Certificateless Designated Verifier Proxy Signature

Cholun Kim 1

1. Faculty of Mathematics, Kim Il Sung University, Pyongyang, Democratic People’s Republic of Korea

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

Abstract

Proxy signature (PS) is a kind of digital signature, in which an entity called original signer can delegate his signing rights to another entity called proxy signer. Designated verifier signature (DVS) is a kind of digital signature where the authenticity of any signature can be verified by only one verifier who is designated by the signer when generating it. Designated verifier proxy signature (DVPS) combines the idea of DVS with the concept of proxy signature (PS) and is suitable for being applied in many scenarios from e-tender, e-voting, e-auction, e-health and e-commerce, etc. Many DVPS schemes have been proposed and Identity-based DVPS (IBDVPS) schemes have also been discussed. Certificateless public-key cryptography (CL-PKC) is acknowledged as an appealing paradigm because there exists neither the certificate management issue as in traditional PKI nor private key escrow problem as in Identity-based setting. A number of certificateless designated verifier signature (CLDVS) schemes and many certificateless proxy signature (CLPS) schemes have been proposed. However, to the best of our knowledge, the concept of Certificateless Designated Verifier Proxy Signature (CLDVPS) has not been appeared in the literature.

In this paper, we formalize the definition and the security model of CLDVPS schemes. We then construct the first CLDVPS scheme and prove its security.

Keywords: certificateless public-key cryptography; designated verifier proxy signature; certificateless designated verifier proxy signature; random oracle model; provable security; bilinear pairings.

1. Introduction

Digital signature is one of the most important primitives in cryptography and serves as a powerful tool in information security, including authentication, data integrity and non-repudiation in a public channel. It falls into
the category of public-key cryptography (PKC), where each user has two keys, i.e., a public key and a corresponding private key, and it is an important issue to guarantee the authenticity of public keys which match with private keys. Digital signature schemes allow any signer to sign a message using its private key in such a way that any other (i.e., verifier) who has an authentic copy of the public key associated with the signer can verify the origin and the integrity of the message. In traditional public key setting such as PKI, certificates generated by a trusted third party (TTP) are used to distribute authentic public keys, but it is costly to use and manage them, and brings about new problems. In order to bypass those problems, Shamir [51] introduced the concept of Identity-based Signature (IBS) along with Identity-based encryption (IBE). In IBS, the public key of a user is its identity information (e.g., name, email address, IP address, or social security number) 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 its private key, but unlike a PKG in IBS schemes, the KGC only produces a partial private key for any user by using the user’s identity and a master secret key, and transfers it to the user securely. Then the user generates its own full private key by combining the partial private key generated by KGC with a secret value chosen by itself. 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 the CLS schemes is no longer its identity, but it can be certified by the structure of the CL-PKC without any certificate.

Proxy signature, introduced by Mambo et al. [44] in 1996, is a kind of digital signature, in which a user called original signer can delegate his signing right 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. Proxy signature schemes are widely used in distributed systems where delegation of signing rights is quite common, including cloud computing, global distribution networks and electronic commerce. On the basis of the delegation mode, proxy signature schemes can be classified three types [44]: 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 which is generated with 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 types of security policies that specify 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 [44]: (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.

In typical digital signature schemes, signatures generated by any signer are publicly verifiable and transferable, and have the non-repudiation property. In other words, anyone who gets a signature and has the signer’s public key can verify the authenticity of the signature, and transfer his conviction to anyone else by presenting the signature. Furthermore, no signer can deny the signature generated by him. However, the public verifiability and non-repudiation property of signatures is not always desirable in some scenarios where the privacy of the signer must be protected, such as e-tender, e-voting, e-auction, e-health and e-commerce applications. In 1996, Jakobsson et al. [28] suggested the notion of designated verifier signature (or proof) (DVS) as a solution for resolving the conflict between authenticity and privacy of signatures. In DVS, the authenticity of
any signature can be verified by only one verifier who is designated by the signer when generating it, in a non-interactive way. That is, for any signature in DVS schemes, no one else than the designated verifier of the signature can get the conviction on the identity of the signer who generated it, while the designated verifier cannot transfer his conviction to anyone else since he can always simulate the signature indistinguishably. Anyone can verify the validity of any signature (i.e., the fact that the signature is generated by one of the signer and the designated verifier), but no one except the designated verifier can confirm the identity of the true signer due to the indistinguishability between the original signature and the fake signature (transcript) simulated by the designated verifier even if the private keys of the signer and the designated verifier are revealed. Jakobsson et al. [28] also proposed the idea of a strong DVS (SDVS), where the designated verifier’s private key is required in the verification algorithm so that no one can even verify the validity of signatures except the designated verifier.

Steinfeld et al. [54] extended the concept of DVS to the one of universal DVS (UDVS), in which anyone who holds the standard signature on a message can transform it to a DVS to protect the privacy of the signature holder from dissemination of signatures by verifiers. In 2004, Laguillaumie and Vergnaud [34] proposed the new primitive called multi-designed verifiers signatures (MDVS) where the signer can designate multiple verifiers for one signature and each verifier can verify the signature individually.

There are some security requirements for secure DVS schemes. The correctness and the unforgeability are the de facto standard requirements for digital signatures. The correctness requires that any signature properly produced by a legitimate signer must be accepted by the designated verifier. The unforgeability means that it is infeasible to produce a valid signature without the knowledge of the private key of either the signer or the designated verifier. The natural requirement derived from the essence of the DVS is the non-transferability which is identified by Steinfeld et al. [54]. The non-transferability means that a designated verifier for any signature cannot use the signature to produce the evidence which convinces any third party that the signature was generated by the signer. As the requirement for anonymity of signatures, Laguillaumie and Vergnaud [35] suggested a weaker notion for ordinary DVS, source hiding and a stronger notion for strong DVS, privacy of signer’s identity (PSI). Source hiding means that given a message and a signature on it, it is infeasible to decide who from the signer or the designated verifier generated the signature, even if the private keys of the signer and designated verifier are known. Privacy of signer’s identity means that given a signature and the public keys of two potential signers for the signature, it is infeasible to determine under which of the two corresponding private keys the signature was generated, without the knowledge of the private key of the designated verifier. Lipmaa et.al [40]
identified a new security requirement for secure DVS schemes, called the non-delegatability. The non-delegatability requires that if a third party can produce a valid signature with respect to a designated verifier or identify the identity of the signer from the signature, it must ‘know’ the private key of either the original signer or the designated verifier. It is a critical property in applications where responsibilities of signers and designated verifiers are so important that they cannot be delegated to third parties.

In 2003, Dai et al. [8] proposed a primitive called a designated verifier proxy signature (DVPS) by combining the idea of DVS with the concept of proxy signature (PS) to capture the security requirements from e-commerce applications, such as sale of digital products (softwares, games, digital music, e-books, etc.) In the definition of DVPS proposed by Dai et al. [8], the verifiers of proxy signatures are designated by original signers in advance and this point makes their DVPS slightly inflexible. In 2004, Wang [61] suggested an improved DVPS scheme where verifiers of proxy signatures are designated by proxy signers instead of the original signers. Then various DVPS schemes have been discussed in many literatures [18,17,24,26,30,36,37,41,59,65,68,69].

1.1 Motivation

In 2005, Cao et al. [2] introduced the idea of DVPS into the identity-based setting to propose the first identity-based DVPS (IBDVPS) scheme. Along their research line, Lal and Verma [36, 37], Kang et al. [31], Yang [65], Islam and Biswas [24, 26], Hu et al. [18], etc., successively proposed various IBDVPS schemes. On the other hand, in 2006, the first DVS scheme in the certificateless setting, namely a certificateless DVS (CLDVS) scheme, proposed by Huang et al. [23]. Since then, Du and Wen [10], Yang et al. [64], He and Chen [14], Islam and Biswas [25], Chen et al. [5], Pakniat [46], Rastegari et al. [48] etc., also proposed CLDVS schemes. In 2005, Li et al. [38] combined the concept of proxy signature with CL-PKC to propose the notion of certificateless proxy signature (CLPS). Then the CLPS and its some extensions (e.g. certificateless multi-proxy signature, certificateless proxy multi-signature, etc) have been discussed in many literatures [3,4,11,29,43,45,58,60,62,63,70].

It is natural to combine the concept of DVPS with CL-PKC which is more flexible public-key setting than the identity-based setting. However, to the best of our knowledge, there are no researches in this direction up to now.

Generally, the security of any cryptographic primitive is ensured by a formal proof, neither by a heuristic proof nor by a claim without any proof, and formal proofs of the security are based on formal definitions and precise assumptions. The power of adversaries and the security of the primitive are defined within the security
model of the primitive. A lot of previously proposed primitives without a formal security proof turned out to be insecure subsequently to their release, then fixed, but they would be proved to be insecure again. We can see this phenomenon on IBDVPS schemes and CLDVS schemes in the literature. For example, Kang et al. [31] showed that Lal and Verma’s scheme in [36] is not secure. Later, the insecurity of Kang et al.’s IBDVPS scheme in [31] was proved. The first IBDVPS scheme proposed by Cao et al. in [2], and Yang’s scheme in [65] have not a formal proof of their security. Islam and Biswas [24] proposed an IBDVPS scheme with only heuristic security proof. Islam and Biswas [26] again proposed an IBDVPS scheme and presented a formal security proof, but did not give a formal definition of the security model. Meanwhile, Huang et al.’s CLDVS scheme in [23] is insecure against malicious KGC attacks [46]. Lin et al. [39] showed that Islam and Biswas’s CLDVS scheme in [25] is vulnerable to key-compromise and malicious KGC attacks. Pakniat [46] disproved Chen et al.’s claim that their certificateless strong DVS scheme in [5] satisfies all the security requirements, and showed that it is forgeable. Li et al. [38] proposed the first scheme of the CLPS without any formal security proof. Later, Yap et al. [66] and Lu et al. [42] found that Li et al.’s scheme is insecure. Furthermore, Lu et al. [42] improved Li et al.’s CLPS scheme, but they, just as Li et al. [38], did not present any formal security proof for the scheme. Padhye et al. [45] proposed an elliptic curve discrete log problem (ECDLP)-based CLPS scheme without pairings. They presented a formal security proof, but Shi et al. [53] showed their scheme is not secure for practical applications. This illustrates the necessity of well-defined security models and a rigorous security analysis.

Hu et al. [18] presented the formal definition of an IBDVPS scheme and a formal security model for it, focusing on only the unforgeability. Hu et al. [17] proposed a well-defined security model for DVPS schemes. Huang et al. [22], Huang et al. [20], Tian et al. [57], Yoneyama et al. [67], Sharma et al. [52], Islam and Biswas [27], Hu et al. [19], Steinfeld et al. [54], Huang et al. [21], Laguillaumie et al. [33], Rastegari et al. [49], Laguillaumie et al. [34], Damgård et al. [9], Rastegari and Berenjkoub [47], etc. proposed a formal security model for identity-based DVS (IBDVS), SDVS, UDVS, or MDVS schemes, respectively. A number of security models for the CLPS schemes were proposed by Wan et al. [60], Chen et al. [3], Chen et al. [4], Jin et al. [29], Zhang et al. [70], Xu et al. [63], Du et al. [11], Padhye et al. [45], etc. However, we cannot find a formal security model for CLDVS schemes in the literature.

Motivated by the above discussion, we propose the notion of certificateless designated verifier proxy signature (CLDVS), the security model for CLDVS schemes and the concrete CLDVS scheme in this paper.
1.2 Our contribution

For the first time in the literature, we introduce the notion of certificateless (strong) designated verifier proxy signature (CLDVPS) as a kind of strong DVS, formalize the security model for CLDVPS schemes and propose the first CLDVPS scheme in this paper.

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

(1) Firstly, we formalize the definition of a CLDVPS scheme and propose the formal security model for CLDVPS schemes. Referring to Hu et al.’s security model [18] for IBDVPS schemes, we propose a formal definition of a CLDVPS scheme, then improve the security models for CLPS schemes from [4, 63, 70, 11, 45] and combine them with Hu et al.’s security model [17] for DVPS schemes and other models for DVS to construct the first security model for CLDVPS schemes.

(2) Secondly, we propose a concrete CLDVPS scheme. This is the first CLDVPS scheme in the literature. Our scheme is inspired by Hu et al.’s DVPS scheme [17], Islam and Biswas’s IBDVPS scheme [26], and CLPS schemes from [70, 45].

(3) Finally, based on our new security model for CLDVPS schemes, we prove that our scheme is secure in the random oracle model under the assumption that the computational Diffie–Hellman (CDH) problem and Gap Bilinear Diffie–Hellman (GBDH) problem are intractable.

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 CLDVPS scheme and formalize the security model for CLDVPS schemes. Then a concrete CLDVPS scheme 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;
- $1^\lambda$: the string consisting of $\lambda$ ones;
- $s \leftarrow S$: the operation of picking an element $s$ uniformly at random from a finite set $S$;
- $r \leftarrow A(i_1, \cdots, i_m)$: the operation of running a probabilistic (randomized) algorithm $A$ with input $i_1, \cdots, i_m$ and assigning the result to $r$;
• \( r \leftarrow A(i_1, \cdots, i_m) \): the operation of running a deterministic algorithm \( A \) 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 \( L \).

• \( \bot \): the null symbol indicating a failure or a false value;

• \( \emptyset \): the empty set or the empty string.

• Let \( \Gamma_1 \) be a cyclic additive group of prime order \( q \) with the addition “+”. If \( P \in \Gamma_1 \) and \( k \in \mathbb{Z}_q \), we denote \( \underbrace{P + \cdots + P}_{k \text{ times}} \in \Gamma \) as \([k]P\).

• PPT stands for probabilistic polynomial-time.

• \( \Pr[E] \) stands for the probability of the event \( E \).

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

**Definition 1.** Let \( \Gamma_1 \) be a cyclic additive group of prime order \( q \) and \( \Gamma_1 \) be a cyclic multiplicative group of the same order with the identity element \( 1_{\Gamma_1} \). A map \( \hat{e} : \Gamma_1 \times \Gamma_1 \rightarrow \Gamma_2 \) is called a bilinear pairing if it has the following properties:

1. **Bilinearity:** \( \hat{e}(P, Q + R) = \hat{e}(P, Q) \cdot \hat{e}(P, R) \) and \( \hat{e}(P + Q, R) = \hat{e}(P, R) \cdot \hat{e}(Q, R) \) for all \( P, Q, R \in \Gamma_1 \).

2. **Non-degeneracy:** There exists \( P \in \Gamma_1 \) such that \( \hat{e}(P, P) \neq 1_{\Gamma_2} \).

3. **Efficient Computability:** There is an efficient algorithm to compute \( \hat{e}(P, Q) \) for all \( P, Q \in \Gamma_1 \).

\((\Gamma_1, \hat{e}, \Gamma_2)\) is called a bilinear group pair.

Numerous computational problems on bilinear pairings have been defined.

**Definition 2.** Let \( \Gamma_1 \) be a cyclic additive group of prime order \( q \), \( G \) be a generator of \( \Gamma_1 \), \( \Gamma_1 \) be a cyclic multiplicative group of prime order \( q \), \( \hat{e} : \Gamma_1 \times \Gamma_1 \rightarrow \Gamma_2 \) be a bilinear pairing.

Decisional Bilinear Diffie–Hellman (DBDH) Problem in $(\Gamma_A, \Gamma_M)$: Given $h \in \Gamma_M$, $G$, and $[u]G$, $[v]G$, $[w]G \in \Gamma_A$ for some unknown $u, v, w \in \mathbb{Z}_q^*$, decide whether $h = \hat{e}(G, G)^{uvw}$ or not.

A DBDH oracle $O_{DBDH}$ is an algorithm that on input $G$, $[u]G$, $[v]G$, $[w]G \in \Gamma_A$ and $h \in \Gamma_M$, output 1 if $h = \hat{e}(G, G)^{uvw}$, or 0 otherwise.


In CL–PKC, two types of adversaries with different capabilities—type I adversaries and type II adversaries, are considered [1]:

- A type I adversary $A_1$ 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 $A_{II}$ acts as a malicious-but-passive KGC, and knows the master secret key and partial private keys of all user but cannot replace any user’s public key.

3. Certificateless designated verifier proxy signature: formal definitions and security model

In this section, we describe the formal definition of certificateless designated verifier proxy signature (CLDVPS) schemes within the category of the strong designated verifier signature (SDVS) and propose security model for CLDVPS schemes. In SDVS schemes, the generation of any signature involves the public key of the designated verifier and the verification of the signature needs the private key of the designated verifier. That is, a strong DVS (SDVS) scheme is more suitable to the main goal of the DVS, i.e., the guarantee of the restrictive verifiability or the anonymity of signatures, and practical applications than any ordinary DVS scheme where anyone can decide whether any signature has been generated by one of the signer and the designated verifier or not, but cannot determine who the true signer is, excepting the designated verifier. In fact, the notion of the ordinary DVS corresponds to the one of the ring signature [50] in the particular case where the set $U$ of possible signers contains two users, $U := \{S, D\}$, i.e., the signer $S$ and the designated verifier $D$. If $S$ generates a ring
signature for the set $U$, then anyone can verify that either $S$ or $D$ has created the signature. User $D$, of course, knows that he has not computed this signature, so he concludes that the actual author of the signature is $S$ [15].

### 3.1 Definition of a CLDVPS scheme

We first define the CLDVPS scheme formally, and then emphasize some features of the new definition.

**Definition 3.** A certificateless (strong) designated verifier proxy signature (CLDVPS) (with appendix) scheme $\Sigma(\lambda)$ comprises legitimate entities of four types: the KGC, the original signer, the proxy signer and the designated verifier. It consists of the following eight polynomial-time algorithms:

1. $(Params, s) \leftarrow \text{Setup}(1^\lambda)$: This is a probabilistic (randomized) algorithm run by the KGC, takes as input $1^\lambda$, and returns a list of public system parameters $Params$ and a master secret key $s$.

2. $(P_U, s_U) \leftarrow \text{GenerateUserKeys}(Params, ID_U)$: This is a probabilistic algorithm run by any entity except the KGC. When a user $U$ runs, it takes as input $Params$ and his identity $ID_U$, and outputs a public/secret key pair $(P_U, s_U)$ for the user $U$.

3. $(K_U, c_U) \leftarrow \text{GeneratePartialKeys}(Params, s, ID_U, P_U)$: This is a probabilistic algorithm run by the KGC, 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 $(K_U, c_U)$ for the user $U$. $(P_U, K_U)(s_U, c_U)$ is the pair of full public/private keys for the user $U$.

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

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

6. $\sigma_m \leftarrow \text{GenerateDVPSign}(Params, ID_O, P_O, K_O, \delta_O, ID_P, P_P, K_P, s_P, c_P, ID_D, P_D, K_D, m)$: This is a probabilistic algorithm run by proxy signers. It takes as input $Params$, the identity $ID_O$ and the full public
key \((P_O, K_O)\) of an original signer \(O\), the delegation \(\delta_O\) issued by \(O\), the identity \(ID_P\) and the full public/private key pair \((P_P, K_P)\)/(\(s_P, c_P\)) of the proxy signer \(P\) specified by the delegation \(\delta_O\), the identity \(ID_D\) and the full public key \((P_D, K_D)\) of the verifier \(D\) designated by the proxy signer \(P\), and a message \(m\). It outputs a signature \(\sigma_m\) of \(P\) on the message \(m\).

(7) \(\text{True/False} \leftarrow \text{VerifyDVPSign} (\text{Params}, ID_O, P_O, K_O, \delta_O, ID_P, P_P, K_P, ID_D, P_D, K_D, s_D, c_D, m, \sigma_m)\): This is a deterministic algorithm run by designated verifiers. It takes as input \(\text{Params}\), the identity \(ID_O\) and the full public key \((P_O, K_O)\) of an original signer \(O\), and the delegation \(\delta_O\) issued by \(O\), the identity \(ID_P\) and the full public key \((P_P, K_P)\) of a proxy signer \(P\), the identity \(ID_D\) and the full public/private key pair \((P_D, K_D)\)/(\(s_D, c_D\)) of a designated verifier \(D\), a message \(m\), and a proxy signature \(\sigma_m\). It outputs \(\text{True}\) if the delegation \(\delta_O\) is valid, the proxy signer \(P\) is specified in the delegation \(\delta_O\), the message \(m\) conforms to the delegation \(\delta_O\) and the proxy signature \(\sigma_m\) is valid, or \(\text{False}\) otherwise.

(8) \(\sigma_m^S \leftarrow \text{SimulateDVPSign} (\text{Params}, ID_O, P_O, K_O, \delta_O, ID_P, P_P, K_P, ID_D, P_D, K_D, s_D, c_D, m)\): This is a probabilistic algorithm run by any user (a potential designated verifier). It takes as input \(\text{Params}\), the identity \(ID_O\) and the full public key \((P_O, K_O)\) of an original signer \(O\), the delegation \(\delta_O\) issued by \(O\), the identity \(ID_P\) and the full public key \((P_P, K_P)\) of a proxy signer \(P\), the identity \(ID_D\) and the full public/private key pair \((P_D, K_D)\)/(\(s_D, c_D\)) of a designated verifier \(D\), and a message \(m\). It outputs a simulated proxy signature \(\sigma_m^S\) on the message \(m\) in behalf of the proxy signer \(P\).

Remark. In fact, the last algorithm \(\text{SimulateDVPSign}\) is neither an essential one, nor a component of the definition for a CLDVPS scheme. It is only used to describe and prove a security requirement for SDVS schemes, namely, non-transferability. We can get the definition of the certificateless ordinary designated verifier proxy signature scheme by eliminating the parameters \(s_D\) and \(c_D\), i.e., the full private key of the designated verifier \(D\) from the input parameters of the algorithm \(\text{VerifyDVPSign}\).

We obtained the above definition of a CLDVPS scheme by improving and modifying the definitions of a CLPS scheme in such a way that it can capture the functions of a DVS scheme. When comparing our definition with the previous definitions of a CLPS scheme (e.g., in [4, 11, 45, 60, 63, 70]), we can find some differences.
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. It 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 [6] has done for the definition of CL-PKC, the public key \( P_U \) of the user \( U \) is added to its input parameters, 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 the schemes achieving Girault’s trust level 3 [13]. 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, e.g., Padhye and Tiwari’s CLPS scheme [45].

Another difference is that some algorithms in the traditional definitions of CL-PKC are eliminated in our definition. Firstly, in the same approach as taken by Wan et al. [60], Xu et al. [63], Zhang et al. [70], Du and Wen [11], 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 [16]. Secondly, we eliminate the algorithm generating a proxy signing key and include the function of the algorithm into the proxy signing algorithm GenerateDVPSign. This makes the our definition more versatile since it is possible that the partial private key \( c_P \) and the secret key \( s_P \) of the proxy signer, the delegation \( \delta_O \), and the others are ‘mixed’ together in some randomized way during proxy signing.

Final 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 parameters of such algorithms as GenerateDelegation, VerifyDelegation, GenerateDVPSign, VerifyDVPSign, etc.

**Definition 4.** (Correctness) For a CLDVPS scheme \( \Sigma(\lambda) \), any \( (\text{Params}, s) \) \( \leftarrow \) Setup(1^\lambda), any \( \text{ID}_O, \text{ID}_P, \text{ID}_D \in \{0,1\}^* \) and any \( m, w_O \in \{0,1\}^* \), let
\[(P_O, s_O) \leftarrow \text{GenerateUserKeys}(\text{Params}, \text{ID}_O),\]

\[(P_p, s_p) \leftarrow \text{GenerateUserKeys}(\text{Params}, \text{ID}_p),\]

\[(P_D, s_D) \leftarrow \text{GenerateUserKeys}(\text{Params}, \text{ID}_D),\]

\[(K_O, c_O) \leftarrow \text{GeneratePartialKeys}(\text{Params}, s, \text{ID}_O, P_O),\]

\[(K_p, c_p) \leftarrow \text{GeneratePartialKeys}(\text{Params}, s, \text{ID}_P, P_P),\]

\[(K_D, c_D) \leftarrow \text{GeneratePartialKeys}(\text{Params}, s, \text{ID}_D, P_D),\]

\[\delta_o \leftarrow \text{GenerateDelegation}(\text{Params}, \text{ID}_O, P_O, K_O, s_O, c_O, w_O),\]

\[\sigma_m \leftarrow \text{GenerateDVPSign}(\text{Params}, \text{ID}_O, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, s_P, c_P, \text{ID}_D, P_D, K_D, m),\]

\[\sigma^s_m \leftarrow \text{SimulateDVPSign}(\text{Params}, \text{ID}_O, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, \text{ID}_D, P_D, K_D, s_D, c_D, m).\]

If the differences of the following values

\[\Pr[\text{VerifyDelegation}(\text{Params}, \text{ID}_O, P_O, K_O, \delta_O) = \text{True}]\]

\[\Pr[\text{VerifyDVPSign}(\text{Params}, \text{ID}_O, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, \text{ID}_D, P_D, K_D, s_D, c_D, m, \sigma_m) = \text{True}],\]

\[\Pr[\text{VerifyDVPSign}(\text{Params}, \text{ID}_O, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, \text{ID}_D, P_D, K_D, s_D, c_D, m, \sigma^s_m) = \text{True}]\]

from 1 are all negligible, the CLDVS scheme \(\Sigma(\lambda)\) is said to be correct or to satisfy the correctness(completeness).

### 3.2 Security Model

We first define adversaries against CLDVS schemes and then specify security requirements for them.

As described in Section 2, there are two types of adversaries, types I and II, against a CLDVS scheme as a cryptographic primitive of CL-PKC. The adversary against a CLDVS scheme of Definition 3 is defined as follows:

**Definition 5.** An adversary \(A(\lambda)\) against a CLDVS scheme \(\Sigma(\lambda)\) is a probabilistic algorithm which takes as input \(1^\lambda\) and can access to some of the following oracles (as well as the random oracles if there exists):

1. \((P_U, K_U) \leftarrow O_{\text{CreateUser}}(\text{ID}_U)\): On input an identity \(\text{ID}_U\), it creates a new user \(U\) whose identity is \(\text{ID}_U\) 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 new full public key \((P_U, K_U)\) is returned.
(2) \( s_U /\perp \leftarrow O_{\text{GetSecretKey}}(\text{ID}_U, P_U) \): On input the identity \( \text{ID}_U \) and the public key \( P_U \) of a user \( U \), it stores \( (\text{ID}_U, P_U) \) in the query list \( S \) 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) \( c_U /\perp \leftarrow O_{\text{GetPartialKey}}(\text{ID}_U, K_U) \): On input the identity \( \text{ID}_U \) and the partial public key \( K_U \) of a user \( U \), it stores \( (\text{ID}_U, K_U) \) in the query list \( P \) and outputs the 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) \( (P_U, K_U) /\perp \leftarrow O_{\text{GetPublicKey}}(\text{ID}_U) \): On input the identity \( \text{ID}_U \) of a user \( U \), it stores \( (\text{ID}_U, K_U) \) in the query list \( P \) and outputs the full public key \( (P_U, K_U) \) currently associated with \( \text{ID}_U \), if there exists the user \( U \). It outputs \( \perp \) otherwise.

(5) \( \text{“OK”} /\perp \leftarrow O_{\text{ReplaceUserKeys}}(\text{ID}_U, P'_{U}, s'_{U}) \): On input the identity \( \text{ID}_U \), a new public key \( P'_{U} \) and a new secret key \( s'_{U} \) of a user \( U \), it stores \( (\text{ID}_U, P'_{U}) \) in the query list \( R \), replaces the current pair of partial public/private keys of the user \( U \) with the new pair \( (P'_{U}, s'_{U}) \) and outputs “OK”, if there exists the user \( U \). It outputs \( \perp \) otherwise. \( s'_{U} \) may be the empty string.

(6) \( \delta_{O} /\perp \leftarrow O_{\text{GetDelegate}}(\text{ID}_O, P_{O}, K_{O}, w_{O}) \): On input the identity \( \text{ID}_O \) of an original signer \( O \), a full public key \( (P_{O}, K_{O}) \) and a warrant \( w_{O} \), it outputs the valid delegation \( \delta_{O} \) (created newly if there does not exist) on the warrant \( w_{O} \) and stores the tuple \( (\text{ID}_O, P_{O}, K_{O}, w_{O}, \delta_{O}) \) in the query-answer list \( D \), if there exists the user \( O \) and \( (P_{O}, K_{O}) \) is the (current or past) full public key of \( O \). It outputs \( \perp \) otherwise.

(7) \( \sigma_{m} /\perp \leftarrow O_{\text{GetDVPSign}}(\text{ID}_O, P_{O}, K_{O}, \delta_{O}, ID_P, P_P, K_P, ID_D, P_D, K_D, m) \): On input the identity \( \text{ID}_O \) and a full public key \( (P_{O}, K_{O}) \) of an original signer \( O \), a delegation \( \delta_{O} \), the identity \( \text{ID}_P \) and a full public key \( (P_P, K_P) \) of a proxy signer \( P \), the identity \( \text{ID}_D \) and a full public key \( (P_D, K_D) \) of a designed verifier \( D \), and a message \( m \), it outputs the valid proxy signature \( \sigma_{m} \) on the message \( m \). If there does not exist one of \( O, P \) and \( D \), \( \delta_{O} \) is not valid, or \( \text{ID}_P \) and \( m \) do not conform to \( \delta_{O} \), it returns \( \perp \).

(8) \( \text{True/False} /\perp \leftarrow O_{\text{VerifyDVPSign}}(\text{ID}_O, P_{O}, K_{O}, \delta_{O}, ID_P, P_P, K_P, ID_D, P_D, K_D, m, \sigma_{m}) \): On input the
identity ID_{O} and a full public key (P_{O}, K_{O}) of an original signer O, a delegation \delta_{O}, the identity ID_{P} and a full public key (P_{P}, K_{P}) of a proxy signer P, the identity ID_{D} and a full public key (P_{D}, K_{D}) of a designed verifier D, and a message m, it returns \perp if there does not exist one of O, P and D. Otherwise, it outputs True or False depending on whether the delegation \delta_{O} is valid, the proxy signer P is specified in the delegation \delta_{O}, the message m conforms to the delegation \delta_{O} and the proxy signature \sigma_m is a valid signature on m for D, or not, and stores (ID_{O}, P_{O}, K_{O}, \delta_{O}, ID_{P}, P_{P}, K_{P}, ID_{D}, P_{D}, K_{D}, m, \sigma_m) in the query list V.

An adversary is called a type I adversary and denoted by \Lambda_{I}(\lambda) if it can access all the above oracles but cannot access to the master secret key. An adversary is called a type II adversary and denoted by \Lambda_{II}(\lambda) if it knows the master secret key and partial private keys of all users but can access only 6 oracles of the above oracles except O_{GetPartialKeys} and O_{ReplaceUseKey}. An adversary is called a trusted adversary and denoted by \Lambda_{t}(\lambda) if it can access all the above oracles and all secret information including the master secret key.

Our definition of oracles for adversaries has some differences from other security models for CLPS, e.g., in [4, 11, 45, 63, 70]. Firstly, the oracle O_{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 O_{GetSecretKey} and O_{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 CLDVPS 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 O_{GetSecretKey} and O_{GetPartialKeys}. Finally, the specifications of the input/output of oracles are updated and the oracle O_{VerifyDVPSign} is added to capture the requirement of DVS.

We consider that a secure CLDVPS scheme, as a strong DVS scheme and a proxy signature scheme, should satisfy the strong unforgeability, the non-transferability, the privacy of signer’s identity, non-delegatability and the prevention of misuse property besides the correctness and the strongness. Since the verifiability, the strong identifiability, the strong undeniability and the distinguishability are conflict with the non-transferability, they are
excluded from the security requirements for a secure CLDVPS scheme. Next, we present the formal definitions of
strong unforgeability, the non-transferability, the privacy of signer’s identity and non-delegatability in turn.

**Definition 6 (Strong Unforgeability).** Let us consider the following interactive game between an adversary
A(λ) ∈ { A₁(λ), A₁(λ) } against a CLDVPS scheme Σ(λ) and a challenger X.

**sEUF Game:**

**Setup:** X runs Setup(1^λ) to obtain the system parameter list Params and the master secret key s. Then X
sends Params to the adversary A(λ). If A(λ) is A₁(λ), X also passes s to the adversary A(λ). Otherwise,
keeps the master secret key s secret. Finally, X initializes the lists P, S, R and D with empty list (∅) respectively.

**Attack:** A(λ) gathers information by querying oracles allowed to him adaptively. X correctly simulates the
oracles called by A(λ) and returns proper values.

**Forgery:** Eventually, A(λ) outputs a tuple (ID₁⁺, P₀⁺, K₀⁺, δ₀⁺, IDₚ⁺, Pₚ⁺, Kₚ⁺, ID_D⁺, P_D⁺, K_D⁺, m⁺, σ⁺). We say that A(λ) wins the game or succeeds in the game if the following conditions are all satisfied:

1. VerifyDVPSign(Params, ID₁⁺, P₀⁺, K₀⁺, δ₀⁺, IDₚ⁺, Pₚ⁺, Kₚ⁺, ID_D⁺, P_D⁺, K_D⁺, m⁺, σ⁺) = True, where
   s_D⁺ and c_D⁺ are the secret key and the partial private key of the designated verifier with identity ID_D⁺, respectively.

2. The tuple (ID₁⁺, P₀⁺, K₀⁺, δ₀⁺, IDₚ⁺, Pₚ⁺, Kₚ⁺, ID_D⁺, P_D⁺, K_D⁺, m⁺) has not been submitted to the oracle O_GetDVPSign in the Attack stage.

3. When A(λ) is A₁(λ), (ID₁⁺, K_D⁺) ∈ P or (ID₁⁺, P_D⁺) ∈ S ∪ R. When A(λ) is A₁(λ), (ID₁⁺, P_D⁺) ∈ S.

4. When A(λ) is A₁(λ), the following boolean expression is true:
   
   (((ID₁⁺, K_D⁺) ∈ P ∨ (ID₁⁺, P_D⁺) ∈ S ∪ R) ∧ (ID₁⁺, P₀⁺, K₀⁺, w₁⁺, *) ∈ D) ∨

   (((IDₚ⁺, Kₚ⁺) ∈ P ∨ (IDₚ⁺, Pₚ⁺) ∈ S ∪ R),

   where w₁⁺ is the warrant in the delegation δ₀⁺ and * stands for an arbitrary value.

When A(λ) is A₁(λ), the following boolean expression is true:

(((ID₁⁺, P₀⁺) ∈ S ∧ (ID₁⁺, P₀⁺, K₀⁺, w₁⁺, *) ∈ D) ∨ ((IDₚ⁺, Pₚ⁺) ∈ S).
A CLDVS scheme $\Sigma(\lambda)$ is said to be strong existentially unforgeable against an adaptively chosen-message attack and an adaptively chosen-warrant attack or to satisfy Strong Unforgeability if the probability of success of any PPT adversary $A(\lambda)$ against the scheme in the above game is negligible in $\lambda$.

An adversary against a CLDVS scheme can win the above game by forging a valid proxy signature or a valid delegation. A valid proxy signature cannot be generated by anyone including the original signer, except for the proxy signer and the designated verifier. According to the success condition (2) of an adversary in the above game, he can request a proxy signature on the challenge message $m^*$ to the oracle $O_{GetDVPSign}$ as long as one of the original signer, the proxy signer and the designated signer of the signature is not a target user (i.e., $ID^*_O$, $ID^*_p$, or $ID^*_D$, correspondingly) or the delegation is not equal to $\delta^*_O$. This is just the reason that the unforgeability of the above definition is strong.

Next, we define the non-transferability for CLDVS, inspired from the works of Emura et al. [12], Huang et al. [20], Yoneyama et al. [67], Hu et al. [17], Rastegari et al. [49].

**Definition 7** (Non-transferability). Let us consider the following interactive game between a trusted adversary (distinguisher) $A_r(\lambda)$ against a CLDVS scheme $\Sigma(\lambda)$ and a challenger $X$.

**NT Game:**

**Setup:** $X$ runs $\text{Setup}(1^\lambda)$ to obtain the system parameter list $\text{Params}$ and the master secret key $s$. Then $X$ sends $\text{Params}$ and $s$ to the adversary $A_r(\lambda)$.

**Challenge:** $A_r(\lambda)$ gathers information by querying oracles adaptively. $X$ correctly simulates the oracles called by $A_r(\lambda)$ and returns proper values. Eventually, $A_r(\lambda)$ submits to $X$ a challenge message $m^*$ and the identities $\{ID^*_O, ID^*_p, ID^*_D\}$ of an original signer, a proxy signer and a designated verifier selected as targets along with a valid delegation $\delta^*_O$ and the public keys $P^*_O, K^*_O, P^*_p, K^*_p, P^*_D, K^*_D$. Then $X$ flips a fair coin $b \leftarrow \{0, 1\}$, produces a signature $\sigma^* = \text{GenerateDVPSign}(\text{Params}, ID^*_O, P^*_O, K^*_O, \delta^*_O, ID^*_p, P^*_p, K^*_p, P^*_D, K^*_D, m^*)$ if $b = 0$ or $\sigma^* = \text{SimulateDVPSign}(\text{Params}, ID^*_O, P^*_O, K^*_O, \delta^*_O, ID^*_p, P^*_p, K^*_p, P^*_D, P^*_D, K^*_D, s^*_D, c^*_D, m^*)$ if $b = 1$ and replies to $A_r(\lambda)$ with $\sigma^*$.

**Guess:** After receiving $\sigma^*$, $A_r(\lambda)$ can continue to issue oracle queries. In the end, $A_r(\lambda)$ returns his guess
\( b' \in \{0, 1\} \).

We say that \( A_*(\lambda) \) wins the game or succeeds in the game if \( b' = b \), and define the advantage of \( A_*(\lambda) \) in the game as \( |\Pr[A_*(\lambda) \text{ wins the NT game}] - 1/2| \).

A CLDVPS scheme \( \Sigma(\lambda) \) is said to be non-transferable or to satisfy Non-transferability if the advantage of any PPT trusted adversary \( A_*(\lambda) \) against the scheme in the above game is negligible in \( \lambda \).

The following definition of the privacy of signer’s identity (PSI) for CLDVPS are inspired from the works of Tian et al. [57] and Damgård et al. [9].

**Definition 8** (Privacy of signer’s identity, PSI). Let us consider the following interactive game between a trusted adversary (distinguisher) \( A_*(\lambda) \) against a CLDVPS scheme \( \Sigma(\lambda) \) and a challenger \( X \).

**PSI Game:**

**Setup:** \( X \) runs \( \text{Setup}(1^\lambda) \) to obtain the system parameter list \( \text{Params} \) and the master secret key \( s \). Then \( X \) sends \( \text{Params} \) and \( s \) to the adversary \( A_*(\lambda) \). \( X \) initializes the lists \( P, S, R, D \) and \( V \) with empty list (\( \emptyset \)) respectively.

**Challenge:** \( A_*(\lambda) \) gathers information by querying oracles adaptively. \( X \) correctly simulates the oracles called by \( A_*(\lambda) \) and returns proper values. Eventually, \( A_*(\lambda) \) submits to \( X \) a challenge message \( m^* \) and the identities \( \{ \ID^*_1, \ID^*_1, \ID^*_2, \ID^*_3, \ID^*_4 \} \) of two original singer, two proxy signer and a designated verifier select \( d \) as targets along with the corresponding delegations \( \delta^*_1, \delta^*_2 \) and public keys \( P^*_1, K^*_1, P^*_1, K^*_1, P^*_2, K^*_2, P^*_3, K^*_3, P^*_4, K^*_4 \).

Then \( X \) flips a fair coin \( b \leftarrow \{0, 1\} \), produces a signature \( \sigma^* = \text{GenerateDVP} \text{Sign}( \text{Params}, \ID^*_1, P^*_1, K^*_1, \delta^*_1, P^*_1, K^*_1, s^*_1, c^*_2, \ID^*_2, P^*_2, K^*_2, m^* ) \) and replies to \( A_*(\lambda) \) with \( \sigma^* \).

**Guess:** After receiving \( \sigma^* \), \( A_*(\lambda) \) can continue to issue oracle queries. In the end, \( A_*(\lambda) \) returns his guess \( b' \in \{0, 1\} \).

We say that \( A_*(\lambda) \) wins the game or succeeds in the game if \( b' = b \) and the following boolean expressions are all true:

\[
(1) ((\ID^*_1, K^*_1) \not\in P \lor (\ID^*_D, P^*_D) \not\in S \cup R )
\]
(2) \((\text{ID}^x_{\Omega}, \text{P}^x_{\Omega}, K^x_{\Omega}, \delta^x_{\Omega}, \text{ID}^\delta_{\Omega}, \text{P}^\delta_{\Omega}, K^\delta_{\Omega}, \text{ID}_D^x, \text{P}_D^x, K_D^x, m^x, \sigma^x) \in V \land \)

\((\text{ID}^x_{\Omega}, \text{P}^x_{\Omega}, K^x_{\Omega}, \delta^x_{\Omega}, \text{ID}^\delta_{\Omega}, \text{P}^\delta_{\Omega}, K^\delta_{\Omega}, \text{ID}_D^x, \text{P}_D^x, K_D^x, m^x, \sigma^x) \in V \). 

We define the *advantage* of \( \Lambda_1(\lambda) \) in the game as \( |Pr[\Lambda_1(\lambda) \text{ wins the PSI game}] – 1/2| \).

A CLDVPS scheme \( \Sigma(\lambda) \) is said to satisfy *Privacy of signer’s identity* if the advantage of any PPT trusted adversary \( \Lambda_1(\lambda) \) against the scheme in the above game is negligible in \( \lambda \).

Inspired from the works of Huang et al. [20], Tian et al. [57] and Hu et al. [17] in which employ some similar definitions of the non-delegatability to one proposed by Lipmaa et al [40], and the work of Rastegari et al. [49] using a relaxed version by Tian et al. [55], we define the non-delegatability for CLDVPS as follows.

**Definition 9** (Non-delegatability). Let us consider the following interactive game between an adversary \( \Lambda_1(\lambda) \in \{ \Lambda_1(\lambda), \Lambda_H(\lambda) \} \) against a CLDVPS scheme \( \Sigma(\lambda) \) and a challenger \( X \).

**NDS Game:**

**Setup:** \( X \) runs \( \text{Setup}(\lambda) \) to obtain the system parameter list \( \text{Params} \) and the master secret key \( s \). \( X \) sends \( \text{Params} \) to the adversary \( \Lambda_1(\lambda) \). If \( \Lambda_1(\lambda) \) is \( \Lambda_1(\lambda) \), \( X \) also passes \( s \) to the adversary \( \Lambda_1(\lambda) \). Finally, \( X \) initializes the lists \( \text{P} , \text{S} , \text{R} , \text{D} \) and \( \text{V} \) with empty list (\( \emptyset \)) respectively.

**Select:** \( \Lambda_1(\lambda) \) gathers information by querying oracles allowed to him adaptively. \( \Lambda_1(\lambda) \) correctly simulates the oracles called by \( \Lambda_1(\lambda) \) and returns proper values. Eventually, \( \Lambda_1(\lambda) \) submits to \( X \) the identity \( \text{ID}_P^x \) and full public key \( (\text{P}_P^x, K_P^x) \) (or \( \text{ID}_D^x, \text{P}_D^x, K_D^x \)) of a proxy signer (or a designated verifier) selected as targets, which satisfy that \((\text{ID}_P^x, K_P^x) \notin \text{P} \) or \((\text{ID}_P^x, \text{P}_P^x) \notin \text{S} \cup \text{R} \) (or, \((\text{ID}_D^x, K_D^x) \notin \text{P} \) or \((\text{ID}_D^x, \text{P}_D^x) \notin \text{S} \cup \text{R} \)) when \( \Lambda_1(\lambda) \) is \( \Lambda_1(\lambda) \), and \((\text{ID}_P^x, \text{P}_P^x) \notin \text{S} \) (or \((\text{ID}_D^x, \text{P}_D^x) \notin \text{S} \)) when \( \Lambda_1(\lambda) \) is \( \Lambda_H(\lambda) \).

**Challenge:** \( X \) sends to \( \Lambda_1(\lambda) \) a challenge identity \( \text{ID}_D^x \) and full public key \( (\text{P}_D^x, K_D^x) \) (or \( \text{ID}_P^x, \text{P}_P^x, K_P^x \)) of a designated verifier (or a proxy signer), an identity \( \text{ID}_O^x \) and full public key \( (\text{P}_O^x, K_O^x) \) of an original signer, a valid delegation \( \delta_O^x \) and a challenge message \( m^x \), and asks \( \Lambda_1(\lambda) \) to submit a valid signature on \((\text{ID}_O^x, \text{P}_O^x, K_O^x, \delta_O^x, \text{ID}_P^x, \text{P}_P^x, K_P^x, \text{ID}_D^x, \text{P}_D^x, K_D^x, m^x) \) to \( X \). \( \Lambda_1(\lambda) \) can issue queries to the oracles allowed to him adaptively, but cannot query the tuple \((\text{ID}_O^x, \text{P}_O^x, K_O^x, \delta_O^x, \text{ID}_P^x, \text{P}_P^x, K_P^x, \text{ID}_D^x, \text{P}_D^x, K_D^x, m^x) \) to the oracle.
O_{GetDVPSign} and cannot query to the oracle O_{VerifyDVPSign} on input ID_D', P_D', K_D' if a designated verifier has been selected by A(α) in the Select stage. X simulates the oracles called by A(α). Eventually, A(λ) submits a valid signature (or a valid simulated signature) on (ID_O', P_O', K_O', s_O', ID_P, P_P, K_P, ID_D, P_D, K_D, m*) to X if the following boolean expression is true:

((ID_D', K_D') ∈ P \cup (ID_D', P_D') \notin S \cup R) \land ((ID_P', K_P') \notin P \cup (ID_P', P_P') \notin S \cup R),

when A(α) is A₁(α), and the following boolean expression is true:

(ID_D', P_D') \notin S \land (ID_P', P_P') \notin S,

when A(α) is Aᵩ(α).

X can repeat the steps in this stage and gather valid signatures enough for the next stage.

Extract: X tries to compute correctly the full private key (s_P', c_P') (or s_D', c_D') of the proxy signer (or the designated verifier) selected by A(α) in the Select stage, using only his replies to the queries of A(α) in the previous stages and the valid signatures returned by A(α) in the Challenge stage.

A CLDVS scheme Σ(λ) is said to satisfy Non-delegatability if there exists a PPT (in λ) algorithm X that can succeed in the above game with a non-negligible probability (in λ) against any PPT (in λ) adversary A(α) which can generate a valid signature with a non-negligible probability (in λ) in the Challenge stage.

As Tian et al. [56] mentioned, although the non-delegatability property has been a subject of many literatures, it is a controversial concept as it may be undesirable in some applications, so that we will no longer discuss the non-delegatability in this work.

4. Our CLDVS scheme

In this section, we propose a CLDVS scheme using bilinear pairings in conformity with Definition 3, and consider the correctness and the efficiency of it.

Our scheme consists of the following eight algorithms:

1. (Params, s) ← Setup(1^λ): KGC takes the security parameter λ ∈ 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 \( \Phi_p \) of order p, an elliptic curve \( E/\Phi_p \) defined
by \( E : Y^2 = X^3 + aX + b \) over \( \Phi_p \), a cyclic additive subgroup \( \Gamma \) of prime order \( q \geq 2^k \) on \( E/\Phi_p \), a generator \( G \) of \( \Gamma \), a cyclic multiplicative group \( \Gamma' \) of the same order \( q \) (e.g., \( Z_q^* \)) and an admissible bilinear map \( \hat{\cdot} : \Gamma \times \Gamma \rightarrow \Gamma' \).

b. \( s \leftarrow_R Z_q^* ; P_{KGC} \leftarrow [s]G \).

c. Chooses three cryptographic hash functions \( H_i : \{0,1\}^* \rightarrow Z_q^* \), \( H_2 : \{0,1\}^* \rightarrow \Gamma \), \( H_3 : \{0,1\}^* \rightarrow \Gamma \).

d. \( B_{Param} = a_\parallel b \parallel G \parallel P_{KGC} \). (We assume that the binary representaions of \( a, b, G \) and \( P_{KGC} \) are obtained in a trivial way.)

e. Publishes \( Param = \{ a, b, p, q, G, P_{KGC}, H_1, H_2, H_3 \} \) and keeps \( s \) secret

(2) \((P_U, s_U) \leftarrow \text{GenerateUserKeys}(Param, ID_U)\): The user \( U \) with the identity \( ID_U \) sets his public key \( P_U \) and secret key \( s_U \) as follows:

a. \( s_U \leftarrow_R Z_q^* ; P_U \leftarrow [s_U]G ; \text{return} (P_U, s_U) \).

(3) \((K_U, c_U) \leftarrow \text{GeneratePartialKeys}(Param, s, ID_U, P_U)\): On input \( Param \), 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. \( h_U \leftarrow H_1 (B_{Param} \parallel ID_U) \).

b. \( k_U \leftarrow_R Z_q^* ; K_U \leftarrow [k_U]G \).

c. \( c_U \leftarrow (k_U + s \cdot h_U) \mod q \).

d. \text{return} \((K_U, c_U) \). (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:

\[ [c_U]G \doteq K_U + [H_1 (B_{Param} \parallel ID_U)]P_{KGC} \].

(4) \( \delta_O \leftarrow \text{GenerateDelegation}(Param, ID_O, P_O, K_O, s_O, c_O, w_O)\): The original signer \( O \) outputs the delegation \( \delta_O \) on the warrant \( w_O \) as follows:

a. \( t_O \leftarrow (s_O + c_O) \mod q \).

b. \( r_O \leftarrow_R Z_q^* ; R_O \leftarrow [r_O]G \).

21
c. \( H_2 \leftarrow H_2( w_o \parallel \text{ID}_O \parallel P_O \parallel K_O ). \) (We assume that the binary representaions of \( P_O \) and \( K_O \) are obtained in a trivial way.)

d. \( \Delta_O \leftarrow [r_O] P_{\text{KGC}}+[t_O] H_2. \)

e. \( \delta_O := (w_O, R_O, \Delta_O); \) \text{return} \( \delta_O. \)

(5) \textbf{True/False} \leftarrow \text{VerifyDelegation}( \text{Params}, \text{ID}_O, P_O, K_O, \delta_O): \) Any verifier, including the proxy signer specified by the delegation \( \delta_O \), can verify \( \delta_O \) as follows:

a. \( h_o \leftarrow H_1( B_{\text{Params}} \parallel \text{ID}_O ). \)

b. \( T_O \leftarrow P_O + K_O + [h_0] P_{\text{KGC}}. \)

c. \( (w_O, R_O, \Delta_O) := \delta_O. \) \( \delta_O \) is parsed as \( (w_O, R_O, \Delta_O). \)

d. \( H_2 \leftarrow H_2( w_O \parallel \text{ID}_O \parallel P_O \parallel K_O ). \)

e. \text{return} \( \hat{e}( \Delta_O , G ) \hat{=} \hat{e}( P_{\text{KGC}}, R_O ) \cdot \hat{e}( H_2, T_O ). \)

(A verifier accepts \( \delta_O \) if \text{VerifyDelegation} returns \textbf{True}, or rejects otherwise.)

(6) \( \sigma_m \leftarrow \text{GenerateDVPSign}( \text{Params}, \text{ID}_O, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, s_P, c_P, \text{ID}_D, P_D, K_D, m): \) If the proxy signer \( P \) specified by the delegation \( \delta_O \) accepts the delegation \( \delta_O \), he returns a proxy signature \( \sigma_m \) on the message \( m \) as follows:

a. \( t_P \leftarrow (s_P + c_P) \mod q. \)

b. \( h_D \leftarrow H_1( B_{\text{Params}} \parallel \text{ID}_D ). \)

c. \( T_D \leftarrow P_D + K_D + [h_D] P_{\text{KGC}}. \)

d. \( r_P \leftarrow_{\text{R}} Z_q^*, \ R_p \leftarrow [r_P] G. \)

e. \( H_3 \leftarrow H_3( m \parallel \delta_O \parallel \text{ID}_P \parallel P_P \parallel K_P \parallel \text{ID}_D \parallel P_D \parallel K_D ). \) (We assume that the binary representaions of \( \delta_O, P_P, K_P, P_D \) and \( K_D \) are obtained in a trivial way.)

f. \( v_m \leftarrow \hat{e}( r_P ] P_{\text{KGC}} + [t_P] H_3, T_D ). \)

g. \( \sigma_m := (R_P, v_m); \) \text{return} \( \sigma_m. \)

(7) \textbf{True/False} \leftarrow \text{VerifyDVPSign}( \text{Params}, \text{ID}_O, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, \text{ID}_D, P_D, K_D, s_D, c_D, m, \sigma_m): \) Having all inputs, the designated verifier \( D \) verifies the proxy signature \( \sigma_m \) on the message \( m \) as
follows:

a. if VerifyDelegation( Params, ID_O, P_O, K_O, δ_O) = False then return False.

b. if (the message m does not conform to the delegation δ_O) then return False.

c. t_D ← (s_D + c_D) mod q.

d. h_p ← H_1( B_{Params} || ID_p).

e. T_p ← P_p + K_p + [h_p] P_{KGC}.

f. (R_p, v_m) := σ_m. (σ_m is parsed as (R_p, v_m).)

g. H_3 ← H_3( m || δ_O || ID_p || P_p || K_p || ID_D || P_D || K_D)

h. return (v_m = \frac{e^*}{e}( R_p, P_{KGC})^{1_p} \cdot e^*(H_3, T_p)^{1_p}).

(The designated verifier D accepts σ_m if VerifyDVPSign returns True, or rejects otherwise.

(8) σ_m^S ← SimulateDVPSign( Params, ID_O, P_O, K_O, δ_O, ID_p, P_p, K_p, ID_D, P_D, K_D, s_D, c_D, m):

Having all inputs, the designated verifier D returns a simulated signature σ_m^S on the message m as follows:

a. t_D ← (s_D + c_D) mod q.

b. h_p ← H_1( B_{Params} || ID_p).

c. T_p ← P_p + K_p + [h_p] P_{KGC}.

d. r_D ← R \cdot Z_q^*; R_D ← [r_D] G.

e. H_3 ← H_3( m || δ_O || ID_p || P_p || K_p || ID_D || P_D || K_D).

f. v_m ← e^*( R_D, P_{KGC})^{1_p} \cdot e^*(H_3, T_p)^{1_p}.

g. σ_m^S := (R_D, v_m); return σ_m^S.

Note that using the binding technique in [1], we can easily convert our scheme to the one which achieves Girault’s trust level 3 [13], by concatenating P_U, P_O, P_D or P_p to the inputs of H_1 in the step (a) of GeneratePartialKeys, the step (a) of VerifyDelegation, the step (b) of GenerateDVPSign, the step (d) of VerifyDVPSign and the step (b) of SimulateDVPSign respectively.

Now, we consider the correctness of the proposed scheme.

The correctness of the algorithm VerifyDelegation verifying a delegation is proved as follows:
\[\hat{e}( \Delta_0, G) = \hat{e}([r_0] P_{KGC} + [t_0] H_2, G)\]

\[= \hat{e}([r_0] P_{KGC}, G) \cdot \hat{e}([t_0] H_2, G)\]

\[= \hat{e}(P_{KGC}, [r_0] G) \cdot \hat{e}(H_2, [t_0] G)\]

\[= \hat{e}(P_{KGC}, R_0) \cdot \hat{e}(H_2, [(s_0 + c_0) \mod q] G)\]

\[= \hat{e}(P_{KGC}, R_0) \cdot \hat{e}(H_2, [s_0] G + [c_0] G)\]

\[= \hat{e}(P_{KGC}, R_0) \cdot \hat{e}(H_2, P_0 + [k_0 + s \cdot h_0] \mod q) G)\]

\[= \hat{e}(P_{KGC}, R_0) \cdot \hat{e}(H_2, P_0 + K_0 + [s \cdot h_0] \mod G)\]

\[= \hat{e}(P_{KGC}, R_0) \cdot \hat{e}(H_2, P_0 + K_0 + [h_0] P_{KGC}).\]

The correctness of the algorithm VerifyDVPSign verifying a proxy signature is proved as follows:

\[v_m = \hat{e}([r_p] P_{KGC} + [t_p] H_3, T_D)\]

\[= \hat{e}([r_p] P_{KGC}, [t_D] G) \cdot \hat{e}([t_p] H_3, [t_D] G)\]

\[= \hat{e}([r_p] P_{KGC}, G)^{t_D} \cdot \hat{e}([t_p] H_3, G)^{t_D}\]

\[= \hat{e}(P_{KGC}, [r_p] G)^{t_D} \cdot \hat{e}(H_3, [t_p] G)^{t_D}\]

\[= \hat{e}(P_{KGC}, R_p)^{t_D} \cdot \hat{e}(H_3, T_p)^{t_D}.\]

Next, we compare the efficiency of our scheme with some existed DVS schemes in Table 1 where \(H_T, P_T, S_T\) and \(A_T\) 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 respectively, \(G_\ell\) denotes the bit length of a point in the elliptic curve group \(\Gamma_\ell\) and \(Z_\ell\) means the length of the integer \(q\), i.e., \(\log q\). According to [7, 32], \(H_T\) is rather longer than \(P_T\) by a narrow margin, \(P_T\) is several times longer than \(S_T\), and \(S_T\) is several tens to hundreds times longer than \(A_T\). We ignore costless operations such as one-way hash operation and operations on integers and the multiplicative group \(\Gamma'\), which are much cheaper than the point addition. \(G_\ell\) is several times longer than \(Z_\ell\).

Table 1 shows that the efficiency of our scheme is not inferior to the schemes in [17, 5].
We construct an algorithm \( CDH \) i.e. that modeled as random oracles which is polynomial in \( (\lambda) \). Let \( N_U \) and \( N_{U\text{Key}} \) be respectively the number of queries to \( O_{\text{CreateUser}} \) that \( A(\lambda) \) makes in sEUF 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 \( \lambda \) and do not exceed \( t(\lambda) \).

Firstly, when \( A(\lambda) \) is a type I adversary \( A_1(\lambda) \), we construct an algorithm \( X_i \) that uses \( A_1(\lambda) \) to solve the CDH problem in the group \( G_1 \) as follows:

**Algorithm** \( X_i \):

**Input:** An instance of the CDH problem, given by \( q \cdot G_1, G, [u]G, [v]G \in \Gamma \) where \( u, v \in Z_q^* \) are unknown.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation of keys</td>
<td>( 1H_T + IS_T )</td>
<td>( IS_T )</td>
<td>( 1H_T + 2S_T )</td>
<td>( 2S_T )</td>
</tr>
<tr>
<td>Generation of delegation</td>
<td>( 4H_T + 1S_T + 1A_T )</td>
<td>( 1H_T + 1S_T )</td>
<td>( - )</td>
<td>( 1H_T + 3S_T + 1A_T )</td>
</tr>
<tr>
<td>Verification of delegation</td>
<td>( 2H_T + 2P_T + 1A_T )</td>
<td>( 1H_T + 2P_T )</td>
<td>( - )</td>
<td>( 1H_T + 3P_T + 1S_T + 2A_T )</td>
</tr>
<tr>
<td>Signing</td>
<td>( 1P_T )</td>
<td>( 1H_T + 1P_T + 3S_T + 1A_T )</td>
<td>( 2H_T + 1P_T + 3S_T + 1A_T )</td>
<td>( 1H_T + 1P_T + 4S_T + 3A_T )</td>
</tr>
<tr>
<td>Verification of signature</td>
<td>( 1P_T )</td>
<td>( 2H_T + 2P_T + 2S_T )</td>
<td>( 3H_T + 1P_T + 3S_T + 1A_T )</td>
<td>( 1H_T + 2P_T + 3S_T + 2A_T )</td>
</tr>
<tr>
<td>Key size</td>
<td>( 1G_T + 1Z_T )</td>
<td>( 1G_T )</td>
<td>( 1G_T + 2Z_T )</td>
<td>( 2G_T + 2Z_T )</td>
</tr>
<tr>
<td>Delegation size</td>
<td>( 1G_T )</td>
<td>( 1G_T )</td>
<td>( 2G_T )</td>
<td>( 2G_T )</td>
</tr>
<tr>
<td>Signature size</td>
<td>( 1Z_T )</td>
<td>( 2G_T )</td>
<td>( 1G_T + 2Z_T )</td>
<td>( 1G_T + 1G_T' )</td>
</tr>
</tbody>
</table>

### 5. Security Analysis of the proposed CLDVPS scheme

In this section, we discuss only the strong unforgeability and the non-transferability of our CLDVPS scheme. We use the notations and symbols in Section 4 as it is.

**Theorem 1.** In the random oracle model, if there exists a PPT (in \( \lambda \)) adversary \( A(\lambda) \) that can impersonates a original signer and forge a valid delegation to win sEUF Game against our CLDVPS scheme with a non-negligible probability of success (in \( \lambda \)), then the CDH problem in the group \( G_1 \) can be solved in polynomial time (in \( \lambda \)) with a non-negligible probability (in \( \lambda \)).

**Proof.** Suppose that \( A(\lambda) \) succeeds in sEUF Game with a non-negligible probability \( \varepsilon(\lambda) \) in time \( t(\lambda) \) which is polynomial in \( \lambda \), and in sEUF Game, \( A(\lambda) \) makes \( N_{H_1}, N_{H_2}, N_{H_3} \) queries to \( H_1, H_2 \) and \( H_3 \) modeled as random oracles respectively. Let \( N_U \) and \( N_{U\text{Key}} \) be respectively the number of queries to \( O_{\text{CreateUser}} \) that \( A(\lambda) \) makes in sEUF 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 \( \lambda \) and do not exceed \( t(\lambda) \).

Firstly, when \( A(\lambda) \) is a type I adversary \( A_1(\lambda) \), we construct an algorithm \( X_i \) that uses \( A_1(\lambda) \) to solve the CDH problem in the group \( G_1 \) as follows:

**Algorithm** \( X_i \):

**Input:** An instance of the CDH problem, given by \( q \cdot G_1, G, [u]G, [v]G \in \Gamma \) where \( u, v \in Z_q^* \) are unknown.
\textbf{Output:} The solution \([uv]G \in \Gamma\) of the above instance.

1. Select four indices \(0 \leq I_0 \leq N_U, 0 \leq I_1 \leq N_U, 0 \leq I_2 \leq N_{H_2}, 0 \leq I_3 \leq N_{H_3}\) and a boolean value \(B \in \{0,1\}\) randomly.

2. Simulate a challenger in sEUF Game as follows:

\textbf{Setup:} Let \(s \leftarrow_r Z_q^* , P_{\text{KGC}} \leftarrow [s]G\), \(\text{Params} = \{a, b, p, q, G, P_{\text{KGC}}\}\), \(B_{\text{Params}} := a \parallel b \parallel G \parallel P_{\text{KGC}}\) and send \(\text{Params}\) to \(A_1(\lambda)\). Initialize the lists \(P, S, R\) and \(D\) with empty list \(\emptyset\) respectively, and prepare a list \(H_1\) of 2-tuples, two lists \(H_2\) and \(H_3\) of 3-tuples and a list \(L\) of tuples in form of \((ID_U, P_U, K_U, s_U, c_U)\). Choose randomly \(h^*_0 \in Z_q^*\) and \(h^*_3 \in Z_q^*\). Let \(\text{Count}_O := 0\).

\textbf{Attack:} Answer \(A_1(\lambda)\)'s queries to oracles as follows:

- **\(H_1(x)\):** If there exists \(h_U \in Z_q^*\) such that \((x, h_U) \in H_1\), then return \(h_U\). If \(|H_1| = I_0 - 1\), then put \((x, h^*_0)\) in \(H_1\) and return \(h^*_0\). Otherwise, choose \(h_U \in Z_q^*\) randomly. If \(h_U = h^*_0\), then abort. (This case is called \(\text{Event E}_1\).) Otherwise, put \((x, h_U)\) in \(H_1\) and return \(h_U\).

- **\(H_2(y)\):** If there exists \(H_2 \in \Gamma_1\) such that \((y, H_2, *) \in H_2\) (where * means for an arbitrary value), then return \(H_2\). If \(|H_2| = I_1 - 1\), then put \((y, [v]G, \bot)\) in \(H_2\) and return \([v]G\). Otherwise, choose \(h^*_2 \in Z_q^*\) randomly, let \(H_2 \leftarrow [h^*_2]G\), put \((y, H_2, h_2)\) in \(H_2\) and return \(H_2\).

- **\(H_3(z)\):** If there exists \(H_3 \in \Gamma_1\) such that \((z, H_3, *) \in H_3\) (where * means for an arbitrary value), then return \(H_3\). If \(|H_3| = I_2 - 1\) then put \((z, [h^*_3]G, h^*_3)\) in \(H_3\) and return \([h^*_3]G\). Otherwise, choose \(h^*_3 \in Z_q^*\) randomly. If \(h_3 = h^*_3\), then abort. (This case is called \(\text{Event E}_3\).) Otherwise, put \((z, [h^*_3]G, h_3)\) in \(H_3\) and return \([h^*_3]G\).

- **\(O_{\text{CreateUse}}(ID_U)\):** Let \(h_U \leftarrow H_1(B_{\text{Params}} \parallel ID_U)\) and respond differently in the following cases:
  a. If there exists \((ID_U, *, *, *, *) \in L\) (where * means for an arbitrary value), then search \(L\) for the latest item \((ID_U, *, K_U, *, c_U)\) indexed by \(ID_U\) to get \(K_U\) and \(c_U\). Otherwise, consider the following cases:
    - If \(B \equiv 0\) and \(h_U = h^*_0\), then let \(K_U \leftarrow [u]G\) and \(c_U := \bot\).
- Else, choose \( k_U \in Z_q^* \) randomly, let \( K_U \leftarrow [k_U]G \) and \( c_U \leftarrow (k_U + s \cdot h_U \mod q) \).

b. If \( B = 1 \) and \( h_U = h_0^* \), then let \( \text{Count}_O \leftarrow \text{Count}_O + 1 \). If \( \text{Count}_O = \emptyset \), then let \( P_U \leftarrow [u]G \), put 

\[
(ID_U , P_U , K_U , c_U) \in L \text{ and return } (P_U , K_U).
\]

Otherwise, go forward.

c. Choose \( s_U \in Z_q^* \) randomly, let \( P_U \leftarrow [s_U]G \), put 

\[
(ID_U , P_U , K_U , s_U , c_U) \in L \text{, and return } (P_U , K_U).
\]

- \( \text{O}_{\text{GetSecretKey}}(ID_U , P_U) \): Search \( L \) for the item \((ID_U , P_U , \ast , s_U , \ast)\) indexed by \((ID_U , P_U)\) to get \( s_U \).

If the search succeeds, put \((ID_U , P_U)\) in \( S \) and return \( s_U \). Otherwise, return \( \bot \).

- \( \text{O}_{\text{GetPartialKey}}(ID_U , K_U) \): Search \( L \) for the item \((ID_U , \ast , K_U , \ast , c_U)\) indexed by \((ID_U , K_U)\) to get \( c_U \).

If the search succeeds, put \((ID_U , K_U)\) in \( P \) and return \( c_U \). Otherwise, return \( \bot \).

- \( \text{O}_{\text{GetPublicKey}}(ID_U) \): Search \( L \) for the latest item \((ID_U , P_U , K_U , \ast , \ast)\) indexed by \( ID_U \) to get \( P_U \) and \( K_U \). If the search succeeds, return \((P_U , K_U)\). Otherwise, return \( \bot \).

- \( \text{O}_{\text{ReplaceUseKey}}(ID_U , P'_U , s'_U) \): Search \( L \) for the latest item \((ID_U , \ast , K_U , \ast , c_U)\) indexed by \( ID_U \) to get \( K_U \) and \( c_U \). If the search succeeds, put \((ID_U , P'_U , K_U , s'_U , c_U)\) in \( L \), put \((ID_U , P'_U)\) in \( R \) and return “OK”. Otherwise, return \( \bot \).

- \( \text{O}_{\text{GetDelegate}}(ID_O , P_O , K_O , w_O) \): Search \( D \) for the item \((ID_O , P_O , K_O , w_O , \delta_O)\) indexed by \((ID_O , P_O , K_O , w_O)\) to get \( \delta_O \). If the search succeeds, return \( \delta_O \). Otherwise, follow the next steps:

  a. let \( H_2 \leftarrow H_2(\text{w}_O \parallel ID_O \parallel P_O \parallel K_O) \).

  b. Search \( H_2 \) for the item \((\text{w}_O \parallel ID_O \parallel P_O \parallel K_O \parallel H_2) \) indexed by \((\text{w}_O \parallel ID_O \parallel P_O \parallel K_O \parallel H_2) \) to get \( h_2 \). If \( h_2 = \bot \), then abort. (This case is called Event D.) Otherwise, go forward.

c. Search \( H_1 \) for the item \((B_{\text{Params}} \parallel ID_O , h_0)\) indexed by \( B_{\text{Params}} \parallel ID_O \) to get \( h_0 \). If the search does not succeed, return \( \bot \).

d. Search \( L \) for the latest item \((ID_O , P_O , K_O , \ast , \ast)\) indexed by \( ID_O \) to get \( P_O , K_O \in \Gamma \). If the search does not succeed, return \( \bot \).

e. Choose \( r_O \in Z_q^* \) randomly, let \( R_O \leftarrow [r_O]G \).
f. Let $\Delta_O \leftarrow \{ r_O \} P_{KGC}^+ [h_3] (P_O + K_O + [h_O] P_{KGC})$.

g. Let $\delta_O := (w_O, R_O, \Delta_O)$, put $(ID_O, P_O, K_O, w_O, \delta_O)$ in $D$ and return $\delta_O$.

- $O_{GetDVPSign}(ID_O, P_O, K_O, \delta_O, ID_P, P_P, K_P, ID_D, P_D, K_D, m)$: Parse $\delta_O$ as $(w_O, R_O, \Delta_O)$ and check whether all input parameters conform to $w_O$. If not, return $\bot$. If the check is passed, follow the next steps:

  a. Search $H_1$ for the item $(B_{Params}, ID_O, h_O)$ indexed by $B_{Params}$ to get $h_O$. If the search does not succeed, return $\bot$.

  b. Let $H_2 \leftarrow H_2 (w_O \parallel ID_O \parallel P_O \parallel K_O)$ and $T_O \leftarrow P_O + K_O + [h_O] P_{KGC}$. If $\hat{\epsilon} (\Delta_O, G) \neq \hat{\epsilon} (P_{KGC}, R_O) \hat{\epsilon} (H_2, T_O)$, then return $\bot$.

  c. Search $H_1$ for the item $(B_{Params}, ID_P, h_P)$ indexed by $B_{Params}$ to get $h_P$. If the search does not succeed, return $\bot$.

  d. If there does not exist $(ID_P, P_P, K_P, *, *) \in L$ (where * means for an arbitrary value) such that $P_O \in \Gamma$ and $K_O \in \Gamma$; then return $\bot$.

  e. Let $H_3 \leftarrow H_3 (m \parallel \delta_O \parallel ID_P \parallel P_P \parallel K_P \parallel ID_D \parallel P_D \parallel K_D)$.

  f. Search $H_3$ for the item $(m \parallel \delta_O \parallel ID_P \parallel P_P \parallel K_P \parallel ID_D \parallel P_D \parallel K_D \parallel H_3, h_3)$ indexed by $(m \parallel \delta_O \parallel ID_P \parallel P_P \parallel K_P \parallel ID_D \parallel P_D \parallel K_D \parallel H_3)$ to get $h_3$.

  g. If $h_3 = h_3^*$, then abort. (This case is called Event $S$.)

  h. Search $H_1$ for the item $(B_{Params}, ID_D, h_D)$ indexed by $B_{Params}$ to get $h_D$. If the search does not succeed, return $\bot$.

  i. Let $T_P \leftarrow P_P + K_P + [h_P] P_{KGC}$ and $T_D \leftarrow P_D + K_D + [h_D] P_{KGC}$.

  j. Choose $r_P \in Z_q^*$ randomly, let $R_P \leftarrow [r_P] G$ and $v_m \leftarrow \hat{\epsilon} ([r_P] P_{KGC} + [h_3] T_P, T_D)$.

  k. Let $\sigma_m := (R_P, v_m)$ and return $\sigma_m$.

- $O_{VerifyDVPSign}(ID_O, P_O, K_O, \delta_O, ID_P, P_P, K_P, ID_D, P_D, K_D, m, \sigma_m)$: Parse $\delta_O$ as $(w_O, R_O, \Delta_O)$ and check whether all input parameters conform to $w_O$. If not, return $\text{False}$. If the check is passed, follow the next steps:
a. Search $H_1$ for the item $(B_{Params} \parallel ID_O, h_O)$ indexed by $B_{Params} \parallel ID_O$ to get $h_O$. If the search does not succeed, return $\text{False}$. 

b. Let $H_2 \leftarrow H_2 (w_O \parallel ID_O \parallel P_O \parallel K_O)$ and $T_O \leftarrow P_O + K_O + [h_O] P_{KGC}$. If $\hat{e} (\Delta_O, G) \neq \hat{e} (P_{KGC}, R) \hat{e} (H_2, T_O)$ then return $\text{False}$. 

c. Parse $\sigma_m$ as $(R_P, v_m)$ and let $H_3 \leftarrow H_3 (m \parallel \delta_O \parallel ID_P \parallel P_P \parallel K_P \parallel ID_D \parallel P_D \parallel K_D)$.

d. Search $H_3$ for the item $(m \parallel \delta_O \parallel ID_P \parallel P_P \parallel K_P \parallel ID_D \parallel P_D \parallel K_D, H_3, h_3)$ indexed by $(m \parallel \delta_O \parallel ID_P \parallel P_P \parallel K_P \parallel ID_D \parallel P_D \parallel K_D, H_3)$ to get $h_3$. 

e. Search $H_1$ for the item $(B_{Params} \parallel ID_P, h_P)$ indexed by $B_{Params} \parallel ID_P$ to get $h_P$. If the search does not succeed, return $\text{False}$. 

f. Search $H_1$ for the item $(B_{Params} \parallel ID_D, h_D)$ indexed by $B_{Params} \parallel ID_D$ to get $h_D$. If the search does not succeed, return $\text{False}$. 

g. Let $T_P \leftarrow P_P + K_P + [h_P] P_{KGC}$ and $T_D \leftarrow P_D + K_D + [h_D] P_{KGC}$. 

h. If $v_m = \hat{e} ([x]R_P + [h_1] T_P, T_D)$, then return $\text{True}$. Otherwise, return $\text{False}$. 

Forgery: Wait until $A_1(\lambda)$ outputs a tuple $(ID_O, P_O^*, K_O^*, \delta_O^*, ID_P^*, P_P^*, K_P^*, ID_D^*, P_D^*, K_D^*, m^*, \sigma_m^*)$ as the final result of the game. 

3. Check $A_1(\lambda)$ ’s final result and output the solution of the above instance (i.e., the input) as follows:

(1) If the following boolean expression is true, then abort. (This case is called $\text{Event } R_1$.)

$$h_O \neq H_1 (B_{Params} \parallel ID_O^*) \lor [v]G \neq H_2 (w_O^* \parallel ID_O^* \parallel P_O^* \parallel K_O^*) \lor [h_3^*] G \neq H_3 (m^* \parallel \delta_O^* \parallel ID_P^* \parallel P_P^* \parallel K_P^* \parallel ID_D^* \parallel P_D^* \parallel K_D^*).$$

(2) Parse $\delta_O^*$ as $(w_O^*, R_O^*, \Delta_O^*)$. If the following boolean expression is true, then abort since $\delta_O^*$ must be forged by $A_1(\lambda)$ from the assumption of the theorem. (This case is called $\text{Event } R_2$.)

$$(((ID_O^*, K_O^*) \in P \land (ID_O^*, P_O^*) \in S \cup R) \lor (ID_O^*, P_O^*, K_O^*, w_O^*, \delta_O^*) \in D).$$

(3) If $O_{\text{verifyDVPS}} (ID_O^*, P_O^*, K_O^*, \delta_O^*, ID_P^*, P_P^*, K_P^*, ID_D^*, P_D^*, K_D^*, m^*, \sigma_m^*) = \text{False}$, then then abort. (This case is called $\text{Event } R_3$.)

(4) Search $L$ for the item $(ID_O^*, P_O^*, K_O^*, s_O^*, c_O^*)$ indexed by $(ID_O^*, P_O^*, K_O^*)$ to get $s_O^*$ and $c_O^*$. If the search
ch does not succeed, then abort. (This case is called Event $R_4$.)

(5) If $B|=0$, then return $(\Delta_0^s-[s] R_0^s -[c_o^s] G)$. If $B|=1$, then return $(\Delta_0^s-[s] R_0^s -[c_o^s] [v] G)$. (The end of the algorithm $X_i$)

Now, we analyze the time complexity of $X_i$. The time complexity of $X_i$ is dominated by the scalar multiplications, additions and pairing in $\Gamma$, operations in $\mathbb{Z}_q^*$ and searches on lists which are performed in the simulations of oracles. All operands for the above operations in $\Gamma$ and $\mathbb{Z}_q^*$ have at most a polynomial size in $\lambda$ and we can assume that the computational cost of any operation in $X_i$ is less than some polynomial $p(\lambda)$. On the other hand, the sizes of all lists in oracle simulations of $X_i$ do not go beyond the number of all queries by $A_1(\lambda)$, which is not larger than $t(\lambda)$. It results that the time of a search on any list in $X_i$ is also not longer than $t(\lambda)$. Therefore, the cost of answering an oracle query in $X_i$ is also less than $O(p(\lambda)+t(\lambda))$, and we conclude that the total time complexity of $X_i$ is $O(t(\lambda)p(\lambda)+t^2(\lambda))$ which is also polynomial in $\lambda$.

It is obvious that if $X_i$ does not abort, then $A_1(\lambda)$’s view in the simulated game with $X_i$ is indistinguishable from its view in the real attack.

Next, we show that $X_i$’s output is the solution for the instance of the CDH problem, given as input of $X_i$, if $X_i$ does not abort.

Assume that $X_i$ does not abort and outputs some result.

If $B|=0$, then $K_0^s=[u]G, c_o^s=\perp, s_0^s \in \mathbb{Z}_q^* \text{ and } P_0^s=[s_0^s] G$ from the step (a) and (c) of $O_{CreateUser}$. In this case, $X_i$’s output is $(\Delta_0^s-[s] R_0^s -[(s_0^s + h_0^s) \mod q] [v] G)$ from the step 3(5) of $X_i$. The validity of $A_1(\lambda)$’s final result (the forged signature) implies the validity of $\delta_0^s$, and we obtain the following equation from the validity of $\delta_0^s$ and the equation $[v] G=H_2(w_o^s \parallel ID_o^s \parallel P_0^s \parallel K_0^s)$:

$$\hat{e}(\Delta_0^s, G) = \hat{e}(P_{KGC}, R_0^s) \cdot \hat{e}(\hat{[v]}G, T_0^s),$$

where $T_0^s = P_o^s + K_o^s + [h_0^s] P_{KGC}$. From the above equation, we can obtain

$$\hat{e}(\Delta_0^s, G) = \hat{e}(P_{KGC}, R_0^s) \cdot \hat{e}([v]G, P_o^s + K_o^s + [h_0^s] P_{KGC})$$
\[
\hat{\epsilon}([s]G) \cdot \hat{\epsilon}([v]G \cdot [s_o^*]G + [u]G + [h_o^*]s)G ) \\
= \hat{\epsilon}([s]G \cdot R_o^*) \cdot \hat{\epsilon}([v]G \cdot ([s_o^* + s \cdot h_o^*] \mod q) G + [u]G) \\
= \hat{\epsilon}([s|R_o^* \cdot G) \cdot \hat{\epsilon}(([s_o^* + s \cdot h_o^*] \mod q) [v]G \cdot G) \cdot \hat{\epsilon}([v]G \cdot [u]G),
\]
and thus,
\[
\hat{\epsilon}([v]G \cdot [u]G) = \hat{\epsilon} (\Delta_o^* - [s]R_o^* - ([s_o^* + s \cdot h_o^*] \mod q) [v]G \cdot G).
\]

From the non-degeneracy of \( \hat{\epsilon} \), we can obtain
\[
[uv]G = \Delta_o^* - [s]R_o^* - ([s_o^* + s \cdot h_o^*] \mod q) [v]G.
\]

Therefore, \( \Delta_o^* - [s]R_o^* - ([s_o^* + s \cdot h_o^*] \mod q) [v]G \) is the solution for the instance given as input of \( X_1 \).

Similarly, If \( B=1 \), then \( P'_o = [u]G, s_o^* = \perp, k_o^* = Z_q, c_o^* = (k_o^* + s \cdot h_o^*) \mod q \) and \( K_o^* = [k_o^*]G \) from the step (b) of \( O_{\text{CreateUser}} \). In this case, \( X_1 \)'s output is \( (\Delta_o^* - [s]R_o^* - [c_o^*]G) \) from the step 3(5) of \( X_1 \). We obtain the following equation from the validity of \( \delta_o^* \) and the equation \( [v]G = H_2( w_o^* \| ID'_o \| P'_o \| K'_o) \):
\[
\hat{\epsilon}(\Delta_o^*G) = \hat{\epsilon}(P_{KG} \cdot R_o^*) \cdot \hat{\epsilon}([v]G \cdot T_o^*).
\]

where \( T_o^* = [u]G + [k_o^*]G + [h_o^*]P_{KG} \). From the above equation, we can obtain
\[
\hat{\epsilon}(\Delta_o^*G) = \hat{\epsilon}(P_{KG} \cdot R_o^*) \cdot \hat{\epsilon}([v]G \cdot [u]G + [k_o^*]G + [h_o^*]P_{KG}) \\
= \hat{\epsilon}([s]G \cdot R_o^*) \cdot \hat{\epsilon}([v]G \cdot [u]G + [k_o^*]G + [h_o^*]s)G ) \\
= \hat{\epsilon}([s]G \cdot R_o^*) \cdot \hat{\epsilon}([v]G \cdot ([k_o^* + s \cdot h_o^*] \mod q) G + [u]G) \\
= \hat{\epsilon}([s]R_o^* \cdot G) \cdot \hat{\epsilon}([c_o^*]G \cdot G) \cdot \hat{\epsilon}([v]G \cdot [u]G),
\]
and thus,
\[
\hat{\epsilon}([v]G \cdot [u]G) = \hat{\epsilon}(\Delta_o^* - [s]R_o^* - [c_o^*]G).
\]

From the non-degeneracy of \( \hat{\epsilon} \), we can obtain
\[
[uv]G = \Delta_o^* - [s]R_o^* - [c_o^*]G.
\]

Therefore, \( \Delta_o^* - [s]R_o^* - [c_o^*]G \) is the solution for the instance given as input of \( X_1 \).

Now, we analyze the probability for \( X_1 \) to output the solution of the given instance of the CDH problem.

We know that Event R_2 and R_3 never occur in the running of \( X_i \) if \( A_1(\lambda) \) succeeds the simulated game.
with $X_i$, i.e., $A_1(\lambda)$ provides a valid delegation and a valid proxy signature. And if $X_i$'s selections of four indices $I_O$, $\vartheta_O$, $K$, $\Lambda$ and a Boolean value $B|$ are correct for the final result $(\text{ID}_O^*, P_0^*, K_O^*, \delta_O^*, \text{ID}^*_P, P_P^*, K_P^*, \text{ID}^*_D, P_D^*, K_D^*, m^*, \sigma_m^*)$ of $A_1(\lambda)$, i.e., the target original signer with $\text{ID}^*_O$ is the $I_O$-th user, $(P_O^*, K_O^*)$ is the $\vartheta_O$-th full public key of the target original signer, $H_2(w_O^* \parallel \text{ID}^*_O \parallel P_O^* \parallel K_O^*)$ (where $w_O^*$ is the warrant in the delegation $\delta_O^*$ ) is the $K$-th new query to $H_2$, $H_3(m^* \parallel \delta_O^* \parallel P_P^* \parallel K_P^* \parallel \text{ID}^*_D \parallel P_D^* \parallel K_D^*)$ is the $\Lambda$-th new query to $H_3$, $B\leftarrow 0$ in the case $(\text{ID}^*_O, K_O^*) \notin P$ and $B\leftarrow 1$ in the case $(\text{ID}^*_O, P_O^*) \notin S \cup R$, Event $D, S, R_1$ and $R_4$ never occur in the running of $X_i$.

The probability that $X_i$'s selections of $I_O$, $\vartheta_O$, $K$, $\Lambda$ and $B|$ are correct is at least $\prod(N_U \cdot N_{U\text{Key}} \cdot N_{H_2} \cdot N_{H_3} \cdot 2)$. The probabilities of Events $E_1$ and $E_3$ in the running of $X_i$ is at most $N_{H_1}$ is $q$ and $N_{H_1}$ is $q$, respectively.

Therefore, the probability that $X_i$ outputs the solution of the given instance of CDH problem is at least the following expression:

$$\varepsilon(\lambda)/((N_U \cdot N_{U\text{Key}} \cdot N_{H_2} \cdot N_{H_3} \cdot 2)(1-N_{H_1}/q)(1-N_{H_1}/q)).$$

Since $N_{H_1}$, $N_{H_2}$, $N_{H_3}$ and $N_{U\text{Key}}$ are polynomial in $\lambda$, $q$ is exponential in $\lambda$, $\varepsilon(\lambda)$ is non-negligible, the above expression is also non-negligible.

Secondly, when $A(\lambda)$ is a type II adversary $A_{II}(\lambda)$, we construct an algorithm $X_{II}$ that uses $A_{II}(\lambda)$ to solve the CDH problem in the group $\Gamma_1$ as follows:

**Algorithm $X_{II}$**:

**Input/Output**: (These are the same as those in Algorithm $X_i$.)

1. Select four indices $0 < I_O \leq N_U, 0 \leq \vartheta_O \leq N_{U\text{Key}}, 0 < K \leq N_{H_2}$ and $0 < \Lambda \leq N_{H_3}$ randomly.

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

**Setup**: Let $s \leftarrow K \leftarrow Z_q^* \leftarrow P_{\text{KGC}} \leftarrow \{s\} \leftarrow \mathcal{G}$, $\text{Params} = \{a, b, p, q, G, P_{\text{KGC}}\}$, $B_{\text{Params}} := a \parallel b \parallel G \parallel P_{\text{KGC}}$. Send $\text{Params}$ and $s$ to $A_{II}(\lambda)$. Initialize the lists $S$ and $D$ with empty list ($\emptyset$) respectively, and prepare a list $H_1$ of 2-tuples, two lists $H_2$ and $H_3$ of 3-tuples and a list $L$ of tuples in form of $(\text{ID}_U, P_U, K_U, s_U, c_U)$. 32
Choose randomly $h_0^* \in \mathbb{Z}_q^*$ and $h_2^* \in \mathbb{Z}_q^*$. Let $\text{Count}_0 := 0$.

**Attack:** Answer $A_{\Pi}(\lambda)$’s queries to oracles as follows:

1. $H_1(x)$, $H_2(y)$, $H_3(z)$: (These are the same as that in Algorithm $X_{\Pi}$.)

2. $O_{\text{CreateUser}}(\text{ID}_U)$: Let $h_U \leftarrow H_1(B_{\text{Params}} || \text{ID}_U)$ and respond differently in the following cases:
   - If there exists $(\text{ID}_U, *, *, *, *) \in \mathcal{L}$ (where * means for an arbitrary value), then search $\mathcal{L}$ for the latest item $(\text{ID}_U, *, K_U, *, c_U)$ indexed by $\text{ID}_U$ to get $K_U$ and $c_U$. Otherwise, choose $k_U \in \mathbb{Z}_q^*$ randomly, let $K_U \leftarrow [k_U]G$ and $c_U \leftarrow (k_U + s \cdot h_U) \mod q$.
   - If $h_U = h_0^*$, then let $\text{Count}_0 \leftarrow \text{Count}_0 + 1$. If $\text{Count}_0 = \theta_O$, then let $P_U \leftarrow [u]G$, put $(\text{ID}_U, P_U, K_U, 0, c_U)$ in $\mathcal{L}$ and return $(P_U, K_U)$. Otherwise, go forward.
   - (This is the same as that in Algorithm $X_{\Pi}$.)

3. $O_{\text{GetSecretKey}}, O_{\text{GetPublicKey}}, O_{\text{GetDelegation}}, O_{\text{GetDVPSign}}, O_{\text{VerifyDVPSig}}$: (These are the same as those in Algorithm $X_{\Pi}$.)

**Forgery:** Wait until $A_{\Pi}(\lambda)$ outputs a tuple $(\text{ID}_O, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, \text{ID}_D, P_D, K_D, m^*, \sigma_m^*)$ as the final result of the game.

3. Check $A_{\Pi}(\lambda)$’s final result and output the solution of the above instance (i.e., the input) as follows:

   1. (This is the same as that in Algorithm $X_{\Pi}$.)

   2. Parse $\delta_O^*$ as $(w_O^*, R_O^*, \Delta_O^*)$. If the following boolean expression is true, then abort. (This case is called $Event_{R_2}$.)

   $$((\text{ID}_O^*, P_O^*) \in \mathbf{S} \lor (\text{ID}_O^*, P_O^*, K_O^*, w_O^*, \delta_O^*) \in \mathbf{D}).$$

   3, 4, 5 (These are the same as those in Algorithm $X_{\Pi}$.)

(The end of the algorithm $X_{\Pi}$)

Since the analysis of $X_{\Pi}$ is similar to the one of $X_{\Pi}$, we omit it. This completes the proof.

**Theorem 2.** In the random oracle model, if there exists a PPT (in $\lambda$) adversary $A'(\lambda)$ that can impersonates a proxy signer on the basis of a legitimate delegation to win sEUF Game against our CLDVPS scheme with a non-
If there exists a solution \( \hat{v}(G,G)^{vw}_{uvwG} \), then return \( \hat{v}(G,G)^{vw}_{uvwG} \). Otherwise, choose \( \hat{h}_p \in \mathbb{Z}^*_q \) randomly. If \( \hat{h}_p = h_p^* \) or \( \hat{h}_p = h_p^* \), then abort. (This case is called Event \( E_1 \).) Otherwise, put \( (x, h_p^*) \) in \( H_1 \) and return \( h_{U_1} \).
• $H_2(y)$: If there exists $H_2 \in \Gamma_1$ such that $(y, H_2, *) \in H_2$ (where * means for an arbitrary value), then return $H_2$. Otherwise, choose $h_2 \in \mathbb{Z}_q^*$ randomly, put $(y, [h_2] G, h_2)$ in $H_2$ and return $[h_2] G$.

• $H_3(z)$: If there exists $H_3 \in \Gamma_1$ such that $(z, H_3, *) \in H_3$ (where * means for an arbitrary value), then return $H_3$. If $|H_3| = \Lambda - 1$ then put $(z, [w] G, \perp)$ in $H_3$ and return $[w] G$. Otherwise, choose $h_3 \in \mathbb{Z}_q^*$ randomly, put $(z, [h_3] G, h_3)$ in $H_3$ and return $[h_3] G$.

• $O_{CreateUser}(\text{ID}_U)$: Let $h_U \leftarrow H_1(B_{Params} \parallel \text{ID}_U)$ and respond differently in the following cases:

  a. If there exists $(\text{ID}_U, *, *, *, *) \in \mathcal{L}$ (where * means for an arbitrary value), then search $\mathcal{L}$ for the latest item $(\text{ID}_U, *, K_U, *, c_U)$ indexed by $\text{ID}_U$ to get $K_U$ and $c_U$. Otherwise, consider the following cases:
  
   - If $B\beta = 0$ and $h_U = h'_p$, then let $K_U \leftarrow [u] G$ and $c_U := \perp$.
   
   - If $B\beta = 0$ and $h_U = h''_p$, then let $K_U \leftarrow [v] G$ and $c_U := \perp$.
   
   - Else, choose $k_U \in \mathbb{Z}_q^*$ randomly, let $K_U \leftarrow [k_U] G$ and $c_U \leftarrow (k_U + s \cdot h_U) \mod q$.

  b. If $B\beta = 1$ and $h_U = h'_p$, then let Count$_p \leftarrow \text{Count}_p + 1$. If $\text{Count}_p = \emptyset_p$, then let $P_U \leftarrow [u] G$, put $(\text{ID}_U, P_U, K_U, \perp, c_U)$ in $\mathcal{L}$ and return $(P_U, K_U)$. Otherwise, go forward.

  c. If $B\beta = 1$ and $h_U = h''_p$, then let Count$_d \leftarrow \text{Count}_d + 1$. If $\text{Count}_d = \emptyset_d$, then let $P_U \leftarrow [v] G$, put $(\text{ID}_U, P_U, K_U, \perp, c_U)$ in $\mathcal{L}$ and return $(P_U, K_U)$. Otherwise, go forward.

  d. Choose $s_U \in \mathbb{Z}_q^*$ randomly, let $P_U \leftarrow [s_U] G$, put $(\text{ID}_U, P_U, K_U, s_U, c_U)$ in $\mathcal{L}$, and return $(P_U, K_U)$.

• $O_{GetSecretKey}, O_{GetPartialKey}, O_{GetPublicKey}, O_{ReplaceUseKey}$: (These are the same as those in Algorithm $X_1$ in the proof of Theorem 1.)

• $O_{GetDelegated}(\text{ID}_O, P_O, K_O, w_O)$: (This is the same as that in Algorithm $X_1$ in the proof of Theorem 1, except that the step (b) in Algorithm $X_1$ is replaced by the following one:

  b. Search $H_2$ for the item $(w_O \parallel \text{ID}_O \parallel P_O \parallel K_O, H_2, h_2)$ indexed by $(w_O \parallel \text{ID}_O \parallel P_O \parallel K_O, H_2)$ to get $h_2$.)
\[ O_{\text{GetDVPSign}}(ID_O, P_O, K_O, \delta_O, ID_P, P_P, K_P, ID_D, P_D, K_D, m) \] (This is the same as that in Algorithm \( X_i \) in the proof of Theorem 1, except that the step (g) in Algorithm \( X_i \) is replaced by the following one:

\( \text{g. If } h_3 = \perp, \text{ then abort. (This case is called Event S.)} \)

\[ O_{\text{VerifyDVP}}(ID_O, P_O, K_O, \delta_O, ID_P, P_P, K_P, ID_D, P_D, K_D, m, \sigma_m) \]: Parse \( \delta_O \) as \( (w_O, R_O, \Delta_O) \) and check whether all input parameters conform to \( w_O \). If not, return \text{False}. If the check is passed, follow the next steps:

\( \text{a-g. (These steps are the same as those in Algorithm } X_i \text{ in the proof of Theorem 1.)} \)

\( \text{h. If } h_3 \neq \perp, \text{ then return } v_m \triangleq \hat{e}([s]R_p + [h_3]T_p, T_D). \) Otherwise, go forward.

\( \text{i. Search } H_1 \text{ for the item } (B_{\text{Params}}||ID_P, h_p) \text{ indexed by } B_{\text{Params}}||ID_P \text{ to get } h_p. \) If the search does not succeed, return \( \perp. \)

\( \text{j. Search } L \text{ for the item } (ID_P, P_P, K_P, s_p, c_p) \text{ indexed by } (ID_P, P_P, K_P) \text{ to get } s_p \text{ and } c_p. \) If the search does not succeed, return \( \perp. \) If \( s_p = \perp \) and \( c_p = \perp, \) return \( \perp. \) If \( s_p \neq \perp \) and \( c_p \neq \perp, \) return \( v_m \triangleq \hat{e}((s_p + c_p) \mod q \mod ID_P, h_p). \)

\( \text{k. Search } H_1 \text{ for the item } (B_{\text{Params}}||ID_D, h_D) \text{ indexed by } B_{\text{Params}}||ID_D \text{ to get } h_D. \) If the search does not succeed, return \( \perp. \)

\( \text{l. Search } L \text{ for the latest item } (ID_D, P_D, K_D, s_D, c_D) \text{ indexed by } (ID_D, P_D, K_D) \text{ to get } s_D \text{ and } c_D. \) If the search does not succeed, return \( \perp. \) If \( s_D = \perp \) and \( c_D = \perp, \) return \( \perp. \) If \( s_D \neq \perp \) and \( c_D \neq \perp, \) return \( v_m \triangleq \hat{e}(h_D, (s_D + c_D) \mod q \mod ID_D, h_p, h_D). \)

\( \text{m. If } c_p = \perp \) and \( c_D = \perp, \) let \( h \leftarrow \hat{e}(H_3, (s_D + s \cdot h_D) \mod q \mod ID_D, h_p). \)

\( \text{n. If } c_p = \perp \) and \( s_D = \perp, \) let \( h \leftarrow \hat{e}(H_3, [c_D]T_D + [s_p + s \cdot h_D] \mod q \mod ID_D, h_p). \)

\( \text{o. If } s_p = \perp \) and \( c_D = \perp, \) let \( h \leftarrow \hat{e}(H_3, [(s_D + s \cdot h_D) \mod q \mod ID_D + [c_P]K_D, h_p]. \)

\( \text{p. If } s_p = \perp \) and \( s_D = \perp, \) let \( h \leftarrow \hat{e}(H_3, [c_D]T_D + [c_P]K_D, h_p]. \)

Forgery: Wait until $A'_1(\lambda)$ outputs a tuple $(ID_O^*, P_O^*, K_O^*, \delta_O^*, ID_P^*, P_P^*, K_P^*, ID_D^*, P_D^*, K_D^*, m^*, \sigma_m^*)$ as the final result of the game.

3. Check $A'_1(\lambda)$ ’s final result and output the solution of the above instance (i.e., the input) as follows:

   (1) If the following boolean expression is true, then abort. (This case is called Event $R_1$.)

   $$h_p^* \neq H_1( B_{Param} \parallel ID_P^* ) \lor h_D^* \neq H_1( B_{Param} \parallel ID_D^* ) \lor$$

   $$[w]G \neq H_3( m^* \parallel \delta_O^* \parallel ID_P^* \parallel P_P^* \parallel K_P^* \parallel ID_D^* \parallel P_D^* \parallel K_D^* ).$$

   (2) If the following boolean expression is true, then abort. (This case is called Event $R_2$.)

   $$((ID_P^* , K_P^* ) \in P \land (ID_P^* , P_P^* ) \in S \cup R ) \lor ((ID_D^* , K_D^* ) \in P \land (ID_D^* , P_D^* ) \in S \cup R )) .$$

   (3) If $O_{Verify_DVB\|n}( (ID_P^* , P_O^* , K_O^* , \delta_O^* , ID_P^* , P_P^* , K_P^* , ID_D^* , P_D^* , K_D^* , m^* , \sigma_m^*) = \text{False} , \text{then abort.}$

   (This case is called Event $R_3$.) Otherwise, parse $\sigma_m^*$ as $( \mathcal{K}_P^*, \mathcal{V}_m^* )$ and return $\mathcal{V}_m^* / h^*$. $(\mathcal{V}_m^* / h^*$ is the last parameter of the query to $O_{DBDH}$ in the oracle $O_{Verify_DVB\|n}$.)

(The end of the algorithm $X_1$)

Since the analysis of $X_1$ is similar to the one of $X_\ell$ in the proof of Theorem 1, we only show that $X_1$ ’s output is the solution for the given instance of the GBDH problem if $X_1$ does not abort.

$X_1$ can output the result $\mathcal{V}_m^* / h^*$ if and only if $A'_1(\lambda)$ succeeds the simulated game with $X_1$, i.e., $A'_1(\lambda)$ provides a valid proxy signature, and $X_1$ ’s selections of five indices $I_p , I_D , \mathcal{S}_p , \mathcal{S}_D , \Lambda$ and two Boolean values $B\|B , B\|B$ are correct for the final result $( ID_O^* , P_O^* , K_O^* , \delta_O^* , ID_P^* , P_P^* , K_P^* , ID_D^* , P_D^* , K_D^* , m^* , \sigma_m^* )$ of $A'_1(\lambda)$, i.e., the target proxy signer with $ID_P^*$ is the $I_p$ -th user, the target designated verifier with $ID_D^*$ is the $I_D$ -th user, $( P_P^* , K_P^* )$ is the $\mathcal{S}_p$ -th full public key of the target proxy signer, $( P_D^* , K_D^* )$ is the $\mathcal{S}_D$ -th full public key of the target designated verifier, $H_3( m^* \parallel \delta_O^* \parallel ID_P^* \parallel P_P^* \parallel K_P^* \parallel ID_D^* \parallel P_D^* \parallel K_D^* )$ is the $\Lambda$ -th new query to $H_3$, $B\|B=0$ in the case $( ID_P^* , K_P^* ) \notin P , B\|B=1$ in the case $( ID_P^* , P_P^* ) \notin S \cup R , B\|B=0$ in the case $( ID_D^* , K_D^* ) \notin P , B\|B=1$ in the case $( ID_D^* , P_D^* ) \notin S \cup R .$

Assume that $X_1$ does not abort and outputs the result.

37
If $B_β=0$, then $K_p^*=u|G, c_p^*\perp s_p^* \in Z_q^*$ and $P_p^*=[s_p^*]G$ from the step (a) and (d) of $O_{CreateUser}$. If $B_β=1$, then $P_p^*=[u|G, c_p^*\perp s_p^* \in Z_q^*, K_p^*=[k_p^*]G, c_p^*=(k_p^*+s \cdot h_p^*) \text{ mod } q$ from the step (a) and (b) of $O_{CreateUser}$. If $B_β=0$, then $K^*_D=[v|G, c^*_D\perp s_D^* \in Z_q^* and P^*_D=[s_D^*]G$ from the step (a) and (d) of $O_{CreateUser}$. If $B_β=1$, then $P^*_D=[v|G, c^*_D\perp s_D^* \in Z_q^*, K^*_D=[k^*_D]G, c^*_D=(k^*_D+s \cdot h^*_D) \text{ mod } q$ from the step (a) and (c) of $O_{CreateUser}$.

Hence, in running of $O_{VerifyProxySign}(ID^*_O, ID^*_P, ID^*_D, P^*_O, P^*_P, P^*_D, m^*, \sigma_m^*)$, the step (m) is executed if $B_β=0$ and $B_β=0$, the step (n) is executed if $B_β=0$ and $B_β=1$, the step (o) is executed if $B_β=1$ and $B_β=0$, and the step (p) is executed if $B_β=1$ and $B_β=1$.

If $X^*_q$'s selections of $\Lambda$ is correct for the final result provided by $A^*_1(\lambda)$, the following equation holds.

$$[w|G=H_3(m \parallel \delta^*_D \parallel ID^*_P \parallel P^*_P \parallel K^*_P \parallel ID^*_D \parallel P^*_D \parallel K^*_D).$$

If $A^*_1(\lambda)$ provides a valid proxy signature, the final result satisfies the following verification equation:

$$\nu^*_m=\hat{e}(R^*_p, P_{KGC})^{t^*_p} \cdot \hat{e}([w|G, T^*_p])^{t^*_o}.$$  

where $\sigma_m^*= (R^*_p, \nu^*_m), T^*_p = P^*_p + K^*_p + [h^*_p]P_{KGC}$ and $t^*_D = (s^*_D + c^*_D) \text{ mod } q$. From the above equation, we can obtain

$$\nu^*_m = \hat{e}(R^*_p, [t^*_D \parallel s]G) \cdot \hat{e}([w|G, [t^*_D \parallel t^*_p]G)$$

$$= \hat{e}(R^*_p, [s \parallel T^*_D] \cdot \hat{e}([w|G, ([t^*_D \parallel t^*_p] \text{ mod } q)G).$$

where $t^*_p = (s^*_p + c^*_p) \text{ mod } q$.

When $B^*_β=0$ and $B^*_β=0$, we can compute

$$\hat{e}([w|G, ([t^*_D \parallel t^*_p] \text{ mod } q)G)$$

$$\hat{e}([w|G, ([s^*_D + c^*_D \perp s^*_p + c^*_p] \text{ mod } q)G$$

$$\hat{e}([w|G, ([s^*_D + v + s \cdot h^*_D \perp s^*_p + c^*_p] \text{ mod } q)G$$

$$\hat{e}([w|G, ([s^*_D + s \cdot h^*_D] \text{ mod } q) \parallel t^*_p]G)$$

$$\hat{e}([w|G, ([s^*_p + s \cdot h^*_p \parallel t^*_p] \text{ mod } q)G)$$

$$\hat{e}([w|G, ([s^*_p + s \cdot h^*_p] \text{ mod } q) \parallel ([uv] \text{ mod } q)G)$$

$$\hat{e}([w|G, ([s^*_p + s \cdot h^*_p] \text{ mod } q) \parallel ([uv] \text{ mod } q)G)$$

38
\[ \hat{e}([(w)G]((s_D^* + s \cdot h_D^*) \mod q] T_p + [(s_p^* + s \cdot h_p^*) \mod q] K_D^*) \cdot \hat{e}([(w)G]((uv) \mod q] G) \]

\[ = \hat{e}([(w)G]((s_D^* + s \cdot h_D^*) \mod q] T_p + [(s_p^* + s \cdot h_p^*) \mod q] K_D^*) \cdot \hat{e}(G, G)^{\text{avg}}. \]

Hence, from the following equation

\[ h^* = \hat{e}([(w)G]((s_D^* + s \cdot h_D^*) \mod q] T_p + [(s_p^* + s \cdot h_p^*) \mod q] K_D^*) \cdot \hat{e}(K_D^*, [s] T_D^*) \]

and the above discussion, we can obtain

\[ \hat{e}(G, G)^{\text{avg}} = v \cdot h^*. \]

Similarly, we can discuss other cases.

Secondly, when \( A'_{\Pi}(\lambda) \) is a type II adversary \( A'_{\Pi}(\lambda) \), we construct an algorithm \( X'_\Pi \) that uses \( A'_{\Pi}(\lambda) \) and the DBDH oracle \( O_{DBDH} \) to solve the GBDH problem in the bilinear group pair \( (\Gamma, \Gamma') \) as follows:

**Algorithm** \( X'_\Pi \):

**Input/Output:** (These are the same as those in Algorithm \( X'_I \).)

1. Select five indices \( 0 < 1 \cdot p, 1 \cdot D \leq N_U, 0 \leq \emptyset \cdot p, \emptyset \cdot D < N_{U_{key}} \) and \( 0 < \lambda \leq N_{H_3} \) randomly.

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

**Setup:** Let \( s \leftarrow_{K} Z_q^* \), \( P_{KGC} \leftarrow [s] G \), \( \text{Params} = \{ a, b, p, q, G, P_{KGC} \} \), \( B_{\text{Params}} \leftarrow a \parallel b \parallel G \parallel P_{KGC} \). Send \( \text{Params} \) and \( s \) to \( A'_{\Pi}(\lambda) \). Initialize the lists \( S \) and \( D \) with empty list \( (\emptyset) \) respectively, and prepare a list \( H_1 \) of 2-tuples, two lists \( H_2 \) and \( H_3 \) of 3-tuples and a list \( L \) of tuples in form of \( (ID_U, P_U, K_U, s_U, c_U) \).

Choose randomly \( h_p^* \in Z_q^* \) and \( h_D^* \in Z_q^* \) such that \( h_p^* \neq h_D^* \). Let \( \text{Count}_p := 0 \) and \( \text{Count}_D := 0 \).

**Attack:** Answer \( A'_{\Pi}(\lambda) \)'s queries to oracles as follows:

- \( H_1 (x) , H_2 (y) , H_3 (z) \) : (These are the same as that in Algorithm \( X'_I \).)

- \( O_{\text{CreateUser}}(ID_U) \): Let \( h_U \leftarrow H_1 (B_{\text{Params}} \parallel ID_U) \) and respond differently in the following cases:

  a. If there exists \( (ID_U, *, *, *, *) \in L \) (where \( * \) means for an arbitrary value), then search \( L \) for the latest item \( (ID_U, *, K_U, *, c_U) \) indexed by \( ID_U \) to get \( K_U \) and \( c_U \). Otherwise, choose \( k_U \in Z_q^* \) randomly, let \( K_U \leftarrow [k_U] G \) and \( c_U \leftarrow (k_U + s \cdot h_U) \mod q \).

  b. If \( h_U = h_p^* \), then let \( \text{Count}_p \leftarrow \text{Count}_p + 1 \). If \( \text{Count}_p = 3 \cdot p \), then let \( P_U \leftarrow [u] G \), put \( (ID_U, P_U, \)
\(K_U, \perp, c_U\) in \(L\) and return \((P_U, K_U)\). Otherwise, go forward.

c. If \(h_U = h_D^*\), then let \(\text{Count}_D \leftarrow \text{Count}_D + 1\). If \(\text{Count}_D = \mathcal{G}_D\), then let \(P_U \leftarrow [v]G\), put \((\text{ID}_U, P_U, K_U, \perp, c_U)\) in \(L\) and return \((P_U, K_U)\). Otherwise, go forward.

d. (This is the same as that in Algorithm \(X_1\).)

- \(O_{\text{GetSecretKey}}, O_{\text{GetPublicKey}}, O_{\text{GetDelegation}}, O_{\text{GetDVPSign}}\): (These are the same as those in Algorithm \(X_1\).)

- \(O_{\text{VerifyDVPS}}(\text{ID}_D, P_O, K_O, \delta_O, \text{ID}_P, P_P, K_P, \text{ID}_D, P_D, K_D, m, \sigma_m)\): Parse \(\delta_O\) as \((w_O, R_O, \Delta_O)\) and check whether all input parameters conform to \(w_O\). If not, return \text{False}. If the check is passed, follow the next steps:

a–i. (These steps are the same as those in Algorithm \(X_1\).)

j. Search \(L\) for the item \((\text{ID}_P, P_P, K_P, s_P, c_P)\) indexed by \((\text{ID}_P, P_P, K_P)\) to get \(s_P\) and \(c_P\). If the search does not succeed, return \(\perp\). If \(c_P = \perp\), return \(\perp\). If \(s_P \neq \perp\), return \(v_m = \hat{e}( [s] R_P + (s_P + c_P) \mod q ) H_3, T_D )\).

k. Search \(H_1\) for the item \((B_{\text{Params}} || \text{ID}_D, h_D)\) indexed by \(B_{\text{Params}} || \text{ID}_D\) to get \(h_D\). If the search does not succeed, return \(\perp\).

l. Search \(L\) for the latest item \((\text{ID}_D, P_D, K_D, s_D, c_D)\) indexed by \((\text{ID}_D, P_D, K_D)\) to get \(s_D\) and \(c_D\). If the search does not succeed, return \(\perp\). If \(c_D = \perp\), return \(\perp\). If \(s_D \neq \perp\), return \(v_m = \hat{e}( H_3, [(s_D + c_D) \mod q ] T_P \cdot \hat{e}( R_P, [s] T_D )\).

m. If \(s_P = \perp\) and \(s_D = \perp\), let \(h \leftarrow \hat{e}( H_3, [c_D] T_P \cdot [c_P] K_D ) \cdot \hat{e}( R_P, [s] T_D )\).


\text{Forgery: Wait until } A'_{II}(\lambda) \text{ outputs a tuple } (\text{ID}_O^*, P_O^*, K_O^*, \delta_O^*, \text{ID}_P^*, P_P^*, K_P^*, \text{ID}_D^*, P_D^*, K_D^*, m^*, \sigma_m^*) \text{ as the final result of the game.}

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

(1) (This is the same as that in Algorithm \(X_1\).)

(2) If the following boolean expression is true, then abort. (This case is called \text{Event } R_2.)

\[ ((\text{ID}_P, P_P^*) \in S \lor (\text{ID}_D, P_D^*) \in S) \].
(3) (This is the same as that in Algorithm X_t.)

(The end of the algorithm X_t)

Since the analysis of X_t is similar to the one of X_3, we omit it. This completes the proof.

From Theorem 1 and 2, we can conclude that our CLDVS scheme is strong existentially unforgeable against an adaptively chosen-message attack and an adaptively chosen-warrant attack in the random oracle model (ROM), under the assumption that the Computational Diffie–Hellman (CDH) Problem in an elliptic curve group and the Gap Bilinear Diffie–Hellman (GBDH) Problem in a bilinear group pair are hard.

Next, we consider the non-transferability of our CLDVS scheme.

When we fix a challenge message \( m^* \), the identities \{ ID_O^p, ID_P^p, ID_D^p \} of an original signer, a proxy signer and a designated verifier, a valid delegation \( \delta^*_O \), the public keys \( P_O^*, P_P^*, K_P^*, P_D^*, K_D^* \), and the corresponding private keys \( s_P^*, c_P^*, s_D^*, c_D^* \) for signing, based on submission of an adversary in NT Game (Definition 7), the following values are also fixed:

\[
\begin{align*}
t_p^* &:= (s_p^* + c_p^*) \mod q, \\
t_D^* &:= (s_D^* + c_D^*) \mod q, \\
T_p^* &:= P_p^* + K_p^* + [H_1(B_{\text{params}} \parallel ID_p^p)] P_{KGC}, \\
T_D^* &:= P_D^* + K_D^* + [H_1(B_{\text{params}} \parallel ID_D^p)] P_{KGC}, \\
H_3 &:= H_3(m^* \parallel \delta^*_O \parallel ID_P^p \parallel P_P^* \parallel K_P^* \parallel ID_D^p \parallel P_D^* \parallel K_D^*).
\end{align*}
\]

Then, the ensemble of valid signatures on \( m^* \) produced by the algorithm GenerateDVPSign is as follows:

\[
\Sigma := \{(R_p, \hat{\epsilon}(|r_p| P_{KGC}[t_p^* H_3, T_D^*]) | r_p \leftarrow_r Z_q^*, R_p \leftarrow [r_p] G \}.
\]

Meanwhile, the the ensemble of valid simulated signatures on \( m^* \) produced by the simulation algorithm SimulateDVPSign is as follows:

\[
\Sigma^S := \{(R_D, \hat{\epsilon}(R_D, P_{\text{KGC}} t_D^* \cdot \hat{\epsilon}(H_3, T_P^*)) | r_D \leftarrow_r Z_q^*, R_D \leftarrow [r_D] G \}.
\]

Considering the equations \( T_p^* = [t_p^*] G \) and \( T_D^* = [t_D^*] \), we can compute the following equations.

\[
\hat{\epsilon}(R_D, P_{\text{KGC}} t_D^* \cdot \hat{\epsilon}(H_3, T_P^*)) \cdot \hat{\epsilon}(H_3, [t_D^*] T_P^*)
\]
\[= \hat{e}(R_D, [t_D^*] s] G) \cdot \hat{e}(H_3, [t_D^*] G)\]
\[= \hat{e}([s] R_D, [t_D^*] G) \cdot \hat{e}([t_P^*] H_3, [t_D^*] G)\]
\[= \hat{e}([t_D] [s] G, T_D^*) \cdot \hat{e}([t_P^*] H_3, T_D^*)\]
\[= \hat{e}([t_D] P_{KGC} + [t_P^*] H_3, T_D^*).
\]

That is, any element of \(\Sigma\) corresponds to only one element of \(\Sigma^S\) and vice versa. The randomness of any element in \(\Sigma\) is determined by \(r_p \leftarrow_R Z_q^*\) and the randomness of any element in \(\Sigma^S\) is also determined by \(r_D \leftarrow_R Z_q^*\). Therefore, the distributions of any element of \(\Sigma\) and any element of \(\Sigma^S\) are identical and any unconditional distinguisher \(A_j\) cannot determine whether the signature is created by the proxy signer or simulated by the designated verifier. Namely, we can obtain the following result.

**Theorem 3.** The proposed CLDVPS scheme is unconditionally (perfectly) non-transferable.

6. Conclusion

Certificateless designated verifier proxy signature (CLDVPS) stands for designated verifier proxy signature (DVPS) 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 formalize the definition of a CLDVPS scheme and the security model for CLDVPS schemes. We then proposed the first CLDVPS scheme and analyze its efficiency. We also prove that in the random oracle model, our scheme is strong existentially unforgeable against an adaptively chosen-message attack and an adaptively chosen-warrant attack under the assumption that the Computational Diffie–Hellman (CDH) Problem in an elliptic curve group and the Gap Bilinear Diffie–Hellman (GBDH) Problem in a bilinear group pair are hard, and it is unconditionally non-transferable.

To the best of our knowledge, the concept of certificateless designated verifier proxy signature (CLDVPS) has not been appeared in the literature but Identity-based DVPS (IBDVPS) schemes have been discussed. Our scheme is the first provably secure CLDVPS scheme.

We believe that our scheme satisfies the privacy of signer’s identity.

We leave constructing a non-delegatable CLDVPS scheme as future work. It is also interesting to construct a CLDVPS schemes without pairings and prove the security in the standard model.
References


