INFORMATICA, 2010, Vol. 21, No. 2, 277–294

A Provably Secure Proxy Signature Scheme in Certificateless Cryptography

Hu XIONG, Fagen LI, Zhiguang QIN
School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu 610054, P.R. China
e-mail: xionghu.uestc@gmail.com

Received: December 2008; accepted: May 2009

Abstract. A proxy signature scheme enables an original signer to delegate its signing capability to a proxy signer and then the proxy signer can sign a message on behalf of the original signer. Recently, in order to eliminate the use of certificates in certified public key cryptography and the key-escrow problem in identity-based cryptography, the notion of certificateless public key cryptography was introduced. In this paper, we first present a security model for certificateless proxy signature schemes, and then propose an efficient construction based on bilinear pairings. The security of the proposed scheme can be proved to be equivalent to the computational Diffie–Hellman problem in the random oracle with a tight reduction.

Keywords: proxy signature, certificateless cryptography, provable security, random oracle model.

1. Introduction

The concept of proxy signature was first introduced by Mambo *et al.* (1996). The proxy signature schemes allow a proxy signer to sign messages on behalf of an original signer within a given context (the context and limitations on proxy signing capabilities are captured by a certain warrant issued by the delegator which is associated with the delegation act). Proxy signatures have been found numerous practical applications, particularly in distributed computing where delegation of rights is quite common, distributed shared object systems, global distribution networks, and mobile communications. Since Mambo *et al.*'s scheme, many proxy signature schemes have been proposed (Alomair *et al.*, 2008; Boldyreva *et al.*, 2003; Kim *et al.*, 1997; Lee *et al.*, 2001; Malkin *et al.*, 2004). Proxy signatures can combine other special signatures to obtain some new types of proxy signatures. These include threshold proxy signatures (Zhang, 1997), blind proxy signatures (Lin and Jan, 2000), proxy ring signatures (Li *et al.*, 2006) and one-time proxy signatures (Kim *et al.*, 2001). However, the theory of proxy signature faces some problems when it comes to reality. The public key of user is usually a "random" string that is unrelated to the identity of the user in traditional public key infrastructure (PKI), so there is a trusted-by-all certificate authority (CA) to assure the relationship between the cryptographic keys and the user. As a result, any verifier of a signature must obtain and verify the user's certificate before checking the validity of the signature. The communication and the validation of a large number of public keys greatly affect the efficiency of the proxy signature.

ID-based cryptography which was introduced in 1984 by Shamir solved these problems: the public key of each user is easily computable from a string corresponding to this user's identity (such as an email address), while the private key associated with that identity is computed and issued secretly to the user by a trusted third party called private key generator (PKG). This property avoids the necessity of certificates, and associates an implicit public key to each person over the world. So ID-based proxy signature has rapidly emerged in recent years and been well studied as well. The first work on ID-based proxy signature was proposed by Zhang and Kim (2003). Then, Xu *et al.* (2005) proposed an ID-based proxy signature scheme from pairings. They extended Boldyreva *el al.*'s (2003) security model for proxy signature schemes to the ID-based setting and proved its security in that model without using forking lemma (Pointcheval and Stern, 2000). After that, Shim (2006) proposed another efficient ID-based proxy signature scheme with more tighter security reduction. However, an inherent problem of ID-based cryptosystems is key escrow, i.e., the PKG knows users' private key. A malicious PKG can frame an innocent user by forging the user's signature. Due to this inherent problem, ID-based cryptosystems are considered to be suitable only for private networks (Shamir, 1984). Thus, eliminating key escrow in ID-based cryptosystems is essential to make them more applicable in the real world.

To overcome the drawback of key escrow in ID-PKC, Al-Riyami and Paterson (2003) proposed a paradigm called certificateless public key cryptography (CL-PKC) in 2003. The concept was introduced to suppress the inherent key-escrow property of identity-based public key cryptosystems (ID-PKC)without losing their most attractive advantage which is the absence of digital certificates and their important management overhead. Like ID-PKC, certificateless cryptography does not use public key certificate (Al-Riyami and Paterson, 2003; Zhang and Wong, 2006), it also needs a third party called Key Generation Center (KGC) to help a user to generate his private key. However, the KGC does not have access to a user's full private key. It just generates a user's partial private key from the user's identity as the PKG in ID-PKC does. A user computes his full private key by combining his partial private key and a secret value chosen by himself. The public key of a user is computed from the KGC's public parameters and the secret value of the user, and it is published by the user himself.

Recently, many researchers have been investigating secure and efficient certificateless signature (CLS) schemes. In their original paper, Al-Riyami and Paterson (2003) presented a CLS scheme. Huang *et al.* (2005) pointed out a security drawback of the original scheme and proposed a secure one. A generic construction of CLS scheme was proposed by Yum and Lee (2004) in ACISP 2004. However, Hu *et al.* (2006) showed that the Yum-Lee construction is insecure and proposed a fix in the standard model. In ACNS 2006, Zhang and Wong (2006) presented an efficient CLS scheme from pairings. Gorantla and Saxena (2005) introduced a new construction of CLS scheme without providing formal proofs. Their scheme has been shown insecure by Cao *et al.* (2006). The survey and discussions of CLS scheme can be found in Huang *et al.* (2007), Hu *et al.* (2006), Dent and

Comley (2006). To the best of our knowledge, Li *et al.* (2005) proposed the first certificateless proxy signature based on bilinear pairings. After that, Lu *et al.* (2007) and Yap *et al.* (2007) showed that Li *et al.*'s scheme is insecure and proposed the fix, respectively. Unfortunately, all of these works only provide informal security analysis, i.e., there are no proven secure certificateless proxy signature schemes until now. Our current work is aimed at filling this void. A security model for certificateless proxy signature is proposed in our paper. The model captures the notion of existential unforgeability of certificateless signature against Type I and Type II adversaries. We then propose an efficient and simple certificateless proxy signature scheme and show its security in our model, with the assumption that Computational Diffie–Hellman problem is intractable.

The rest of this paper is organized as follows. A brief review of some basic concepts and tools used in our scheme is described in Section 2. The proposed certificateless proxy signature scheme is given in Section 3. The security of our scheme is analyzed in Section 4. Finally, the conclusions are given in Section 5.

2. Preliminaries

In this section, we will review some fundamental backgrounds required in this paper, namely bilinear pairing and the definition of certificateless proxy signature scheme.

2.1. *Bilinear Pairing and Complexity Assumption*

Let $\mathbb{G}_1$ denote an additive group of prime order q and $\mathbb{G}_2$ be a multiplicative group of the same order. Let P be a generator of $\mathbb{G}_1$, and $\hat{e}$ be a bilinear map such that $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties:

1. Bilinearity: For all $P, Q \in \mathbb{G}_1$, and $a, b \in \mathbb{Z}_q$, $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$.
2. Non-degeneracy: $\hat{e}(P, P) \neq 1_{\mathbb{G}_2}$.
3. Computability: It is efficient to compute $\hat{e}(P, Q)$ for all $P, Q \in \mathbb{G}_1$.

The security of our signature scheme will be reduced to the hardness of the Computational Diffe–Hellman (CDH) problem in the group in which the signature is constructed. We briefly review the definition of the CDH problem:

DEFINITION 1. Given the elements P, aP and bP, for some random values $a, b \in \mathbb{Z}_q$ the Computational Diffe–Hellman (CDH) problem consists of computing the element abP.

The success probability of any probabilistic polynomial-time algorithm $\mathcal{A}$ in solving CDH problem in $\mathbb{G}_1$ is defined to be

$$\mathrm{Succ}_{\mathcal{A},\mathbb{G}_1}^{CDH} = Pr\big[\mathcal{A}(P, aP, bP) = abP\colon a, b \in \mathbb{Z}_q\big].$$

The CDH assumption states that for every probabilistic polynomial-time algorithm $\mathcal{A}$, $\mathrm{Succ}_{\mathcal{A},\mathbb{G}_1}^{CDH}$ is negligible.

2.2. *Security Notions*

Component of Certificateless Proxy Signature Schemes

A Certificateless Proxy Signature (CL-PS) scheme is a tuple CL-PS=(MasterKeyGen, PartialKeyGen, UserKeyGen, (Delegation, Proxy), Sign and Verify), and the description of each algorithm is as follows.

1. The randomized parameters generation algorithm MasterKeyGen takes as input 1^k, where k is the security parameter and outputs a master public/secret key pair (mpk, msk). The algorithm is assumed to be run by a Key Generation Center (KGC) for the initial setup of a certificateless proxy signature scheme.
2. The randomized private key generation algorithm PartialKeyGen takes as input msk and user's identity $ID \in \{0,1\}^*$ and generates a key psk_{ID} called user partial key. This algorithm is run by the KGC once for each user, and the partial private key is assumed to be distributed securely to the corresponding user.
3. The randomized user key generation algorithm UserKeyGen takes as input mpk and user's identity ID and generates a user public/secret key pair (upk_{ID}, usk_{ID}). This algorithm is supposed to be run by each user in the system.
4. (Delegation, Proxy) is a pair of interactive randomized algorithms forming the (two-party) proxy-designation protocol. The input to each algorithm includes two identities $\{ID_i, ID_j\}$ with a warrant ω (the warrant made by the original signer ID_i is public and it implies that the original signer ID_i delegates ID_j as a proxy singer). The order of $\{ID_i, ID_j\}$ is important, i.e., $\{ID_i, ID_j\}$ and $\{ID_j, ID_i\}$ are different inputs in the proxy signing key generation algorithms. Delegation also takes as input the user secret key usk_{ID_i} and the user partial key psk_{ID_i} of the original signer, and Proxy also takes as input the user secret key usk_{ID_j} and the user partial key psk_{ID_j} of the proxy signer. As result of the interaction, a proxy signing key $\sigma_P =$ (Delegation$(ID_i, ID_j, \omega, usk_{ID_i}, psk_{ID_i})$, Proxy$(ID_i, ID_j, \omega, usk_{ID_j}, psk_{ID_j})$) for ID_j is output. This algorithm is run by the original signer and the proxy signer interactively.
5. The randomized proxy signing algorithm Sign takes as input a proxy signing key σ_P corresponding to an identity ID_j, a message $m \in \{0,1\}^*$ and outputs a proxy signature $sig \leftarrow \mathsf{Sign}(\sigma_P, m)$.
6. The randomized verification algorithm Verify takes as input mpk, a set of identities $\{ID_i, ID_j\}$ with a warrant ω, the corresponding user public key (upk_{ID_i}, upk_{ID_j}), a message $m \in \{0,1\}^*$ and a proxy signature sig of m for $\{ID_i, ID_j\}$, and outputs True if the signature is correct, or $\perp$ otherwise, i.e., $\{\mathsf{True}, \perp\}$ $\leftarrow \mathsf{Verify}(\omega, m, mpk, ID_i, ID_j, upk_{ID_i}, upk_{ID_j}, sig)$.

Adversaries Model of Certificateless Proxy Signature Scheme

Combining the security notions of certificateless public key cryptography and security models of proxy signature schemes in traditional PKC and ID-PKC, we define two types of security for CL-PS scheme, Type-I security and Type-II security, along with two types of adversaries, $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Adversary $\mathcal{A}_1$ models a malicious adversary which compromises the user secret key usk_{ID} or replaces the user public key upk_{ID}, however, cannot compromise the master secret key msk nor get access to the user partial key psk_{ID}. Adversary $\mathcal{A}_2$ models the malicious-but-passive KGC who controls the generation of the master public/secret key pair, and that of any user partial key psk_{ID}. Furthermore, we give both of adversaries the power to request proxy signing keys on any desired identity. The following are six oracles which can be accessed by the adversaries.

1. **CreateUser**: On input an identity $ID \in \{0,1\}^*$, if ID has already been created, nothing is to be carried out. Otherwise, the oracle generates $psk_{ID} \leftarrow$ PartialKeyGen(msk, ID) and $(upk_{ID}, usk_{ID}) \leftarrow$ UserKeyGen(mpk, ID). It then stores $(ID, psk_{ID}, upk_{ID}, usk_{ID})$ into a list List. In both cases, upk_{ID} is returned.
2. **RevealPartialKey**: On input an identity ID, the oracle searches List for an corresponding entry to ID. If it is not found, $\perp$ is returned; otherwise, the corresponding psk_{ID} is returned.
3. **RevealSecretKey**: On input an identity ID, the oracle searches List for an corresponding entry to ID. If it is not found, $\perp$ is returned; otherwise, the corresponding usk_{ID} is returned.
4. **ReplaceKey**: On input an identity ID and a user public/secret key pair (upk^*, usk^*), the oracle searches List for the entry of ID. If it is not found, nothing will be carried. Otherwise, the oracle updates $(ID, psk_{ID}, upk_{ID}, usk_{ID})$ to $(ID, psk_{ID}, upk^*_{ID}, usk^*_{ID})$.
5. **RevealProxyKey**: Proceeding adaptively, for a given pair of identities $\{ID_i, ID_j\}$ with a warrant ω, i.e., it implies that an original signer ID_i designates ID_j as a proxy signers, the oracle proceeds in one of the three cases below.
   (a) A valid proxy signing key σ_P for ID_j is returned if $\{ID_i, ID_j\}$ have both been created but the corresponding user public/secret key pairs (upk_{ID_i}, usk_{ID_i}) and (upk_{ID_j}, usk_{ID_j}) have not been replaced.
   (b) If ID_k, where k is one of i and j, has not been created, a symbol $\perp$ is returned.
   (c) If the user public/secret key pair of ID_k, where k is one of i and j, has been replaced with, say $(upk^*_{ID_k}, usk^*_{ID_k})$, then the oracle returns the result of σ^*_P.
6. **Sign**: On input a message $m \in \{0,1\}^*$ for $\{ID_i, ID_j\}$ with a warrant ω, the signing oracle firstly runs the **RevealProxyKey** oracle to obtain the proxy singing key, then the signing oracle runs the Sign algorithm and generates the proxy signature sig.

REMARK. When querying the oracle **ReplaceKey**, usk^*_{ID} can be an empty string. In this case, it means that the user secret key is not provided. If usk^*_{ID} is an empty string and the original user secret key of an identity ID is replaced with usk^*_{ID}, then the empty string will be returned if the **RevealSecretKey** oracle is queried on ID. Also note that even if usk^*_{ID} is not an empty string, it does not mean that usk^*_{ID} is the corresponding secret key of upk^*_{ID}. Hence as mentioned, the proxy signing key generated by the proxy key generation oracle **RevealProxyKey** will be an execution of (Delegation, Proxy) using the replaced user secret key usk^*_{ID} regardless of the value of upk^*_{ID}. In other words, the proxy signing key and the corresponding proxy signature may not be valid.

We define two games, one for $\mathcal{A}_1$ and the other one for $\mathcal{A}_2$.

Game I: Let $\mathcal{S}_1$ be the game simulator/challenger and $k \in \mathbb{N}$ be a security parameter.

1. $\mathcal{S}_1$ executes $\mathsf{MasterKeyGen}(1^k)$ to get (mpk, msk).
2. $\mathcal{S}_1$ runs $\mathcal{A}_1$ on 1^k and mpk. During the simulation, $\mathcal{A}_1$ can make queries onto oracle **CreateUser**, **RevealPartialKey**, **RevealSecretKey**, **ReplaceKey**, **RevealProxyKey** and **Sign**.
3. $\mathcal{S}_1$ is to output $(\omega^*, m^*, ID_i^*, ID_j^*, sig^*)$.

$\mathcal{A}_1$ outputs sig^* on a message m^* for $\{ID_i^*, ID_j^*\}$ with a warrant ω^* such that

- m^* is not equal to the inputs of any query to **Sign** under ID_j^*,
- $\{ID_i^*, ID_j^*\}$ with a warrant ω^* is not requested to **RevealProxyKey** query, i.e., ID_j^* was not designated by ID_i^* as a proxy signer,
- ID_k^*, where k is one of i and j, has not been submitted to both **RevealPartialKey** oracle and, **ReplaceKey** oracle or **RevealSecretKey** oracle.

$\mathcal{A}_1$ wins the game if sig^* is a valid proxy signature.

DEFINITION 2. A CL-PS scheme is said to be Type-I secure if there is no probabilistic polynomial-time adversary $\mathcal{A}_1$ which wins **Game I** with non-negligible advantage.

Game II: Let $\mathcal{S}_2$ be the game challenger and $k \in \mathbb{N}$ be a security parameter. There are two phases of interactions between $\mathcal{S}_2$ and $\mathcal{A}_2$.

1. $\mathcal{S}_2$ executes $\mathcal{A}_2$ on input 1^k, which returns a master public/secret key pair (mpk, msk) to $\mathcal{A}_2$. Note that $\mathcal{A}_2$ cannot make any query at this stage.
2. During this stage of simulation, $\mathcal{A}_2$ can make queries onto oracle **RevealSecretKey**, **RevealProxyKey** and **Sign**. $\mathcal{A}_2$ can also make queries to **CreateUser**. Note that oracle **RevealPartialKey** is not accessible and no longer needed as $\mathcal{A}_2$ has the master secret key, and when $\mathcal{A}_2$ issues a query to **CreateUser** oracle, it has to additionally provide the user partial key psk_{ID}.
3. At the end of this phase, $\mathcal{A}_2$ is to output a triple $(\omega^*, m^*, ID_i^*, ID_j^*, sig^*)$.

$\mathcal{A}_2$ outputs sig^* on a message m^* for $\{ID_i^*, ID_j^*\}$ with a warrant ω^* such that

- m^* is not equal to the inputs of any query to **Sign** under ID_j^*,
- $\{ID_i^*, ID_j^*\}$ with a warrant ω^* is not requested to **RevealProxyKey** query, i.e., ID_j^* was not designated by ID_i^* as a proxy signer.
- $\mathcal{A}_2$ has never queried **RevealSecretKey**(ID_k^*) to get the user secret key $usk_{ID_k^*}$, where k is one of i and j.

$\mathcal{A}_2$ wins the game if sig^* is a valid proxy signature.

DEFINITION 3. A CL-PS scheme is said to be Type-II secure if there is no probabilistic polynomial-time adversary $\mathcal{A}_2$ which wins **Game II** with non-negligible advantage.

Security Requirements of Certificateless Proxy Signature Schemes

Like the general proxy signature, a certificateless proxy ring signature scheme should satisfy the following requirements.

1. **Distinguishability**: Proxy signatures are distinguishable from normal signatures by everyone.
2. **Verifiability**: From the proxy signature, the verifier can be convinced of the original signers agreement on the signed message.
3. **Strong Non-Forgeability**: A designated proxy signer can create a valid proxy signature for the original signer. But the original signer and other third parties who are not designated as a proxy signer cannot create a valid proxy signature.
4. **Strong Identifiability**: Anyone can determine the corresponding proxy signers from the proxy signature.
5. **Strong Non-Deniability**: Once a proxy signer creates a valid proxy signature of an original signer, he/she cannot repudiate the signature creation.
6. **Prevention of Misuse**: The proxy signer cannot use the proxy key for other purposes than generating a valid proxy signature. That is, it cannot sign messages that have not been authorized by the original signer.

3. Construction of Our Scheme

In this section, we will give the concrete construction of a certificateless proxy signature scheme. In our scheme, we employ some ideas of the certificateless signature scheme in Zhang and Wong (2006), and the ID-based proxy signature scheme in Shim (2006). The proposed certificateless proxy signature scheme comprises the following algorithms.

MasterKeyGen: Given a security parameter $k \in \mathbb{Z}$, the algorithm works as follows:

1. Run the parameter generator on input k to generate a prime q, two groups $\mathbb{G}_1$, $\mathbb{G}_2$ of prime order q, two different generator P and Q in $\mathbb{G}_1$ and an admissible pairing $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.
2. Select a master-key $s \in_R \mathbb{Z}_q^*$ and set $P_{pub} = sP$.
3. Choose cryptographic hash functions $H_1, H_3 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$. The security analysis will review H_1, H_2 and H_3 as random oracles. The system parameters is Params= $\{q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, Q, P_{pub}, H_1, H_2, H_3\}$. The master-key is s.

PartialKeyGen: Given a user's identity $ID \in \{0,1\}^*$, KGC first computes $Q_{ID} = H_1(ID)$. It then sets this user's partial key $psk_{ID} = sQ_{ID}$ and transmits it to ID secretly.

It is easy to see that psk_{ID} is actually a signature (Boneh *et al.*, 2001) on ID for the key pair (P_{pub}, s), and user ID can check its correctness by checking whether $\hat{e}(psk_{ID}, P) = \hat{e}(Q_{ID}, P_{pub})$.

UserKeyGen: The user ID selects a secret value $x_{ID} \in_R \mathbb{Z}_q^*$ as his secret key usk_{ID}, and computes his public key as $upk_{ID} = x_{ID}P$.

(Delegation, Proxy:)

1. The original signer, A prepares a warrant ω which is explicit description of the delegation relation.
2. On inputs Params, original singer A's identity ID_A, his partial key psk_{ID_A} and user secret key usk_{ID_A}, the signer A randomly chooses $r_A \in_R \mathbb{Z}_q^*$, computes

$U_A = r_A P$, $h_A = H_2(\omega, ID_A, upk_{ID_A}, U_A)$ and $V_A = h_A \cdot psk_{ID_A} + r_A Q + x_{ID_A} H_3(\omega, ID_A, upk_{ID_A})$, where $upk_{ID_A} = x_{ID_A} P$. Then A sends (ω, U_A, V_A) to the proxy signer B.

3. The proxy signer verifies whether $\hat{e}(V_A, P) = \hat{e}(h_A Q_{ID_A}, P_{pub})\hat{e}(U_A, Q)$ $\hat{e}(upk_{ID_A}, H_3(\omega, ID_A, upk_{ID_A}))$ holds or not. If it holds, B computes $h_B = H_2(\omega, ID_B, upk_{ID_B}, U_A)$ and $\sigma_P = V_A + h_B \cdot psk_{ID_B} + x_{ID_B} H_3(\omega, ID_B, upk_{ID_B})$ and keeps it as a proxy signing key.

Sign: Given its proxy signing key σ_P, and a message $m \in \{0,1\}^*$, B does:

1. Choose a random $r \in_R \mathbb{Z}_q^*$ and compute $U = rP$ and $h = H_2(\omega, m, ID_A, upk_{ID_A}, ID_B, upk_{ID_B}, U)$.
2. Compute $V = h \cdot \sigma_P + rQ$.
3. Output the proxy signature (ω, m, U_A, U, V).

Verify: Given Params, upk_{ID_A}, ID_A, upk_{ID_B}, ID_B, and proxy signature (ω, m, U_A, U, V) for the original signer ID_A and the proxy signer ID_B, a verifier does:

1. Compute $Q_{ID_A} = H_1(ID_A)$, $Q_{ID_B} = H_1(ID_B)$, $h_A = H_2(\omega, ID_A, upk_{ID_A}, U_A)$, $h_B = H_2(\omega, ID_B, upk_{ID_B}, U_A)$ and $h = H_2(\omega, m, ID_A, upk_{ID_A}, ID_B, upk_{ID_B}, U)$.
2. Verify whether $\hat{e}(V, P) = \hat{e}(h[h_A Q_{ID_A} + h_B Q_{ID_B}], P_{pub})\hat{e}(h(H_3(\omega, ID_A, upk_{ID_A}) + H_3(\omega, ID_B, upk_{ID_B})), upk_{ID_A} + upk_{ID_B})\hat{e}(Q, U + hU_A)$ holds or not. If it holds, accept the signature.

4. Security Analysis

4.1. *Unforgeability of the Scheme*

Theorem 1. *In the random oracle model, our certificateless proxy signature scheme is existentially unforgeable against adaptive chosen-message attacks under the assumption that the CDH problem in $\mathbb{G}_1$ in intractable.*

The theorem follows at once from Lemmas 1 and 2, according to Definitions 2 and 3.

Lemma 1. *If a probabilistic polynomial-time forger $\mathcal{A}_1$ has an advantage ε in forging a proxy signature in an attack modelled by **Game I** of Definition 2 after running in time t and making q_{H_i} queries to random oracles H_i for $i = 1,2,3$, q_{CreU} queries to the **CreateUser** request oracle, q_{RPar} queries to the **RevealPartialKey** extraction oracle, q_{RSec} queries to the **RevealSecretKey** extraction oracle, q_{PE} queries to the **RevealProxyKey** extraction oracle, and q_{Sig} queries to the **Sign** oracle, then the CDH problem can be solved with probability $\varepsilon' > \frac{1}{e} \cdot \frac{q_{RPar}+q_{Sig}}{(q_{RPar}+q_{Sig}+1)^2} \cdot \varepsilon$ with time $t' < t + (q_{H_1}+q_{H_2}+q_{H_3}+q_{RPar}+q_{CreU}+q_{RSec}+q_{PE}+q_{Sig})t_m + (q_{PE}+q_{Sig}+1)t_{mm}$, where t_m is the time to compute a scalar multiplication in $\mathbb{G}_1$ and t_{mm} is the time to perform a multi-exponentiation in $\mathbb{G}_1$.*

Proof. Let $(X = aP, Y = bP)$ be a random instance of the CDH problem in $\mathbb{G}_1$. Here P is a generator of $\mathbb{G}_1$, with prime order q, and the elements a, b are taken uniformly at random in $\mathbb{Z}_q^*$. By using the forgery algorithm $\mathcal{A}_1$, we will construct an algorithm $\mathcal{S}_1$ which outputs the CDH solution abP in $\mathbb{G}_1$.

Algorithm $\mathcal{S}_1$ chooses a random $t \in \mathbb{Z}_q^*$, and sets $P_{pub} = X$ and $Q = tP$, and then starts performing oracle simulation. Without loss of generality, we assume that, for any key extraction or signature query involving an identity, a $H_1(\cdot)$ oracle query has previously been made on that identity. And $\mathcal{S}_1$ maintains a list $L = \{(ID, psk_{ID}, upk_{ID}, usk_{ID})\}$ while $\mathcal{A}_1$ is making queries throughout the game. $\mathcal{S}_1$ responds to $\mathcal{A}_1$'s oracle as follows. □

Queries on Oracle H_1: When an identity ID is submitted to oracle H_1, $\mathcal{S}_1$ first flips a coin $W \in \{0, 1\}$ that yields 0 with probability ζ and 1 with probability $1 - \zeta$, and picks $t_1 \in \mathbb{Z}_q^*$ at random. If $W = 0$, then the hash value $H_1(ID)$ is defined as $t_1 P \in \mathbb{G}_1$. If $W = 1$, then $\mathcal{S}_1$ returned $t_1 Y \in \mathbb{G}_1$. In both cases, $\mathcal{S}_1$ inserts a tuple (ID, t_1, W) in a list $L_1 = \{(ID, t_1, W)\}$ to keep track the way it answered the queries.

Queries on Oracle H_2: Suppose $(\omega, ID, upk_{ID}, U)$ is submitted to oracle $H_2(\cdot)$. $\mathcal{S}_1$ first scans $L_2 = \{(\omega, ID, upk_{ID}, U, t_2, H_2)\}$ to check whether H_2 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_1$ picks at random $t_2 \in \mathbb{Z}_q^*$ and returns $H_2 = t_2$ as a hash value of $H_2(\omega, ID, upk_{ID}, U)$ to $\mathcal{A}_1$ and also stores the values in the list L_2.

Queries on Oracle H_3: Suppose (ω, ID, upk_{ID}) is submitted to oracle $H_3(\cdot)$. $\mathcal{S}_1$ first scans $L_3 = \{(\omega, ID, upk_{ID}, t_3, H_3)\}$ to check whether H_3 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_1$ picks at random $t_3 \in \mathbb{Z}_q^*$ and returns $H_3 = t_3 P \in \mathbb{G}_1$ as a hash value of $H_3(\omega, ID, upk_{ID})$ to $\mathcal{A}_1$ and also stores the values in the list L_3.

RevealPartialKey Oracle: Suppose the request is on an identity ID. $\mathcal{S}_1$ recovers the corresponding (ID, t_1, W) from the list L_1. If $W = 1$, then $\mathcal{S}_1$ outputs "failure" and halts because it is unable to coherently answer the query. Otherwise, $\mathcal{S}_1$ looks up the list L and performs as follows.

- If the list L contains $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ checks whether $psk_{ID} = \perp$. If $psk_{ID} \neq \perp$, $\mathcal{S}_1$ returns psk_{ID} to $\mathcal{S}_1$. If $psk_{ID} = \perp$, $\mathcal{S}_1$ recovers the corresponding (ID, t_1, W) from the list L_1. Noting $W = 0$ means that $H_1(ID)$ was previously defined to be $t_1 P \in \mathbb{G}_1$ and $psk_{ID} = t_1 P_{pub} = t_1 X \in \mathbb{G}_1$ is the partial key associated to ID. Thus $\mathcal{S}_1$ returns psk_{ID} to $\mathcal{A}_1$ and writes psk_{ID} in the list L.
- If the list L does not contain $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ recovers the corresponding (ID, t_1, W) from the list L_1, sets $psk_{ID} = t_1 P_{pub} = t_1 X$ and returns psk_{ID} to $\mathcal{A}_1$ and adds an element $(ID, psk_{ID}, upk_{ID}, usk_{ID})$ to the list L.

CreateUser Oracle: Suppose the request is on an identity ID.

- If the list L contains $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ checks whether $upk_{ID} = \perp$. If $upk_{ID} \neq \perp$, $\mathcal{S}_1$ returns upk_{ID} to $\mathcal{S}_1$. Otherwise, $\mathcal{S}_1$ randomly chooses $\nu \in \mathbb{Z}_q^*$ and $upk_{ID} = \nu P$ and $usk_{ID} = \nu$. $\mathcal{S}_1$ returns upk_{ID} to $\mathcal{A}_1$ and saves (upk_{ID}, usk_{ID}) into the list L.

- If the list L does not contain $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ sets $psk_{ID} = \perp$, and then randomly chooses $\nu \in \mathbb{Z}_q^*$ and sets $upk_{ID} = \nu P$ and $usk_{ID} = \nu$. $\mathcal{S}_1$ returns upk_{ID} to $\mathcal{A}_1$ and adds $(ID, psk_{ID}, upk_{ID}, usk_{ID})$ to the list L.

RevealSecretKey Oracle: Suppose the request is on an identity ID.

- If the list L contains $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ checks whether $usk_{ID} = \perp$. If $usk_{ID} \neq \perp$, $\mathcal{S}_1$ returns usk_{ID} to $\mathcal{S}_1$. Otherwise, $\mathcal{S}_1$ makes a **CreateUser** query itself to generate $(upk_{ID} = \nu P, usk_{ID} = \nu)$. Then $\mathcal{S}_1$ saves these values in the list L and returns $usk_{ID} = \nu$ to $\mathcal{A}_1$.
- If the list L does not contain $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ makes a **CreateUser** query itself, and then adds $(ID, psk_{ID}, upk_{ID}, usk_{ID})$ to the list L and returns usk_{ID}.

ReplaceKey Oracle: Suppose $\mathcal{A}_1$ makes the query with an input (ID, upk'_{ID}).

- If the list L contains an element $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ sets $upk_{ID} = upk'_{ID}$ and $usk_{ID} = \perp$.
- If the list L does not contain an item $(ID, psk_{ID}, upk_{ID}, usk_{ID})$, $\mathcal{S}_1$ sets $psk_{ID} = \perp$, $upk_{ID} = upk'_{ID}$ and $usk_{ID} = \perp$, and adds an element $(ID, psk_{ID}, upk_{ID}, usk_{ID})$ to L.

RevealProxyKey Oracle: Suppose $\mathcal{A}_1$ queries a proxy signing key with inputs $\{ID_i, ID_j, \omega\}$ (it means that an original signer ID_i designates ID_j as a proxy signer). $\mathcal{S}_1$ recovers the corresponding (ID_i, t_{1i}, W_i) and (ID_j, t_{1j}, W_j) from the list L_1. If $W_i = 1$ or $W_j = 1$, then $\mathcal{S}_1$ outputs "failure" and halts because it is unable to coherently answer the query. Otherwise $\mathcal{S}_1$ looks up the list L and performs as follows.

- If the list L contains $(ID_i, psk_{ID_i}, upk_{ID_i}, usk_{ID_i})$ and $(ID_j, psk_{ID_j}, upk_{ID_j}, usk_{ID_j})$, $\mathcal{S}_1$ checks whether $psk_{ID_i} = \perp$, $psk_{ID_j} = \perp$, $upk_{ID_i} = \perp$ and $upk_{ID_j} = \perp$. If $psk_{ID_i} = \perp$ or $psk_{ID_j} = \perp$, $\mathcal{S}_1$ makes the query to **Reveal-PartialKey Oracle** itself to obtain $psk_{ID_i} = t_{1i}P_{pub}$ or $psk_{ID_j} = t_{1j}P_{pub}$. If $upk_{ID_i} = \perp$ or $upk_{ID_j} = \perp$, $\mathcal{S}_1$ makes the query to **CreateUser Oracle** itself to generate $(usk_{ID_i} = \nu_i, upk_{ID_i} = \nu_i P)$ or $(usk_{ID_j} = \nu_j, upk_{ID_j} = \nu_j P)$. After that, $\mathcal{S}_1$ chooses $r_i, t_{2i}, t_{2j}, t_{3i}, t_{3j} \in \mathbb{Z}_q^*$ and computes $U_i = r_i P$. If the tuples containing t_{2i} and t_{2j} already appear in list L_2, and if the tuples containing t_{3i}, t_{3j} already appear in list L_3, then $\mathcal{S}_1$ chooses another $t_{2i}, t_{2j}, t_{3i}, t_{3j}$ and tries again. Then $\mathcal{S}_1$ computes

$$\sigma_P = t_{2i}(t_{1i}P_{pub}) + r_i Q + \nu_i t_{3i} P + t_{2j}(t_{1j}P_{pub}) + \nu_j t_{3j} P$$

  and stores $(\omega, ID_i, upk_{ID_i}, U_i, t_{2i}, H_{2i})$, $(\omega, ID_j, upk_{ID_j}, U_j, t_{2j}, H_{2j})$ in list L_2, and $(\omega, ID_i, upk_{ID_i}, t_{3i}, H_{3i})$, $(\omega, ID_j, upk_{ID_j}, t_{3j}, H_{3j})$ in list L_3, respectively. Finally, $\mathcal{S}_1$ responds to $\mathcal{A}_1$ with σ_P as ID_j's proxy signing key.
- If the list L does not contain the item $(ID_i, psk_{ID_i}, upk_{ID_i}, usk_{ID_i})$ or $(ID_j, psk_{ID_j}, upk_{ID_j}, usk_{ID_j})$, $\mathcal{S}_1$ makes queries to **RevealPartialKey Oracle** and **CreateUser Oracle** on ID_i or ID_j itself, and then adds $(ID_i, psk_{ID_i}, upk_{ID_i}, usk_{ID_i})$ or $(ID_j, psk_{ID_j}, upk_{ID_j}, usk_{ID_j})$ to the list L. Finally, $\mathcal{S}_1$ computes σ_P and returns it to $\mathcal{A}_1$ as before.

Sign Oracle: When $\mathcal{A}_1$ makes a **Sign-**query on m with $\{ID_i, ID_j, \omega\}$, $\mathcal{S}_1$ first finds the corresponding (ID_i, t_{1i}, W_i) and (ID_j, t_{1j}, W_j) from the list L_1. If $W_i = 1$ or $W_j = 1$, then $\mathcal{S}_1$ outputs "failure" and halts because it is unable to coherently answer the query. Otherwise $\mathcal{S}_1$ searches the list L and performs as follows.

- If the list L contains $(ID_i, psk_{ID_i}, upk_{ID_i}, usk_{ID_i})$ and $(ID_j, psk_{ID_j}, upk_{ID_j}, usk_{ID_j})$, $\mathcal{S}_1$ checks whether $psk_{ID_i} = \perp$, $psk_{ID_j} = \perp$, $upk_{ID_i} = \perp$ and $upk_{ID_j} = \perp$. If $psk_{ID_i} = \perp$ or $psk_{ID_j} = \perp$, $\mathcal{S}_1$ makes the query to **RevealPartialKey Oracle** itself to obtain $psk_{ID_i} = t_{1i}P_{pub}$ or $psk_{ID_j} = t_{1j}P_{pub}$. If $upk_{ID_i} = \perp$ or $upk_{ID_j} = \perp$, $\mathcal{S}_1$ makes the query to **CreateUser Oracle** itself to generate $(usk_{ID_i} = \nu_i, upk_{ID_i} = \nu_i P)$ or $(usk_{ID_j} = \nu_j, upk_{ID_j} = \nu_j P)$.
- Otherwise, if the list L does not contain the item $(ID_i, psk_{ID_i}, upk_{ID_i}, usk_{ID_i})$ or $(ID_j, psk_{ID_j}, upk_{ID_j}, usk_{ID_j})$, $\mathcal{S}_1$ makes queries to **RevealPartialKey Oracle** and **CreateUser Oracle** on ID_i or ID_j itself, and then adds $(ID_i, psk_{ID_i}, upk_{ID_i}, usk_{ID_i})$ or $(ID_j, psk_{ID_j}, upk_{ID_j}, usk_{ID_j})$ to the list L.

Then, $\mathcal{S}_1$ chooses $r_i, r_j \in \mathbb{Z}_q^*$ and computes $U_i = r_i P, U_j = r_j P$. After that $\mathcal{S}_1$ picks $t_{2i}, t_{2j}, t'_{2j}, t_{3i}, t_{3j} \in \mathbb{Z}_q^*$ randomly, and if the tuples containing t_{2i}, t_{2j} and t'_{2j} already appear in list L_2, or the tuples containing t_{3i} and t_{3j} already appear in list L_3, then $\mathcal{S}_1$ chooses another $t_{2i}, t_{2j}, t'_{2j}, t_{3i}, t_{3j}$ and tries again. Then $\mathcal{S}_1$ computes $V = t'_{2j}(t_{2i}(t_{1i}P_{pub}) + r_i Q + \nu_i t_{3i} P + t_{2j}(t_{1j}P_{pub}) + \nu_j t_{3j} P) + r_j Q$ and stores $(\omega, ID_i, upk_{ID_i}, U_i, t_{2i}, H_{2i})$, $(\omega, ID_j, upk_{ID_j}, U_j, t_{2j}, H_{2j})$ and $(\omega, m, ID_i, ID_j, upk_{ID_i}, upk_{ID_j}, U_j, t'_{2j}, H'_{2j})$ in list L_2, and $(\omega, ID_i, upk_{ID_i}, t_{3i}, H_{3i})$, $(\omega, ID_j, upk_{ID_j}, t_{3j}, H_{3j})$ in list L_3, respectively. Finally, $\mathcal{S}_1$ responds to $\mathcal{A}_1$ with $sig = (U_i, U_j, V)$.

All responses to **Sign** queries are valid, indeed, the output (ω, m, U_i, U_j, V) of **Sign** query is a valid proxy signature on m for $\{ID_i, ID_j, \omega\}$, to see this,

$\hat{e}(V, P) = \hat{e}(t'_{2j}(t_{2i}(t_{1i}P_{pub}) + r_i Q + \nu_i t_{3i} P + t_{2j}(t_{1j}P_{pub}) + \nu_j t_{3j} P) + r_j Q, P) = \hat{e}(P_{pub}, H'_{2j}(H_{2i}Q_{ID_i} + H_{2j}Q_{ID_j}))\hat{e}(H'_{2j}(H_{3i} + H_{3j}), upk_{ID_i} + upk_{ID_j})\hat{e}(Q, H'_{2j}U_i + U_j)$.

If $\mathcal{S}_1$ does not abort as a result of $\mathcal{A}_1$'s **Sign** queries, **CreateUser** queries, **RevealPartialKey** queries, **RevealSecretKey** queries and **RevealProxyKey** queries, then $\mathcal{A}_1$'s view is identical to its view in the real attack.

Eventually, $\mathcal{A}_1$ outputs a forgery $sig^* = (U_i^*, U_j^*, V^*)$ on a message m^* for $\{ID_i^*, ID_j^*, \omega^*\}$ with public key $\{upk_{ID_i^*}, upk_{ID_j^*}\}$. Now $\mathcal{S}_1$ recovers the corresponding $(ID_i^*, t_{1i}^*, W_i^*)$ and $(ID_j^*, t_{1j}^*, W_j^*)$ from the list L_1. If $W_i^* = 0$ or $W_j^* = 0$, then $\mathcal{S}_1$ outputs "failure" and stops. Otherwise, it goes on and finds out the items $(\omega^*, ID_i^*, upk_{ID_i^*}, U_i^*, t_{2i}^*, H_{2i}^*)$, $(\omega^*, ID_j^*, upk_{ID_j^*}, U_j^*, t_{2j}^*, H_{2j}^*)$ and $(\omega^*, m^*, ID_i^*, ID_j^*, upk_{ID_i^*}, upk_{ID_j^*}, U_j^*, t'^*_{2j}, H'^*_{2j})$ in the list L_2, and the items $(\omega^*, ID_i^*, upk_{ID_i^*}, t_{3i}^*, H_{3i}^*)$, $(\omega^*, ID_j^*, upk_{ID_j^*}, t_{3j}^*, H_{3j}^*)$ in list L_3. Note that the list L_2 and L_3 must contain such entries with overwhelming probability (otherwise, $\mathcal{S}_1$ stops and outputs "failure"). Note that $H_{2i}^* = H_2(\omega^*, ID_i^*, upk_{ID_i^*}, U_i^*)$ is $t_{2i}^* \in \mathbb{Z}_q^*$, $H_{2j}^* = H_2(\omega^*, ID_j^*, upk_{ID_j^*}, U_j^*)$ is $t_{2j}^* \in \mathbb{Z}_q^*$, $H'^*_{2j} = H_2(\omega^*, m^*, ID_i^*, ID_j^*, upk_{ID_i^*}, upk_{ID_j^*}, U_j^*)$ is $t'^*_{2j} \in \mathbb{Z}_q^*$ and $H_{3i}^* = H_3(\omega^*, ID_i^*, upk_{ID_i^*})$ is $t_{3i}^* P \in \mathbb{G}_1$, $H_{3j}^* = H_3(\omega^*, ID_j^*, upk_{ID_j^*})$ is $t_{3j}^* P \in \mathbb{G}_1$. If $\mathcal{A}_1$ succeeds in the game, then $\hat{e}(V^*, P) = \hat{e}(X, H'^*_{2j}(H_{2i}^* Q_{ID_i^*} + H_{2j}^* Q_{ID_j^*}))\hat{e}(H'^*_{2j}(H_{3i}^* +$

$H_{3j}^*), upk_{ID_i^*} + upk_{ID_j^*})\hat{e}(Q, H_{2j}'^* U_i^* + U_j^*)$ with $H_{2i}^* = t_{2i}^*$, $H_{2j}^* = t_{2j}^*$, $H_{2j}'^* = t_{2j}'^*$, $H_{3i}^* = t_{3i}^* P$, $H_{3j}^* = t_{3j}^* P$, $Q_{ID_i^*} = t_{1i}^* Y$, $Q_{ID_j^*} = t_{1j}^* Y$ and $Q = tP$ for known elements $t_{2i}^*, t_{2j}^*, t_{2j}'^*, t_{3i}^*, t_{3j}^*, t_{1i}^*, t_{1j}^*, t \in \mathbb{Z}_q^*$. Therefore, $\hat{e}(V^* - t_{2j}'^*(t_{3i}^* + t_{3j}^*)(upk_{ID_i^*} + upk_{ID_j^*}) - t(t_{2j}'^* U_i^* + U_j^*), P) = \hat{e}(X, t_{2j}'^*(t_{2i}^* t_{1i}^* + t_{2j}^* t_{1j}^*)Y)$ and thus $(t_{2j}'^*)^{-1}(t_{2i}^* t_{1i}^* + t_{2j}^* t_{1j}^*)^{-1}(V^* - t_{2j}'^*(t_{3i}^* + t_{3j}^*)(upk_{ID_i^*} + upk_{ID_j^*}) - t(t_{2j}'^* U_i^* + U_j^*))$ is the solution to the target CDH instance $(X, Y) \in \mathbb{G}_1 \times \mathbb{G}_1$.

Now, we evaluate $\mathcal{S}_1$'s probability of failure. By an analysis similar to Coron's technique (Coron, 2000), the probability $\zeta^{q_{RPar}+q_{Sig}}(1-\zeta)$ for $\mathcal{S}_1$ not to fail in key extraction queries or because $\mathcal{A}_1$ produces its forgery on a 'bad' identity ID^* is greater than $\frac{1}{e} \cdot (q_{RPar} + q_{Sig})$ when the optimal probability $\zeta_{opt} = (q_{RPar} + q_{Sig})/(q_{RPar} + q_{Sig} + 1)$ is taken. And, the probability $\mathcal{S}_1$ does not abort after $\mathcal{A}_1$ outputs a valid and nontrivial forgery is at least $(\frac{1}{q_{RPar}+q_{Sig}+1})^2$, since $\mathcal{S}_1$ succeeds only if $\mathcal{A}_1$ generates a forgery such that $W_i^* = 1$ and $W_j^* = 1$ for (ID_i^*, ID_j^*). Therefore, it results that $\mathcal{S}_1$'s advantage in solving the CDH problem in $\mathbb{G}_1$ is at least $\frac{1}{e} \cdot \frac{q_{RPar}+q_{Sig}}{(q_{RPar}+q_{Sig}+1)^2}$.

Lemma 2. *If a probabilistic polynomial-time forger $\mathcal{A}_2$ has an advantage ε in forging a proxy signature in an attack modelled by **Game II** of Definition 3 after running in time t and making q_{H_i} queries to random oracles H_i for $i = 2, 3$, q_{CreU} queries to the **CreateUser** request oracle, q_{RSec} queries to the **RevealSecretKey** extraction oracle, q_{PE} queries to the **RevealProxyKey** extraction oracle, and q_{Sig} queries to the **Sign** oracle, then the CDH problem can be solved with probability $\varepsilon' > \frac{1}{e} \cdot \frac{q_{RPar}+q_{Sig}}{(q_{RPar}+q_{Sig}+1)^2} \cdot \varepsilon$ with time $t' < t + (q_{H_2} + q_{H_3} + q_{CreU} + q_{RSec} + q_{PE} + q_{Sig})t_m + (q_{PE} + q_{Sig} + 1)t_{mm}$, where t_m is the time to compute a scalar multiplication in $\mathbb{G}_1$ and t_{mm} is the time to perform a multi-exponentiation in $\mathbb{G}_1$.*

Proof. Suppose $\mathcal{A}_2$ is a **Type II** adversary that (t, ε)-breaks our certificateless proxy signature scheme. We show how to construct a t'-time algorithm $\mathcal{S}_2$ that solves the CDH problem on $\mathbb{G}_1$ with probability at least ε'. Let $(X = aP, Y = bP) \in \mathbb{G}_1 \times \mathbb{G}_1$ be a random instance of the CDH problem taken as input by $\mathcal{S}_2$.

$\mathcal{S}_2$ randomly chooses $s \in \mathbb{Z}_q^*$ as the master key, and then initializes $\mathcal{A}_2$ with $P_{pub} = sP$ and also the master key s. After that, $\mathcal{S}_2$ chooses a random $t \in \mathbb{Z}_q^*$ and sets $Q = tP$. The adversary $\mathcal{A}_2$ then starts making oracle queries such as described in Definition 3. Note that the user's partial key $psk_{ID} = sH_1(ID)$ can be computed by both $\mathcal{S}_2$ and $\mathcal{A}_2$, thus the hash function $H_1(\cdot)$ is not modelled as a random oracle in this case.

$\mathcal{S}_2$ maintains a list $L = \{(ID, upk_{ID}, usk_{ID}, W)\}$, which does not need to be made in advance and is populated when $\mathcal{A}_2$ makes certain queries specified below. □

CreateUser Oracle: Suppose the request is on an identity ID.

- If the list L contains $(ID, upk_{ID}, usk_{ID}, W)$, $\mathcal{S}_2$ returns upk_{ID} to $\mathcal{A}_2$.
- If the list L does not contain $(ID, upk_{ID}, usk_{ID}, W)$, as in Coron's proof (Coron, 2000), $\mathcal{S}_2$ flips a coin $W \in \{0, 1\}$ that yields 0 with probability ζ and 1 with probability $1 - \zeta$. $\mathcal{S}_2$ also picks a number $t_1 \in \mathbb{Z}_q^*$ at random. If $W = 0$, the value of upk_{ID} is defined as $t_1 P \in \mathbb{G}_1$. If $W = 1$, $\mathcal{S}_2$ returns $t_1 X \in \mathbb{G}_1$. In both

cases, $\mathcal{S}_2$ sets $usk_{ID} = t_1$, and inserts a tuple $(ID, upk_{ID}, usk_{ID}, W)$ in a list $L = \{(ID, upk_{ID}, usk_{ID}, W)\}$ to keep track the way it answered the queries. $\mathcal{S}_2$ returns upk_{ID} to $\mathcal{A}_2$.

RevealSecretKey Oracle: Suppose the request is on an identity ID.

- If the list L contains $(ID, upk_{ID}, usk_{ID}, W)$, $\mathcal{S}_2$ returns usk_{ID} to $\mathcal{A}_2$ if $W = 0$, and halts otherwise.
- If the list L does not contain $(ID, upk_{ID}, usk_{ID}, W)$, $\mathcal{S}_2$ makes a **CreateUser** query itself, and then adds $(ID, upk_{ID}, usk_{ID}, W)$ to the list L. Then it returns usk_{ID} if $W = 0$, and halts otherwise.

Queries on Oracle H_2: Suppose $(\omega, ID, upk_{ID}, U)$ is submitted to oracle $H_2(\cdot)$. $\mathcal{S}_2$ first scans $L_2 = \{(\omega, ID, upk_{ID}, U, t_2, H_2)\}$ to check whether H_2 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_2$ picks at random $t_2 \in \mathbb{Z}_q^*$ and returns $H_2 = t_2$ as a hash value of $H_2(\omega, ID, upk_{ID}, U)$ to $\mathcal{A}_2$ and also stores the values in the list L_2.

Queries on Oracle H_3: Suppose (ω, ID, upk_{ID}) is submitted to oracle $H_3(\cdot)$. $\mathcal{S}_1$ first scans $L_3 = \{(\omega, ID, upk_{ID}, t_3, H_3)\}$ to check whether H_3 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_2$ picks at random $t_3 \in \mathbb{Z}_q^*$ and returns $H_3 = t_3 Y \in \mathbb{G}_1$ as a hash value of $H_3(\omega, ID, upk_{ID})$ to $\mathcal{A}_2$ and also stores the values in the list L_3.

RevealProxyKey Oracle: Suppose $\mathcal{A}_2$ queries a proxy signing key with inputs $\{ID_i, ID_j, \omega\}$. $\mathcal{S}_2$ first finds the corresponding $(ID_i, upk_{ID_i}, usk_{ID_i}, W_i)$ and $(ID_j, upk_{ID_j}, usk_{ID_j}, W_j)$ from the list L. If $W_i = 1$ or $W_j = 1$, then $\mathcal{S}_2$ outputs "failure" and halts because it is unable to coherently answer the query. Otherwise $\mathcal{S}_2$ chooses $r_i, t_{2i}, t_{2j}, t_{3i}, t_{3j} \in \mathbb{Z}_q^*$ and computes $U_i = r_i P$. If the tuples containing t_{2i} and t_{2j} already appear in list L_2, or the tuples containing t_{3i} and t_{3j} already appear in list L_3, then $\mathcal{S}_2$ chooses another $t_{2i}, t_{2j}, t_{3i}, t_{3j}$ and tries again. Then $\mathcal{S}_2$ computes

$$\sigma_P = t_{2i}\big(sH_1(ID_i)\big) + r_i Q + usk_{ID_i}(t_{3i}P) + t_{2j}\big(sH_1(ID_j)\big) + usk_{ID_j}(t_{3j}P)$$

and stores $(\omega, ID_i, upk_{ID_i}, U_i, t_{2i}, H_{2i})$, $(\omega, ID_j, upk_{ID_j}, U_j, t_{2j}, H_{2j})$ in list L_2, and $(\omega, ID_i, upk_{ID_i}, t_{3i}, H_{3i})$, $(\omega, ID_j, upk_{ID_j}, t_{3j}, H_{3j})$ in list L_3, respectively. Finally, $\mathcal{S}_2$ responds to $\mathcal{A}_2$ with σ_P as ID_j's proxy signing key.

Sign Oracle: When $\mathcal{A}_2$ makes a **Sign-**query on m with $\{ID_i, ID_j, \omega\}$, $\mathcal{S}_2$ first finds the corresponding $(ID_i, upk_{ID_i}, usk_{ID_i}, W_i)$ and $(ID_j, upk_{ID_j}, usk_{ID_j}, W_j)$ from the list L. If $W_i = 1$ or $W_j = 1$, then $\mathcal{S}_2$ outputs "failure" and halts because it is unable to coherently answer the query. Otherwise $\mathcal{S}_2$ chooses $r_i, r_j \in \mathbb{Z}_q^*$ and computes $U_i = r_i P, U_j = r_j P$. After that $\mathcal{S}_2$ picks $t_{2i}, t_{2j}, t'_{2j}, t_{3i}, t_{3j} \in \mathbb{Z}_q^*$ randomly, and if the tuples containing t_{2i}, t_{2j} and t'_{2j} already appear in list L_2, or the tuples containing t_{3i} and t_{3j} already appear in list L_3, then $\mathcal{S}_2$ chooses another $t_{2i}, t_{2j}, t'_{2j}, t_{3i}, t_{3j}$ and tries again. Then $\mathcal{S}_2$ computes $V = t'_{2j}(t_{2i}(sH_1(ID_i)) + r_i Q + usk_{ID_i}(t_{3i}P) + t_{2j}(sH_1(ID_j)) + usk_{ID_j}(t_{3j}P)) + r_j Q$ and stores $(\omega, ID_i, upk_{ID_i}, U_i, t_{2i}, H_{2i})$, $(\omega, ID_j, upk_{ID_j}, U_j, t_{2j}, H_{2j})$ and $(\omega, m, ID_i, ID_j, upk_{ID_i}, upk_{ID_j}, U_j, t'_{2j}, H'_{2j})$ in list

L_2, and $(\omega, ID_i, upk_{ID_i}, t_{3i}, H_{3i})$, $(\omega, ID_j, upk_{ID_j}, t_{3j}, H_{3j})$ in list L_3, respectively. Finally, $sig = (U_i, U_j, V)$ is returned to $\mathcal{A}_2$, which appears to be a valid signature since

$$\begin{aligned}\hat{e}(V,P) &= \hat{e}\big(t'_{2j}(t_{2i}(sH_1(ID_i)) + r_iQ + usk_{ID_i}(t_{3i}P) \\ &\quad + t_{2j}(sH_1(ID_j)) + usk_{ID_j}(t_{3j}P)) + r_jQ, P\big) \\ &= \hat{e}\big(P_{pub}, H'_{2j}(H_{2i}Q_{ID_i} + H_{2j}Q_{ID_j})\big) \\ &\quad \times \hat{e}\big(H'_{2j}(H_{3i} + H_{3j}), upk_{ID_i} + upk_{ID_j}\big)\hat{e}(Q, H'_{2j}U_i + U_j).\end{aligned}$$

Eventually, $\mathcal{A}_2$ outputs a forgery $sig^* = (U_i^*, U_j^*, V^*)$ on a message m^* for $\{ID_i^*, ID_j^*, \omega^*\}$ with public key $\{upk_{ID_i^*}, upk_{ID_j^*}\}$. Now $\mathcal{S}_2$ recovers the corresponding $(ID_i^*, upk_{ID_i^*}, usk_{ID_i^*}, W_i^*)$ and $(ID_j^*, upk_{ID_j^*}, usk_{ID_j^*}, W_j^*)$ from the list L. If $W_i^* = 0$ or $W_j^* = 0$, then $\mathcal{S}_2$ outputs "failure" and stops. Otherwise, it goes on and finds out the items $(\omega^*, ID_i^*, upk_{ID_i^*}, U_i^*, t_{2i}^*, H_{2i}^*)$, $(\omega^*, ID_j^*, upk_{ID_j^*}, U_j^*, t_{2j}^*, H_{2j}^*)$ and $(\omega^*, m^*, ID_i^*, ID_j^*, upk_{ID_i^*}, upk_{ID_j^*}, U_j^*, t'^*_{2j}, H'^*_{2j})$ in the list L_2, and the items $(\omega^*, ID_i^*, upk_{ID_i^*}, t_{3i}^*, H_{3i}^*)$, $(\omega^*, ID_j^*, upk_{ID_j^*}, t_{3j}^*, H_{3j}^*)$ in list L_3. Note that the list L_2 and L_3 must contain such entries with overwhelming probability. If $\mathcal{A}_2$ succeeds in the game, then $\hat{e}(V^*, P) = \hat{e}(P_{pub}, H'^*_{2j}(H_{2i}^*Q_{ID_i^*} + H_{2j}^*Q_{ID_j^*}))\hat{e}(H'^*_{2j}(H_{3i}^* + H_{3j}^*), (t_{1i}^* + t_{1j}^*)X)\hat{e}(Q, H'^*_{2j}U_i^* + U_j^*)$ with $H_{2i}^* = t_{2i}^*$, $H_{2j}^* = t_{2j}^*$, $H'^*_{2j} = t'^*_{2j}$, $H_{3i}^* = t_{3i}^*Y$, $H_{3j}^* = t_{3j}^*Y$ and $Q = tP$ for known elements $t_{2i}^*, t_{2j}^*, t'^*_{2j}, t_{3i}^*, t_{3j}^*, t_{1i}^*, t_{1j}^*, t \in \mathbb{Z}_q^*$. Therefore, $\hat{e}(V^* - st'^*_{2j}(t_{2i}^*Q_{ID_i^*} + t_{2j}^*Q_{ID_j^*}) - t(t'^*_{2j}U_i^* + U_j^*), P) = \hat{e}((t_{1i}^* + t_{1j}^*)X, t'^*_{2j}(t_{3i}^* + t_{3j}^*)Y)$ and thus $(t_{1i}^* + t_{1j}^*)^{-1}(t'^*_{2j}(t_{3i}^* + t_{3j}^*))^{-1}(V^* - st'^*_{2j}(t_{2i}^*Q_{ID_i^*} + t_{2j}^*Q_{ID_j^*}) - t(t'^*_{2j}U_i^* + U_j^*)$ is the solution to the target CDH instance (X, Y).

Now, we evaluate $\mathcal{S}_2$'s probability of failure. By an analysis similar to Coron's technique (Coron, 2000), the probability $\zeta^{q_{RPar}+q_{Sig}}(1-\zeta)$ for $\mathcal{S}_2$ not to fail in key extraction queries or because $\mathcal{A}_2$ produces its forgery on a 'bad' identity ID^* is greater than $\frac{1}{e}\cdot(q_{RPar}+q_{Sig})$ when the optimal probability $\zeta_{opt} = (q_{RPar}+q_{Sig})/(q_{RPar}+q_{Sig}+1)$ is taken. And, the probability $\mathcal{S}_1$ does not abort after $\mathcal{A}_2$ outputs a valid and nontrivial forgery is at least $(\frac{1}{q_{RPar}+q_{Sig}+1})^2$, since $\mathcal{S}_2$ succeeds only if $\mathcal{A}_1$ generates a forgery such that $W_i^* = 1$ and $W_j^* = 1$ for (ID_i^*, ID_j^*). Therefore, it results that $\mathcal{S}_2$'s advantage in solving the CDH problem in $\mathbb{G}_1$ is at least $\frac{1}{e}\cdot\frac{q_{RPar}+q_{Sig}}{(q_{RPar}+q_{Sig}+1)^2}$.

4.2. *Further Security Analysis*

Now, we show that our certificateless proxy signature scheme satisfies all the requirements described in the Section 2.

1. **Distinguishability**: This is obvious, because there is a warrant ω in a valid proxy signature, at the same time, this warrant ω and the public keys of the original signer and the proxy signers must occur in the verification equations of proxy signatures.
2. **Verifiability**: It derived from correctness of the proposed certificateless proxy signature scheme. In general, the warrant contains the identity information and the limitation of the delegated signing capacity and so satisfies the verifiability.
3. **Strong Non-Forgeability**: It derived from correctness of the Theorem 1.

4. **Strong Identifiability**: It contains the warrant ω in a valid proxy signature, so anyone can determine the identity of the corresponding proxy signers from the warrant ω.
5. **Strong Non-Deniability**: As the identifiability, the valid proxy signature contains the warrant ω, which must be verified in the verification phase, it cannot be modified by the proxy signer. Thus once a proxy signer creates a valid proxy signature of an original signer, he cannot repudiate the signature creation.
6. **Prevention of Misuse**: In our proxy signature scheme, using the warrant ω, we had determined the limit of the delegated signing capacity in the warrant ω, so the proxy signer cannot sign some messages that have not been authorized by the original signer.

5. Conclusion

The notion and security models of certificateless proxy signature are formalized. The models capture the essence of the possible adversaries in the notion of certificateless system and proxy signature. A concrete construction of certificateless proxy signature scheme from the bilinear maps is presented. The unforgeability of our CL-PS scheme is proved in the random oracle based on the hardness of Computational Diffie–Hellman problem. We note that CL-PS schemes may be more efficient than proxy signature schemes in traditional PKC since they avoid the costly computation for the verification of the public key certificates of the signers. And no key escrow in CL-PKC makes it impossible for the KGC to forge any valid proxy signatures.

References

Alomair, B., Sampigethaya, K., Poovendran, R. (2008). Efficient generic forward-secure signatures and proxy signatures. In: *EuroPKI*, *LNCS*, Vol. 5057. Springer, Berlin, pp. 166–181.

Al-Riyami, S.S., Paterson, K. (2003). Certificateless public key cryptography. In: *AsiaCrypt 2003*, *LNCS*, Vol. 2894. Springer, Berlin, pp. 452–473.

Boldyreva, A., Palacio, A., Warinschi, B. (2003). Secure proxy signature schemes for delegation of signing rights. *IACR ePrint Archive*. Available at: `http://eprint.iacr.org/2003/096`.

Boneh, D., Lynn, B., Shacham, H. (2001). Short signatures from the weil pairing. In: *AsiaCrypt 2001*, *LNCS*, Vol. 2248. Springer, Berlin, pp. 514–532. *J. Cryptography*, 17(4), 297–319.

Cao, X., Paterson, K.G., Kou, W. (2006). An attack on a certificateless signature scheme. In: *Cryptography ePrint Archive*. Available at: `http://eprint.iacr.org/2006/367`.

Chen, T.H., Horng G., Yang C.-S. (2008). Public key authentication schemes for local area networks. *Informatica*, 19(1), 3–16.

Coron, J.S. (2000). On the exact security of full domain hash. In: *Advances in Cryptology – CRYPTO 2000*, *LNCS*, Vol. 1880. Springer, Berlin, pp. 229–235.

Dent, A.W., Comley, R. (2006). Efficient certificateless encryption schemes and security models. *Cryptology ePrint Archive*, Report 2006/211,2006. `http://eprint.iacr.org/2006/211`.

Gorantla, M.C., Saxena, A. (2005). An efficient certificateless signature scheme. In: Hao, Y., Liu, J., Wang, Y.-P., Cheung, Y.-M., Yin, H., Jiao, L., Ma, J., Jiao, Y.-C. (Eds.), *CIS 2005*. *LNCS*, Vol. 3802. Springer, Berlin, pp. 110–116.

Hu, B.C., Wong, D.S. *et al.* (2006). Key replacement attack against a generic construction of certificateless signature. In: *Proc. of Information Security and Privacy*: *Australasian Conference, ACISP 2006*, *LNCS*, Vol. 4058. Springer, Berlin, pp. 235–246.

Huang, X.Y., Susilo, W. *et al.* (2003). On the security of certificateless signature schemes from Asiacrypt 2003. In: *Cryptology and Network Security*: *4th International Conference*, *LNCS*, Vol. 3810. Springer, Berlin, pp. 13–25.

Huang, X., Mu, Y. *et al.* (2007). Certificateless signature revisited. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (Eds.), *ACISP 2007*. *LNCS*, Vol. 4586. Springer, Berlin, pp. 308–322.

Kim, S., Park, S., Won, D. (1997). Proxy signatures, revisited. In: *International Conference on Information and Communications Security* (*ICICS'97*), *LNCS*, Vol. 1334. Springer, Berlin, pp. 223–232.

Kim, H., Baek, J., Lee, B., Kim, K. (2001). Secret computation with secrets for mobile agent using one-time proxy signature. In: *Cryptography and Information Security*.

Lee, B., Kim, H., Kim, K. (2001). Secure mobile agent using strong non-designated proxy signature. In: *ACISP*, *LNCS*, Vol. 2119. Springer, Berlin, pp. 474–486.

Li, X., Chen, K., Sun, L. (2005). Certificateless signature and proxy signature schemes from bilinear pairings. *Lith. Math. J.*, 45(1), 95–103.

Li, J., Yuen, T.H., Chen, X.F. *et al.* (2006). Proxy ring signature: Formal definitions, efficient construction and new variant. In: *Proc. International Conferene of Computational Intelligence and Security* (*CIS '06*), Vol. 2, pp. 1259–1264.

Lin, W.D., Jan, J.K. (2000). A security personal learning tools using a proxy blind signature scheme. In: *Proceedings of International Conference on Chinese Language Computing*, Illinois, USA, pp. 273–277.

Lu, R.B., He, D.K., Wang, C.J. (2007). Cryptanalysis and improvement of a certificateless proxy signature scheme from bilinear pairings. In: *International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, SNPD*, pp. 285–290.

Malkin, T., Obana, S., Yung, M. (2004). The hierarchy of key evolving signatures and a characterization of proxy signatures. In: *Advances in Cryptology – EUROCRYPT 2004*, *LNCS*, Vol. 3027. Springer, Berlin, pp. 306–322.

Mambo, M., Usuda, K., Okamoto, E. (1996). Proxy signatures for delegating signing operation. In: *3rd ACM Conference on Computer and Communications Security* (*CCS'96*), 48C57. ACM, pp. 48–57.

Pointcheval, D., Stern, J. (2000). Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3), 361–369.

Shamir, A. (1984). Identity-based cryptosystems and signature schemes. In: *Crypto 1984*, *LNCS*, Vol. 196. Springer, Berlin, pp. 47–53.

Shim, K.A. (2006). An identity-based proxy signature scheme from pairings. In: *International Conference on Information and Communications Security* (*ICICS'06*), *LNCS*, Vol. 4307. Springer, Berlin, pp. 60–71.

Tseng, Y.-M., Wu, T.-Y., Wu, J.-D. (2008). A pairing-based user authentication scheme for wireless clients with smart cards. *Informatica*, 19(2), 285–302.

Xu, J., Zhang, Z., Feng, D. (2005). ID-based proxy signature using bilinear pairings. In: *Parallel and Distributed Processing and Applications – ISPA 2005 Workshops*, *LNCS*, Vol. 3559. Springer, Berlin, pp. 359–367.

Yap, W.S., Heng, S.H., Goi, B.M. (2007). Cryptanalysis of some proxy signature schemes without certificates. In: *WISTP 2007*, *LNCS*, Vol. 4462. Springer, Berlin, pp. 115–126.

Yoon, E.J., Yoo, K.Y. (2009). Robust key exchange protocol between set-top box and smart card in DTV broadcasting. *Informatica*, 20(1), 139–150.

Yum, D.H., Lee, P.J. (2004). Generic construction of certificateless signature. In: *Proc. of Information Security and Privacy*: *9th Australasian Conference, ACISP 2004*, *LNCS*, Vol. 3108. Springer, Berlin, pp. 200–211.

Zhang, K. (1997). Threshold proxy signature schemes. In: *Proceedings of the First International Workshop on Information Security*, pp. 282–290.

Zhang, F., Kim, K. (2003). Efficient ID-based blind signature and proxy signature from bilinear pairings. In: *ACISP'03*, *LNCS*, Vol. 2727. Springer, Berlin, pp. 218–219.

Zhang, Z., Wong, D. (2006). Certificateless public-key signature: Security model and efficient construction. In: Zhou, J., Yung, M., Bao, F. (Eds.), *ACNS 2006*, *LNCS*, Vol. 3989. Springer, Berlin, pp. 293–308.

H. Xiong is a lecturer in the School of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC). He received his MS and PhD degree from UESTC in 2004 and 2009, respectively. His research interests include: information security and cryptography.

F. Li received his BS degree from Luoyang Institute of Technology in 2001 and MS degree from Hebei University of Technology in 2004. He is an associate professor in the School of Computer Science and Engineering, University of Electronic Science and Technology of China. His recent research interests include network security, mobile ad hoc network and cryptography.

Z. Qin is a professor in the School of Computer Science and Engineering, University of Electronic Science and Technology of China. He received his PhD degree from University of Electronic Science and Technology of China in 1996. His research interests include: information security and computer network.

Saugus įgaliotojo parašo algoritmas sertifikatų nenaudojančioje kriptografijoje

Hu XIONG, Fagen LI, Zhiguang QIN

Įgaliotojo parašo algoritmas įgalina asmenį perduoti savo pasirašymo teisę įgaliotajam asmeniui, kuris gali pasirašyti dokumentą įgaliojančiojo asmens vardu. Neseniai, norint nenaudoti sertifikatų viešojo rakto kriptografijoje, buvo pasiūlyta viešojo rakto be sertifikatų kriptografija. Straipsnyje nagrinėjama įgaliotojo parašo be sertifikatų algoritmas, kuriame panaudota efektyvi struktūra grįsta bitiesiniais poravimais (bilinear pairings). Gali būti įrodyta, kad pasiūlyto algoritmo saugumas yra ekvivalentus Diffie–Hellman'o uždavinio sprendimo sudėtingumui.