International Journal of Computer Mathematics, 2015
Vol. 92, No. 8, 1520–1535 http://dx.doi.org/10.1080/00207160.2014.957196

Taylor & Francis
Taylor & Francis Group

A provable secure pairing-free certificateless identification scheme

Ji-Jian Chin[a]*, Syh-Yuan Tan[b], Swee-Huay Heng[b], Raphael C.-W. Phan[a] and Rouzbeh Behnia[b]

[a]*Faculty of Engineering, Multimedia University, 63100 Cyberjaya, Selangor, Malaysia;* [b]*Faculty of Information Science and Technology, Multimedia University, 75450 Bukit Beruang, Melaka, Malaysia*

(*Received 8 November 2013; revised version received 8 June 2014; accepted 18 August 2014*)

Certificateless identification (CLI) schemes offer an alternative solution to the certificate management problem of traditional identification schemes, as well as remove the key escrow from key generation, an inherent property of identity-based identification. In this paper, we provide a pairing-free CLI scheme, provable secure against passive and active/concurrent attacks for both Type-1 and Type-2 adversaries. This shows that our scheme is computationally efficient because no bilinear pairings are involved.

Keywords: certificateless identification; pairing-free; provable secure; no key escrow; discrete logarithm

2010 AMS Subject Classifications: 94A60; 94A62; 14G50, 11T71; 68R99

1. Introduction

Identification schemes allow a user to prove himself to a verifier without any revelation of his secret key. Usually this is used to control allocation of access to resources to the appropriate users. In traditional cryptography, a user's public key is bound to his entity via a certificate issued by a Certificate Authority, making a scheme susceptible to key replacement attacks if a certificate is not used. However, as the number of users grew larger, certificate management becomes an issue.

Shamir proposed the notion of identity-based cryptography in [15], where a user can implicitly certify himself through the binding of a unique identity string to his user secret key. The first identity-based identification schemes were proposed by Neven and colleagues [4] and by Kurosawa and Heng [13] independently and has seen considerable advances within the latter of the last decade. However, key escrow exists for identity-based cryptographic schemes, naturally because the trusted authority who holds the master secret key generates all the user secret keys and has access to them. While desirable in some cases, key escrow can be a security issue in other scenarios, where a user wants full confidentiality of his secret key, even from the key generation authority that created it.

Certificateless cryptography, first proposed by Al-Riyami and Paterson [1], key generation is split between the key generation centre (KGC), who only creates a partial user secret key, and the user who completes the key generation process to create public and private key pairs

*Corresponding author. Email: jjchin@mmu.edu.my

using his own secret value. Certificateless cryptography has seen its own evolution in research in the last decade with many security notions defined and revisited, and with schemes proposed and broken under different security assumptions. We direct the interested reader to the comprehensive survey by Dent [9] for certificateless encryption schemes and to Huang *et al.*'s revisited notions of certificateless signature schemes in [12] for the current research progress in certificateless cryptography.

However, while much research has taken place in encryption and signature schemes for certificateless cryptography, such as can be seen by works like [11] and [10], certificateless identification (CLI) schemes have remained largely unexplored. The first CLI schemes were first proposed by Chin *et al.* [5] and Dehkordi and Alimoradi [8] independently. However, Dehkordi and Alimoradi's [8] work did not propose a proper security model for their scheme nor provide any proof and subsequently was shown to be insecure by Chin *et al.* [6]. On the other hand, Chin *et al.* [5] defined the first security model for CLI and subsequently proposed the first CLI scheme based on pairings.

In this paper, we expand on the work in the area of CLI. We utilize the notions from [5] but provide an alternative to the pairing-based CLI scheme. Our motivation for this work is to provide a CLI scheme that is fast and efficient and is both free from certificate management issues as well as key escrow. Our scheme is pairing-free, thus providing a computationally efficient scheme, since it is widely known that pairings cost a lot in terms of computing operation. We show that our scheme is secure against both passive and active/concurrent attackers of Super-Type-1 and Super-Type-2 categories, as will be defined in the next section.

We divide our paper into the following sections: in Section 2, we introduce the basic security notions of CLI schemes, including intractable mathematical assumptions used and security definitions. In Section 3, we show the construction of our scheme. In Section 3.1, we provide 4 security proofs – passive security against Type-1 and Type-2 adversaries, as well as active/concurrent security against Type-1 and Type-2 adversaries. In Section 4, we provide the efficiency analysis of the algorithms of our scheme and conclude in Section 5.

2. Preliminaries

2.1 *Problems and assumptions*

Define G as a cyclic group of order q where $q = (p-1)/2$ and p is a large prime. g is a generator for the group G. The discrete logarithm problem is defined as follows:

(1) *Discrete logarithm problem*: Given $g, Z = g^z$ for some $z \in \mathbb{Z}_q^*$, compute z.
(2) *One-more discrete logarithm problem*: The one-more version of the discrete logarithm problem was initially proposed for proving security of blind signatures by Bellare *et al.* [3]. Subsequently, Bellare and Palacio used the same problem to prove the security against active and concurrent attacks for the Schnorr identification scheme in [2]. Neven *et al.* also used the same problem in [4] and to prove security against impersonation under active and concurrent attacks for the BNN-IBI scheme in their paper. Subsequent work also utilized the one-more problems frequently to achieve active and concurrent level security for identification schemes.
The one-more discrete logarithm problem is modelled as a game played by an adversary where the adversary is given $1^k, G, q, g$ as input and access to two oracles *CHALL* and *DLOG*. *CHALL* on any input returns a random point W_i, while *DLOG* on any input h will return z, where $z = \log_g h$. The adversary is required to compute the discrete logarithms to all the target points $W_0 \ldots W_n$ while using strictly less queries to the *DLOG* oracle. In other words,

the adversary is required to find $w_0, \ldots, w_n$ where $w_0 = \log_g W_0, \ldots, w_n = \log_g W_n$ using the *DLOG* oracle only $i \leq n$ times.

The discrete logarithm assumption and the one-more discrete logarithm assumption state that there are no polynomial time algorithms for solving both the problems with non-negligible probability.

2.2 *Definition of CLI schemes*

We extend the key generation process of traditional identification and identity-based identification schemes to create the definition for CLI schemes. The CLI scheme consists of six polynomial time algorithms:

- **Setup** is run by the KGC. It takes in the security parameter 1^k as input and returns the master public key MPK and the master secret key MSK. It publishes MPK and keeps MSK to itself.
- **Partial-Private-Key-Extract** is run by the KGC upon a user's request for a partial private key. It takes in MPK, MSK and a user's identity ID, returns the partial private key $\mathsf{PPK}_{\mathsf{ID}}$ for the user. We assume this is done via a secure channel between the KGC and the user.
- **Set-User-Key** is run by the user when creating his own account. It takes in the security parameter 1^k and the user's identity ID as input, generates the secret value for a user $\mathsf{SV}_{\mathsf{ID}}$ and a corresponding user's public key $\mathsf{UPK}_{\mathsf{ID}}$.
- **Set-Private-Key** is done by the user to combine the user's identity ID, partial private key $\mathsf{PPK}_{\mathsf{ID}}$, public key $\mathsf{UPK}_{\mathsf{ID}}$ and secret value $\mathsf{SV}_{\mathsf{ID}}$ into the full private key. It returns the user private key as USK.
- **Identification-Protocol** is the interactive protocol run by the two algorithms **Prover** and **Verifier**. Both algorithms take in the master public key MPK, prover's identity string ID, public key $\mathsf{UPK}_{\mathsf{ID}}$, with the prover taking in the user private key $\mathsf{USK}_{\mathsf{ID}}$ as auxiliary input. They perform the three-step canonical honest verifier zero knowledge proof of knowledge protocol with the following steps:
  (1) **Prover** sends the COMMITMENT to the **Verifier**.
  (2) **Verifier** sends the CHALLENGE to the **Prover**.
  (3) **Prover** sends the RESPONSE to the **Verifier**, which the **Verifier** will choose to either accept or reject.

2.3 *Security notion for CLI schemes*

We consider four types of adversaries for the CLI scheme, the Type-1 passive impersonator IMP-PA-1 and the active/concurrent impersonator IMP-AA/CA-1, and the Type-2 passive impersonator IMP-PA-2 and the active/concurrent impersonator IMP-AA/CA-2. The capability between passive and active impersonator differs in that the passive impersonator can only eavesdrop on conversations between honest parties, while the active impersonator can act as a cheating verifier to gain knowledge from honest provers through interacting with them. The concurrent impersonator is an active impersonator who can run several instances of the protocol at the same time.

Type-1 impersonators model malicious third-party impersonators who do not have access to the master secret key, but is able to request and replace public keys with values of his own selection. Type-2 impersonators model the malicious KGC who can generate partial private keys of users. Based on certificateless signature schemes according to the definitions by Huang *et al.* [12], adversarial classifications can be further broken down into the Normal-, Strong-

and Super-type adversary for Type-1 and Type-2 categories, differing in what parameters they have.

- Normal-type adversaries cannot obtain signatures of users once their public keys have been replaced. In the identification setting, an adversary cannot use a prover to converse with a verifier once its public key is replaced.
- Strong-type adversaries can obtain signatures of users with their public keys replaced, provided they supply the secret value of the replaced public key. In the identification setting, an adversary can continue using a prover whose public key has been replaced, provided they supply the secret value corresponding to the replaced public key for the conversation.
- Super-type adversaries can obtain signatures of users with their public key replaced without having to supply the secret value of the replaced public key. In the identification setting, an adversary can replace a prover's public key and still use it to correspond with a verifier without the new secret value.

The levels of security in terms of adversary types are Super>Strong>Normal, that is, if an identification scheme is secure against Super-type adversaries, it means that they are secure against Strong- and Normal-type adversaries as well.

We describe the security model of CLI schemes against Type-1 and Type-2 impersonators as the following games, and highlight the differences in capabilities when making identification queries within the game for both passive and active/concurrent impersonators as well as Normal, Strong and Super categories.

Game I. The game is played between a challenger C and the Type-1 Impersonator I_1 for the CLI scheme Π as follows:

- **Setup** C runs **Setup** and passes the system parameters params to I_1. It keeps the master secret key MSK to itself.
- **Phase 1**: In this training phase, I_1 will be allowed to make the following queries adaptively to C:
  (a) **ExtrPartSK(**ID**)**. On request for the partial private key PPK on user ID, C will run **Partial-Private-Key-Extract** and returns the user's partial private key to I_1.
  (b) **ExtrFullSK(**ID**)**. On request for the full private key USK on user ID, C will run **Partial-Private-Key-Extract**, **Set-User-Key** and **Set-Private-Key** algorithms to generate the complete user's private key and passes it to I_1.
  (c) **RequestPK(**ID**)**. On request for the public key UPK on user ID, C will run **Set-User-Key** to generate the user's public key and passes it to I_1.
  (d) **ReplacePK(**ID,UPK'**)**. I_1 is able to replace the user ID's public key UPK with the public key UPK' chosen by him. Note that the corresponding secret value is not required for public key replacement queries.
  (e) **Identification(**ID**)**. For passive I_1, C will construct a valid transcript for a round of interaction between user ID and itself as the verifier and returns the transcript to I_1. For active/concurrent I_1, C will play the role of the prover to interact with I_1 as the cheating verifier:
    (i) Normal-type adversaries cannot make an identification query for a prover if its public key has been replaced.
    (ii) Strong-type adversaries has to additionally supply SV which is the corresponding secret value for the public key. If $SV = \perp$ then the public key is the original one. Otherwise C will use SV in the conversation for the replaced public key.
    (iii) Super-type adversaries can continue to make identification queries without supplying SV even for replaced public keys.

- **Phase 2**. I_1 will eventually output ID^* on which it wants to be challenged on. I_1 will now play the role of the cheating prover while C assumes the role of the verifier. I_1 wins the game it manages to convince C to accept.

DEFINITION 2.1 *We say an CLI scheme* Π *is* (t, q_I, ε) *-secure under passive or active/concurrent attacks if for any passive or active/concurrent Type-1 impersonator* I_1 *who runs in time t,* $Pr[I_1$*can impersonate]* $< \varepsilon$*, where* I_1 *can make at most* q_I *extract queries on partial or full private keys.*

Game II. The game is played between a challenger C and the Type-2 Impersonator I_2 for the CLI scheme Π as follows:

- **Setup** C runs **Setup** and passes both the system parameters params and the master secret key MSK to I_2.
- **Phase 1**: In this training phase, I_2 will be allowed to make the following queries adaptively to C.
  (a) **ExtrFullSK(**ID**)**. On request for the full private key USK on user ID, C will run **Partial-Private-Key-Extract**, **Set-User-Key** algorithms to generate the complete user's private key and passes it to I_2
  (b) **RequestPK(**ID**)**. On request for the public key UPK on user ID, C will run **Set-User-Key** to generate the user's public key and passes it to I_2.
  (c) **ReplacePK(**ID,UPK'**)**. I_2 is able to replace the user ID's public key UPK with the public key UPK' chosen by him. Once again, the corresponding secret value is not required for public key replacement queries. The only exception is the target ID, ID^*, otherwise it will be trivial to win the game.
  (d) **Identification(**ID**)**. For passive I_2, C will generate a valid transcript on between user ID and itself as the verifier and returns the transcript to I_2. For active/concurrent I_2, C will play the role of the prover to interact with I_2 as the cheating verifier:
    (i) Normal-type adversaries cannot make an identification query for a prover if its public key has been replaced.
    (ii) Strong-type adversaries has to additionally supply SV which is the corresponding secret value for the public key. If $SV = \perp$ then the public key is the original one. Otherwise C will use SV in the conversation for the replaced public key.
    (ii) Super-type adversaries can continue to make identification queries without supplying SV even for replaced public keys.
- **Phase 2**. I_2 will eventually output ID^* on which it wants to be challenged on. I_2 will now play the role of the cheating prover while C assumes the role of the verifier. I_2 wins the game it manages to convince C to accept.

Note that I_2 does not need to perform **ExtrPartSK** queries as it already has the master secret key and can generate partial private keys itself. I_2 is also not allowed to replace the public key of the challenge identity, but is able to do so for any other user.

DEFINITION 2.2 *We say an CLI scheme is* (t, q_I, ε) *-secure under passive or active/concurrent attacks if for any passive or active/concurrent Type-2 impersonator* I_2 *who runs in time t,* $\Pr[I_2$*can impersonate*$] < \varepsilon$*, where* I_2 *can make at most* q_I *extract queries on full private keys.*

3. Construction

In this section, the construction of the Schnorr-CLI scheme is presented. This scheme is named Schnorr-CLI due to its shared key generation algorithm inspired by the signature scheme from [14]. The scheme is described as follows:

- Setup(1^k)
  (1) Select a group $\mathbb{G}$ of order q and a generator $g \in \mathbb{G}$. Choose a random $a \xrightarrow{\$} \mathbb{Z}_q$ and compute $g_1 = g^{-a}$.
  (2) Select two hash functions $H_1 : \{0,1\}^* \times G \times G \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \times G \times G \times G \to \mathbb{Z}_q$.
  (3) Publish the master public key $mpk = \langle G, q, g, g_1, H_1, H_2 \rangle$ and securely store the master secret key $msk = a$.
- Partial-Private-Key-Extract(mpk, msk, ID)
  (1) Select a random $x \xrightarrow{\$} \mathbb{Z}_q$ and compute $X = g^x$.
  (2) Compute $\alpha = H_1(ID, g_1, X)$.
  (3) Compute $d = x + a\alpha$.
  (4) Return the partial private key $ppk = \langle \alpha, d \rangle$.
- Set-User-Key(1^k)
  (1) Select a random $b \xrightarrow{\$} \mathbb{Z}_q$ and set the user's secret value $sv = b$.
  (2) Computes $g_2 = g^{-b}$.
  (3) Outputs $UPK_1 = g_2$.
- Set-Private-Key(mpk, ppk, sv, upk, ID)
  (1) Calculates $X = g^d g_1^\alpha$ and checks if $\alpha = H_1(ID, g_1, X)$.
  (2) If correct, then calculate $\beta = H_2(ID, g_1, X, g_2)$.
  (3) Compute $s = d + b\beta$.
  (4) Compute $UPK_2 = g_2^\beta$.
  (5) Publish $upk = \langle UPK_1, UPK_2 \rangle = \langle g_2, g_2^\beta \rangle$.
  (6) $usk = \langle \alpha, \beta, s_{ID} \rangle$.
- Identification-Protocol:Prover(mpk,ID,usk) and Verifier(mpk,ID,upk)
  (1) Prover selects a random $r \xrightarrow{\$} \mathbb{Z}_q$ and computes $R = g^r$.
  (2) Prover also calculates $X = g^s g_1^\alpha g_2^\beta$ and sends X,R to Verifier.
  (3) Verifier selects a random $c \xrightarrow{\$} \mathbb{Z}_q$ and sends c to Prover.
  (4) Prover computes response $y = r + cs$ and sends y to Verifier.
  (5) Verifier accepts if and only if $g^y = R(X/g_1^\alpha g_2^\beta)^c$ and $UPK_1^\beta = UPK_2$, where $\alpha = H_1(ID, g_1, X)$ and $\beta = H_2(ID, g_1, X, g_2)$.

To prove correctness, one can show that

$$g^y = g^{r+cs} \tag{1}$$
$$= g^r (g^s)^c \tag{2}$$
$$= g^r (g^{x+a\alpha+b\beta})^c \tag{3}$$
$$= g^r (g^{x-(-a\alpha)-(-b\beta)})^c \tag{4}$$
$$= g^r \left(\frac{g^x}{g^{-a\alpha} g^{-b\beta}} \right)^c \tag{5}$$
$$= R \left(\frac{X}{g_1^\alpha g_2^\beta} \right)^c. \tag{6}$$

3.1 *Security analysis*

In this section, the four proofs of security of the Schnorr-CLI scheme is presented:

- Security against impersonation under passive attack for Super-Type-1 adversaries (IMP-PA-1).
- Security against impersonation under active/concurrent attack for Super-Type-1 adversaries (IMP-AA/CA-1).
- Security against impersonation under passive attack for Super-Type-2 adversaries (IMP-PA-2).
- Security against impersonation under active/concurrent attack for Super-Type-2 adversaries (IMP-AA/CA-2).

3.2 *Type-1 impersonation under passive attack*

THEOREM 3.1 *The Schnorr-CLI scheme is* (t, q_I, ε)*-secure against impersonation under passive attacks against* Super-Type-1 *Impersonators in the random oracle if the discrete logarithm problem is* (t', ε')*-hard, where*

$$\varepsilon \leq \sqrt{\varepsilon' e(q_I + 1)} + \frac{1}{q}, \quad t = t' - O(q_{CLI}), \tag{7}$$

where $q_{CLI} = q_{H_\alpha} + q_{H_\beta} + q_{ExtrPartSK} + q_{ExtrFullSK} + q_{RequestPK} + q_{ReplacePK} + q_{Identification}$ *represent the number of respective oracle queries by the adversary.*

Proof Assume the discrete logarithm problem is not (t', ε')-hard. We then show a simulator M that (t', ε')-breaks the discrete logarithm problem if the Schnorr-CLI scheme is not (t, q_I, ε)-secure. M takes in input $G, q, g, Z = g^z$ and runs the Type-1 impersonator I_1 as a subroutine.

Assume that any **ExtrPartSK**, **RequestPK**, **ExtrFullSK** and **Identify** queries are preceded by a **CreateUser** query, while **Identification** and **ExtrFullSK** queries are preceded by the **RequestPK** query. M also selects two hash functions $H_1 : \{0,1\}^* \times G \times G \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \times G \times G \times G \to \mathbb{Z}_q$ that are programmed as random oracles.

M keeps two lists to respond to I_1's queries: $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i} = g^x_{ID_i}, x_{ID_i}, d_{ID_i} \rangle$ and $L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, \varphi(= 0|1) \rangle$ which are initially empty. The following shows how M simulates the environment and oracle queries for I_1:

(1) **Setup**: M selects $a \xleftarrow{\$} \mathbb{Z}_q$, computes $g_1 = g^{-a}$ and sets the master public key as $mpk = \langle G, q, g, g_1, H_1, H_2 \rangle$.
(2) **CreateUser(ID_i)**: M handles **CreateUser** for two separate cases as follows:
   (a) Challenge identity: To initialize the challenge identity, M then randomly picks $j \in q_H$ out of $i \ldots q_H$ sets $ID_j = ID^*$. M then sets $X_{ID^*} = Z$, randomly chooses $b_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and computes $g_{2,ID^*} = g^{b_{ID^*}}$. M also chooses $\alpha_{ID^*}, \beta_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and sets the first entry in $L_\alpha = \langle ID^*, \alpha_{ID^*}, X_{ID^*}, \perp, \perp \rangle$ and $L_\beta = \langle ID^*, \beta_{ID^*}, b_{ID^*}, g_{2,ID^*}, 0 \rangle$.
   (b) Normal identities: Upon request of ID_i by I_1, M will create an entry for a new user by choosing $x_{ID_i}, b_{ID_i} \xleftarrow{\$} \mathbb{Z}_q$ and sets $X_{ID_i} = g^x_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}$. M also selects $\alpha_{ID_i}, \beta_{ID_i} \xleftarrow{\$} \mathbb{Z}_q$ and lets $d_{ID_i} = x_{ID_i} + a\alpha_{ID_i}$. M then creates entries in both lists as $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i}, x_{ID_i}, d_{ID_i} \rangle$ and $L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, 0 \rangle$.
(3) **ExtrPartSK(ID_i) query**: If $ID_i = ID^*$ then M aborts the simulation. Otherwise M finds $\langle ID_i, \alpha_{ID_i}, X_{ID_i}, x_{ID_i}, d_{ID_i} \rangle$ in L_α and returns $PPK_{ID_i} = \langle d_{ID_i}, \alpha_{ID_i} \rangle$.
(4) **RequestPK(ID_i) query**: M retrieves $\langle g_{2,ID_i}, g^{\beta_{ID_i}}_{2,ID_i} \rangle$ from L_β and sends it to I_1.
(5) **ExtrFullSK(ID_i) query**: If $ID_i = ID^*$ then M aborts the simulation. Otherwise M finds d_{ID_i} from L_α and b_{ID_i}, β_{ID_i} from L_β. M returns $USK_{ID_i} = \langle s_{ID_i} = d_{ID_i} + b_{ID_i}\beta_{ID_i}, \beta \rangle$ to I_1.

(6) **ReplacePK($ID_i, \tilde{UPK}_{1,ID_i}, \tilde{UPK}_{2,ID_i}$) query**: M first checks if the new public key is valid, that is, the pair fulfils $g^{USK_{1,ID_i}} = g^{PPK_{1,ID_i}} \tilde{UPK}_{1,ID_i}^{\beta_{ID_i}}$ and $\tilde{UPK}_{2,ID_i} = \tilde{UPK}_{1,ID_i}^{\beta_{ID_i}}$. This is because I_1 cannot change the hash value β_{ID_i} which is publicly verifiable. If yes, M sets $UPK_{ID_i} = \langle \tilde{UPK}_{1,ID_i}, \tilde{UPK}_{2,ID_i} \rangle$ and $\varphi = 1$ for ID_i in L_β.

(7) **Identification(ID_i) query**: I_1 will act as the cheating verifier to learn information from valid conversation transcripts from M. M retrieves ID_i's entries from L_α and L_β. If $ID_i \neq ID^*$, then M just runs **ExtrFullSK** on ID_i to obtain the full user secret key, and uses that to run the identification protocol. Otherwise, it must be that $ID_i = ID^*$. M then creates a valid transcript for each mth query by picking $X_{m,ID^*} \overset{\$}{\leftarrow} \mathbb{G}$ and $y_{m,ID^*}, c_m \overset{\$}{\leftarrow} \mathbb{Z}_q$, sets $R_{m,ID^*} = (g^{y_{m,ID^*}} / (X_{m,ID^*} / g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}}))^{c_m}$ and passes the transcript $\langle X_{m,ID^*}, R_{m,ID^*}, c_m, y_{m,ID^*} \rangle$ to I_1. It can be shown that this is a valid transcript by using **Verifier**'s checking equation:

$$R_{m,ID^*} \left(\frac{X_{m,ID^*}}{g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}}} \right)^{c_m} \tag{8}$$

$$= \left(\frac{g^{y_{m,ID^*}}}{(X_{m,ID^*} / g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}})} \right)^{c_m} \left(\frac{X_{m,ID^*}}{g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}}} \right)^{c_m} \tag{9}$$

$$= g^{y_{m,ID^P*}}. \tag{10}$$

Note that the value b_{ID^*} is not required even for replaced public keys; therefore, the Schnorr-CLI scheme is able to answer identification queries for Super-Type-1 I_1.

Eventually I_1 stops phase 1 and outputs the challenge identity, ID^*. M checks if $ID^* = ID_j$ and aborts if not. Otherwise, M runs I_1 now as a cheating prover. M then obtains I_1's commitment, X,R, selects a challenge $c_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$ and obtains the response y_1 from I_1. M then resets I_1 to the state where it just sent its commitment, selects a second challenge $c_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$ and receives y_2 as response. M is then able to extract the full user secret key s_{ID^*} as

$$\frac{y_1 - y_2}{c_1 - c_2} \tag{11}$$

$$= \frac{r - c_1 s_{ID^*} - r - c_2 s_{ID^*}}{c_1 - c_2} \tag{12}$$

$$= \frac{(c_1 - c_2) s_{ID^*}}{c_1 - c_2} \tag{13}$$

$$= s_{ID^*}. \tag{14}$$

By using the knowledge of exponent assumption from [7], M extracts b_{ID^*} for $UPK_1, UPK_2 = \langle g^{-b_{ID^*}}, g^{-b_{ID^*}\beta_{ID^*}} \rangle$ that are generated from $g, g^{\beta_{ID^*}}$. It does not matter if b_{ID^*} is the original or replaced value, since it has to fulfil $g^{s_{ID^*}} = g^{PPK_{1,ID_i}} g_2^{-b_{ID^*}\beta_{ID_i}}$ and $UPK'_{2,ID_i} = g_2^{-b_{ID^*}\beta_{ID_i}}$.

Once b_{ID^*} is extracted, M then calculates the solution to the discrete logarithm problem as

$$s_{ID^*} - a\alpha_{ID^*} - b_{ID^*}\beta_{ID^*} = z. \tag{15}$$

It remains to calculate the probability of M solving the discrete logarithm problem and winning the game. The probability of M successfully extracting two valid conversation transcripts from I_1 is bounded by $(\epsilon - 1/q)^2$ as given by the Reset Lemma [2]:

$$\Pr[M \text{ wins DLP}] \tag{16}$$

$$= \Pr\left[M \text{ computes } z \bigwedge \neg \text{abort} \right] \tag{17}$$

$$= \Pr[M \text{ computes } z|\neg\text{abort}]\Pr[\neg\text{abort}] \tag{18}$$

$$\varepsilon' \geq \left(\varepsilon - \frac{1}{q}\right)^2 \Pr[\neg\text{abort}]. \tag{19}$$

Finally calculate $Pr[\neg abort]$. Let δ be the probability that I_1 issues an extract query (of either **ExtrPartSK** or **ExtrFullSK** type) on ID^* and that I_1 makes a total of q_I of such queries. The probability of M answering all the extraction queries is δ^{q_I}. In Phase 2, the probability of M not aborting is if I_1 outputs the challenge identity ID^* which it has not queried before. This is given by the probability $1-\delta$. Putting them together, the probability of M not aborting is $\delta^{q_I}(1-\delta)$. This value is maximized at $\delta_{\text{opt}} = 1 - 1/(q_I+1)$. Using δ_{opt}, the probability M does not abort is at least $1/e(q_I+1)$ because the value $(1-1/(q_I+1))^{q_I}$ approaches $1/e$ for large q_I. Therefore, the advantage of M, ε', and the bound of the simulation is given as

$$\varepsilon' \geq \left(\varepsilon - \frac{1}{q}\right)^2 \frac{1}{e(q_I+1)} \tag{20}$$

$$\varepsilon' e(q_I+1) \geq \left(\varepsilon - \frac{1}{q}\right)^2 \tag{21}$$

$$\varepsilon \leq \sqrt{\varepsilon' e(q_I+1)} + \frac{1}{q}. \tag{22}$$

■

3.3 *Type-1 impersonation under active/concurrent attack*

THEOREM 3.2 *The Schnorr-CLI scheme is* (t, q_I, ε)*-secure against impersonation under active and concurrent attacks against Super-Type-1 Impersonators* (*IMP-AA/CA-2*) *in the random oracle if the one-more discrete logarithm problem is* $(t'', q'', \varepsilon'')$*-hard where*

$$\varepsilon \leq \sqrt{\varepsilon'' e(q''+1)} + \frac{1}{q}, \quad t = t' - O(q_{\text{CLI}}), \tag{23}$$

where $q_{\text{CLI}} = q_{H_\alpha} + q_{H_\beta} + q_{ExtrPartSK} + q_{ExtrFullSK} + q_{RequestPK} + q_{ReplacePK} + q_{Identification}$ *represent the number of respective oracle queries by the adversary.*

Proof Assume the one-more discrete logarithm problem is not $(t'', q'', \varepsilon'')$-hard. We then show a simulator M that $(t'', q'', \varepsilon'')$-breaks the one-more discrete logarithm problem if the Schnorr-CLI scheme is not (t, q_I, ε)-secure. M takes in input G,q,g, has access to *CHALL* and *DLOG* oracles, and runs the Type-1 impersonator I_1 as a subroutine. Assume that any **ExtrPartSK**, **RequestPK**, **ExtrFullSK** and **Identify** queries are preceded by a **CreateUser** query, while **Identification** and **ExtrFullSK** queries are preceded by the **RequestPK** query. M also selects two hash functions $H_1 : \{0,1\}^* \times G \times G \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \times G \times G \times G \to \mathbb{Z}_q$ that are programmed as random oracles. M keeps two lists to respond to I_1's queries: $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i} = g_{ID_i}^x, x_{ID_i}, d_{ID_i}\rangle, L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, \varphi(=0|1)\rangle$ which are initially empty. The following shows how M simulates the environment and oracle queries for I_1:

(1) **Setup**: M selects $a \xleftarrow{\$} \mathbb{Z}_q$, computes $g_1 = g^{-a}$ and sets the master public key as $mpk = \langle G, q, g, g_1, H_1, H_2\rangle$ and passes it to I_1.
(2) **CreateUser(**ID_i**)**: M handles **CreateUser** for two separate cases as follows:
   (a) Challenge identity: To initialize the challenge identity, M then randomly picks $j \in q_H$ out of $i \ldots q_H$ sets $ID_j = ID^*$. M also queries *CHALL* for the original challenge W_0 then sets

$X_{ID^*} = W_0$. Additionally M randomly chooses $b_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and computes $g_{2,ID^*} = g^{b_{ID^*}}$. M also chooses $\alpha_{ID^*}, \beta_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and sets the first entry in $L_\alpha = \langle ID^*, \alpha_{ID^*}, X_{ID^*}, \perp, \perp \rangle$ and $L_\beta = \langle ID^*, \beta_{ID^*}, b_{ID^*}, g_{2,ID^*}, 0 \rangle$.

(b) Normal identities: Upon request of ID_i by I_1, M will create an entry for a new user by choosing $x_{ID_i}, b_{ID_i} \xleftarrow{\$} \mathbb{Z}_q$ and sets $X_{ID_i} = g^x_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}$. M also selects $\alpha_{ID_i}, \beta_{ID_i} \xleftarrow{\$} \mathbb{Z}_q$ and lets $d_{ID_i} = x_{ID_i} + a\alpha_{ID_i}$. M then creates entries in both lists as $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i}, x_{ID_i}, d_{ID_i} \rangle$ and $L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, 0 \rangle$.

(3) **ExtrPartSK(ID_i) query**: If $ID_i = ID^*$ then M aborts the simulation. Otherwise M finds $\langle ID_i, \alpha_{ID_i}, X_{ID_i}, x_{ID_i}, d_{ID_i} \rangle$ in L_α and returns $PPK_{ID_i} = \langle d_{ID_i}, \alpha_{ID_i} \rangle$.

(4) **ExtrFullSK(ID_i) query**: If $ID_i = ID^*$ then M aborts the simulation. Otherwise M finds d_{ID_i} from L_α and b_{ID_i}, β_{ID_i} from L_β. M returns $USK_{ID_i} = \langle s_{ID_i} = d_{ID_i} + b_{ID_i}\beta_{ID_i}, \beta \rangle$ to I_1.

(5) **ReplacePK($ID_i, U\tilde{P}K_{1,ID_i}, U\tilde{P}K_{2,ID_i}$) query**: M first checks if the new public key is valid, that is, the pair fulfils $g^{USK_{1,ID_i}} = g^{PPK_{1,ID_i}} U\tilde{P}K_{1,ID_i}^{\beta_{ID_i}}$ and $U\tilde{P}K_{2,ID_i} = U\tilde{P}K_{1,ID_i}^{\beta_{ID_i}}$. This is because I_1 cannot change the hash value β_{ID_i} which is publicly verifiable. If yes, M sets $UPK_{ID_i} = \langle U\tilde{P}K_{1,ID_i}, U\tilde{P}K_{2,ID_i} \rangle$ and $\varphi = 1$ for ID_i in L_β;

(6) **Identification(ID_i) query**: I_1 will act as the cheating verifier to learn information from valid conversation transcripts from M. M retrieves ID_i's entries from L_α and L_β. If $ID_i \neq ID^*$, then M just runs **ExtrFullSK** on ID_i to obtain the full user secret key and uses that to run the identification protocol. Otherwise, it must be that $ID_i = ID^*$. M then creates a valid conversation for each mth query by first querying $CHALL$ for W_i and sets $R_{m,ID^*} = W_i$ and $X_{m,ID^*} = W_0$. M then sends $\langle X_{m,ID^*}, R_{m,ID^*} \rangle$ to I_1. I_1 returns with a random $c_m \xleftarrow{\$} \mathbb{Z}_q$. M then queries $DLOG$ with $W_i(W_0/g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}})^{c_m}$ and returns the response as $y_{m,ID^*} = DLOG(W_i(W_0/g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}})^{c_m})$. It can be shown that this is a valid conversation by using **Verifier**'s checking equation:

$$R_{m,ID^*} \left(\frac{X_{m,ID^*}}{g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}}} \right)^{c_m} \tag{24}$$

$$W_i \left(\frac{W_0}{g_1^{\alpha_{ID^*}} g_2^{\beta_{ID^*}}} \right)^{c_m} \tag{25}$$

$$= g^{y_{m,ID^*}}. \tag{26}$$

Note that the value b_{ID^*} is not required even for replaced public keys; therefore, the Schnorr-CLI scheme is able to answer identification queries for Super-Type-1 I_1.

Eventually I_1 stops phase 1 and outputs the challenge identity, ID^*. M checks if $ID^* = ID_j$ and aborts if not. Otherwise, M runs I_1 now as a cheating prover. M then obtains I_1's commitment, X,R, selects a challenge $c_1 \xleftarrow{\$} \mathbb{Z}_q$ and obtains the response y_1 from I_1. M then resets I_1 to the state where it just sent its commitment, selects a second challenge $c_2 \xleftarrow{\$} \mathbb{Z}_q$ and receives y_2 as response. M is then able to extract the full user secret key s_{ID^*} as

$$\frac{y_1 - y_2}{c_1 - c_2} \tag{27}$$

$$= \frac{r - c_1 s_{ID^*} - r - c_2 s_{ID^*}}{c_1 - c_2} \tag{28}$$

$$= \frac{(c_1 - c_2) s_{ID^*}}{c_1 - c_2} \tag{29}$$

$$= s_{ID^*}. \tag{30}$$

By using the knowledge of exponent assumption from [7], M extracts b_{ID^*} for $UPK_1, UPK_2 = \langle g^{-b_{ID^*}}, g^{-b_{ID^*}\beta_{ID^*}} \rangle$ that are generated from $g, g^{\beta_{ID^*}}$. It does not matter if b_{ID^*} is the original or replaced value, since it has to fulfil $g^{s_{ID^*}} = g^{PPK_{1,ID_i}} g_2^{-b_{ID^*}\beta_{ID_i}}$ and $UPK'_{2,ID_i} = g_2^{-b_{ID^*}\beta_{ID_i}}$.

Once b_{ID^*} is extracted, M then calculates the solution to the discrete logarithm problem as

$$s_{ID^*} - a\alpha_{ID^*} - b_{ID^*}\beta_{ID^*} = w_0. \tag{31}$$

M is then able to calculate the solutions for the challenges $w_1, \ldots, w_m$ as

$$w_j = y_j - c_j(w_0 + a\alpha_{ID^*} + b_{ID^*}\beta_{ID^*}). \tag{32}$$

The probability of M winning one-more discrete logarithm game is the same as the IMP-PA-1 game, except that ε', the advantage of M in solving the discrete logarithm problem, is substituted with ε", the advantage of M over the one-more discrete logarithm game. ■

3.4 *Type-2 impersonation under passive attack*

THEOREM 3.3 *The Schnorr-CLI scheme is* (t, q_I, ε)*-secure against impersonation under passive attacks against* Super-Type-2 *Impersonators in the random oracle if the discrete logarithm problem is* (t', ε')*-hard where*

$$\varepsilon \leq \sqrt{\varepsilon' e(q_I + 1)} + \frac{1}{q}, \quad t = t' - O(q_{\text{CLI}}), \tag{33}$$

where $q_{\text{CLI}} = q_{H_\alpha} + q_{H_\beta} + q_{ExtrFullSK} + q_{RequestPK} + q_{ReplacePK} + q_{Identification}$ *represent the number of respective oracle queries by the adversary.*

Proof Assume the discrete logarithm problem is not (t', ε')-hard. We then show a simulator M that (t', ε')-breaks the discrete logarithm problem if the Schnorr-CLI scheme is not $(t_{IBI}, q_I, \varepsilon)$-secure. M takes in input $G, q, g, Z = g^z$ and runs the Type-2 impersonator I_2 as a subroutine. Assume that any **RequestPK**, **ExtrFullSK** and **Identify** queries are preceded by a **CreateUser** query, while **Identification** and **ExtrFullSK** queries are preceded by the **RequestPK** query. M also selects two hash functions $H_1 : \{0,1\}^* \times G \times G \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \times G \times G \times G \to \mathbb{Z}_q$ that are programmed as random oracles. M keeps two lists to respond to I_2's queries: $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i} = g_{ID_i}^x, x_{ID_i}, d_{ID_i} \rangle, L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, \varphi(= 0|1) \rangle$ which are initially empty. The following shows how M simulates the environment and oracle queries for I_2:

(1) **Setup**: M picks $a \xleftarrow{\$} \mathbb{Z}_q$, computes $g_1 = g^{-a}$ and sets the master public key as $mpk = \langle G, q, g, g_1 = g^- a, H_1, H_2 \rangle$ and passes it to I_2. It keeps the master secret key a to itself.
(2) **CreateUser(ID_i)**: M handles **CreateUser** for two separate cases as follows:
   (a) Challenge identity: To initialize the challenge identity, M then randomly picks $j \in q_H$ out of $i \ldots q_H$ sets $ID_j = ID^*$. M then sets $g_{2,ID^*} = Z$, randomly chooses $x_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and computes $X_{ID^*} = g^{x_{ID^*}}$. M also chooses $\alpha_{ID^*}, \beta_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and sets the first entry in $L_\alpha = \langle ID^*, \alpha_{ID^*}, X_{ID^*}, x_{ID^*}, d_{ID^*} \rangle$ and $L_\beta = \langle ID^*, \beta_{ID^*}, \perp, g_{2,ID^*}, 0 \rangle$.
   (b) Normal identities: Upon request of ID_i by I_2 where $i \neq j$, M will create an entry for a new user by choosing $x_{ID_i}, b_{ID_i} \xleftarrow{\$} \mathbb{Z}_q$ and sets $X_{ID_i} = g_{ID_i}^x, g_{2,ID_i} = g^{b_{ID_i}}$. M also selects $\alpha_{ID^*}, \beta_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and lets $d_{ID_i} = x_{ID_i} + a\alpha_{ID_i}$. M then creates entries in both lists as $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i}, x_{ID_i}, d_{ID_i} \rangle$ and $L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, 0 \rangle$.
(3) **ExtrFullSK(ID_i) query**: If $ID_i = ID^*$ then M aborts the simulation. Otherwise M finds d_{ID_i} from L_α and b_{ID_i}, β_{ID_i} from L_β. M returns $USK_{ID_i} = \langle s_{ID_i} = d_{ID_i} + b_{ID_i}\beta_{ID_i}, \beta \rangle$ to I_1.

(4) **ReplacePK(ID_i, $U\tilde{P}K_{1,ID_i}$, $U\tilde{P}K_{2,ID_i}$) query**: This query is only applicable to identities where $ID_i \neq ID^*$. M first checks if the new public key is valid, that is, the pair fulfils $g^{USK_{1,ID_i}} = g^{PPK_{1,ID_i}} U\tilde{P}K_{1,ID_i}^{\beta_{ID_i}}$ and $U\tilde{P}K_{2,ID_i} = U\tilde{P}K_{1,ID_i}^{\beta_{ID_i}}$. This is because I_1 cannot change the hash value β_{ID_i} which is publicly verifiable. If yes, M sets $UPK_{ID_i} = \langle U\tilde{P}K_{1,ID_i}, U\tilde{P}K_{2,ID_i}\rangle$ and $\varphi = 1$ for ID_i in L_β.

(5) **Identification(ID_i) query**: I_2 will act as the cheating verifier to learn information from valid conversation transcripts from M. M retrieves ID_i's entries from L_α and L_β. If $ID_i \neq ID^*$, then M just runs **ExtrFullSK** on ID_i to obtain the full user secret key and uses that to run the identification protocol. Otherwise, it must be that $ID_i = ID^*$. M then creates a valid transcript for each mth query by picking $X_{m,ID_i} \xleftarrow{\$} \mathbb{G}$ and $y_{m,ID_i}, c_m \xleftarrow{\$} \mathbb{Z}_q$, sets $R_{m,ID_i} = (g^{y_{m,ID_i}}/(X_{m,ID_i}/g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}}))^{c_m}$ and passes the transcript $\langle X_{m,ID_i}, R_{m,ID_i}, c_m, y_{m,ID_i}\rangle$ to I_2. It can be shown that this is a valid transcript by using **Verifier**'s checking equation:

$$R_{m,ID_i}\left(\frac{X_{m,ID_i}}{g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}}}\right)^{c_m} \tag{34}$$

$$= \left(\frac{g^{y_{m,ID_i}}}{X_{m,ID_i}/g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}}}\right)^{c_m} \left(\frac{X_{m,ID_i}}{g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}}}\right)^{c_m} \tag{35}$$

$$= g^{y_{m,ID_i}}. \tag{36}$$

Note that the value b_{ID^*} is not required even for replaced public keys; therefore, the Schnorr-CLI scheme is able to answer identification queries for Super-Type-2 I_2.

Eventually I_2 stops phase 1 and outputs the challenge identity, ID^*. M checks if $ID^* = ID_j$ and aborts if not. Otherwise, M runs I_2 now as a cheating prover. Since I_2 is not allowed to replace ID^*'s public key, the public key must be the one as originally created. M then obtains I_2's commitment, X,R, selects a challenge $c_1 \xleftarrow{\$} \mathbb{Z}_q$ and obtains the response y_1 from I_2. M then resets I_2 to the state where it just sent its commitment, selects a second challenge $c_2 \xleftarrow{\$} \mathbb{Z}_q$ and receives y_2 as response. M is then able to extract the full user secret key s_{ID^*} as

$$\frac{y_1 - y_2}{c_1 - c_2} \tag{37}$$

$$= \frac{r - c_1 s_{ID^*} - r - c_2 s_{ID^*}}{c_1 - c_2} \tag{38}$$

$$= \frac{(c_1 - c_2) s_{ID^*}}{c_1 - c_2} \tag{39}$$

$$= s_{ID^*}. \tag{40}$$

M then calculates the solution to the discrete logarithm problem as

$$\frac{s_{ID^*} - a\alpha_{ID^*} - x_{ID^*}}{\beta_{ID^*}} = z. \tag{41}$$

It remains to calculate the probability of M solving the discrete logarithm problem and winning the game. The probability of M successfully extracting two valid conversation transcripts from I_2 is bounded by $(\epsilon - 1/q)^2$ as given by the Reset Lemma [2]:

$$\Pr[M \text{ wins DLP}] \tag{42}$$

$$= \Pr\left[M \text{ computes } z \bigwedge \neg\text{abort}\right] \tag{43}$$

$$= \Pr[M \text{ computes } z|\neg\text{abort}]\Pr[\neg\text{abort}] \tag{44}$$

$$\varepsilon' \geq \left(\varepsilon - \frac{1}{q}\right)^2 \Pr[\neg\text{abort}]. \tag{45}$$

Finally calculate Pr[¬abort]. Let δ be the probability that I_2 issues an **ExtrFullSK** on ID^* and that I_2 makes a total of q_I of such queries. The probability of M answering all the extraction queries is δ^{q_I}. In Phase 2, the probability of M not aborting is if I_2 outputs the challenge identity ID^* which it has not queried before. This is given by the probability $1-\delta$. Putting them together, the probability of M not aborting is $\delta^{q_I}(1-\delta)$. This value is maximized at $\delta_{\text{opt}} = 1 - 1/(q_I+1)$. Using δ_{opt}, the probability M does not abort is at least $1/e(q_I+1)$ because the value $(1-1/(q_I+1))^{q_I}$ approaches $1/e$ for large q_I. Therefore, the advantage of M, ϵ', and the bound of the simulation is given as

$$\varepsilon' \geq \left(\varepsilon - \frac{1}{q}\right)^2 \frac{1}{e(q_I+1)} \tag{46}$$

$$\varepsilon' e(q_I+1) \geq \left(\varepsilon - \frac{1}{q}\right)^2 \tag{47}$$

$$\varepsilon \leq \sqrt{\varepsilon' e(q_I+1)} + \frac{1}{q}. \tag{48}$$

■

3.5 *Type-2 impersonation under active/concurrent attack*

THEOREM 3.4 *The Schnorr-CLI scheme is* (t, q_I, ε)*-secure against impersonation under active and concurrent attacks against Super-Type-*2 *Impersonators* (*IMP-AA/CA-2*) *in the random oracle if the one-more discrete logarithm problem is* $(t'', q'', \varepsilon'')$*-hard where*

$$\varepsilon \leq \sqrt{\varepsilon'' e(q_I+1)} + \frac{1}{q}, \quad t = t' - O(q_{\text{CLI}}), \tag{49}$$

where $q_{\text{CLI}} = q_{H_\alpha} + q_{H_\beta} + q_{ExtrFullSK} + q_{RequestPK} + q_{ReplacePK} + q_{Identification}$ *represent the number of respective oracle queries by the adversary.*

Proof Assume the one-more discrete logarithm problem is not $(t'', q'', \varepsilon'')$-hard. We then show a simulator M that $(t'', q'', \varepsilon'')$-breaks the discrete logarithm problem if the Schnorr-CLI scheme is not (t, q_I, ε)-secure. M takes in input G,q,g, has access to *CHALL* and *DLOG* oracles, and runs the Type-2 impersonator I_2 as a subroutine. Assume that any **RequestPK**, **ExtrFullSK** and **Identify** queries are preceded by a **CreateUser** query, while **Identification** and **ExtrFullSK** queries are preceded by the **RequestPK** query. M also selects two hash functions $H_1 : \{0,1\}^* \times G \times G \to \mathbb{Z}_q$ and $H_2 : \{0,1\}^* \times G \times G \times G \to \mathbb{Z}_q$ that are programmed as random oracles. M keeps two lists to respond to I_2's queries: $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i} = g^x_{ID_i}, x_{ID_i}, d_{ID_i}\rangle, L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, \varphi(=0|1)\rangle$ which are initially empty. The following shows how M simulates the environment and oracle queries for I_2:

(1) **Setup**: M picks $a \xleftarrow{\$} \mathbb{Z}_q$, computes $g_1 = g^{-a}$ and sets the master public key as $mpk = \langle G, q, g, g_1 = g^- a, H_1, H_2\rangle$ and passes it to I_2. It keeps the master secret key a to itself.
(2) **CreateUser(ID_i)**: M handles **CreateUser** for two separate cases as follows:

(a) Challenge identity: To initialize the challenge identity, M then randomly picks $j \in q_H$ out of $i \ldots q_H$ sets $ID_j = ID^*$. M also queries *CHALL* for the original challenge W_0 then sets $g_{2,ID^*} = W_0$. Additionally M randomly chooses $x \xleftarrow{\$} \mathbb{Z}_q$ and computes $X_{ID^*} = g^{x_{ID^*}}$. M also chooses $\alpha_{ID^*}, \beta_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and sets the first entry in $L_\alpha = \langle ID^*, \alpha_{ID^*}, X_{ID^*}, x_{ID^*}, d_{ID^*} \rangle$ and $L_\beta = \langle ID^*, \beta_{ID^*}, \perp, g_{2,ID^*}, 0 \rangle$.

(b) Normal identities: Upon request of ID_i by I_2, M will create an entry for a new user by choosing $x_{ID_i}, b_{ID_i} \xleftarrow{\$} \mathbb{Z}_q$ and sets $X_{ID_i} = g^x_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}$. M also selects $\alpha_{ID^*}, \beta_{ID^*} \xleftarrow{\$} \mathbb{Z}_q$ and lets $d_{ID_i} = x_{ID_i} + a\alpha_{ID_i}$. M then creates entries in both lists as $L_\alpha = \langle ID_i, \alpha_{ID_i}, X_{ID_i}, x_{ID_i}, d_{ID_i} \rangle$ and $L_\beta = \langle ID_i, \beta_{ID_i}, b_{ID_i}, g_{2,ID_i} = g^{b_{ID_i}}, 0 \rangle$.

(3) **ExtrFullSK(ID_i) query**: If $ID_i = ID^*$ then M aborts the simulation. Otherwise M finds d_{ID_i} from L_α and b_{ID_i}, β_{ID_i} from L_β. M returns $USK_{ID_i} = \langle s_{ID_i} = d_{ID_i} + b_{ID_i}\beta_{ID_i}, \beta \rangle$ to I_1.

(4) **ReplacePK($ID_i, U\tilde{P}K_{1,ID_i}, U\tilde{P}K_{2,ID_i}$) query**: This query is only applicable to identities where $ID_i \neq ID^*$. M first checks if the new public key is valid, that is, the pair fulfils $g^{USK_{1,ID_i}} = g^{PPK_{1,ID_i}} U\tilde{P}K_{1,ID_i}^{\beta_{ID_i}}$ and $U\tilde{P}K_{2,ID_i} = U\tilde{P}K_{1,ID_i}^{\beta_{ID_i}}$. This is because I_1 cannot change the hash value β_{ID_i} which is publicly verifiable. If yes, M sets $UPK_{ID_i} = \langle U\tilde{P}K_{1,ID_i}, U\tilde{P}K_{2,ID_i} \rangle$ and $\varphi = 1$ for ID_i in L_β.

(5) **Identification(ID_i) query**: I_2 will act as the cheating verifier to learn information from valid conversation transcripts from M. M retrieves ID_i's entries from L_α and L_β. If $ID_i \neq ID^*$, then M just runs **ExtrFullSK** on ID_i to obtain the full user secret key, and uses that to run the identification protocol. Otherwise, it must be that $ID_i = ID^*$. M then creates a valid conversation for each mth query by first querying *CHALL* for W_i and sets $R_{m,ID_i} = W_i$ and $X_{m,ID_i} = g^{x_{ID^*}}$. M then sends $\langle X_{m,ID_i}, R_{m,ID_i} \rangle$ to I_2. I_2 returns with a random $c_m \xleftarrow{\$} \mathbb{Z}_q$. M then queries *DLOG* with $W_i(W_0/g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}})^{c_m}$ and returns the response as $y_{m,ID_i} = DLOG(W_i(W_0/g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}})^{c_m})$. It can be shown that this is a valid conversation by using **Verifier**'s checking equation:

$$R_{m,ID_i} \left(\frac{X_{m,ID_i}}{g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}}} \right)^{c_m} \tag{50}$$

$$= W_i \left(\frac{W_0}{g_1^{\alpha_{ID_i}} g_2^{\beta_{ID_i}}} \right)^{c_m} \tag{51}$$

$$= g^{y_{m,ID_i}}. \tag{52}$$

Note that the value b_{ID^*} is not required even for replaced public keys; therefore, the Schnorr-CLI scheme is able to answer identification queries for Super-Type-2 I_2.

Eventually I_2 stops phase 1 and outputs the challenge identity, ID^*. M checks if $ID^* = ID_j$ and aborts if not. Otherwise, M runs I_2 now as a cheating prover. If I_2 has previously replaced ID^*'s public key, M also obtains the corresponding secret value b' from I_2 and sets $b_{ID^*} = b'$. M then obtains I_2's commitment, X, R, selects a challenge $c_1 \xleftarrow{\$} \mathbb{Z}_q$ and obtains the response y_1 from I_2. M then resets I_2 to the state where it just sent its commitment, selects a second challenge $c_2 \xleftarrow{\$} \mathbb{Z}_q$ and receives y_2 as response. M is then able to extract the full user secret key s_{ID^*} as

$$\frac{y_1 - y_2}{c_1 - c_2} \tag{53}$$

$$= \frac{r - c_1 s_{ID^*} - r - c_2 s_{ID^*}}{c_1 - c_2} \tag{54}$$

$$= \frac{(c_1 - c_2)s_{ID^*}}{c_1 - c_2} \tag{55}$$

$$= s_{ID^*}. \tag{56}$$

M then calculates the solution to the initial challenge as

$$\frac{s_{ID^*} - x_{ID_i} - a\alpha_{ID^*}}{\beta_{ID^*}} = w_0. \tag{57}$$

M is then able to calculate the solutions for the challenges $w_1, \ldots, w_m$ as

$$w_j = y_j - c_j(x_{ID^*} + a\alpha_{ID^*} + w_0\beta_{ID^*}). \tag{58}$$

The probability of M winning one-more discrete logarithm game is the same as the IMP-PA-2 game, except that ε', the advantage of M in solving the discrete logarithm problem, is substituted with ε", the advantage of M over the one-more discrete logarithm game. ■

4. Efficiency analysis

The only other CLI scheme from [5] utilizes bilinear pairings. Since there are no other CLI schemes that are pairing-free in existence, our proposed scheme is currently the fastest in literature, and therefore, we are unable to do any comparisons. Without pre-computation, the Schnorr-CLI scheme requires the operational costs shown in Table 1.

However, it is possible to increase the scheme's efficiency by pre-computing the value of X and storing it as part of *usk* since it has to be sent to the verifier every run of the protocol. The identification protocol with pre-computation has the operational costs given in Table 2.

Table 1. Operation costs for the Schnorr-CLI scheme without pre-computation.

Algorithm	H	A	M	GM	E
Setup	0	0	0	0	1
PPK-Extract	1	1	1	0	1
Set-User-Key	0	0	0	0	1
Set-Private-Key	2	1	1	1	3
Prover	0	1	1	2	4
Verifier	2	0	0	3	5

Note: H, Hash operation; A, Addition mod q; M, Multiplication mod q; GM, Group Multiplication; E, Exponentiation mod q.

Table 2. Operation costs for the Schnorr-CLI scheme's identification protocol with pre-computation.

Algorithm	H	A	M	GM	E
Prover	0	1	1	0	1
Verifier	2	0	0	3	5

Note: H, Hash operation; A, Addition mod q; M, Multiplication mod q; GM, Group Multiplication; E, Exponentiation mod q.

5. Conclusion

In this paper, we show a pairing-free CLI scheme secure against Type-1 and Type-2 adversaries, both passive and active/concurrent. Our scheme does not utilize any bilinear pairings and therefore is efficient computationally.

Acknowledgements

This research was funded by Ministry of Higher Education Malaysia through the Exploratory Research Grant Scheme ERGS/1/2011/PK/MMU/03/1 and the Fundamental Research Grant Scheme FRGS/2/2013/ICT07/MMU/03/5.

References

[1] S.S. Al-Riyami and K.G. Paterson, *Certificateless public key cryptography*, in *ASIACRYPT, Taipei, Taiwan*, Lecture Notes in Computer Science, Vol. 2894, C.-S. Laih, ed., Springer, Berlin, Heidelberg, 2003, pp. 452–473.
[2] M. Bellare and A. Palacio, *GQ and Schnorr identification schemes: Proofs of security against impersonation under active and concurrent attacks*, in *CRYPTO, Santa Barbara, California*, Lecture Notes in Computer Science, Vol. 2442, M. Yung, ed., Springer, Berlin, Heidelberg, 2002, pp. 162–177.
[3] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko, *The one-more-rsa-inversion problems and the security of Chaum's blind signature scheme*, J. Cryptol. 16 (2003), pp. 185–215.
[4] M. Bellare, C. Namprempre and G. Neven, *Security proofs for identity-based identification and signature schemes*, in *EUROCRYPT, Interlaken, Switzerland*, Lecture Notes in Computer Science, Vol. 3027, C. Cachin and J. Camenisch, eds., Springer, Berlin, Heidelberg, 2004, pp. 268–286.
[5] J.J. Chin, R.C.W. Phan, R. Behnia and S.H. Heng, *An efficient and provably secure certificateless identification scheme*, in *SECRYPT, Rejkavik, Iceland*, P. Samarati, ed., SciTePress, Lisbon, 2013, pp. 371–378.
[6] J.J. Chin, R. Behnia, S.H. Heng and R.C.W. Phan, *Cryptanalysis of a certificateless identification scheme*, Secur. Commun. Networks (2014). Available at http://dx.doi.org/10.1002/sec.963.
[7] I. Damgård, *Towards practical public key systems secure against chosen ciphertext attacks*, in *CRYPTO, Santa Barbara, California*, Lecture Notes in Computer Science, Vol. 576, J. Feigenbaum, ed., Springer, Berlin, Heidelberg, 1991, pp. 445–456.
[8] M.H. Dehkordi and R. Alimoradi, *Certificateless identification protocols from super singular elliptic curve*, Secur. Commun. Networks 7(6) (2014), pp. 979–986.
[9] A.W. Dent, *A survey of certificateless encryption schemes and security models*, Int. J. Inf. Secur. 7 (2008), pp. 349–377.
[10] D. He, S. Padhye, and J. Chen, *An efficient certificateless two-party authenticated key agreement protocol*, Comput. Math. Appl. 64 (2012), pp. 1914–1926.
[11] D. He, Y. Chen and J. Chen, *An efficient certificateless proxy signature scheme without pairing*, Math. Comput. Model. 57 (2013), pp. 2510–2518.
[12] X. Huang, Y. Mu, W. Susilo, D.S. Wong and W. Wu, *Certificateless signature revisited*, in *ACISP, Townsville, Australia*, Lecture Notes in Computer Science, Vol. 4586, J. Pieprzyk, H. Ghodosi, and E. Dawson, eds., Springer, Berlin, Heidelberg, 2007, pp. 308–322.
[13] K. Kurosawa and S.H. Heng, *From digital signature to ID-based identification/signature*, in *Public Key Cryptography*, Singapore, Lecture Notes in Computer Science, Vol. 2947, F. Bao, R.H. Deng, and J. Zhou, eds., Springer, Berlin, Heidelberg, 2004, pp. 248–261.
[14] C.P. Schnorr, *Efficient identification and signatures for smart cards*, in *CRYPTO*, Santa Barbara, California, Lecture Notes in Computer Science, Vol. 435, G. Brassard, ed., Springer, Berlin, Heidelberg, 1989, pp. 239–252.
[15] A. Shamir, *Identity-based cryptosystems and signature schemes*, in *CRYPTO, Santa Barbara, California*, Lecture Notes in Computer Science, Vol. 196, G. R. Blakley and D. Chaum, eds., Springer, Berlin, Heidelberg, 1984, pp. 47–53.