ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.3, March 2009, pp.692−701
doi: 10.3724/SP.J.1001.2009.00574

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

可证安全的无证书代理签名方案*

陈 虎+, 张福泰, 宋如顺

(南京师范大学 数学与计算机科学学院,江苏 南京 210097)

Certificateless Proxy Signature Scheme with Provable Security

CHEN Hu+, ZHANG Fu-Tai, SONG Ru-Shun

(School of Mathematics and Computer Science, Nanjing Normal University, Nanjing 210097, China)

+ Corresponding author: E-mail: chenhuchh@163.com

Chen H, Zhang FT, Song RS. Certificateless proxy signature scheme with provable security. *Journal of Software*, 2009,20(3):692−701. http://www.jos.org.cn/1000-9825/574.htm

Abstract: This paper studies proxy signatures in the newly proposed certificateless public key setting. The authors present a very strong security model for certificateless proxy signature schemes against both Super Type I Adversary and Super Type II Adversary. And also an efficient construction of certificateless proxy signature scheme using bilinear maps is put forward. The security of this scheme is based on the infeasibility of the Computational Diffie-Hellman problem and is formally proven under the security model of certificateless proxy signature schemes. Due to its security, high efficiency and freedom from certificate management, it may have practical applications in electronic commerce and mobile agent systems, etc.

Key words: certificateless cryptography; bilinear map; computational Diffie-Hellman problem; proxy signature; random Oracle

摘 要: 研究在新提出的无证书公钥密码系统下的代理签名问题,给出了无证书代理签名方案非常强的安全模型.该安全模型下的攻击者是能力最强的超级类型 I 和类型 II 攻击者.同时,利用双线性映射设计了一个高效的无证书代理签名方案.其安全性基于计算 Diffie-Hellman 问题的困难性,并在此安全模型下给出正式的安全证明. 鉴于方案的安全、高效和无证书管理的优点,它可广泛应用于电子商务、移动代理系统等方面.

关键词: 无证书密码系统;双线性映射;计算 Diffie-Hellman 问题;代理签名;随机预言器

中图法分类号: TP309 **文献标识码**: A

1 Introduction

The concept of proxy signature was first introduced by Mambo, Usuda and Okamoto[1] in 1996. In a proxy signature scheme, one user *A*, called original signer, delegates his signing capability to another user *B*, called proxy signer. Upon receiving a proxy signature on some message, a verifier can check its correctness according to a given

* Supported by the National Natural Science Foundation of China under Grant No.60673070 (国家自然科学基金); the Natural Science Foundation of Jiangsu Province of China under Grant No.BK2006217 (江苏省自然科学基金)

Received 2008-05-13; Accepted 2008-10-27

verification procedure, and further be convinced of the original signer's agreement on the signed message. For a secure proxy signature scheme, the following requirements must be satisfied: correctness, strong unforgeability, verifiability, prevention of misuse, strong undeniability and strong identifiability. Proxy signatures have found lots of practical applications in areas such as electronic commerce, global distribution networks, and mobile agent systems, etc. To adapt to different situations, many variants of proxy signature scheme are studied, such as threshold proxy signature[2], proxy multi-signature[3], Designated Verifier Proxy Signature[4], ID-based proxy signature[5] and so on. We notice that almost all proxy signature schemes available in the literature are based on the traditional public key cryptography (TPKC) or the identity-based cryptography (IBC). And it is widely known that the TPKC requires heavy cost on certificate management while IBC suffers from the key escrow problem.

In 2003, Al-Riyami and Paterson[6] introduced an intermediate model between TPKC and IBC, known as certificateless public key cryptography (CL-PKC). Having no certificates that are essential in TPKC, CL-PKC achieves implicit certification without suffering from the inherent key escrow problem in IBC. Therefore, CL-PKC still keeps the advantages enjoyed by TPKC and IBC. Since the appearance of CL-PKC, it has attracted the attention of many researchers and there have been several interesting works on certificateless signature schemes[7−10]. The advantages of certificateless cryptography and the distinguished characteristics of proxy signature schemes make it very interesting to construct secure and efficient certificateless proxy signature (CLPS) schemes. In 2005, Li, *et al*.[9] proposed the first certificateless proxy signature scheme without any formal security proof. Unfortunately, their scheme was found insecure. Recently, Lu, *et al*.[10] and Yap, *et al*.[11] respectively pointed out its security flaws. To the best of our knowledge, no appropriate security model and secure CLPS scheme are available in the literature.

We investigate the appropriate security model and the construction of secure CLPS scheme in this paper. We introduce a security model of certificateless proxy signature schemes. In the security model, the adversaries are Super Type I Adversaries and Super Type II Adversaries[7] with the strongest attack power. At the same time, a provably secure CLPS scheme in the given security model is put forward. Our scheme meets only two pairing operations in the proxy signing and verification processes and enjoys all the security requirements for proxy signatures. We provide formal security proofs for our scheme under the assumption that the Computational Diffie-Hellman problem is intractable.

In the next section, we show some preliminaries and the background knowledge required throughout the paper. In Section 3, we introduce the security model of the CLPS schemes. In Section 4, we present our concrete CLPS scheme. Its security and efficiency analysis are given in Section 5. Section 6 concludes this paper.

2 Preliminaries

2.1 Bilinear maps and computational problem

Let $\boldsymbol{G}_1$ denote an additive group of prime order q and $\boldsymbol{G}_2$ be a multiplicative group of the same order. Let P denote a generator of $\boldsymbol{G}_1$. A map $\boldsymbol{e}$: $\boldsymbol{G}_1\times\boldsymbol{G}_1\rightarrow\boldsymbol{G}_2$ is called a bilinear map, if it has the following properties:

1. Bilinear: $\boldsymbol{e}(aP, bQ)=\boldsymbol{e}(P, Q)^{ab}$ for $P, Q\in\boldsymbol{G}_1$, $a, b\in Z_q^*$.
2. Non-Degeneracy: There exists $P, Q\in\boldsymbol{G}_1$ such that $\boldsymbol{e}(P, Q)\neq 1_{G2}$.
3. Computable: There exists an efficient algorithm to compute $\boldsymbol{e}(P, Q)$ for any $P, Q\in\boldsymbol{G}_1$.

Computational Diffie-Hellman (CDH) Problem.

Given a randomly chosen $P\in\boldsymbol{G}_1$, as well as aP, bP (for unknown $a, b\in Z_q^*$), to compute abP.

2.2 The concept of CLPS

A certificateless proxy signature scheme involves an original signer and a proxy signer. It consists of ten

algorithms: Setup, Partial-Private-key-Extract, Set-Secret-Value, Set-Public-Key, Set-Private-Key, Partial-Proxy-Key-Generate, Partial-Proxy-Key-Verify, Set-Proxy-Key, Proxy-Sign and Proxy-Verify. The formal definitions of the first five algorithms are the same as those in a certificateless signature scheme. Readers can refer to Ref.[6] for details. The others are formally defined as follows:

Partial-Proxy-Key-Generate: An algorithm which takes as input a parameter list *param*, a warrant m_w, an original signer's public/private key and identity to generate a partial proxy key. This algorithm is run by the original signer.

Partial-Proxy-Key-Verify: An algorithm which accepts a parameter list *param*, a warrant m_w, an identity and public key of an original signer, and a partial proxy key to returns *True* if the partial proxy key is correct, or *False* otherwise. This algorithm is run by a proxy signer.

Set-Proxy-Key: An algorithm which accepts a parameter list *param*, a partial proxy key, and a proxy signer's private key to output a proxy key. This algorithm is run by a proxy signer.

Proxy-Sign: An algorithm which accepts a parameter list *param*, a warrant m_w, a message m, an identity and public key of the original signer, an identity and public key of the proxy signer, and a proxy key to generate a proxy signature σ on message m. This algorithm is run by a proxy signer.

Proxy-Verify: An algorithm which accepts a parameter list *param*, a message m, a warrant m_w, a proxy signature σ, an original signer's identity and public key, and a proxy signer's identity and public key to return *True* if the signature is correct , or *False* otherwise.

3 Security Model of CLPS

Similar to the adversaries against certificateless signature scheme defined in Ref.[7], for the security of certificateless proxy signature schemes we consider two types of adversaries, namely Super Type I Adversary, Super Type II Adversary with different capabilities in CLPS schemes.

Super Type I Adversary: A Type I Adversary $\boldsymbol{A}_I$ does not have access to the master-key, but $\boldsymbol{A}_I$ has the ability to replace the public key of any entity with a value of his choice.

Super Type II Adversary: A Type II Adversary $\boldsymbol{A}_{II}$ has access to the master-key but cannot replace the target user's public key.

Informally speaking, a secure **CLPS** scheme should prevent an adversary from producing any valid new message-proxy signature pair without the knowledge of the private proxy key of the proxy signer even if he has already gotten many valid message-proxy signature pairs.

To formally define the security of **CLPS** schemes, we demonstrate a game played between a challenger $\boldsymbol{\Omega}$ and an adversary $\boldsymbol{\Gamma} \in \{\boldsymbol{A}_I, \boldsymbol{A}_{II}\}$.

Setup: Ω runs the **Setup** algorithm of the **CLPS** scheme, takes as input a security parameter ℓ to obtain a master-key and the system parameter lists *param*. $\boldsymbol{\Omega}$ then sends *param* to the adversary $\boldsymbol{\Gamma}$. If $\boldsymbol{\Gamma}$ is a $\boldsymbol{A}_{II}$, $\boldsymbol{\Omega}$ also sends the master-key to $\boldsymbol{\Gamma}$. Note that ℓ is a security parameter throughout the paper.

Attack: The adversary $\boldsymbol{\Gamma}$ can get access to the following oracles (as well as the random oracles if there exists), which are controlled by $\boldsymbol{\Omega}$.

• ***Create-User oracle***: This algorithm takes as input an identity *ID*. If *ID* has already been created, nothing is to be done by this algorithm. Otherwise, it runs the algorithms **Partial-Private-key-Extract, Set-Secret-Value, Set-Public-Key** to obtain the partial private key D_{ID}, secret value x_{ID} and public key P_{ID}. Then it adds (ID, D_{ID}, x_{ID}, P_{ID}) to the list $\boldsymbol{L}$. In this case, *ID* is said to be created. In both cases, P_{ID} is returned.

• ***Partial-Private-Key oracle***: (For $\boldsymbol{A}_I$ only) On input an identity *ID*, which has been created, the oracle browses

the list $\boldsymbol{L}$ and returns the partial private key D_{ID} corresponding to the ID as answer.

• ***Public-Key-Replacement oracle***: Taking as input an identity ID and a new public key P'_{ID}, where ID denotes the created identity, the oracle replaces the public key of the given identity ID with the new one and updates the corresponding information in the list $\boldsymbol{L}$.

• ***Secret-Value oracle***: Accepting a created identity ID, the oracle browses the list $\boldsymbol{L}$ and returns the secret value x_{ID} as answer. Note that, the secret value output by this oracle is the one which is used to generate ID's original public key P_{ID}. In addition, it doesn't output the secret value associated with the replaced public key P'_{ID}.

• ***Partial-Proxy-Key oracle***: On input an original signer's identity ID_A and a warrant m_w, the oracle outputs a partial proxy key Θ_A as answer.

• ***Proxy-Key oracle***: Accepting an original signer's identity ID_A, a proxy signer's identity ID_B, and a warrant m_w, the oracle outputs a proxy key for the proxy signer as answer.

• ***Proxy-Sign oracle***: On input a message m, a warrant m_w, an original signer's identity ID_A, a proxy signer's identity ID_B, the oracle outputs a proxy signature σ as answer.

Forgery: Finally, $\boldsymbol{\Gamma}$ outputs a tuple (m_w^*, ID_A^*, P_A^*, Θ_A^*) or (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, σ^*) as its forgery. We say $\boldsymbol{\Gamma}$ wins the game, if one of the following conditions is satisfied:

case1: Γ outputs a tuple (m_w^*, ID_A^*, P_A^*, Θ_A^*) satisfying:

(1) True ←Verify($param$, m_w^*, ID_A^*, P_A^*, Θ_A^*);

(2) If Γ is $\boldsymbol{A}_I$, ID_A^* has never been made Partial-Private-Key query. If Γ is $\boldsymbol{A}_{II}$, ID_A^* has never been made Secret-Value query;

(3) (m_w^*, ID_A^*, P_A^*) has never been made Partial-Proxy-Key query.

case2: Γ outputs a tuple (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, σ^*) satisfying:

(1) True ←Verify($param$, m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, σ^*);

(2) If Γ is $\boldsymbol{A}_I$, ID_A^* has never been made Partial-Private-Key query. If Γ is $\boldsymbol{A}_{II}$, ID_A^* has never been made Secret-Value query;

(3) (m_w^*, ID_A^*, P_A^*) has never been made Partial-Proxy-Key query;

(4) (m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*) has never been made Proxy-Key query;

(5) (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*) has never been made the Proxy-Sign query.

case3: Γ outputs a tuple (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, σ^*) satisfying:

(1) True ←Verify($param$, m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, σ^*);

(2) If Γ is $\boldsymbol{A}_I$, ID_B^* has never been made Partial-Private-Key query. If Γ is $\boldsymbol{A}_{II}$, ID_B^* has never been made Secret-Value query;

(3) (m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*) has never been made Proxy-Key query;

(4) (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*) has never been made the Proxy-Sign query.

The success probability of an adversary to win the game is denoted by $Succ_{CLPS,\Gamma}^{cma,cida}(\ell) \le \varepsilon$.

Definition. A certificateless proxy signature scheme is existentially unforgeable against adaptively chosen message and chosen identity attack if the success probability of any polynomially bounded adversary in the above game is negligible. In other words, $Succ_{CLPS,\Gamma}^{cma,cida}(\ell) \le \varepsilon$,where ε is negligible.

4 Our CLPS Scheme

We use some ideas of the certificateless signature scheme in Ref.[8]. It consists of the following algorithms:

Setup: ***IG*** is a bilinear map instance generator. This algorithm runs as follows.

1. Run ***IG*** on input ℓ to generate output ($\boldsymbol{G}_1$, $\boldsymbol{G}_2$, $\boldsymbol{e}$), where $\boldsymbol{e}$: $\boldsymbol{G}_1 \times \boldsymbol{G}_1 \rightarrow \boldsymbol{G}_2$ is bilinear map.
2. Choose a random generator $P \in \boldsymbol{G}_1$.
3. Choose a random master-key $s \in_R Z_q^*$ and set $P_0=sP$.
4. Choose cryptographic hash functions $\boldsymbol{H}_1$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$: $\{0,1\}^* \rightarrow \boldsymbol{G}_1^*$.

The system parameters $param$ = ($\boldsymbol{G}_1$, $\boldsymbol{G}_2$, $\boldsymbol{e}$, q, P, P_0, $\boldsymbol{H}_1$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$). The message space is $\mathcal{M}$= {0, 1}*.

Partial-Private-Key-Extract: This algorithm accepts a user's identity $ID_i \in \{0,1\}^*$ and computes $Q_i=\boldsymbol{H}_1(ID_i)$ to output the partial private key $D_i=sQ_i$.

Set-Secret-Value: This algorithm takes as input $param$ and a user's identity ID_i, and selects a random $x_i \in_R Z_q^*$ and outputs x_i as the user's secret value.

Set-Public-Key: This algorithm accepts $param$ and a user's secret value x_i to produce the user's public key $P_i=x_iP$.

Set-Private-Key: This algorithm takes as input $param$, a user's partial private key D_i, secret value x_i, public key P_i, and identity ID_i. The output of the algorithm is the private key $S_i=D_i+x_iT_i$, where $T_i=\boldsymbol{H}_2(ID_i||P_i)$.

Partial-Proxy-Key-Generate: On input $param$, a private key S_A and a warrant m_w, the original signer A with the identity ID_A and the public key P_A computes partial proxy key for the proxy signer B.

1. Randomly pick $r_A \in_R Z_q^*$ and compute $R_A=r_AP$.
2. Compute $U_A = \boldsymbol{H}_3(m_w||ID_A||P_A||R_A)$ and $K_A=S_A+r_AU_A$.
3. Output (m_w, R_A, K_A) to B and take $\Theta_A=(R_A, K_A)$ as the *partial proxy key*.

Note that a warrant m_w specifies the delegation relation, the delegation period, what kind of the messages can be delegated, etc.

Partial-Proxy-Key-Verify: Upon receiving (m_w, R_A, K_A), the proxy signer B checks whether $e(K_A, P) = \boldsymbol{e}(Q_A, P_0)\boldsymbol{e}(T_A, P_A)\ \boldsymbol{e}(U_A, R_A)$ holds. If it does, accept (m_w, R_A, K_A). Otherwise, reject it.

Set-Proxy-Key: If the proxy signer B with the private keyS_B accepts(m_w, R_A, K_A), B sets its *proxy key* as (R_A, K_A, S_B).

Proxy-Sign: To sign a message m, the proxy signer B with identity ID_B, public key P_B and proxy key (R_A, K_A, S_B) performs the following steps:

1. Randomly pick $r_B \in_R Z_q^*$ and compute $R_B= r_BP$.
2. Compute $U_B=\boldsymbol{H}_4(m||m_w||ID_B||P_B||R_B)$ and $V= K_A +S_B + r_B U_B$.
3. Output $\sigma=(R_A, R_B, V)$ as the proxy signature.

Proxy-Verify: To verify (m, m_w, σ) with the original signer's identity ID_A and public key P_A, the proxy signer's identity ID_B and public key P_B, a verifier executes the following steps;

1. Check whether or not the message m conforms to m_w. If not, reject σ; otherwise, continue.
2. Compute $Q_A=\boldsymbol{H}_1(ID_A)$, $Q_B=\boldsymbol{H}_1(ID_B)$, $T_A=\boldsymbol{H}_2(ID_A||P_A)$, $T_B=\boldsymbol{H}_2(ID_B||P_B)$, $U_A = \boldsymbol{H}_3(m_w||ID_A||P_A||R_A)$, and $U_B = \boldsymbol{H}_4(m||m_w||ID_B||P_B||R_B)$.
3. Check whether or not the equation $\boldsymbol{e}(V, P)= \boldsymbol{e}(Q_A+Q_B, P_0)\ \boldsymbol{e}(T_A, P_A)\ \boldsymbol{e}(T_B, P_B)\ \boldsymbol{e}(U_A, R_A)\ \boldsymbol{e}(U_B, R_B)$ holds. If it does, accept σ. Otherwise, reject σ.

5 Security and Efficiency Analysis

5.1 Correctness

The correctness of the proposed scheme can be easily verified.

$$\boldsymbol{e}(V, P) = \boldsymbol{e}(K_A +S_B + r_B U_B, P)$$

$$= \boldsymbol{e}(S_A + r_A U_A, P)\ \boldsymbol{e}(S_B, P)\ \boldsymbol{e}(U_B, R_B)$$
$$= \boldsymbol{e}(D_A + x_A T_A, P)\ \boldsymbol{e}(D_B + x_B T_B, P)\ \boldsymbol{e}(U_A, R_A)\ \boldsymbol{e}(U_B, R_B)$$
$$= \boldsymbol{e}(Q_A + Q_B, P_0)\ \boldsymbol{e}(T_A, P_A)\ \boldsymbol{e}(T_B, P_B)\ \boldsymbol{e}(U_A, R_A)\ \boldsymbol{e}(U_B, R_B).$$

5.2 Strong unforgeability

Assuming that the **CDH** problem is hard, we prove the unforgeability of our **CLPS** scheme.

Theorem 1. In the random oracle model, if $\boldsymbol{A}_I$ is a super type I adaptively chosen message and chosen identity attacker against our **CLPS** scheme with the success probability $Succ^{cma,cida}_{CLPS,A_I}(\ell)$ within a time span t and after asking at most q_{CU} Create-user queries, q_{PPK} Partial-Private-Key queries, q_{PKR} Public-Key-Replacement queries, q_{SV} Secret-Value queries, q_{H2} $\boldsymbol{H}_2$ queries, q_{H3} $\boldsymbol{H}_3$ queries, q_{H4} $\boldsymbol{H}_4$ queries, q_{PProK} Partial-Proxy-Key queries, q_{ProK} Proxy-Key queries and q_{PS} Proxy-Sign queries, then there exists an algorithm $\boldsymbol{\Omega}$ which can use $\boldsymbol{A}_I$ to solve a random instance of the **CDH** problem in G_1 within time $t' \le t + q_{CU}t_{CU} + q_{PPK}\,t_{PPK} + q_{PKR}\,t_{PKR} + q_{SV}\,t_{SV} + q_{H2}t_{H2} + q_{H3}t_{H3} + q_{H4}t_{H4} + q_{PProK}\,t_{PProK} + q_{ProK}\,t_{ProK} + q_{PS}\,t_{PS}$ and with the success probability $Succ^{G_1-G_2}_{CDH,\Omega}(\ell) \ge q_{CU}^{-1}(1-q_{CU}^{-1})^{q_{PPK}+q_{ProK}}\,Succ^{cma,cida}_{CLPS,A_I}(\ell)$, where t_{CU} (resp. t_{PPK}, t_{PKR}, t_{SV}, t_{H2}, t_{H3}, t_{H4}, t_{PProK}, t_{ProK} and t_{PS}) is the time cost of a Create-user (resp. Partial-Private-Key, Public-Key-Replacement, Secret-Value, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$, Partial-Proxy-Key, Proxy-Key and Proxy-Sign) query.

Proof: Given a random instance (P, $P_1=aP$, $P_2=bP$) of the **CDH** problem in $\boldsymbol{G}_1$, we show how $\boldsymbol{\Omega}$ can obtain the value of abP with the help of the $\boldsymbol{A}_I$. In the proof, we regard the hash functions $\boldsymbol{H}_1$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$ as the random oracles. We assume that $\boldsymbol{A}_I$ doesn't repeat any two identical queries.

Setup: In this game, $\boldsymbol{\Omega}$ sets $P_0=P_1=aP$ and the system parameters $param = (\boldsymbol{G}_1, \boldsymbol{G}_2, \boldsymbol{e}, q, P, P_0, \boldsymbol{H}_1, \boldsymbol{H}_2, \boldsymbol{H}_3, \boldsymbol{H}_4)$. $\boldsymbol{\Omega}$ returns $param$ to $\boldsymbol{A}_I$.

Attack: $\boldsymbol{A}_I$ can ask $\boldsymbol{\Omega}$ ***Create-User***, **Partial-Private-Key**, **public-key-Replacement**, **Secret-Value**, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$, **Partial-Proxy-Key**, **Proxy-Key** and **Proxy-Sign** queries. In order to maintain consistency and avoid conflict, $\boldsymbol{\Omega}$ keeps four lists $\boldsymbol{L}$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$ to store the used answers, where $\boldsymbol{L}$-list (resp. $\boldsymbol{H}_2$-list, $\boldsymbol{H}_3$-list, $\boldsymbol{H}_4$-list) includes items of the form (ID_i, Q_i, D_i,α_i, x_i, P_i) (resp. (ID_i, P_i,β_i, T_i), ($m_w{}^i$, $ID_A{}^i$, $P_A{}^i$, $R_A{}^i$, $U_A{}^i$, γ_i), (m_i, $m_w{}^i$, $ID_B{}^i$, $P_B{}^i$, $R_B{}^i$, $U_B{}^i$, h_i)). All of these lists are initially empty.

• ***Create-User oracle***: $\boldsymbol{\Omega}$ first picks a random $f \in \{1,2,\ldots,q_{CU}\}$. Upon receiving $\boldsymbol{A}_I$'s query ***CU*** (ID_i), $\boldsymbol{\Omega}$ picks random x_i, $\alpha_i \in Z_q^*$ such that there is no item (*,*,*,α_i,*,*) in the $\boldsymbol{L}$-list. If $i \ne f$, $\boldsymbol{\Omega}$ sets $Q_i=\alpha_i P$, $D_i=\alpha_i P_0$, $P_i=x_i P$. If $i=f$, $\boldsymbol{\Omega}$ sets $Q_f=x_f P + P_2$, $D_f=\perp$, $P_f= x_f P$. Finally, $\boldsymbol{\Omega}$ adds (ID_i, Q_i, D_i, α_i, x_i, P_i) into the $\boldsymbol{L}$-list and returns P_i to $\boldsymbol{A}_I$ as answer.

• ***Partial-Private-Key queries***: Whenever $\boldsymbol{\Omega}$ receives a query ***PPK***(ID_i), $\boldsymbol{\Omega}$ first checks the $\boldsymbol{L}$-list. If $i \ne f$, $\boldsymbol{\Omega}$ returns D_i as answer. If $i=f$, $\boldsymbol{\Omega}$ aborts.

• ***Public-Key-Replacement queries***: Accepting a query ***PKR***(ID_i,P_i'), $\boldsymbol{\Omega}$ checks the $\boldsymbol{L}$-list and updates the tuple (ID_i, Q_i, D_i, α_i, x_i, P_i) as (ID_i, Q_i, D_i, α_i,$\perp$, P_i').

• ***Secret-Value oracle***: On receiving a query ***SV***(ID_i), $\boldsymbol{\Omega}$ first checks the $\boldsymbol{L}$-list. If $x_i \ne \perp$, $\boldsymbol{\Omega}$ returns x_i as answer. Otherwise, $\boldsymbol{\Omega}$ returns $\perp$ as answer.

• $\boldsymbol{H}_2$ ***Queries***: On receiving $\boldsymbol{A}_I$'s query $\boldsymbol{H}_2(ID_i \| P_i)$, $\boldsymbol{\Omega}$ first picks a random $\beta_i \in Z_q^*$ such that there is no item (*, *,β_i,*) in the $\boldsymbol{H}_2$-list, sets $T_i=\beta_i P$. Then $\boldsymbol{\Omega}$ adds (ID_i, P_i, β_i, T_i) into the $\boldsymbol{H}_2$-list and returns T_i to $\boldsymbol{A}_I$ as answer.

• $\boldsymbol{H}_3$ ***Queries***: On receiving $\boldsymbol{A}_I$'s query $\boldsymbol{H}_3(m_w{}^i \| ID_A{}^i \| P_A{}^i \| R_A{}^i)$, $\boldsymbol{\Omega}$ first picks a random $\gamma_i \in Z_q^*$ such that there is no item (*,*,*,*,*, γ_i) in the $\boldsymbol{H}_3$-list and sets $U_A{}^i=\gamma_i P$. Then $\boldsymbol{\Omega}$ adds ($m_w{}^i$, $ID_A{}^i$, $P_A{}^i$, $R_A{}^i$, $U_A{}^i$, γ_i) into the $\boldsymbol{H}_3$-list and returns $U_A{}^i$ to $\boldsymbol{A}_I$ as answer.

• $\boldsymbol{H}_4$ ***Queries***: On receiving $\boldsymbol{A}_I$'s query $\boldsymbol{H}_4(m_i \| m_w{}^i \| ID_B{}^i \| P_B{}^i \| R_B{}^i)$, $\boldsymbol{\Omega}$ first picks a random $h_i \in Z_q^*$ such that there is no item (*,*,*,*,*,*,h_i) in the $\boldsymbol{H}_4$-list and sets $U_B{}^i=h_i P$. Then $\boldsymbol{\Omega}$ adds (m_i, $m_w{}^i$, $ID_B{}^i$, $P_B{}^i$, $R_B{}^i$,$U_B{}^i$, h_i) into the $\boldsymbol{H}_4$-list and returns $U_B{}^i$ to $\boldsymbol{A}_I$ as answer.

• ***Partial-Proxy-Key oracle***: Upon receiving a query ***PProK***(m_w^i, ID_A^i), **Ω** first checks the ***L***-list to get the current public key of the ID_A^i. Then **Ω** makes $\boldsymbol{H}_2(ID_A^i \| P_A^i)$ to obtain (ID_A^i, P_A^i,β_i, T_A^i) and executes the following steps:

(1) Randomly pick $a_i, b_i \in Z_q^*$.

(2) Set $R_A^i = a_iP_1$, $U_A^i = \boldsymbol{H}_3(m_w^i \| ID_A^i \| P_A^i \| R_A^i) = a_i^{-1}(b_iP - Q_A^i)$, and $K_A^i = b_iP_1 + \beta_i P_A^i$.

If there is a tuple (m_w^i, ID_A^i, P_A^i, R_A^i) in the $\boldsymbol{H}_3$-list, **Ω** updates a_i in order to avoid conflict. **Ω** returns (R_A^i, K_A^i) to $\boldsymbol{A}_I$ as answer and adds (m_w^i, ID_A^i, P_A^i, R_A^i,U_A^i,⊥) into the $\boldsymbol{H}_3$-list.

• ***Proxy-Key oracle***: Upon receiving a query ***ProK***(m_w^i, ID_A^i, ID_B^i), **Ω** checks the ***L***-list to obtain (ID_B^i, Q_B^i, D_B^i,α_B^i, x_B^i, P_B^i). If the public key of ID_B^i has been replaced, **Ω** returns ⊥. Otherwise, **Ω** first performs ***PProK***(m_w^i, ID_A^i), $\boldsymbol{H}_2(ID_B^i \| P_B^i)$to obtain the tuples ($R_A^i$, K_A^i) and (ID_B^i, P_B^i, β_B^i, T_A^i) respectively.

If $ID_B^i = ID_f$, **Ω** aborts. Otherwise, **Ω** sets $S_B^i = D_B^i + x_B^i T_B^i$ and returns (K_A^i, S_B^i) to $\boldsymbol{A}_I$ as answer.

• ***Proxy-Sign oracle***: Upon receiving a query ***PS***(m_i, m_w^i, ID_A^i, ID_B^i), **Ω** first checks the ***L***-list to get the current public keys of the ID_A^i and ID_B^i. Then **Ω** makes $\boldsymbol{H}_2(ID_A^i \| P_A^i)$ and $\boldsymbol{H}_2(ID_B^i \| P_B^i)$ to obtain (ID_A^i, P_A^i,β_A^i, T_A^i) and (ID_B^i, P_B^i, β_B^i, T_B^i) respectively and executes the following steps:

(1) Randomly pick $a_i, b_i, c_i, d_i \in Z_q^*$.

(2) Set $R_A^i = a_iP_1$, $U_A^i = \boldsymbol{H}_3(m_w^i \| ID_A^i \| P_A^i \| R_A^i) = a_i^{-1}(b_iP - Q_A^i)$, and $K_A^i = b_iP_1 + \beta_A^i P_A^i$.

(3) Set $R_B^i = c_iP_1$, $U_B^i = \boldsymbol{H}_4(m_i \| m_w^i \| ID_A^i \| P_A^i \| R_A^i) = c_i^{-1}(d_iP - Q_B^i)$, and $V_i = K_A^i + d_iP_1 + \beta_B^i P_B^i$.

If there is a tuple (m_w^i, ID_A^i, P_A^i, R_A^i) or (m_i, m_w^i, ID_A^i, P_A^i, R_A^i) in the $\boldsymbol{H}_3$-list or $\boldsymbol{H}_4$-list, **Ω** updates a_i or c_i in order to avoid conflict. **Ω** returns (R_A^i, R_B^i, V_i) to $\boldsymbol{A}_I$ as answer and adds (m_w^i, ID_A^i, P_A^i, R_A^i, U_A^i,⊥) and (m_i, m_w^i, ID_B^i, P_B^i, R_B^i, U_B^i,⊥) into the $\boldsymbol{H}_3$-list and $\boldsymbol{H}_4$-list respectively.

Forgery: $\boldsymbol{A}_I$ outputs a tuple (m_w^*,ID_A^*,P_A^*,$\Theta_A^* = (R_A^*, K_A^*)$) or ($m^*$,$m_w^*$,$ID_A^*$,$P_A^*$,$ID_B^*$,$P_B^*$, $\sigma^* = (R_A^*, R_B^*, V^*)$).

(1) If the output is a valid tuple (m_w^*,ID_A^*,P_A^*,$\Theta_A^* = (R_A^*, K_A^*)$) satisfying Case 1 as defined in Section 3, **Ω** first checks ***L***-list, $\boldsymbol{H}_2$-list and $\boldsymbol{H}_3$-list to find (ID_A^*,Q_A^*,D_A^*,α^*,x_A^*,P_A^*), (ID_A^*,P_A^*,β^*,T_A^*), (m_w^*,ID_A^*,P_A^*,R_A^*,U_A^*, γ^*) respectively.

If $ID_A^* \neq ID_f$, **Ω** aborts. Otherwise, **Ω** can compute $abP = K_A^* - \alpha^* P_1 - \beta^* P_A^* - \gamma^* R_A^*$.

(2) If the output is a valid tuple (m^*,m_w^*,ID_A^*,P_A^*,ID_B^*,P_B^*, $\sigma^* = (R_A^*, R_B^*, V^*)$) satisfying Case 2 as defined in Section 3, **Ω** first checks ***L***-list, $\boldsymbol{H}_2$-list, $\boldsymbol{H}_3$-list and $\boldsymbol{H}_4$-list to find (ID_A^*,Q_A^*,D_A^*,α_A^*,x_A^*,P_A^*), (ID_B^*,Q_B^*,D_B^*,α_B^*, x_B^*,P_B^*), (ID_A^*,P_A^*,β_A^*,T_A^*), (ID_B^*,P_B^*,β_B^*,T_B^*), (m_w^*,ID_A^*,P_A^*,R_A^*,U_A^*, γ^*) and (m^*,m_w^*,ID_B^*,P_B^*,R_B^*,U_B^*,h^*) respectively.

If $ID_A^* \neq ID_f$, **Ω** aborts. Otherwise, **Ω** can compute $abP = V^* - \alpha_A^* P_1 - \beta_A^* P_A^* - \gamma^* R_A^* - \alpha_B^* P_1 - \beta_B^* P_B^* - h^* R_B^*$.

(3) If the output is a valid tuple (m^*,m_w^*,ID_A^*,P_A^*,ID_B^*,P_B^*, $\sigma^* = (R_A^*, R_B^*, V^*)$) satisfying Case 3 as defined in Section 3, **Ω** first checks ***L***-list, $\boldsymbol{H}_2$-list, $\boldsymbol{H}_3$-list and $\boldsymbol{H}_4$-list to find (ID_A^*,Q_A^*,D_A^*,α_A^*, x_A^*,P_A^*),(ID_B^*,Q_B^*,D_B^*,α_B^*, x_B^*,P_B^*),(ID_A^*,P_A^*,β_A^*,T_A^*),(ID_B^*,P_B^*,β_B^*,T_B^*),(m_w^*,ID_A^*,P_A^*,R_A^*,U_A^*, γ^*) and (m^*,m_w^*,ID_B^*,P_B^*,R_B^*,U_B^*,h^*) respectively.

If $ID_B^* \neq ID_f$, **Ω** aborts. Otherwise, **Ω** can compute $abP = V^* - \alpha_A^* P_1 - \beta_A^* P_A^* - \gamma^* R_A^* - \alpha_B^* P_1 - \beta_B^* P_B^* - h^* R_B^*$.

Probability of success: We show that **Ω** solves the given instance of **CDH** problem with the probability $Succ_{CDH,\Omega}^{G_1-G_2}(\ell)$. To do so, we analyze the three events that result in **Ω**'s success.

$\mathbf{E}_1$: **Ω** does not abort in all the queries of ***Partial-Private-Key*** and ***Proxy-Key***.

$\mathbf{E}_2$: $\boldsymbol{A}_I$ can forge a valid partial proxy key or proxy signature.

$\mathbf{E}_3$: After Event 2 occurs, one of the following events happens.

(1) $\boldsymbol{A}_I$ outputs a valid tuple (m_w^*, ID_A^*, P_A^*, R_A^*, K_A^*) satisfying $ID_A^* = ID_f$.

(2) $\boldsymbol{A}_I$ outputs a valid tuple (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, R_A^*, R_B^*, V^*) satisfying $ID_A^* = ID_f$.

(3) $\boldsymbol{A}_I$ outputs a valid tuple (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, R_A^*, R_B^*, V^*) satisfying $ID_B^*=ID_f$.

The probability that $\boldsymbol{\Omega}$ solves the given **CDH** problem is $Pr(\mathbf{E_1}\cap\mathbf{E_2}\cap\mathbf{E_3}) = Pr(\mathbf{E_1})Pr(\mathbf{E_2}|\mathbf{E_1})\ Pr(\mathbf{E_3}|\ \mathbf{E_2}\cap\mathbf{E_1})$. We have $Pr(\mathbf{E_1})\geq(1-q_{CU}^{-1})^{q_{PPK}+q_{ProK}}$, $Pr(\mathbf{E_2}|\mathbf{E_1})\geq Succ_{CLPS,A_I}^{cma,cida}(\ell)$ and $Pr(\mathbf{E_3}|\mathbf{E_2}\cap\mathbf{E_1})\geq q_{CU}^{-1}$.

Hence, $Succ_{CDH,\Omega}^{G_1-G_2}(\ell)=Pr(\mathbf{E_1}\cap\mathbf{E_2}\cap\mathbf{E_3})\geq q_{CU}^{-1}(1-q_{CU}^{-1})^{q_{PPK}+q_{ProK}}\ Succ_{CLPS,A_I}^{cma,cida}(\ell)$.

In summary, if $\boldsymbol{A}_I$ succeeds within a time span t for a security parameter ℓ, then the **CDH** problem in $\boldsymbol{G}_1$ can be solved by $\boldsymbol{\Omega}$ within a time span $t' \leq t + q_{CU}t_{CU} + q_{PPK}\ t_{PPK} + q_{PKR}\ t_{PKR} + q_{SV}\ t_{SV} + q_{H2}t_{H2} + q_{H3}t_{H3} + q_{H4}t_{H4} + q_{PProK}\ t_{PProK} + q_{ProK}\ t_{ProK} + q_{PS}\ t_{PS}$ and with the success probability $Succ_{CDH,\Omega}^{G_1-G_2}(\ell)\geq q_{CU}^{-1}(1-q_{CU}^{-1})^{q_{PPK}+q_{ProK}}\ Succ_{CLPS,A_I}^{cma,cida}(\ell)$.

Theorem 2. In the random oracle model, if $\boldsymbol{A}_{II}$ is a super type II adaptively chosen message and chosen identity attacker against our **CLPS** scheme with the success probability $Succ_{CLPS,A_{II}}^{cma,cida}(\ell)$ within a time span t and after asking at most q_{CU} Create-user queries, q_{PKR} Public-Key- Replacement queries, q_{SV} Secret-Value queries, q_{H2} $\boldsymbol{H}_2$ queries, q_{H3} $\boldsymbol{H}_3$ queries, q_{H4} $\boldsymbol{H}_4$ queries, q_{PProK} Partial-Proxy-Key queries, q_{ProK} Proxy-Key queries and q_{PS} Proxy-Sign queries, then there exists an algorithm $\boldsymbol{\Omega}$ which can use $\boldsymbol{A}_{II}$ to solve a random instance of the **CDH** problem in G_1 within time $t' \leq t + q_{CU}t_{CU} + q_{PKR}\ t_{PKR} + q_{SV}\ t_{SV} + q_{H2}t_{H2} + q_{H3}t_{H3} + q_{H4}t_{H4} + q_{PProK}\ t_{PProK} + q_{ProK}\ t_{ProK} + q_{PS}\ t_{PS}$ and with the success probability $Succ_{CDH,\Omega}^{G_1-G_2}(\ell)\geq q_{CU}^{-1}(1-q_{CU}^{-1})^{q_{SV}+q_{ProK}}\ Succ_{CLPS,A_{II}}^{cma,cida}(\ell)$, where t_{CU} (resp. t_{PKR}, t_{SV}, t_{H2}, t_{H3}, t_{H4}, t_{PProK}, t_{ProK} and t_{PS}) is the time cost of a Create-user (resp. Public-Key-Replacement, Secret-Value, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$, Partial-Proxy-Key, Proxy-Key and Proxy- Sign) query.

Proof: Given a random instance (P, $P_1=aP$, $P_2=bP$) of the **CDH** problem in $\boldsymbol{G}_1$, we show how $\boldsymbol{\Omega}$ can obtain the value of abP with the help of the $\boldsymbol{A}_{II}$. In the proof, we regard the hash functions $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$ as the random oracles. We assume that $\boldsymbol{A}_{II}$ doesn't repeat any two identical queries.

Setup: In the game, $\boldsymbol{\Omega}$ selects a random $s\in Z_q^*$, set $P_0= sP$ and the system parameters $param$ = ($\boldsymbol{G}_1$, $\boldsymbol{G}_2$, $\boldsymbol{e}$, q, P, P_0, $\boldsymbol{H}_1$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$). $\boldsymbol{\Omega}$ returns $param$ and the master-key s to $\boldsymbol{A}_{II}$.

Attack: $\boldsymbol{A}_{II}$ can ask $\boldsymbol{\Omega}$ **Create-User**, **Public-Key-Replacement**, **Secret-Value**, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$, **Partial-Proxy-Key**, **Proxy-Key** and **Proxy-Sign** queries. In order to maintain consistency and avoid conflict, $\boldsymbol{\Omega}$ keeps four lists $\boldsymbol{L}$, $\boldsymbol{H}_2$, $\boldsymbol{H}_3$, $\boldsymbol{H}_4$ to store the used answers, where $\boldsymbol{L}$-list includes items of the form (ID_i, D_i, x_i, P_i), $\boldsymbol{H}_2$-list includes items of the form (ID_i, P_i, β_i, T_i), $\boldsymbol{H}_3$-list includes items of the form (m_w^i, ID_A^i, P_A^i, R_A^i, U_A^i, γ_i), $\boldsymbol{H}_4$-list includes items of the form (m_i, m_w^i, ID_B^i, P_B^i, R_B^i, U_B^i, h_i). All of these lists are initially empty.

- ***Create-User oracle***: $\boldsymbol{\Omega}$ first picks a random $f\in\{1,2,\ldots,q_{CU}\}$. Upon receiving $\boldsymbol{A}_{II}$'s query $\boldsymbol{CU}$ (ID_i), $\boldsymbol{\Omega}$ picks random $x_i\in Z_q^*$. If $i\neq f$, $\boldsymbol{\Omega}$ sets $D_i=s\boldsymbol{H}_1(ID_i)$, $P_i= x_iP$. If $i=f$, $\boldsymbol{\Omega}$ sets $D_f= s\boldsymbol{H}_1(ID_f)$, $P_f=x_fP+P_1$. Finally, $\boldsymbol{\Omega}$ adds (ID_i, D_i, x_i, P_i) into the $\boldsymbol{L}$-list and returns P_i to $\boldsymbol{A}_{II}$ as answer.

- ***Public-Key-Replacement queries***: On receiving a query $\boldsymbol{PKR}(ID_i, P_i')$, $\boldsymbol{\Omega}$ checks the $\boldsymbol{L}$-list and updates the tuple (ID_i, D_i, x_i, P_i) as (ID_i, D_i, $\perp$, P_i').

- ***Secret-Value oracle***: On receiving a query $\boldsymbol{SV}(ID_i)$, $\boldsymbol{\Omega}$ first checks the $\boldsymbol{L}$-list. If $i=f$, $\boldsymbol{\Omega}$ aborts. Otherwise, if $x_i\neq\perp$, $\boldsymbol{\Omega}$ returns x_i as answer; if $x_i=\perp$, $\boldsymbol{\Omega}$ returns $\perp$ as answer.

- $\boldsymbol{H}_2$ ***Queries***: On receiving $\boldsymbol{A}_{II}$'s query $\boldsymbol{H}_2(ID_i||P_i)$, $\boldsymbol{\Omega}$ first picks a random $\beta_i\in Z_q^*$ such that there is no item (*,*,β_i, *) in the $\boldsymbol{H}_2$-list. If $i=f$, $\boldsymbol{\Omega}$ sets $T_f=\beta_fP+P_2$. Otherwise, $\boldsymbol{\Omega}$ sets $T=\beta_iP$. Then $\boldsymbol{\Omega}$ adds (ID_i, P_i,β_i, T_i) into the $\boldsymbol{H}_2$-list and returns T_i to $\boldsymbol{A}_{II}$ as answer.

- $\boldsymbol{H}_3$ ***Queries***: On receiving $\boldsymbol{A}_{II}$'s query $\boldsymbol{H}_3(m_w^i||ID_A^i||P_A^i||R_A^i)$, $\boldsymbol{\Omega}$ first picks a random $\gamma_i\in Z_q^*$ such that there is no item (*,*,*,*,*, γ_i) in the $\boldsymbol{H}_3$-list and sets $U_A^i=\gamma_iP$. Then $\boldsymbol{\Omega}$ adds (m_w^i, ID_A^i, P_A^i, R_A^i, U_A^i, γ_i) into the $\boldsymbol{H}_3$-list and returns U_A^i to $\boldsymbol{A}_{II}$ as answer.

- $\boldsymbol{H}_4$ ***Queries***: On receiving $\boldsymbol{A}_{II}$'s query $\boldsymbol{H}_4(m_i||m_w^i||ID_B^i||P_B^i||R_B^i)$, $\boldsymbol{\Omega}$ first picks a random $h_i\in Z_q^*$ such that there is no item (*,*,*,*,*,*,h_i) in the $\boldsymbol{H}_4$-list and sets $U_B^i=h_iP$. Then $\boldsymbol{\Omega}$ adds (m_i, m_w^i, ID_B^i, P_B^i, R_B^i,U_B^i, h_i) into the $\boldsymbol{H}_4$-list

and returns U_B^i to $\boldsymbol{A}_{II}$ as answer.

• ***Partial-Proxy-Key oracle***: Upon receiving a query ***PProK***(m_w^i, ID_A^i), $\boldsymbol{\Omega}$ first checks the $\boldsymbol{L}$-list to get the current public key of the ID_A^i. Then $\boldsymbol{\Omega}$ makes $\boldsymbol{H}_2(ID_A^i \| P_A^i)$ to obtain (ID_A^i, P_A^i, β_i, T_A^i) and executes the following steps:

(1) Randomly pick a_i, $b_i \in Z_q^*$.

(2) Set $R_A^i = b_i P_A^i$, $U_A^i = \boldsymbol{H}_3(m_w^i \| ID_A^i \| P_A^i \| R_A^i) = b_i^{-1}(a_i P - T_A^i)$, and $K_A^i = a_i P_A^i + D_A^i$.

If there is the tuple (m_w^i, ID_A^i, P_A^i, R_A^i) in the $\boldsymbol{H}_3$-list, $\boldsymbol{\Omega}$ updates b_i in order to avoid this conflict. $\boldsymbol{\Omega}$ returns (R_A^i, K_A^i) to $\boldsymbol{A}_{II}$ as answer and adds (m_w^i, ID_A^i, P_A^i, R_A^i, U_A^i, $\perp$) into the $\boldsymbol{H}_3$-list.

• ***Proxy-Key oracle***: Upon receiving a query ***ProK***(m_w^i, ID_A^i, ID_B^i), $\boldsymbol{\Omega}$ checks the $\boldsymbol{L}$-list to obtain (ID_B^i, D_B^i, x_B^i, P_B^i). If the public key of ID_B^i has been replaced, $\boldsymbol{\Omega}$ returns $\perp$. Otherwise, $\boldsymbol{\Omega}$ first performs ***PProK***(m_w^i,ID_A^i), $\boldsymbol{H}_2(ID_B^i \| P_B^i)$ to obtain the tuples (R_A^i,K_A^i) and (ID_B^i,P_B^i, β_B^i,T_B^i) respectively.

If $ID_B^i = ID_f$, $\boldsymbol{\Omega}$ aborts. Otherwise, $\boldsymbol{\Omega}$ sets $S_B^i = D_B^i + x_B^i T_B^i$ and returns (K_A^i, S_B^i) to $\boldsymbol{A}_{II}$ as answer

• ***Proxy-Sign oracle***: Upon receiving a query ***PS***(m_i, m_w^i, ID_A^i, ID_B^i), $\boldsymbol{\Omega}$ first checks the $\boldsymbol{L}$-list to get the current public keys of the ID_A^i and ID_B^i. Then $\boldsymbol{\Omega}$ makes $\boldsymbol{H}_2(ID_A^i \| P_A^i)$ and $\boldsymbol{H}_2(ID_B^i \| P_B^i)$ to obtain (ID_A^i, P_A^i, β_A^i, T_A^i) and (ID_B^i, P_B^i, β_B^i, T_B^i) respectively and executes the following steps:

(1) Randomly pick a_i, b_i, c_i, $d_i \in Z_q^*$.

(2) Set $R_A^i = b_i P_A^i$, $U_A^i = \boldsymbol{H}_3(m_w^i \| ID_A^i \| P_A^i \| R_A^i) = b_i^{-1}(a_i P - T_A^i)$, and $K_A^i = a_i P_A^i + D_A^i$.

(3) Set $R_B^i = d_i P_B^i$, $U_B^i = \boldsymbol{H}_4(m_i \| m_w^i \| ID_A^i \| P_A^i \| R_A^i) = d_i^{-1}(c_i P - T_B^i)$, and $V_i = K_A^i + c_i P_B^i + D_B^i$.

If there is a tuple (m_w^i, ID_A^i, P_A^i, R_A^i) or (m_i, m_w^i, ID_A^i, P_A^i, R_A^i) in the $\boldsymbol{H}_3$-list or $\boldsymbol{H}_4$-list, $\boldsymbol{\Omega}$ updates a_i or c_i in order to avoid conflict. $\boldsymbol{\Omega}$ returns (R_A^i, R_B^i, V_i) to $\boldsymbol{A}_{II}$ as answer and adds (m_w^i, ID_A^i, P_A^i, R_A^i,U_A^i,$\perp$) and (m_i, m_w^i, ID_B^i, P_B^i, R_B^i, U_B^i,$\perp$) into the $\boldsymbol{H}_3$-list and $\boldsymbol{H}_4$-list respectively.

Forgery: $\boldsymbol{A}_{II}$ outputs a tuple (m_w^*,ID_A^*,P_A^*,$\Theta_A^* = (R_A^*, K_A^*)$) or ($m^*$,$m_w^*$,$ID_A^*$,$P_A^*$,$ID_B^*$,$P_B^*$, $\sigma^* = (R_A^*, R_B^*, V^*)$).

(1) If the output is a valid tuple (m_w^*,ID_A^*,P_A^*,$\Theta_A^* = (R_A^*, K_A^*)$) satisfying Case 1 as defined in Section 3, $\boldsymbol{\Omega}$ first checks $\boldsymbol{L}$-list, $\boldsymbol{H}_2$-list and $\boldsymbol{H}_3$-list to find (ID_A^*,D_A^*,x_A^*,P_A^*),(ID_A^*,P_A^*, β^*,T_A^*),(m_w^*,ID_A^*,P_A^*,R_A^*,U_A^*, γ^*) respectively.

If $ID_A^* \neq ID_f$, $\boldsymbol{\Omega}$ aborts. Otherwise, $\boldsymbol{\Omega}$ can compute $abP = K_A^* - D_f - x_f P_2 - \beta_f P_f - \gamma^* R_A^*$.

(2) If the output is a valid tuple (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, $\sigma^* = (R_A^*, R_B^*, V^*)$) satisfying Case 2 as defined in Section 3, $\boldsymbol{\Omega}$ first checks $\boldsymbol{L}$-list, $\boldsymbol{H}_2$-list, $\boldsymbol{H}_3$-list and $\boldsymbol{H}_4$-list to find (ID_A^*, D_A^*, x_A^*, P_A^*), (ID_B^*, D_B^*, x_B^*, P_B^*), (ID_A^*, P_A^*,β_A^*, T_A^*), (ID_B^*, P_B^*,β_B^*, T_B^*), (m_w^*, ID_A^*, P_A^*, R_A^*,U_A^*, γ^*) and (m^*, m_w^*, ID_B^*, P_B^*, R_B^*,U_B^*, h^*) respectively.

If $ID_A^* \neq ID_f$, $\boldsymbol{\Omega}$ aborts. Otherwise, $\boldsymbol{\Omega}$ can compute $abP = V^* - D_f - x_f P_2 - \beta_f P_f - \gamma^* R_A^* - D_B^* - \beta_B^* P_B^* - h^* R_B^*$.

(3) If the output is a valid tuple (m^*, m_w^*, ID_A^*, P_A^*, ID_B^*, P_B^*, $\sigma^* = (R_A^*, R_B^*, V^*)$) satisfying Case 3 as defined in Section 3, $\boldsymbol{\Omega}$ first checks $\boldsymbol{L}$-list, $\boldsymbol{H}_2$-list, $\boldsymbol{H}_3$-list and $\boldsymbol{H}_4$-list to find (ID_A^*, D_A^*, x_A^*, P_A^*),(ID_B^*, D_B^*, x_B^*, P_B^*),(ID_A^*, P_A^*,β_A^*, T_A^*), (ID_B^*, P_B^*,β_B^*, T_B^*), (m_w^*, ID_A^*, P_A^*, R_A^*,U_A^*, γ^*) and (m^*, m_w^*, ID_B^*, P_B^*, R_B^*,U_B^*, h^*) respectively.

If $ID_B^* \neq ID_f$, $\boldsymbol{\Omega}$ aborts. Otherwise, $\boldsymbol{\Omega}$ can compute $abP = V^* - D_f - x_f P_2 - \beta_f P_f - h^* R_B^* - D_A^* - \beta_A^* P_A^* - \gamma^* R_A^*$.

Probability of success: With the similar method as in Theorem 1, we have the following conclusion: If $\boldsymbol{A}_{II}$ succeeds within a time span t for a security parameter ℓ, then the **CDH** problem in $\boldsymbol{G}_1$ can be solved by $\boldsymbol{\Omega}$ within a time span $t' \le t + q_{CU} t_{CU} + q_{PKR} t_{PKR} + q_{SV} t_{SV} + q_{H2} t_{H2} + q_{H3} t_{H3} + q_{H4} t_{H4} + q_{PProK} t_{PProK} + q_{ProK} t_{ProK} + q_{PS} t_{PS}$ and with the success probability $Succ_{CDH,\Omega}^{G_1-G_2}(\ell) \ge q_{CU}^{-1}(1-q_{CU}^{-1})^{q_{SV}+q_{ProK}} Succ_{CLPS,A_{II}}^{cma,cida}(\ell)$.

According to the above theorems, we can conclude that an original signer and other third parties who are not designated as proxy signers cannot create a valid proxy signature. Thus our scheme enjoys strong unforgeability. In addition, our scheme also enjoys the security requirements of proxy signatures such as verifiability, prevention of misuse, strong undeniability, strong identifiability, etc. Due to page limitation, we will not describe them here.

5.3 Efficiency

In comparing our scheme with Lu, *et al*.'s scheme[10] in detail, we only consider the costly operations including the bilinear pairing operation (BP), scalar multiplication in $\boldsymbol{G}_1$ (SM), exponentiation in $\boldsymbol{G}_2$ (E) and hash operation (H). In both schemes, $\boldsymbol{e}(P, P)$, $\boldsymbol{e}(Y_{ID}, Q_{ID})$, $\boldsymbol{e}(Y_{ID}, (\boldsymbol{H}_2(m_w,U)Q_{ID}+U))$, $\boldsymbol{e}(Q_{ID}, P_0)$, $\boldsymbol{e}(T_{ID}, P_{ID})$, $\boldsymbol{e}(U_{ID}, R_{ID})$, $\boldsymbol{H}_1(ID)$, $\boldsymbol{H}_2(ID||P_{ID})$ and $\boldsymbol{H}_3(m_w||ID\ ||P_{ID}||R_{ID})$ can be pre-computed in the proxy signature verification phase, so they are not counted into the operation cost in the table below. We use |*| to denote the bit length of *. The table shows that our scheme has much less operations cost than the scheme in Ref.[10]. Therefore, our scheme is more efficient.

Table 1 Efficiency comparison

Schemes	PPro-K-Gen	PPro-K-Ver	Pro-Sign	Pro-Ver	Provable security
Scheme in Ref.[10]	2SM+1H	4BP+1SM+1H	2SM+1E+1H	5BP+3E+1H	No formal proof provided
Our scheme	1SM+1H	2BP+1H	2SM+1H	2BP+2H	Yes

6 Conclusion

In this paper, we have presented an appropriate security model as well as a concrete construction of certificateless proxy signature scheme. Our security model takes into account the strongest adversaries in certificateless public key settings. Without pairing operation in proxy signature phase and with two pairing operations in proxy signature verification phase, our scheme is efficient. It enjoys all the security requirements of proxy signatures. The security of our scheme is proved in the random model with the intractability of the Computational Diffie-Hellman problem. Due to its efficiency and certificateless, it can be widely used in areas such as electronic commerce, mobile agent systems, etc.

References:

[1] Mambo M, Usuda K, Okamoto E. Proxy signature: Delegation of the power to sign messages. IEICE Trans. on Fundamentals, 1996, E79-A(9):1338−1353.
[2] Zhang K. Threshold proxy signature schemes. In: Proc. of the 1997 Information Security Workshop. Japan, 1997. 191−197.
[3] Yi LJ, Bai GQ, Xiao GZ. Proxy multi-signature scheme: A new type of proxy signature scheme. Electronics Letters, 2000,36(6): 527−528.
[4] Huang XY, Mu Y, Susilo W, Zhang FT. Short designated verifier proxy signature from pairings. In: Proc. of the SecUbiq 2005. LNCS3823, Nagasaki, Springer-Verlag, 2005. 835−844.
[5] Zhang FG, Kim K. Efficient ID-based blind signature and proxy signature from bilinear pairings. In: Safavi-Naini R, Seberry J, eds. Proc. of the ACISP 2003. LNCS 2727, Springer-Verlag, 2003. 312−323.
[6] Al-Riyami S, Paterson K. Certificateless public key cryptography. In: Proc. of the Asiacrypt 2003. LNCS 2894, Springer-Verlag, 2003. 452−473.
[7] Huang XY, Mu Y, Susilo W, Wong DS, Wu W. Certificateless signature revisited. In: Proc. of the Acisp 2007. LNCS 4586, Springer-Verlag, 2007. 308−322.
[8] Zhang ZF, Wong DS, Xu J, Feng DG. Certificateless public-key signature: Security model and efficient construction. In: Zhou J, Yung M, Bao F, eds. Proc. of the ACNS 2006. LNCS 3989, Springer-Verlag, 2006. 293−308.
[9] Li X, Chen K, Sun L. Certificateless signature and proxy signature schemes from bilinear pairings. Lithuanian Mathematical Journal, 2005,45(1):76−83.
[10] Lu R, He D, Wang CJ. Cryptanalysis and improvement of a certificateless proxy signature scheme from bilinear pairings. In: Proc. of the 8th ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing. 2007. 285−290. doi: 10.1109/SNPD
[11] Yap WS, Heng SH, Goi BK. Cryptanalysis of some proxy signature schemes without certificates. In: Sauveron D, *et al*., eds. In: Proc. of the WISTP 2007. LNCS 4462, Springer-Verlag, 2007. 115−126.

CHEN Hu was born in 1975. He is a M.D. candidate at School of Mathematics and Computer Science Nanjing Normal University. His current research areas are information security and cryptography.

ZHANG Fu-Tai was born in 1965. He is a professor and doctoral supervisor at School of Mathematics and Computer Science Nanjing Normal University. His research area is cryptography.

SONG Ru-Shun was born in 1953. He is a professor at School of Mathematics and Computer Science Nanjing Normal University. His research area is network security.