A Certificateless Proxy Ring Signature Scheme with Provable Security

Hu Xiong, Zhiguang Qin, and Fagen Li
(Corresponding author: Hu Xiong)

School of Computer Science and Engineering, University of Electronic Science and Technology of China
No. 4, Section 2, North Jianshe Road, Chengdu, 610054, China (Email: xionghu.uestc@gmail.com)
(Received Feb. 28, 2010; revised and accepted Apr. 7, 2010)

Abstract

Proxy ring signature allows proxy signer to sign messages on behalf of the original signer while providing anonymity. Certificateless public key cryptography was first introduced by Al-Riyami and Paterson in Asiacrypt 2003. In certificateless cryptography, it does not require the use of certificates to guarantee the authenticity of users' public keys. Meanwhile, certificateless cryptography does not have the key escrow problem, which seems to be inherent in the Identity-based cryptography. In this paper, we introduce the notion of proxy ring signature into certificateless public key cryptography and propose a concrete certificateless proxy ring signature scheme. The security models of certificateless proxy ring signature are also formalized. 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: Certificateless cryptography, provable security, proxy ring signature, random oracle model

1 Introduction

The concept of proxy signature was first introduced by Mambo, Usuda, and Okamoto in 1996 [24, 25]. 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 *el al.*s scheme, many proxy signature schemes have been proposed [2, 5, 19, 21, 26]. Proxy signatures can combine other special signatures to obtain some new types of proxy signatures [10, 18, 31]. These include threshold proxy signatures [23, 35], blind proxy signatures [20, 32], proxy ring signatures [4, 22] and one-time proxy signatures [17].

Ring signature, introduced by Rivest, Shamir and Tauman [27], is characterized by two main properties: anonymity and spontaneity. Anonymity in ring signature means 1-out-of-n signer verifiability, which enables the signer to keep anonymous in these "rings" of diverse signers [33]. Spontaneity is a property which makes distinction between ring signatures and group signatures [7]. In group signature schemes, there exists a trusted third party (TTP), usually known as the group manager, who handles the joining of group members by interacting with them. In ring signature schemes, no such trusted party exists and the rest of the $n-1$ members in the ring are totally unaware that they have been included in the ring. These two properties make ring signatures widely applicable to various cryptographic schemes. The survey of ring signatures and related applications can be found in [28, 29].

Proxy ring signatures [3, 22, 36] are designated for the following situation: an entity delegates his signing capability to many proxies, called proxy signer group. Any proxy signer can perform the signing operation on behalf of the original signer while providing anonymity, we can use group signature to solve it (take the group manger as the original entity). But in some applications, it is necessary to protect the privacy of participants (we believe that unconditional anonymity is necessary in many occasions). If the proxies hope that nobody (including the original signer) can open their identities, the group signature is not suitable for this situation. So the proxy ring signature is proposed to solve this problem [36]. On one hand, the proxy ring signature allows the proxy signer generates a proxy ring signature such that any verifier can be sure that the secret is indeed given out by the proxy signer group, on the other hand, nobody can figure out who the proxy signer is.

Certificateless public key cryptography (CL-PKC) is a new paradigm proposed by Al-Riyami and Paterson [1] 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 [1, 37], 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. 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 [1], Al-Riyami and Paterson presented a CLS scheme. Huang *et al.* [14] 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 [34] in ACISP 2004. However, Hu *et al* [15] showed that the Yum-Lee construction is insecure and proposed a fix in the standard model. In ACNS 2006, Zhang *et al.* [37] presented an efficient CLS scheme from pairings. Gorantla and Saxena [12] introduced a new construction of CLS scheme without providing formal proofs. Their scheme has been shown insecure by Cao *et al* [9]. The survey and discussions of CLS scheme can be found in [11, 15, 16].

As a useful primitive, proxy ring signature have been studied in traditional PKC and ID-PKC for more than several years. Even in a theoretic point of view, proxy ring signature should be studied in CL-PKC to rich the theories and techniques of CL-PKC. In practice, to generate a proxy ring signature on behalf of a group in traditional PKC, the signer must first verify all the certificates of the group members, otherwise his anonymity is jeopardized and the proxy ring signature will be rejected if he uses invalid certificates of some group members. Given a proxy ring signature, the verifier must perform the same verification as well before checking the validity of the proxy ring signature. These verifications inevitably lead to the inefficiency of the whole scheme since the computational cost increases linearly with the group size. Although Identity-based proxy ring signatures eliminate such costly verifications, they suffer from a security drawback induced by the inherent key escrow problem of ID-PKC. As CL-PKC does not use public key certificates, and in the meantime, it removes the key escrow problem of ID-PKC, we think it supplies an appropriate environment for implementing proxy ring signatures. So it is necessary to extend the notion and security model of proxy ring signatures to CL-PKC.

To the best of our knowledge, certificateless proxy ring signature based on bilinear pairings has not been treated in the literature. Our current work is aimed at filling this void. A security model for certificateless proxy ring 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 ring 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 ring 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 ring 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 Diffie-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 Diffie-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 $\mathrm{Succ}^{CDH}_{\mathcal{A},\mathbb{G}_1} = Pr[\mathcal{A}(P, aP, bP) = abP: a, b \in \mathbb{Z}_q]$. The CDH assumption states that for every probabilistic polynomial-time algorithm $\mathcal{A}$, $\mathrm{Succ}^{CDH}_{\mathcal{A},\mathbb{G}_1}$ is negligible.

2.2 Security Notions

COMPONENT OF CERTIFICATELESS PROXY RING SIGNATURE SCHEMES. A Certificateless Proxy Ring Signature (CL-PRS) scheme is a tuple CL-PRS=(MasterKeyGen, PartialKeyGen, UserKeyGen, Sign, Verify, (Delegation, Proxy), PRSign and PRVerify), 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 ring 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) The randomized standard signing algorithm Sign takes as input mpk, a message $m \in \{0, 1\}^*$, user secret key usk_{ID} and user partial key psk_{ID}, and outputs a signature sig on message m.

5) The deterministic verification algorithm Verify takes as input mpk, user identity ID, user public key upk_{ID}, message m and signature sig, and outputs True if the signature is correct, or $\perp$ otherwise.

6) (Delegation, Proxy) is a pair of interactive randomized algorithms forming the proxy-designation protocol. The input to each algorithm includes an identity ID_o and a set of identities $L_{ID} = \{ID_1, \cdots, ID_n\}$ with a warrant ω (the warrant made by the original signer ID_o is public and it implies that the original signer ID_o delegates L_{ID} as a set of proxy singers). Delegation is run by the original signer and it also takes as input the user secret key usk_{ID_o} and the user partial key psk_{ID_o} of the original signer. Proxy is run by the actual proxy signer and it also takes as input the user secret key usk_{ID_s} and the user partial key psk_{ID_s} of the actual proxy signer ID_s, where $ID_s \in \{ID_1, \cdots, ID_n\}$. As result of the interaction, a proxy signing key S_{ID_s}=(Delegation(ID_o, L_{ID}, ω, usk_{ID_o}, psk_{ID_o}), Proxy(ID_o, L_{ID}, ω, usk_{ID_s}, psk_{ID_s})) for ID_s is output.

7) The randomized proxy ring signing algorithm PRSign takes as input a message $m \in \{0, 1\}^*$, the identity ID_o and its corresponding public key upk_{ID_o} of original singer, a set of n proxy signers whose identities form the set $L_{ID} = \{ID_1, \cdots, ID_n\}$ and their corresponding public keys form the set $L_{upk} = \{upk_{ID_1}, \cdots, upk_{ID_n}\}$, the corresponding warrant ω, a proxy signing key S_{ID_s}, and outputs a proxy ring signature $prsig \leftarrow$ PRSign(S_{ID_s}, m, ω, ID_o, upk_{ID_o}, L_{ID}, L_{upk}).

8) The deterministic verification algorithm PRVerify takes as input mpk, the identity of original singer ID_o and its corresponding public key upk_{ID_o}, the set L_{ID} of the proxy signers' identities and the set L_{upk} of the corresponding public keys of the proxy signers, the corresponding warrant ω, a message $m \in \{0, 1\}^*$ and a proxy ring signature $prsig$, and outputs True if the signature is correct, or $\perp$ otherwise, i.e., {True, $\perp$} $\leftarrow$ PRVerify (ω, m, mpk, ID_o, upk_{ID_o}, L_{ID}, L_{upk}, $prsig$).

Adversaries Model of Certificateless Proxy Ring Signature Scheme. Combining the security notions of certificateless public key cryptography and security models of proxy ring signature schemes in traditional PKC and ID-PKC, we define two types of security for CL-PRS 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.

We define the security of a CL-PRS scheme via the following 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 MasterKeyGen(1^k) to get (mpk, msk). $\mathcal{S}_1$ then runs $\mathcal{A}_1$ on 1^k and mpk while keeping msk secret. In addition, $\mathcal{S}_1$ will maintain three lists L_1, L_2, L_3 where
   - L_1 is used to record the identities which have been chosen by $\mathcal{A}_1$ in the **RevealPartialKey** queries.
   - L_2 is used to record the identities whose public keys have been replaced by $\mathcal{A}_1$.
   - L_3 is used to record the identities which have been chosen by $\mathcal{A}_1$ in the **RevealSecretKey** queries.

   All these three lists L_1, L_2, L_3 are the empty set $\emptyset$ at the beginning of the game.

2) The adversary $\mathcal{A}_1$ can adaptively issue a polynomial bounded number of queries as defined below:
   - **CreateUser**: On input an identity $ID \in \{0, 1\}^*$, if ID has already been created, nothing is to be carried out. Otherwise, $\mathcal{S}_1$ generates (upk_{ID}, usk_{ID}) $\leftarrow$ UserKeyGen(mpk, ID). In both cases, upk_{ID} is returned.

- **RevealPartialKey**: On input an identity ID, $\mathcal{S}_1$ resets $L_1 = L_1 \bigcup \{ID\}$ and generates $psk_{ID} \leftarrow$ PartialKeyGen(msk, ID). $\mathcal{S}_1$ outputs the user partial key psk_{ID} as answer.
- **ReplaceKey**: For any user whose identity is ID, $\mathcal{A}_1$ can choose a new public key upk^*_{ID}. $\mathcal{A}_1$ then sets upk^*_{ID} as the new public key of this user and submits (ID, upk^*_{ID}) to $\mathcal{S}_1$. On receiving such a query, $\mathcal{S}_1$ resets $L_2 = L_2 \bigcup \{ID\}$ and updates the public key of this user to the new value upk^*_{ID}.
- **RevealSecretKey**: On input an identity ID, $\mathcal{S}_1$ first checks the set L_2. If $ID \in L_2$ (that is, the public key of the user ID has been replaced), $\mathcal{S}_1$ will return the symbol $\perp$ which means $\mathcal{S}_1$ cannot output the private key of an identity whose public key has been replaced. Otherwise, $ID \notin L_2$ and $\mathcal{S}_1$ resets $L_3 = L_3 \bigcup \{ID\}$. $\mathcal{S}_1$ then generates $(upk_{ID}, usk_{ID}) \leftarrow$ UserKeyGen(mpk, ID) and outputs the user secret key usk_{ID} as answer.
- **Sign**: On input a message $m \in \{0, 1\}^*$ and an identity ID, $\mathcal{S}_1$ outputs a standard signature sig on m for ID.
- **Delegation-Proxy**:
  a. $\mathcal{A}_1$ can request to interact with user ID_o, user ID_o playing the role of original signer, i.e., the original signer is user ID_o and the actual proxy signer is ID_s, where $ID_s \in \{ID_1, \cdots, ID_n\}$. $\mathcal{S}_1$ responses by running algorithm (Delegation, Proxy), taken warrant ω is chosen by $\mathcal{A}_1$ as input, and outputs a valid proxy signing key S_{ID_s}, if $\{ID_o\} \bigcap L_1 \bigcap L_2 = \emptyset$ and $\{ID_o\} \bigcap L_3 = \emptyset$.
  b. $\mathcal{A}_1$ can request to interact with the group of users $\{ID_1, \cdots, ID_n\}$, $\{ID_1, \cdots, ID_n\}$ playing the role of the set of proxy signers, and $\mathcal{A}_1$ playing the role of original signer. $\mathcal{S}_1$ responses by running algorithm (Delegation, Proxy), taken warrant ω is chosen by $\mathcal{A}_1$ as input, and outputs a valid proxy signing key S_{ID_s}, if $\{ID_1 \bigcap \cdots \bigcap ID_n\} \bigcap L_1 \bigcap L_2 = \emptyset$ and $\{ID_1 \bigcap \cdots \bigcap ID_n\} \bigcap L_3 = \emptyset$.
- **Proxy-Ring-Sign**: On input a message $m \in \{0, 1\}^*$ for $\{ID_o, L_{ID}\}$ with a warrant ω, $\mathcal{S}_1$ generates a valid proxy ring signature $prsig$ for m.

3) Eventually, $\mathcal{A}_1$ outputs a forge. The adversary $\mathcal{A}_1$ wins the game if any of the following events occurs:

   - $\mathcal{A}_1$ forges a standard signature (m^*, sig^*) of user ID^*, where sig^* is a valid signature and m^* has never been queried during the **Sign** queries. Note that ID^* cannot be an identity for which the user secret key has been extracted. Also, ID^* cannot be an identity for which both the public key has been replaced and the user partial key has been extracted.
   - $\mathcal{A}_1$ forges a proxy ring signature (m^*, ω^*, $prsig^*$) for the original signer ID^*_o and the set of proxy signers L^*_{ID} such that
     a. $prsig^*$ is a valid proxy ring signature.
     b. (m^*, ω^*) has never been queried during the **Proxy-Ring-Sign** queries.
     c. $\{ID^*_o, L^*_{ID}\}$ with a warrant ω^* is not requested to **Delegation-Proxy** query, i.e., L^*_{ID} was not designated by ID^*_o as a set of proxy signers.
     d. $L^*_{ID} \bigcap L_1 \bigcap L_2 = \emptyset$ and $L^*_{ID} \bigcap L_3 = \emptyset$.

Definition 2. *A CL-PRS 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 $\mathcal{A}_2$. $\mathcal{S}_2$ will maintain two lists L_1, L_2 where

   - L_1 is used to record the identities whose public keys have been replaced by $\mathcal{A}_2$.
   - L_2 is used to record the identities which have been chosen by $\mathcal{A}_2$ in the **RevealSecretKey** queries.

   Both two lists L_1, L_2 are empty at the beginning of the game.

2) As defined in **Game I**, $\mathcal{A}_2$ can issue a polynomially bounded number of **RevealSecretKey** queries, **Delegation-Proxy** queries, **ReplaceKey** queries **Sign** queries and **Proxy-Ring-Sign** queries. $\mathcal{A}_2$ can also make queries to **CreateUser**. $\mathcal{S}_2$ will answer those queries in the same way in **Game I**. Note that oracle **RevealPartialKey** is not accessible and no longer needed as $\mathcal{A}_2$ has the master secret key.

3) Eventually, $\mathcal{A}_2$ outputs a forge. The adversary $\mathcal{A}_2$ wins the game if any of the following events occurs:

   - $\mathcal{A}_2$ forges a standard signature (m^*, sig^*) of user ID^*, where sig^* is a valid signature and m^* has never been queried during the **Sign** queries. Note that ID^* cannot be an identity for which the user secret key has been extracted. Also, ID^* cannot be an identity for which both the public key has been replaced and the user partial key has been extracted.
   - $\mathcal{A}_2$ forges a proxy ring signature (m^*, ω^*, $prsig^*$) for the original signer ID^*_o and the set of proxy signers L^*_{ID} such that

a. $prsig^*$ is a valid proxy ring signature.

b. (m^*, ω^*) has never been queried during the **Proxy-Ring-Sign** queries.

c. $\{ID_o^*, L_{ID}^*\}$ with a warrant ω^* is not requested to **Delegation-Proxy** query, i.e., L_{ID}^* was not designated by ID_o^* as a set of proxy signers.

d. $L_{ID}^* \bigcap L_1 = \emptyset$ and $L_{ID}^* \bigcap L_2 = \emptyset$.

Definition 3. *A CL-PRS 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 Ring Signature Schemes. Like the general proxy ring signature, a certificateless proxy ring signature scheme should satisfy the following requirements.

1) **Distinguishability:** Proxy ring signatures are distinguishable from normal signatures by everyone.

2) **Verifiability:** From the proxy ring 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 ring 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 original signer and the set of proxy signers from the proxy ring signature.

5) **Signer-ambiguity:** No one except the actual signer himself can tell the identity of the actual signer with a probability large than $1/n$, where n is the cardinality of the ring, even if he/she has unlimited computing resources.

6) **Prevention of Misuse:** The proxy signer cannot use the proxy key for other purposes than generating a valid proxy ring 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 ring signature scheme. In our scheme, we employ some ideas of the certificateless signature scheme in [37], the ID-based ring signature scheme in [13], and the ID-based proxy signature scheme in [30]. The proposed certificateless proxy ring 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, a generator P in $\mathbb{G}_1$, an admissible pairing $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and $g = \hat{e}(P, P)$.

2) Select a master-key $\kappa \in_R \mathbb{Z}_q^*$ and set $P_{pub} = \kappa P$.

3) Choose cryptographic hash functions H_1, H_3, H_4: $\{0, 1\}^* \rightarrow \mathbb{G}_1$ and H_2, H_5, H_6: $\{0, 1\}^* \rightarrow \mathbb{Z}_q^*$. The security analysis will review H_1, H_2, H_3, H_4, H_5 and H_6 as random oracles. The system parameters is Params= $\{q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, P, g, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6\}$. The master-key is κ.

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 $psk_{ID} = \kappa Q_{ID}$ and transmits it to ID secretly.

It is easy to see that psk_{ID} is actually a signature [6] on ID for the key pair (P_{pub}, κ), 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 $upk_{ID} = x_{ID}P$.

Sign: On inputs Params, a message $m \in \{0, 1\}^*$, signers' identity ID, his partial key psk_{ID} and user secret key usk_{ID}, the signer randomly picks $r \in \mathbb{Z}_q^*$, computes $U = rP \in \mathbb{G}_1$, $h = H_2(m, U)$, and $V = h \cdot psk_{ID} + rH_3(m, ID, upk_{ID}, U) + x_{ID}H_4(m, ID, upk_{ID}) \in \mathbb{G}_1$. The signature is $sig = (U, V)$.

Verify: Given Params, message m, ID and signature $sig = (U, V)$, the algorithm accepts the signature if the following equations holds:

$$\hat{e}(V, P) = \hat{e}(hQ_{ID}, P_{pub})\hat{e}(U, H_3(m, ID, upk_{ID}, U)) \hat{e}(upk_{ID}, H_4(m, ID, upk_{ID})).$$

Delegation, Proxy:

1) Delegation Generation: The original signer publishes a warrant ω where there is an explicit description of the delegation relation including the identity of the original signer ID_o and a group of n proxy signers whose identities form the set $L_{ID} = \{ID_1, \cdots, ID_n\}$. Note that the corresponding public keys of the proxy signers form the set $L_{upk} = \{upk_{ID_1}, \cdots, upk_{ID_n}\}$. The original signer ID_o chooses $r_o \in \mathbb{Z}_q^*$ and computes $U_o = r_oP \in \mathbb{G}_1$, $h_o = H_2(\omega, U_o)$, and $V_o = h_o \cdot psk_{ID_o} + r_oH_3(\omega, ID_o, upk_{ID_o}, U_o) + x_{ID_o}H_4(\omega, ID_o, upk_{ID_o}) \in \mathbb{G}_1$. Then the original signer broadcasts (ω, U_o, V_o) to the set of proxy signers.

2) Delegation Verification: The proxy signer ID_s, where $ID_s \in \{ID_1, \cdots, ID_n\}$, verifies whether

$\hat{e}(V_o,\ P) = \hat{e}(h_o Q_{ID_o},\ P_{pub})\ \hat{e}(U_o,\ H_3(\omega,\ ID_o,\ upk_{ID_o},\ U_o))\ \hat{e}(upk_{ID_o},\ H_4(\omega,\ ID_o,\ upk_{ID_o}))$ holds or not.

3) Proxy Key Generation: If it holds, ID_s computes $h'_o = H_5(\omega,\ U_o)$ and $S_{ID_s} = V_o + h'_o psk_{ID_s} + x_{ID_s} H_4(\omega,\ L_{ID},\ L_{upk}) \in \mathbb{G}_1$ and keeps it as a proxy signing key.

PRSign: To sign a message $m \in \{0, 1\}^*$ on behalf of the proxy signers, the actual signer, indexed by s using the proxy signing key S_{ID_s}, performs the following steps.

1) For all $i \in \{1,\ \cdots,\ n\}$, $i \neq s$, choose $r_i \in \mathbb{Z}_q^*$ uniformly at random, compute $y_i = g^{r_i}$.

2) Compute $h_i = H_6(\omega,\ m,\ ID_o,\ upk_{ID_o},\ L_{ID},\ L_{upk},\ y_i)$ for all $i \in \{1,\ \cdots,\ n\}$, $i \neq s$.

3) Choose random $r_s \in \mathbb{Z}_q^*$, compute $y_s = g^{r_s}\hat{e}(-U_o,\ \sum_{i\neq s} h_i H_3(\omega,\ ID_o,\ upk_{ID_o},\ U_o))\ \hat{e}(-P_{pub},\ \sum_{i\neq s} h_i\ h_o\ Q_{ID_o})\ \hat{e}(-P_{pub},\ \sum_{i\neq s} h'_o\ h_i\ Q_{ID_i})\ \hat{e}(-upk_{ID_o},\ \sum_{i\neq s} h_i\ H_4(\omega,\ ID_o,\ upk_{ID_o}))\ \hat{e}\ (-H_4(\omega,\ L_{ID},\ L_{upk}),\ \sum_{i\neq s} h_i\ upk_{ID_i})$. If $y_s = 1_{\mathbb{G}_2}$ or $y_s = y_i$ for some $i \neq s$, then go to the previous step.

4) Compute $h_s = H_6(\omega,\ m,\ ID_o,\ upk_{ID_o},\ L_{ID},\ L_{upk},\ y_s)$.

5) Compute $V = \sum_{i=1}^{n} r_i P + h_s S_{ID_s}$.

6) Output the proxy ring signature $prsig = (y_1,\ \ldots,\ y_n,\ V,\ U_o)$.

PRVerify: To verify a proxy ring signature $prsig = (y_1,\ \ldots,\ y_n,\ V,\ U_o)$ on a message m with original signer ID_o, the set of proxy singers L_{ID}, and the corresponding ω, a verifier does:

1) Compute $h_i = H_6(\omega,\ m,\ ID_o,\ upk_{ID_o},\ L_{ID},\ L_{upk},\ y_i)$ for all $i \in \{1,\ \ldots,\ n\}$.

2) Compute $h_o = H_2(\omega,\ U_o)$ and $h'_o = H_5(\omega,\ U_o)$.

3) Verify whether $\hat{e}(V,\ P) = y_1\ \cdots\ y_n\hat{e}(\sum_{i=1}^{n} h_i\ H_3(\omega,\ ID_o,\ upk_{ID_o},\ U_o),\ U_o)\hat{e}(\sum_{i=1}^{n} h_i h_o Q_{ID_o},\ P_{pub})\ \hat{e}(h'_o \sum_{i=1}^{n} h_i Q_{ID_i},\ P_{pub})\hat{e}(\sum_{i=1}^{n} h_i\ H_4(\omega,\ ID_o,\ upk_{ID_o}),\ upk_{ID_o})\hat{e}(\sum_{i=1}^{n} h_i upk_{ID_i},\ H_4(\omega,\ L_{ID},\ L_{upk}))$ holds or not. If it holds, accept the signature.

4 Security Analysis

4.1 Correctness and Signer Ambiguity

The property of correctness is satisfied. In effect, if the proxy ring signature has been correctly generated, then

$$\begin{aligned}
\hat{e}(V,P) &= \hat{e}(\sum_{i=1}^{n} r_i P + h_s S_{ID_s}, P) \\
&= \hat{e}(\sum_{i=1}^{n} r_i P, P)\hat{e}(h_s S_{ID_s}, P) \\
&= y_1 \cdots y_n \hat{e}(U_o, \sum_{i\neq s} h_i H_3(\omega, ID_o, upk_{ID_o}, U_o)) \\
&\quad \hat{e}(P_{pub}, \sum_{i\neq s} h_i h_o Q_{ID_o})\hat{e}(P_{pub}, h'_o \sum_{i\neq s} h_i Q_{ID_i}) \\
&\quad \hat{e}(upk_{ID_o}, \sum_{i\neq s} h_i H_4(\omega, ID_o, upk_{ID_o})) \\
&\quad \hat{e}(H_4(\omega, L_{ID}, L_{upk}), \sum_{i\neq s} h_i upk_{ID_i} + h_s upk_{ID_s}) \\
&\quad \hat{e}(h_s h'_o Q_{ID_s}, P_{pub}) \\
&\quad \hat{e}(upk_{ID_o}, h_s H_4(\omega, ID_o, upk_{ID_o})) \\
&\quad \hat{e}(h_s H_3(\omega, ID_o, upk_{ID_o}, U_o), U_o) \\
&\quad \hat{e}(h_s h_o Q_{ID_o}, P_{pub}) \\
&= y_1 \cdots y_n \hat{e}(\sum_{i=1}^{n} h_i H_3(\omega, ID_o, upk_{ID_o}, U_o), U_o) \\
&\quad \hat{e}(\sum_{i=1}^{n} h_i h_o Q_{ID_o}, P_{pub})\hat{e}(h'_o \sum_{i=1}^{n} h_i Q_{ID_i}, P_{pub}) \\
&\quad \hat{e}(\sum_{i=1}^{n} h_i H_4(\omega, ID_o, upk_{ID_o}), upk_{ID_o}) \\
&\quad \hat{e}(\sum_{i=1}^{n} h_i upk_{ID_i}, H_4(\omega, L_{ID}, L_{upk})).
\end{aligned}$$

With respect to the anonymity of the scheme, we can argue as follows: let $(\omega,\ m,\ y_1,\ \ldots,\ y_n,\ V)$ be a valid proxy ring signature of a message m on behalf of the original signer ID_o, n proxy signers specified by identities in L_{ID} and public keys in L_{upk}. Since all the r_i, $i \in \{1,\ \cdots,\ n\}\backslash\{s\}$ are randomly generated, hence all y_i, $i \in \{1,\ \cdots,\ n\}\backslash\{s\}$ are also uniformly distributed. The randomness of r_s chosen by the signer implies $y_s = g^{r_s}\hat{e}(-U_o, \sum_{i\neq s} h_i H_3(\omega,\ ID_o,\ upk_{ID_o},\ U_o))\hat{e}(-P_{pub},\ \sum_{i\neq s} h_i h_o Q_{ID_o})\hat{e}(-P_{pub},\ \sum_{i\neq s} h'_o h_i Q_{ID_i})\hat{e}(-upk_{ID_o},\ \sum_{i\neq s} h_i H_4(\omega,\ ID_o,\ upk_{ID_o}))\hat{e}(-H_4(\omega,\ L_{ID},\ L_{upk}),\ \sum_{i\neq s} h_i upk_{ID_i})$ is also uniformly distributed. So $(y_1,\ \ldots,\ y_n)$ in the signature reveals no information about the signer.

It remains to consider whether $V = \sum_{i=1}^{n} r_i P + h_s S_{ID_s}$ leaks information about the actual signer. From the construction of V, it is obvious to see that $S_{ID_s} = h_s^{-1}(V - \sum_{i=1}^{n} r_i P)$. To identify whether ID_s is the identity of the actual signer, the only way is to check $\hat{e}(h_o Q_{ID_o},\ P_{pub})\hat{e}(U_o,\ H_3(\omega,\ ID_o,\ upk_{ID_o},$

$U_o))\hat{e}(upk_{ID_o},\ H_4(\omega,\ ID_o,\ upk_{ID_o}))\hat{e}(upk_{ID_s},\ H_4(\omega,\ L_{ID},\ L_{upk}))\hat{e}(h'_oQ_{ID_s},\ P_{pub}) \stackrel{?}{=} \hat{e}(S_{ID_s},\ P)$. Namely, $\hat{e}(h_oQ_{ID_o},\ P_{pub})\hat{e}(U_o,\ H_3(\omega,\ ID_o,\ upk_{ID_o},\ U_o))\hat{e}(upk_{ID_o},\ H_4(\omega,\ ID_o,\ upk_{ID_o}))\hat{e}(h'_oQ_{ID_s},\ P_{pub})\hat{e}(upk_{ID_s},\ H_4(\omega,\ L_{ID},\ L_{upk})) \stackrel{?}{=} \hat{e}(h_s^{-1}(V - \sum_{i=1}^n r_iP),\ P)$. If ID_s is the identity of the actual signer, it should hold $y_s = g^{r_s}\hat{e}(-U_o, \sum_{i\neq s} h_iH_3(\omega, ID_o, upk_{ID_o}, U_o))\hat{e}(-P_{pub}, \sum_{i\neq s} h_ih_oQ_{ID_o})\hat{e}(-P_{pub},\ \sum_{i\neq s} h'_oh_iQ_{ID_i})\hat{e}(-upk_{ID_o}, \sum_{i\neq s} h_iH_4(\omega,\ ID_o,\ upk_{ID_o}))\hat{e}(-H_4(\omega,\ L_{ID},\ L_{upk}), \sum_{i\neq s} h_iupk_{ID_i})$.

It remains to check

$$(\frac{\hat{e}(V,P)}{\rho})^{h_s^{-1}} \stackrel{?}{=} \hat{e}(h_oQ_{ID_o},P_{pub})\hat{e}(U_o,H_3)\hat{e}(upk_{ID_o},H_4)\hat{e}(h'_oQ_{ID_s},P_{pub})\hat{e}(upk_{ID_s},H'_4)$$

where

$$\begin{aligned} H_3 &= H_3(\omega, ID_o, upk_{ID_o}, U_o),\\ H_4 &= H_4(\omega, ID_o, upk_{ID_o}),\\ H'_4 &= H_4(\omega, L_{ID}, L_{upk}), \text{ and}\\ \rho &= y_1\cdots y_n\hat{e}(U_o,\sum_{i\neq s}h_iH_3)\hat{e}(P_{pub},\sum_{i\neq s}h_ih_oQ_{ID_o})\\ &\quad \hat{e}(P_{pub},\sum_{i\neq s}h'_oh_iQ_{ID_i})\hat{e}(upk_{ID_o},\sum_{i\neq s}h_iH_4)\\ &\quad \hat{e}(H'_4,\sum_{i\neq s}h_iupk_{ID_i}). \end{aligned}$$

However, we have for each $j \in \{1, 2, \ldots, n\}$

$$\begin{aligned} (\frac{\hat{e}(V,P)}{\xi_0})^{h_s^{-1}} &= (\frac{\hat{e}(\sum_{i=1}^n r_iP + h_sS_{ID_s},P)}{\xi_0\xi_1\xi_2})^{h_j^{-1}}\\ &= (\frac{\hat{e}(\sum_{i=1}^n r_iP,P)}{\xi_0\xi_2})^{h_j^{-1}}\\ &= (\frac{\prod_{i\neq s} y_i\cdot g^{r_s}}{\xi_0\xi_2})^{h_j^{-1}}\\ &= (\xi_2)^{-h_j^{-1}}\\ &= \hat{e}(h_oQ_{ID_o},P_{pub})\hat{e}(U_o,H_3)\\ &\quad \hat{e}(upk_{ID_o},H_4)\hat{e}(h'_oQ_{ID_j},P_{pub})\\ &\quad \hat{e}(upk_{ID_j},H'_4) \end{aligned}$$

where

$$\begin{aligned} \xi_0 &= y_1\cdots y_n\hat{e}(U_o,\sum_{i\neq j}h_iH_3)\hat{e}(P_{pub},\sum_{i\neq j}h_ih_oQ_{ID_o})\\ &\quad \hat{e}(P_{pub},\sum_{i\neq j}h'_oh_iQ_{ID_i})\hat{e}(upk_{ID_o},\sum_{i\neq j}h_iH_4)\\ &\quad \hat{e}(H'_4,\sum_{i\neq j}h_iupk_{ID_i}),\\ \xi_1 &= \hat{e}(U_o,h_sH_3)\hat{e}(P_{pub},h_sh_oQ_{ID_o})\hat{e}(P_{pub},h'_oh_sQ_{ID_s})\\ &\quad \hat{e}(upk_{ID_o},h_sH_4)\hat{e}(H'_4,h_supk_{ID_s}),\\ \xi_2 &= \hat{e}(-U_o,h_jH_3)\hat{e}(-P_{pub},h_jh_oQ_{ID_o})\\ &\quad \hat{e}(-P_{pub},h'_oh_jQ_{ID_j})\hat{e}(-upk_{ID_o},h_jH_4)\\ &\quad \hat{e}(-H'_4,h_jupk_{ID_j}) \end{aligned}$$

and ID_s is the identity of the actual signer. This fact shows that V in the signature does not leak any information about the identity of the actual signer. And hence, the unconditional anonymity of our CL-PRS scheme is proved.

4.2 Unforgeability of the Scheme

Theorem 1. *In the random oracle model, our certificateless proxy ring 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 ring 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 $i = 1, 2, 3, 4, 5, 6$, 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_{DP} queries to the **Delegation-Proxy** extraction oracle, q_{Sig} queries to the **Sign** oracle, and q_{PRSig} queries to the **Proxy-Ring-Sign** oracle, then the CDH problem can be solved with probability $\varepsilon' > (1 - \frac{1}{e(q_{RPar}+1)})\varepsilon + \frac{((\frac{q_{RPar}}{q_{RPar}+n+1})^{q_{RPar}}(\frac{n+1}{q_{RPar}+n+1})^{n+1}\cdot\varepsilon)^2}{66C_{q_{H_1},n}}$ with time $t' < 2(t + q_{H_1}T_1 + q_{H_2}T_2 + q_{H_3}T_3 + q_{H_4}T_4 + q_{H_5}T_5 + q_{H_6}T_6 + q_{RPar}T_{RPar} + q_{CreU}T_{CreU} + q_{RSec}T_{RSec} + q_{DP}T_{DP} + q_{Sig}T_{Sig} + q_{PRSig}T_{PRSig})$, where $C_{q_{H_1},n}$ is defined as the number of n-permutations of q_{H_1} elements i.e. $C_{q_{H_1},n} = q_{H_1}\cdot\ldots\cdot(q_{H_1}-n+2)(q_{H_1}-n+1)$, T_1(resp. T_2, T_3, T_4, T_5, T_{RPar}, T_{CreU}, T_{RSec}, T_{DP}, T_{Sig} and T_{PRSig}) is the time cost of an H_1(resp. H_2, H_3, H_4, H_5, **RevealPartialKey**, **CreateUser**, **RevealSecretKey**, **Delegation-Proxy**, **Sign** and **Proxy-Ring-Sign** query).*

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 $\mathcal{S}_1$ sets $P_{pub} = X$ 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. $\mathcal{S}_1$ maintains a list $L = \{(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})\}$ while $\mathcal{A}_1$ is making queries throughout the game. $\mathcal{S}_1$ also keeps three lists L_1, L_2, L_3, the functions of these lists are the same as mentioned in **Game I** in Section 2. $\mathcal{S}_1$ 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 $t_1P \in \mathbb{G}_1$. If $W = 1$, then $\mathcal{S}_1$ returned $t_1Y \in \mathbb{G}_1$. In both cases, $\mathcal{S}_1$ inserts a tuple $(ID,\ t_1,\ W)$ in a list $\mathbf{H}_1 = \{(ID,\ t_1,\ W)\}$ to keep track the way it answered the queries.

Queries on Oracle H_2: Suppose $(m,\ U)$ is submitted to oracle $H_2(\cdot)$. $\mathcal{S}_1$ first scans list $\mathbf{H}_2 = \{(m,\ U,\ t_2,\ H_2)\}$ 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 $H_2 = t_2 \in \mathbb{Z}_q^*$ as a hash value of $H_2(m,\ U)$ to $\mathcal{A}_1$ and also stores the values in the list $\mathbf{H}_2$.

Queries on Oracle H_3: Suppose $(m,\ ID,\ upk_{ID},\ U)$ is submitted to oracle $H_3(\cdot)$. $\mathcal{S}_1$ first scans $\mathbf{H}_3 = \{(m,\ ID,\ upk_{ID},\ U,\ t_3,\ H_3)\}$ 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 $H_3 = t_3P \in \mathbb{G}_1$ as a hash value of $H_3(m,\ ID,\ upk_{ID},\ U)$ to $\mathcal{A}_1$ and also stores the values in the list $\mathbf{H}_3$.

Queries on Oracle H_4: Suppose $(m,\ ID,\ upk_{ID})$ is submitted to oracle $H_4(\cdot)$. $\mathcal{S}_1$ first scans $\mathbf{H}_4 = \{(m,\ ID,\ upk_{ID},\ t_4,\ H_4)\}$ to check whether H_4 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_1$ picks at random $t_4 \in \mathbb{Z}_q^*$ and returns $H_4 = t_4P \in \mathbb{G}_1$ as a hash value of $H_4(m,\ ID,\ upk_{ID})$ to $\mathcal{A}_1$ and also stores the values in the list $\mathbf{H}_4$.

Queries on Oracle H_5: Suppose $(\omega,\ U)$ is submitted to oracle $H_5(\cdot)$. $\mathcal{S}_1$ first scans list $\mathbf{H}_5 = \{(\omega,\ U,\ t_5,\ H_5)\}$ to check whether H_4 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_1$ picks at random $t_5 \in \mathbb{Z}_q^*$ and returns $H_5 = t_5 \in \mathbb{Z}_q^*$ as a hash value of $H_5(\omega,\ U)$ to $\mathcal{A}_1$ and also stores the values in the list $\mathbf{H}_5$.

Queries on Oracle H_6: Suppose $(\omega,\ m,\ ID,\ upk_{ID},\ L_{ID},\ L_{upk},\ y)$ is submitted to oracle $H_6(\cdot)$. $\mathcal{S}_1$ first scans list $\mathbf{H}_6 = \{(\omega,\ m,\ ID,\ upk_{ID},\ L_{ID},\ L_{upk},\ y,\ t_6,\ H_6)\}$ to check whether H_6 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_1$ picks at random $t_6 \in \mathbb{Z}_q^*$ and returns $H_6 = t_6 \in \mathbb{Z}_q^*$ as a hash value of $H_6(\omega,\ m,\ ID,\ upk_{ID},\ L_{ID},\ L_{upk},\ y)$ to $\mathcal{A}_1$ and also stores the values in the list $\mathbf{H}_6$.

RevealPartialKey Oracle: Suppose the request is on an identity ID. $\mathcal{S}_1$ recovers the corresponding $(ID,\ t_1,\ W)$ from the list $\mathbf{H}_1$. 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.

1) 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 $\mathbf{H}_1$. Noting $W = 0$ means that $H_1(ID)$ was previously defined to be $t_1P \in \mathbb{G}_1$ and $psk_{ID} = t_1P_{pub} = t_1X \in \mathbb{G}_1$ is the partial key associated to ID. Thus $\mathcal{S}_1$ returns psk_{ID} to $\mathcal{A}_1$, writes psk_{ID} in the list L and sets $L_1 = L_1 \bigcup \{ID\}$.

2) 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 $\mathbf{H}_1$, sets $psk_{ID} = t_1P_{pub} = t_1X$ and returns psk_{ID} to $\mathcal{A}_1$, adds an element $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$ to the list L and sets $L_1 = L_1 \bigcup \{ID\}$.

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

1) 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 sets $upk_{ID} = \nu P$, $usk_{ID} = \nu$. $\mathcal{S}_1$ returns upk_{ID} to $\mathcal{A}_1$ and saves (upk_{ID}, usk_{ID}) into the list L.

2) 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.

ReplaceKey Oracle: Suppose $\mathcal{A}_1$ makes the query with an input $(ID,\ upk'_{ID})$, then $\mathcal{S}_1$ sets $L_2 = L_2 \bigcup \{ID\}$.

1) If the list L contains an element $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$, $\mathcal{S}_1$ sets $upk_{ID} = upk'_{ID}$ and $usk_{ID} = \perp$.

2) 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.

RevealSecretKey Oracle: Suppose the request is on an identity ID, if $ID \in L_2$, $\mathcal{S}_1$ returns $\perp$, otherwise

1) 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 $\mathcal{A}_1$ and sets $L_3 = L_3 \bigcup \{ID\}$. Otherwise, $\mathcal{S}_1$ makes a **CreateUser** query itself to generate $(upk_{ID} = \nu P,\ usk_{ID} = \nu)$. Then $\mathcal{S}_1$ returns $usk_{ID} = \nu$ to $\mathcal{A}_1$, saves these values in the list L and sets $L_3 = L_3 \bigcup \{ID\}$.

2) 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, sets $L_3 = L_3 \bigcup \{ID\}$ and returns usk_{ID}.

Sign Oracle: Suppose that $\mathcal{A}_1$ queries the oracle with an input $(m,\ ID)$. Without loss of generality, we assume that the list L contains an item $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$, and $upk_{ID} \neq \perp$. (If the list L does not contain such an item, or if $upk_{ID} = \perp$, $\mathcal{S}_1$ runs a **CreateUser** query itself to generate (upk_{ID}, usk_{ID}).)

Then $\mathcal{S}_1$ picks at random two numbers u, $v \in \mathbb{Z}_q^*$ and sets $U = vP_{pub}$, and looks up the list $\mathbf{H}_2$ for (m, U, t_2, H_2) such that the hash value of $H_2(m, U)$ has been defined to $H_2 = t_2$ (If such an item does not exist, $\mathcal{S}_1$ makes a query on oracle H_2). After that $\mathcal{S}_1$ defines the hash value of $H_3(m, ID, upk_{ID}, U)$ as $H_3 = v^{-1}(uP - t_2Q_{ID}) \in \mathbb{G}_1$ ($\mathcal{S}_1$ halts and outputs "failure" if H_3 turns out to have already been defined for (m, ID, upk_{ID}, U)). Then $\mathcal{S}_1$ looks up the list $\mathbf{H}_4$ for (m, ID, upk_{ID}, t_4, H_4) such that the hash value of $H_4(m, ID, upk_{ID})$ has been defined to $H_4 = t_4P \in \mathbb{G}_1$ (If such an item does not exist, $\mathcal{S}_1$ makes a query on oracle H_4). Finally, $\mathcal{S}_1$ sets $V = uP_{pub} + t_4upk_{ID}$ and returns (U, V) to $\mathcal{A}_1$.

Delegation-Proxy Oracle:

1) If $\mathcal{A}_1$ requests to interact with ID_o, ID_o playing the role of original signer. We assume that ID_s, where $ID_s \in \{ID_1, \cdots, ID_n\}$, is the actual proxy signer. $\mathcal{A}_1$ creates a warrant ω, and requests ID_o to sign the warrant ω. $\mathcal{S}_1$ queries ω to its $\mathbf{Sign}(ID_o, \cdot)$ oracle. Upon receiving an answer sig, it forwards (ω, sig) to $\mathcal{A}_1$.

2) If $\mathcal{A}_1$ requests to interact with ID_s, where $ID_s \in \{ID_1, \cdots, ID_n\}$, ID_s playing the role of actual proxy signer, the original signer is ID_o. $\mathcal{A}_1$ outputs a warrant ω and computes the signature $sig = (U, V)$ for warrant ω under the user secret key and user partial key of ID_o. Then sends $sig = (U, V)$ to $\mathcal{S}_1$. After receiving the (ω, sig), $\mathcal{S}_1$ checks the validity of (U, V).

Proxy-Ring-Sign Oracle: $\mathcal{A}_1$ chooses an original signer ID_o, a group of n users whose identities form the set $L_{ID} = \{ID_1, \cdots, ID_n\}$ and their corresponding public keys form the set $L_{upk} = \{upk_1, \cdots, upk_n\}$, and may ask a valid proxy ring signature for a message m on $\{ID_o, L_{ID}, \omega\}$, where ω explicitly denotes that an original signer ID_o designates L_{ID} as a set of proxy signers. To answer such a query, the algorithm $\mathcal{S}_1$ proceeds as follows.

1) Choose at random an index $s \in \{1, \ldots, n\}$.

2) For all $i \in \{1, \cdots, n\}\backslash\{s\}$, choose r_i at random in $\mathbb{Z}_q^*$, pairwise different, and compute $y_i = g^{r_i}$.

3) Compute $h_i = H_6(\omega, m, ID_o, upk_{ID_o}, L_{ID}, L_{upk}, y_i)$ for all $i \in \{1, \cdots, n\}\backslash\{s\}$.

4) Choose $h_s \in \mathbb{Z}_q^*$, V, $U_o \in \mathbb{G}_1$ at random.

5) Compute $y_s = \hat{e}(V - (\sum_{i \neq s} r_i)P, P)\hat{e}(-U_o, (\sum_{i=1}^n h_i) H_3(\omega, ID_o, upk_{ID_o}, U_o))\hat{e}(-P_{pub}, (\sum_{i=1}^n h_i)h_oQ_{ID_o}) \hat{e}(-P_{pub}, h'_o \sum_{i=1}^n h_iQ_{ID_i})\hat{e}(-upk_{ID_o}, (\sum_{i=1}^n h_i)H_4(\omega, ID_o, upk_{ID_o}))\hat{e}(-H_4(\omega, L_{ID}, L_{upk}), \sum_{i=1}^n h_iupk_{ID_i})$, where $h_o = H_2(\omega, U_o)$ and $h'_o = H_5(\omega, U_o)$. If $y_s = 1_{\mathbb{G}_2}$ or $y_s = y_i$ for some $i \neq s$, then go to the previous step.

6) Now $\mathcal{S}_1$ "falsifies" the random oracle H_5, by imposing the relation $H_6(\omega, m, ID_o, upk_{ID_o}, L_{ID}, L_{upk}, y_s) = h_s$. Later, if $\mathcal{A}_1$ asks the random oracle H_6 for this input, then $\mathcal{S}_1$ will answer with h_s. Since h_s is a random value and we are in the random oracle model for H_6, this relation seems consistent to $\mathcal{A}_1$.

7) Return the tuple (ω, m, y_1, …, y_n, V, U_o).

Eventually, $\mathcal{A}_1$ halts. It either concedes failure, in which case so does $\mathcal{S}_1$, or it returns a forgery.

1) $\mathcal{A}_1$ outputs a forgery $sig^* = (U^*, V^*)$ on a message m^*, for an identity ID^* with public key upk_{ID^*}. Now $\mathcal{S}_1$ recovers the triple (ID^*, t_1^*, W^*) from $\mathbf{H}_1$. If $W^* = 0$, then $\mathcal{S}_1$ outputs "failure" and stops. Otherwise, it goes on and finds out an item (m^*, U^*, t_2^*, H_2^*) in the list $\mathbf{H}_2$, an item (m^*, ID^*, upk_{ID^*}, U^*, t_3^*, H_3^*) in the list $\mathbf{H}_3$, and an item (m^*, ID^*, upk_{ID^*}, t_4^*, H_4^*) in the list $\mathbf{H}_4$. Note that list $\mathbf{H}_2$, $\mathbf{H}_3$, and $\mathbf{H}_4$ must contain such entries with overwhelming probability (otherwise, $\mathcal{S}_1$ outputs "failure" and stops). Note that $H_2^* = H_2(m^*, U^*)$ is $t_2^* \in \mathbb{Z}_q^*$, $H_3^* = H_3(m^*, ID^*, upk_{ID^*}, U^*)$ is $t_3^*P \in \mathbb{G}_1$, and $H_4^* = H_4(m^*, ID^*, upk_{ID^*}, t_4^*)$ is $t_4^*P \in \mathbb{G}_1$. If $\mathcal{A}_1$ succeeds in the game, then

$$\hat{e}(V^*, P) = \hat{e}(H_2^* \cdot Q_{ID^*}, X)\hat{e}(U^*, H_3^*)\hat{e}(upk_{ID^*}, H_4^*)$$

with $H_2^* = t_2^*$, $H_3^* = t_3^*P$, $H_4^* = t_4^*P$, and $Q_{ID^*} = t_1^*Y$ for known elements t_1^*, t_2^*, t_3^*, $t_4^* \in \mathbb{Z}_q^*$. Therefore, $\hat{e}(V^*, P) = \hat{e}(t_2^*t_1^*Y, X)\hat{e}(U^*, t_3^*P)\hat{e}(upk_{ID^*}, t_4^*P)$, and thus $(t_2^*t_1^*)^{-1}(V^* - t_3^*U^* - t_4^*upk_{ID^*})$ is the solution to the target CDH instance $(X, Y) \in \mathbb{G}_1 \times \mathbb{G}_1$.

2) $\mathcal{A}_1$ outputs a forgery of the form (m^*, $L_{ID}^* = \{ID_1^*, \cdots, ID_n^*\}$, $L_{upk}^* = \{upk_{ID_1}^*, \cdots, upk_{ID_n}^*\}$, ID_o^*, $upk_{ID_o}^*$, ω^*, $prsig^* = (y_1^*, \cdots, y_n^*, V^*, U_o^*)$), where ID_o^* is the original signer with public key $upk_{ID_o}^*$, L_{ID}^* is the set of proxy signers with public keys L_{upk}^*, and ω^* is the corresponding warrant. It is required that $\mathcal{S}_1$ does not know the private key of original singer and any member in the set of proxy signers, $\{ID_o^*\} \bigcap L_{ID}^* \bigcap ((L_1 \bigcap L_2) \bigcup L_3) = \emptyset$ and the proxy ring signature sig^* must be valid. Now, applying the 'ring forking lemma' [13], if $\mathcal{A}_1$ succeeds in outputting a valid proxy ring signature sig^* with probability $\varepsilon \geq \frac{7C_{q_{H_1}, n}}{2^k}$ in a time t in the above interaction, then within time $2t$ and probability $\geq \frac{\varepsilon^2}{66C_{q_{H_1}, n}}$, $\mathcal{S}_1$ can get two valid proxy ring signatures (m^*, L_{ID}^*, L_{upk}^*, ID_o^*, $upk_{ID_o^*}$, ω^*, $sig^* = (y_1^*, \cdots, y_n^*, V^*, U_o^*)$) and ($m^*$, L_{ID}^*, L_{upk}^*, ID_o^*, $upk_{ID_o^*}$, ω^*, $sig'^* = (y_1^*, \cdots, y_n^*, V'^*, U_o^*)$). From these two valid proxy ring

signatures, $\mathcal{S}_1$ obtains

$$\begin{aligned}\hat{e}(V^*, P) &= y_1^* \cdots y_n^* \hat{e}(\sum_{i=1}^{n} h_i^* h_o^* Q_{ID_o^*}, P_{pub}) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i^* H_3(\omega^*, ID_o^*, upk_{ID_o^*}, U_o*), U_o^*) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i^* upk_{ID_i^*}, H_4(\omega^*, L_{ID}^*, L_{upk}^*)) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i^* H_4(\omega^*, ID_o^*, upk_{ID_o^*}), upk_{ID_o^*}) \\ &\quad \hat{e}(h_o'^* \sum_{i=1}^{n} h_i^* Q_{ID_i^*}, P_{pub})\end{aligned}$$

and

$$\begin{aligned}\hat{e}(V'^*, P) &= y_1^* \cdots y_n^* \hat{e}(\sum_{i=1}^{n} h_i'^* h_o^* Q_{ID_o^*}, P_{pub}) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i'^* H_3(\omega^*, ID_o^*, upk_{ID_o^*}, U_o*), U_o^*) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i'^* upk_{ID_i^*}, H_4(\omega^*, L_{ID}^*, L_{upk}^*)) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i'^* H_4(\omega^*, ID_o^*, upk_{ID_o^*}), upk_{ID_o^*}) \\ &\quad \hat{e}(h_o'^* \sum_{i=1}^{n} h_i'^* Q_{ID_i^*}, P_{pub})\end{aligned}$$

where

$$\begin{aligned}h_o^* &= H_2(\omega^*, U_o^*), \\ h_o'^* &= H_5(\omega^*, U_o^*), \\ h_i^* &= H_6(\omega^*, m^*, ID_o^*, upk_{ID_o^*}, L_{ID}^*, L_{upk}^*, y_i^*), \\ h_i'^* &= H_6'(\omega^*, m^*, ID_o^*, upk_{ID_o^*}, L_{ID}^*, L_{upk}^*, y_i^*),\end{aligned}$$

and for some $s \in \{1, \cdots, n\}$, $h_s^* \neq h_s'^*$, while for $i \in \{1, \cdots, n\} \backslash \{s\}$, $h_i^* = h_i'^*$. From the above two equations we have

$$\begin{aligned}&\hat{e}(V^* - V'^*, P) \\ = &\ \hat{e}((h_s^* - h_s'^*)H_3^*, U_o^*)\hat{e}(h_o^*(h_s^* - h_s'^*)Q_{ID_o^*}, P_{pub}) \\ &\ \hat{e}(h_o'^*(h_s^* - h_s'^*)Q_{ID_s^*}, P_{pub}) \\ &\ \hat{e}((h_s^* - h_s'^*)H_4^*, upk_{ID_o^*}) \\ &\ \hat{e}((h_s^* - h_s'^*)upk_{ID_s^*}, H_4'^*),\end{aligned}$$

where $H_3^* = H_3(\omega^*, ID_o^*, upk_{ID_o^*}, U_o^*)$, $H_4^* = H_4(\omega^*, ID_o^*, upk_{ID_o^*})$, $H_4'^* = H_4(\omega^*, L_{ID}^*, L_{upk}^*)$. At this stage, $\mathcal{S}_1$ may find the item $(ID_o^*, t_{1o}^*, W_o^*)$, $(ID_s^*, t_{1s}^*, W_s^*)$ from $\mathbf{H}_1$, $(\omega^*, ID_o^*, upk_{ID_o^*}, U_o^*, t_3^*, H_3^*)$ from $\mathbf{H}_3$, $(\omega^*, ID_o^*, upk_{ID_o^*}, t_4^*, H_4^*)$, $(\omega^*, L_{ID}^*, L_{upk}^*, t_4'^*, H_4'^*)$ from $\mathbf{H}_4$. If the coins flipped by $\mathcal{S}_1$ for the query to ID_o^* and ID_s^* show 0 then $\mathcal{S}_1$ fails. Otherwise, $(W_o^* = 1, W_s^* = 1)$ then $Q_{ID_o^*} = H_1(ID_o^*) = t_{1o}^* Y$ and $Q_{ID_s^*} = H_1(ID_s^*) = t_{1s}^* Y$. In this case, $\hat{e}(V^* - V'^*, P) = \hat{e}((h_s^* - h_s'^*)t_3^* P, U_o^*)\hat{e}(h_o^*(h_s^* - h_s'^*)t_{1o}^* Y, X)\hat{e}(h_o'^*(h_s^* - h_s'^*)t_{1s}^* Y, X)\hat{e}((h_s^* - h_s'^*)t_4^* P, upk_{ID_o^*})\hat{e}((h_s^* - h_s'^*)upk_{ID_s^*}, t_4'^* P)$. So $\mathcal{S}_1$ can get $abP = ((V^* - V'^*) - t_3^*(h_s^* - h_s'^*)U_o^* - t_4^*(h_s^* - h_s'^*)upk_{ID_o^*} - t_4'^*(h_s^* - h_s'^*)upk_{ID_s^*})(h_s^* - h_s'^*)^{-1}(h_o^* t_{1o}^* + h_o'^* t_{1s}^*)^{-1}$ as 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 [8], the probability $\zeta^{q_{RPar}}(1 - \zeta)$ for $\mathcal{S}_1$ not to fail in key extraction queries or because $\mathcal{A}_1$ produces its forgery of standard signature on a 'bad' identity ID^* is greater than $1 - \frac{1}{e(q_{RPar}+1)}$ when the optimal probability $\zeta_{opt} = \frac{q_{RPar}}{q_{RPar}+1}$ is taken. Furthermore, the probability $\zeta^{q_{RPar}}(1 - \zeta)^{n+1}$ for $\mathcal{S}_1$ not to fail in key extraction queries, or because $\mathcal{A}_1$ produces its forgery of proxy ring signature on a 'bad' identity ID^* is greater than $(\frac{q_{RPar}}{q_{RPar}+n+1})^{q_{RPar}} \cdot (\frac{n+1}{q_{RPar}+n+1})^{n+1}$ when the optimal probability $\zeta_{opt} = \frac{q_{RPar}}{q_{RPar}+n+1}$ is taken. Based on the bound from the ring forking lemma [13], if $\mathcal{A}_1$ succeeds with probability $\varepsilon \geq \frac{7C_{q_{H_1},n}}{2^k}$ to forge the proxy ring signature, then the CDH problem in $\mathbb{G}_1$ can be solved by $\mathcal{S}_1$ with probability $\geq \frac{((\frac{q_{RPar}}{q_{RPar}+n+1})^{q_{RPar}}(\frac{n+1}{q_{RPar}+n+1})^{n+1} \cdot \varepsilon)^2}{66C_{q_{H_1},n}}$. Finally, the probability for $\mathcal{S}_1$ to solve the CHD problem is $(1 - \frac{1}{e(q_{RPar}+1)})\varepsilon + \frac{((\frac{q_{RPar}}{q_{RPar}+n+1})^{q_{RPar}}(\frac{n+1}{q_{RPar}+n+1})^{n+1} \cdot \varepsilon)^2}{66C_{q_{H_1},n}}$. □

Lemma 2. *If a probabilistic polynomial-time forger $\mathcal{A}_2$ has an advantage ε in forging a proxy ring 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 $i = 1, 2, 3, 4, 5, 6$, q_{CreU} queries to the **CreateUser** request oracle, q_{RSec} queries to the **RevealSecretKey** extraction oracle, q_{DP} queries to the **Delegation-Proxy** extraction oracle, q_{Sig} queries to the **Sign** oracle, and q_{PRSig} queries to the **Proxy-Ring-Sign** oracle, then the CDH problem can be solved with probability $\varepsilon' > (1 - \frac{1}{e(q_{CreU}+1)})\varepsilon + \frac{((\frac{q_{CreU}}{q_{CreU}+n+1})^{q_{CreU}}(\frac{n+1}{q_{CreU}+n+1})^{n+1} \cdot \varepsilon)^2}{66C_{q_{H_1},n}}$ with time $t' < 2(t + q_{H_1}T_1 + q_{H_2}T_2 + q_{H_3}T_3 + q_{H_4}T_4 + q_{H_5}T_5 + q_{H_6}T_6 + q_{CreU}T_{CreU} + q_{RSec}T_{RSec} + q_{DP}T_{DP} + q_{Sig}T_{Sig} + q_{PRSig}T_{PRSig})$.*

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 $\kappa \in \mathbb{Z}_q^*$ as the master key, and then initializes $\mathcal{A}_2$ with $P_{pub} = \kappa P$ and also the master key κ. The adversary $\mathcal{A}_2$ then starts making oracle queries such as described in Definition 3. Note that the user's partial key $psk_{ID} = \kappa H_1(ID)$ 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. $\mathcal{S}_2$ also keeps two lists L_1, L_2, the functions of these two lists are the same as mentioned in **Game II** in Section 2.

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 [8], $\mathcal{S}_2$ 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 $t_1P \in \mathbb{G}_1$. If $W = 1$, $\mathcal{S}_2$ returns $t_1X \in \mathbb{G}_1$. 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 $\mathcal{A}_2$.

ReplaceKey Oracle: Suppose $\mathcal{A}_2$ makes the query with an input $(ID,\ upk'_{ID})$, then $\mathcal{S}_2$ sets $L_1 = L_1 \bigcup \{ID\}$.

1) If the list L contains an element $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$, $\mathcal{S}_2$ sets $upk_{ID} = upk'_{ID}$ and $usk_{ID} = \perp$.
2) If the list L does not contain an item $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$, $\mathcal{S}_2$ 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.

RevealSecretKey Oracle: Suppose the request is on an identity ID, if $ID \in L_1$, $\mathcal{S}_2$ returns $\perp$, otherwise

1) If the list L contains $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$, $\mathcal{S}_2$ checks whether $usk_{ID} = \perp$. If $usk_{ID} \neq \perp$, $\mathcal{S}_2$ returns usk_{ID} to $\mathcal{A}_2$ and sets $L_2 = L_2 \bigcup \{ID\}$. Otherwise, $\mathcal{S}_2$ makes a **CreateUser** query itself to generate $(upk_{ID} = t_1P,\ usk_{ID} = t_1)$, while $W = 1$, $\mathcal{S}_2$ aborts. Then $\mathcal{S}_2$ returns $usk_{ID} = t_1$ to $\mathcal{A}_2$, saves these values in the list L and sets $L_2 = L_2 \bigcup \{ID\}$.
2) If the list L does not contain $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$, $\mathcal{S}_2$ makes a **CreateUser** query itself, while $W = 1$, $\mathcal{S}_2$ aborts. And then $\mathcal{S}_2$ adds $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$ to the list L, sets $L_2 = L_2 \bigcup \{ID\}$ and returns usk_{ID}.

Queries on Oracle H_1: On receiving a query $H_1(ID)$. If $(ID,\ Q_{ID})$ exists in $\mathbf{H}_1$, $\mathcal{S}_2$ returns Q_{ID} as answer. Otherwise, $\mathcal{S}_2$ picks a random $Q_{ID} \in \mathbb{G}_1$ which has not been used in the former H_1 queries, then returns Q_{ID} as answer and adds $(ID,\ Q_{ID})$ to $\mathbf{H}_1$.

Queries on Oracle H_2: Suppose $(m,\ U)$ is submitted to oracle $H_2(\cdot)$. $\mathcal{S}_2$ first scans list $\mathbf{H}_2 = \{(m, U, t_2, H_2)\}$ 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 $H_2 = t_2 \in \mathbb{Z}_q^*$ as a hash value of $H_2(m,\ U)$ to $\mathcal{A}_2$ and also stores the values in the list $\mathbf{H}_2$.

Queries on Oracle H_3: Suppose $(m,\ ID,\ upk_{ID},\ U)$ is submitted to oracle $H_3(\cdot)$. $\mathcal{S}_2$ first scans $\mathbf{H}_3 = \{(m,\ ID,\ upk_{ID}, U, t_3, H_3)\}$ 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 $H_3 = t_3P \in \mathbb{G}_1$ as a hash value of $H_3(m,\ ID,\ upk_{ID},\ U)$ to $\mathcal{A}_2$ and also stores the values in the list $\mathbf{H}_3$.

Queries on Oracle H_4: Suppose $(\omega,\ ID,\ upk_{ID})$ is submitted to oracle $H_4(\cdot)$. $\mathcal{S}_2$ first scans $\mathbf{H}_4 = \{(\omega,\ ID,\ upk_{ID},\ t_4,\ H_4)\}$ to check whether H_4 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_2$ picks at random $t_4 \in \mathbb{Z}_q^*$ and returns $H_4 = t_4Y \in \mathbb{G}_1$ as a hash value of $H_4(\omega,\ ID,\ upk_{ID})$ to $\mathcal{A}_2$ and also stores the values in the list $\mathbf{H}_4$.

Queries on Oracle H_5: Suppose $(\omega,\ U)$ is submitted to oracle $H_5(\cdot)$. $\mathcal{S}_2$ first scans list $\mathbf{H}_5 = \{(\omega,\ U,\ t_5,\ H_5)\}$ to check whether H_4 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_2$ picks at random $t_5 \in \mathbb{Z}_q^*$ and returns $H_5 = t_5 \in \mathbb{Z}_q^*$ as a hash value of $H_5(\omega,\ U)$ to $\mathcal{A}_2$ and also stores the values in the list $\mathbf{H}_5$.

Queries on Oracle H_6: Suppose $(\omega,\ m,\ ID,\ upk_{ID},\ L_{ID},\ L_{upk},\ y)$ is submitted to oracle $H_6(\cdot)$. $\mathcal{S}_2$ first scans list $\mathbf{H}_6 = \{(\omega,\ m,\ ID,\ upk_{ID},\ L_{ID},\ L_{upk},\ y,\ t_6,\ H_6)\}$ to check whether H_5 has already been defined for that input. If so, the previously defined value is returned. Otherwise, $\mathcal{S}_2$ picks at random $t_6 \in \mathbb{Z}_q^*$ and returns $H_6 = t_6 \in \mathbb{Z}_q^*$ as a hash value of $H_6(\omega,\ m,\ ID,\ upk_{ID},\ L_{ID},\ L_{upk},\ y)$ to $\mathcal{A}_2$ and also stores the values in the list $\mathbf{H}_6$.

Sign Oracle: Suppose that $\mathcal{A}_2$ queries the oracle with an input $(m,\ ID)$. Without loss of generality, we assume that the list L contains an item $(ID,\ psk_{ID},\ upk_{ID},\ usk_{ID})$, and $upk_{ID} \neq \perp$. (If the list L does not contain such an item, or if $upk_{ID} = \perp$, $\mathcal{S}_2$ runs a **CreateUser** query itself to generate $(upk_{ID},\ usk_{ID})$.)

Then $\mathcal{S}_2$ picks at random two numbers $u,\ v \in \mathbb{Z}_q^*$ and sets $U = vP_{pub}$, and looks up the list $\mathbf{H}_2$ for $(m,\ U,\ t_2,\ H_2)$ such that the hash value of $H_2(m,\ U)$ has been defined to $H_2 = t_2$ (If such an item does not exist, $\mathcal{S}_2$ makes a query on oracle H_2). After that $\mathcal{S}_2$ defines the hash value of $H_3(m,\ ID,\ upk_{ID},\ U)$ as $H_3 = v^{-1}(uP - t_2Q_{ID}) \in \mathbb{G}_1$ ($\mathcal{S}_2$ halts and outputs "failure" if H_3 turns out to have already been defined for $(m,\ ID,\ upk_{ID},\ U)$). Then $\mathcal{S}_2$ looks up the list $\mathbf{H}_4$ for $(m,\ ID,\ upk_{ID},\ t_4,\ H_4)$ such that the hash value of $H_4(m,\ ID,\ upk_{ID})$ has been defined to $H_4 = t_4P \in \mathbb{G}_1$

(If such an item does not exist, $\mathcal{S}_2$ makes a query on oracle H_4). Finally, $\mathcal{S}_2$ sets $V = uP_{pub} + t_4 upk_{ID}$ and returns (U, V) to $\mathcal{A}_2$.

Delegation-Proxy Oracle:

1) If $\mathcal{A}_2$ requests to interact with ID_o, ID_o playing the role of original signer. We assume that ID_s, where $ID_s \in \{ID_1, \cdots, ID_n\}$, is the actual proxy signer. $\mathcal{A}_2$ creates a warrant ω, and requests ID_o to sign the warrant ω. $\mathcal{S}_2$ queries ω to its **Sign**$(ID_o, \cdot)$ oracle. Upon receiving an answer sig, it forwards (ω, sig) to $\mathcal{A}_2$.

2) If $\mathcal{A}_2$ requests to interact with ID_s, where $ID_s \in \{ID_1, \cdots, ID_n\}$, ID_s playing the role of actual proxy signer, the original signer is ID_o. $\mathcal{A}_2$ outputs a warrant ω and computes the signature $sig = (U, V)$ for warrant ω under the user secret key and user partial key of ID_o. Then sends $sig = (U, V)$ to $\mathcal{S}_2$. After receiving the (ω, sig), $\mathcal{S}_2$ checks the validity of (U, V).

Proxy-Ring-Sign Oracle: $\mathcal{A}_2$ chooses an original signer ID_o, a group of n users whose identities form the set $L_{ID} = \{ID_1, \cdots, ID_n\}$ and their corresponding public keys form the set $L_{upk} = \{upk_1, \cdots, upk_n\}$, and may ask a valid proxy ring signature for a message m on $\{ID_o, L_{ID}, \omega\}$, where ω explicitly denotes that an original signer ID_o designates L_{ID} as a set of proxy signers. To answer such a query, the algorithm $\mathcal{S}_2$ proceeds as follows.

1) Choose at random an index $s \in \{1, \ldots, n\}$.

2) For all $i \in \{1, \cdots, n\}\backslash\{s\}$, choose r_i at random in $\mathbb{Z}_q^*$, pairwise different, and compute $y_i = g^{r_i}$.

3) Compute $h_i = H_6(\omega, m, ID_o, upk_{ID_o}, L_{ID}, L_{upk}, y_i)$ for all $i \in \{1, \cdots, n\}\backslash\{s\}$.

4) Choose $h_s \in \mathbb{Z}_q^*$, $V, U_o \in \mathbb{G}_1$ at random.

5) Compute $y_s = \hat{e}(V - (\sum_{i\neq s} r_i)P, P)\hat{e}(-U_o, (\sum_{i=1}^n h_i)H_3(\omega, ID_o, upk_{ID_o}, U_o))\hat{e}(-P_{pub}, (\sum_{i=1}^n h_i)h_o Q_{ID_o})$ $\hat{e}(-P_{pub}, h_o' \sum_{i=1}^n h_i Q_{ID_i})\hat{e}(-upk_{ID_o}, (\sum_{i=1}^n h_i)H_4(\omega, ID_o, upk_{ID_o}))\hat{e}(-H_4(\omega, L_{ID}, L_{upk}), \sum_{i=1}^n h_i upk_{ID_i})$, where $h_o = H_2(\omega, U_o)$ and $h_o' = H_5(\omega, U_o)$. If $y_s = 1_{\mathbb{G}_2}$ or $y_s = y_i$ for some $i \neq s$, then go to the previous step.

6) Now $\mathcal{S}_1$ "falsifies" the random oracle H_5, by imposing the relation $H_6(\omega, m, ID_o, upk_{ID_o}, L_{ID}, L_{upk}, y_s) = h_s$. Later, if $\mathcal{A}_1$ asks the random oracle H_6 for this input, then $\mathcal{S}_1$ will answer with h_s. Since h_s is a random value and we are in the random oracle model for H_6, this relation seems consistent to $\mathcal{A}_1$.

7) Return the tuple $(\omega, m, y_1, \ldots, y_n, V, U_o)$.

Eventually, $\mathcal{A}_2$ halts. It either concedes failure, in which case so does $\mathcal{S}_2$, or it returns a forgery.

1) $\mathcal{A}_2$ outputs a forgery $sig^* = (U^*, V^*)$ on a message m^*, for an identity ID^* with public key upk_{ID^*}. Now $\mathcal{S}_2$ recovers the triple $(ID^*, upk_{ID^*}, usk_{ID^*}, W^*)$ from L. If $W^* = 0$, then $\mathcal{S}_2$ outputs "failure" and stops. Otherwise, it goes on and finds out an item (m^*, U^*, t_2^*, H_2^*) in the list $\mathbf{H}_2$, an item $(m^*, ID^*, upk_{ID^*}, U^*, t_3^*, H_3^*)$ in the list $\mathbf{H}_3$, and an item $(m^*, ID^*, upk_{ID^*}, t_4^*, H_4^*)$ in the list $\mathbf{H}_4$. Note that list $\mathbf{H}_2$, $\mathbf{H}_3$, and $\mathbf{H}_4$ must contain such entries with overwhelming probability (otherwise, $\mathcal{S}_2$ outputs "failure" and stops). Note that $H_2^* = H_2(m^*, U^*)$ is $t_2^* \in \mathbb{Z}_q^*$, $H_3^* = H_3(m^*, ID^*, upk_{ID^*}, U^*)$ is $t_3^*P \in \mathbb{G}_1$, and $H_4^* = H_4(m^*, ID^*, upk_{ID^*}, t_4^*)$ is $t_4^*Y \in \mathbb{G}_1$. If $\mathcal{A}_2$ succeeds in the game, then

$$\hat{e}(V^*, P) = \hat{e}(H_2^* \cdot Q_{ID^*}, P_{pub})\hat{e}(U^*, H_3^*)\hat{e}(upk_{ID^*}, H_4^*)$$

with $H_2^* = t_2^*$, $H_3^* = t_3^*P$, $H_4^* = t_4^*Y$, and $upk_{ID^*} = t_1^*X$ for known elements $t_1^*, t_2^*, t_3^*, t_4^* \in \mathbb{Z}_q^*$. Therefore, $\hat{e}(V^*, P) = \hat{e}(t_2^* Q_{ID^*}, \kappa P)\hat{e}(U^*, t_3^*P)\hat{e}(t_1^*X, t_4^*Y)$, and thus $(t_4^*t_1^*)^{-1}(V^* - t_3^*U^* - t_2^*\kappa Q_{ID^*})$ is the solution to the target CDH instance $(X, Y) \in \mathbb{G}_1 \times \mathbb{G}_1$.

2) $\mathcal{A}_2$ outputs a tuple $(m^*, L_{ID}^* = \{ID_1^*, \cdots, ID_n^*\}, L_{upk}^* = \{upk_{ID_1}^*, \cdots, upk_{ID_n}^*\}, ID_o^*, upk_{ID_o}^*, \omega^*, prsig^* = (y_1^*, \cdots, y_n^*, V^*, U_o^*))$ which means $prsig^*$ is a proxy ring signature on a message m^* on behalf of the original signer specified by identity ID_o^* and public key $upk_{ID_o}^*$, and the set of proxy signers specified by identities in L_{ID}^* and the corresponding public keys in L_{upk}^*. It is required that $\mathcal{S}_2$ does not know the private key of original singer and any member in the set of proxy signers, $\{ID_o^*\} \bigcap L_{ID}^* \bigcap (L_1 \bigcup L_2) = \emptyset$ and the proxy ring signature $prsig^*$ must be valid. Now, applying the 'ring forking lemma' [13], if $\mathcal{A}_2$ succeeds in outputting a valid proxy ring signature sig^* with probability $\varepsilon \geq \frac{7C_{q_{H_1},n}}{2^k}$ in a time t in the above interaction, then within time $2t$ and probability $\geq \frac{\varepsilon^2}{66C_{q_{H_1},n}}$, $\mathcal{S}_2$ can get two valid proxy ring signatures $(m^*, L_{ID}^*, L_{upk}^*, ID_o^*, upk_{ID_o^*}, \omega^*, sig^* = (y_1^*, \cdots, y_n^*, V^*, U_o^*))$ and $(m^*, L_{ID}^*, L_{upk}^*, ID_o^*, upk_{ID_o^*}, \omega^*, sig'^* = (y_1^*, \cdots, y_n^*, V'^*, U_o^*))$. From these two valid proxy ring signatures, $\mathcal{S}_2$ obtains

$$\begin{aligned}\hat{e}(V^*, P) \quad = \quad & y_1^* \cdots y_n^* \hat{e}(h_o'^* \sum_{i=1}^n h_i^* Q_{ID_i^*}, P_{pub}) \\ & \hat{e}(\sum_{i=1}^n h_i^* H_3(\omega^*, ID_o^*, upk_{ID_o^*}, U_o*), U_o^*) \\ & \hat{e}(\sum_{i=1}^n h_i^* H_4(\omega^*, ID_o^*, upk_{ID_o^*}), upk_{ID_o^*}) \\ & \hat{e}(\sum_{i=1}^n h_i^* h_o^* Q_{ID_o^*}, P_{pub}) \\ & \hat{e}(\sum_{i=1}^n h_i^* upk_{ID_i^*}, H_4(\omega^*, L_{ID}^*, L_{upk}^*))\end{aligned}$$

and

$$\begin{aligned}\hat{e}(V'^*, P) &= y_1^* \cdots y_n^* \hat{e}(h_o'^* \sum_{i=1}^{n} h_i'^* Q_{ID_i^*}, P_{pub}) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i'^* H_3(\omega^*, ID_o^*, upk_{ID_o^*}, U_o*), U_o^*) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i'^* H_4(\omega^*, ID_o^*, upk_{ID_o^*}), upk_{ID_o^*}) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i'^* h_o^* Q_{ID_o^*}, P_{pub}) \\ &\quad \hat{e}(\sum_{i=1}^{n} h_i'^* upk_{ID_i^*}, H_4(\omega^*, L_{ID}^*, L_{upk}^*))\end{aligned}$$

where

$$\begin{aligned} h_o^* &= H_2(\omega^*, U_o^*), \\ h_o'^* &= H_5(\omega^*, U_o^*), \\ h_i^* &= H_6(\omega^*, m^*, ID_o^*, upk_{ID_o^*}, L_{ID}^*, L_{upk}^*, y_i^*), \\ h_i'^* &= H_6'(\omega^*, m^*, ID_o^*, upk_{ID_o^*}, L_{ID}^*, L_{upk}^*, y_i^*), \end{aligned}$$

and for some $s \in \{1, \cdots, n\}$, $h_s^* \neq h_s'^*$, while for $i \in \{1, \cdots, n\} \backslash \{s\}$, $h_i^* = h_i'^*$. From the above two equations we have

$$\begin{aligned} & \hat{e}(V^* - V'^*, P) \\ = & \hat{e}((h_s^* - h_s'^*)H_3^*, U_o^*)\hat{e}(h_o^*(h_s^* - h_s'^*)Q_{ID_o^*}, P_{pub}) \\ & \hat{e}(h_o'^*(h_s^* - h_s'^*)Q_{ID_s^*}, P_{pub}) \\ & \hat{e}((h_s^* - h_s'^*)H_4^*, upk_{ID_o^*}) \\ & \hat{e}((h_s^* - h_s'^*)upk_{ID_s^*}, H_4'^*) \end{aligned}$$

where

$$\begin{aligned} H_3^* &= H_3(\omega^*, ID_o^*, upk_{ID_o^*}, U_o^*), \\ H_4^* &= H_4(\omega^*, ID_o^*, upk_{ID_o^*}), \\ H_4'^* &= H_4(\omega^*, L_{ID}^*, L_{upk}^*). \end{aligned}$$

At this stage, $\mathcal{S}_2$ may find the item (ID_o^*, $upk_{ID_o^*}$, $usk_{ID_o^*}$, W_o^*), (ID_s^*, $upk_{ID_s^*}$, $usk_{ID_s^*}$, W_s^*) from L, (ω^*, ID_o^*, $upk_{ID_o^*}$, t_4^*, H_4^*), (ω^*, L_{ID}^*, L_{upk}^*, $t_4'^*$, $H_4'^*$) from $\mathbf{H}_4$. If the coins flipped by $\mathcal{S}_2$ for the query to ID_o^* and ID_s^* show 0 then $\mathcal{S}_2$ fails. Otherwise, ($W_o^* = 1$,$W_s^* = 1$) then $upk_{ID_o^*} = t_{1o}^* X$ and $upk_{ID_s^*} = t_{1s}^* X$. In this case,

$$\begin{aligned} & \hat{e}(V^* - V'^*, P) \\ = & \hat{e}((h_s^* - h_s'^*)t_3^* P, U_o^*)\hat{e}(h_o^*(h_s^* - h_s'^*)Q_{ID_o^*}, \kappa P) \\ & \hat{e}(h_o'^*(h_s^* - h_s'^*)Q_{ID_s^*}, \kappa P) \\ & \hat{e}((h_s^* - h_s'^*)t_4^* Y, usk_{ID_o^*} X) \\ & \hat{e}((h_s^* - h_s'^*)usk_{ID_s^*} X, t_4'^* Y). \end{aligned}$$

So $\mathcal{S}_2$ can get

$$\begin{aligned} abP &= ((V^* - V'^*) - t_3^*(h_s^* - h_s'^*)U_o^* \\ &\quad - \kappa h_o^*(h_s^* - h_s'^*)Q_{ID_o^*} \\ &\quad - \kappa h_o'^*(h_s^* - h_s'^*)Q_{ID_s^*})(h_s^* - h_s'^*)^{-1} \\ &\quad (t_4^* usk_{ID_o^*} + t_4'^* usk_{ID_s^*})^{-1} \end{aligned}$$

as the solution to the target CDH instance $(X, Y) \in \mathbb{G}_1 \times \mathbb{G}_1$.

Now, we evaluate $\mathcal{S}_2$'s probability of failure. By an analysis similar to Lemma 1, the CDH problem in $\mathbb{G}_1$ can be solved by $\mathcal{S}_2$ with probability $(1 - \frac{1}{e(q_{CreU}+1)})\varepsilon + \frac{((\frac{q_{CreU}}{q_{CreU}+n+1})^{q_{CreU}}(\frac{n+1}{q_{CreU}+n+1})^{n+1}\cdot\varepsilon)^2}{66C_{q_{H_1},n}}$. □

4.3 Further Security Analysis

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

1) **Distinguishability:** This is obvious, because there is a warrant ω in a valid proxy ring signature, at the same time, this warrant ω and the public keys of the original signer and the set of proxy signers must occur in the verification equations of proxy ring signatures.

2) **Verifiability:** It derived from correctness of the proposed certificateless proxy ring 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 ring signature, so anyone can determine the identity of the corresponding original signer and the set of proxy signers from the warrant ω.

5) **Signer-ambiguity:** It derived from correctness of Section 4.1.

6) **Prevention of Misuse:** In our proxy ring 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 ring signature are formalized. The models capture the essence of the possible adversaries in the notion of certificateless system and proxy ring signature. A concrete construction of certificateless proxy ring signature scheme from the bilinear maps is presented. The unforgeability of our CL-PRS scheme is proved in the random oracle based on the hardness of Computational Diffie-Hellman problem. We note that CL-PRS schemes may be more efficient than proxy ring 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 ring signatures.

References

[1] S. S. Al-Riyami, K. Paterson, "Certificateless public key cryptography," *AsiaCrypt 2003*, LNCS 2894, pp. 452-473, Springer, 2003.

[2] B. Alomair, K. Sampigethaya, R. Poovendran, "Efficient generic forward-secure signatures and proxy signatures," *EuroPKI 2008*, LNCS 5057, Springer-Verlag, pp.166-181, 2008.

[3] A. K. Awasthil, S. Lal, "A new proxy ring signature scheme," *Proceeding of RMS 2004*, Agra, INDIA, 2004.

[4] A. K. Awasthil, S. Lal, "ID-based ring signature and proxy ring signature schemes from bilinear pairings," *International Journal of Network Security*, vol. 4, no. 2, pp. 187–192, Mar. 2007.

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

[6] D. Boneh, B. Lynn and H. Shacham, "Short signatures from the weil pairing," *Journal of Cryptography*, vol. 17, no. 4, pp. 297-319, 2004.

[7] D. Chaum, E. Hevst, "Group signature," *EUROCRYPT 1991*, LNCS 547, pp. 257-265, Springer-Verlag, 1991.

[8] J. S. Coron, "On the exact security of full domain hash," *Advances in Cryptology (CRYPTO 2000)*, LNCS 1880, pp. 229-235, Springer-Verlag, 2000.

[9] X. Cao, K. G. Paterson, W. Kou, "An attack on a certificateless signature scheme," *Cryptography ePrint Archive*, 2006. (http://eprint. iacr. org/ 2006/ 367)

[10] M. L. Das, A. Saxena, and D. B. Phatak, "Algorithms and approaches of proxy signature: A survey," *International Journal of Network Security*, vol. 9, no. 3, pp. 264-284, 2009.

[11] A. W. Dent, "A survey of certificateless encryption schemes and security models," *Cryptology ePrint Archive*, Report 2006/211, 2006. (http://eprint. iacr. org/ 2006/ 211)

[12] M. C. Gorantla, A. Saxena, "An eficient certificateless signature scheme," *CIS 2005*, LNCS 3802, pp. 110-116, Springer, 2005.

[13] J. Herranz, G. Sáez, "New identity based ring signature," *International Conference on Information and Communications Security (ICICS'2004)*, LNCS 3269, pp. 27-39, Springer-Verlag, Berlin, 2004.

[14] X. Y. Huang, W. Susilo, et al., "On the security of certificateless signature schemes from Asiacrypt 2003," *4th International Conference on Cryptology and Network Security*, LNCS 3810, pp. 13-25, Springer, 2005.

[15] B. C. Hu, D. S. Wong, et al., "Key replacement attack against a generic construction of certificateless signature," *Proceedings of Australasian Conference on Information Security and Privacy (ACISP 2006)*, LNCS 4058, pp. 235-246, Springer, 2006.

[16] X. Huang, Y. Mu, et al., "Certificateless signature revisited," *ACISP 2007*, LNCS 4586, pp. 308-322, Springer, 2007.

[17] H. Kim, J. Baek, B. Lee, and K. Kim, "Secret computation with secrets for mobile agent using one-time proxy signature," *Cryptography and Information Security 2001*, pp. 845-850, 2001.

[18] K. Kim, I. Yie, and S. Lim, "Remark on Shao et al.'s bidirectional proxy re-signature scheme in Indocrypt'07," *International Journal of Network Security*, vol. 9, no. 1, pp. 8-11, 2009.

[19] S. Kim, S. Park, D. Won, "Proxy signatures, revisited," *International Conference on Information and Communications Security (ICICS'97)*, LNCS 1334, pp. 223–232, Springer-Verlag, Berlin, 1997.

[20] W. D. Lin, J. K. Jan, "A security personal learning tools using a proxy blind signature scheme," *Proceedings of International Conference on Chinese Language Computing*, pp.273–277, Illinois, USA, July 2000.

[21] B. Lee, H. Kim, K. Kim, "Secure mobile agent using strong non-designated proxy signature," *ACISP 2001*, LNCS 2119, pp. 474–486, Springer-Verlag, Berlin, 2001.

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

[23] C. Ma and J. Ao, "Group-based proxy re-encryption scheme secure against chosen ciphertext attack," *International Journal of Network Security*, vol. 8, no. 3, pp. 266-270, 2009.

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

[25] M. Mambo, K. Usuda, and E. Okamoto, "Proxy signature: Delegation of the power to sign messages," *IEICE Transactions on Fundamentals*, vol. E79-A, no. 9, pp. 1338–1353, 1996.

[26] T. Malkin, S. Obana, M. Yung, "The hierarchy of key evolving signatures and a characterization of proxy signatures," *Advances in Cryptology (EUROCRYPT 2004)*, LNCS 3027, pp. 306–322, Springer-Verlag, Berlin, 2004.

[27] R. Rivest, A. Shamir, Y. Tauman, "How to leak a secret," *AsiaCrypt'01*, LNCS 2248, pp. 552–565, Springer-Verlag, 2001.

[28] R. Rivest, A. Shamir, Y. Tauman, "How to leak a secret: Theory and applications of ring signatures," *Essays in Theoretical Computer Science: in Memory of Shimon Even*, LNCS 3895, pp. 164-186, Springer-Verlag, 2006.

[29] S. Chow, R. Lui, L. Hui, et al., "Identity based ring signature: Why, how and what next," *The Second European Public Key Infrastructure Workshop (EuroPKI 2005)*, LNCS 3545, pp. 144–161, Springer-Verlag, 2005.

[30] K. A. Shim, "An identity-based proxy signature scheme from pairings," *International Conference on Information and Communications Security (ICICS'06)*, LNCS 4307, pp. 60–71, Springer-Verlag, 2006.

[31] Z. W. Tan, "Improvement on nominative proxy signature schemes," *International Journal of Network Security*, vol. 7, no. 2, pp. 175-180, 2008.

[32] G. K. Verma, "A proxy blind signature scheme over braid groups," *International Journal of Network Security*, vol. 9, no. 3, pp. 214-217, 2009.

[33] H. Xiong, Z. Qin, and F. Li, "An anonymous sealed-bid electronic auction based on ring signature," *International Journal of Network Security*, vol. 8, no. 3, pp. 235–242, Mar. 2009.

[34] D. H. Yum, P. J. Lee, "Generic construction of certificateless signature," *Proceedings of 9th Australasian Conference on Information Security and Privacy (ACISP 2004)*, LNCS 3108, pp. 200-211, Springer, 2004.

[35] K. Zhang, "Threshold proxy signature schemes," *Proceedings of the First International Workshop on Information Security*, pp. 282–290, 1997.

[36] F. G. Zhang, R. Safavi-Naini, C. Y. Lin, "Some new proxy signature schemes from pairings," *Progress on Cryptography: 25 Years of Cryptography in China*, Kluwer International Series in Engineering and Computer Science, vol. 769, pp. 59–66, 2004.

[37] Z. Zhang, D. Wong, "Certificateless public-key signature: Security model and efficient construction," *ACNS 2006*, LNCS 3989, pp. 293-308, Springer, 2006.

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

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

Fagen Li received his B.S. degree from Luoyang Institute of Technology, Luoyang, P.R. China in 2001 and M.S. degree from Hebei University of Technology, Tianjin, P.R. China in 2004. He is now 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.