Self-Generated-Certificate Public Key Encryption Without Pairing

Junzuo Lai[1] and Weidong Kou[2]

[1] Department of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai 200030, China
laijunzuo@sjtu.edu.cn
[2] School of Computer Science and Technology
Xi Dian University, Xi'an 710071, China
kou_weidong@yahoo.com.cn

Abstract. Certificateless Public Key Cryptography (CL-PKC) 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). However, it does suffer to the Denial-of-Decryption (DoD) Attack called by Liu and Au [1], as its nature is similar to the well known Denial-of-Service (DoS) Attack. Based on CL-PKC, they introduced a new paradigm called Self-Generated-Certificate Public Key Cryptography (SGC-PKC) that captured the DoD Attack and proposed a first scheme derived from a novel application of Water's Identity-Based Encryption scheme. In this paper, we propose a new SGC-PKE scheme that does not depend on the bilinear pairings, which make it be more efficient and more short public keys than Liu and Au's scheme. More importantly, our scheme reaches Girault's trusted level 3 (cf. Girault's trusted level 2 of Liu and Au's scheme), the same level as is enjoyed in a traditional PKI.

Key words: Certificateless Public Key Cryptography, Self-Generated-Certificate Public Key Cryptography, Self-Certified-Key

1 Introduction

In traditional Public Key Cryptography (PKC), each user selects his own private key and computes the corresponding public key, which is published. If a user wants to send an encrypted message to other user, he needs to know the user's public key. However, it is easy to suffer from the man-in-the-middle attack. To address this threat, there is a need to provide an assurance to the user about the relationship between a public key and the identity (or authority) of the holder of the corresponding private key. In a traditional Public Key Infrastructure (PKI), this assurance is delivered in the form of certificate, essentially a signature by a Certification Authority (CA) on a public key. However, a PKI faces with many challenges in the practice, such as revocation, storage and distribution of certificates.

Identity-Based Public Key Cryptography (ID-PKC), first proposed by Shamir [13], tackles the problem of authenticity of keys in a different way to traditional PKI. In ID-PKC, a user's public key is derived directly from certain aspects of its identity, for example, an IP address belonging to a network host, or an e-mail address associated with a user. Private keys are generated for entities by a trusted third party called a Private Key Generator (PKG). In this way, the certificate is provided implicitly due to the fact that the user will not have the ability of performing any cryptographic operations, if he hasn't obtained a correct private key associated with the published identity. The only disadvantage of ID-PKC is an unconditional trust to the PKG, which results that PKG can impersonate any user, or decrypt any ciphertext.

In order to solve for the above problem, Certificateless Public Key Cryptography (CL-PKC) was introduced by Al-Riyami and Paterson [2,3]. It is a new paradigm which lies between Identity-Based Cryptography and traditional Public Key Cryptography. The concept is to eliminate the inherent key-escrow problem of Identity-Based Cryptography (IBC). At the same time, it preserves the attractive advantage of IBC which is the absence of digital certificates (issued by Certificate Authority) and their important management overhead. Different from IBC, 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. A crucial difference between them is that the public key in CL-PKC 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 user's private keys since they contain secret information generated by the users themselves, thereby removing the escrow problem in IBC.

It seems that CL-PKC can solve the problem of explicit certification. Nevertheless it suffers Denial-of-Decryption (DoD) Attack called by Liu and Au [1]. Suppose Alice wants to send an encrypted message to Bob. She takes Bob's public key and his identity (or personal information) as input to the encryption function. However, Carol, the adversary, has replaced Bob's public key by someone's public key. Although Carol cannot decrypt the ciphertext, Bob also cannot decrypt the message while Alice is unaware of this. This is similar to Denial of Service (DoS) Attack in the way that the attacker cannot gain any secret information but precluding others from getting the normal service.

Liu and Au [1] propose a new paradigm called Self-Generated-Certificate Public Key Cryptography (SGC-PKC) to defend the above attack while preserving all advantages of Certificateless Public Key Cryptography. Similar to CL-PKC, every user is given a partial secret key by the KGC and generates his own secret key and corresponding public key. In addition, he also needs to generate a certificate using his own secret key. The purpose of this self-generated certificate is similar to the one in traditional PKC. That is, to bind the identity (or personal information) and the public key together. The main difference is that, it can be verified by using the user's identity and public key only and does not require any trusted party. It is implicitly included in the user's public key. If

Carol uses her public key to replace Alice's public key (or certificate), Bob can be aware of this and he may ask Alice to send him again her public key for the encryption.

Related Work. Al-Riyami and Paterson [2,3] introduced Certificateless Public Key Cryptography and proposed a CL-encryption scheme and a CL-signature scheme. Some concrete efficient implementations were proposed in [8,9]. In addition, some generic construction were proposed in [7,5,6].

In [4], Baek et al. proposed a CL-encryption scheme without pairing, which was related to the early works on the self-certified keys [10,11]. However, their scheme can't be converted to SGC-PKE directly and only reaches Girault's trusted level 2. We modify their scheme to get a new CL-encryption scheme without pairing. Our scheme can be converted to SGC-PKE directly and reaches Girault's trusted level 3, which makes our scheme more appealing. Our works are related to the works on Self-Certificate-PKI [12].

Liu and Au proposed the first SGC-PKE scheme in [1], which defends the DoD attack that exists in CL-PKE. However, their scheme is based on a CL-encryption scheme and a CL-signature scheme that are using the same set of public parameters and user key generation algorithm. In addition, their scheme has long public keys due to their CL-PKC derived from a novel application of Water's Identity-Based Encryption scheme and only reaches Girault's trusted level 2. All there make their scheme impractical.

Contribution. In this paper, we propose a SGC-PKE scheme without pairing and prove that it is secure in a fully adaptive adversarial model, provided that the standard Computational Diffie-Hellman (CDH) problem is hard. Compared with the first scheme, our scheme is more efficient, has short public keys and reaches Girault's trusted level 3, which makes our scheme more practical.

Organization. The rest of the paper is organized as follow. We give some definitions in Section 2. We propose a CL-encryption scheme in Section 3. The proposed SGC-PKE scheme is presented in Section 4. We compare our SGC-PKE scheme to Liu and Au's scheme in Section 5. Finally a concluding remark is given in Section 6.

2 Definition

In this section we first introduce our model of CL-PKE and its security definition. Next, we recall the security definition of SGC-PKE defined by Liu and Au [1].

2.1 Certificateless Public Key Encryption

Our model of CL-PKE is similar to that of Baek et al. [4]. Only slight difference lies in our model. However, it is the crucial point that makes our scheme reach Girault's trusted level 3 and is easy to be converted to SGC-PKE. Below, we formally describe our model of CL-PKE.

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

- **Setup:** is a probabilistic polynomial time (PPT) algorithms run by a Key Generation Center (KGC), given a security parameter k as input, outputs a randomly chosen master secret **mk** and a list of public parameter **param**. We write (**mk**, **param**) = **Setup** (k).
- **UserKeyGeneration:** is PPT algorithm, run by the user, given a list of public parameters **param** as inputs, outputs a secret key **sk** and a public key **pk**. We write (**sk**, **pk**) = **UserKeyGeneration** (**param**).
- **PartialKeyExtract:** Taking **param**, **mk**, a user's identity $\mathtt{ID}$ and **pk** received from the user, the KGC runs this PPT algorithm to generate a partial private key $D_{\mathtt{ID}}$ and a partial public key $P_{\mathtt{ID}}$. We write ($P_{\mathtt{ID}}$, $D_{\mathtt{ID}}$) = **PartialKeyExtract** (**param**, **mk**, $\mathtt{ID}$, **pk**).
- **SetPrivateKey:** Taking **param**, $D_{\mathtt{ID}}$ and **sk** as input, the user runs this PPT algorithm to generate a private key $SK_{\mathtt{ID}}$. We write $SK_{\mathtt{ID}}$ = **SetPrivateKey** (**param**, $D_{\mathtt{ID}}$, **sk**).
- **SetPublicKey:** Taking **param**, $P_{\mathtt{ID}}$ and **pk** as input, the user runs this PPT algorithm to generate a public key $PK_{\mathtt{ID}}$. We write $PK_{\mathtt{ID}}$ = **SetPublicKey** (**param**, $P_{\mathtt{ID}}$, **pk**).
- **Encrypt:** Taking a plaintext M, list of parameters **param**, a receiver's identity $\mathtt{ID}$ and $PK_{\mathtt{ID}}$ as inputs, a sender runs this PPT algorithm to create a ciphertext C. We write C = **Encrypt** (**param**, $\mathtt{ID}$, $PK_{\mathtt{ID}}$, M).
- **Decrypt:** Taking **param**, $SK_{\mathtt{ID}}$, the ciphertext C as inputs, the user as a recipient runs this deterministic algorithm to get a decryption δ, which is either a plaintext message or a "Reject" message. We write δ = **Decrypt** (**param**, $SK_{\mathtt{ID}}$, C).

For correctness, as usual we require that **Decrypt** (**param**, $SK_{\mathtt{ID}}$, C) = M whenever C = **Encrypt** (**param**, $\mathtt{ID}$, $PK_{\mathtt{ID}}$, M).

The function of **UserKeyGeneration** algorithm is the same as the **SetSecretValue** algorithm in Baek's definition. However, note that the **UserKeyGeneration** algorithm in our definition must run precede the **PartialKeyExtract** algorithm, compared with the **PartialKeyExtract** algorithm can run precede **SetSecretValue** algorithm in Baek's definition. We emphasize that this is the crucial point to make our scheme desirable.

Security Model. According to the original scheme in [2], there are two types of adversaries. Type I adversary does not have the KGC's mater secret key but it can replace public keys of arbitrary identities with other public keys of its own choices. It can also obtain partial and full secret keys of arbitrary identities.

Type II adversary knows the master secret key (hence it can compute partial secret key by itself). It is still allowed to obtain full secret key for arbitrary identities but is not allowed to replace public keys at any time.

Definition 2 (IND-CCA Security). *A Certificateless Public Key Encryption scheme Π is IND-CCA secure if no PPT adversary $\mathcal{A}$ of Type I or Type II has a non-negligible advantage in the following game played against the challenger:*

1. The challenger takes a security parameter k and runs the **Setup** algorithm. It gives $\mathcal{A}$ the resulting system parameters **param**. If $\mathcal{A}$ is of Type I, the challenger keeps the master secret key **mk** to itself, otherwise, it gives **mk** to $\mathcal{A}$.
2. $\mathcal{A}$ is given access to the following oracles:
   - **Public-Key-Request-Oracle:** on input a user's identity $\mathtt{ID}$, it computes (**sk**, **pk**) = **UserKeyGeneration** (**param**) and ($P_{\mathtt{ID}}$, $D_{\mathtt{ID}}$) = **PartialKeyExtract** (**param**, **mk**, $\mathtt{ID}$, **pk**). It then computes $PK_{\mathtt{ID}}$ = **SetPublicKey** (**param**, $P_{\mathtt{ID}}$, **pk**) and returns it to $\mathcal{A}$.
   - **Partial-Key-Extract-Oracle:** on input a user's identity $\mathtt{ID}$ and **pk**, it computes ($P_{\mathtt{ID}}$, $D_{\mathtt{ID}}$) = **PartialKeyExtract** (**param**, **mk**, $\mathtt{ID}$, **pk**) and returns it to $\mathcal{A}$. (Note that it is only useful to Type I adversary.)
   - **Private-Key-Request-Oracle:** on input a user's identity $\mathtt{ID}$, it computes (**sk**, **pk**) = **UserKeyGeneration** (**param**) and ($P_{\mathtt{ID}}$, $D_{\mathtt{ID}}$) =**PartialKeyExtract** (**param**, **mk**, $\mathtt{ID}$, **pk**). It then computes $SK_{\mathtt{ID}}$ = **SetPrivateKey** (**param**, $D_{\mathtt{ID}}$, **sk**) and returns it to $\mathcal{A}$. it outputs $\perp$ if the uesr's public key has been replaced (in the case of Type I adversary.)
   - **Public-Key-Replace-Oracle:** (For Type I adversary only) on input identity and a valid public key, it replaces the associated user's public key with the new one.
   - **Decryption-Oracle:** on input a ciphertext and an identity, returns the decrypted plaintext using the private key corresponding to the current value of the public key associated with the identity of the user.
3. After making oracle queries a polynomial times, $\mathcal{A}$ outputs and submits two message (M_0, M_1), together with an identity $\mathtt{ID}^*$ of uncorrupted secret key to the challenger. The challenger picks a random bit $\beta \in \{0,1\}$ and computers C^*, the encryption of M_β under the current public key $PK_{\mathtt{ID}^*}$ for $\mathtt{ID}^*$. If the output of the encryption is $\perp$, then $\mathcal{A}$ immediately losses the game. Otherwise C^* is delivered to $\mathcal{A}$.
4. $\mathcal{A}$ makes a new sequence of queries.
5. $\mathcal{A}$ outputs a bit β'. It wins if $\beta' = \beta$ and fulfills the following conditions:
   - At any time, $\mathtt{ID}^*$ has not been submitted to **Private-Key-Request-Oracle**.
   - In Step (4), C^* has not been submitted to **Decryption-Oracle** for the combination ($\mathtt{ID}^*$, $PK_{\mathtt{ID}^*}$) under which M_β was encrypted.
   - If it is Type I, $\mathtt{ID}^*$ has not been submitted to both **Public-Key-Replace-Oracle** before Step (3) and **Partial-Key-Extract-Oracle** at some step.

Define the guessing advantage of $\mathcal{A}$ as $Adv_{\mathtt{CLE}}^{\mathtt{IND\text{-}CCA}}(\mathcal{A}) = |\Pr[\beta' = \beta] - \frac{1}{2}|$. A Type I adversary $\mathcal{A}_I$ breaks a IND-CCA secure CL-PKE scheme Π with $(t, q_{par}, q_{pub}, q_{prv}, q_D, \epsilon)$ if and only if the guessing advantage of $\mathcal{A}_I$ that accesses

q_{par} times **Partial-Key-Extract-Oracle**, q_{pub} times **Public-Key-Request-Oracle**, q_{prv} times **Private-Key-Request-Oracle** and q_D times **Decryption-Oracle** is greater than ϵ within running time t. The scheme Π is said to be $(t, q_{par}, q_{pub}, q_{prv}, q_D, \epsilon)$-IND-CCA secure against Type I adversary if there is no attacker $\mathcal{A}_I$ that breaks IND-CCA secure scheme Π with $(t, q_{par}, q_{pub}, q_{prv}, q_D, \epsilon)$. There is the similar definition about Type II adversary.

2.2 Self-Generated-Certificate Public Key Encryption

The definition of SGC Encryption is the same as the definition of CL-encryption given in Definition 1, except for **SetPublicKey** in which the user generates a certificate using his own secret key.

For security, in addition to IND-CCA, we require the scheme to be DoD-Free, which is formally defined as follow as a game played between the challenger and a PPT adversary (DoD Adversary), which has the same power of a Type I adversary defined in CL-encryption.

Definition 3 (DoD-Free Security). *A SGC Encryption scheme is DoD-Free secure if no PPT adversary $\mathcal{A}$ has a non-negligible advantage in the following game played against the challenger:*

1. The challenger takes a security parameter k and runs the **Setup** algorithm. It gives $\mathcal{A}$ the resulting systems parameters **param**. The challenger keeps the master secret key **mk** to itself.
2. $\mathcal{A}$ is given access to **Public-Key-Request-Oracle**, **Partial-Key-Extract-Oracle**, **Private-Key-Request-Oracle** and **Public-Key-Replace-Oracle.**
3. After making oracle queries a polynomial times, $\mathcal{A}$ outputs a message M^*, together with an identity $\mathtt{ID}^*$ to the challenger. The challenger computes C^*, the encryption of M^* under the current public key $PK_{\mathtt{ID}^*}$ for $\mathtt{ID}^*$. If the output of the encryption is $\perp$, then $\mathcal{A}$ immediately losses the game. Otherwise it outputs C^*.
4. $\mathcal{A}$ wins if the following conditions are fulfilled:
   - The output of the encryption in Step (3) is not $\perp$.
   - **Decrypt** (**param**, $SK_{\mathtt{ID}^*}$, C^*) $= M^*$.
   - At any time, $\mathtt{ID}^*$ has not been submitted to **Partial-Key-Extract-Oracle.**

Define the advantage of $\mathcal{A}$ as $Adv_{\mathtt{SGCE}}^{\mathtt{DoD\text{-}Free}}(\mathcal{A}) = \Pr[\mathcal{A}\ wins]$

3 Our CL-PKE Scheme without Pairing

Our scheme modifies from the first CL-PKE Scheme without pairing [4].

3.1 Construction

Setup(k): Generate two large 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 $H_1 : \{0,1\}^* \times \mathbb{Z}_p^* \times \mathbb{Z}_p^* \rightarrow \mathbb{Z}_q^*$, $H_2 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \rightarrow \mathbb{Z}_q^*$ and $H_3 : \mathbb{Z}_p^* \rightarrow \{0,1\}^l$, where $l = l_0 + l_1 \in N$. Return **param** $=(p, q, g, y, H_1, H_2, H_3)$ and **mk** $=(p, q, g, x, H_1, H_2, H_3)$.
UserKeyGeneration(**param**): Pick $z \in \mathbb{Z}_q^*$ at random and compute $\mu = g^x$. Return (**sk**, **pk**) $=(z, \mu)$.
PartialKeyExtract (**param**, **mk**, $\mathtt{ID}$, **pk**): Pick $s \in \mathbb{Z}_q^*$ at random and compute $w = g^s$ and $t = s + xH_1(\mathtt{ID}, w, \mathbf{pk}) = s + xH_1(\mathtt{ID}, w, \mu)$, Return ($P_{\mathtt{ID}}$, $D_{\mathtt{ID}}$) $=(w, t)$.
SetPrivateKey (**param**, $D_{\mathtt{ID}}$, **sk**): Set $SK_{\mathtt{ID}} = (\mathbf{sk}, D_{\mathtt{ID}}) = (z, t)$. Return $SK_{\mathtt{ID}}$.
SetPublicKey (**param**, $P_{\mathtt{ID}}$, **pk**): Set $PK_{\mathtt{ID}} = (\mathbf{pk}, P_{\mathtt{ID}}) = (\mu, w)$. Return $PK_{\mathtt{ID}}$.
Encrypt (**param**, $\mathtt{ID}$, $PK_{\mathtt{ID}}$, M) where the bit-length of M is l_0: Parse $PK_{\mathtt{ID}}$ as (μ, 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 $c_1 = g^r; c_2 = H_3((\mu w y^{H_1(\mathtt{ID},w,\mu)})^r) \oplus (M \| \sigma)$.
Decrypt (**param**, $SK_{\mathtt{ID}}$, C): Parse C as (c_1, c_2) and $SK_{\mathtt{ID}}$ as (z, t). Compute $M \| \sigma = H_3((c_1)^{z+t}) \oplus c_2$. If $g^{H_1(M,\sigma)} = c_1$, return M. Else return "Reject".

Due to $g^{z+t} = g^z \cdot g^t = \mu g^{s+xH_1(\mathtt{ID},w,\mu)} = \mu w y^{H_1(\mathtt{ID},w,\mu)}$, it can be easily seen that the above decryption algorithm is consistent.

Note that in **PartialKeyExtract** algorithm, it includes **pk** generated by the user as input. It is the same binding technique used by the original certificateless encryption scheme [2,3] which raises our scheme to trust level 3 in the trust hierarchy of [10]. Now, with the binding technique in place, a KGC who replaces an entity's public key will be implicated in the event of a dispute: the existence of two working public keys for an identity can only result from the existence of two partial private keys binding that identity to two different public keys; only the KGC could have created these two partial private keys. Thus this binding technique makes the KGC's replacement of a public key apparent and equivalent to a CA forging a certificate in a traditional PKI.

3.2 Security Analysis

The security proofs of our scheme is similar to the first CL-PKE Scheme without Pairing [4]. Basically, the main idea of the security proofs given in this section is to have the CDH attacker $\mathcal{B}$ simulate the "environment" of the Type I and Type II attackers $\mathcal{A}_I$ and $\mathcal{A}_{II}$ respectively until it can compute a Diffie-Hellman key g^{ab} of g^a and g^b using the ability of $\mathcal{A}_I$ and $\mathcal{A}_{II}$.

For the attacker $\mathcal{A}_I$, $\mathcal{B}$ 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 $\mathcal{A}_{II}$, $\mathcal{B}$ set g^a as a part of the challenge ciphertext but uses g^b to generate a public key associated with the challenge identity.

The following two theorems show that our scheme is IND-CCA secure in the random oracle, assuming that the CDH problem is intractable. We will give the proofs of Theorem 2 and omit the certification process of Theorem 1 due to the similarity of Theorem 2.

Theorem 1. *The CL-PKE scheme is* $(t, q_{H_1}, q_{H_2}, q_{H_3}, q_{par}, q_{pub}, q_{prv}, q_D, \epsilon)$*-IND-CCA secure against the Type I attacker* $\mathcal{A}_I$ *in the random oracle assuming the CDH problem is* $(t^{'}, \varepsilon^{'})$*-intractable, where* $\epsilon^{'} > \frac{1}{q_{H_2}}(\frac{2\epsilon}{e(q_{prv}+1)} - \frac{q_{H_2}}{2^{l_1}} - \frac{q_D q_{H_2}}{2^{l_1}} - \frac{q_D}{q})$ *and* $t^{'} > t + 2(q_{par} + q_{pub} + q_{prv})t_{ex} + 2q_D q_{H_2} q_{H_3} t_{ex} + 3t_{ex}$ *where* t_{ex} *denotes the time for computing exponentiation in* $\mathbb{Z}_p^*$.

Theorem 2. *The CL-PKE scheme is* $(t, q_{H_1}, q_{H_2}, q_{H_3}, q_{pub}, q_{prv}, q_D, \epsilon)$*-IND-CCA secure against the Type II attacker* $\mathcal{A}_{II}$ *in the random oracle assuming the CDH problem is* $(t^{'}, \varepsilon^{'})$*-intractable, where* $\epsilon^{'} > \frac{1}{q_{H_2}}(\frac{2\epsilon}{e(q_{prv}+1)} - \frac{q_{H_2}}{2^{l_1}} - \frac{q_D q_{H_2}}{2^{l_1}} - \frac{q_D}{q})$ *and* $t^{'} > t + 2(q_{pub} + q_{prv})t_{ex} + 2q_D q_{H_2} q_{H_3} t_{ex} + 3t_{ex}$ *where* t_{ex} *denotes the time for computing exponentiation in* $\mathbb{Z}_p^*$.

Proof. Assume there is a Type II adversary $\mathcal{A}_{II}$ exists. We are going to construct another PPT $\mathcal{B}$ that make uses of $\mathcal{A}_{II}$ to solve the CDH problem with probability at least $\epsilon^{'}$ and in the time at most $t^{'}$.

$\mathcal{B}$ is given (p, q, g, g^a, g^b) as an instance of the CDH problem. In order to use $\mathcal{A}_{II}$ to solve for the problem, $\mathcal{B}$ needs to simulates a challenger and all oracles for $\mathcal{A}_{II}$. $\mathcal{B}$ does it in the following way.

Setup. $\mathcal{B}$ picks $x \in \mathbb{Z}_q^*$ uniformly at random and computes $y = g^x$, then sets **param** $=(p, q, g, y, H_1, H_2, H_3)$ and **mk** $=(p, q, g, x, H_1, H_2, H_3)$. Finally gives $\mathcal{A}_{II}$ **param** and **mk**.

We suppose that H_1, H_2, H_3 are random oracles [14]. Adversary $\mathcal{A}_{II}$ may make queries of all random oracles at any time during its attack. $\mathcal{B}$ handles as follows:

H_1 **queries:** On receiving a query $(\mathtt{ID}, w, \mu)$ to H_1:

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

H_2 **queries:** On receiving a query (M, σ) to H_2:

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

H_3 **queries:** On receiving a query k to H_3:

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

Phase 1. $\mathcal{A}_{II}$ can issue the following oracle queries.
Public-Key-Request: On receiving a query $\mathtt{ID}$:

1. If $\langle \mathtt{ID}, (\mu, w), coin \rangle$ exists in **PublicKeyList**, return $PK_{\mathtt{ID}} = (\mu, w)$ as answer.
2. Otherwise, pick $coin \in \{0,1\}$ at random, so that $\Pr[coin = 0] = \delta$. (δ will be determined later.)
3. If $coin = 0$, pick $z, s \in \mathbb{Z}_q^*$ at random and compute $\mu = g^z, w = g^s$, and $t = s + xH_1(\mathtt{ID}, w, \mu)$; add $\langle \mathtt{ID}, (z,t) \rangle$ to **PrivateKeyList** and $\langle \mathtt{ID}, (\mu, w), coin \rangle$ to **PublicKeyList**; return $PK_{\mathtt{ID}} = (\mu, w)$ as a answer.
4. Otherwise (if $coin = 1$), pick $z, s \in \mathbb{Z}_q^*$ at random and compute $\mu = g^z, w = (g^b)^s$; add $\langle \mathtt{ID}, (z, ?) \rangle$ to **PrivateKeyList** and $\langle \mathtt{ID}, (\mu, w), coin \rangle$ to **PublicKeyList**; return $PK_{\mathtt{ID}} = (\mu, w)$ as a answer.

Private-Key-Request: On receiving a query $\mathtt{ID}$:

1. Run **Public-Key-Request** on $\mathtt{ID}$ to get a tuple $\langle \mathtt{ID}, (\mu, w), coin \rangle \in$ **PublicKeyList**.
2. If $coin = 0$, search **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.

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

1. Search **PublicKeyList** for tuple $\langle \mathtt{ID}, (\mu, w), coin \rangle$. If $coin = 0$, search **PrivateKeyList** for a tuple $\langle \mathtt{ID}, (z,t) \rangle$. (Note that $\langle \mathtt{ID}, (\mu, w), coin \rangle$ must exist in **PublicKeyList** and when $coin = 0$, $\langle \mathtt{ID}, (z,t) \rangle$ exist in **PrivateKeyList.**) Then set $SK_{\mathtt{ID}} = (z,t)$ and run **Decrypt** (**param**, $SK_{\mathtt{ID}}$, C). Finally, return the result of **Decrypt** algorithm.
2. Otherwise (if $coin = 1$), run H_1 **query** to get a tuple $\langle (\mathtt{ID}, w, \mu), e \rangle$. If there exist $\langle (M, \sigma), r \rangle \in$ **$\mathbf{H_2}$List** and $\langle k, R \rangle \in$ **$\mathbf{H_3}$List** such that $c_1 = g^r, c_2 = R \oplus (M || \sigma)$ and $k = (\mu w y^e)^r$, return M and "Reject" otherwise.

Challenge. $\mathcal{A}_{II}$ then output two message (M_0, M_1) and a challenge identity $\mathtt{ID}^*$.
$\mathcal{B}$ run **Public-Key-Request** taking $\mathtt{ID}^*$ as input to get a tuple $\langle \mathtt{ID}^*, (\mu^*, w^*), coin \rangle \in$ **PublicKeyList**.

1. If $coin = 0$ return "Abort" and terminate.
2. Otherwise, do the following:
   (a) Search **PrivateKeyList** for a tuple $\langle \mathtt{ID}^*, (z^*, ?), s^* \rangle$.
   (b) Pick $\sigma^* \in \{0,1\}^{l_1}, c_2^* \in \{0,1\}^l$ and $\beta \in \{0,1\}$ at random.
   (c) Set $c_1^* = g^a$ and $e^* = H_1(\mathtt{ID}^*, w^*, \mu^*)$.
   (d) Define $a = H_2(M_\beta, \sigma^*)$ and $H_3((\mu^* w^* y^{e^*})^a)$. (Note that $\mathcal{B}$ does not know "a", $(\mu^* w^* y^{e^*})^a = (g^a)^{z^*} \cdot (g^{ab})^{s^*} \cdot (g^a)^{x e^*}$.
3. Return $C^* = (c_1^*, c_2^*)$ as a target ciphertext.

Phase 2. $\mathcal{B}$ repeats the same method it used in Phase 1.

Guess. Finally, $\mathcal{A}_{II}$ output a guess $\beta^{'}$. Now $\mathcal{B}$ choose a tuple $\langle k, R\rangle$ form the **H_3List** and outputs $(\frac{k}{(g^a)^{z^*}\cdot(g^a)^{xe^*}})^{1/s^*}$ as the solution the the CDH problem.

Analysis : From the construction of H_1, it is clear that the simulation of H_1 is perfect. As long as $\mathcal{A}_{II}$ does not query (M_β, σ^*) to H_2 nor $(\mu^* w^* y^{e^*})^a$ to H_3, the simulations of H_2 and H_3 are perfect. By $\mathbf{AskH_3^*}$ we denote the event that $(\mu^* w^* y^{e^*})^a$ has not been queried to H_3. Also, by $\mathbf{AskH_2^*}$ we denote the event that (M_β, σ^*) has been queried to H_2. If happens then $\mathcal{B}$ will be able to solve the CDH problem by choosing a tuple $\langle k, R\rangle$ form the **H_3List** and computing $(\frac{k}{(g^a)^{z^*}\cdot(g^a)^{xe^*}})^{1/s^*}$ with the probability at least $\frac{1}{q_{H_3}}$. Hence we have $\epsilon^{'} \geq \frac{1}{q_{H_3}}\Pr[\mathbf{AskH_3^*}]$.

It is easy to notice that if $\mathcal{B}$ does not abort, the simulations of **Public-Key-Request**, **Private-Key-Request** and 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 been produced under $coin = 0$, the simulation is perfect as B knows the private key $SK_{\mathtt{ID}}$ corresponding to $PK_{\mathtt{ID}}$. Otherwise, simulation errors may occur while $\mathcal{B}$ running the decryption oracle simulator specified above. Let **DecErr** be this event. We compute the probability of this event: Suppose that ($\mathtt{ID}$, $PK_{\mathtt{ID}}$, C), where $C = (c_1, c_2)$ and $PK_{\mathtt{ID}} = (\mu, w)$, has been issued as a *valid* decryption query. Even if C is valid, there is a possibility that C can be produced without querying $(\mu w y^e)^r$ to H_3, where $e = H_1(\mathtt{ID}, w, \mu)$ and $r = H_2(M, \sigma)$. Let **Valid** be an event that C is valid. Let $\mathbf{AskH_3}$ and $\mathbf{AskH_2}$ respectively be events that $(\mu w y^e)^r$ has been queried to H_3 and (M, σ) has been queried to H_2 with respect to $C = (c_1, c_2) = (g^r, H_3((\mu w y^{H_1(\mathtt{ID},w,\mu)})^r) \oplus (M||\sigma))$ and $PK_{\mathtt{ID}} = (\mu, w)$, where $r = H_2(M, \sigma)$ and $e = H_1(\mathtt{ID}, w, \mu)$. We then have $\Pr[\mathbf{DecErr}] = q_D \Pr[\mathbf{Valid}|\neg\mathbf{AskH_3}]$. But

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

So, $\Pr[\mathbf{DecErr}] \leq \frac{q_D q_{H_2}}{2^{l_1}} + \frac{q_D}{q}$.

Now,the event $(\mathbf{AskH_3^*} \vee (\mathbf{AskH_2^*}|\neg\mathbf{AskH_3^*}) \vee \mathbf{DecErr})|\neg\mathbf{Abort}$ denoted by **Good**, where **Abort** denotes an event that $\mathcal{B}$ aborts during the simulation. The probability $\neg\mathbf{Abort}$ that happens is given by $\delta^{q_{prv}}(1-\delta)$ which is maximized at $\delta = 1 - 1/(q_{prv} - 1)$. Hence we have $\Pr[\neg\mathbf{Abort}] \leq \frac{1}{e(q_{prv}+1)}$, where e denotes the base of the natural logarithm.

If **Good** does not happen, it is clear that $\mathcal{A}_{II}$ 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 $\Pr[\beta^{'} = \beta|\neg\mathbf{Good}] \leq \frac{1}{2}$.

By definition of ϵ, we then have

$$\begin{aligned}
\epsilon &< |\Pr[\beta^{'} = \beta] - \frac{1}{2}| \\
&= |\Pr[\beta^{'} = \beta|\neg\mathbf{Good}]\Pr[\neg\mathbf{Good}] + \Pr[\beta^{'} = \beta|\mathbf{Good}]\Pr[\mathbf{Good}] - \frac{1}{2}| \\
&\leq |\frac{1}{2}\Pr[\neg\mathbf{Good}] + \Pr[\mathbf{Good}] - \frac{1}{2}| \\
&\leq \frac{1}{2}\Pr[\mathbf{Good}] \\
&\leq \frac{1}{2\Pr[\neg\mathbf{Abort}]}(\Pr[\mathbf{AskH_3^*}] + \Pr[\mathbf{AskH_2^*}|\neg\mathbf{AskH_3^*}] + \Pr[\mathbf{DecErr}]) \\
&\leq \frac{e(q_{prv}+1)}{2}(q_{H_3}\epsilon^{'} + \frac{q_{H_2}}{2^{l_1}} + \frac{q_D q_{H_2}}{2^{l_1}} + \frac{q_D}{q})
\end{aligned}$$

Consequently, we obtain $\epsilon^{'} > \frac{1}{q_{H_2}}(\frac{2\epsilon}{e(q_{prv}+1)} - \frac{q_{H_2}}{2^{l_1}} - \frac{q_D q_{H_2}}{2^{l_1}} - \frac{q_D}{q})$. The running time of the CDH attacker $\mathcal{B}$ is $t^{'} > t + 2(q_{pub} + q_{prv})t_{ex} + 2q_D q_{H_2} q_{H_3} t_{ex} + 3t_{ex}$ where t_{ex} denotes the time for computing exponentiation in $\mathbb{Z}_p^*$.

4 Our SGC-PKE Scheme without Pairing

We give our Self-Generated-Certificate (SGC) encryption scheme without pairing based on the above Certificateless encryption scheme. The most algorithms are the same as the algorithms of Certificateless encryption scheme, except for **SetPublicKey** and **Encrypt.**

In order to distinguish the algorithm of CL-encryption, we will add the prefix "**CL.**" to the corresponding algorithms. For example, we use "**CL.Setup**" to denote the encryption algorithm of the CL-encryption scheme. The proposed SGC-encryption scheme is described as follow:

Setup: Same as **CL.Setup**, outputs parameters **param** $= (p, q, g, y = g^x, H_1, H_2, H_3)$ and master secret key **mk** $= (p, q, g, x, H_1, H_2, H_3)$.
UserKeyGeneration: Same as **CL.UserKeyGeneration**, outputs (**sk**, **pk**) $= (z, g^z)$.
PartialKeyExtract: We modify **CL.PartialKeyExtract** slightly. Taking **param**, **mk**, ID and **pk** as input, it outputs $(P_{\mathtt{ID}}, D_{\mathtt{ID}}) = (w = g^s, t = s + xH_1(\mathtt{ID}, w * \mathbf{pk}) = s + xH_1(\mathtt{ID}, w\mu))$. In order to make this changes, it must modify the domain of hash function $H_1 : \{0,1\}^* \times \mathbb{Z}_p^* \rightarrow \mathbb{Z}_q^*$.
SetPrivateKey: Same as **CL.SetPrivateKey**, outputs $SK_{\mathtt{ID}} = \mathbf{sk} + D_{\mathtt{ID}} = z + t$.
SetPublicKey: Except for taking **param**, $P_{\mathtt{ID}}$ and **pk** as input, it includes ID and $SK_{\mathtt{ID}}$ as inputs. Chooses a new hash function $H_0 : \{0,1\}^* \times \mathbb{Z}_p^* \times \mathbb{Z}_p^* \times \mathbb{Z}_p^* \rightarrow$

$\mathbb{Z}_q^*$, then computes $PK_{\mathtt{ID}}^1 = \mathbf{pk} * P_{\mathtt{ID}} = \mu w$ and $PK_{\mathtt{ID}}^2 = \mathbf{pk} * P_{\mathtt{ID}} * y^{H_1(\mathtt{ID}, \mathbf{pk}, P_{\mathtt{ID}})} = \mu w y^{H_1(\mathtt{ID}, \mu, w)} = g^{z+t} = g^{SK_{\mathtt{ID}}}$. Next, it does the following performances to sign the user's identity $\mathtt{ID}$ and $PK_{\mathtt{ID}}^1, PK_{\mathtt{ID}}^2$ using the user's private key $SK_{\mathtt{ID}}$ and Schnorr's signature scheme [15]. (1) choose a random $r \in \mathbb{Z}_q^*$, (2) compute $R = g^r mod\ p$, and (3) set the signature to be (R, σ), where $\sigma = r + SK_{\mathtt{ID}} * H_0(\mathtt{ID}, PK_{\mathtt{ID}}^1, PK_{\mathtt{ID}}^2, R)$. Finally, returns $PK_{\mathtt{ID}} = (PK_{\mathtt{ID}}^1, PK_{\mathtt{ID}}^2, (R, \sigma))$.
Encrypt: Parses $PK_{\mathtt{ID}}$ as $(PK_{\mathtt{ID}}^1, PK_{\mathtt{ID}}^2, (R, \sigma))$. If $PK_{\mathtt{ID}}^2 \neq PK_{\mathtt{ID}}^1 * y^{H_1(\mathtt{ID}, PK_{\mathtt{ID}}^1)}$ or $g^\sigma \neq R * (PK_{\mathtt{ID}}^2)^{H_0(\mathtt{ID}, PK_{\mathtt{ID}}^1, PK_{\mathtt{ID}}^2, R)}$, it returns $\perp$, else outputs **CL.Encrypt**(**param**, $\mathtt{ID}$, $PK_{\mathtt{ID}}$, M).
Decrypt: Same as **CL.Decrypt**, outputs a plaintext M for a valid ciphertext C, or "Reject" otherwise.

Security Analysis
The IND-CCA security depends on our CL-encryption scheme (defined in Section 3). In addition to IND-CCA, we require the scheme to be DoD-Free. Here we analyze the DoD-Free Security.

Theorem 3. *The SGC-encryption scheme proposed in this section in secure against DoD adversary, assuming that the Schnorr's signature scheme is secure against the adaptively chosen message attack in the random oracle model [16].*

Proof. Assume there is a DoD adversary $\mathcal{A}$ exists. We are going to construct another PPT $\mathcal{B}$ that makes use of $\mathcal{A}$ to break the Schnorr signature scheme.

$\mathcal{B}$ is now the schnorr's signature adversary. Note that in fact, the **PartialKeyExtract** algorithm in our SGC-encryption scheme signs the user's identity $\mathtt{ID}$ using the schnorr's signature scheme. So using his signing-oracle, $\mathcal{B}$ can answer all oracle queries for $\mathcal{A}$. After a polynomial number of oracle queries, $\mathcal{A}$ outputs a message M^* and an identity $\mathtt{ID}^*$. $\mathcal{A}$ wins if the following conditions fulfill:

1. The public key $PK_{\mathtt{ID}^*}$ of $\mathtt{ID}^*$ is valid.
2. **Decrypt**(**param**, $SK_{\mathtt{ID}^*}, C^*$) $\neq M^*$ where C^*= **Encrypt** (**param**, $\mathtt{ID}^*$, $PK_{\mathtt{ID}^*}$, M^*).
3. $\mathcal{A}$ does not query the **Partial-Key-Extract-Oracle** for $\mathtt{ID}^*$.

If the public key of $\mathtt{ID}^*$ has not been replaced, due to correctness we always have **Decrypt**(**param**, $SK_{\mathtt{ID}^*}, C^*) = M^*$. Condition (2) implies the public key of $\mathtt{ID}^*$ has been replaced. Together with condition (1) and (3), it implies that $\sigma^* = (PK_{\mathtt{ID}^*}^1, PK_{\mathtt{ID}^*}^2)$ is a successful forgery for $\mathtt{ID}^*$. $\mathcal{B}$ outputs it.

5 Comparison to Previous Work

Our scheme is the second SGC-encryption scheme. In this section, we compare the scheme we have presented to the first scheme in [1].

1. Our scheme has more short public keys due to their scheme based on the Water's Identity-Based Encryption scheme [17].

2. Our scheme is more efficient due to our scheme without pairing computation. 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.
3. Our scheme reaches Girault's trusted level 3 (same as the traditional PKI), but their scheme only reaches Girault's trusted level 2 (a cheating KGC could replace an entity's public key bye one for which it knows the secret value without fear of being identified).
4. Their scheme is IND-CCA$^-$ (the challenger is forced to decrypt ciphertexts for which the public key has been replaced) and DoD-Free secure in the standard model. Our scheme is IND-CCA and DoD-Free secure in the random oracle model.

6 Concluding Remarks

We have presented the first SGC-encryption scheme that does not depend on the pairing. We have proven in the random oracle that the scheme is IND-CCA and DoD-Free secure, relative to the hardness of the standard CDH problem and DL problem.

However, we can only achieve security in the random oracle although our scheme has many appealing properties. It is still an open problem to design a CL-PKC and SGC-PKC scheme without pairing that is secure in the standard model.

References

1. J. K. Liu and M. H. Au. Self-Generated-Certificate Public Key Cryptosystem. Cryptology ePrint Archive, Report 2006/194, 2006. http://eprint.iacr.org/2006/194.
2. S. S. Al-Riyami and K. Paterson. Certificateless public key cryptography. In Proc. ASIACRYPT 2003, LNCS 2894, pp. 452-473, Springer-Verlag, 2003.
3. S. S. Al-Riyami and K. Paterson. Certificateless public key cryptography. Cryptology ePrint Archive, Report 2003/126, 2003. http://eprint.iacr.org/2003/126.
4. J. Baek, R. Safavi-Naini, and W. Susilo. Certificateless public key encryption without pairing. In ISC 05, LNCS 3650, pp. 134-148, Springer-Verlag, 2005.
5. K. Bentahar, P. Farshim, and J. Malone-Lee. Generic constructions of identity-based and certificateless KEMs. Cryptology ePrint Archive, Report 2005/058, 2005. http://eprint.iacr.org/2005/058.
6. B. Libert and J. Quisquater. On constructing certificateless cryptosystems from identity based encryption. In PKC 2006, LNCS 3958, pp. 474-490, Springer-Verlag, 2006.
7. D. H. Yum and P. J. Lee. Generic construction of certificateless encryption. In ICCSA'04, LNCS 3040, pp. 802-811, Springer-Verlag, 2004.
8. Y. Shi and J. Li. Provable efficient certificateless public key encryption. Cryptology ePrint Archive, Report 2005/287, 2005. http://eprint.iacr.org/2005/287.
9. Z. Cheng and R. Comley. Efficient certificateless public key encryption. Cryptology ePrint Archive, Report 2005/012, 2005. http://eprint.iacr.org/2005/012.

10. M. Girault. Self-certified public keys. In Proc. EUROCRYPT 91, LNCS 547, pp. 490-497, Springer-Verlag, 1992.
11. H. Petersen and P. Horster. Self-certified keys - concepts and applications. In 3rd Int. Conference on Communications and Multimedia Security, pp. 102-116, Chapnam and Hall, 1997.
12. B. Lee and K. Kim. Self-Certificate: PKI using Self-Certified Key. In Proc. of Conference on Information Security and Cryptology 2000, Vol. 10, No. 1, pp. 65-73, 2000.
13. A. Shamir. Identity-based Cryptosystems and Signature Schemes. In Crypto '84, LNCS 196, pp. 47-53, Springer-Verlag, 1984.
14. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In ACM CCCS '93, pp. 62-73, 1993.
15. C. P. Schnorr. Efficient signature generation by smart cards. Journal of Cryptology, Vol. 4, No. 3, pp. 161-174, 1991.
16. D. Pointcheval and J. Stern. Security proofs for signature schemes. In Proc. Eurocrypt 96, LNCS 1070, pp. 387-398, Springer-Verlag, 1996.
17. B. Waters. Efficient identity-based encryption without random oracles. In Proc. EUROCRYPT 2005, LNCS 3494, pp. 114-127, Springer-Verlag, 2005.