K}_U \leftarrow [\mathrm{k}_U]G$, $\mathrm{c}_U \leftarrow (\mathrm{k}_U + s \cdot h_{2O}'^*) \bmod q$ and put $(\mathrm{K}_U \| h_1, h_{2O}'^*)$ in $\mathbf{H}_2$. Otherwise, search $\mathbf{L}$ for the latest item $(\mathrm{ID}_U, *, \mathrm{K}_U, *, \mathrm{c}_U)$ indexed by ID_U to get K_U and c_U.

b. $\mathrm{P}_U \leftarrow [h_3^*]P - [h_{2O}'^*]P_{\mathrm{KGC}} - \mathrm{K}_U$.

c. Put $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \perp, \mathrm{c}_U)$ in $\mathbf{L}$, put $(\mathrm{P}_U \| \mathrm{K}_U \| h_1, h_{2O}^*)$ in $\mathbf{H}_2$ and return $(\mathrm{P}_U, \mathrm{K}_U)$.

– If $\mathcal{B}$ =1, $h_1 = h_{1P}^*$ and the current query is the $\mathcal{J}_P$-th one on ID_U, then follow the next steps:

a. If $\mathcal{J}_P$ =1 then choose $\mathrm{k}_U \in \mathbb{Z}_q^*$ randomly, let $\mathrm{K}_U \leftarrow [\mathrm{k}_U]G$, $\mathrm{c}_U \leftarrow (\mathrm{k}_U + s \cdot h_{2P}'^*) \bmod q$ and put $(\mathrm{K}_U \| h_1, h_{2P}'^*)$ in $\mathbf{H}_2$. Otherwise, search $\mathbf{L}$ for the latest item $(\mathrm{ID}_U, *, \mathrm{K}_U, *, \mathrm{c}_U)$ indexed by ID_U to get K_U and c_U.

b. $\mathrm{P}_U \leftarrow [(h_4^*/h_3^*) \bmod q]P - [h_{2P}'^*]P_{\mathrm{KGC}} - [(\sigma^*/h_3^*) \bmod q]G - \mathrm{K}_U$.

c. Put $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \perp, \mathrm{c}_U)$ in $\mathbf{L}$, put $(\mathrm{P}_U \| \mathrm{K}_U \| h_1, h_{2P}^*)$ in $\mathbf{H}_2$ and return $(\mathrm{P}_U, \mathrm{K}_U)$.

– Else, follow the next steps:

a. $\mathrm{s}_U \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $\mathrm{P}_U \leftarrow [\mathrm{s}_U]G$; $\mathrm{k}_U \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$; $\mathrm{K}_U \leftarrow [\mathrm{k}_U]G$; $h_2' \leftarrow_{\mathrm{R}} \mathbb{Z}_q^*$.

b. If there exists $h \in \mathbb{Z}_q^*$ such that $(\mathrm{K}_U \| h_1, h) \in \mathbf{H}_2$ and $h \neq h_2'$ then terminate the game.

c. If $(\mathrm{K}_U \| h_1, h_2') \notin \mathbf{H}_2$ then check whether h_2' is in $\{h_{2O}^*, h_{2P}^*\}$. If it is, terminate the game. Otherwise, put $(\mathrm{K}_U \| h_1, h_2')$ in $\mathbf{H}_2$.

d. $\mathrm{c}_U \leftarrow (\mathrm{k}_U + s \cdot h_2') \bmod q$.

e. Put $(\mathrm{ID}_U, \mathrm{P}_U, \mathrm{K}_U, \mathrm{s}_U, \mathrm{c}_U)$ in $\mathbf{L}$ and return $(\mathrm{P}_U, \mathrm{K}_U)$.

- $\mathcal{O}_{\mathrm{GetSecretKey}}$, $\mathcal{O}_{\mathrm{GetPublicKey}}$, $\mathcal{O}_{\mathrm{GetDelegation}}$, $\mathcal{O}_{\mathrm{GetProxySign}}$: (Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$.)

Forgery: Wait until $\mathcal{A}_{\mathrm{II}}(\lambda)$ outputs a tuple $(\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \delta_O^*, \mathrm{ID}_P^*, \mathrm{P}_P^*, \mathrm{K}_P^*, m^*, \sigma_{m^*})$ as the final result of the game.

3. Check $\mathcal{A}_{\mathrm{II}}(\lambda)$'s final result and output the solution of the above instance (i.e. the input) as follows:

(1) (Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$.)

(2) Parse δ_O^* as $(\mathrm{w}_O^*, R_O^*, \sigma_O^*)$. If the following boolean expression is true, then reject the final result and output $\perp$.

$$((\mathrm{ID}_O^*, \mathrm{P}_O^*) \in \mathbf{S} \vee (\mathrm{ID}_O^*, \mathrm{P}_O^*, \mathrm{K}_O^*, \mathrm{w}_O^*, \delta_O^*) \in \mathbf{D}) \wedge (\mathrm{ID}_P^*, \mathrm{P}_P^*) \in \mathbf{S}.$$

(3), (4), (5) (Same as defined in Algorithm $\mathcal{C}_{\mathrm{I}}$.) (The end of the algoritm $\mathcal{C}_{\mathrm{II}}$)

Since the analysis of $\mathcal{C}_{\mathrm{II}}$ is similar to the one of $\mathcal{C}_{\mathrm{I}}$, we omit it. This completes the proof.

6. Conclusion

Certificateless proxy signature (CLPS) stands for proxy signature in the certificateless setting which is intermediate between traditional PKI and Identity-based setting and has neither the certificate management issue nor private key escrow problem.

In this paper, we formalized the definition of a CLPS scheme and the security model for CLPS schemes. Comparing with others, our definition of a CLPS scheme is more versatile and our security model is more practical. We then proposed a pairing-free CLPS scheme using the standard ECDSA (Elliptic Curve Digital Signature Algorithm) and proved that in the random oracle model our scheme is existentially unforgeable against an adaptively chosen-message attack and an adaptively chosen-warrant attack under the assumption that the semi-logarithm problem is hard. To the best of our knowledge, our scheme is one of the provably secure CLPS schemes without bilinear pairings.

Using Cheng and Chen's technique [5] and modified ECDSA [16], we can easily convert our scheme to the one based on a standard complexity assumption, i.e. the discrete logarithm assumption.

Our future work is to construct a CLPS schemes without pairings in the standard model.

References

[1] S.S. Al-Riyami, K.G. Paterson, Certificateless public key cryptography, In: C.S. Laih

(Eds.), Proc. 9th Int. Conf. Theory and Application of Cryptology and Information Security, ASIACRYPT 2003, LNCS 2894, Springer, 2003, pp. 452–473.

[2] D. Brown, On the provable security of ECDSA, In: I.F. Blake, G. Seroussi, N.P. Smart (Eds.), Advances in Elliptic Curve Cryptography, Cambridge University Press, 2005, pp. 21–40.

[3] H. Chen, F.-T. Zhang, R.-S. Song, Certificateless proxy signature scheme with provable security, Journal of Sofware 20(3) (2009) 692–701.

[4] Y.-C. Chen, C.-L. Liu, G. Horng, K.-C. Chen, A provably secure certificateless proxy signature scheme, International Journal of Innovative Computing, Information and Control, 7(9) (2011) 5557–5569.

[5] Z. Cheng, L. Chen, Certificateless public key signature schemes from standard algorithms, In: C. Su, H. Kikuchi (Eds.) Proc. 14th Int. Conf. Information Security Practice and Experience, ISPEC 2018, LNCS 11125, Springer, 2018, pp. 179–197.

[6] J. Cui, J. Zhang, H. Zhong, R. Shi, Y. Xu, An efficient certificateless aggregate signature without pairings for vehicular ad hoc networks, Information Sciences 451–452 (2018) 1–15. https://doi.org/10.1016/j.ins.2018.03.060.

[7] H. Du, Q. Wen, Certificateless proxy multi-signature, Information Sciences 276 (2014) 21–30. http://dx.doi.org/10.1016/j.ins.2014.02.043.

[8] M. Girault, Self-certified public keys, In: D.W. Davies (Eds.) Proc. Workshop on the Theory and Application of Cryptographic Techniques, Advances in Cryptology-EUROCRYPT '91, LNCS 547, Springer-Verlag, 1991, pp. 490–497.

[9] B. C. Hu, D.S. Wong, X. Deng, Certificateless signature: a new security model and an improved generic construction, Design, Codes and Cryptography 42(2) (2007) 109–126. https://doi.org/10.1007/s10623-006-9022-9.

[10] ISO/IEC 14888-3:2016, Information Technology – Security Techniques – Digital Signatures with Appendix – Part 3: Discrete Logarithm Based Mechanisms, ISO/IEC (2016)

[11] Z. Jin, Q. Wen, Certificateless multi-proxy signature. Computer Communications 34(3) (2011) 344–352.

[12] A. Karati, SK H. Islam, G.P. Biswas, A Pairing-free and provably secure certificateless Signature scheme, Information Sciences 450 (2018) 378–391. https://doi.org/10.1016/ j.ins.2018.03.053.

[13] X. Li, K. Chen, L. Sun, Certificateless signature and proxy signature schemes from bilinear Pairings, Lithuanian Mathematical Journal 45(1) (2005) 76–83.

[14] R. Lu, D. He, C. Wang, Cryptanalysis and improvement of a certificateless proxy signature scheme from bilinear pairings, In: Proc. of the 8th ACIS Intern. Conf. on Software Engineering, Artifcial Intelligence, Networking, and Parallel/Distributed Computing, SNPD 07, IEEE Computer Society, 2007, pp. 285–290. https://doi.org/ 10.1109/ SNPD.2007.166.

[15] Y. Lu, J. Li, Provably secure certificateless proxy signature scheme in the standard model, Theoretical Computer Science 639 (2016) 42–59. https://doi.org/10.1016/j.tcs. 2016. 05.019

[16] J. Malone-Lee, N. Smart, Modifications of ECDSA, In: K. Nyberg, H. Heys (Eds.) Revised Papers of the 9th Annual Intern. Workshop on Selected Areas in Cryptography, SAC 2002, LNCS 2595, Springer, 2003, pp. 1–12.

[17] M. Mambo, K. Usuda, and E. Okamoto, Proxy signatures: delegation of the power to sign messages, IEICE Transactions on Fundamentals of Electronic Communications and Computer Science, E79-A (9) (1996) 1338–1354.

[18] S. Padhye, N. Tiwari, ECDLP-based certificateless proxy signature scheme with message recovery, Transactions on Emerging Telecommunications Technologies 26 (2015) 346–354. https://doi.org/10.1002/ett.2608.

[19] A. Shamir, Identity-based cryptosystems and signature schemes. In: G.R. Blakley, D. Chaum (Eds.) Proc. 3rd Annual Intern. Cryptology Conf., Advances in Cryptology–CRYPTO ’84, LNCS 196, Springer-Verlag, 1984, pp. 47–53. https://doi.org/10.1007/3-

540-39568-7 5.

[20] W. Shi, D. He, P. Gong, On the security of a certificateless proxy signature scheme with message recovery, Mathematical Problems in Engineering 2013, Article ID 761694 (2013) 1–4. http://dx.doi.org/10.1155/2013/761694.

[21] M. Tian, W. Yang, L. Huang, Cryptanalysis and improvement of a certificateless multi-proxy signature scheme, IACR Cryptology ePrint Archive: Report 2011/379, pp. 1–11 (2011).

[22] Z. Wan, X. Lai, J. Weng, X. Hong, Y. Long, W.-W. Jia, On constructing certificateless proxy signature from certificateless signature. Journal of Shanghai Jiaotong University (Science), 13(6) (2008) 692–694. https://doi.org/10.1007/s12204-008-0692-5.

[23] H. Xiong, F. Li, Z. Qin, A provably secure proxy signature scheme in certificateless cryptography, Informatica, 21(2) (2010) 277–294.

[24] J. Xu, H. Sun, Q. Wen, H. Zhang, Improved certificateless multi-proxy signature. The Journal of China Universities of Posts and Telecommunications 19(4) (2012) 94–105. https://doi.org/10.1016/S1005-8885(11)60288-4.

[25] W. Yap, S. Heng, B. Goi, Cryptanalysis of some proxy signature scheme without certificates. In: D. Sauveron, K. Markantonakis, A. Bilas, J.-J. Quisquater (Eds.) Proc. of the 1st Workshop on Information Security Theory and Practices, WISTP ’07, LNCS 4462, Springer, 2007, pp. 115–126.

[26] L. Zhang, F. Zhang, Q. Wu, Delegation of signing rights using certificateless proxy Signatures, Information Sciences 184(2012) 298–309 https://doi.org/10.1016/j.ins.2011.08.015