Certificateless Public Key Encryption without Pairing

Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo

Centre for Information Security Research
School of Information Technology and Computer Science
University of Wollongong
Wollongong NSW 2522, Australia
{baek, rei, wsusilo}@uow.edu.au

Abstract. "Certificateless Public Key Cryptography" has very appealing features, namely it does not require any public key certification (cf. traditional Public Key Cryptography) nor having key escrow problem (cf. Identity-Based Cryptography). Unfortunately, construction of Certificateless Public Key Encryption (CLPKE) schemes has so far depended on the use of Identity-Based Encryption, which results in the bilinear pairing-based schemes that need costly operations. In this paper, we consider a relaxation of the original model of CLPKE and propose a new CLPKE scheme that does not depend on the bilinear pairings. We prove that in the random oracle model, our scheme meets the strong security requirements of the new model of CLPKE such as security against public key replacement attack and chosen ciphertext attack, assuming that the standard Computational Diffie-Hellman problem is intractable.

1 Introduction

Motivation. Consider a situation where Alice wants to send a confidential message to Bob. Using a public key encryption (PKE) scheme, Alice needs to obtain Bob's public key and encrypts her message using this key. When this operation is performed correctly, then only Bob who is in possession of a private key matched to his public key can decrypt the ciphertext and read the message. One direct implication of this mechanism is an assurance that Bob's public key is authentic. In the normal Public Key Cryptography (PKC), this assurance is obtained via certification by a Certification Authority (CA). More precisely, the CA digitally signs on Bob's public key and the "Digital Certificate" which contains the resulting signature and the public key should be checked against the CA's public key by any interested party. However, the realization of this authentication mechanism called "Public Key Infrastructure (PKI)" has long been a concern for implementers as the issues associated with revocation, storage and distribution of certificates must be resolved.

On the other hand, a very different approach to the above authenticity problem in public key cryptography was made by Shamir [16]. In this new approach named "Identity-Based Cryptography (IBC)", every user's public key is just his/her identity (identifier) which is an arbitrary string such as an email address while the corresponding private key is a result of some mathematical operation that takes as input the user's identity and the secret master key of a trusted authority, sometimes referred to as "Private Key Generator (PKG)". Notice that in this setting, certification of the public keys is provided *implicitly* based on the fact that if the user has obtained a correct private key associated with the published identity, he/she will be able to perform some cryptographic operations such as decrypt or sign. Hence, it is no longer necessary to *explicitly* authenticate public keys, i.e. verifying the digital certificates of the public keys, as in the traditional PKI setting. However, an obvious drawback of IBC is an unconditional trust that must be placed to the PKG, as the PKG can always impersonate any single entity as every user's private key is known to the PKG.

In order to resolve the above escrow problem in IBC while keeping the implicit certification property of IBC, a new paradigm called "Certificateless Public Key cryptography (CLPKC)" was introduced by Al-Riyami and Paterson [1]. In CLPKC, the user's public key is no longer an arbitrary string. Rather, it is similar to the public key used in the traditional PKC generated by the user. However, a crucial difference between them is that the public key in CLPKC does not need to be *explicitly* certified as it has been generated using some "partial private key" obtained from the trusted authority called "Key Generation Center (KGC)". Note here that the KGC does not know the users' private keys since they contain secret information generated by the users themselves, thereby removing the escrow problem in IBC.

Therefore, it is sometimes said that CLPKC lies in between PKC and IBC. However, it should be emphasized that so far "Certificateless Public Key Encryption (CLPKE)" schemes have been constructed within the framework of Identity-Based Encryption (IBE) schemes proposed by Boneh and Franklin [5], and Cocks [7]. As a result, the CLPKE schemes in the literature had to be based on either the bilinear pairings or somewhat inefficient IBE scheme proposed in [7]. In spite of the recent advances in implementation technique, the pairing computation is still considered as expensive compared with "standard" operations such as modular exponentiations in finite fields. According to the current MIRACL [12] implementation, a 512-bit Tate pairing takes 20 ms whereas a 1024-bit prime modular exponentiation takes 8.80 ms. Also, it is known that Cock's IBE scheme [7] uses bit-by-bit encryption and hence outputs long ciphertexts.

Being aware of the above problem of the current constructions of CLPKE, we focus on constructing a *CLPKE scheme that does not depend on the pairings*. This way, our scheme will be more efficient than all of the CLPKE schemes proposed so far [1, 2, 17]. The approach we make to achieve such a goal is to construct a CLPKE scheme that tends more towards a PKE scheme in the traditional PKI setting. We note that the reason why the CLPKE schemes in [1, 2, 17] have to depend on IBE is that in those schemes, a user need not be in possession of a partial private key before generating a public key, which is indeed a feature provided by IBE. By relaxing this requirement, however, we could construct a very efficient CLPKE scheme *without* pairings.

Related Work. Al-Riyami and Paterson [1] proposed CLPKE and Certificateless Public Key Signature (CLPKS) schemes, all of which are based on the bilinear pairing used in Boneh and Franklin's [5] IBE scheme. We note that their new construction of a CLPKE scheme given in [2] is also based on the bilinear pairing.

Recently, a generic construction of CLPKE was given by Yum and Lee [17], who showed that any IBE and normal public key encryption schemes, if combined together properly, can yield a CLPKE scheme. Although their result indeed brings some flexibility in constructing CLPKE schemes, one should still expect a new IBE scheme to emerge to obtain a CLPKE scheme that does not depend on the bilinear pairings or Cock's IBE scheme [7].

More recently, Castellucia et al. [6] proposed a new Secret Handshake (SH) scheme. An interesting feature of this scheme compared with the original SH scheme [3] is that it does not depend on the bilinear pairings but the key issuing technique based on the Schnorr signature [15], which is very similar to the "Self-Certified Keys" technique presented in [13], so that the required computational cost is twice less expensive than the original one. We note that Castellucia et al. [6] mentioned that their technique can also be applied to build a Hidden Credential (HC) scheme [11], however, no further application of it was considered.

Finally, we remark that CLPKC in general and our work are related to the early works on the "self-certified keys" [10, 13, 14]. One crucial difference between schemes based on CLPKC and those based on self-certified keys is that the former depends more on the "identity-based" property, so that a user does not need to obtain any (private) key from the KGC *before* generating a public key. This property is useful as mentioned in [1], but we emphasize that if one merely wants the "*certificate-less* property" for public key encryption, there is an alternative method to construct a certificateless public key encryption scheme, which bypasses the use of IBE. The technique of self-certified keys is such a method and is similar to our method to construct the CLPKE scheme presented in this paper. However, we point out that no schemes in [10, 13, 14] are supported by *formal* security analysis. Moreover, the CLPKE scheme presented in this paper is structurally different from any schemes presented in [10, 13, 14]. Hence, one can view our work as formal treatment and extension of the early works on the self-certified keys.

Our Contributions. In this paper, we elaborate on a new formal model of CLPKE and construct a CLPKE scheme that does not depend on the bilinear pairings: We extend the technique of [3, 13] non-trivially to the CLPKE setting and construct a new CLPKE scheme which is almost as efficient as the "hashed" ElGamal encryption scheme modified by the Fujisaki-Okamoto transform technique [8]. We prove in the random oracle model [4] that our scheme is secure against adaptive chosen ciphertext attacks, relative to the Computational Diffie-Hellman (CDH) problem.

2 Definitions

Model. The main goal of CLPKE [1] is to allow a sender to transmit a confidential message to a recipient by encrypting the message using the recipient's public key which does not have to be contained in a certificate issued by CA. As a result, one can remove the certificate checking process that increases the system complexity. In spite of the absence of the checking process, the sender is guaranteed that *only* the honest recipient who has gone through appropriate authentication procedure and has obtained a right "partial private key" associated with his identifier $\texttt{ID}$ from the Key Generation Center (KGC) will be able to decrypt the message.

Our model of CLPKE is very similar to that of original CLPKE [1]. In fact, the sub-algorithms of our CLPKE, Setup, SetSecretValue, SetPrivateKey, Encrypt and Decrypt are identical to those of the original CLPKE. Two different algorithms are PartialKeyExtract and SetPublicKey. PartialKeyExtract is similar to the "Partial Private Key Extract" algorithm of the original CLPKE with a difference that the output of PartialKeyExtract consists of not only a partial private key which should be kept secret but a "*partial public key*" which will be used to generate a public key later by the user. The only difference between the "Set Public Key" algorithm of the original CLPKE and SetPublicKey of our CLPKE is that in our model of CLPKE, the partial public key output by PartialKeyExtract should be provided as input to SetPublicKey, which makes it impossible for the user to set a public key if he/she has not contacted the KGC and obtained a partial private/public pair.

We note that our model of CLPKE is slightly weaker than the one given in [1] as a user must authenticated himself/herself to the KGC and obtain an appropriate partial public key to create a public key, while the original CLPKE does not require a user to contact the KGC to set up his/her public keys. (As discussed in Section 1, one can view our CLPKE is close to the public key encryption in the normal PKI setting while Al-Riyami and Paterson's original CLPKE of is close to IBE).

However, we argue that *our CLPKE does not lose the unique property of CLPKE* that the use of certificates to guarantee the authenticity of public keys is not required any more, which is the main motivation for CLPKE. Below, we formally describe our model of CLPKE.

Definition 1 (CLPKE). A generic CLPKE (Certificateless Public Key Encryption) scheme, denoted by Π, consists of the following algorithms.

- Setup: The Key Generation Center (KGC) runs this algorithm to generate a common parameter $\texttt{params}$ and a master key $\texttt{masterKey}$. Note that $\texttt{params}$ is given to all interested parties. We write $(\texttt{params}, \texttt{masterKey}) = \mathsf{Setup}()$.
- PartialKeyExtract: Taking $\texttt{params}$, $\texttt{masterKey}$ and an identity $\texttt{ID}$ received from a user as input, the KGC runs this algorithm to generate a partial private key $D_{\texttt{ID}}$ and a partial public key $P_{\texttt{ID}}$. We write $(P_{\texttt{ID}}, D_{\texttt{ID}}) = \mathsf{PartialKeyExtract}(\texttt{params}, \texttt{masterKey}, \texttt{ID})$.
- SetSecretValue: Taking $\texttt{params}$ and $\texttt{ID}$ as input, the user runs this algorithm to generate a secret value $s_{\texttt{ID}}$. We write $s_{\texttt{ID}} = \mathsf{SetSecretValue}(\texttt{params}, \texttt{ID})$.

– SetPrivateKey: Taking `params`, $D_{\mathtt{ID}}$ and $s_{\mathtt{ID}}$ as input, the user runs this algorithm to generate a private key $SK_{\mathtt{ID}}$. We write $SK_{\mathtt{ID}} = \mathsf{SetPrivateKey}(\mathtt{params}, D_{\mathtt{ID}}, s_{\mathtt{ID}})$.
– SetPublicKey: Taking `params`, $P_{\mathtt{ID}}$, $s_{\mathtt{ID}}$ and `ID` as input, the user runs this algorithm to generate a public key $PK_{\mathtt{ID}}$. We write $PK_{\mathtt{ID}} = \mathsf{SetPublicKey}(\mathtt{params}, P_{\mathtt{ID}}, s_{\mathtt{ID}}, \mathtt{ID})$.
– Encrypt: Taking `params`, `ID`, $PK_{\mathtt{ID}}$, and a plaintext message M as input, a sender runs this algorithm to create a ciphertext C. We write $C = \mathsf{Encrypt}(\mathtt{params}, \mathtt{ID}, PK_{\mathtt{ID}}, M)$.
– Decrypt: Taking `params`, $SK_{\mathtt{ID}}$ and the ciphertext C as input, the user as a recipient runs this algorithm to get a decryption δ, which is either a plaintext message or a "*Reject*" message. We write $\delta = \mathsf{Decrypt}(\mathtt{params}, SK_{\mathtt{ID}}, C)$.

Security Notion. We also modify the security notion for the original CLPKE and present a new notion, which we call "indistinguishability of CLPKE ciphertexts under chosen ciphertext attack (IND-CLPKE-CCA)". We note that the modification is very small: In our security notion of CLPKE, the attacker's "public key request" queries should be answered by running the PartialKeyExtract algorithm, which is not needed in the original CLPKE.

Like the security notion for the original CLPKE, we assume two types of attackers A_I and A_{II}. A difference between these two attackers is that A_I does not have access to the master key of KGC while A_{II} does have. Now a formal definition follows.

Definition 2 (IND-CLPKE-CCA). Let A_I and A_{II} denote Type I attacker and Type II attacker respectively. Let Π be a generic CLPKE scheme. We consider two games "Game I" and "Game II" where A_I and A_{II} interact with their "Challenger" respectively. Note that the Challenger keeps a history of "query-answer" while interacting with the attackers.

Game I: This is the game in which A_I interacts with the "Challenger":

Phase I-1: The Challenger runs Setup() to generate `masterKey` and `params`. The Challenger gives `params` to A_I while keeping `masterKey` secret.
Phase I-2: A_I performs the following:

- Issuing partial key extraction queries, each of which is denoted by (`ID`, "partial key extract"): On receiving each of these queries, the Challenger computes $(P_{\mathtt{ID}}, D_{\mathtt{ID}}) = \mathsf{PartialKeyExtract}(\mathtt{params}, \mathtt{masterKey}, \mathtt{ID})$ and returns it to A_I.
- Issuing private key extraction queries, each of which is denoted by (`ID`, "private key extract"): On receiving each of these queries, the Challenger computes $(P_{\mathtt{ID}}, D_{\mathtt{ID}}) = \mathsf{PartialKeyExtract}(\mathtt{params}, \mathtt{masterKey}, \mathtt{ID})$ and $s_{\mathtt{ID}} = \mathsf{SetSecretValue}(\mathtt{params}, \mathtt{ID})$. It then computes $SK_{\mathtt{ID}} = \mathsf{SetPrivateKey}(\mathtt{params}, D_{\mathtt{ID}}, s_{\mathtt{ID}})$ and returns it to A_I.
- Issuing public key request queries, each of which is denoted by (`ID`, "public key request"): On receiving each of these queries, the Challenger computes $(P_{\mathtt{ID}}, D_{\mathtt{ID}}) = \mathsf{PartialKeyExtract}(\mathtt{params}, \mathtt{masterKey}, \mathtt{ID})$ and $s_{\mathtt{ID}} = \mathsf{SetSecretValue}(\mathtt{params}, \mathtt{ID})$. It then computes $PK_{\mathtt{ID}} = \mathsf{SetPublicKey}(\mathtt{params}, P_{\mathtt{ID}}, s_{\mathtt{ID}})$ and returns it to A_I.

- Replacing the User's public key: A_I replaces a public key $PK_{\mathtt{ID}}$ with its own at any time.
- Issuing decryption queries, each of which is denoted by ($\mathtt{ID}$, $PK_{\mathtt{ID}}$, C, "decryption"): On receiving such a query, the Challenger finds $SK_{\mathtt{ID}}$ from its "query-answer" list for public key request, computes $\delta = \mathsf{Decrypt}(\mathtt{params}, SK_{\mathtt{ID}}, C)$, which is either a plaintext message or a "*Reject*" message and returns δ to A_I. If the Challenger cannot find $SK_{\mathtt{ID}}$, it runs a special "knowledge extractor" to obtain a decryption δ and returns it to A_I. (As discussed in [1], it is not unreasonable to assume that the Challenger cannot answer a decryption query when a corresponding public key has been replaced, and hence returns "*Reject*". However, as also pointed out in [1]), replacing public keys gives a huge power to the attacker. Hence, we assume that the Challenger uses other means, called "knowledge extractor" [1], to decrypt a requested ciphertext. Note that a construction of the knowledge extractor is specific to each CLPKE scheme).

Phase I-3: A_I outputs two equal-length plaintext messages (M_0, M_1) and a target identity $\mathtt{ID}^*$. Note that $\mathtt{ID}^*$ has not been queried to extract a partial private key nor a private key at any time. Note also that $\mathtt{ID}^*$ cannot be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. On receiving (M_0, M_1) and $\mathtt{ID}^*$, the Challenger picks $\beta \in \{0,1\}$ at random and creates a target ciphertext $C^* = \mathsf{Encrypt}(\mathtt{params}, PK_{\mathtt{ID}^*}, M_\beta)$. The Challenger returns C^* to A_I.
Phase I-4: A_I issues queries as in Phase 2. The same rule the game applies here: $\mathtt{ID}^*$ has not been queried to extract a partial private key nor a private key at any time; $\mathtt{ID}^*$ cannot be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. Additionally, no decryption queries should be made on C^* for the combination of $\mathtt{ID}^*$ and $PK_{\mathtt{ID}^*}$ that was used to encrypt M_β.
Phase I-5: A_I outputs its guess $\beta' \in \{0,1\}$.

Game II: This is the game in which A_{II} interacts with the "Challenger":

Phase II-1: The Challenger runs $\mathsf{Setup}()$ to generate $\mathtt{masterKey}$ and $\mathtt{params}$. The Challenger gives $\mathtt{params}$ *and* $\mathtt{masterKey}$ to A_{II}.
Phase II-2: A_{II} performs the following:

- Computing partial key associated with $\mathtt{ID}$: A_{II} computes $(P_{\mathtt{ID}}, D_{\mathtt{ID}}) = \mathsf{PartialKeyExtract}(\mathtt{params}, \mathtt{masterKey}, \mathtt{ID})$.
- Issuing private key extraction queries, each of which is denoted by ($\mathtt{ID}$, "private key extract"): On receiving each of these queries, the Challenger computes $(P_{\mathtt{ID}}, D_{\mathtt{ID}}) = \mathsf{PartialKeyExtract}(\mathtt{params}, \mathtt{masterKey}, \mathtt{ID})$ and $s_{\mathtt{ID}} = \mathsf{SetSecretValue}(\mathtt{params}, \mathtt{ID})$. It then computes $SK_{\mathtt{ID}} = \mathsf{SetPrivateKey}(\mathtt{params}, D_{\mathtt{ID}}, s_{\mathtt{ID}})$ and returns it to A_{II}.
- Issuing public key request queries, each of which is denoted by ($\mathtt{ID}$, "public key request"): On receiving each of these queries, the Challenger computes $D_{\mathtt{ID}} = \mathsf{PartialKeyExtract}(\mathtt{params}, \mathtt{masterKey}, \mathtt{ID})$ and $s_{\mathtt{ID}} =$

$\mathsf{SetSecretValue}(\mathtt{params}, \mathtt{ID})$. It then computes $PK_{\mathtt{ID}} = \mathsf{SetPublicKey}(\mathtt{params}, P_{\mathtt{ID}}, s_{\mathtt{ID}})$ and returns it to A_{II}.

- Issuing decryption queries, each of which is denoted by ($\mathtt{ID}$, $PK_{\mathtt{ID}}$, C, "decryption"): On receiving each of these queries, the Challenger finds $SK_{\mathtt{ID}}$ from its "query-answer" list, computes $\delta = \mathsf{Decrypt}(\mathtt{params}, SK_{\mathtt{ID}}, C)$, which is either a plaintext message or a "*Reject*" message, and returns δ to A_{II}.

Phase II-3: A_{II} outputs two equal-length plaintext messages (M_0, M_1) and a target identity $\mathtt{ID}^*$. Note that $\mathtt{ID}^*$ has not been issued as a private key extraction query. On receiving (M_0, M_1) and $\mathtt{ID}^*$, the Challenger picks $\beta \in \{0, 1\}$ at random and creates a target ciphertext $C^* = \mathsf{Encrypt}(\mathtt{params}, PK_{\mathtt{ID}^*}, M_\beta)$. The Challenger returns C^* to A_{II}.
Phase II-4: A_{II} issues queries as in Phase 2 subject to the same rules. (That is, $\mathtt{ID}^*$ has not been issued as a private key extraction query). But in this phase, no decryption queries should be made on C^* for the combination of $\mathtt{ID}^*$ and $PK_{\mathtt{ID}^*}$ used to encrypt M_β.
Phase II-5: A_{II} outputs its guess $\beta' \in \{0, 1\}$.

We define A_i's guessing advantage in Game i, where $i \in \{I, II\}$, by $\mathbf{Adv}_{\Pi,\text{Game } i}^{\text{IND-CLPKE-CCA}}(A_i) = |\Pr[\beta' = \beta] - \frac{1}{2}|$. A_i breaks IND-CLPKE-CCA of Π with $(t, q_{paex}, q_{prex}, \epsilon)$ if and only if the guessing advantage of A_i that makes q_{paex} partial key extraction and q_{prex} private key extraction queries is greater than ϵ within running time t. The scheme Π is said to be $(t, q_{paex}, q_{prex}, \epsilon)$-IND-CLPKE-CCA secure if there is no attacker A_i that breaks IND-CLPKE-CCA of Π with $(t, q_{paex}, q_{prex}, \epsilon)$.

Computational Problem. We now review the *standard* "Computational Diffie-Hellman (CDH)" problem used in a large number of cryptographic schemes.

Definition 3 (CDH). Let p and q be primes such that $q|p-1$. Let g be a generator of $\mathbb{Z}_p^*$. Let A be an attacker. A tries to solve the following problem: *Given (g, g^a, g^b) for uniformly chosen $a, b, c \in \mathbb{Z}_q^*$, compute $\kappa = g^{ab}$.*

Formally, we define A's advantage $\mathbf{Adv}_{\mathbb{Z}_p^*}^{\text{CDH}}(A)$ by $\Pr[A(g, g^a, g^b) = g^{ab}]$. A solves the CDH problem with (t, ϵ) if and only if the advantage of A is greater than ϵ within running time t. The CDH problem is said to be (t, ϵ)-intractable if there is no attacker A that solves the CDH problem with (t, ϵ).

We remark that the current CLPKE schemes presented in [1] and [2] all depend on the "Bilinear Diffie-Hellman (BDH)" problem which is a *pairing* version of the CDH problem used in the construction of Boneh and Franklin's IBE scheme [5]. (Informally, the BDH problem is to compute $\hat{e}(g, g)^{abc}$ given g^a, g^b and g^c, where g is a generator, $\hat{e}$ denotes a bilinear pairing and a, b, c are chosen at random from $\mathbb{Z}_q^*$).

3 Our CLPKE Scheme

We now present our CLPKE scheme based on the Schnorr signature [15]. As mentioned previously, our CLPKE scheme is motivated by the construction of

PKI-enabled encryption scheme given in [6]. However, we apply this scheme non-trivially to construct an *efficient* CLPKE scheme: The computational cost for realizing our scheme is very low due to not only the efficiency brought from the Schnorr signature but also the effective method that combines the Schnorr signature and the public key encryption scheme. – We remark that the encryption algorithm of our CLPKE scheme requires two more modular exponentiations compared with the "hashed" ElGamal encryption transformed by the technique proposed by Fujisaki and Okamoto [8]; the decryption algorithm requires one more exponentiation compared with the same scheme. Below, we describe the scheme:

- $\mathsf{Setup}()$: Generate two primes p and q such that $q|p-1$. Pick a generator g of $\mathbb{Z}_p^*$. Pick $x \in \mathbb{Z}_q^*$ uniformly at random and compute $y = g^x$. Choose hash functions $H_1 : \{0,1\}^* \times \mathbb{Z}_q^* \to \mathbb{Z}_q^*$, $H_2 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \to \mathbb{Z}_q^*$ and $H_3 : \mathbb{Z}_p^* \times \mathbb{Z}_p^* \to \{0,1\}^l$, where $l = l_0 + l_1 \in \mathbb{N}$. Return $\mathtt{params} = (p, q, g, y, H_1, H_2, H_3)$ and $\mathtt{masterKey}{=}(p, q, g, x, H_1, H_2, H_3)$.
- $\mathsf{PartialKeyExtract}(\mathtt{params}, \mathtt{masterKey}, \mathtt{ID})$: Pick $s \in \mathbb{Z}_q^*$ at random and compute $w = g^s$ and $t = s + xH_1(\mathtt{ID}, w)$. Return $(P_{\mathtt{ID}}, D_{\mathtt{ID}}) = (w, t)$.
- $\mathsf{SetSecretValue}(\mathtt{params}, \mathtt{ID})$: Pick $z \in \mathbb{Z}_q^*$ at random. Return $s_{\mathtt{ID}} = z$.
- $\mathsf{SetPrivateKey}(\mathtt{params}, D_{\mathtt{ID}}, s_{\mathtt{ID}})$: Set $SK_{\mathtt{ID}} = (s_{\mathtt{ID}}, D_{\mathtt{ID}}) = (z, t)$. Return $SK_{\mathtt{ID}}$.
- $\mathsf{SetPublicKey}(\mathtt{params}, P_{\mathtt{ID}}, s_{\mathtt{ID}}, \mathtt{ID})$: Let $P_{\mathtt{ID}} = w$ and $s_{\mathtt{ID}} = z$. Compute $\mu = g^z$ and set $PK_{\mathtt{ID}} = (w, \mu)$. Return $PK_{\mathtt{ID}}$.
- $\mathsf{Encrypt}(\mathtt{params}, \mathtt{ID}, PK_{\mathtt{ID}}, M)$ where the bit-length of M is l_0: Parse $PK_{\mathtt{ID}}$ as (w, μ) and compute $\gamma_{\mathtt{ID}} = wy^{H_1(\mathtt{ID}, w)}$. Pick $\sigma \in \{0,1\}^{l_1}$ at random, and compute $r = H_2(M, \sigma)$. Compute $C = (c_1, c_2)$ such that

  $$c_1 = g^r; c_2 = H_3(k_1, k_2) \oplus (M||\sigma),$$

  where $k_1 = \mu^r$ and $k_2 = \gamma_{\mathtt{ID}}^r$. (Note that "$||$" denotes"concatenation". Note also that the bit-length of $(M||\sigma)$ equals to $l = l_0 + l_1$). Return C.
- $\mathsf{Decrypt}(\mathtt{params}, SK_{\mathtt{ID}}, C)$: Parse C as (c_1, c_2) and $SK_{\mathtt{ID}}$ as (z, t). Compute

  $$M||\sigma = H_2(c_1^z, c_1^t) \oplus c_2.$$

  If $g^{H_2(M,\sigma)} = c_1$, return M. Else return "*Reject*".

It can be easily seen that the above decryption algorithm is consistent: If $C = (c_1, c_2)$ is a valid cipheretxt, we obtain

$$\begin{aligned} H_2(c_1^z, c_1^t) \oplus c_2 &= H_2(g^{rz}, g^{rt}) \oplus H_2(\mu^r, \gamma_{\mathtt{ID}}^r) \oplus (M||\sigma) \\ &= H_2((g^z)^r, (g^{s+xH_1(\mathtt{ID},w)})^r) \oplus H_2(\mu^r, \gamma_{\mathtt{ID}}^r) \oplus (M||\sigma) \\ &= H_2(\mu^r, (g^s y^{H_1(\mathtt{ID},w)})^r) \oplus H_2(\mu^r, \gamma_{\mathtt{ID}}^r) \oplus (M||\sigma) \\ &= H_2(\mu^r, \gamma_{\mathtt{ID}}^r) \oplus H_2(\mu^r, \gamma_{\mathtt{ID}}^r) \oplus (M||\sigma) = M||\sigma. \end{aligned}$$

4 Security Analysis

Basically, the main idea of the security proofs given in this section is to have the CDH attacker B simulate the "environment" of the Type I and Type II attackers A_I and A_{II} respectively until it can compute a Diffie-Hellman key g^{ab} of g^a and g^b using the ability of A_I and A_{II}. As described in Definition 2, A_I and A_{II} will issue various queries such as random oracle, partial key extraction, public key request, private key extraction and decryption queries. B will respond to these queries with the answers identically distributed as those in the real attack.

We note that for the attacker A_I, B sets g^a as a part of the challenge ciphertext and g^b as a KGC's public key. On the other hand, for the attacker A_{II}, B sets g^a as a part of the challenge ciphertext but uses g^b to generate a public key associated with the challenge identity. The KGC's public key is set up as g^x where B knows random $x \in \mathbb{Z}_q^*$. This way, B can give the master key of the KGC to A_{II}.

We remark that care must be taken when the answers for the attackers' public key request queries are simulated. One reason is that a public key in our scheme is related to not only a private key but also partial private and public keys obtained from the KGC. The other reason is that during the attack, the attackers are entitled to see (or receive) any public keys even associated with the target identity. The proofs given in this section address these two issues.

Theorem 1. *The CLPKE scheme based on the Schnorr signature is IND-CLPKE-CPA secure in the random oracle model, assuming that the CDH problem is intractable.*

In order to prove the above theorem, we prove two lemmas. Lemma 1 shows that our CLPKE scheme is secure against the Type I attacker whose behavior is as described in Definition 2.

Lemma 1. *The CLPKE scheme based on the Schnorr signature is $(t, q_{H_1}, q_{H_2}, q_{H_3}, q_{paex}, q_{prex}, \epsilon)$-IND-CLPKE-CCA secure against the Type I attacker A_I in the random oracle model assuming that the CDH problem is (t', ϵ')-intractable, where $\varepsilon' > \frac{1}{q_{H_3}}\left(\frac{2\varepsilon}{e(q_{prex}+1)} - \frac{q_{H_2}}{2^{l_1}} - \frac{q_D q_{H_2}}{2^{l_1}} - \frac{q_D}{q}\right)$ and $t' > t + (q_{H_1} + q_{H_2})O(1) + q_{H_3}(2T_{EX} + O(1)) + (q_{paex} + q_{prex})(T_{EX} + O(1)) + q_D(2T_{EX} + O(1))$ where T_{EX} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$.*

Proof. Let A_I be an IND-CLPKE-CCA Type I attacker. The number of queries to the oracles that A_I makes and its running time are as defined in the above theorem statement. We show that using A_I, one can construct an attacker B that can solve the CDH problem (Definition 3).

Suppose that B is given (p, q, g, g^a, g^b) as an instance of the CDH problem. (Note that the number of queries to the oracles that B makes and its running time are as defined in the above theorem statement). B can simulate the Challenger's execution of each phase of IND-CLPKE-CCA game for A_I as follows.

[Simulation of Phase I-1] B sets $y = g^b$ and gives A_I (p, q, g, y, H_1, H_2, H_3) as `params`, where H_1, H_2 and H_3 are random oracles controlled by B as follows.

On receiving a query $(\texttt{ID}, w)$ to H_1:

1. If $\langle(\texttt{ID}, w), e\rangle$ exists in $\mathsf{H_1List}$, return e as answer.
2. Otherwise, pick $e \in \mathbb{Z}_q^*$ at random, add $\langle(\texttt{ID}, w), e\rangle$ to $\mathsf{H_1List}$ and return e as answer.

On receiving a query (M, σ) to H_2:

1. If $\langle(M, \sigma), r\rangle$ exists in $\mathsf{H_2List}$, return r as answer.
2. Otherwise, pick $r \in \mathbb{Z}_q^*$ at random, add $\langle(M, \sigma), r\rangle$ to $\mathsf{H_2List}$ and return r as answer.

On receiving a query (k_1, k_2) to H_3:

1. If $\langle(k_1, k_2), R\rangle$ exists in $\mathsf{H_3List}$, return R as answer.
2. Otherwise, pick $R \in \{0,1\}^l$ at random, add $\langle(k_1, k_2), R\rangle$ to $\mathsf{H_3List}$ and return R as answer.

[Simulation of Phase I-2] B answers A_I's queries as follows.

On receiving a partial key extraction query ($\texttt{ID}$, "partial key extract"):

1. If $\langle\texttt{ID}, (w, t)\rangle$ exists in $\mathsf{PartialKeyList}$, return (w, t) as answer.
2. Otherwise, do the following:
   (a) Pick $t, e \in \mathbb{Z}_q^*$ at random and compute $w = g^t y^{-e}$; add $\langle(\texttt{ID}, w), e\rangle$ to $\mathsf{H_1List}$ (That it, e is defined to be $H_1(\texttt{ID}, w)$.) and $\langle\texttt{ID}, (w, t)\rangle$ to $\mathsf{PartialKeyList}$; return (w, t) as answer.

Note from the above simulation that we have $wy^{H_1(\texttt{ID},w)} = g^t y^{-e} y^e = g^t$, which holds in the real attack too.
On receiving a public key request query ($\texttt{ID}$, "public key request"):

1. If $\langle\texttt{ID}, (w, \mu), coin\rangle$ exists in $\mathsf{PublicKeyList}$, return $PK_{\texttt{ID}} = (w, \mu)$ as answer.
2. Otherwise, pick $coin \in \{0, 1\}$ so that $\Pr[coin = 0] = \delta$. (δ will be determined later).
3. If $coin = 0$, do the following:
   (a) If $\langle\texttt{ID}, (w, t)\rangle$ exists in $\mathsf{PartialKeyList}$, pick $z \in \mathbb{Z}_q^*$ at random and compute $\mu = g^z$; add $\langle\texttt{ID}, (z, t)\rangle$ to $\mathsf{PrivateKeyList}$ and $\langle\texttt{ID}, (w, \mu), coin\rangle$ to $\mathsf{PublicKeyList}$; return $PK_{\texttt{ID}} = (w, \mu)$ as answer.
   (b) Otherwise, run the above simulation algorithm for partial key extraction taking $\texttt{ID}$ as input to get a partial key (w, t); pick $z \in \mathbb{Z}_q^*$ at random and compute $\mu = g^z$; add $\langle\texttt{ID}, (z, t)\rangle$ to $\mathsf{PrivateKeyList}$ and $\langle\texttt{ID}, (w, \mu), coin\rangle$ to $\mathsf{PublicKeyList}$; return $PK_{\texttt{ID}} = (w, \mu)$ as answer.
4. Otherwise (if $coin = 1$), pick $s, z \in \mathbb{Z}_q^*$ at random and compute $w = g^s$ and $\mu = g^z$; add $\langle\texttt{ID}, (z, ?), s\rangle$ to $\mathsf{PrivateKeyList}$ and $\langle\texttt{ID}, (w, \mu), coin\rangle$ to $\mathsf{PublicKeyList}$; return $PK_{\texttt{ID}} = (w, \mu)$ as answer.

On receiving a private key extraction query ($\texttt{ID}$, "private key extract"):

1. Run the above simulation algorithm for public key request taking $\mathtt{ID}$ as input to get a tuple $\langle \mathtt{ID}, (w, \mu), \mathit{coin}\rangle \in \mathsf{PublicKeyList}$.
2. If $\mathit{coin} = 0$, search $\mathsf{PrivateKeyList}$ for a tuple $\langle \mathtt{ID}, (z, t)\rangle$ and return $SK_{\mathtt{ID}} = (z, t)$ as answer.
3. Otherwise, return "*Abort*" and terminate.

On receiving a decryption query ($\mathtt{ID}$, $PK_{\mathtt{ID}}$, C, "decryption "), where $C = (c_1, c_2)$ and $PK_{\mathtt{ID}} = (w, \mu)$:

1. Search $\mathsf{PublicKeyList}$ for a tuple $\langle \mathtt{ID}, (w, \mu), \mathit{coin}\rangle$.
2. If such a tuple exists and $\mathit{coin} = 0$
   (a) Search $\mathsf{PrivateKeyList}$ for a tuple $\langle \mathtt{ID}, (z, t)\rangle$. (Note that from the simulation of public key request, $\langle \mathtt{ID}, (z, t)\rangle$ must exist in $\mathsf{PrivateKeyList}$ as long as one can find $\langle \mathtt{ID}, (w, \mu), \mathit{coin}\rangle$ with $\mathit{coin} = 0$ in $\mathsf{PublicKeyList}$).
   (b) Compute $M||\sigma = H_3(c_1^z, c_1^t) \oplus c_2$.
   (c) If $c_1 = g^{H_2(M,\sigma)}$, return M and "*Reject*" otherwise.
3. Else if such a tuple exists and $\mathit{coin} = 1$
   (a) Run the above simulation algorithm for H_1 to get a tuple $\langle(\mathtt{ID}, w), e\rangle$.
   (b) If there exist $\langle(M, \sigma), r\rangle \in \mathsf{H_2List}$ and $\langle(k_1, k_2), R\rangle \in \mathsf{H_3List}$ such that

$$c_1 = g^r, c_2 = R \oplus (M||\sigma), k_1 = \mu^r \text{ and } k_2 = \gamma_{\mathtt{ID}}^r,$$

   where $\gamma_{\mathtt{ID}} = wy^e$, return M and "*Reject*" otherwise. We remark that the pair $\langle(M, \sigma), r\rangle$ that satisfies the above condition uniquely exists in $\mathsf{H_2List}$ as the encryption function is injective with respect to $(\mathtt{ID}, w)$.
4. Else if such a tuple does not exist (This is the case when the public key of a target user is replaced by A_I)
   (a) Run the above simulation algorithm for H_1 to get a tuple $\langle(\mathtt{ID}, w), e\rangle$.
   (b) If there exist $\langle(M, \sigma), r\rangle \in \mathsf{H_2List}$ and $\langle K, R\rangle \in \mathsf{H_3List}$ such that

$$c_1 = g^r, c_2 = R \oplus (M||\sigma), k_1 = \mu^r \text{ and } k_2 = \gamma_{\mathtt{ID}}^r,$$

   where $\gamma_{\mathtt{ID}} = wy^e$, return M and "*Reject*" otherwise.

[Simulation of Phase I-3] B answers A_I's queries as follows.
On receiving a challenge query ($\mathtt{ID}^*$, (M_0, M_1)):

1. Run the above simulation algorithm for public key request taking $\mathtt{ID}^*$ as input to get a tuple $\langle \mathtt{ID}^*, (w^*, \mu^*), \mathit{coin}\rangle \in \mathsf{PublicKeyList}$.
2. If $\mathit{coin} = 0$ return "*Abort*" and terminate.
3. Otherwise, do the following:
   (a) Search $\mathsf{PrivateKeyList}$ for a tuple $\langle \mathtt{ID}^*, (z^*, ?), s^*\rangle$.
      – In this case, we know that $\mu^* = g^{z^*}$ and $w^* = g^{s^*}$.
   (b) Pick $\sigma^* \in \{0, 1\}^{l_1}$, $c_2^* \in \{0, 1\}^l$ and $\beta \in \{0, 1\}$ at random.
   (c) Set $c_1^* = g^a$, $\gamma_{\mathtt{ID}^*} = w^* y^{e^*}$ and $e^* = H_1(\mathtt{ID}^*, w^*)$.
   (d) Define $a = H_2(M_\beta, \sigma^*)$ and $H_3(\mu^{*a}, \gamma_{\mathtt{ID}^*}^a) = c_2^* \oplus (M_\beta||\sigma^*)$. (Note that B does not know "a").

4. Return $C^* = (c_1^*, c_2^*)$ as a target ciphertext.
* Note that by the construction given above, $c_2^* = H_3(\mu^{*a}, \gamma_{\mathtt{ID}^*}^a) \oplus (M_\beta || \sigma^*) = H_3(g^{az^*}, g^{a(s^* + bH_1(\mathtt{ID}^*, g^{s^*}))}) \oplus (M_\beta || \sigma^*)$.

[Simulation of Phase I-4] In this phase, B answers A_I's queries in the same way as it did in Phase I-2. Note that $\mathtt{ID}^*$ cannot be issued as a partial key extraction query and a private key extraction query while A_I can freely replace public keys. Note also that no decryption queries should be made on C^* for the combination of $\mathtt{ID}^*$ and $PK_{\mathtt{ID}^*} = (w^*, \mu^*)$ that was used to encrypt M_β. The decryption queries can be answered in the same way as in Phase 2. We just repeat the following important case:

On receiving a decryption query $(\mathtt{ID}^*, PK_{\mathtt{ID}^*}\ C)$, where $C = (c_1, c_2)$ and $PK_{\mathtt{ID}^*} = (w^*, \mu^*)$ (In this case, we know that $\langle \mathtt{ID}^*, (w^*, \mu^*), coin \rangle$ exists in $\mathsf{PublicKeyList}$ with $coin = 1$).:

– If there exist $\langle (\mathtt{ID}^*, w^*), e^* \rangle \in \mathsf{H_1List}$, $\langle (M, \sigma), r \rangle \in \mathsf{H_2List}$ and $\langle (k_1, k_2), R \rangle \in \mathsf{H_3List}$ such that

$$c_1 = g^r, c_2 = R \oplus (M || \sigma)\ , k_1 = \mu^r \text{ and } k_2 = \gamma_{\mathtt{ID}^*}^r,$$

where $\gamma_{\mathtt{ID}^*} = w^* y^{e^*}$, return M and "*Reject*" otherwise. Again, we remark that the pair $\langle (M, \sigma), r \rangle$ that satisfies the above condition uniquely exists in $\mathsf{H_2List}$ as the encryption function is injective with respect to $(\mathtt{ID}^*, w)$.

[Simulation of Phase I-5] When A_I outputs its β', B returns the set

$$\mathcal{S} = \{ \left(\frac{k_{2i}}{g^{as^*}} \right)^{1/e^*} | k_{2i} \text{ is the second component of queries to } H_3 \text{ for } i \in [1, q_{H_3}]$$

such that $e^* = H_1(\mathtt{ID}^*, w^*)$ and $k_{1i} = g^{az^*}$ where k_{2i} is the first component of queries to $H_3\}$.

[Analysis] We first evaluate the simulations of the random oracles given above. From the construction of H_1, it is clear that the simulation of H_1 is perfect. As long as A_I does not query (M_β, σ^*) to H_2 nor $g^{az^*} (\stackrel{\text{def}}{=} \mu^*)$ and $g^{a(s^*+be^*)} (\stackrel{\text{def}}{=} (w^* y^{e^*})^a)$ to H_3, where σ^*, z^* and s^* are chosen by B in Phase I-3 and $e^* \stackrel{\text{def}}{=} H_1(\mathtt{ID}^*, g^{s^*})$, the simulations of H_2 and H_3 are perfect. By AskH_3^* we denote the event that $(\mu^{*a}, (w^* y^{e^*})^a)$ has been queried to H_3. Also, by AskH_2^* we denote the event that (M_β, σ^*) has been queried to H_2.

Next, one can notice that the simulated target ciphertext is identically distributed as the real one from the construction.

Now, we evaluate the simulation of the decryption oracle. If a public key $PK_{\mathtt{ID}}$ has not been replaced nor $PK_{\mathtt{ID}}$ has not been produced under $coin = 1$, the simulation is perfect as B knows the private key $SK_{\mathtt{ID}}$ corresponding to $PK_{\mathtt{ID}}$. Otherwise, simulation errors may occur while B running the decryption oracle simulator specified above. However, these errors are not significant as shown below: Suppose that $(\mathtt{ID}, PK_{\mathtt{ID}}, C)$, where $C = (c_1, c_2)$ and $PK_{\mathtt{ID}} = (w, \mu)$, has been issued as a *valid* decryption query. Even if C is valid, there is a possibility that C can be produced without querying $(\mu^r, (wy^e)^r)$ to H_3, where $e = H_1(\mathtt{ID}, w)$ and $r = H_2(M, \sigma)$. Let Valid be an event that C is valid. Let $\mathsf{AskH_3}$ and $\mathsf{AskH_2}$ respectively be events that $(\mu^r, (wy^e)^r)$ has been queried to H_3 and (M, σ) has been queried to H_2 with respect to $C = (c_1, c_2) = (g^r, (M||\sigma) \oplus H_3(\mu^r, (wy^e)^r)$ and $PK_{\mathtt{ID}} = (w, \mu)$, where $r = H_2(M, \sigma)$ and $e = H_1(\mathtt{ID}, w)$. We then have

$$\begin{aligned}\Pr[\mathsf{Valid}|\neg\mathsf{AskH_3}] &\le \Pr[\mathsf{Valid} \wedge \mathsf{AskH_2}|\neg\mathsf{AskH_3}] + \Pr[\mathsf{Valid} \wedge \neg\mathsf{AskH_2}|\neg\mathsf{AskH_3}] \\ &\le \Pr[\mathsf{AskH_2}|\neg\mathsf{AskH_3}] + \Pr[\mathsf{Valid}|\neg\mathsf{AskH_2} \wedge \neg\mathsf{AskH_3}] \\ &\le \frac{q_{H_2}}{2^{l_1}} + \frac{1}{q}.\end{aligned}$$

Let DecErr be an event that $\mathsf{Valid}|\neg\mathsf{AskH_3}$ happens during the entire simulation. Then, since q_D decryption oracle queries are made, we have $\Pr[\mathsf{DecErr}] \le \frac{q_D q_{H_2}}{2^{l_1}} + \frac{q_D}{q}$.

Now define an event E to be $(\mathsf{AskH_3^*} \vee (\mathsf{AskH_2^*}|\neg\mathsf{AskH_3^*}) \vee \mathsf{DecErr})|\neg\mathsf{Abort}$, where Abort denotes an event that B aborts during the simulation. (Notice that $\mathsf{AskH_2^*}$ and $\mathsf{AskH_3^*}$ are as defined above in the beginning).

If E does not happen, it is clear that A_I does not gain any advantage greater than $1/2$ to guess β due to the randomness of the output of the random oracle H_3. Namely, we have $\Pr[\beta' = \beta|\neg\mathsf{E}] \le \frac{1}{2}$. Hence, by splitting $\Pr[\beta' = \beta]$, we obtain

$$\begin{aligned}\Pr[\beta' = \beta] &= \Pr[\beta' = \beta|\neg\mathsf{E}]\Pr[\neg\mathsf{E}] + \Pr[\beta' = \beta|\mathsf{E}]\Pr[\mathsf{E}] \\ &\le \frac{1}{2}\Pr[\neg\mathsf{E}] + \Pr[\mathsf{E}] = \frac{1}{2} + \frac{1}{2}\Pr[\mathsf{E}]\end{aligned}$$

and

$$\Pr[\beta' = \beta] \ge \Pr[\beta' = \beta|\neg\mathsf{E}]\Pr[\neg\mathsf{E}] = \frac{1}{2} - \frac{1}{2}\Pr[\mathsf{E}].$$

By definition of ε, we then have

$$\begin{aligned}\varepsilon &< \big|\Pr[\beta' = \beta] - \frac{1}{2}\big| \le \frac{1}{2}\Pr[\mathsf{E}] \\ &\le \frac{1}{2\Pr[\neg\mathsf{Abort}]}\Big(\Pr[\mathsf{AskH_3^*}] + \Pr[\mathsf{AskH_2^*}|\neg\mathsf{AskH_3^*}] + \Pr[\mathsf{DecErr}]\Big).\end{aligned}$$

First, notice that the probability that B does not abort during the simulation (the probability that $\neg\mathsf{Abort}$ happens) is given by $\delta^{q_{prv}}(1-\delta)$ which is maximized at $\delta = 1 - 1/(q_{prex}+1)$. Hence we have $\Pr[\neg\mathsf{Abort}] \le \frac{1}{e(q_{prex}+1)}$, where e denotes the base of the natural logarithm.

Since $\Pr[\mathsf{AskH_2^*}|\neg\mathsf{AskH_3^*}] \le \frac{q_{H_2}}{2^{l_1}}$ and $\Pr[\mathsf{DecErr}] \le \frac{q_D q_{H_2}}{2^{l_1}} + \frac{q_D}{q}$, we obtain

$$\Pr[\mathsf{AskH_3^*}] \ge \frac{2\varepsilon}{e(q_{prex}+1)} - \frac{q_{H_2}}{2^{l_1}} - \frac{q_D q_{H_2}}{2^{l_1}} - \frac{q_D}{q}.$$

Meanwhile, if AskH_3^* happens then B will be able to solve the CDH problem by picking $\left(\frac{k_{2i}}{g^{as^*}}\right)^{1/e^*}$ from the set $\mathcal{S}$ defined in the simulation of Phase I-5. Hence we have $\varepsilon' \geq \frac{1}{q_{H_3}} \Pr[\mathsf{AskH}_3^*]$. Consequently, we obtain

$$\varepsilon' > \frac{1}{q_{H_3}}\Big(\frac{2\varepsilon}{e(q_{prex}+1)} - \frac{q_{H_2}}{2^{l_1}} - \frac{q_D q_{H_2}}{2^{l_1}} - \frac{q_D}{q}\Big).$$

The running time of the CDH attacker B is $t' > t + (q_{H_1} + q_{H_2})O(1) + q_{H_3}(2T_{EX} + O(1)) + O(1)) + (q_{paex} + q_{prex})(T_{EX} + O(1)) + q_D(2T_{EX} + O(1))$ where T_{EX} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$.

The following lemma shows that our CLPKE scheme is secure against the Type II attacker.

Lemma 2. *The CLPKE scheme based on the Schnorr signature is $(t, q_{H_1}, q_{H_2}, q_{H_3}, q_{paex}, q_{prex}, \epsilon)$-IND-CLPKE-CCA secure against the Type II attacker A_{II} in the random oracle model assuming that the CDH problem is (t', ϵ')-intractable, where $\varepsilon' > \frac{1}{q_{H_3}}\left(\frac{2\varepsilon}{e(q_{prex}+1)} - \frac{q_{H_2}}{2^{l_1}} - \frac{q_D q_{H_2}}{2^{l_1}} - \frac{q_D}{q}\right)$ and $t' > t + (q_{H_1} + q_{H_2})O(1) + q_{H_3}(T_{EX} + O(1)) + (q_{paex} + q_{prex})(T_{EX} + O(1)) + q_D(2T_{EX} + O(1))$ where T_{EX} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$.*

Due to lack of space, the proof of the above theorem is given in the full version of this paper.

5 Concluding Remarks

We have presented the first CLPKE scheme that does not depend on the pairing. We have proven in the random oracle that that the scheme is IND-CLPKE-CCA-secure (Definition 2), relative to the hardness of the standard CDH problem.

We remark that one may also construct a "Certificate-Based Encryption (CBE) [9]" scheme without pairing using a similar technique presented in this paper, which will be our future work.

Acknowledgement

The authors are grateful to the anonymous referees for their helpful comments.

References

1. S. Al-Riyami and K. Paterson, *Certificateless Public Key Cryptography*, A full version, A short version appeared at Asiacrypt 2003, LNCS 2894, pp. 452–473, Springer-Verlag, 2003.
2. S. Al-Riyami and K. Paterson, *CBE from CLPKE: A Generic Construction and Efficient Schemes*, In PKC '05, LNCS 3386, pp. 398–415, Springer-Verlag, 2005.

3. D. Balfanz, G. Durfee, N. Shankar, D. K. Smetters, J. Staddon, and H.C. Wong, *Secret Handshakes from Pairing-Based Key Agreements*, In IEEE Symposium on Security and Privacy '03, pp. 180–196, IEEE Press, 2003.
4. M. Bellare and P. Rogaway, *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*, In ACM CCCS '93, pp. 62–73, 1993.
5. D. Boneh and M. Franklin, *Identity-Based Encryption from the Weil Pairing*, In Crypto '01, LNCS 2139, pp. 213–229, Springer-Verlag, 2001.
6. C. Castellucia, S. Jarecki and G. Tsudik, *Secret Handshake from CA-Oblivious Encryption*, In Asiacrypt '04, LNCS 3329, pp. 293–307, Springer-Verlag, 2004.
7. C. Cocks, *An Identity Based Encryption Scheme Based on Quadratic Residues*, In IMA '01, LNCS 2260, pp. 360–363, Springer-Verlag, 2001.
8. E. Fujisaki and T. Okamoto, *Secure Integration of Asymmetirc and Symmetric Encryption Schemes*, In Crypto '99, LNCS 1666, pp. 537–554, Springer-Verlag, 1999.
9. C. Gentry, *Certificate-Based Encryption and the Certificate Revocation Problem*, In Eurocrypt '03, LNCS 2656, pp. 272–293, Springer-Verlag, 2003.
10. M. Girault, *Self Certified Public Keys*, In Eurocrypt '91, LNCS 547, pp. 490–497, Springer-Verlag, 1992.
11. J. Holt, R. Bradshaw, K. E. Seamons and H. Orman, *Hidden Credentials*, In ACM Workshop on Privacy in the Electronic Society (WPES) '03, pp. 1–8, ACM Press, 2003.
12. MIRACL, Multiprecision Integer and Rational Arithmetic C/C++ Library, http://indigo.ie/ mscott/
13. H. Petersen and P. Horster, *Self-Certified Keys – Concepts and Applications*, In International Conference on Communications and Multimedia Security, Chapman and Hall, 1997.
14. S. Saeednia, *Identity-Based and Self-Certified Key-Exchange Protocols*, In ACISP '97, LNCS 1270, pp. 303–313, Springer-Verlag, 1997.
15. C. P. Schnorr, *Efficient Identifications and Signatures for Smart Cards*, In Crypto '89, LNCS 435, pp. 239–251, Springer-Verlag, 1990.
16. A. Shamir: *Identity-based Cryptosystems and Signature Schemes*, In Crypto '84, LNCS 196, pp. 47–53, Springer-Verlag, 1984.
17. D. Yum and P. Lee, *Generic Construction of Certificateless Encryption*, In ICCSA '04, LNCS 3043, pp. 802–811, Springer-Verlag, 2004.