Provably Secure Certificateless Proxy Signature Scheme in the Standard Model

Lunzhi Deng[1*], Zhenyu Hu[2], Yu Ruan[1], Tao Wang[1]

[1] School of Mathematical Sciences, Guizhou Normal University, China
[2] School of Big Data and Computer Science, Guizhou Normal University, China
denglunzhi@163.com, 592631964 @qq.com, 2455392912@qq.com, 1321682980@qq.com

Abstract

Proxy signature frees the original signer from the heavy signature work. Many certificateless proxy signature (CLPS) schemes have been proposed in the last ten years. The security proofs of most known schemes are given in the random oracle model (ROM). There are only two CLPS schemes with provably security in the standard model (SM). However, in which the size of the system parameter increase linearly with the size of the user's identity information. That increase the storage burden of the key generation center. In this paper, a new CLPS scheme is constructed and the security proofs are showed in SM. The size of system parameters and the master key are constant in the scheme. Requiring only three pairing operations, the new scheme is more efficient and suitable for mobile computing.

Keywords: Certificateless cryptography, Mobile computing, Proxy signature, Pairing, Standard model

1. Introduction

With the development of mobile communication technology, mobile services have penetrated into every aspect of people's lives. In 2017, more than 5 billion people were associated with mobile services, and by 2025 independent mobile users will reach 5.9 billion, equivalent to 71% of the global population. The number of mobile Internet users will increase by 1.75 billion new users by then, reaching a milestone of 5 billion users in 2025. Global mobile data traffic grew strongly, with a compound annual growth rate of 73.46% in 2010-2017, with smartphone mobile data traffic growing at a compound annual growth rate of 121.69% from 2010 to 2017.

People are enjoying the convenience of mobile communications while also facing the risk of personal privacy leaks. Due to size constraints, the computing power of personal mobile communication devices is limited. Some previous cryptographic schemes require more computing costs, so they are not suitable for personal mobile communication devices. Therefore, it is meaningful to design secure and efficient cryptographic schemes for mobile computing.

In 1996, Mambo et al. [1] put forward the notion of proxy signatures, which means that when the authorizer (original signer) cannot exercise the signature right for some reason, he can authorize the designated agent to exercise the signature right instead of himself. The original signer designated the agent to sign instead of himself, and does not need to provide his private key to the agent.

Al-Riyami et al. [2] presented the certificateless public key cryptography (CLPKC). The user's private key is made up of two parts: a partial private key yield by key generation center (KGC) and a secret value chosen by the user himself. It avoids key escrow and certificate management.

1.1 Related Work

Li et al. [3] put forward the first certificateless proxy signature (CLPS) scheme. Unfortunately, they did not show the security proofs. Choi et al. [4] and Lu et al. [5] indicated that the scheme [3] is insecure against proxy signature forgery attacks, and proposed an improved scheme, respectively. However, the security proofs of the improved schemes were not shown too. Chen et al. [6] put forward a new security model and constructed a new CLPS scheme with provably security. Xiong et al. [7] presented a CLPS scheme that needs to perform eight pairing operations. Seo et al. [8] and Zhang et al. [9] presented a CLPS scheme and showed the security proofs, respectively. Deng et al. [10] proposed a CLPS scheme that needs to perform two pairing operations. He et al. [11] presented a CLPS scheme from elliptic curve group. Deng et al. [12] proposed a CLPS scheme and showed the security proofs based on RSA problem. These two schemes [11-12] do not use pairing operations. Jin and Wen [13] put forward a certificateless multi-proxy signature (CLMPS) scheme. However, Xu et al. [14] indicated that the scheme has three disadvantages and proposed an improved scheme. Qu et al. [15] presented a CLMPS scheme that does not need to perform pairing operation. The security proofs of the above schemes were shown in ROM. Eslami and Pakniat [16] put forward the first CLPS scheme with provably security in SM. However, they did not provide a concrete security proof. Lu and Li [17] showed that the scheme [16] has some security drawbacks and presented a new scheme that is provably secure in SM. Ming and Wang [18] constructed a CLPS scheme, and gave the security proofs in SM. Yang et al. [19] came up with a new CLPS scheme and claimed that their scheme is provable secure in SM. However, Lin et al. [20] indicated that a Type II adversary can forge a valid signature in the scheme [19].

1.2 Motivations and Contributions

*Corresponding Author: Lunzhi Deng; E-mail: denglunzhi@163.com
DOI: 10.53106/160792642022032302008

In the last ten years, several concrete CLPS schemes were proposed. There are only two schemes [17-18] with provably security in SM. In the two schemes, the size of the system parameter increases linearly with the size of the user's identity information, and the number of addition operations on the elliptic curve group increases linearly with the size of the user's identity information. Therefore, these two schemes are not suitable for mobile computing scenarios. So it is attractive to design a CLPS scheme for mobile computing that is provable secure in SM and requires a constant number of pairing operations.

In this paper, we constructed a new CLPS scheme with the following features:

- It is secure against Type I/II adversary in SM
- It was constant that the size of system parameters and the size of master secret key.
- It was constant that the number of three kinds of operations (addition, scalar multiplication, and pairing).

1.3 Roadmap

The rest of the content are arranged as follows: First, we introduced the bilinear pairing and computation attack algorithm problem in Sec.2. Second, we proposed the system model and a concrete CLPS scheme in Sec.3 and Sec.4, respectively. Third, we presented the security model and the security proofs of new scheme in Sec.5 and Sec.6, respectively. Next, we made the performance comparisons on several schemes in Sec.7. Lastly, we gave some conclusions in Sec.8.

2 Preliminaries

For ease of understanding, we listed the symbols used in the paper in Table 1.

Bilinear pairing

Let $\hat{e}: G_1 \times G_1 \rightarrow G_2$ be a map with the following properties, where G_1 and G_2 are an additive group and a multiplicative group, respectively, and their order is q

- Bilinearity: $\hat{e}(aP_1, bP_2) = \hat{e}(P_1, P_2)^{ab}$ for all
- $P_1, P_2 \in G_1$ and $a, b \in Z_q$.
- Non-degeneracy: There exist $P_1, P_2 \in G_1$ such that $\hat{e}(P_1, P_2) \neq 1_{G_2}$.
- Computability: There is an efficient algorithm to compute $\hat{e}(P_1, P_2)$ for all P_1, P_2 $\in G_1$.

Definition 1 *CAA* (Computation attack algorithm) problem[21]. Given a generator P of the group G_1 and a tuple (P, xP), output a pair $\left(c, \frac{1}{x+c}P\right)$, where $c \in Z_q^*$.

3 System Model

A CLPS scheme consists of four entities: Key generation center (KGC), original signer, proxy signer and verifier.

KGC: It generates system parameters and publishes them to the outside world, generates a partial private key (PPK) according to the user's identity, and sends it to the user through an authenticated channel.

Original signer: He/she generates a delegation and sends it to the proxy signer.

Proxy signer: He/she generates a signature on a message on behalf of the original signer according to the delegation and sends it to the verifier.

Verifier: He/she checks its validity after receiving the signature.

A CLPS scheme contains the following eight algorithms:

- Setup: Inputs a security parameter v, KGC generates the system parameters (*params*) and master secret key (*msk*).
- PPK-Extract: Inputs an identity $ID_i \in \{0,1\}^*$, PKG generates a partial private key.
- SV-Set: Inputs an identity $ID_i \in \{0,1\}^*$, the user sets his own secret value.
- UPK-Generate: Inputs an identity $ID_i \in \{0,1\}^*$, the users generates a user public key.
- Delegate: Inputs a tuple (m_w, t_o, D_o), the original signer generates a delegation.
- Delegation-Verify: Inputs a tuple (m_w, δ), the proxy signer checks whether the delegation is valid.
- Proxy-Sign: Inputs a tuple (m, m_w, δ, t_p, D_p), the proxy signer generates a signature.
- PS-Verify: Inputs a tuple (m, m_w, δ, σ), the receiver checks whether the signature is valid.

Table 1. Notations

Symbol	Meaning
F_p	A prime finite field
q	A prime number
Z_q^*	A set making up of positive integers less than q
G_1	An additive group with prime order q
G_2	An multiplicative group with prime order q
$\hat{e}$	A bilinear pairing, where $\hat{e}: G_1 \times G_1 \rightarrow G_2$
x	The master secret key of system
P	A generator of the group G_1
P_{pub}	The public key of system, where $P_{pub} = xP$
H_1, H_2, H_3	Three secure hash functions
ID_o	The identity of the original signer
ID_p	The identity of the proxy signer
ID_i	The identity of the i^{th} user
$D_i = (R_i, d_i)$	The partial private key of the i^{th} user, where $R_i = r_i P$

t_i	The secret value of the i^{th} user, where $T_i = t_i P$
$PK_i = (R_i, T_i)$	The public key of the i^{th} user
m_w	$\{ID_o, PK_o, ID_p, PK_p\} \subseteq m_w$
δ	A delegation
σ	A proxy signature

4 Our Scheme

We proposed a new CLPS scheme as follows.

• Setup: Inputs a security parameter v, KGC performs following steps.

1. Selects a bilinear pairing $\hat{e}: G_1 \times G_1 \to G_2$, as that defined in sec.2 .
2. Picks a generator P of G_1, computes $E = \hat{e}(P, P)$.
3. Selects three secure hash functions $H_1, H_2, H_3 : \{0,1\}^* \to Z_q^*$.
4. Chooses $x \in Z_q^*$, computes $P_{pub} = xP$, sets $msk = \{x\}$
5. Publishes the system parameters $params = \{G_1, G_2, q, \hat{e}, P, P_{pub}, E, H_1, H_2, H_3\}$.

•PPK-Extract: For an identity $ID_i \in \{0,1\}^*$, KGC randomly picks $r_i \in Z_q^*$, computes $R_i = r_i P$, $k_i = H_1(ID_i, R_i)$ and $d_i = r_i + k_i x \bmod q$, then forwards $D_i = (R_i, d_i)$ to the user via an authenticated channel.

The user computes $k_i = H_1(ID_i, R_i)$, and checks whether $d_i P = R_i + k_i P_{pub}$. Accepts the partial private key if and only if the equation holds.

• SV-Set: The user ID_i randomly chooses $t_i \in Z_q^*$.

•UPK-Generate: The user ID_i computes $T_i = t_i P$, and sets $PK_i = (T_i, R_i)$

•Delegate: The original signer ID_o / PK_o performs the following steps to generate a delegation.

1. Computes $h = H_2(m_w, ID_o, PK_o)$.
2. Computes $Y = \dfrac{1}{d_o + ht_o} P$.
3. Outputs $\delta = Y$ as the delegation.

Where m_w includes ID_o / PK_o, ID_p / PK_p, and the delegation duration and so on.

•Delegation-Verify: To verify a delegation $(m_w, \delta = Y)$, the verifier does as follows.

1.Computes
$k_o = H_1(ID_o, R_o)$, $h = H_2(m_w, ID_o, PK_o)$.

2. Checks whether
$\hat{e}(Y, R_o + k_o P_{pub} + hT_o) = E$.

Accepts the delegation if and only if the equation holds.

• Proxy-Sign: The proxy signer ID_p / PK_p performs the following steps to generate a signature.

1.Computes $l = H_3(m, m_w, Y, ID_o, PK_o, ID_p, PK_p)$
2. Computes $Z = \dfrac{1}{d_p + lt_p} P$.
3. Outputs $\sigma = (Y, Z)$ as the signature.

• PS-Verify: To verify a signature ($m, m_w, \sigma = (Y, Z)$), the verifier does as follow:

1.Computes
$k_o = H_1(ID_o, R_o)$, $k_p = H_1(ID_p, R_p)$,
$h = H_2(m_w, ID_o, PK_o)$,
$l = H_3(m, m_w, Y, ID_o, PK_o, ID_p, PK_p)$.

2. Checks whether
$\hat{e}(Y, R_o + k_o P_{pub} + hT_o) = E$,
$\hat{e}(Z, R_p + k_p P_{pub} + lT_p) = E$.

Accepts the signature if and only if both of equalities hold.

5 Security Model

The security requirements of a CLPS scheme are presented as follows.

Definition 3. In the following two games, if the adversary's advantage is negligible, then the CLPS scheme is unforgeable (UNF-CLPS)

Game I. Challenger $\mathfrak{C}$ plays this game with a Type I adversary $\mathcal{A}_1$.

Initialization. $\mathfrak{C}$ gets the *msk* and *params* by running the Setup algorithm, then forwards *params* to $\mathcal{A}_1$ and keeps *msk* as secret.

Query. $\mathcal{A}_1$ issues various queries as follows.

• UPK-Query: $\mathcal{A}_1$ inputs an identity ID_i, $\mathfrak{C}$ returns a public key PK_i.

•UPK-Replacement: $\mathcal{A}_1$ submits a tuple ($PK_i^/$, ID_i), $\mathfrak{C}$ replaces PK_i with $PK_i^/$.

• PPK-Query: $\mathcal{A}_1$ submits an identity ID_i, $\mathfrak{C}$ returns a partial private key D_i. $\mathcal{A}_1$ cannot do it if the value R_i has been replaced

• SV-Query: $\mathcal{A}_1$ submits an identity ID_i, $\mathfrak{C}$ returns a secret value t_i. $\mathcal{A}_1$ can not do it if the value T_i has been replaced

•Delegation-Query: $\mathcal{A}_1$ submits a warrant m_w, $\mathfrak{C}$ returns a delegation δ.

•PS-Query: $\mathcal{A}_1$ submits a tuple (m, m_w, δ), $\mathfrak{C}$ returns a proxy signature σ.

Forge. $\mathcal{A}_1$ outputs a tuple (m_w^*, δ^*) or (m^*, m_w^*, σ^*). The adversary wins if one of following cases holds.

•Case 1: The final output is (m_w^*, δ^*) and it satisfies the following requirements.

1. Verify(m_w^*, δ^*) =1.
2. δ^* is not obtained by Delegation-Query.
3. $\mathcal{A}_1$ did not make PPK-Query for the original signer ID_o.

•Case 2: The final output is (m^*, m_w^*, σ^*) and it satisfies the following requirements.

1. Verify (m^*, m_w^*, σ^*) =1.
2. σ^* is not obtained by PS-Query..
3. $\mathcal{A}_1$ did not make Delegation-Query for the warrant m_w^*.
4. $\mathcal{A}_1$ did not make PPK-Query for the original signer ID_o.

•Case 3: The final output is (m^*, m_w^*, σ^*) and it satisfies the following requirements.

1. Verify (m^*, m_w^*, σ^*) =1.
2. σ^* is not obtained by PS-Query.
3. $\mathcal{A}_1$ did not make PPK-Query for the proxy signer ID_p.

The advantage of $\mathcal{A}_1$ is defined as: $Adv_{\mathcal{A}_1}^{UNF-CLS} = \Pr$ [$\mathcal{A}_1$ wins]

Game II. Challenger $\mathfrak{C}$ plays this game with a Type II adversary $\mathcal{A}_2$

Initialization. $\mathfrak{C}$ gets the *msk* and *params* by running the Setup algorithm, then forwards them to $\mathcal{A}_2$.

Query. $\mathcal{A}_2$ issues various queries as those in Game I.

Forge. $\mathcal{A}_2$ outputs a tuple (m_w^*, δ^*) or (m^*, m_w^*, σ^*). The adversary wins if one of following cases holds.

•Case 1: The final output is (m_w^*, δ^*) and it satisfies the following requirements.

1. Verify(m_w^*, δ^*) =1.
2. δ^* is not obtained by Delegate-Query.
3. $\mathcal{A}_2$ did not make SV-Query for the original signer ID_o.
4. $\mathcal{A}_2$ did not make UPK-Replacement for the original signer ID_o.

•Case 2: The final output is (m^*, m_w^*, σ^*) and it satisfies the following requirements.

1. Verify (m^*, m_w^*, σ^*) =1.
2. σ^* is not obtained by PS-Query..
3. $\mathcal{A}_2$ did not make Delegation-Query for the warrant m_w^*.
4. $\mathcal{A}_2$ did not make SV-Query for the original signer ID_o.
5. $\mathcal{A}_2$ did not make UPK-Replacement for the original signer ID_o.

•Case 3: The final output is (m^*, m_w^*, σ^*) and it satisfies the following requirements.

1. Verify (m^*, m_w^*, σ^*) =1.
2. The value σ^* is not obtained through PS-Query.
3. $\mathcal{A}_2$ did not make SV-Query for the proxy signer ID_p.
4. $\mathcal{A}_2$ did not make UPK-Replacement for the proxy signer ID_p.

The advantage of $\mathcal{A}_2$ is defined as: $Adv_{\mathcal{A}_2}^{UNF-CLS} = \Pr$ [$\mathcal{A}_2$ wins]

Remark: To forge a delegation (m_w^*, δ^*), $\mathcal{A}_1$ can make SV-Query for the original signer ID_o^* or even replace the value T_o^*. However, he can not get the value d_o^*. On the other hand, $\mathcal{A}_2$ can make PPK-Query for the original signer ID_o^*. However, he can not get the value t_o^*.

To forge a proxy signature (m^*, m_w^*, σ^*), $\mathcal{A}_1$ can make SV-Query for the proxy signer ID_p^* or even replace the value T_p^*. However, he can not get the value d_p^*. On the other hand, $\mathcal{A}_2$ can make PPK-Query for the proxy signer ID_p^*. However, he can not get the value t_p^*.

6 Security of Scheme

We gave the security proofs in SM. In the following proofs, the adversary can directly calculate the hash function instead of querying the challenger

6.1 Correctness of Delegation

$$\begin{aligned}
&\hat{e}\left(Y, R_o + k_o P_{pub} + hT_o\right) \\
&= \hat{e}\left(\frac{1}{d_o + ht_o} P, r_o P + k_o xP + ht_o P\right) \\
&= \hat{e}\left(\frac{1}{r_o + k_o x + ht_o} P, \left(r_o + k_o x + ht_o\right) P\right) = \hat{e}\left(P, P\right) \\
&= E
\end{aligned}$$

6.2 Correctness of Proxy Signature

$$\begin{aligned}
&\hat{e}\left(Z, R_p + k_p P_{pub} + lT_p\right) \\
&= \hat{e}\left(\frac{1}{d_p + lt_p} P, r_p P + k_p xP + lt_p P\right) \\
&= \hat{e}\left(\frac{1}{r_p + k_p x + lt_p} P, \left(r_p + k_p x + lt_p\right) P\right) \\
&= \hat{e}\left(P, P\right) \\
&= E
\end{aligned}$$

Theorem 1. If the CAA problem is hard, the scheme is unforgeable against an adversary $\mathcal{A}_1$ in SM.

Proof. Suppose that the challenger $\mathfrak{C}$ want to solve an instance of the CAA problem (P, aP), he does as follows.

Initialization. $\mathfrak{C}$ runs the Setup algorithm with a parameter v, then gives $\mathcal{A}_1$ the

$params = \{G_1, G_2, q, \hat{e}, P, P_{pub} = xP, E, H_1, H_2, H_3\}$

Queries. $\mathcal{A}_1$ will first perform UPK-query for each identity.

• UPK-Query: $\mathfrak{C}$ maintains a list L_U of tuple (ID_i, t_i, r_i) .When $\mathcal{A}_1$ inputs an identity ID_i, $\mathfrak{C}$ does as follows:

1. For $i = j$, picks at random $t_j \in Z_q^*$, sets $ID_j = ID^\Diamond$, returns $PK_j = PK^\Diamond = (t_j P, aP)$, then stores the tuple $(ID_j, t_j, \Diamond)$ in the list L_U

2. For $i \neq j$, randomly picks $t_i, r_i \in Z_q^*$ and returns $PK_i = (t_i P, r_i P)$, then stores the tuple (ID_i, t_i, r_i) in the list L_U.

• UPK-Replacement: $\mathfrak{C}$ maintains a list L_R of tuple $(ID_i, PK_i, PK_i^/)$. When $\mathcal{A}_1$ submits a tuple (ID_i, $PK_i^/$), $\mathfrak{C}$ replaces PK_i with $PK_i^/$ and adds $(ID_i, PK_i, PK_i^/)$ to the list L_R.

• PPK-Query: $\mathfrak{C}$ maintains a list L_D of tuple (ID_i, D_i). When $\mathcal{A}_1$ submits an identity ID_i. If $ID_i = ID^\Diamond$, $\mathfrak{C}$ fails. Otherwise, $\mathfrak{C}$ finds (ID_i, t_i, r_i) in the list L_U, gives the D_i by running PPK-Extract algorithm and adds (ID_i, D_i) to the list L_D.

• SV-Query: When $\mathcal{A}_1$ submits an identity ID_i, $\mathfrak{C}$ finds the tuple (ID_i, t_i, r_i) in the list L_U, and responds with t_i.

• Delegation-Query: When $\mathcal{A}_1$ submits a warrant m_w, $\mathfrak{C}$ generates a delegation as follows:

1. $ID_o \neq ID^\Diamond$ and $ID_o \notin L_R$, $\mathfrak{C}$ outputs a delegation δ by running Delegate algorithm.

2. $ID_o \in L_R$, then $PK_o = (t_o P, r_o P)$ has been replaced by $PK_o^/ = (t_o^/ P, r_o^/ P)$. If $t_o^/ \neq t_o$ (or $r_o^/ \neq r_o$), $\mathcal{A}_1$ must send the value $t_o^/$ (or $r_o^/$) to $\mathfrak{C}$, $\mathfrak{C}$ obtains the value $t_o^/$ (or $d_o^/$) by running SV-Set (or PPK-Extraction) algorithm, and finally outputs a delegation δ by running Delegate algorithm.

3. $ID_o{=}ID^\Diamond$, then $\mathfrak{C}$ fails.

• PS-Query: When $\mathcal{A}_1$ submits a tuple (m, m_w, δ), $\mathfrak{C}$ gerenates a signature as follows:

1. $ID_p \neq ID^\Diamond$ and $ID_p \notin L_R$, $\mathfrak{C}$ outputs a proxy signature σ by running Proxy Sign algorithm.

2. $ID_p \in L_R$, then $PK_p = (t_p P, r_p P)$ has been replaced by $PK_p^/ = (t_p^/ P, r_p^/ P)$. If $t_p^/ \neq t_p$ (or $r_p^/ \neq r_p$), $\mathcal{A}_1$ must send the value $t_p^/$ (or $r_p^/$) to $\mathfrak{C}$, $\mathfrak{C}$ then obtains the value $t_p^/$ (or $d_p^/$) by running SV-Set (or PPK-Extract) algorithm, and finally outputs a proxy signature σ by running Proxy Sign algorithm.

3. $ID_p{=}ID^\Diamond$, then $\mathfrak{C}$ fails.

Forge. $\mathcal{A}_1$ outputs a tuple (m_w^*, δ^*) or (m^*, m_w^*, σ^*). $\mathfrak{C}$ aborts if the output does not satisfy any of the cases in Game I. Otherwise, $\mathfrak{C}$ resolves the CAA example as follows:

• Case 1. The final output is (m_w^*, δ^*) and fulfills the conditions of Case 1 in Game I.

In fact, δ^* is a signature on the warrant m_w^*, then $\delta^* = Y^* = \frac{1}{r_o^* + k_o^* x + h^* t_o^*} P$. If $ID_o^*{=}ID^\Diamond$, then $ID_o^*{=}ID_j, PK_o^*{=}PK_j$, namely, $(t_o^* P, r_o^* P) = (t_j P, aP)$. $\mathfrak{C}$ finds t_j in the list L_U, computes $h^* = H_2(m_w^*, ID_o^*, PK_o^*)$, $k_o^* = H_1(ID_o^*, aP)$ and $c = k_o^* x + h^* t_o^*$ (where $t_o^* = t_j$), outputs a solution of the CAA example in the end.

$$(c, \delta^*) = \left(c, \frac{1}{r_o^* + k_o^* x + h^* t_o^*} P\right)$$
$$= \left(c, \frac{1}{a + k_o^* x + h^* t_o^*} P\right) = \left(c, \frac{1}{a+c} P\right).$$

Probability. Let q_U, q_R and q_D be the number of UPK-Query, UPK-Replacement and PPK-Query , respectively.

Some notations are defined as follows.

π_1 : $\mathcal{A}_1$ did not make PPK-Query on ID_o^*, nor did make UPK-Replacement on it.

π_2 : $\mathfrak{C}$ did not fail in Delegation-Query and PS-Query.

π_3: $ID_o^*{=}ID^\Diamond$.

It is a reasonable assumption that $L_R \cap L_D = \varnothing$. Hence it is not difficult to obtain the following results:

$$\Pr[\pi_1] = \frac{q_U - q_R - q_D}{q_U}, \Pr[\pi_2 | \pi_1] = 1 - \frac{1}{q_U},$$

$$\Pr[\pi_3 | \pi_1 \wedge \pi_2] = \frac{1}{q_U - q_R - q_D}$$

$$\Pr[\mathfrak{C} \text{ success}] = \Pr[\pi_1 \wedge \pi_2 \wedge \pi_3]$$
$$= \Pr[\pi_1] \cdot \Pr[\pi_2 | \pi_1] \cdot \Pr[\pi_3 | \pi_1 \wedge \pi_2]$$
$$= \frac{q_U - q_R - q_D}{q_U} \cdot \left(1 - \frac{1}{q_U}\right) \cdot \frac{1}{q_U - q_R - q_D}$$
$$\approx \frac{1}{q_U}$$

Therefore, $\mathfrak{C}$ can resolve the CAA example with the probability $\frac{\varepsilon}{q_U}$ if $\mathcal{A}_1$ can succeed with the probability ε

• Case 2. The final output is (m^*, m_w^*, σ^*) and fulfills the conditions of Case 2 in Game I.

In fact, $\sigma^* = (Y^*, Z^*)$ is a proxy signature on the tuple (m^*, m_w^*), then $Y^* = \frac{1}{r_o^* + k_o^* x + h^* t_o^*} P$. If $ID_o^*{=}ID^\Diamond$, then $ID_o^*{=}ID_j, PK_o^*{=}PK_j$, namely, $(t_o^* P, r_o^* P) = (t_j P, aP)$. $\mathfrak{C}$ finds t_j in the list L_U, computes $k_o^* = H_1(ID_o^*, aP)$, $h^* = H_2(m_w^*, ID_o^*, PK_o^*)$ and

$c=k_o^*x+h^*t_o^*$ (where $t_o^*=t_j$), outputs a solution of the CAA example in the end.

$$(c,\ Y^*)=\left(c,\frac{1}{r_o^*+k_o^*x+h^*t_o^*}P\right)$$
$$=\left(c,\frac{1}{a+k_o^*x+h^*t_o^*}P\right)=\left(c,\frac{1}{a+c}P\right)$$

Probability. Same as that in Case 1.

• Case 3. The final output is (m^*,m_w^*,σ^*) and fulfills the conditions of Case 3 in Game I.

In fact, $\sigma^*=(Y^*,Z^*)$ is a proxy signature on the tuple (m^*,m_w^*), then $Z^*=\frac{1}{r_p^*+k_p^*x+l^*t_p^*}P$

. If $ID_p^*=ID^\Diamond$, then $ID_p^*=ID_j, PK_p^*=PK_j$, namely, $\left(t_p^*P,r_p^*P\right)=\left(t_jP,aP\right)$. $\mathfrak{C}$ finds t_j in the list L_U, computes $k_p^*=H_1\left(ID_p^*,aP\right)$, $l^*=H_3\left(m^*,m_w^*,Y^*,ID_o^*,PK_o^*,ID_p^*,PK_p^*\right)$ and $c=k_p^*x+l^*t_p^*$ (where $t_p^*=t_j$), outputs a solution of the CAA example in the end.

$$\left(c,Z^*\right)=\left(c,\frac{1}{r_p^*+k_p^*x+l^*t_p^*}P\right)$$
$$=\left(c,\frac{1}{a+k_p^*x+l^*t_p^*}P\right)=\left(c,\frac{1}{a+c}P\right)$$

Probability. Same as that in Case 1.

Theorem 2. If the CAA problem is tricky, the scheme is unforgeable against an adversary $\mathcal{A}_2$ in SM.

Proof. Suppose that the challenger $\mathfrak{C}$ want to solve an instance of the CAA problem $\left(P,aP\right)$, he does as follows.

Initialization. $\mathfrak{C}$ runs the Setup algorithm with a parameter v, then gives $\mathcal{A}_2$ the $params=\left\{G_1,G_2,q,\hat{e},P,P_{pub}=xP,E,H_1,H_2,H_3\right\}$ and $msk=\left\{x\right\}$

Queries. $\mathcal{A}_2$ will first perform UPK-query for each identity..

• UPK-Query: $\mathfrak{C}$ maintains a list L_U of tuple $\left(ID_i,t_i,r_i\right)$ .When $\mathcal{A}_2$ inputs an identity ID_i, $\mathfrak{C}$ does as follows:

1. For $i=j$, selects at random $r_j\in Z_q^*$, sets $ID_j=ID^\Diamond$, returns $PK_j=PK^\Diamond=\left(aP,r_jP\right)$, then stores the tuple $\left(ID_j,\Diamond,r_j\right)$ in the list L_U

2. For $i\neq j$, selects at random $t_i,r_i\in Z_q^*$ and returns $PK_i=\left(t_iP,r_iP\right)$, then stores the tuple $\left(ID_i,t_i,r_i\right)$ in the list L_U.

• UPK-Replacement: Same as that in the Theorem 1.

• PPK-Query: When $\mathcal{A}_2$ submits an identity ID_i, $\mathfrak{C}$ finds $\left(ID_i,t_i,r_i\right)$ in the list L_U, gives the D_i by running the PPK-Extract algorithm.

• SV-Query: $\mathfrak{C}$ maintains a list L_E of tuple $\left(ID_i,t_i\right)$. When $\mathcal{A}_2$ submits an identity ID_i, If $ID_i=ID^\Diamond$, $\mathfrak{C}$ fails. Otherwise, $\mathfrak{C}$ finds $\left(ID_i,t_i,r_i\right)$ in the list L_U, responds with t_i, then adds $\left(ID_i,D_i\right)$ to the list L_E.

• Delegation-Query: Same as that in the **Theorem 1.**

• PS -Query: Same as that in the **Theorem 1.**

Forge. $\mathcal{A}_2$ outputs a tuple (m_w^*,δ^*) or (m^*,m_w^*,σ^*). $\mathfrak{C}$ aborts if the output does not satisfy any of the cases in Game II. Otherwise, $\mathfrak{C}$ resolves the CAA example as follows:

• Case 1. The final output is (m_w^*,δ^*) and fulfills the conditions of Case 1 in Game II.

In fact, δ^* is a signature on the warrant m_w^*, then $\delta^*=Y^*=\frac{1}{r_o^*+k_o^*x+h^*t_o^*}P$. If $ID_o^*=ID^\Diamond$, then $ID_o^*=ID_j$, $PK_o^*=PK_j$ namely, $\left(t_o^*P,r_o^*P\right)=\left(aP,r_jP\right)$. $\mathfrak{C}$ finds r_j in the list L_U, computes $h^*=H_2\left(m_w^*,ID_o^*,PK_o^*\right)$, $k_o^*=H_1\left(ID_o^*,aP\right)$ and $c=h^{*-1}(r_o^*+k_o^*x)$ (where $r_o^*=r_j$), outputs a solution of the CAA example in the end.

$$(c,h^*\delta^*)=\left(c,\frac{h^*}{r_o^*+k_o^*x+h^*t_o^*}P\right)$$
$$=\left(c,\frac{h^*}{r_o^*+k_o^*x+h^*a}P\right)=\left(c,\frac{1}{h^{*-1}(r_o^*+k^*x)+a}P\right)$$
$$=\left(c,\frac{1}{c+a}P\right)$$

Probability. Let q_U,q_R and q_E be the number of UPK-Query, UPK-Replacement and SV-Query, respectively. Some notations are defined as follows.

π_1: $\mathcal{A}_2$ did not make SV-Query on ID_o^*, nor did make UPK-Replacement on it.

π_2: $\mathfrak{C}$ did not fail during Delegation-Query and PS-Query.

π_3: $ID_o^*=ID^\Diamond$.

It is a reasonable assumption that $L_R\bigcap L_E=\varnothing$. Hence it is not difficult to obtain the following results:

$$\Pr\left[\pi_1\right]=\frac{q_U-q_R-q_E}{q_U},\quad \Pr\left[\pi_2|\pi_1\right]=1-\frac{1}{q_U}$$
$$\Pr\left[\pi_3|\pi_1\wedge\pi_2\right]=\frac{1}{q_U-q_R-q_E}$$
$$\Pr[\mathfrak{C}\ \ \text{success}]=\Pr[\pi_1\wedge\pi_2\wedge\pi_3]$$
$$=\Pr[\pi_1]\cdot\Pr[\pi_2\mid\pi_1]\cdot\Pr[\pi_3\mid\pi_1\wedge\pi_2]$$
$$=\frac{q_U-q_R-q_E}{q_U}\cdot\left(1-\frac{1}{q_U}\right)\cdot\frac{1}{q_U-q_R-q_E}$$
$$\approx\frac{1}{q_U}$$

Therefore, $\mathfrak{C}$ can resolve the CAA example with the probability ε / q_U if $\mathcal{A}_2$ can succeed with the probability ε.

• Case 2. The final output is (m^*, m_w^*, σ^*) and fulfills the conditions of Case 2 in Game II.

In fact, $\sigma^* = (Y^*, Z^*)$ is a proxy signature on the tuple (m^*, m_w^*), then $Y^* = \frac{1}{r_o^* + k_o^* x + h^* t_o^*} P$

If $ID_o^* = ID^\Diamond$, then $ID_o^* = ID_j, PK_o^* = PK_j$, namely, $\left(t_o^* P, r_o^* P\right) = \left(aP, r_j P\right)$. $\mathfrak{C}$ finds r_j in the list L_U, computes $h^* = H_2\left(m_w^*, ID_o^*, PK_o^*\right)$, $k_o^* = H_1\left(ID_o^*, aP\right)$ and $c = h^{*-1}(r_o^* + k_o^* x)$ (where $r_o^* = r_j$), outputs a solution of the CAA example in the end.

$$(c, h^* Y^*) = \left(c, \frac{h^*}{r_o^* + k_o^* x + h^* a} P\right)$$
$$= \left(c, \frac{1}{h^{*-1}(r_o^* + k_o^* x) + a} P\right) = \left(c, \frac{1}{a+c} P\right)$$

Probability. Same as that in Case 1.

• Case 3. The final output is (m^*, m_w^*, σ^*) and fulfills the conditions of Case 3 in Game II.

In fact, $\sigma^* = (Y^*, Z^*)$ is a proxy signature on the tuple (m^*, m_w^*), then $Z^* = \frac{1}{r_p^* + k_p^* x + l^* t_p^*} P$. If $ID_p^* = ID^\Diamond$, then $ID_p^* = ID_j$, $PK_p^* = PK_j$, namely, $\left(t_p^* P, r_p^* P\right) = \left(aP, r_j P\right)$. $\mathfrak{C}$ finds r_j in the list L_U, computes $k_p^* = H_1\left(ID_p^*, aP\right)$, $l^* = H_3\left(m^*, m_w^*, Y^*, ID_o^*, PK_o^*, ID_p^*, PK_p^*\right)$ and $c = l^{*-1}(r_p^* + k_p^* x)$ (where $r_p^* = r_j$), outputs a solution of the CAA example in the end.

$$\left(c, l^* Z^*\right) = \left(c, \frac{l^*}{r_p^* + k_p^* x + l^* a} P\right)$$
$$= \left(c, \frac{1}{l^{*-1}(r_p^* + k_p^* x) + a} P\right) = \left(c, \frac{1}{a+c} P\right)$$

Probability. Same as that in Case 1.

7 Efficiency and Comparison

We compared the performance of the new scheme with the other two CLPS schemes. We listed the symbols in Table 2 that need to be used in this section.

Table 2. Notations

B_P	S_{G_1}	A_{G_1}	M_{G_2}	E_{G_2}	H
5.427	2.165	0.013	0.001	0.339	0.007

We used third-party data to analyze the performance of the three CLPS schemes. The running times on basic cryptographic operations are listed in Table 3. In order to accomplish 1024-bit RSA security level, the hardware/software parameters used in the experiments [22] are as follows: cryptographic library (MIRACL) and a computer (Dell with an I5-4460S 2.90GHz processor, 4G bytes memory and the Window 8 operating system), a Tate pairing $\hat{e}: G_1 \times G_1 \to G_2$ and G_1 with order q is an additive group over a super singular curve $E / E_p : y^2 = x^3 + 1$, where p and q are 512-bits prime number and 160-bits prime number, respectively.

We used a simple, intuitive method to evaluate the computational efficiency of the three schemes, as shown in Figure 1. Without loss of generality, it is assumed that the size of the identity information of a user is 60 bits, and the size of the output of a hash function (SHA1) is 160 bits. Namely, $|B_{ID}| = n = 60$ (bits) and $|B_m| = |B_w| = u = 160$ (bits). Further,

$$|\Delta_{ID}| = \frac{n}{2} = 30 \text{ (bits) and } \quad |\Delta_m| = |\Delta_w| = \frac{u}{2} = 80 \text{ (bits).}$$

In Delegation, the scheme [17] requires 3 scale multiplication operations in G_1, 2 addition operations in G_1 and 1 hash function opretions. In Delegation-Verify, it requires 4 pairing operations, 1 scale multiplication operations in G_1, $\frac{n+1}{2}$ addition operations in G_1, 2 multiplication operations in G_2 and 2 hash function opretions. In Proxy-Sign, it requires 6 scale multiplication operations in G_1, 6 addition operations in G_1 and 2 hash function opretions. In PS-Verify, it requires 5 pairing operations, $n+2$ addition operations in G_1, and 2 scale multiplication operations in G_1, 5 multiplication operation in G_2 and 2 hash function opretions. So the resulting computation time is $5.427 \times 9 + 2.165 \times 12 +$

$$0.013 \times \left(\frac{3n}{2} + 11\right) + 0.001 \times 7 + 0.007 \times 6 = 0.013$$

$\times \frac{3n}{2} + 75.015$. When $n = 60$, the computation time is $0.013 \times \frac{3 \times 60}{2} + 75.015 = 76.185$ ms.

Table 3. Operation time (in milliseconds)

Symbol	Meaning
B_{ID}	A bit string of the identity information of a user, where $\lvert B_{ID} \rvert = n$
$B[i]_{ID}$	The i^th bit of the identity information of a user
Δ_{ID}	A set of indices i such that $B[i]_{ID} = 1$, namely $\Delta_{ID} = \{i : B[i]_{ID} = 1\}$
B_w	The output bit string of a warrant hush function, where $\lvert B_w \rvert = u$
$B[i]_w$	The i^th bit of a warrant
Δ_w	A set of indices i such that $B[i]_w = 1$, namely $\Delta_w = \{i : B[i]_w = 1\}$
B_m	The output bit string of a message hush function, where $\lvert B_m \rvert = u$

$B[i]_m$	The i[th] bit of a message
Δ_m	A set of indices i such that $B[i]_m = 1$, namely $\Delta_m = \{i : B[i]_m = 1\}$
B_P	A pairing operation
S_{G_1}	A scale multiplication operation in G_1
A_{G_1}	A addition operation in G_1
M_{G_2}	A multiplication operation in G_2
E_{G_2}	An exponentiation operation in G_2
H	A hash function operation
$\lvert G_1 \rvert$	An element in G_1
$\lvert G_2 \rvert$	An element in G_2
$\lvert Z_q^* \rvert$	An element in Z_q^*

In Delegation, the scheme [18] requires 4 scale multiplication operations in G_1 and $\frac{u}{2}+1$ addition operations in G_1 and 1 hash function opretions. In Delegation-Verify, it requires 3 pairing operations, $\frac{n+u}{2}$ addition operations in G_1, 3 multiplication operations in G_2 and 1 hash function opretions. In Proxy-Sign, it requires 10 scale multiplication operations in G_1, $n+u+8$ addition operations in G_1 and 4 hash function opretions. In PS-Verify, it requires 5 pairing operations, $n+u$ addition operations in G_1, 5 multiplication operation in G_2 and 4 hash function opretions. So the resulting computation time is $5.427\times 8+2.165\times 14+0.013\times\left(\frac{5}{2}n+3u+9\right)+0.001\times 8+0.007\times 10=0.013\times\left(\frac{5}{2}n+3u+9\right)+73.804$ ms. When $n=60, u=160$, the computation time is $0.013\times(\frac{5\times 60}{2}+3\times 160+9)+73.804=82.111$ ms.

In Delegation, new scheme requires 1 scale function opretions. In Delegation-Verify, it requires 1 pairing operations, 2 scale multiplication operations in G_1, 2 addition operations in G_1 and 1 hash function opretions. In Proxy-Sign, it requires 1 scale multiplication operations in G_1 and 1 hash function opretions. In PS-Verify, it requires 2 pairing operations, 4 scale multiplication operations in G_1, 4 addition operations in G_1 and 4 hash function opretions. So the resulting computation time is $5.427\times 3+2.165\times 8+0.013\times 6+0.007\times 8=33.735$ m.

We listed the computation costs for the three CLPS schemes in Table 4.

Table 4. Comparison of three CLPS schemes

Scheme	Lu [17]	Ming [18]	Our scheme
Delegate	$3S_{G_1}+2A_{G_1}+H$	$4S_{G_1}+(\frac{u}{2}+1)A_{G_1}+H$	$S_{G_1}+H$
Delegation-Verify	$4B_P+(\frac{n}{2}+1)A_{G_1}+S_{G_1}+2M_{G_2}+H$	$3B_P+(\frac{n+u}{2})A_{G_1}+3M_{G_2}+H$	$B_P+2S_{G_1}+2A_{G_1}+2H$
Time $(n=60, u=160)$	$0.013\times\frac{n}{2}+30.423$ (30.813)	$0.013\times(\frac{n}{2}+u)+24.971$ (27.441)	11.969
Proxy-Sign	$6S_{G_1}+6A_{G_1}+2H$	$10S_{G_1}+(n+u+8)A_{G_1}+4H$	$S_{G_1}+H$
PS-Verify	$5B_P+(n+2)A_{G_1}+2S_{G_1}+5M_{G_2}+2H$	$5B_P+(n+u)A_{G_1}+5M_{G_2}+4H$	$2B_P+4S_{G_1}+4A_{G_1}+4H$
Time $(n=60, u=160)$	$0.013n+44.592$ (45.372)	$0.013\times(2n+2u)+48.95$ (54.670)	21.766
Total Time $(n=60, u=160)$	$0.013\times\frac{3n}{2}+75.015$ (76.185)	$0.013\times(\frac{5n}{2}+3u)+73.921$ (82.111)	33.735
Size of Params $(n=60, u=160)$	$(n+8)\lvert G_1\rvert$ (4352 bytes)	$(n+2u+5)\lvert G_1\rvert$ (24640 bytes)	$2\lvert G_1\rvert+\lvert G_2\rvert$ (192 bytes)
Size of MSK	$\lvert G_1\rvert$ (64 bytes)	$\lvert G_1\rvert$ (64 bytes)	$\lvert Z_q^*\rvert$ (20 bytes)
Size of Signature	$5\lvert G_1\rvert$ (320 bytes)	$5\lvert G_1\rvert$ (320 bytes)	$2\lvert G_1\rvert$ (128bytes)

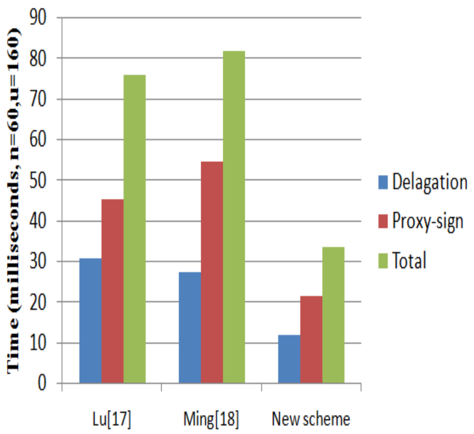

Figure 1. Computation cost

Follow on, we evaluated the size of system parameters, the size of master key and the size of signature, as shown in Figure 2. In the scheme [17], the system parameters contain $n+8$ points over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $[(60+8)\times 512]/8 = 4352$ bytes. The master secret key contains one point over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $512/8 = 64$ bytes. The signature contains five points over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $(5\times 512)/8 = 320$ bytes. In the scheme [18], the system parameters contain $n+2u+5$ points over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $[(60+320+5)\times 512]/8 = 24640$ bytes. The master secret key contains one point over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $512/8 = 64$ bytes. The signature contains five points over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $(5\times 512)/8 = 320$ bytes. In our scheme, the system parameters contain three points over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $[3\times 512]/8 = 192$ bytes. The master secret key contains one point over Z_q^*, thus the size is $160/8 = 20$ bytes. The signature contains two points over an elliptic curve $E/E_p: y^2 = x^3 + 1$, thus the size is $(2\times 512)/8 = 128$ bytes.

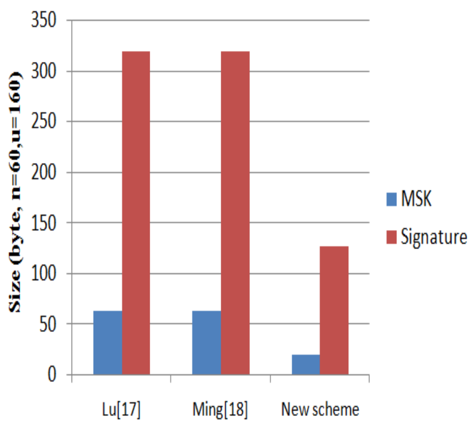

Figure 2. Storage cost

With the continuous advancement of network technology, electronic signatures are widely used in various scenarios, such as e-commerce, electronic voting, and remote access control. The original signer may be inconvenient to generate a signature in certain situations (for example sickness, imprisonment, etc.), and he/she can authorize a trustworthy person to exercise the right to sign. In the new sheme, the proxy signer generates a new signature with his own private key and delegation generated by the original signer, then sends it to the receiver. Verifying the signature, the receiver can determine whether the signature and delegation are valid. The new scheme requires only three pairing operations and enjoys lower computation cost.

8 Conclusion

It was in ROM that most known CLPS schemes is proved to be secure. There are only two CLPS schemes with provably secure in SM. In ROM, the hash function value obtained by the adversary is provided by the challenger, rather than by a real function computation. A cryptography scheme that has been proven to be secure in ROM is not necessarily secure in real-world applications. In this paper, we constructed a new CLPS scheme and showed the security proofs in SM. In the scheme, it was constant that the size of system parameters and

the size of master secret key, it was constant that the number of three kinds of operations (addition, scalar multiplication and pairing). It was shown that the proposed scheme is more efficient and suitable for mobile computing.

Acknowledgment

This research is supported by the National Natural Science Foundation of China under Grant No. 61962011, Guizhou Provincial Science and Technology Foundation Science under Grant No. [2019] 1434. Guiyang City Science and Technology Plan Project under Grant No. [2021] 43-8. Guizhou Normal University Academic New Seedling Project under Grant No. [2019].

References

[1] M. Mambo, K. Usuda, E. Okamoto, Proxy signature: Delegation of the power to sign messages, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E79-A, No. 9, pp. 1338-1354, September, 1996.

[2] S. Al-Riyami, K. Paterson, Certificateless public key cryptography, in: C. S. Laih (Eds.), *Advances in Cryptology- Asiacrypt, Lecture Notes in Computer Sciences, vol. 2894*, Springer, 2003, pp. 452-473.

[3] X. Li, K. Chen, L. Sun, Certificateless signature and proxy signature schemes from bilinear pairings, *Lithuanian Mathematical Journal*, Vol. 45, No. 1, pp. 76-83, January, 2005.

[4] R. Lu, D. He, C. Wang, Cryptanalysis and improvement of a certificateless proxy signature scheme from bilinear pairings, *8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, Qingdao, China, 2007, pp. 285-290.

[5] K. Choi, D. Lee, Certificateless proxy signature scheme, *3rd International Conference on Multimedia, Information Technology and its Applications (MITA*

2007), Manila, Philippines, 2007, pp. 437-440.
[6] H. Chen, F. Zhang, R. Song, Certificateless proxy signature scheme with provable security, *Journal of Software*, Vol. 20, No. 3, pp. 692-701, March, 2009.
[7] H. Xiong, F. Li, Z. Qin, A provably secure proxy signature scheme in certificateless cryptography, *Informatica*, Vol. 21, No. 2, pp. 277-294, April, 2010.
[8] S. Seo, K. Choi, J. Hwang, S. Kim, Efficient certificateless proxy signature scheme with provable security, *Information Sciences*, Vol. 188, pp. 322-337, April, 2012.
[9] L. Zhang, F. Zhang, Q. Wu, Delegation of signing rights using certificateless proxy signatures, *Information Sciences*, Vol. 184, No. 1, pp. 298-309, February, 2012.
[10] L. Deng, J. Zeng, H. Huang, Efficient certificateless proxy signature scheme, *International Journal of Foundations of Computer Science*, Vol. 27, No. 1, pp. 85-100, January, 2016.
[11] D. He, Y. Chen, J. Chen, An efficient certificateless proxy signature scheme without pairing, *Mathematical and Computer Modelling*, Vol. 57, No. 9-10, pp. 2510-2518, May, 2013.
[12] L. Deng, J. Zeng, Y. Qu, Certificateless proxy signature from RSA, *Mathematical Problems in Engineering*, Vol. 2014, Article No. 373690, pp. 1-10, June, 2014,
[13] Z. Jin, Q. Wen, Certificateless multi-proxy signature, *Computer Communications*, Vol. 34, No. 3, pp. 344-352, March, 2011.
[14] J. Xu, H, Sun, Q. Wen, H. Zhang, Improved certificateless multi-proxy signature, *Journal of China Universities of Posts and Telecommunications*, Vol. 19, No. 4, pp. 94-105, August, 2012.
[15] Y. Qu, L. Deng, X. Bao, H. Huang, An efficient certificateless multi-proxy signature scheme without pairing, *International Journal of Electronic Security and Digital Forensics*, Vol. 8, No. 2, pp. 148-163, March, 2016.
[16] Z. Eslami, N. Pakniat, A certificateless proxy signature scheme secure in standard model, *International Conference on Latest Computational Technologies*, Bangkok, Thailand, 2012, pp. 81-84.
[17] Y. Lu, J. Li, Provably secure certificateless proxy signature scheme in the standard model, *Theoretical Computer Science*, Vol. 639, pp. 42-59, August, 2016.
[18] Y. Ming, Y. Wang, Certificateless proxy signature scheme in the standard model, *Fundamenta Informaticae*, Vol. 160, No. 4, pp. 409-445, July, 2018.
[19] W. Yang, J. Weng, X. Huang, A. Yang, A provably secure certificateless proxy signature scheme against malicious-but-passive KGC attacks, *The Computer Journal*, Vol. 63, No. 1, pp. 1139-1147, January, 2020.
[20] X. Lin, Q. Wang, L. Sun, Z. Yan, P. Liu, Security analysis of the first certificateless proxy signature scheme against malicious-but-passive KGC attacks, *The Computer Journal*, Vol. 64, No. 4, pp. 653-660, April, 2021.
[21] L. Deng, Y. Yang, R. Gao, Y. Chen, Certificateless short signature scheme from pairing in the standard model, *International Journal of Communication Systems*, Vol. 31, No. 17, Article No. e3796, November, 2018.
[22] D. He, N. Kumar, K. Choo, W. Wu, Efficient hierarchical identity-based signature with batch verification for automatic dependent surveillance-broadcast system, *IEEE Transactions on Information Forensics and Security*, Vol. 12, No. 2, pp. 454-464, February, 2017.

Biographies

Lunzhi Deng received his B.S. from Guizhou Normal University, Guiyang, China, in 2002; M.S. from Guizhou Normal University, Guiyang, China, in 2008; and Ph.D. from Xiamen University, Xiamen, China, in 2012. He is now a professor in the School of Mathematical Sciences, Guizhou Normal University, Guiyang, China. His recent research interests include cryptography and information security.

Zhenyu Hu received the B.S. degree from Shanghai University of Electric Power, Shanghai, China, in 2016. He is currently a postgraduate student with Guizhou Normal University, Guiyang, China. His research interests include cryptography and information security.

Yu Ruan received her B.S. from Qiushi College of Guizhou Normal University, Guiyang, China, in 2020. She is currently a postgraduate student with Guizhou Normal University, Guiyang, China. Her research interests include cryptography and information security.

Tao Wang received his B.S. from Information College of Huaibei Normal University, Huaibei, China, in 2020. He is currently a postgraduate student with Guizhou Normal University, Guiyang, China. His research interests include cryptography and information security.