Research Article

An Efficient and Provably-Secure Certificateless Proxy-Signcryption Scheme for Electronic Prescription System

Li Li,1,2 Siqin Zhou,1,2 Kim-Kwang Raymond Choo,3 Xiaohong Li,4 and Debiao He1,2

1School of Cyber Science and Engineering, Wuhan University, Wuhan, China
2Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing, China
3Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, USA
4Computer School, Wuhan University, Wuhan, China

Correspondence should be addressed to Xiaohong Li; leexh@whu.edu.cn

Received 8 April 2018; Accepted 10 June 2018; Published 29 August 2018

1. Introduction

Recent advances in cryptographic techniques and consumer and communication technologies have resulted in the migration of services from the brick-and-mortar model to an online model, where transactions are being conducted from mobile devices (e.g., Android and iOS devices and potentially wearable and embedded devices). One such industry application is electronic prescriptions (e-prescriptions) [1,2], where prescriptions are being sent electronically from a medical practitioner/medical practice to the pharmacist/pharmacy. Payments can also be made online (e.g., using credit cards or bank transfers), and the medications can either be picked up from the pharmacy or delivered to the user’s home [3]. Benefits of an e-prescription system extend beyond mere convenience to the users. For example, pharmacists no longer have to ‘decipher’ the hand-written prescription, which saves time and costs (e.g., having to call the medical practitioner to confirm the actual prescription) and minimizing the chance for errors [4]. Such errors can be fatal. For example, in a study by Brits and Verma [5], it was found that illegible handwriting and other prescription errors on prescriptions resulted in “lorazepam injection 4 mg” being misread as “40 mg (lethal dose) by 20% of [the] healthcare workers.”

There are situations where the patient may not be able to collect the medication in person, for example, due to physical injury or medical condition (e.g., severe gout attack resulting in the patient unable to walk and collect prescribed medication such as Colchicine). Thus, the patient has to give another individual (e.g., family member or neighbor) the proxy delegation to collect the medication on his/her behalf. Pharmacists have legal obligations when handling, dispensing, and supplying medications, particularly drugs of dependence. Therefore, it is important to ensure the security and efficiency of generating such a delegation in the e-prescription system.
One such solution is proxy signcryption, as it allows the delegation of signing privileges in computing devices such as mobile devices. The security of such schemes, as well as many other cryptographic schemes (e.g., key agreement and signature schemes), generally relies on the intractability of hard problems such as Diffie-Hellman problem, integer factorization problem, and discrete logarithm problem [6–10]. In recent times, there have been a large number of proxy-signcryption schemes proposed that are based on bilinear pairings [11–13]. However, such schemes often have high computational and communication costs; thus, they are not suited for deployment on mobile devices. Hence, there have been attempts to design pairing-free proxy-signcryption schemes, such as the certificateless proxy-signcryption (CLPSC) schemes of Liu et al. [14] and Qi et al. [15]. The design of such schemes is challenging. For example, Liu et al. [14] proposed a pairing-free CLPSC scheme based on elliptic curve cryptography (ECC), with reduced computational and communication costs. This scheme is, however, vulnerable to public key replacement attack when deployed on resource-constrained devices.

More recently in 2017, Bhatia and Verma [16] proposed an efficient ECC-based pairing-free CLPSC scheme. However, we reveal in this paper that Bhatia and Verma’s scheme also cannot resist the public key replacement attack. Specifically, we demonstrate that it is vulnerable to a public key replacement attack by a Type 1 adversary. Then, we propose an improved protocol to mitigate the security weakness. We also demonstrate the security of the improved protocol in the random oracle model and compare with other related schemes in terms of computation costs and security properties.

In the next section, we present relevant background materials. In Section 3, we reveal the vulnerability in the scheme of Bhatia and Verma. Then, we present the proposed scheme in Section 4 and analyze its security in Section 5. A comparative analysis with existing schemes is presented in Section 6. Finally, we conclude this paper in Section 7.

2. Preliminaries

2.1. Syntax Definition of CLPSC Scheme. In general, the CLPSC scheme comprises three different entities: an original signcrypter (OS), a receiver (R), and a proxy signcrypter (PS). An OS (e.g., a patient) delegates to PS (e.g., a trusted individual such as a family member or neighbor) the authority to signcrypt a message [17, 18]. During proxy signcryption, OS sends his/her signing authority to PS with a delegation warrant, which consists of the identities of the delegator, a message space, and the validity time of the delegation. The warrant requires OS’s signature and PS’s public key. PS will generate a signcrypted ciphertext with its signature and send a signcrypted ciphertext to R. Upon receiving the signcrypted ciphertext, R (e.g., the pharmacist) unsigncrypts it and checks whether the proxy signature is valid. If it is valid, then PS is authorized to perform tasks such as collect OS’ medication; otherwise, PS’ request is denied. The CLPSC scheme contains the following polynomial algorithms:

(i) **Setup:** This algorithm invoked by a Key Generation Center (KGC). It takes security parameter \( k \) as an input and runs setup algorithm to obtain system parameter \( \psi \) and the master key \( s \).

(ii) **Extract-Partial-Private-Key:** KGC takes system parameters \( \psi \), master key \( s \), and a user ID as inputs and outputs the partial private key \( d_{ID} \) of the user.

(iii) **Set-Secret:** This algorithm takes security parameter \( k \) and system parameters \( \psi \) as inputs and outputs the secret value \( s_{ID} \).

(iv) **Set-Private-Key:** It inputs system parameters \( \psi \), a user’s secret value \( s_{ID} \), and a user’s partial private key \( d_{ID} \) and outputs the public key \( SK_{ID} = (s_{ID}, d_{ID}) \).

(v) **Set-Public-Key:** It inputs system parameters \( \psi \) and a user’s secret value \( x_{ID} \) and outputs the public key \( PK_{ID} = (P_{ID}, K_{ID}) \).

(vi) **Gen-Delegation:** It inputs system parameter \( \psi \), a warrant \( M_{w} \), an ID, and public/private key of the original signer and then outputs a partial proxy key.

(vii) **Verify-Delegation:** It inputs system parameter \( \psi \), a warrant \( M_{w} \), a partial proxy key, an original signer’s identity and public key, and a proxy signer’s partial key and outputs a proxy key.

(ix) **Proxy-Signcryption:** It inputs the system parameter \( \psi \), a delegation warrant \( M_{w} \), a message \( m \), an ID and public key of an original signcrypter (OS), an ID and public key of a proxy signcrypter (PS), and a proxy key and outputs a proxy signcrypted ciphertext \( \sigma \).

(x) **Proxy-Unsigncryption:** It inputs the system parameter \( \psi \), a message \( M \), a warrant \( M_{w} \), an original signcrypter (OS)’s identity and public key, and a proxy signcrypter (PS)’s identity and public key. If the signature is verified to be correct, then it returns \( 1 \); otherwise, it returns \( 0 \).

2.2. Formal Security Model for CLPSC Scheme

2.2.1. Adversaries. In this section, we discuss two kinds of adversaries in the CLPSC schemes, as well as the types of oracle queries the adversaries have access to.

Type I adversary \( A_1 \) is a dishonest user who has the ability to replace public key, but \( A_1 \) is not capable of obtaining the system master key. Type II adversary \( A_2 \) is a malicious-but-passive KGC. This adversary can access the master key and generates the partial private key of users, but it is not able to replace the public key. Now, we describe eight oracle queries that can be accessed by both adversaries:

(i) **Create-User-Oracle:** This oracle inputs a users’ identity ID. If the ID exists, then \( PK_{ID} \) of the corresponding ID is returned. Otherwise, it generates the private key \( SK_{ID} = (d_{ID}, s_{ID}) \) and the public key
PK_{ID} = (P_{ID}, K_{ID}), adds (ID, SK_{ID}, PK_{ID}) to the list L, and returns PK_{ID}.

(ii) **Reveal-Partial-Private-Key-Oracle:** This oracle looks for list L of an input users’ ID. If the ID exists and then returns the corresponding d_{ID}. Otherwise, it return null.

(iii) **Reveal-Secret-Key-Oracle:** This oracle looks for list L of an input users’ ID. If the ID exists, then returns the corresponding s_{ID}. Otherwise, returns null.

(iv) **Replace-Public-Key-Oracle:** This oracle can pick a random value instead of the users’ public key. Upon receiving the target ID, the oracle replaces a corresponding public key in the list L.

(v) **Generate-Delegation-Oracle:** Upon receiving the system parameter ψ, an original signs private key SK_{ID} = (d_{ID}, s_{ID}), and a warrant M_{W}, this oracle can generate a delegation and send to the proxy signcryption at a later stage.

(vi) **Proxy-Key-Oracle:** This oracle takes an original signers identity ID_{O}, a proxy signers identity ID_{P}, and a warrant M_{W} as inputs and outputs the proxy key and sends it to the proxy signer.

(vii) **Proxy-Signcryption-Oracle:** This oracle takes a message M, a warrant M_{W}, an original signs identity ID_{O}, and a proxy signers identity ID_{P} as inputs and generates a proxy signature σ_{ID_{P}} as an output.

(viii) **Proxy-Unsigncryption-Oracle:** This oracle takes the system parameter ψ, the delegation warrant M_{W}, the signcrypted message δ, the public keys PK_{ID}, and ID of the original user and the proxy signer, and the private key SK_{ID} of the receiver as inputs and checks if the delegation warrant M_{W} is valid. If the verification is true, then it unsigncrypts signcrypted message δ and returns a plaintext m. Otherwise it returns error.

2.2.2. Security Notions

(i) **Confidentiality**

(1) **Definition:** A certificateless signcryption scheme has ciphertext indistinguishability (IND-CLSC-CCA2) for adaptive selective ciphertext attacks, only if no attacker has no unfair advantage in winning the following games 1 and 2 in polynomial bounded time.

(2) **Game 1 IND-CCA:** This game captures the confidentiality requirement, based on the indistinguishability of encryptions under adaptively chosen ciphertext attacks against A_1.

(3) **Initialization:** Upon receiving an input k, the setup algorithm is executed to get system parameters ψ and the master key s, then sends system parameters ψ to A_1, and keeps the system master key s secretly.

(4) **Phase I:** A_1 can ask for a polynomial bounded number of challenger queries from oracles.

(ii) **Unforgeability**

(1) **Definition 2:** The CLPSC scheme is EUF-CMA secure only if no attacker has an unfair advantage in winning the following games 3 and 4 in the polynomial bounded time.

(2) **Game 3 EUF-CMA:** In this game, the adversary needs to successfully fabricate a valid ciphertext without any delegation warrant.

(3) **Initialization:** Upon receiving an input k, the setup algorithm is executed to generate the system parameter ψ, the system master key s, and then sends system parameters ψ to A_1 but keeps the system master key s secretly.

(4) **Queries:** A_1 can ask for a polynomial bounded number of challenger queries from oracles.

(5) **Forgery:** At last, $A_1$ outputs a signcryption $\delta^*$ on message $M$ under $ID_{O^{'}}$, $ID_O$. If $\delta^*$ is a valid ciphertext in Proxy-Signcrypt-Oracle and then $A_1$ succeeds in this game. However, $A_1$ is not permitted to query the Reveal-Partial-Private-Key oracle, the Replace-Public-Key oracle, or the Reveal-Secret-Key oracle of the original user in the game.

(6) **Game 4 EUF-CMA:** In the game, the challenger interacts with $A_2$ as follows:

(7) **Initialization:** Upon receiving an input $k$, the setup algorithm is executed to get system parameters $\psi$ and a system master key $s$ which are sent to $A_2$ later.

(8) **Queries:** $A_2$ may make adaptively a polynomial bounded number of queries to oracles like Create-User, Reveal-Secret-Key, Generate-Delagation, Reveal-Proxy-Key, Proxy-Signcrypt, and Proxy-Unsignedcrypt through the challenger.

(9) **Forgery:** At last, $A_2$ outputs a signcryption $\delta^*$ on message $M$ under $ID_{O^{'}}$, $ID_O$. If $\delta^*$ is a valid ciphertext in Proxy-Signcrypt-Oracle, then $A_1$ succeeds in this game. It is mandatory that $A_1$ has not queried Reveal-Secret-Key oracle during the game.

### 3. Review and Analysis of Bhatia and Verma’s CLPSC Scheme

#### 3.1. Review of Bhatia and Verma’s CLPSC Scheme

In this section, we review the scheme of Bhatia and Verma, which consists of the following 10 polynomial time algorithms.

**Setup.** After the key generation center (KGC, who has the responsibility for system keys and the partial private keys of users) has chosen a security parameter $k$, the algorithm performs the following steps:

1. Chooses an elliptic curve $E/E_p$ over prime finite field $F_p$
2. Chooses a cyclic subgroup $G$ of the elliptic curve group, sets $p$ as a generator of order $q$
3. Chooses a master secret key $s \in Z_q^*$ and generates $P_{pub} = sP$ as a master public key
4. Lets the message space be $\{0, 1\}^l$ and selects four different hash functions $H_1, H_2, H_3, H_4$:

$$H_1 : \{0, 1\}^l \times G \rightarrow Z_q^*$$

$$H_2 : \{0, 1\}^l \times G \times G \times G \times G \rightarrow Z_q^*$$

$$H_3 : G \rightarrow \{0, 1\}^l$$

$$H_4 : \{0, 1\}^* \rightarrow Z_q^*$$

(1)

5. At last, outputs the system parameters $\psi = \{E, F_p, G, P, H_1, H_2, H_3, H_4, P_{pub}\}$.

**Extract-Partial-Private-Key.** Taking the system parameters $\psi$, system private key $s$, and $ID_O$ as inputs, KGC can calculate the partial private key $d_O = k_O + h_OS$ of user $O$, where $k_O \in Z_q^*$ is randomly chosen. Then, $K_O = k_OP, h_O = H_1(ID_O, K_O), (d_O, K_O)$ are sent to the user in a secure communication channel.

**Set-Secret.** Upon receiving $(d_O, K_O)$, the user $O$ verifies whether the parameters come from a legitimate KGC by computing $d_O^2 = K_O + h_0P_{pub}$. After successful verification, $O$ picks a random $s_O$ as its secret value and computes $P_O = s_OP$.

**Set-Private-Key.** Given system parameters, partial private key $d_O$, and secret value $s_O$ as inputs, this algorithm outputs a private key pair $SK_O = (s_O, d_O)$.

**Set-Public-Key.** Given system parameters $\psi$, $P_{pub}, K_O$ as inputs, this algorithm outputs a public key pair $PK_O = (P_O, K_O)$.

**Gen-Delegation.** Having the inputs, the original signcypeters private key pair $SK_O = (s_O, d_O)$, public key pair $PK_O = (P_O, K_O)$, and message warrant $M_W$, this algorithm generates the delegation $D_{O\rightarrow p}$ on $M_W$. Then, the user $O$ randomly chooses $a \in Z_q^*$, computes $K = aP$, and further computes $t$ as follows:

$$t = (a + h_1(d_O + s_O)),$$  (2)

where $h_1 = H_2(M_W \parallel ID_P, K, K_O, P_O, P_{pub})$. The delegation $D_{O\rightarrow p} = (ID_O, ID_P, M_W, K, K_O, P_O, t)$ is sent to the proxy signcypetor (PS) later.

**Verify-Delegation.** The PS verifies whether the delegation is legitimate by computing

$$h_1 = H_2(M_W \parallel ID_P, K, K_O, P_O, P_{pub})$$

and checks whether

$$tp = h_1(K_O + h_0P_{pub} + P_O)$$  (3)

If not, the proxy signcypeter rejects the delegation request.

**Gen-Proxy-Key.** Upon successful verification, PS computes a proxy signing key

$$D_p = (t + h_2(d_p + s_p)),$$  (4)

where $h_2 = H_2(M_W \parallel ID_O, K, K_p, P_p, P_{pub})$. 
Proxy-Signcryption. Given proxy key $D_p$, message M, and public key of the receiver $(P_R, K_R)$ as inputs, it generates a signcrypted ciphertext on OS’s behalf. Specifically, PS randomly chooses $u \in Z_q^*$ and further calculates $U = uP, V = uP_R, h = H_2(V), h_5 = H_4(M \| ID_O \| ID_R \| ID_p)$. The detail processes of generating proxy signcryption on message M are described as follows:

$$z = M \oplus h$$

$$S = (u + h_5D_p)$$

After that, the signcrypted ciphertext $(ID_O, K_O, P_O, M_W, ID_p, P_R, P_P, U, K, z, S)$ is sent to R by PS.

Proxy-Unsignedcryption. After receiving a complete signcrypted ciphertext $(ID_O, K_O, P_O, M_W, ID_p, P_R, P_P, U, K, z, S)$, the receiver R unsigncrypts $V = z_2U, M = z \oplus H_3(V)$ and

$$SP = h_3[K + h_1(K_O + P_O + h_5P_{pub}) + h_2(K_P + P_p + h_5P_{pub})] + U$$

If the above equation holds, then R accepts the message.

3.2. Analysis of Bhatia and Verma’s CLPSC Scheme. We will now present a successful public key replacement attack by a Type I adversary $A_1$ against the scheme.

Step 1. $A_1$ chooses three random numbers $a^* \in Z_q^*, z_1 \in Z_q^*, z_2 \in Z_q^*$ and computes

$$K' = a^*P$$

$$K'_O = z_1P$$

$$P'_O = z_2P - h'_5P_{pub}$$

$$h'_O = H_1(ID_O, K'_O)$$

Then, it generates a forged public key pair $K'_O, P'_O$ and substitutes the original public key of OS.

Step 2. Given a message warrant $M'_W$, which can be intercepted from the communication channel between OS and PS, OS computes $t' = (a^* + h'_1(z_1 + z_2))$, where $h'_1 = H_2(M_W \| ID_p, K', K'_O, P'_O, P_{pub})$. Then, it sends the delegation $D_{(O \rightarrow P)} = (ID_O, ID_p, M'_W, K', K'_O, P'_O, t')$ to the proxy signcryption (PS).

Step 3. After PS receives $D_{(O \rightarrow P)}$, it computes $h'_1 = H_2(M'_W \| ID_p, K', K'_O, P'_O, P_{pub})$ and $h'_O = H_1(ID_O, K'_O)$ to verify the delegation. Then, it checks whether $t'P = K'_O + h'_5P_{pub} + P'_O + K'$. Note that

$$t'P = (a^* + h'_1(z_1 + z_2))P$$

$$t'P = a^*P + h'_1z_1P + h'_1z_2P$$

According (10), (11), and (12), we compute

$$t'P = K'_O + h'_5P_{pub} + P'_O$$

$$t'P = K'_O + h'_5P_{pub}$$

By using the above $K'_O, K'_O, P'_O, t'$ from $A_1$ to the proxy signcryption, the verification is successful. In other words, the delegation $D_{(O \rightarrow P)}$ in the scheme of Bhatia and Verma can be forged.

Given the linear relationship between $d_O$ and $s_O$ in the equation, the adversary can use a fake public key to bypass the process of verify-delegation. Specifically, the adversary forges the fake secret key and computes the fake public key, because there is no equation to verify or bind the public key $(K_O, P_O)$ in the verification process. Therefore, in our improved CLPSC scheme, we construct a hash function $h_i$ that contains $P_O$ as the coefficient of $s_O$. If the adversary executes the public key replacement attack, then the adversary will need to randomly choose $s'_O \in Z_q^*$ and the coefficient $h'_i$ will be changed too. This prevents the forgery of $P'_O$.

4. Proposed CLPSC Scheme

Here, we present our proposed scheme that consists of the following three basic components: prescriber (e.g., a medical practitioner), transaction hub, and pharmacy that has implemented the electronic prescription system. Patient’s medical information (e.g., patient’s medical record, medication history) is stored in the database. The prescriber can find this information by searching on the database using the patient’s unique information, such as names, dates of birth, and current addresses. Once the record is found, the doctor can update or upload a new prescription recording new medical information to the server after reviewing it (see Figure 1). The transaction hub works like a database for recording the patient file or all prescriptions. After downloading the patients prescription and successfully executing the proxy-unsignedcryption, the pharmacy will dispense the medication listed in the electronic prescription to the proxy signcryption.

Now, we describe how to send the proxy delegation from the original signcryption OS to the proxy signcryption PS securely.

(i) Setup: KGC picks a security parameter $s \in Z_q^*$ as an input of this algorithm. After running this algorithm, system parameters $\psi = E, F, p, G, P, H_1, H_2, H_3, H_4, P_{pub}$ will be published, where $E/P$ is an elliptic curve chosen by KGC over prime finite field $p$. G is a cyclic subgroup of the elliptic curve group, $P$ is a generator of $G$, $q$ is the order of $G$, and $P_{pub} = sP$ can be easily computed. In addition, there are four cryptographic collision resistant hash functions as follows:

$$H_1 : \{0, 1\}^* \times G \rightarrow Z_q^*$$

$$H_2 : \{0, 1\}^* \times G \times G \times G \rightarrow Z_q^*$$
Let $H_1: G \rightarrow \{0, 1\}^l$ and $H_4: \{0,1\}^* \rightarrow Z_q^*$.

The message space is $\{0,1\}^l$.

(ii) **Extract-Partial-Private-Key:** This algorithm takes system parameters $\psi$, system private key $s$, and user $O$'s identity $ID_O$ as inputs and computes $d_O = k_O + h_O s$ to obtain the extract-partial-private-key of entity $O$, where $k_O \in Z_q^*$ is randomly chosen. $K_O = k_O P, h_O = H_1(ID_O, K_O)$ is computed, and the results $(d_O, K_O)$ are sent to entity $O$ via a secure communication channel (this algorithm is run by the KGC).

(iii) **Set-Secret:** This algorithm takes system parameters $\psi$, user $O$'s identity $ID_O$ as inputs and runs to verify whether $d_O P = K_O + h_O P_{pub}$. If the verification passes, then entity $O$ picks a random $s_O \in Z_q^*$ as its secret value and computes its partial public key $P_O = s_O P$; otherwise, the process fails (his algorithm is run by users).

(iv) **Set-Private-Key:** This algorithm takes system parameters $\psi$, partial private key $d_O$, and secret value $s_O$ as inputs and generates a private key pair $SK_O = (s_O, d_O)$ as the output (this algorithm is run by users).

(v) **Set-Public-Key:** Given system parameters $\psi, P_O$, and $K_O$ as inputs, this algorithm generates a public key pair $PK_O = (P_O, K_O)$ as the output. When this algorithm completes executing, user publishes the public key $PK_O$ (this algorithm is run by users).

(vi) **Gen-Delegation:** Given the original signcryption private key pair $SK_O = (s_O, d_O)$, public key pair $PK_O = (P_O, K_O)$, and message warrant $M_W$ as input, this algorithm generates the delegation $D_{O \rightarrow p}$ on $M_W$ as the output. Then, the entity $O$ randomly chooses $a \in Z_q^*$, computes $K = a P$ and computes $t$ as follows: $t = (a + h_1(d_O + s_O h_3)), $ where $h_1 = H_2(M_W \parallel ID_P, K, K_O, P_O, P_{pub}), h_1 = H_1(ID_O, P_O)$. The delegation $D_{O \rightarrow p} = ID_O, ID_P, M_W, K_O, P_O, t$ is finally sent to PS.

(vii) **Verify-Delegation:** PS verifies whether the delegation is valid by computing

$$h_1 = H_2\left(M_W \parallel ID_P, K, K_O, P_O, P_{pub}\right)$$

$$h_O = H_1\left(ID_O, K_O\right)$$

$$h_t = H_1\left(ID_O, P_O\right)$$

and checks whether

$$tp = h_1\left(K_O + h_O P_{pub} + h_t P_O\right) + K$$

If the following does not hold, then it implies that the delegation is invalid.

(viii) **Gen-Proxy-Key:** Upon successful verification, PS generates a proxy signing key $D_p = (t + h_2(d_P + s_P))$, where $h_2 = H_2(M_W \parallel ID_O, K, P_P, K_P, P_{pub})$.

(ix) **Proxy-Signcryption:** Given the proxy key $D_p$, an original message $M$, and the receiver's public key $PK_R$, $K_R$ as inputs, a signcrypted message is generated for $R$. The following describe the details: PS randomly chooses $u \in Z_q^*$, computes $U = u P, V = u P_R, h = H_3(V), h_s = H_4(M \parallel ID_O \parallel ID_R \parallel ID_R)$, and the proxy signcryption of message $M$ is generated.

Figure 1: Improved CLPSc scheme for electronic prescription system.
as follows: \( z = M \oplus h, S = (u + h_3D_p) \). After that, the signcrypted ciphertext \((ID_C, K_C, P_O, M_W, ID_P, K_p, P_p, U, K, z, S)\) is sent to the receiver R by PS.

**Proxy-Unsigncryption:** Upon receiving the signcrypted ciphertext \((ID_C, K_C, P_O, M_W, ID_P, K_p, P_p, U, K, z, S)\), R calculates \( V = s_P U, M = z \oplus H_2(V) \) and only if the following holds, will R accept the message:

\[
SP = h_3 \left[ K + h_1 \left( K_O + P_O + h_3P_{pub} \right) \right] + h_2 \left( K_p + P_p + h_Pp_{pub} \right) + U \tag{18}
\]

### 4.1. Correctness Analysis

The CLPSC scheme contains two parts of authentication, namely, verification of delegation (i.e., proxy signcrypter PS checks whether \( tp = h_1(K_O + h_3P_{pub} + h_3P_{pub} + K) \)) and proxy-unsigncryption (i.e., message receiver checks whether \( SP = h_3[K + h_1(K_O + h_3P_{pub} + h_3P_{pub}) + h_2(K_p + P_p + h_Pp_{pub})] + U \)).

(i) **Decryption process**

\[
V' = s_R U' = s_RuP = uP_R = V \tag{19}
\]

(ii) **Verification process**

One part is

\[
t = (a + h_1(d_O + s_cj_1))
\]

\[
tP = (a + h_1(d_O + s_cj_1)) P
= (aP + h_1(d_OP + s_cj_1P))
= (K + h_1((K_O + h_3O) + h_3P_O))
= (K + h_1((K_O + h_3O + h_3P + h_3P_O))
= (K + h_1((K_O + h_3P_{pub} + h_3P_O))
\]

Another part is

\[
SP = (u + h_3D_p) P
SP = uP + h_3 \left( (d + s_P) \right) P
= (uP + h_3(d_O + s_cj_1)) P
= (uP + h_3(d_OP + s_cj_1P)) + U
\]

\[
SP = h_3 \left( tp + h_2 \left( d_O + s_cj_1 \right) \right) + U
\]

\[
SP = h_3 \left( (uP + h_3(d_O + s_cj_1)) P + h_3P_{pub} \right)
+ h_2 \left( K_p + P_p + h_Pp_{pub} \right) + U
\]

### 5. Security Analysis

As defined in Section 2.2, there are two types of adversaries: \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \). We consider four games where \( \mathcal{A}_1 \) and \( \mathcal{A}_2 \) can get honest answer when they interact with the challenger.

5.1. **Confidentiality.** We define the fact that the certificateless signcryption scheme is IND-CLSC-CCA2-secure to adaptive chosen ciphertext attacks, only if no attacker with a nonnegligible advantage can win the games in polynomial time.

**Game 1.** In this game, we assume that \( \mathcal{A}_1 \) is a dishonest user.

**Lemma 1.** Assume that \( \mathcal{A}_1 \) can break the proposed CLPSC scheme with \( v' \). Let \( q_{H_1}, q_{C_s}, q_{P_p}, q_{S_1}, q_{R_{k_1}}, q_{P_{k_1}}, q_{S_1}, q_{R_{k_2}}, q_{C_s}, \) and \( q_{U_{k_3}} \) denote the number of \( H_1 \)-queries, Create-User-queries, Reveal-Partial-Private-Key-queries, Reveal-Secret-Key-queries, Replace-PublicKey-queries, Generate-Delegation-queries, Reveal-Poxy-Proxy-Key-queries, Proxy-Signcrypt-queries, and Proxy-Unsigncryption-queries, respectively. There is an algorithm that can solve the ECCDH problem with advantage \( v' \) in probabilistic polynomial time:

\[
v' \geq \frac{1}{q_{H_1}} \left( 2u - q_{C_s} \left( \frac{1}{2} \right) \right) \tag{22}
\]

\[
t' \leq t + O (1) \left( q_{H_1}, q_{H_2}, q_{H_3}, q_{C_s}, q_{P_p}, q_{S_1}, q_{R_{k_1}}, q_{C_s}, \right)
\]

where \( t_M \) is the time for an ECC-based scalar point multiplication operation.

**Proof.** Suppose that an algorithm accepts an ECCDH instance \((P, xP, yP) \in G_2 \) with unknown \( x \) and \( y \); the problem is to compute \( xyP \). In order to solve this problem, the algorithm uses \( A_1 \) as a subroutine and acts as a challenger \( C \) to interact with it in IND-CCA2-I. When the game starts, the challenger \( C \) creates and maintains \( L_{H_1}, L_{H_1}, L_{H_1}, L_{P_p}, L_{S_1}, \) lists, which stores the responses to the queries by the adversary \( A_1 \). The lists are initially set to be empty. When \( A_1 \) makes a query, \( C \) will respond as follows:

(i) **H₁-Query:** \( C \) maintains a hash list \( L_{H_1} = \langle (ID, K_{ID}) \rangle, z_1 > \) as explained below. When \( A_1 \) asks for the \( H_1 \)-query-Oracle and if the query ID is found in \( L_{H_1} \), then the algorithm returns the record to \( A_1 \). Otherwise, \( C \) chooses a random number \( z_1 \in Z_q^* \) and adds it to \( L_{H_1} \).

(ii) **H₂-Query:** \( C \) maintains a hash list \( L_{H_2} = \langle (M_W || ID_p, K, K_{ID}, P_{pub}, P_{pub}), z_2 \rangle > \) as explained below. When \( A_1 \) asks for the \( H_2 \)-query-Oracle and if the tuple is found in \( L_{H_2} \), then the algorithm returns \( z_2 \) to \( A_1 \). Otherwise, \( C \) chooses a random number \( z_2 \in Z_q^* \) and adds \( \langle (M_W || ID_p, K, K_{ID}, P_{pub}, P_{pub}), z_2 \rangle > \) to \( L_{H_2} \).

(iii) **H₃-Query:** \( C \) maintains a hash list \( L_{H_3} = \langle (V, z_3) > \) as explained below. When \( A_1 \) asks for the \( H_3 \)-query-Oracle and if the tuple \( < V, z_3 > \) is found in \( L_{H_3} \), then the algorithm returns \( z_3 \) to \( A_1 \). Otherwise, \( C \) chooses a random number \( z_3 \in \{0, 1\} \) and adds it to \( L_{H_3} \).

(iv) **H₄-Query:** \( C \) maintains a hash list \( L_{H_4} = \langle (M || ID_O, || ID_P, || ID_O), z_4 \rangle > \) as explained below. When \( A_1 \) asks for the \( H_4 \)-query-Oracle and if the tuple is
found in \(L_{H_t}\), then the algorithm returns \(z_4\) to \(A_1\). Otherwise, \(\mathcal{C}\) chooses a random number \(z_4 \in Z_q^*\) and adds it to \(L_{H_t}\).

(v) **Create-User query:** \(\mathcal{C}\) maintains a list \(L_c \leftarrow< ID, K_{ID}P_{ID}, s_{ID}, d_{ID} >\). If the query \(ID\) is found on the \(L_c\), then it returns < \(ID, K_{ID}P_{ID}, s_{ID}, d_{ID} >\). Otherwise, the oracle is simulated as follows:

1. \(\mathcal{C}\) chooses three numbers \(s_O, k_O, z_1 \in Z_q^*\) randomly
2. computing \(P_O = s_O P, K_O = k_O P, d_O = k_O + z_1 s\)
3. adding the tuple \((ID_O, K_O, s_O, d_O)\) to list \(L_C\) and \((ID_O, K_O, z_1)\) to \(L_{H_t}\), respectively.

(vi) **Extract-Partial-Private-Key-Query:** When \(A_1\) queries \(d_{ID}\) and if \(ID = ID^*\), then the simulation is stopped. Otherwise, \(\mathcal{C}\) probes the list \(L_c\) for the query \(ID\). If it exists on \(L_c\), then it returns the corresponding \(d_{ID}\). Otherwise, \(\mathcal{C}\) performs the Create-User query, gets < \(ID, K_{ID}P_{ID}, s_{ID}, d_{ID} >\), and sends its partial private key \(d_{ID}\) to \(A_1\).

(vii) **Reveal-Secret-Key-Query:** If \(ID = ID^*\), then the simulation is stopped. Otherwise, \(\mathcal{C}\) probes the list \(L_c\) for the query \(ID\). If it exists on \(L_c\), then it returns the corresponding \(s_{ID}\). Otherwise, \(\mathcal{C}\) performs the Create-User query and sends its partial private key \(s_{ID}\) to \(A_1\).

(viii) **Replace-Public-Key-Query:** \(A_1\) can replace the public key \(PK_{ID} = (P_{ID}, K_{ID})\) with a random value \(PK'_{ID}\).

(ix) **Generate-Delegation-Query:** When \(A_1\) asks the Generate-Delegation query on \((ID_O, ID_P, M_W)\), \(\mathcal{C}\) starts to run the Gen-Delegation algorithm and sends the results \(D_O\leftarrow P = ID_O, ID_P, M_W, K_O, P_O, a, t\) to \(A_1\).

(x) **Reveal-Proxy-Key-Query:** \(\mathcal{C}\) maintains a list of issued proxy keys \(L_P = ID_O, ID_P, M_W, a, K_t, t, D_P\). If the query \((ID_O, ID_P, M_W)\) exists on the \(L_P\), then it returns a proxy key \(D_P\). Otherwise, \(\mathcal{C}\) works as follows:

1. performing Generate-Delegation-Query to obtain \(D_{O\leftarrow P} = ID_O, ID_P, M_W, K_O, P_O, a, t\).
2. querying the list \(L_P\) with \(ID_P\) for the corresponding secret key \((s_P, d_P)\).
3. computing \(D_P = (t + z_1(s_P + s_P))\) and adds the tuple \((ID_O, ID_P, M_W, K, a, t, D_P)\) to list \(L_P\).
4. Finally, it sends \(D_P\) to \(A_1\).

(xi) **Proxy-Signcrypt-Query:** When \(A_1\) makes a query for signcrypting a message \(M\) with the message warrant \(M_w\) and three input identities \((ID_O, ID_P, ID_P)\) and if the query entry \((ID_O, ID_P, M_W)\) exists on list \(L_{P}\), then the corresponding proxy key \(D_P\) and \(K\) on the tuple can be used by \(\mathcal{C}\) to create a signcrypted message \(\delta\). Otherwise, \(\mathcal{C}\) performs the aforementioned Reveal-Proxy-Key query to obtain the proxy key \(D_P\). To generate a signcrypted message on behalf of an original signer, \(\mathcal{C}\) checks whether \(ID_O\) and \(ID_P\) are correct. If not, the proxy-signcryption algorithm is run until the secret key \((s_O, d_O)\) of user \(O\) is obtained. Then, the tuple \((ID_O, PK_O, M_W, ID_P, PK_P, M, \delta)\) is added to \(L_S\) as the result. Otherwise, \(\mathcal{C}\) performs the following:

1. choosing random numbers \(x, z_4 \in Z_q^*\) and computing \(U = xP, V = xyP\).
2. probing \(L_{H_t}\) for \(V\) and returning \(z_3\) as a result.
3. computing \(z = Mz_4S = (x + z_4D_P)\) and adding a tuple \((ID_O \parallel ID_R \parallel ID_P, M_W)\) to list \(L_{H_t}\), where \(L_{H_t} \parallel ID_O \parallel ID_R \parallel ID_P = z_4\)

(xii) **Proxy-Unsigncrypt-Query:** Taking the signcrypted message \(\delta = (ID_O, P_O, K_O, M_W, ID_P, P_P, K_P, U, K, z, S)\) as inputs, \(\mathcal{C}\) checks if \(ID_R\) is a challenging identity. If it is true, \(\mathcal{C}\) probes \(L_{P_o}\) for a tuple \((ID_O, PK_O, ID_R, PK_P, *, *, \delta)\) and obtains the corresponding plain-text message \(M\). Otherwise, \(\mathcal{C}\) performs as follows:

1. taking the secret key of the receiver as an input and calculating \(V = s_PU\).
2. probing \(L_{H_t}\) for \(V\) and returning \(z_3\) to calculate \(M = z \oplus z_3\).
3. probing the list \(L_{H_t}\) for \((ID_O, K_O)\) and \((ID_P, K_P)\), \(L_{H_t}\) for \((ID_P \parallel M_W, K_{ID}, K_{ID}, P_{ID}, P_{ID})\) and \((ID_P \parallel M_W, K_{ID}, K_{ID}, P_{ID}, P_{ID})\) and \(L_{H_t}\) for \((ID_O \parallel ID_R \parallel ID_P \parallel M_W, K_{ID}, K_{ID}, P_{ID}, P_{ID})\) to obtain \(z\) and \(z_4\) respectively. If \(SP = z_4(K + z_3(K_O + h_P_0 + h_P_1P_{ID}) + z_2(K + P + z_2P_{ID}) + U\), valid, \(\mathcal{C}\) returns \(M\). Otherwise, it throws an error message.

Challenge. \(A_1\) submits three distinct identities: \(ID_O\) of original signcryption, \(ID_P\) of proxy signcryption, and \(ID_R\) of receiver and two equal length messages \(M_0\) and \(M_1\). A random number \(b \in \{0, 1\}\) is chosen by \(\mathcal{C}\). What is more, \(\mathcal{C}\) signcrypts \(M_b\) to produce a corresponding signcrypted ciphertext \(\delta^*\) to \(A_1\). \(\mathcal{C}\) further probes \(L_{P_o}\) for \(z_3\) to obtain \(z_3\), calculates \(z^* = M_b \oplus z_3^*, S^* = x + z_4D_P^*\), and adds \((M_b \parallel ID_O^* \parallel ID_R^* \parallel ID_P^* \parallel ID_{P^*} \parallel ID_{P_{ID}}^*, z^*, S^*)\) for \(M_b\) is returned by \(\mathcal{C}\) to \(A_1\). \(\mathcal{C}\) can continue to ask queries with the exception of the Proxy-Unsigncrypt query on \(\delta^*\), Reveal-Partial-Private-Key-query, and Reveal-Secret-Key-query in the game.

Output. At last, \(A_2\) outputs \(b'\) as the guess of value \(b\). If \(b' = b\), then \(\mathcal{C}\) outputs \(V = s_PU = s_PuP\) as the solution of the ECCDH question; otherwise, the challenge fails.

Game 2. Let \(A_2\) be a malicious-but-passive KGC.

**Lemma 2.** Assume that \(A_2\) has the ability of breaking the proposed CLPSC scheme with an advantage of \(\epsilon\). Let \(q_{H_t}\).
\( q_{C_s}, q_{P_{\text{pub}}}, q_{S_k}, q_{G_{\text{t}}}, q_{p_{\text{k}}}, q_{S_{\text{a}}}, \) and \( q_{U_{\text{v}}} \) denote the number of \( H_1 \)-queries, Create-User-queries, Reveal-Partial-Private-Key-queries, Reveal-Secret-Key-queries, Generate-Delegation-queries, Reveal-Proxy-Key-queries, Proxy-Signcrypt-queries, and Proxy-Signcrypt-queries, respectively. In probabilistic polynomial time, there is an algorithm which can solve ECCDH problem with advantage \( v' \):

\[
v' \geq \frac{1}{q_{H_1}} \left( 2v - q_{U_{\text{v}}} \left( \frac{1}{2} \right)^t \right)
\]

\[
t' \leq t + O \left( 1 \right) \left( q_{H_1} + q_{H_2} + q_{H_1} + q_{C_s} + q_{p_{\text{k}}} + q_{S_{\text{a}}} + q_{U_{\text{v}}} \right)
\]

\[
+ t \left( 2q_{C_s} + 2q_{S_{\text{a}}} \right)
\]  

where \( t_M \) is the time for an ECC-based scalar point multiplication operation.

**Proof.** The proof for this game is similar to that of Game 1, and hence we will not repeat the proof.

### 5.2. Unforgeability

We defined that the certificateless signcryption scheme is EUF-CMA-secure to adaptive chosen ciphertext attacks, only if no attacker with a nonnegligible advantage can win the following games in polynomial time.

**Game 3.** Assume that \( \mathcal{A}_1 \) is a dishonest user.

**Lemma 3.** Assume that \( \mathcal{A}_1 \) can break the proposed CLPSC scheme with \( v' \). Let \( q_{H_1}, q_{C_s}, q_{p_{\text{k}}}, q_{S_{\text{a}}}, q_{G_{\text{t}}}, q_{p_{\text{K}}}, \) and \( q_{S_{\text{a}}} \) denote the number of \( H_1 \)-queries, Create-User-queries, Reveal-Partial-Private-Key-queries, Reveal-Secret-Key-queries, Generate-Delegation-queries, Reveal-Proxy-Key-queries, and Proxy-Signcrypt-queries, respectively. There is an algorithm in probabilistic polynomial time that can solve ECCDH problem with advantage \( v' \):

\[
v' \geq \frac{1}{q_{H_1} q_{H_2}} \left( v - \left( 1 + q_{H_2} \right) \left( \frac{1}{2} \right)^t \right)
\]

\[
t' \leq t + O \left( 1 \right) \left( q_{H_1} + q_{H_2} + q_{H_1} + q_{C_s} + q_{p_{\text{k}}} + q_{S_{\text{a}}} + q_{U_{\text{v}}} \right)
\]

\[
+ t \left( 2q_{C_s} + 2q_{S_{\text{a}}} \right)
\]  

where \( t_M \) is the time for an ECC-based scalar point multiplication operation.

**Proof.** Suppose that an algorithm accepts an ECCDH instance \( (P, xP, yP) \in G_q \) with unknown \( x \) and \( y \); the problem is to compute \( xyP \). In order to solve this problem, the algorithm uses \( A_1 \) as a subroutine and acts as a challenger \( \mathcal{C} \) to interact with it. \( \mathcal{C} \) gives honest answer to the queries of \( A_1 \). When the game starts, the relevant system parameters are created and sent to \( A_1 \) by \( \mathcal{C} \). We set \( P_{\text{pub}} = xP \). Then, \( A_1 \) can make queries for information. The challenger \( \mathcal{C} \) creates and maintains \( L_{H_1}, L_{H_2}, L_{H_1}, L_{H_2}, L_{P_{\text{K}}} \) lists, which store the returns of the responses to the adversary's queries. The lists are initially set to be empty.

1. **Forgery.** At last, a signcrypted ciphertext \( C^* = (ID^*_{A_1}, K_{A_1}^*, P_A^*, M_{\text{w}^*}, ID^*_{P_{\text{K}}}, P_{\text{K}}^*, U^*, K^*, z^*, S^*) \) on message \( M \) and warrant \( M_{\text{w}} \) is produced as the output, where \( ID_{A_1}^* \) is the original signer and \( ID_{P_{\text{K}}}^* \) is the receiver. Unlike normal cases, it does not go through the ProxySigncrypt oracle. \( ID_{P_{\text{K}}}^* \) is a forge ID as a proxy signer for \( ID_{A_1}^* \). \( A_1 \) takes this signcrypted ciphertext \( C^* \) as an input of the Proxy-Unsigncrypt oracle and makes queries to \( \mathcal{C} \) except for Reveal-Partial-Private-Key-Oracle and Replace-Public-Key-Oracle or Reveal-Secret-Key-Oracle in probabilistic polynomial time. If Proxy-Unsigncrypt-Oracle is not an error, then \( A_1 \) succeeds in this game. Otherwise, \( A_1 \) fails. The solution of the problem is \( x = ((D_2 - t)/h_2) - k_{c} - s_{c})/h_1. \)

**Game 4.** Let \( \mathcal{A}_2 \) be a malicious-but-passive KGC.

**Lemma 4.** Assume that \( \mathcal{A}_2 \) can break the proposed CLPSC scheme with \( v' \). Let \( q_{H_1}, q_{C_s}, q_{S_{\text{a}}}, q_{G_{\text{t}}}, q_{p_{\text{k}}}, \) and \( q_{S_{\text{a}}} \) denote the number of \( H_1 \)-queries, Create-User-queries, Reveal-Partial-Private-Key-queries, Reveal-Secret-Key-queries, Generate-Delegation-queries, Reveal-Proxy-Key-queries, and Proxy-Signcrypt-queries, respectively. In probabilistic polynomial time, there is an algorithm that can solve ECCDH problem with advantage \( v' \):

\[
v' \geq \frac{1}{q_{H_1} q_{H_2}} \left( v - \left( 1 + q_{U_{\text{v}}} \right) \left( \frac{1}{2} \right)^t \right)
\]

\[
t' \leq t + O \left( 1 \right) \left( q_{H_1} + q_{H_2} + q_{H_1} + q_{C_s} + q_{S_{\text{a}}} + q_{G_{\text{t}}} + q_{p_{\text{k}}} + q_{S_{\text{a}}} + q_{U_{\text{v}}} \right)
\]

\[
+ t \left( 2q_{C_s} + 2q_{S_{\text{a}}} \right)
\]  

where \( t_M \) is the time for an ECC-based scalar point multiplication operation.

**Proof.** The proof for this game is similar to that of Game 3, and hence we will not repeat the proof.

### 6. Performance Evaluation

In this section, we compare the efficiency and security of our improved scheme with other proxy-signcryption schemes [15, 16] in the literature. We use the standard cryptographic library MIRACL [19] to measure the runtime, whose comparative summary is given in Table 1, where \( t_{\text{pub}} \) denotes an ECC-based point multiplication operation time and \( t_{\text{h}} \) denotes general hash operation time.

**Our evaluation environment is a personal computer (PC; Dell with an i5-4460S 2.90GHz processor, 4G bytes memory and the Window 8 operating system) with the MIRACL library [20]. The curve is \( E : y^2 + x^3 + xy = (x^3 + ax^2 + b) \) over 163 bits random prime, where \( a = 1, b = 1 \). A Koblitz curve ect163k1 over \( F_2^{163} \) is selected from the list of elliptic.
curves indicated by NIST [21]. In this setup, we have \( t_{pm} = 2.21, t_h = 0.007 \) approximately. Then, we compute the total runtime with their operation times. From Table 1, we observe that the runtime of [16] and our scheme are less than other schemes. Because our scheme is an improved scheme from [16], the runtime is very close to that of the original scheme. However, from Table 2, it is clear that our scheme is the most secure of these schemes. It is also trivial to note that the original scheme in [16] cannot resist forgery attack, unlike our scheme (at a slight cost of about 0.112 ms).

### 7. Conclusion

While signcryption is a fairly established research area, designing secure signcryption schemes remains challenging. For example, in this paper we revisited a recently proposed certificateless proxy-signcryption (CLPSC) scheme of Bhatia and Verma and revealed that the scheme is susceptible to the public key replacement attack by a Type 1 adversary. Then, we presented an improved scheme to mitigate such an attack from both Type 1 and Type 2 adversaries. We also evaluated its security and performance to demonstrate its utility in an electronic prescription system.

Future research includes implementing a prototype of the improved protocol for evaluation in a real-world environment.

### Data Availability

The data used to support the findings of this study are included within the article.

### Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of the paper.

### Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant nos. 61501333, 61572370, 61572379, and U1536204), the National High-Tech Research and Development Program of China (863 Program) (Grant no. 2015AA016004), and the open funding of the Jiangsu Key Laboratory of Big Data Security & Intelligent Processing (Grant no. BDSIP1807).

### References


