Certificateless Hybrid Signcryption*

Fagen Li[1,2,3], Masaaki Shirase[3], and Tsuyoshi Takagi[3]

[1] School of Computer Science and Engineering,
University of Electronic Science and Technology of China, Chengdu 610054, China
[2] Key Laboratory of Computer Networks and Information Security,
Xidian University, Xi'an 710071, China
[3] School of Systems Information Science,
Future University-Hakodate, Hakodate 041-8655, Japan
{fagenli,shirase,takagi}@fun.ac.jp

Abstract. Signcryption is a cryptographic primitive that fulfills both the functions of digital signature and public key encryption simultaneously, at a cost significantly lower than that required by the traditional signature-then-encryption approach. In this paper, we address a question whether it is possible to construct a hybrid signcryption scheme in the certificateless setting. This question seems to have never been addressed in the literature. We answer the question positively in this paper. In particular, we extend the concept of signcryption tag-KEM to the certificateless setting. We show how to construct a certificateless signcryption scheme using certificateless signcryption tag-KEM. We also give an example of certificateless signcryption tag-KEM.

Keywords: Certificateless signcryption, hybrid signcryption, signcryption tag-KEM, DEM.

1 Introduction

Confidentiality, integrity, non-repudiation and authentication are the important requirements for many cryptographic applications. A traditional approach to achieve these requirements is to sign-then-encrypt the message. Signcryption, first proposed by Zheng [38], is a cryptographic primitive that fulfills both the functions of digital signature and public key encryption simultaneously, at a cost significantly lower than that required by the traditional signature-then-encryption approach. Several efficient signcryption schemes have been proposed since 1997 [5,19,22,29,30,32,36,39]. The original scheme in [38] is based on the discrete logarithm problem but no security proof is given. Zheng's original scheme was only proven secure by Baek, Steinfeld, and Zheng [4] who described a formal security model in a multi-user setting. In above traditional signcryption schemes, the public key of a user is essentially a random bit string picked from a given set. So, the signcryption does not provide the authorization of the user by itself. This problem can be solved via a certificate which provides an unforgeable and trusted link between the public key and the identity of the user by the signature of a certificate authority (CA), and there is a hierarchical framework that is called public key infrastructure (PKI) to issue and manage certificates. However, the certificates management including revocation, storage, distribution and the computational cost of certificates verification is the main difficulty against traditional PKI.

To simplify key management procedures of traditional PKI, Shamir [33] proposed the concept of identity-based cryptography (IBC) in 1984. The idea of IBC is to get rid of certificates by allowing

* Full version of a paper published in The 5th Information Security Practice and Experience Conference (ISPEC 2009), LNCS 5451, pp. 112–123, Springer-Verlag, 2009.

the user's public key to be any binary string that uniquely identifies the user. Examples of such strings include email addresses and IP addresses. Several practical identity-based signature (IBS) schemes have been devised since 1984 [18,20], but a satisfying identity-based encryption (IBE) scheme only appeared in 2001 [10]. It was devised by Boneh and Franklin and cleverly uses bilinear maps (the Weil or Tate pairing) over supersingular elliptic curves. Subsequently, several identity-based signcryption (IBSC) schemes are also proposed [7,11,12,14,26,27,28]. The main practical benefit of IBC is in greatly reducing the need for the public key certificates. But IBC uses a trusted third party called private key generator (PKG). The PKG generates the secret keys of all of its users, so a user can decrypt only if the PKG has given a secret key to it (so, certification is implicit), hence reduces the amount of storage and computation. On the other hand, the dependence on the PKG who can generate all users' private keys inevitably causes the key escrow problem to the IBC. For example, the PKG can decrypt any ciphertext in an IBE scheme. Equally problematical, the PKG could forge any user's signature in an IBS scheme.

To solve the key escrow problem in the IBC, Al-Riyami and Paterson [2] introduced a new paradigm called certificateless cryptography. The certificateless cryptography does not require the use of certificates and yet does not have the built-in key escrow feature of IBC. It is a model for the use of public key cryptography that is intermediate between traditional PKI and IBC. A certificateless system still makes use of a trusted third party which is called the key generating center (KGC). By way of contrast to the PKG in the IBC, the KGC does not have access to the user's private key. Instead, the KGC supplies a user with a partial private key that the KGC computes from the user's identity and a master key. The user then combines the partial private key with some secret information to generate the actual private key. The system is not identity-based, because the public key is no longer computable from a user's identity. When Alice wants to send a message to Bob in a certificateless system, she must obtain Bob's public key. However, no authentication of Bob's public key is necessary and no certificate is required. In 2008, Barbosa and Farshim [6] introduced the notion of certificateless signcryption (CLSC) and proposed an efficient scheme.

The practical way to perform secrecy communication for large messages is to use hybrid encryption that separates the encryption into two parts: one part uses public key techniques to encrypt a one-time symmetric key; the other part uses the symmetric key to encrypt the actual message. In such a construction, the public key part of the algorithm is known as the key encapsulation mechanism (KEM) while the symmetric key part is known as the data encapsulation mechanism (DEM). A formal treatment of this paradigm originates in the work of Cramer and Shoup [15]. The resulting KEM-DEM hybrid encryption paradigm has received much attention in recent years [1,24,25]. It is very attractive as it gives a clear separation between the various parts of the cipher allowing for modular design. In [1], Abe, Gennaro, and Kurosawa introduced tag-KEM which takes as input a tag in KEM. Bentahar et al. [8] extended KEM into identity-based and certificateless settings and gave generic constructions of identity-based KEM (IB-KEM) and certificateless KEM (CL-KEM). Chen et al. [13] proposed an efficient IB-KEM based on the Sakai-Kasahara key construction [31]. Kiltz and Galindo [23] proposed a direct construction of IB-KEM in the standard model, based on Waters's IBE scheme [35]. Huang and Wong [21] proposed a generic construction of CL-KEM in the standard model.

The use of hybrid techniques to build signcryption schemes has been studied by Dent [16,17]. He generalized KEM to signcryption KEM which includes an authentication in KEM. However, he only consider the insider security for authenticity. That is, if the sender's private key is exposed, an attacker is able to recover the key generated by signcryption KEM. The full insider security [3] means that (a) if the sender's private key is exposed, an attacker is still not able to recover the message from the ciphertext and (b) if the receiver's private key is exposed, an attacker is still not able to forge a ciphertext. In 2006, Bjørstad and Dent [9] showed how to built signcryption schemes using tag-KEM. However, they also only consider the insider security for authenticity and not for confidentiality. In 2008, Tan [34] proposed full insider secure signcryption KEM and tag-KEM in the standard model. Tan's schemes are insider secure for both authenticity and confidentiality. Note that the using of tag-KEM yields simpler scheme descriptions and better generic security reductions.

All the above hybrid signcryption schemes [9,16,17,34] are not in the certificateless setting. In this paper, we address a question whether it is possible to construct a hybrid signcryption scheme in the certificateless setting. This question seems to have never been addressed in the literature. We answer the question positively in this paper. In particular, we extend the concept of signcryption tag-KEM to the certificateless setting. We show that a CLSC scheme can be constructed by using a certificateless signcryption tag-KEM (CLSC-TKEM) and a DEM. We also give an example of CLSC-TKEM. Our scheme is insider secure for both authenticity and confidentiality.

The rest of this paper is organized as follows. We introduce the preliminary work in Section 2. We give the formal model of CLSC-TKEM in Section 3. We show how to construct a CLSC scheme using a CLSC-TKEM and a DEM in Section 4. An example of CLSC-TKEM is described in Section 5. Finally, the conclusions are given in Section 6.

2 Preliminaries

2.1 Certificateless Signcryption (CLSC)

A generic CLSC scheme consists of the following six algorithms.

- `Setup`: This algorithm takes as input the security parameter 1^k and returns the KGC's master secret key msk and system parameters $params$ including a master public key mpk and descriptions of message space $\mathcal{M}$, ciphertext space $\mathcal{C}$ and randomness space $\mathcal{R}$. This algorithm is executed by the KGC, which publishes $params$.
- `Extract-Partial-Private-Key`: This algorithm takes as input $params$, msk and a user's identity $ID \in \{0,1\}^*$, and returns a partial private key D_{ID}. This algorithm is run by the KGC, after verifying the user's identity.
- `Generate-User-Keys`: This algorithm takes as input $params$ and an identity ID, and outputs a secret value x_{ID} and a public key PK_{ID}. This algorithm is run by a user to obtain a public key and a secret value which can be used to construct a full private key. The public key is published without certification.
- `Set-Private-Key`: This algorithm takes as input a partial private key D_{ID} and a secret value x_{ID}, and returns the full private key S_{ID}. Again, this algorithm is run by a user to construct the full private key.

– `Signcrypt`: This algorithm takes as input $params$, a plaintext message $m \in \mathcal{M}$, the sender's full private key S_{ID_s}, identity ID_s and public key PK_{ID_s}, and the receiver's identity ID_r and public key PK_{ID_r}, and outputs a ciphertext $\sigma \in \mathcal{C}$.
– `Unsigncrypt`: This algorithm takes as input $params$, a ciphertext σ, the sender's identity ID_s and public key PK_{ID_s}, and the receiver's full private key S_{ID_r}, identity ID_r and public key PK_{ID_r}, and outputs a plaintext m or a failure symbol $\perp$ if σ is an invalid ciphertext.

We make the consistency constraint that if

$$\sigma \leftarrow \texttt{Signcrypt}(params, m, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r}),$$

then

$$m \leftarrow \texttt{Unsigncrypt}(params, \sigma, ID_s, PK_{ID_s}, S_{ID_r}, ID_r, PK_{ID_r}).$$

Barbosa and Farshim [6] defines the security notions for CLSC schemes. A CLSC scheme should satisfy confidentiality (indistinguishability against adaptive chosen ciphertext attacks (IND-CCA2)) and unforgeability (existential unforgeability against adaptive chosen messages attacks (UF-CMA)). For the stronger notion of insider security, we use the notion of strong existential unforgeability (sUF-CMA). The strong existential unforgeability means that an adversary wins if it outputs a valid message/signcryption pair (m, σ) for identities ID_s and ID_r and the signcryption σ was not returned by the signcryption oracle when queried on the message m. As in [11,12], we do not consider attacks targeting signcryptions where the identities of the sender and receiver are the same. That is, we disallow such queries to relevant oracles and do not accept this type of signcryption as a valid forgery.

There are two types of adversaries, Type I and Type II. A Type I adversary models an attacker which is a common user of the system and is not in possession of the KGC's master secret key. But it is able to adaptively replace users'public keys with (valid) public keys of its choice. A Type II adversary models an honest-but-curious KGC who knows the KGC's master secret key. But it cannot replace users' public keys.

For the confidentiality, we consider two games "`IND-CCA2-I`" and "`IND-CCA2-II`" where a Type I adversary $\mathcal{A}_I$ and a Type II adversary $\mathcal{A}_{II}$ interact with their "challenger" in these two games, respectively. Note that the challenger keeps a history of "query-answer" while interacting with the attackers. Now we describe the two games.

`IND-CCA2-I`: This is the game in which $\mathcal{A}_I$ interacts with the "challenger":

`Initial`: The challenger runs $(params, msk) \leftarrow \texttt{Setup}(1^k)$ and gives $params$ to $\mathcal{A}_I$. The challenger keeps master secret key msk to itself.

`Phase 1`: The adversary $\mathcal{A}_I$ can perform a polynomially bounded number of queries in an adaptive manner.

– `Extract partial private key`: The adversary $\mathcal{A}_I$ chooses an identity ID. The challenger computes $D_{ID} \leftarrow$`Extract-Partial-Private-Key`$(params, msk, ID)$ and sends D_{ID} to $\mathcal{A}_I$.
– `Extract private key`: The adversary $\mathcal{A}_I$ chooses an identity ID. The challenger first computes $D_{ID} \leftarrow$`Extract-Partial-Private-Key`$(params, msk, ID)$ and then computes $(x_{ID}, PK_{ID}) \leftarrow$ `Generate-User-Keys`$(params, ID)$. Finally, it sends the result of $S_{ID} \leftarrow$`Set-Private-Key`(x_{ID}, D_{ID}) to $\mathcal{A}_I$. The adversary is not allowed to query any identity for which the corresponding public

key has been replaced. This restriction is imposed due to the fact that it is unreasonable to expect that the challenger is able to provide a full private key for a user for which it does not know the secret value.

- `Request public key`: The adversary $\mathcal{A}_I$ chooses an identity ID. The challenger computes $(x_{ID}, PK_{ID}) \leftarrow$`Generate-User-Keys`$(params, ID)$ and sends PK_{ID} to $\mathcal{A}_I$.
- `Replace public key`: $\mathcal{A}_I$ may replace a public key PK_{ID} with a value chosen by it.
- `Signcryption queries`: The adversary $\mathcal{A}_I$ chooses a m, a sender's identity ID_s and a receiver's identity ID_r, the challenger finds S_{ID_s} from its "query-answer" list, computes $\sigma \leftarrow$ `Signcrypt`$(params, m, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$, and returns σ to $\mathcal{A}_I$. Note that, it is possible that the challenger is not aware of the sender's secret value, if the associated public key has been replaced. In this case, we require the adversary to provide it. We disallow queries where $ID_s = ID_r$.
- `Unsigncryption queries`: $\mathcal{A}_I$ chooses a σ, a sender's identity ID_s and a receiver's identity ID_r, the challenger finds S_{ID_r} from its "query-answer" list, computes `Unsigncrypt`$(params, \sigma, ID_s, PK_{ID_s}, S_{ID_r}, ID_r, PK_{ID_r})$, and returns the result to $\mathcal{A}_I$. The result is either a plaintext message m or $\perp$. Note that, it is possible that the challenger is not aware of the receiver's secret value, if the associated public key has been replaced. In this case, we require the adversary to provide it. We also disallow queries where $ID_s = ID_r$.

`Challenge`: The adversary $\mathcal{A}_I$ decides when Phase 1 ends. $\mathcal{A}_I$ generates two equal length plaintexts (m_0, m_1), a sender's identity ID_s^*, and a receiver's identity ID_r^* on which it wishes to be challenged. Note that ID_r^* should not be queried to extract a private key in Phase 1. Note also that ID_r^* cannot be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. The challenger picks a random bit δ from $\{0,1\}$, computes $\sigma^* \leftarrow$ `Signcrypt`$(params, m_\delta, S_{ID_s^*}, ID_s^*, PK_{ID_s^*}, ID_r^*, PK_{ID_r^*})$, and returns σ^* to $\mathcal{A}_I$.

`Phase 2`: The adversary $\mathcal{A}_I$ can ask a polynomially bounded number of queries adaptively again as in Phase 1. The same rule is applied here: $\mathcal{A}_I$ cannot extract the private key for ID_r^*. $\mathcal{A}_I$ cannot extract the partial private key for ID_r^* if the public key of this identity has been replaced before the challenge phase. In addition, $\mathcal{A}_I$ cannot make a unsigncryption query on σ^* under ID_s^* and ID_r^*, unless the public key $PK_{ID_s^*}$ or $PK_{ID_r^*}$ has been replaced after the challenge phase.

`Guess`: $\mathcal{A}_I$ produces a bit δ' and wins the game if $\delta' = \delta$.

The advantage of $\mathcal{A}_I$ is defined to be

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-I}}(\mathcal{A}_I) = |2\Pr[\delta' = \delta] - 1|,$$

where $\Pr[\delta' = \delta]$ denotes the probability that $\delta' = \delta$.

`IND-CCA2-II`: This is the game in which $\mathcal{A}_{II}$ interacts with the "challenger":

`Initial`: The challenger runs $(params, msk) \leftarrow$ `Setup`(1^k) and gives both $params$ and msk to $\mathcal{A}_{II}$.

`Phase 1`: The adversary $\mathcal{A}_{II}$ can perform a polynomially bounded number of queries in an adaptive manner. Note that we do not need `Extract partial private key` since $\mathcal{A}_{II}$ can computes partial private keys by itself.

- `Extract private key`: Same to the `IND-CCA2-I` game.
- `Request public key`: Same to the `IND-CCA2-I` game.

– `Signcryption queries`: Same to the `IND-CCA2-I` game.
– `Unsigncryption queries`: Same to the `IND-CCA2-I` game.

`Challenge`: The adversary $\mathcal{A}_{II}$ decides when Phase 1 ends. $\mathcal{A}_{II}$ generates two equal length plaintexts (m_0, m_1), a sender's identity ID_s^*, and a receiver's identity ID_r^* on which it wishes to be challenged. ID_r^* should not be queried to extract a private key in Phase 1. The challenger picks a random bit δ from $\{0,1\}$, computes $\sigma^* \leftarrow \texttt{Signcrypt}(params, m_\delta, S_{ID_s^*}, ID_s^*, PK_{ID_s^*}, ID_r^*, PK_{ID_r^*})$, and returns σ^* to $\mathcal{A}_{II}$.

`Phase 2`: The adversary $\mathcal{A}_{II}$ can ask a polynomially bounded number of queries adaptively again as in Phase 1. $\mathcal{A}_{II}$ cannot extract the private key for ID_r^*. In addition, $\mathcal{A}_{II}$ cannot make a unsigncryption query on σ^* under ID_s^* and ID_r^*, unless the public key $PK_{ID_s^*}$ or $PK_{ID_r^*}$ has been replaced after the challenge phase.

`Guess`: $\mathcal{A}_{II}$ produces a bit δ' and wins the game if $\delta' = \delta$.

The advantage of $\mathcal{A}_{II}$ is defined to be

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-II}}(\mathcal{A}_{II}) = |2\Pr[\delta' = \delta] - 1|,$$

where $\Pr[\delta' = \delta]$ denotes the probability that $\delta' = \delta$.

Definition 1. *A CLSC scheme is said to be IND-CCA2-I secure (resp. IND-CCA2-II secure) if there is no probabilistic polynomial time (PPT) adversary $\mathcal{A}_I$ (resp. $\mathcal{A}_{II}$) which wins* `IND-CCA2-I` *(resp.* `IND-CCA2-II`*) with non-negligible advantage. A CLSC scheme is said to be IND-CCA2 secure if it is both IND-CCA2-I secure and IND-CCA2-II secure.*

Notice that the adversary is allowed to extract the private key of ID_s^* in the `IND-CCA2-I` and `IND-CCA2-II` games. This condition corresponds to the stringent requirement of insider security for confidentiality of signcryption [3]. On the other hand, it ensures the forward security of the scheme, i.e. confidentiality is preserved in case the sender's private key becomes compromised.

For the strong existential unforgeability, we consider two games "`sUF-CMA-I`" and "`sUF-CMA-II`" where a Type I adversary $\mathcal{F}_I$ and a Type II adversary $\mathcal{F}_{II}$ interact with their "challenger" in these two games, respectively. Note that the challenger keeps a history of "query-answer" while interacting with the attackers. These two games are described as follows.

`sUF-CMA-I`: This is the game in which $\mathcal{F}_I$ interacts with the "challenger":

`Initial`: The challenger runs $(params, msk) \leftarrow \texttt{Setup}(1^k)$ and gives $params$ to $\mathcal{F}_I$. The challenger keeps master secret key msk to itself.

`Attack`: The adversary $\mathcal{F}_I$ performs a polynomially bounded number of queries just like in the `IND-CCA2-I` game.

`Forgery`: $\mathcal{F}_I$ produces a quaternion $(m^*, \sigma^*, ID_s^*, ID_r^*)$. Note that ID_s^* should not be queried to extract a private key. Note also that ID_s^* cannot be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. In addition, σ^* was not returned by the signcryption oracle on the input (m^*, ID_s^*, ID_r^*) during Attack stage. $\mathcal{F}_I$ wins the game if the result of $\texttt{Unsigncrypt}(params, \sigma^*, ID_s^*, PK_{ID_s^*}, S_{ID_r^*}, ID_r^*, PK_{ID_r^*})$ is not the $\perp$ symbol.

The advantage of $\mathcal{F}_I$ is defined as the probability that it wins.

`sUF-CMA-II`: This is the game in which $\mathcal{F}_{II}$ interacts with the "challenger":

`Initial:` The challenger runs $(params, msk) \leftarrow$ `Setup`(1^k) and gives both $params$ and msk to $\mathcal{F}_{II}$.

`Attack:` The adversary $\mathcal{F}_{II}$ performs a polynomially bounded number of queries just like in the `IND-CCA2-II` game.

`Forgery:` $\mathcal{F}_{II}$ produces a quaternion $(m^*, \sigma^*, ID_s^*, ID_r^*)$. ID_s^* should not be queried to extract a private key. In addition, σ^* was not returned by the signcryption oracle on the input (m^*, ID_s^*, ID_r^*) during Attack stage. $\mathcal{F}_{II}$ wins the game if the result of `Unsigncrypt`$(params, \sigma^*, ID_s^*, PK_{ID_s^*}, S_{ID_r^*}, ID_r^*, PK_{ID_r^*})$ is not the $\perp$ symbol.

The advantage of $\mathcal{F}_{II}$ is defined as the probability that it wins.

Definition 2. *A CLSC scheme is said to be sUF-CMA-I secure (resp. sUF-CMA-II secure) if there is no PPT adversary $\mathcal{F}_I$ (resp. $\mathcal{F}_{II}$) which wins* `sUF-CMA-I` *(resp.* `sUF-CMA-II`*) with non-negligible advantage. A CLSC scheme is said to be sUF-CMA secure if it is both sUF-CMA-I secure and sUF-CMA-II secure.*

Note that the adversary is allowed to extract the private key of ID_r^* in the above definition. Again, this condition corresponds to the stringent requirement of insider security for signcryption [3].

2.2 Date Encapsulation Mechanism (DEM)

A DEM is a symmetric encryption scheme which consists of the following two algorithms.

- `Enc:` This algorithm takes as input 1^k, a key K and a message $m \in \{0,1\}^*$, and outputs a ciphertext $c \in \{0,1\}^*$, where $K \in \mathcal{K}_{\text{DEM}}$ is a key in the given key space, and m is a bit string of arbitrary length. We denote this as $c \leftarrow$ `Enc`(K, m).
- `Dec:` This algorithm takes as input a key K and a ciphertext c, and outputs the message $m \in \{0,1\}^*$ or a symbol $\perp$ to indicate that the ciphertext is invalid.

For the purposes of this paper, it is only required that a DEM is secure with respect to indistinguishability against passive attackers (IND-PA). Formally, this security notion is captured by the following game played between a PPT adversary $\mathcal{A}$ and a challenger.

`Initial:` $\mathcal{A}$ runs on input 1^k and submits two equal length messages, m_0 and m_1.

`Challenge:` The challenger chooses a random key $K \in \mathcal{K}_{\text{DEM}}$ as well as a random bit $\lambda \in \{0,1\}$, and sends $c^* \leftarrow$ `Enc`(K, m_λ) to $\mathcal{A}$ as a challenge ciphertext.

`Guess:` The adversary $\mathcal{A}$ produces a bit λ' and wins the game if $\lambda' = \lambda$.

The advantage of $\mathcal{A}$ is defined to be

$$Adv_{\text{DEM}}^{\text{IND-PA}}(\mathcal{A}) = |2\Pr[\lambda' = \lambda] - 1|,$$

where $\Pr[\lambda' = \lambda]$ denotes the probability that $\lambda' = \lambda$.

Definition 3. *A DEM is said to be IND-PA secure if there is no PPT adversary $\mathcal{A}$ which wins the above game with non-negligible advantage.*

3 Certificateless Signcryption Tag-KEM (CLSC-TKEM)

In this section, we extend the concept of signcryption tag-KEM to the certificateless setting. We give the formal definition for certificateless signcryption tag-KEM (CLSC-TKEM).

3.1 Generic Scheme

A generic CLSC-TKEM consists of the following seven algorithms.

- `Setup`: Same to CLSC described in Section 2.
- `Partial-Private-Key-Extract`: Same to CLSC described in Section 2.
- `Generate-User-Keys`: Same to CLSC described in Section 2.
- `Set-Private-Key`: Same to CLSC described in Section 2.
- `Sym`: This is symmetric key generation algorithm which takes as input the $params$, the sender's full private key S_{ID_s}, identity ID_s and public key PK_{ID_s}, the receiver's identity ID_r and public key PK_{ID_r}, and outputs a symmetric key K together with internal state information ω. Here $K \in \mathcal{K}_{\mathrm{CLSC-TKEM}}$ is a key in the space of possible session keys at a given security level. We denote this as $(K, \omega) \leftarrow \mathtt{Sym}(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$.
- `Encap`: This is key encapsulation algorithm which takes as input the state information ω and an arbitrary tag τ, and returns an encapsulation $\psi \in \mathcal{E}_{\mathrm{CLSC-TKEM}}$. We denote this as $\psi \leftarrow \mathtt{Encap}(\omega, \tau)$.
- `Decap`: This is decapsulation algorithm which takes as input the $params$, an encapsulation ψ, a tag τ, the sender's identity ID_s and public key PK_{ID_s}, the receiver's full private key S_{ID_r}, identity ID_r and public key PK_{ID_r}, and outputs a key K or a special symbol $\perp$ indicating invalid encapsulation. We denote this as $K \leftarrow \mathtt{Decap}(params, \psi, \tau, ID_s, PK_{ID_s}, S_{ID_r}, ID_r, PK_{ID_r})$.

We make the consistency constraint that if

$$(K, \omega) \leftarrow \mathtt{Sym}(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r}) \text{ and } \psi \leftarrow \mathtt{Encap}(\omega, \tau),$$

then

$$K \leftarrow \mathtt{Decap}(params, \psi, \tau, ID_s, PK_{ID_s}, S_{ID_r}, ID_r, PK_{ID_r}).$$

3.2 Security Notions

A CLSC-TKEM should satisfy confidentiality and unforgeability. To define the security notions for CLSC-TKEM, we simply adapt the security notions of CLSC into the TKEM framework.

Again there are two types of adversary against a CLSC-TKEM: Type I and Type II. A Type I adversary models an attacker which is a common user of the system and is not in possession of the KGC's master secret key. But it is able to adaptively replace users'public keys with (valid) public keys of its choice. A Type II adversary models an honest-but-curious KGC who knows the KGC's master secret key. But it cannot replace users' public keys.

For the confidentiality, we consider two games "`IND-CCA2-I`" and "`IND-CCA2-II`" where a Type I adversary $\mathcal{A}_I$ and a Type II adversary $\mathcal{A}_{II}$ interact with their "challenger" in these two games, respectively. Note that the challenger keeps a history of "query-answer" while interacting with the attackers. Now we describe the two games.

`IND-CCA2-I`: This is the game in which $\mathcal{A}_I$ interacts with the "challenger":

`Initial`: The challenger runs $(params, msk) \leftarrow \texttt{Setup}(1^k)$ and gives $params$ to $\mathcal{A}_I$. The challenger keeps master secret key msk to itself.

`Phase 1`: The adversary $\mathcal{A}_I$ can perform a polynomially bounded number of queries in an adaptive manner.

- `Extract partial private key`: Same to CLSC's `IND-CCA2-I` game described in Section 2.
- `Extract private key`: Same to CLSC's `IND-CCA2-I` game described in Section 2.
- `Request public key`: Same to CLSC's `IND-CCA2-I` game described in Section 2.
- `Replace public key`: Same to CLSC's `IND-CCA2-I` game described in Section 2.
- `Symmetric key generation queries`: $\mathcal{A}_I$ chooses a sender's identity ID_s and a receiver's identity ID_r. The challenger finds S_{ID_s} from its "query-answer" list and runs $(K, \omega) \leftarrow \texttt{Sym}(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$. The challenger then stores the value ω (hidden from the view of the adversary, and overwriting any previously stored values), and sends the symmetric key K to $\mathcal{A}_I$. Note that, it is possible that the challenger is not aware of the sender's secret value, if the associated public key has been replaced. In this case, we require the adversary to provide it. We disallow queries where $ID_s = ID_r$.
- `Key encapsulation queries`: $\mathcal{A}_I$ produces an arbitrary tag τ. The challenger checks whether there exists a stored value ω. If not, it returns $\perp$ and terminates. Otherwise it erases the value from storage and returns $\psi \leftarrow \texttt{Encap}(\omega, \tau)$ to $\mathcal{A}_I$.
- `Key decapsulation queries`: The adversary $\mathcal{A}_I$ chooses a sender's identity ID_s, a receiver's identity ID_r, an encapsulation ψ, and a tag τ. The challenger finds S_{ID_r} from its "query-answer" list and sends the result of $\texttt{Decap}(params, \psi, \tau, ID_s, PK_{ID_s}, S_{ID_r}, ID_r, PK_{ID_r})$ to $\mathcal{A}_I$. Note that, it is possible that the challenger is not aware of the receiver's secret value, if the associated public key has been replaced. In this case, we require the adversary to provide it. We also disallow queries where $ID_s = ID_r$.

`Challenge`: The adversary $\mathcal{A}_I$ decides when Phase 1 ends. $\mathcal{A}_I$ generates a sender's identity ID_s^* and a receiver's identity ID_r^* on which it wishes to be challenged. Note that ID_r^* should not be queried to extract a private key in Phase 1. Note also that ID_r^* cannot be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. The challenger computes $(K_1, \omega^*) \leftarrow \texttt{Sym}(params, S_{ID_s^*}, ID_s^*, PK_{ID_s^*}, ID_r^*, PK_{ID_r^*})$. Then the challenger chooses $K_0 \leftarrow \mathcal{K}_{\mathrm{CLSC-TKEM}}$ and a bit $b \in \{0, 1\}$ randomly, and sends K_b to $\mathcal{A}_I$. When $\mathcal{A}_I$ receives K_b, it may ask the same queries as previously. Then $\mathcal{A}_I$ generates a tag τ^*. The challenger computes $\psi^* \leftarrow \texttt{Encap}(\omega^*, \tau^*)$ and sends it to $\mathcal{A}_I$ as a challenge encapsulation.

`Phase 2`: The adversary $\mathcal{A}_I$ can ask a polynomially bounded number of queries adaptively again as in Phase 1. The same rule is applied here: $\mathcal{A}_I$ cannot extract the private key for ID_r^*. $\mathcal{A}_I$ cannot extract the partial private key for ID_r^* if the public key of this identity has been replaced before the challenge phase. In addition, $\mathcal{A}_I$ cannot make a decapsulation query on (K_b, ψ^*) under ID_s^* and ID_r^*, unless the public key $PK_{ID_s^*}$ or $PK_{ID_r^*}$ has been replaced after the challenge phase.

`Guess`: The adversary $\mathcal{A}_I$ produces a bit b' and wins the game if $b' = b$.

The advantage of $\mathcal{A}_I$ is defined to be

$$\mathrm{Adv}_{\mathrm{CLSC-TKEM}}^{\mathrm{IND-CCA2-I}}(\mathcal{A}_I) = |2\Pr[b' = b] - 1|,$$

where $\Pr[b' = b]$ denotes the probability that $b' = b$.

`IND-CCA2-II`: This is the game in which $\mathcal{A}_{II}$ interacts with the "challenger":

`Initial`: The challenger runs $(params, msk) \leftarrow \texttt{Setup}(1^k)$ and gives both $params$ and msk to $\mathcal{A}_{II}$.

`Phase 1`: The adversary $\mathcal{A}_{II}$ can perform a polynomially bounded number of queries in an adaptive manner. Note that we do not need `Extract partial private key` since $\mathcal{A}_{II}$ can computes partial private keys by itself.

- `Extract private key`: Same to CLSC's `IND-CCA2-I` game described in Section 2.
- `Request public key`: Same to CLSC's `IND-CCA2-I` game described in Section 2.
- `Symmetric key generation queries`: Same to CLSC-TKEM's `IND-CCA2-I` game described in Section 3.
- `Key encapsulation queries`: Same to CLSC-TKEM's `IND-CCA2-I` game described in Section 3.
- `Key decapsulation queries`: Same to CLSC-TKEM's `IND-CCA2-I` game described in Section 3.

`Challenge`: The adversary $\mathcal{A}_{II}$ decides when Phase 1 ends. $\mathcal{A}_{II}$ generates a sender's identity ID_s^* and a receiver's identity ID_r^* on which it wishes to be challenged. Note that ID_r^* should not be queried to extract a private key in Phase 1. The challenger runs $(K_1, \omega^*) \leftarrow \texttt{Sym}(params, S_{ID_s^*}, ID_s^*, PK_{ID_s^*}, ID_r^*, PK_{ID_r^*})$. Then the challenger chooses $K_0 \leftarrow \mathcal{K}_{\text{CLSC-TKEM}}$ and a bit $b \in \{0,1\}$ randomly, and sends K_b to $\mathcal{A}_I$. When $\mathcal{A}_{II}$ receives K_b, it may ask the same queries as previously. Then $\mathcal{A}_{II}$ generates a tag τ^*. The challenger computes $\psi^* \leftarrow \texttt{Encap}(\omega^*, \tau^*)$ and sends it to $\mathcal{A}_{II}$ as a challenge encapsulation.

`Phase 2`: The adversary $\mathcal{A}_{II}$ can ask a polynomially bounded number of queries adaptively again as in Phase 1. $\mathcal{A}_{II}$ cannot extract the private key for ID_r^*. In addition, $\mathcal{A}_{II}$ cannot make a decapsulation query on (K_b, ψ^*) under ID_s^* and ID_r^*, unless the public key $PK_{ID_s^*}$ or $PK_{ID_r^*}$ has been replaced after the challenge phase.

`Guess`: The adversary $\mathcal{A}_{II}$ produces a bit b' and wins the game if $b' = b$.

The advantage of $\mathcal{A}_{II}$ is defined to be

$$\text{Adv}^{\text{IND-CCA2-II}}_{\text{CLSC-TKEM}}(\mathcal{A}_{II}) = |2\Pr[b' = b] - 1|,$$

where $\Pr[b' = b]$ denotes the probability that $b' = b$.

Definition 4. *A CLSC-TKEM scheme is said to be IND-CCA2-I secure (resp. IND-CCA2-II secure) if there is no PPT adversary $\mathcal{A}_I$ (resp. $\mathcal{A}_{II}$) which wins* `IND-CCA2-I` *(resp.* `IND-CCA2-II`*) with non-negligible advantage. A CLSC-TKEM scheme is said to be IND-CCA2 secure if it is both IND-CCA2-I secure and IND-CCA2-II secure.*

Notice that the adversary is allowed to extract the private key of ID_s^* in the `IND-CCA2-I` and `IND-CCA2-II` games. This condition corresponds to the stringent requirement of insider security for confidentiality of signcryption [3]. On the other hand, it ensures the forward security of the scheme, i.e. confidentiality is preserved in case the sender's private key becomes compromised.

For the strong existential unforgeability, we consider two games "`sUF-CMA-I`" and "`sUF-CMA-II`" where a Type I adversary $\mathcal{F}_I$ and a Type II adversary $\mathcal{F}_{II}$ interact with their "challenger" in these two games, respectively. Note that the challenger keeps a history of "query-answer" while interacting with the attackers. Now we describe the two games.

`sUF-CMA-I`: This is the game in which $\mathcal{F}_I$ interacts with the "challenger":

`Initial`: The challenger runs $(params, msk) \leftarrow \texttt{Setup}(1^k)$ and gives $params$ to $\mathcal{F}_I$. The challenger keeps master secret key msk to itself.

`Attack`: The adversary $\mathcal{F}_I$ performs a polynomially bounded number of queries just like in the CLSC-TKEM's `IND-CCA2-I` game.

`Forgery`: $\mathcal{F}_I$ produces a quaternion $(\tau^*, \psi^*, ID_s^*, ID_r^*)$. Note that ID_s^* should not be queried to extract a private key. Note also that ID_s^* cannot be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. In addition, ψ^* was not returned by the key encapsulation oracle on the input (τ^*, ID_s^*, ID_r^*) during Attack stage. $\mathcal{F}_I$ wins the game if the result of $\texttt{Decap}(params, \psi^*, \tau^*, ID_s^*, PK_{ID_s^*}, S_{ID_r^*}, ID_r^*, PK_{ID_r^*})$ is not the $\perp$ symbol.

The advantage of $\mathcal{F}_I$ is defined as the probability that it wins.

`sUF-CMA-II`: This is the game in which $\mathcal{F}_{II}$ interacts with the "challenger":

`Initial`: The challenger runs $(params, msk) \leftarrow \texttt{Setup}(1^k)$ and gives both $params$ and msk to $\mathcal{F}_{II}$.

`Attack`: The adversary $\mathcal{F}_{II}$ performs a polynomially bounded number of queries just like in the CLSC-TKEM's `IND-CCA2-II` game.

`Forgery`: $\mathcal{F}_{II}$ produces a quaternion $(\tau^*, \psi^*, ID_s^*, ID_r^*)$. ID_s^* should not be queried to extract a private key. In addition, ψ^* was not returned by the key encapsulation oracle on the input (τ^*, ID_s^*, ID_r^*) during Attack stage. $\mathcal{F}_{II}$ wins the game if the result of $\texttt{Decap}(params, \psi^*, \tau^*, ID_s^*, PK_{ID_s^*}, S_{ID_r^*}, ID_r^*, PK_{ID_r^*})$ is not the $\perp$ symbol.

The advantage of $\mathcal{F}_{II}$ is defined as the probability that it wins.

Definition 5. *A CLSC-TKEM scheme is said to be sUF-CMA-I secure (resp. sUF-CMA-II secure) if there is no PPT adversary $\mathcal{F}_I$ (resp. $\mathcal{F}_{II}$) which wins* `sUF-CMA-I` *(resp.* `sUF-CMA-II`*) with non-negligible advantage. A CLSC-TKEM scheme is said to be sUF-CMA secure if it is both sUF-CMA-I secure and sUF-CMA-II secure.*

Note that the adversary is allowed to extract the private key of ID_r^* in the above definition. Again, this condition corresponds to the stringent requirement of insider security for signcryption [3].

4 Certificateless Hybrid Signcryption

We can combine a CLSC-TKEM with a DEM to form a CLSC scheme. We describe it in Figure 1. Note that the tag is the ciphertext output by the DEM. Such construction yields simpler scheme descriptions and better generic security reductions.

We give the security results for such construction in Theorems 1 and 2.

Theorem 1. *Let CLSC be a certificateless hybrid signcryption scheme constructed from a CLSC-TKEM and a DEM. If the CLSC-TKEM is IND-CCA2 secure and the DEM is IND-PA secure,* `CLSC.Setup`: On input 1^k:

1. $(params, msk) \leftarrow$ `CLSC-TKEM.Setup`(1^k)

2. Output the system parameters $params$ and the master secret key msk

`CLSC.Partial-Private-Key-Extract`: On input the $params$, msk, and an identity $ID \in \{0,1\}^*$:

1. $D_{ID} \leftarrow$ `CLSC-TKEM.Partial-Private-Key-Extract`$(params, msk, ID)$

2. Output the partial private key D_{ID} of the identity ID

`CLSC.Generate-User-Keys`: On input the $params$ and an identity $ID \in \{0,1\}^*$:

1. $(x_{ID}, PK_{ID}) \leftarrow$ `CLSC-TKEM.Generate-User-Keys`$(params, ID)$

2. Output the secret value x_{ID} and the public key PK_{ID} of the identity ID

`CLSC.Set-Private-Key`: On input the partial private key D_{ID} and the secret value x_{ID}:

1. $S_{ID} \leftarrow$ `CLSC-TKEM.Set-Private-Key`(D_{ID}, x_{ID})

2. Output the full private key S_{ID}

`CLSC.Signcrypt`: On input the $params$, a message $m \in \{0,1\}^*$, the sender's full private key S_{ID_s}, identity ID_s and public key PK_{ID_s}, the receiver's identity ID_r and public key PK_{ID_r}:

1. $(K, \omega) \leftarrow$ `CLSC-TKEM.Sym`$(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$

2. $c \leftarrow$ `DEM.Enc`(K, m)

3. $\psi \leftarrow$ `CLSC-TKEM.Encap`(ω, c)

4. Output the ciphertext $\sigma \leftarrow (\psi, c)$

`CLSC.Unsigncrypt`: On input the $params$, a ciphertext σ, the sender's identity ID_s and public key PK_{ID_s}, the receiver's full private key S_{ID_r}, identity ID_r and public key PK_{ID_r}:

1. $K \leftarrow$`CLSC-TKEM.Decap`$(params, \psi, c, ID_s, PK_{ID_s}, S_{ID_r}, ID_r, PK_{ID_r})$

2. If $K = \perp$, then output $\perp$ and stop

3. $m \leftarrow$`DEM.Dec`(K, c)

4. Output the message m

Fig. 1. Certificateless hybrid signcryption

then CLSC is IND-CCA2 secure. In particular, we have

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-i}}(\mathcal{A}) \leq 2\mathrm{Adv}_{\mathrm{CLSC-TKEM}}^{\mathrm{IND-CCA2-i}}(\mathcal{B}_1) + \mathrm{Adv}_{\mathrm{DEM}}^{\mathrm{IND-PA}}(\mathcal{B}_2),$$

where $i \in \{I, II\}$

Proof. See the appendix A. □

Theorem 2. *Let CLSC be a certificateless hybrid signcryption scheme constructed from a CLSC-TKEM and a DEM. If the CLSC-TKEM is sUF-CMA secure, then CLSC is sUF-CMA secure. In* *particular, we have*

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-i}}(\mathcal{F}) \leq \mathrm{Adv}_{\mathrm{CLSC-TKEM}}^{\mathrm{sUF-CMA-i}}(\mathcal{B}),$$

where $i \in \{I, II\}$*,* $\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-i}}(\mathcal{F})$ *is the advantage of the sUF-CMA adversary against CLSC, and* $\mathrm{Adv}_{\mathrm{CLSC-TKEM}}^{\mathrm{sUF-CMA-i}}(\mathcal{B})$ *is the advantage of the resulting sUF-CMA adversary against CLSC-TKEM.*

Proof. See the appendix B. □

5 An Example of CLSC-TKEM

The Barbosa-Farshim CLSC scheme [6] fits the new generic framework. Here we give an example of CLSC-TKEM based on the Barbosa-Farshim scheme. If we combine the CLSC-TKEM with a DEM as Figure 1, we can get a scheme that is very similar to the Barbosa-Farshim scheme. Since the Barbosa-Farshim scheme uses the bilinear pairings, we describe some basic knowledge about bilinear pairings in the appendix C.

5.1 CLSC-TKEM

The CLSC-TKEM consists of the following seven algorithms.

- `Setup:` Define G_1, G_2 and $\hat{e}$ as in appendix C. Let H_1, H_2, H_3, and H_4 be four cryptographic hash functions where $H_1 : \{0,1\}^* \to G_1$, $H_2 : \{0,1\}^* \to \{0,1\}^n$, $H_3 : \{0,1\}^* \to G_1$, and $H_4 : \{0,1\}^* \to G_1$. Here n is the key length of a DEM. Let P be a generator of G_1. The PKG chooses a master secret key $s \in Z_q^*$ randomly and computes $P_{pub} \leftarrow sP$. The PKG publishes system parameters $\{G_1, G_2, n, \hat{e}, P, P_{pub}, H_1, H_2, H_3, H_4\}$ and keeps the master key s secret.
- `Partial-Private-Key-Extract:` Given an identity $ID \in \{0,1\}^*$, the PKG computes $Q_{ID} \leftarrow H_1(ID)$ and returns the partial private key $D_{ID} \leftarrow sQ_{ID}$.
- `Generate-User-Keys:` A user with identity ID chooses a random element x_{ID} from Z_q as the secret value, and sets $PK_{ID} \leftarrow x_{ID}P$ as the public key.
- `Set-Private-Key:` Given a partial private key D_{ID} and a secret value x_{ID}, this algorithm returns the full private key $S_{ID} \leftarrow (x_{ID}, D_{ID})$.
- `Sym:` Given the sender's full private key S_{ID_s}, identity ID_s and public key PK_{ID_s}, the receiver's identity ID_r and public key PK_{ID_r}, this algorithm works as follows.
  1. Choose $r \in Z_q^*$ randomly.
  2. Compute $U = rP$ and $T \leftarrow \hat{e}(P_{pub}, Q_{ID_r})^r$.
  3. Compute $K \leftarrow H_2(U, T, rPK_{ID_r}, ID_r, PK_{ID_r})$.
  4. Output K and set $\omega \leftarrow (r, U, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$.
- `Encap:` Given the state information ω and an arbitrary tag τ, this algorithm works as follows.
  1. Compute $H \leftarrow H_3(U, \tau, ID_s, PK_{ID_s})$.
  2. Compute $H' \leftarrow H_4(U, \tau, ID_s, PK_{ID_s})$.
  3. Compute $W \leftarrow D_{ID_s} + rH + x_{ID_s}H'$
  4. Output $\psi \leftarrow (U, W)$

– `Decap`: Given the the sender's identity ID_s and public key PK_{ID_s}, the receiver's full private key S_{ID_r}, identity ID_r and public key PK_{ID_r}, an encapsulation ψ and a tag τ, this algorithm works as follows.
  1. Compute $H \leftarrow H_3(U, \tau, ID_s, PK_{ID_s})$.
  2. Compute $H' \leftarrow H_4(U, \tau, ID_s, PK_{ID_s})$.
  3. If $\hat{e}(P_{pub}, Q_{ID_s})\hat{e}(U, H)\hat{e}(PK_{ID_s}, H') = \hat{e}(P, W)$, compute $T = \hat{e}(D_{ID_r}, U)$ and output the $K \leftarrow H_2(U, T, x_{ID_r}U, ID_r, PK_{ID_r})$. Otherwise, output symbol $\perp$.

5.2 Security

We give the security results for the CLSC-TKEM in Theorems 3 and 4.

Theorem 3. *In the random oracle model, the above CLSC-TKEM is IND-CCA2 secure under the assumption that the gap bilinear Diffie-Hellman problem is intractable.*

Proof. See the appendix D. □

Theorem 4. *In the random oracle model, the above CLSC-TKEM is sUF-CMA secure under the assumption that the GDH′ problem is intractable.*

Proof. See the appendix E. □

6 Conclusions

In this paper, we extended the concept of signcryption tag-KEM to the certificateless setting. We showed that a certificateless signcryption scheme can be constructed by combining a certificateless signcryption tag-KEM with a DEM. To show that our framework is reasonable, we also gave an example of certificateless signcryption tag-KEM based on the Barbosa-Farshim certificateless signcryption scheme.

Acknowledgements

This work is supported by the National Natural Science Foundation of China (Grant Nos. 60673075, 60803133 and 60873233), the Key Laboratory of Computer Networks and Information Security of Xidian University (2008CNIS-02), and the Youth Science and Technology Foundation of UESTC. Fagen Li is supported by the JSPS postdoctoral fellowship for research in Japan.

References

1. M. Abe, R. Gennaro, and K. Kurosawa. Tag-KEM/DEM: a new framework for hybrid encryption. *Journal of Cryptology*, Vol. 21, No. 1, pp. 97–130, 2008.
2. S.S. Al-Riyami and K.G. Paterson. Certificateless public key cryptography. In *Advances in Cryptology-ASIACRYPT 2003*, LNCS 2894, pp. 452–474, Springer-Verlag, 2003.
3. J.H. An, Y. Dodis, T. Rabin. On the security of joint signature and encryption. In *Advances in Cryptology-EUROCRYPT 2002*, LNCS 2332, pp. 83–107, Springer-Verlag, 2002.

4. J. Baek, R. Steinfeld, and Y. Zheng. Formal proofs for the security of signcryption. *Journal of Cryptology*, Vol. 20, No 2, pp. 203–235, 2007.
5. F. Bao and R.H. Deng. A signcryption scheme with signature directly verifiable by public key. In *Public Key Cryptography-PKC'98*, LNCS 1431, pp. 55–59, Springer-Verlag, 1998.
6. M. Barbosa and P. Farshim. Certificateless signcryption. In *ACM Symposium on Information, Computer and Communications Security-ASIACCS 2008*, pp. 369–372, Tokyo, Japan, 2008.
7. P.S.L.M. Barreto, B. Libert, N. McCullagh, and J.J. Quisquater. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In *Advances in Cryptology-ASIACRYPT 2005*, LNCS 3788, pp. 515–532, Springer-Verlag, 2005.
8. K. Bentahar, P. Farshim, J. Malone-Lee, and N.P. Smart. Generic constructions of identity-based and certificateless KEMs. *Journal of Cryptology*, Vol. 21, No 2, pp. 178–199, 2008.
9. T.E. Bjørstad and A.W. Dent. Building better signcryption schemes with tag-KEMs. In *Public Key Cryptography-PKC 2006*, LNCS 3958, pp. 491–507, Springer-Verlag, 2006.
10. D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology-CRYPTO 2001*, LNCS 2139, pp. 213–229, Springer-Verlag, 2001.
11. X. Boyen. Multipurpose identity-based signcryption: a swiss army knife for identity-based cryptography. In *Advances in Cryptology-CRYPTO 2003*, LNCS 2729, pp. 383–399, Springer-Verlag, 2003.
12. L. Chen and J. Malone-Lee. Improved identity-based signcryption. In *Public Key Cryptography-PKC 2005*, LNCS 3386, pp. 362–379, Springer-Verlag, 2005.
13. L. Chen, Z. Cheng, J. Malone-Lee, N.P. Smart. Efficient ID-KEM based on the Sakai-Kasahara key construction. *IEE Proceedings-Information Security*, Vol. 153, No 1, pp. 19–26, 2006.
14. S.S.M. Chow, S.M. Yiu, L.C.K. Hui, and K.P. Chow. Efficient forward and provably secure ID-based signcryption scheme with public verifiability and public ciphertext authenticity. In *Information Security and Cryptology-ICISC 2003*, LNCS 2971, pp. 352–369, Springer-Verlag, 2004.
15. R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, Vol. 33, No. 1, pp. 167–226, 2003.
16. A.W. Dent. Hybrid signcryption schemes with outsider security. In *Information Security-ISC 2005*, LNCS 3650, pp. 203–217, Springer-Verlag, 2005.
17. A.W. Dent. Hybrid signcryption schemes with insider security. In *Information Security and Privacy-ACISP 2005*, LNCS 3574, pp. 253–266, Springer-Verlag, 2005.
18. A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology-CRYPTO'86*, LNCS 263, pp. 186–194, Springer-Verlag, 1986.
19. C. Gamage, J. Leiwo, and Y. Zheng. Encrypted message authentication by firewalls. In *Public Key Cryptography-PKC'99*, LNCS 1560, pp. 69–81, Springer-Verlag, 1999.
20. L. Guillou and J.J. Quisquater. A "Paradoxical" Identity-based signature scheme resulting from zero-knowledge. In *Advances in Cryptology-CRYPTO'88*, LNCS 403, pp. 216–231, Springer-Verlag, 1988.
21. Q. Huang and D.S. Wong. Generic certificateless key encapsulation mechanism. In *Information Security and Privacy-ACISP 2007*, LNCS 4586, pp. 215–229, Springer-Verlag, 2007.
22. H.Y. Jung, D.H. Lee, J.I. Lim, and K.S. Chang. Signcryption schemes with forward secrecy. In *Information Security Application-WISA 2001*, pp. 463–475, Seoul, Korea, 2001.
23. E. Kiltz and D. Galindo. Direct chosen-ciphertext secure identity-based key encapsulation without random oracles. In *Information Security and Privacy-ACISP 2006*, LNCS 4058, pp. 336–347, Springer-Verlag, 2006.
24. E. Kiltz. Chosen-ciphertext secure key-encapsulation based on gap hashed diffie-hellman. In *Public Key Cryptography-PKC 2007*, LNCS 4450, pp. 282–297, Springer-Verlag, 2007.
25. K. Kurosawa and Y. Desmedt. A new paradigm of hybrid encryption scheme. In *Advances in Cryptology-CRYPTO 2004*, LNCS 3152, pp. 426–442, Springer-Verlag, 2004.
26. B. Libert and J.J. Quisquater. A new identity based signcryption schemes from pairings. In *2003 IEEE Information Theory Workshop*, pp. 155–158, Paris, France, 2003.
27. F. Li, Y. Hu, and Chuanrong Zhang. An identity-based signcryption scheme for multi-domain ad hoc networks. In *Applied Cryptography and Network Security-ACNS 2007*, LNCS 4521, pp. 373–384, Springer-Verlag, 2007.
28. J. Malone-Lee. Identity based signcryption. *Cryptology ePrint Archive*, Report 2002/098, 2002.
29. J. Malone-Lee and W. Mao. Two birds one stone: signcryption using RSA. In *Topics in Cryptology-CT-RSA 2003*, LNCS 2612, pp. 211–226, Springer-Verlag, 2003.

30. Y. Mu and V. Varadharajan. Distributed signcryption. In *Progress in Cryptology-INDOCRYPT 2000*, LNCS 1977, pp. 155–164, Springer-Verlag, 2000.
31. R. Sakai and M. Kasahara. ID based cryptosystems with pairing on elliptic curve. *Cryptology ePrint Archive*, Report 2003/054, 2003.
32. M. Seo and K. Kim. Electronic funds transfer protocol using domain-verifiable signcryption scheme. In *Information Security and Cryptology-ICISC'99*, LNCS 1787, pp. 269–277, Springer-Verlag, 1999.
33. A. Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology-CRYPTO'84*, LNCS 196, pp. 47–53, Springer-Verlag, 1984.
34. C.H. Tan. Insider-secure signcryption KEM/tag-KEM schemes without random oracles. In *The Third International Conference on Availability, Reliability and Security-ARES 2008*, pp. 1275–1281, Barcelona, Spain, 2008.
35. B. Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology-EUROCRYPT 2005*, LNCS 3494, pp. 114–127, Springer-Verlag, 2005,
36. D.H. Yum and P.J. Lee. New signcryption schemes based on KCDSA. In *Information Security and Cryptology-ICISC 2001*, LNCS 2288, pp. 305–317, Springer-Verlag, 2002.
37. V. Shoup. OAEP reconsidered. In *Advances in Cryptology-CRYPTO 2001*, LNCS 2139, pp. 239–259, Springer-Verlag, 2001.
38. Y. Zheng. Digital signcryption or how to achieve cost (signature & encryption) $\ll$ cost (signature) + cost(encryption). In *Advances in Cryptology-CRYPTO'97*, LNCS 1294, pp. 165–179, Springer-Verlag, 1997.
39. Y. Zheng and H. Imai. How to construct efficient signcryption schemes on elliptic curves. *Information Processing Letters*, Vol. 68, No.5, pp. 227–233, 1998.

Appendix

A Proof of Theorem 1

Proof. Our proof strategy is as follows. We define a sequence $\mathtt{Game}_0$, $\mathtt{Game}_1$, $\mathtt{Game}_2$ of modified attack games. The only difference between games is how the environment responds to $\mathcal{A}$'s oracle queries.

Let $\sigma^* \leftarrow (\psi^*, c^*)$ be the challenge ciphertext submitted to $\mathcal{A}$ by its challenge oracle that encrypts either m_0 or m_1 according to a bit b. Let K^* denote the symmetric key used by the challenge oracle in the generation of the challenge ciphertext, or alternatively, the decapsulation of ψ^* using the identities ID_s^* and ID_r^* that are chosen by the adversary. For any $i = 0, 1, 2$, we let S_i be the event that $\delta' = \delta$ in game $\mathtt{Game}_i$, where δ is the bit chosen by $\mathcal{A}$'s challenge oracle and δ' is the bit output by $\mathcal{A}$. This probability is taken over the random choices of $\mathcal{A}$ and those of $\mathcal{A}$'s oracles.

We will use the following useful Lemma 1 from [37].

Lemma 1. *Let E, E', and F be events defined on a probability space such that* $\Pr[E \wedge \neg F] = \Pr[E' \wedge \neg F]$. *Then we have*

$$|\Pr[E] - \Pr[E']| \leq \Pr[F].$$

$\mathtt{Game}_0$: We simulate the view of the adversary in a real attack by running the suitable key generation algorithms and using the resulting keys to respond to $\mathcal{A}$'s queries. So the view of $\mathcal{A}$ is the same as it would be in a real attack. Therefore, we have

$$|\Pr[S_0] - \frac{1}{2}| = \frac{1}{2}\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-i}}(\mathcal{A}),$$

where $i \in \{I, II\}$.

$\mathtt{Game}_1$: In this game, we slightly modify how the unsigncryption oracle responds to queries from $\mathcal{A}$. When a sender' identity ID_s, a receiver's identity ID_r, and (ψ, c) is presented to the unsigncryption oracle after the invocation of the challenge signcryption oracle, if $ID_s = ID_s^*$, $ID_r = ID_r^*$ and $\psi = \psi^*$, and in the case of a Type I adversary, the public keys of ID_s^* and ID_r^* have not been replaced, then the unsigncryption oracle does not use the genuine unsigncryption procedure for the hybrid scheme, instead it uses the key K^* to decrypt c and returns the result to the adversary $\mathcal{A}$.

Clearly this change has no impact on the adversary and so

$$\Pr[S_1] = \Pr[S_0].$$

$\mathtt{Game}_2$: In this game, we modify $\mathtt{Game}_1$ by replacing K^* with a random key K' from $\mathcal{K}_{\text{DEM}}$. The result then follows from the following Lemmas 2 and 3. □

Lemma 2. *There exists a PPT algorithm $\mathcal{B}_1$, whose running time is essentially the same as that of $\mathcal{A}$, such that*

$$|\Pr[S_2] - \Pr[S_1]| = \text{Adv}_{\text{CLSC-TKEM}}^{\text{IND-CCA2-i}}(\mathcal{B}_1),$$

where $i \in \{I, II\}$.

Proof. To prove this we demonstrate how to construct an adversary $\mathcal{B}_1$ of the CLSC-TKEM to violate the IND-CCA2-I (resp. IND-CCA2-II) attack.

Adversary $\mathcal{B}_1$ is constructed by running adversary $\mathcal{A}$. We respond to $\mathcal{A}$'s queries as follows.

- When $\mathcal{A}$ calls any oracle, bar its signcryption, unsigncryption and challenge signcryption oracles, $\mathcal{B}_1$ simply relays these queries to its own equivalent oracle.
- When $\mathcal{A}$ make a signcryption query with a sender's identity ID_s, a receiver's identity ID_r and a plaintext m, $\mathcal{B}_1$ follows the steps below.
  1. Make a symmetric key generation query on (ID_s, ID_r) to its own symmetric key generation oracle to obtain K.
  2. Compute $c \leftarrow \mathtt{DEM.Enc}(K, m)$.
  3. Make a key encapsulation query on c to its own key encapsulation oracle to obtain ψ.
  4. Return the ciphertext $\sigma \leftarrow (\psi, c)$ to $\mathcal{A}$.
- When $\mathcal{A}$ make a unsigncryption query with a sender's identity ID_s, a receiver's identity ID_r and a ciphertext $\sigma \leftarrow (\psi, c)$, $\mathcal{B}_1$ follows the steps below.
  1. Make a key decapsulation query on (ψ, c, ID_s, ID_r) to its own key decapsulation oracle to obtain K.
  2. If $K = \perp$, return $\perp$ and stop.
  3. Compute $m \leftarrow \mathtt{DEM.Dec}(K, c)$ and return m.
- When $\mathcal{A}$ calls its challenge signcryption oracle with two equal length plaintexts m_0, m_1, a sender's identity ID_s^*, and a receiver's identity ID_r^*, $\mathcal{B}_1$ follows the steps below.
  1. Submit ID_s^* and ID_r^* to its challenger to obtain K_b, where $b \in \{0, 1\}$.
  2. Pick a random bit δ from $\{0, 1\}$.
  3. Compute $c^* \leftarrow \mathtt{DEM.Enc}(K_b, m_\delta)$.
  4. Submit c^* to its challenger to obtain ψ^*.
  5. Return the ciphertext $\sigma^* \leftarrow (\psi^*, c^*)$ to $\mathcal{A}$.

- To respond to $\mathcal{A}$'s unsigncryption query for a sender's identity ID_s, a receiver's identity ID_r and a ciphertext $\sigma \leftarrow (\psi, c)$ after $\mathcal{A}$ has queried its challenge signcryption oracle, $\mathcal{B}_1$ proceeds as follows.
  - If $(ID_s, ID_r, \psi) \neq (ID_s^*, ID_r^*, \psi^*)$ then it uses the same procedure that it used before $\mathcal{A}$'s call to its challenge signcryption oracle.
  - In the case of a Type I adversary against a CLSC scheme, if $(ID_s, ID_r, \psi) = (ID_s^*, ID_r^*, \psi^*)$ and the public keys have been replaced, then $\mathcal{B}_1$ responds by calling the key decapsulation oracle provided to it by $\mathcal{A}$ with input $(ID_s^*, ID_r^*, \psi^*, c^*)$ to obtain K. It then uses K to decrypt c and relays the response to $\mathcal{A}$.
  - Otherwise, $\mathcal{B}_1$ uses K_b to decrypt c and relays the result to $\mathcal{A}$.

At the end of the simulation, $\mathcal{A}$ outputs δ'. If $\delta' = \delta$, $\mathcal{B}_1$ outputs $b' = 1$ indicating K_b is the real key; otherwise it outputs $b' = 0$ indicating K_b is a random key.

When K_b is the real key, $\mathcal{A}$ is run exactly as it would be run in $\texttt{Game}_1$. This means that

$$\Pr[S_1] = \Pr[\delta' = \delta | b = 1] = \Pr[b' = 1 | b = 1].$$

When K_b is the random key, $\mathcal{A}$ is run exactly as it would be in $\texttt{Game}_2$. This means that

$$\Pr[S_2] = \Pr[\delta' = \delta | b = 0] = \Pr[b' = 1 | b = 0].$$

From the definition of security for CLSC-TKEM, we have

$$\mathrm{Adv}_{\mathrm{CLSC-TKEM}}^{\mathrm{IND-CCA2-i}}(\mathcal{B}_1) = |2\Pr[b' = b] - 1| = |\Pr[b' = 1|b = 1] - \Pr[b' = 1|b = 0]|.$$

So the result holds. □

Lemma 3. *There exists a PPT algorithm $\mathcal{B}_2$, whose running time is essentially the same as that of $\mathcal{A}$, such that*

$$|\Pr[S_2] - \frac{1}{2}| = \frac{1}{2}\mathrm{Adv}_{\mathrm{DEM}}^{\mathrm{IND-PA}}(\mathcal{B}_2).$$

Proof. To construct such a $\mathcal{B}_2$ we simply run $\mathcal{A}$ as it would be run in game $\texttt{Game}_2$. We run the suitable CLSC-TKEM algorithms so we can respond to $\mathcal{A}$'s queries before it calls its challenge signcryption oracle. When $\mathcal{A}$ calls its challenge signcryption oracle with a sender's identity ID_s^*, a receiver's identity ID_r^*, and messages (m_0, m_1), we simply relay (m_0, m_1) to the challenge encryption oracle of $\mathcal{B}_2$ to obtain c^*. We then make a symmetric key generation query and a key encapsulation query to obtain K^* and ψ^*, respectively. We discard K^* and return (ψ^*, c^*) to $\mathcal{A}$. We continue to respond to $\mathcal{A}$'s queries as before except if it a makes unsigncryption query on $(ID_s^*, ID_r^*, \psi^*, c)$ for some c. In this instance there are two cases:

- If we are dealing with a Type I adversary $\mathcal{A}$ of a CLSC scheme, and the public keys have been replaced, then $\mathcal{B}_2$ decapsulates $(ID_s^*, ID_r^*, \psi^*, c)$ using the provided secret key to obtain K, decrypts c and relays the response to $\mathcal{A}$.
- Otherwise we query $\mathcal{B}_2$'s decryption oracle with c and relay the response to $\mathcal{A}$.

In this simulation $\mathcal{A}$ is run by $\mathcal{B}_2$ in exactly the same manner as the former would be run in game $\texttt{Game}_2$; moreover, $\Pr[S_2]$ corresponds exactly to the probability that $\mathcal{B}_2$ correctly determines the hidden bit of its challenge encryption oracle since $\mathcal{B}_2$ outputs whatever $\mathcal{A}$ outputs. The result follows. □

B Proof of Theorem 2

Proof. Suppose that $\mathcal{F}$ is an adversary that breaks the CLSC scheme with probability $\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-i}}(\mathcal{F})$, where $i \in \{I, II\}$. We use this to construct an algorithm $\mathcal{B}$ that breaks the sUF-CMA-i for the CLSC-TKEM with probability at least $\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-i}}(\mathcal{F})$ too.

Adversary $\mathcal{B}$ is constructed by running adversary $\mathcal{F}$. We respond to $\mathcal{F}$'s queries as follows.

- When $\mathcal{F}$ calls any oracle, bar its signcryption and unsigncryption oracles, $\mathcal{B}$ simply relays these queries to its own equivalent oracle.
- When $\mathcal{F}$ make a signcryption query with a sender's identity ID_s, a receiver's identity ID_r and a plaintext m, $\mathcal{B}$ follows the steps below.
  1. Make a symmetric key generation query on (ID_s, ID_r) to its own symmetric key generation oracle to obtain K.
  2. Compute $c \leftarrow \mathtt{DEM.Enc}(K, m)$.
  3. Make a key encapsulation query on c to its own key encapsulation oracle to obtain ψ.
  4. Return the ciphertext $\sigma \leftarrow (\psi, c)$ to $\mathcal{F}$.
- When $\mathcal{F}$ make a unsigncryption query with a sender's identity ID_s, a receiver's identity ID_r and a ciphertext $\sigma \leftarrow (\psi, c)$, $\mathcal{B}$ follows the steps below.
  1. Make a key decapsulation query on (ψ, c, ID_s, ID_r) to its own key decapsulation oracle to obtain K.
  2. If $K = \perp$, return $\perp$ and stop.
  3. Compute $m \leftarrow \mathtt{DEM.Dec}(K, c)$ and return m.

Finally, $\mathcal{F}$ outputs a forgery $(m^*, \sigma^*, ID_s^*, ID_r^*)$, where $(\psi^*, c^*) \leftarrow \sigma^*$. $\mathcal{B}$ outputs $(\tau^*, \psi^*, ID_s^*, ID_r^*)$, where $\tau^* = c^*$.

Clearly, this algorithm perfectly simulates the environment in which $\mathcal{F}$ should be running. If $\mathcal{F}$ wins the sUF-CMA-i for the CLSC, $\mathcal{B}$ have the same probability to win the sUF-CMA-i for CLSC-TKEM. □

C Bilinear Pairings

Let G_1 be a cyclic additive group generated by P, whose order is a prime q, and G_2 be a cyclic multiplicative group of the same order q. A bilinear pairing is a map $\hat{e} : G_1 \times G_1 \rightarrow G_2$ with the following properties:

1. Bilinearity: $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ for all $P, Q \in G_1$, $a, b \in Z_q$.
2. Non-degeneracy: There exists P and $Q \in G_1$ such that $\hat{e}(P, Q) \neq 1$.
3. Computability: There is an efficient algorithm to compute $\hat{e}(P, Q)$ for all P,$Q \in G_1$.

The modified Weil pairing and the Tate pairing [10] are admissible maps of this kind. The security of our scheme described here relies on the hardness of the following problems.

Definition 6. *We say the gap bilinear Diffie-Hellman (GBDH) assumption holds if the advantage of any PPT adversary as defined below is negligible.*

$$\mathrm{Adv}_{GBDH}(\mathcal{A}, q_{DBDH}) = \Pr[T = \hat{e}(P, P)^{abc} | a, b, c \leftarrow Z_q; T \leftarrow \mathcal{A}^{\mathcal{O}}(P, aP, bP, cP)]$$

In the above equation, $\mathcal{O}$ denotes a decision bilinear Diffie-Hellman oracle which on input (P, aP, bP, cP, T) outputs 1 if $T = \hat{e}(P,P)^{abc}$ and 0 otherwise. By q_{DBDH} we denote the maximum number of queries that $\mathcal{A}$ asks its decision oracle.

The following weaker assumption is implied by the above.

Definition 7. *We say the computational Diffie-Hellman assumption in the presence of a decision bilinear Diffie-Hellman oracle (GDH′) holds in G_1 if the advantage of any PPT adversary as defined below is negligible.*

$$\mathrm{Adv}_{GDH'}(\mathcal{A}, q_{DBDH}) = \Pr[Q = abP | a, b \leftarrow Z_q; Q \leftarrow \mathcal{A}^{\mathcal{O}}(P, aP, bP)]$$

Here $\mathcal{O}$ and q_{DBDH} are as in the above definition.

This assumption in turn implies:

Definition 8. *We say the computational Diffie-Hellman (CDH) assumption holds in G_1 if the advantage of any PPT adversary as defined below is negligible.*

$$\mathrm{Adv}_{CDH}(\mathcal{A}) = \Pr[Q = abP | a, b \leftarrow Z_q; Q \leftarrow \mathcal{A}(P, aP, bP)]$$

D Proof of Theorem 3

Proof. In the Barbosa-Farshim CLSC scheme [6], they use a weaker formulation of Type I adversary which they refer to as Type I′. In confidentiality games, the Type I′ adversary is not allowed to extract the partial private key of ID_r^*. They proved that If a CLSC scheme is IND-CCA2 secure against Type II and Type I′ attackers, then it is also IND-CCA2 secure against Type I attackers. It is easy to extend this conclusion to CLSC-TKEM setting. That is, we have the following Lemma 4.

Lemma 4. *If a CLSC-TKEM is IND-CCA2 secure against Type II and Type I′ attackers then it is also IND-CCA2 secure against Type I attackers. In particular, we have*

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-I}}(\mathcal{A}) \leq 2\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-I'}}(\mathcal{C}_1) + \mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-II}}(\mathcal{C}_2).$$

This theorem follows from Lemmas 4, 5 and 6. □

Lemma 5. *Under the GBDH assumption, no PPT attacker $\mathcal{A}$ has non-negligible advantage in winning the IND-CCA2-I′ game against the above CLSC-TKEM, when all hash functions are modeled as random oracles. More precisely, there exists an algorithm $\mathcal{C}$ which uses $\mathcal{A}$ to solve the GBDH problem such that:*

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-I'}}(\mathcal{A}) \leq q_T \mathrm{Adv}_{GBDH}(\mathcal{C}, q_D^2 + 2q_D q_2 + q_2),$$

where $q_T = q_1 + q_P + q_K + 2q_D + 2$. Here q_1, q_2, q_P, q_K and q_D are the maximum number of queries that the adversary can ask H_1, ask H_2, extract partial private key, extract private key and make key decapsulation queries.

Proof. The challenger $\mathcal{C}$ takes as input (P, aP, bP, cP) and attempts to compute $\hat{e}(P,P)^{abc}$. $\mathcal{C}$ will run $\mathcal{A}$ as a subroutine and act as $\mathcal{A}$'s challenger in the IND-CCA2-I′ game for CLSC-TKEM. During the game, $\mathcal{A}$ will consult $\mathcal{C}$ for answers to the random oracles H_1, H_2, H_3 and H_4. Roughly speaking, these answers are randomly generated, but to maintain the consistency and to avoid collision, $\mathcal{C}$ keeps three lists L_1, L_2, L_3, L_4 respectively to store the answers. The following assumptions are made.

1. $\mathcal{A}$ will ask for $H_1(ID)$ before ID is used in any partial private key extraction, private key extraction, symmetric key generation, key encapsulation and key decapsulation queries.
2. Key encapsulation returned from a key encapsulation query will not be used by $\mathcal{A}$ in a key decapsulation query.

At the beginning of the game, $\mathcal{C}$ gives $\mathcal{A}$ the system parameters with $P_{pub} \leftarrow aP$. Note that a is unknown to $\mathcal{C}$. This value simulates the master secret key for the KGC in the game. $\mathcal{C}$ chooses a random number $j \in \{1, 2, \ldots, q_T\}$ and answers various oracle queries as follows.

H_1 queries: $\mathcal{A}$ asks a polynomially bounded number of H_1 queries on identities of his choice. At the j-th H_1 query, $\mathcal{C}$ answers by $H_1(ID_j) \leftarrow bP$ and puts $(ID_j, \perp)$ to list L_1. For queries $H_1(ID_i)$ with $i \neq j$, $\mathcal{C}$ chooses $e_i \in Z_q^*$ randomly, puts (ID_i, e_i) in list L_1 and answers $H_1(ID_i) \leftarrow e_iP$.

Extract partial private key: When $\mathcal{A}$ asks a partial private key extraction query on identity ID_i, if $ID_i = ID_j$, then $\mathcal{C}$ fails and stops. If $ID_i \neq ID_j$, then the list L_1 must contain (ID_i, e_i) for some e_i (this indicates $\mathcal{C}$ previously answered $H_1(ID_i) \leftarrow e_iP$ on a H_1 query on ID_i). $\mathcal{C}$ returns the partial private key $D_{ID_i} \leftarrow e_iaP$.

Request public key: When $\mathcal{A}$ asks a public key query on identity ID_i, $\mathcal{C}$ checks the list L_K, which is initially empty. If there is a tuple $(ID_i, PK_{ID_i}, x_{ID_i})$, then $\mathcal{C}$ returns PK_{ID_i}. Otherwise, $\mathcal{C}$ generates a new key pair, updates the list L_K, and returns the public key.

Replace public key: On input (ID_i, PK_{ID_i}), $\mathcal{C}$ inserts/updates L_K with tuple $(ID_i, PK_{ID_i}, \perp)$.

Extract private key: When $\mathcal{A}$ asks a private key extraction query on identity ID_i, $\mathcal{C}$ calls H_1 on ID_i and obtains (ID_i, e_i). If $ID_i = ID_j$, then $\mathcal{C}$ fails and stops. Otherwise, $\mathcal{C}$ searches L_K for the entry $(ID_i, PK_{ID_i}, x_{ID_i})$, generating a new key pair if this does not exist, and returns $S_{ID_i} \leftarrow (x_{ID_i}, e_iaP)$.

H_3 Queries: When $\mathcal{A}$ asks a H_3 query on $(U_i, \tau_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ checks if the list L_3 contains a tuple $(U_i, \tau_i, ID_i, PK_{ID_i}, t_i, t_iP)$. If such a tuple is found, $\mathcal{C}$ answers t_iP. Otherwise, $\mathcal{C}$ chooses a random value $t \in Z_q$, puts the $(U_i, \tau_i, ID_i, PK_{ID_i}, t, tP)$ into L_3, and returns tP.

H_4 Queries: When $\mathcal{A}$ asks a H_4 query on $(U_i, \tau_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ checks if the list L_4 contains a tuple $(U_i, \tau_i, ID_i, PK_{ID_i}, l_i, l_iP)$. If such a tuple is found, $\mathcal{C}$ answers l_iP. Otherwise, $\mathcal{C}$ chooses a random value $l \in Z_q$, puts the $(U_i, \tau_i, ID_i, PK_{ID_i}, l, lP)$ into L_4, and returns lP.

H_2 queries: For each new query $(U_i, T_i, R_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ proceeds as follows:

1. It checks if the decision bilinear Diffie-Hellman oracle returns 1 when queried with the tuple (aP, bP, cP, T_i). If this is the case, $\mathcal{C}$ returns T_i and stop.
2. $\mathcal{C}$ goes through the list L_2 with entries $(U_i, \star, R_i, ID_i, PK_{ID_i}, h_i)$, for different values of h_i, such that the decision bilinear Diffie-Hellman oracle returns 1 when queried on the tuple (aP, bP, U_i, T_i). Note that in this case $ID_i = ID_j$. If such a tuple exists, it returns h_i (and replaces the symbol $\star$ with T_i)

3. If $\mathcal{C}$ reaches this point of execution, it returns a random h and updates the list L_2, which is initially empty, with a tuple containing the input and return values.

Symmetric key generation queries: Let ID_s, ID_r be the identity of the sender and that of the receiver respectively used by $\mathcal{A}$ in a symmetric key generation query. For each new query (ID_s, ID_r), $\mathcal{C}$ proceeds as follows:

1. If $ID_s \neq ID_j$, $\mathcal{C}$ computes the private key S_{ID_s} corresponding to ID_s by running the private key extraction query algorithm. Then $\mathcal{C}$ runs $(K, \omega) \leftarrow \texttt{Sym}(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ and sends K to $\mathcal{A}$. Note that $\mathcal{C}$ needs to store ω and to overwrite any previous value.
2. If $ID_s = ID_j$ (and hence $ID_r \neq ID_j$), $\mathcal{C}$ chooses $u, v \in Z_q^*$, sets $U \leftarrow vaP$, and computes $T \leftarrow \hat{e}(U, D_{ID_r})$($\mathcal{C}$ could obtain D_{ID_r} from a partial private key extraction query because $ID_r \neq ID_j$). Note that the ω is $(u, v, U, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ in this case.
3. It goes through list L_2 looking for an entry $(U, T, R, ID_r, PK_{ID_r}, h)$ for some R such that $\hat{e}(U, PK_{ID_r}) = \hat{e}(P, R)$, where PK_{ID_r} is obtained by calling the request public key oracle on ID_r. If such an entry exists, it computes $K \leftarrow h$. Otherwise it uses a random h and updates the list L_2 with $(U, T, \star, ID_r, PK_{ID_r}, h)$.

Key encapsulation queries: $\mathcal{A}$ produces a arbitrary tag τ. $\mathcal{C}$ checks whether there exists a stored value ω. If there is not, it returns $\perp$ and terminates. Otherwise $\mathcal{C}$ proceeds as follows.

1. If $ID_s \neq ID_j$, $\mathcal{C}$ answers the query by a call to $\texttt{Encap}(\omega, \tau)$.
2. If $ID_s = ID_j$ (and $ID_r \neq ID_j$), $\mathcal{C}$ defines the hash value $H_3(U, \tau, ID_s, PK_{ID_s})$ as $H \leftarrow v^{-1}(uP - Q_{ID_s})$. If a such a hash queries has been responded with a different value before, it aborts the simulation. This means that $\mathcal{C}$ updates list L_3 with tuple $(U, \tau, ID_s, PK_{ID_s}, \perp, H)$. Finally, $\mathcal{C}$ sets $W = uaP + lPK_{ID_s}$, where l is the value obtained by querying H_4 on $(U, \tau, ID_s, PK_{ID_s})$. $\mathcal{C}$ returns $\psi \leftarrow (U, W)$.

Key decapsulation queries: For a key decapsulation query on a (ψ', τ') for identities ID_s and ID_r, $\mathcal{C}$ proceeds as follows.

1. It executes the verification part of the decapsulation algorithm by obtaining Q_{ID_s} and PK_{ID_s} by calling H_1 and request public key oracles. It returns $\perp$ if the verification does not succeed.
2. It computes $R \leftarrow x_{ID_r}U$, obtaining x_{ID_r} (and hence PK_{ID_r}) from either the adversary or by calling the request public key oracle.
3. If $ID_r \neq ID_j$, $\mathcal{C}$ computes the partial private key D_{ID_r} corresponding to ID_r by running the partial private key extraction query algorithm. Then $\mathcal{C}$ computes $T \leftarrow \hat{e}(D_{ID_r}, U)$, and returns $K \leftarrow H_2(U, T, R, ID_r, PK_{ID_r})$.
4. If $ID_r = ID_j$, then the pairing cannot be computed. In order to return a consistent answer, $\mathcal{C}$ goes through L_2 and looks for a tuple $(U, T, R, ID_r, PK_{ID_r}, h)$, for different values of T, such that the decision bilinear Diffie-Hellman oracle returns 1 when queried on (aP, bP, U, T). If such an entry exists, the correct pairing value is found and returns $K \leftarrow h$.
5. If $\mathcal{C}$ reaches this point of execution, it places the entry $(U, \star, R, ID_r, PK_{ID_r}, h)$ for a random h on list L_2 and returns $K \leftarrow h$. The symbol $\star$ denotes an unknown value of pairing. Note that the identity component of all entries with a $\star$ is ID_j.

After the first stage, $\mathcal{A}$ picks two identities ID_s^* and ID_r^* on which it wishes to be challenged. If $ID_r^* \neq ID_j$, $\mathcal{C}$ fails and stops. Otherwise it proceeds to construct a challenge as follows. It obtains the public key $PK_{ID_s^*}$ corresponding to ID_s^* form the list L_K. Then it sets $U^* = cP$, chooses a random hash value h^* and sets $K_1 \leftarrow h^*$. $\mathcal{C}$ chooses $K_0 \leftarrow \mathcal{K}_{\mathrm{CLSC-TKEM}}$ and a bit $b \in \{0,1\}$ randomly, and sends K_b to $\mathcal{A}$. $\mathcal{A}$ then sends a tag τ^* to $\mathcal{C}$. $\mathcal{C}$ computes $W^* = D_{ID_s^*} + rH + x_{ID_s^*}H' = D_{ID_s^*} + tcP + lPK_{ID_s^*}$, where t is obtained from L_3, l is obtained from L_4 and $D_{ID_s^*}$ is computed by calling the partial private key extraction oracle on ID_s^*. Note that, since $ID_s^* \neq ID_r^*$ the partial private key extraction oracle simulation always give $\mathcal{C}$ the correct value of D_{ID_s}. $\mathcal{C}$ sends the challenge encapsulation $\psi^* \leftarrow (U^*, W^*)$ to $\mathcal{A}$.

$\mathcal{A}$ then performs a second series of queries which is treated in the same way as the first one. At the end of the simulation, it produces a bit b' for which it believes the relation $(K_b, \omega^*) \leftarrow \mathtt{Sym}(params, S_{ID_s^*}, ID_s^*, PK_{ID_s^*}, ID_r^*, PK_{ID_r^*})$ and $\psi^* \leftarrow \mathtt{Encap}(\omega^*, \tau^*)$ hold.

Since ID_j is independent of adversary's view, and the list L_1 can be easily seen to have at most q_T elements, with probability $1/q_T$ the adversary will output an identity ID_j. If this event occurs, the simulation is perfect unless the adversary queries H_2 on the challenge-related tuple $(U^*, T^*, R^*, ID_r^*, PK_{ID_r}^*)$. Since the hash function H_2 is modeled as a random oracle, the adversary will not have any advantage if this tuple does not appear on L_2. However, if this happens, $\mathcal{C}$ will win the game due to the first step in the simulation of H_2. The Lemma follows from this observation and the fact that the total number of decision bilinear Diffie-Hellman oracle calls that $\mathcal{C}$ makes is at most $q_D^2 + 2q_D q_2 + q_2$. □

Lemma 6. *Under the CDH assumption in G_1, no PPT attacker $\mathcal{A}$ has non-negligible advantage in winning the IND-CCA-II game against the above CLSC-TKEM, when all hash functions are modeled as random oracles. More precisely, there exists an algorithm $\mathcal{C}$ which uses $\mathcal{A}$ to solve the CDH problem such that:*

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{IND-CCA2-II}}(\mathcal{A}) \leq q_T \mathrm{Adv}_{CDH}(\mathcal{C}),$$

where $q_T = q_{RK} + q_{PK} + q_K + 2q_D + 2$. Here q_{RK} and q_{PK} are the maximum number of queries that the adversary can request public key and replace public key, respectively. q_K and q_D are as before.

Proof. The challenger $\mathcal{C}$ takes as input (P, aP, bP) and attempts to compute abP. $\mathcal{C}$ will run $\mathcal{A}$ as a subroutine and act as $\mathcal{A}$'s challenger in the IND-CCA2-II game for CLSC-TKEM. During the game, $\mathcal{A}$ will consult $\mathcal{C}$ for answers to the random oracles H_1, H_2, H_3 and H_4. Roughly speaking, these answers are randomly generated, but to maintain the consistency and to avoid collision, $\mathcal{C}$ keeps three lists L_1, L_2, L_3, L_4 respectively to store the answers. The following assumptions are made.

1. $\mathcal{A}$ will ask for $H_1(ID)$ before ID is used in any private key extraction, symmetric key generation, key encapsulation and key decapsulation queries.
2. Key encapsulation returned from a key encapsulation query will not be used by $\mathcal{A}$ in a key decapsulation query.

At the beginning of the game, $\mathcal{C}$ generates a master secret key s and system parameters $params$ including a master public key $P_{pub} \leftarrow sP$. Then $\mathcal{C}$ gives both $params$ and s to $\mathcal{A}$. $\mathcal{C}$ first chooses a random number $j \in \{1, 2, \ldots, q_T\}$, and answers various oracle queries as follows.

H_1 queries: For a query on $H_1(ID_i)$, $\mathcal{C}$ chooses $e_i \in Z_q^*$ randomly, puts (ID_i, e_i) in list L_1 and answers $H_1(ID_i) \leftarrow e_iP$.

Request public key: When $\mathcal{A}$ asks a public key query on identity ID_i, if $ID_i \neq ID_j$, $\mathcal{C}$ generates a new key pair (x_{ID_i}, PK_{ID_i}), updates the list L_K with $(ID_i, PK_{ID_i}, x_{ID_i})$, and returns the public key. If $ID_i = ID_j$, $\mathcal{C}$ returns aP and adds $(ID_j, aP, \perp)$ to L_K.

Extract private key: When $\mathcal{A}$ asks a private key extraction query on identity ID_i, $\mathcal{C}$ calls request public key on ID_i and obtains $(ID_i, PK_{ID_i}, x_{ID_i})$. If $ID_i = ID_j$, then $\mathcal{C}$ fails and stops. Otherwise, $\mathcal{C}$ calls H_1 on ID_i and gets (ID_i, e_i). It returns (x_{ID_i}, se_iP).

H_3 Queries: When $\mathcal{A}$ asks a H_3 query on $(U_i, \tau_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ checks if the list L_3 contains a tuple $(U_i, \tau_i, ID_i, PK_{ID_i}, t_i, t_iP)$. If such a tuple is found, $\mathcal{C}$ answers t_iP. Otherwise, $\mathcal{C}$ chooses a random value $t \in Z_q$, puts the $(U_i, \tau_i, ID_i, PK_{ID_i}, t, tP)$ into L_3, and returns tP.

H_4 Queries: When $\mathcal{A}$ asks a H_4 query on $(U_i, \tau_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ checks if the list L_4 contains a tuple $(U_i, \tau_i, ID_i, PK_{ID_i}, l_i, l_iP)$. If such a tuple is found, $\mathcal{C}$ answers l_iP. Otherwise, $\mathcal{C}$ chooses a random value $l \in Z_q$, puts the $(U_i, \tau_i, ID_i, PK_{ID_i}, l, lP)$ into L_4, and returns lP.

H_2 queries: For each new query $(U_i, T_i, R_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ proceeds as follows:

1. It checks if $\hat{e}(aP, bP) = \hat{e}(P, R_i)$. If so, $\mathcal{C}$ returns R_i and stops.
2. $\mathcal{C}$ goes through the list L_2 looking for entries $(U_i, T_i, \star, ID_i, PK_{ID_i}, h_i)$ such that $\hat{e}(U_i, aP) = \hat{e}(P, R_i)$. Note that in this case $ID_i = ID_j$. If such a tuple exists, it returns h_i (and replaces the symbol $\star$ with R_i).
3. If $\mathcal{C}$ reaches this point of execution, it returns a random h and updates the list L_2, which is initially empty, with a tuple containing the input and return values.

Symmetric key generation queries: Let ID_s, ID_r be the identity of the sender and that of the receiver respectively used by $\mathcal{A}$ in a symmetric key generation query. For each new query (ID_s, ID_r), $\mathcal{C}$ proceeds as follows:

1. If $ID_s \neq ID_j$, $\mathcal{C}$ computes the private key S_{ID_s} corresponding to ID_s by running the private key extraction query algorithm. Then $\mathcal{C}$ runs $(K, \omega) \leftarrow \mathtt{Sym}(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ and sends K to $\mathcal{A}$. Note that $\mathcal{C}$ needs to store ω and to overwrite any previous value.
2. If $ID_s = ID_j$ (and hence $ID_r \neq ID_j$), $\mathcal{C}$ chooses $u, v \in Z_q^*$, sets $U \leftarrow vaP$, and computes $T \leftarrow \hat{e}(U, D_{ID_r})$($\mathcal{C}$ could computes D_{ID_r} because it knows the master secret key s). Note that the ω is $(u, v, U, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ in this case.
3. It goes through list L_2 looking for an entry $(U, T, R, ID_r, PK_{ID_r}, h)$ for some R such that $\hat{e}(U, PK_{ID_r}) = \hat{e}(P, R)$, where PK_{ID_r} is obtained by calling the request public key oracle on ID_r. If such an entry exists, it computes $K \leftarrow h$. Otherwise it uses a random h and updates the list L_2 with $(U, T, \star, ID_r, PK_{ID_r}, h)$.

Key encapsulation queries: $\mathcal{A}$ produces a arbitrary tag τ. $\mathcal{C}$ checks whether there exists a stored value ω. If there is not, it returns $\perp$ and terminates. Otherwise $\mathcal{C}$ proceeds as follows.

1. If $ID_s \neq ID_j$, $\mathcal{C}$ answers the query by a call to $\mathtt{Encap}(\omega, \tau)$.
2. If $ID_s = ID_j$ (and $ID_r \neq ID_j$), $\mathcal{C}$ defines the hash value $H_3(U, \tau, ID_s, PK_{ID_s})$ as $H \leftarrow v^{-1}(uP - H_4)$, where H_4 is the output of $H_4(U, \tau, ID_s, PK_{ID_s})$. If a such a hash queries has been responded with a different value before, it aborts the simulation. This means that $\mathcal{C}$ updates list L_3 with tuple $(U, \tau, ID_s, PK_{ID_s}, \perp, H)$. Finally, $\mathcal{C}$ sets $W = D_{ID_s} + uaP$. $\mathcal{C}$ returns $\psi \leftarrow (U, W)$.

Key decapsulation queries: For a key decapsulation query on a (ψ', τ') for identities ID_s and ID_r, $\mathcal{C}$ proceeds as follows.

1. It executes the verification part of the decapsulation algorithm obtaining Q_{ID_s} and PK_{ID_s} by calling H_1 and request public key oracles. It returns $\perp$ if the verification does not succeed.
2. It calculates $T = \hat{e}(U, e_r P_{pub})$, where (ID_r, e_r) is obtained from H_1.
3. If $ID_r \neq ID_j$, it computes $R \leftarrow x_{ID_r} U$, where x_{ID_r} is obtained (and hence PK_{ID_r}) from either the adversary or by calling the request public key oracle. Then $\mathcal{C}$ returns $K \leftarrow H_2(U, T, R, ID_r, PK_{ID_r})$.
4. If $ID_r = ID_j$, the correct value of R cannot be computed. To return a consistent answer, $\mathcal{C}$ goes through L_2 and looks for a tuple $(U, T, R, ID_r, PK_{ID_r}, h)$, for different values of R, such that $\hat{e}(U, aP) = \hat{e}(P, R)$. If such an entry exists, the correct value of R is found and returns $K \leftarrow h$.
5. If $\mathcal{C}$ reaches this point of execution, it places the entry $(U, T, \star, ID_r, PK_{ID_r}, h)$ for a random h on list L_2 and returns $K \leftarrow h$. The symbol $\star$ denotes an unknown value of R.

After the first stage, $\mathcal{A}$ picks two identities ID_s^* and ID_r^* on which it wishes to be challenged. If $ID_r^* \neq ID_j$, $\mathcal{C}$ fails and stops. Otherwise it proceeds to construct a challenge as follows. It obtains the public key $PK_{ID_s^*}$ corresponding to ID_s^* form the list L_K. Then it sets $U^* = bP$, chooses a random hash value h^* and sets $K_1 \leftarrow h^*$. $\mathcal{C}$ chooses $K_0 \leftarrow \mathcal{K}_{\mathrm{CLSC-TKEM}}$ and a bit $b \in \{0, 1\}$ randomly, and sends K_b to $\mathcal{A}$. $\mathcal{A}$ then sends a tag τ^* to $\mathcal{C}$. $\mathcal{C}$ computes $W^* = D_{ID_s^*} + rH + x_{ID_s^*} H' = D_{ID_s^*} + tcP + lPK_{ID_s^*}$, where t is obtained from L_3, l is obtained from L_4 and $D_{ID_s^*}$ is computed by calling the partial private key extraction oracle on ID_s^*. $\mathcal{C}$ sends the challenge encapsulation $\psi^* \leftarrow (U^*, W^*)$ to $\mathcal{A}$.

$\mathcal{A}$ then performs a second series of queries which is treated in the same way as the first one. At the end of the simulation, it produces a bit b' for which it believes the relation $(K_b, \omega^*) \leftarrow \mathtt{Sym}(params, S_{ID_s^*}, ID_s^*, PK_{ID_s^*}, ID_r^*, PK_{ID_r^*})$ and $\psi^* \leftarrow \mathtt{Encap}(\omega^*, \tau^*)$ hold.

Since ID_j is independent of adversary's view, and the list L_1 can be easily seen to have at most q_T elements, with probability $1/q_T$ the adversary will output an identity ID_j. If this event occurs, the simulation is perfect unless the adversary queries H_2 on the challenge-related tuple $(U^*, T^*, R^*, ID_r^*, PK_{ID_r}^*)$. Since the hash function H_2 is modeled as a random oracle, the adversary will not have any advantage if this tuple does not appear on L_2. However, if this happens, $\mathcal{C}$ will win the game due to its simulation of H_2. The Lemma follows from this observation and the fact that the maximum length of the list L_K is q_T, as stated in the Lemma. □

E Proof of Theorem 4

Proof. In the Barbosa-Farshim CLSC scheme [6], they use a weaker formulation of Type I adversary which they refer to as Type I′. In unforgeability games, the Type I′ adversary is not allowed to extract the partial private key of ID_s^*. They proved that If a CLSC scheme is sUF-CMA secure against Type II and Type I′ attackers, then it is also sUF-CMA secure against Type I attackers. It is easy to extend these conclusions to CLSC-TKEM setting. That is, we have the following Lemma 7.

Lemma 7. *If a CLSC-TKEM is sUF-CMA secure against Type II and Type I′ attackers then it is also sUF-CMA secure against Type I attackers. In particular, we have*

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-I}}(\mathcal{F}) \leq 2\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-I'}}(\mathcal{C}_1) + \mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-II}}(\mathcal{C}_2).$$

This theorem follows from Lemmas 7, 8 and 9.

$\square$

Lemma 8. *Under the GDH′ assumption in G_1, no PPT attacker $\mathcal{F}$ has non-negligible advantage in winning the sUF-CMA-I′ game against the above CLSC-TKEM, when all hash functions are modeled as random oracles. More precisely, there exists an algorithm $\mathcal{C}$ which uses $\mathcal{F}$ to solve the GDH′ problem such that:*

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-I}'}(\mathcal{F}) \leq q_T \mathrm{Adv}_{GDH'}(\mathcal{C}, q_D^2 + 2q_D q_2) + (q_{SK}(q_{SK} + q_D + q_3 + 1) + 2)/2^k,$$

where $q_T = q_1 + q_P + q_K + 2q_D + 2q_{SK} + 1$. Here q_3 and q_{SK} are the maximum number of queries that the adversary could ask H_3 and make symmetric key generation queries, respectively. q_1, q_P, q_K, and q_D are as before.

Proof. The challenger $\mathcal{C}$ takes as input (P, aP, bP) and attempts to compute abP. $\mathcal{C}$ will run $\mathcal{A}$ as a subroutine and act as $\mathcal{A}$'s challenger in the sUF-CMA-I′ game for CLSC-TKEM. During the game, $\mathcal{A}$ will consult $\mathcal{C}$ for answers to the random oracles H_1, H_2, H_3 and H_4. Roughly speaking, these answers are randomly generated, but to maintain the consistency and to avoid collision, $\mathcal{C}$ keeps three lists L_1, L_2, L_3, L_4 respectively to store the answers. The following assumptions are made.

1. $\mathcal{F}$ will ask for $H_1(ID)$ before ID is used in any partial private key extraction, private key extraction, symmetric key generation, key encapsulation and key decapsulation queries.
2. Key encapsulation returned from a key encapsulation query will not be used by $\mathcal{F}$ in a key decapsulation query.

At the beginning of the game, $\mathcal{C}$ gives $\mathcal{F}$ the system parameters with $P_{pub} \leftarrow aP$. Note that a is unknown to $\mathcal{C}$. This value simulates the master key value for the KGC in the game. $\mathcal{C}$ chooses a random number $j \in \{1, 2, \ldots, q_T\}$ and answers various oracle queries as follows.

H_1 queries: Same to Lemma 5.

Extract partial private key: Same to Lemma 5.

Request public key: Same to Lemma 5.

Replace public key: Same to Lemma 5.

Extract private key: Same to Lemma 5.

H_3 Queries: Same to Lemma 5.

H_4 Queries: Same to Lemma 5.

H_2 queries: For each new query $(U_i, T_i, R_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ proceeds as follows:

1. It checks if $\hat{e}(aP, bP) = \hat{e}(P, R_i)$. If this is the case, $\mathcal{C}$ returns R_i and stop.
2. $\mathcal{C}$ goes through the list L_2 with entries $(U_i, \star, R_i, ID_i, PK_{ID_i}, h_i)$, for different values of h_i, such that the decision bilinear Diffie-Hellman oracle returns 1 when queried on the tuple (aP, bP, U_i, T_i). Note that in this case $ID_i = ID_j$. If such a tuple exists, it returns h_i (and replaces the symbol $\star$ with T_i)
3. It goes through the list L_2 with entries $(U_i, T_i, \star, ID_i, PK_{ID_i}, h_i)$, for different values of h_i, such that $\hat{e}(U_i, PK_{ID_i}) = \hat{e}(P, R_i)$. If such a tuple exists, it returns h_i (and replaces the symbol $\star$ with R_i).

4. If $\mathcal{C}$ reaches this point of execution, it returns a random h and updates the list L_2, which is initially empty, with a tuple containing the input and return values.

Symmetric key generation queries: Let ID_s, ID_r be the identity of the sender and that of the receiver respectively used by $\mathcal{F}$ in a symmetric key generation query. For each new query (ID_s, ID_r), $\mathcal{C}$ proceeds as follows:

1. If $ID_s \neq ID_j$, $\mathcal{C}$ computes the private key S_{ID_s} corresponding to ID_s by running the private key extraction query algorithm. Then $\mathcal{C}$ runs $(K, \omega) \leftarrow \mathtt{Sym}(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ and sends K to $\mathcal{F}$. Note that $\mathcal{C}$ needs to store ω and to overwrite any previous value.
2. If $ID_s = ID_j$ (and hence $ID_r \neq ID_j$), $\mathcal{C}$ chooses $u, v \in Z_q^*$, sets $U \leftarrow vaP$, and computes $T \leftarrow \hat{e}(U, D_{ID_r})$($\mathcal{C}$ could obtain D_{ID_r} from a partial private key extraction query because $ID_r \neq ID_j$). Note that the ω is $(u, v, U, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ in this case.
3. It goes through list L_2 looking for an entry $(U, T, R, ID_r, PK_{ID_r}, h)$ for some R such that $\hat{e}(U, PK_{ID_r}) = \hat{e}(P, R)$, where PK_{ID_r} is obtained by calling the request public key oracle on ID_r. If such an entry exists, it computes $K \leftarrow h$. Otherwise it uses a random h and updates the list L_2 with $(U, T, \star, ID_r, PK_{ID_r}, h)$.

Key encapsulation queries: $\mathcal{F}$ produces a arbitrary tag τ. $\mathcal{C}$ checks whether there exists a stored value ω. If there is not, it returns $\perp$ and terminates. Otherwise $\mathcal{C}$ proceeds as follows.

1. If $ID_s \neq ID_j$, $\mathcal{C}$ answers the query by a call to $\mathtt{Encap}(\omega, \tau)$.
2. If $ID_s = ID_j$ (and $ID_r \neq ID_j$), $\mathcal{C}$ defines the hash value $H_3(U, \tau, ID_s, PK_{ID_s})$ as $H \leftarrow v^{-1}(uP - Q_{ID_s})$. If a such a hash queries has been responded with a different value before, it aborts the simulation. This means that $\mathcal{C}$ updates list L_3 with tuple $(U, \tau, ID_s, PK_{ID_s}, \perp, H)$. Finally, $\mathcal{C}$ sets $W = uaP + lPK_{ID_s}$, where l is the value obtained by querying H_4 on $(U, \tau, ID_s, PK_{ID_s})$. $\mathcal{C}$ returns $\psi \leftarrow (U, W)$.

Key decapsulation queries: For a key decapsulation query on a (ψ', τ') for identities ID_s and ID_r, $\mathcal{C}$ proceeds as follows.

1. It executes the verification part of the decapsulation algorithm obtaining Q_{ID_s} and PK_{ID_s} by calling H_1 and request public key oracles. It returns $\perp$ if the verification does not succeed.
2. It checks if $ID_i = ID_j$ and if this is the case then $\mathcal{C}$ can solve the GDH$'$ problem as described below.
3. It computes $R \leftarrow x_{ID_r}U$, obtaining x_{ID_r} (and hence PK_{ID_r}) from either the adversary or by calling the request public key oracle.
4. If $ID_r \neq ID_j$, $\mathcal{C}$ computes the private key D_{ID_r} corresponding to ID_r by running the partial private key extraction query algorithm. Then $\mathcal{C}$ computes $T \leftarrow \hat{e}(D_{ID_r}, U)$, and returns $K \leftarrow H_2(U, T, R, ID_r, PK_{ID_r})$.
5. If $ID_r = ID_j$, then the pairing cannot be computed. In order to return a consistent answer, $\mathcal{C}$ goes through L_2 and looks for a tuple $(U, T, R, ID_r, PK_{ID_r}, h)$, for different values of T, such that the decision bilinear Diffie-Hellman oracle returns 1 when queried on (aP, bP, U, T). If such an entry exists, the correct pairing value is found and returns $K \leftarrow h$.
6. If $\mathcal{C}$ reaches this point of execution, it places the entry $(U, \star, R, ID_r, PK_{ID_r}, h)$ for a random h on list L_2 and returns $K \leftarrow h$. The symbol $\star$ denotes an unknown value of pairing. Note that the identity component of all entries with a $\star$ is ID_j.

Finally, $\mathcal{F}$ outputs a produces a quaternion $(\tau^*, \psi^*, ID_s^*, ID_r^*)$. $\mathcal{C}$ checks if $ID_s^* = ID_j$. If not, it aborts execution. Otherwise, it obtains $PK_{ID_s^*}$ by calling the request public key oracle on ID_s^* and retrieves t^* and l^* from lists L_3 and L_4 by querying H_3 and H_4 on $(U^*, \tau^*, ID_s^*, PK_{ID_s^*})$. Note that if $\mathcal{C}$ succeeded, then the verification condition holds:

$$\begin{aligned}\hat{e}(P, W^*) &= \hat{e}(P_{pub}, Q_{ID_s^*})\hat{e}(U^*, H^*)\hat{e}(PK_{ID_s^*}, H'^*)\\ \hat{e}(P, W^*) &= \hat{e}(aP, bP)\hat{e}(U^*, t^*P)\hat{e}(PK_{ID_s^*}, l^*P)\\ \hat{e}(P, abP) &= \hat{e}(P, W^* - t^*U - l^*PK_{ID_s^*})\end{aligned}$$

and thus $\mathcal{C}$ can compute

$$abP = W^* - t^*U - l^*PK_{ID_s^*}$$

Let us now analyze the probability that $\mathcal{C}$ succeeds in solving the GDH$'$ problem instance. For this to happen, the simulation must run until the end of the game, the adversary must pick a specific identity as ID_j^*, and it must query the hash functions H_3 and H_4 to properly construct the forgery. The probability that $\mathcal{F}$ is able to produce a forgery without querying both hash functions is upper bounded by $2/2^k$.

The probability that $\mathcal{C}$ aborts the simulation is related with the following events:

- $\mathcal{F}$ places a partial key extraction on ID_j.
- $\mathcal{F}$ places a full private key extraction on ID_j.
- $\mathcal{C}$ wants to simulate a key encapsulation query and this leads to an inconsistency in the H_3 simulation.

Note that if $\mathcal{F}$ places either of the first two fatal queries, then it could not possibly use ID_j as the sender identity in the forgery it produces at the end of the game, so we can pinpoint the probability that $\mathcal{C}$ does not abort the simulation due to these events and $\mathcal{F}$ picks the only useful case for solving GDH$'$ as $1/q_T$. Note that the maximum length of the list L_1 is q_T, as stated in the Lemma

The latter fatal event occurs if $\mathcal{C}$'s simulation triggers a collision in its simulation of H_3. Since the maximum size of L_3 is $q_{SK} + q_D + q_3 + 1$, we can upper bound the probability that this occurs as $q_{SK}(q_{SK} + q_D + q_3 + 1)/2^k$. The result follows by noting that $\mathcal{C}$ makes at most $q_D^2 + 2q_Dq_2$ queries to its decision bilinear Diffie-Hellman oracle. □

Lemma 9. *Under the CDH assumption in G_1, no PPT attacker $\mathcal{F}$ has non-negligible advantage in winning the sUF-CMA-II game against the above CLSC-TKEM, when all hash functions are modeled as random oracles. More precisely, there exists an algorithm $\mathcal{C}$ which uses $\mathcal{F}$ to solve the CDH problem such that:*

$$\mathrm{Adv}_{\mathrm{CLSC}}^{\mathrm{sUF-CMA-II}}(\mathcal{F}) \leq q_T \mathrm{Adv}_{CDH}(\mathcal{C}) + (q_{SK}(q_{SK} + q_D + q_3 + 1) + 2)/2^k,$$

where $q_T = q_{RK} + q_{PK} + q_K + 2q_D + 2q_{SK} + 1$ and various q's are as before.

Proof. The challenger $\mathcal{C}$ takes as input (P, aP, bP) and attempts to compute abP. $\mathcal{C}$ will run $\mathcal{F}$ as a subroutine and act as $\mathcal{F}$'s challenger in the sUF-CMA-II game for CLSC-TKEM. During the game, $\mathcal{F}$ will consult $\mathcal{C}$ for answers to the random oracles H_1, H_2, H_3 and H_4. Roughly speaking, these answers are randomly generated, but to maintain the consistency and to avoid collision, $\mathcal{C}$ keeps three lists L_1, L_2, L_3, L_4 respectively to store the answers. The following assumptions are made.

1. $\mathcal{F}$ will ask for $H_1(ID)$ before ID is used in any private key extraction, symmetric key generation, key encapsulation and key decapsulation queries.
2. Key encapsulation returned from a key encapsulation query will not be used by $\mathcal{F}$ in a key decapsulation query.

At the beginning of the game, $\mathcal{C}$ generates a master secret key s and system parameters $params$ including a master public key P_{pub}. Then $\mathcal{C}$ gives both $params$ and s to $\mathcal{F}$. $\mathcal{C}$ first chooses a random number $j \in \{1, 2, \ldots, q_T\}$, and answers various oracle queries as follows.

H_1 queries: Same to Lemma 6.

Request public key: Same to Lemma 5.

Extract private key: Same to Lemma 5.

H_3 Queries: Same to Lemma 6.

H_4 Queries: When $\mathcal{F}$ asks a H_4 query on $(U_i, V_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ checks if the list L_4 contains a tuple $(U_i, V_i, ID_i, PK_{ID_i}, l_i, l_ibP)$. If such a tuple is found, $\mathcal{C}$ answers l_ibP. Otherwise, $\mathcal{C}$ chooses a random value $l \in Z_q$, puts the $(U_i, V_i, ID_i, PK_{ID_i}, l, lbP)$ into L_4, and returns lbP.

H_2 queries: For each new query $(U_i, T_i, R_i, ID_i, PK_{ID_i})$, $\mathcal{C}$ proceeds as follows:

1. It checks if $\hat{e}(aP, bP) = \hat{e}(P, R_i)$. If so, $\mathcal{C}$ returns R_i and stops.
2. $\mathcal{C}$ goes through the list L_2 looking for entries $(U_i, T_i, \star, ID_i, PK_{ID_i}, h_i)$ such that $\hat{e}(U_i, PK_{ID_i}) = \hat{e}(P, R_i)$. If such a tuple exists, it returns h_i (and replaces the symbol $\star$ with R_i).
3. If $\mathcal{C}$ reaches this point of execution, it returns a random h and updates the list L_2, which is initially empty, with a tuple containing the input and return values.

Symmetric key generation queries: Let ID_s, ID_r be the identity of the sender and that of the receiver respectively used by $\mathcal{F}$ in a symmetric key generation query. For each new query (ID_s, ID_r), $\mathcal{C}$ proceeds as follows:

1. If $ID_s \neq ID_j$, $\mathcal{C}$ computes the private key S_{ID_s} corresponding to ID_s by running the private key extraction query algorithm. Then $\mathcal{C}$ runs $(K, \omega) \leftarrow \texttt{Sym}(params, S_{ID_s}, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ and sends K to $\mathcal{F}$. Note that $\mathcal{C}$ needs to store ω and to overwrite any previous value.
2. If $ID_s = ID_j$ (and hence $ID_r \neq ID_j$), $\mathcal{C}$ chooses $u, v \in Z_q^*$, sets $U \leftarrow vaP$, and computes $T \leftarrow \hat{e}(U, D_{ID_r})$($\mathcal{C}$ could computes D_{ID_r} because it knows the master secret key s). Note that the ω is $(u, v, U, ID_s, PK_{ID_s}, ID_r, PK_{ID_r})$ in this case.
3. It goes through list L_2 looking for an entry $(U, T, R, ID_r, PK_{ID_r}, h)$ for some R such that $\hat{e}(U, PK_{ID_r}) = \hat{e}(P, R)$, where PK_{ID_r} is obtained by calling the request public key oracle on ID_r. If such an entry exists, it computes $K \leftarrow h$. Otherwise it uses a random h and updates the list L_2 with $(U, T, \star, ID_r, PK_{ID_r}, h)$.

Key encapsulation queries: $\mathcal{F}$ produces a arbitrary tag τ. $\mathcal{C}$ checks whether there exists a stored value ω. If there is not, it returns $\perp$ and terminates. Otherwise $\mathcal{C}$ proceeds as follows.

1. If $ID_s \neq ID_j$, $\mathcal{C}$ answers the query by a call to $\texttt{Encap}(\omega, \tau)$.
2. If $ID_s = ID_j$ (and $ID_r \neq ID_j$), $\mathcal{C}$ defines the hash value $H_3(U, \tau, ID_s, PK_{ID_s})$ as $H \leftarrow v^{-1}(uP - H_4)$, where H_4 is the output of $H_4(U, \tau, ID_s, PK_{ID_s})$. If a such a hash queries has been responded with a different value before, it aborts the simulation. This means that $\mathcal{C}$ updates list L_3 with tuple $(U, \tau, ID_s, PK_{ID_s}, \perp, H)$. Finally, $\mathcal{C}$ sets $W = D_{ID_s} + uaP$. $\mathcal{C}$ returns $\psi \leftarrow (U, W)$.

Key decapsulation queries: For a key decapsulation query on a (ψ', τ') for identities ID_s and ID_r, $\mathcal{C}$ proceeds as follows.

1. It executes the verification part of the decapsulation algorithm obtaining Q_{ID_s} and PK_{ID_s} by calling H_1 and request public key oracles. It returns $\perp$ if the verification does not succeed.
2. It checks if $ID_s = ID_j$ and if this is the case then $\mathcal{C}$ can solve the CDH problem as described below.
3. It calculates $T = \hat{e}(U, e_r P_{pub})$, where (ID_r, e_r) is obtained from H_1.
4. If $ID_r \neq ID_j$, it computes $R \leftarrow x_{ID_r} U$, where x_{ID_r} is obtained (and hence PK_{ID_r}) from either the adversary or by calling the request public key oracle. Then $\mathcal{C}$ returns $K \leftarrow H_2(U, T, R, ID_r, PK_{ID_r})$.
5. If $ID_r = ID_j$, the correct value of R cannot be computed. To return a consistent answer, $\mathcal{C}$ goes through L_2 and looks for a tuple $(U, T, R, ID_r, PK_{ID_r}, h)$, for different values of R, such that $\hat{e}(U, aP) = \hat{e}(P, R)$. If such an entry exists, the correct value of R is found and returns $K \leftarrow h$.
6. If $\mathcal{C}$ reaches this point of execution, it places the entry $(U, T, \star, ID_r, PK_{ID_r}, h)$ for a random h on list L_2 and returns $K \leftarrow h$. The symbol $\star$ denotes an unknown value of R.

Finally, $\mathcal{F}$ outputs a produces a quaternion $(\tau^*, \psi^*, ID_s^*, ID_r^*)$. $\mathcal{C}$ checks if $ID_s^* = ID_j$. If not, it aborts execution. Otherwise, it obtains $PK_{ID_s^*}$ by calling the request public key oracle on ID_s^* and retrieves t^* and l^* from lists L_3 and L_4 by querying H_3 and H_4 on $(U^*, \tau^*, ID_s^*, PK_{ID_s^*})$. Note that if $\mathcal{C}$ succeeded, then the verification condition holds:

$$\begin{aligned}
\hat{e}(P, W^*) &= \hat{e}(P_{pub}, Q_{ID_s^*})\hat{e}(U^*, H^*)\hat{e}(PK_{ID_s^*}, H'^*) \\
\hat{e}(P, W^*) &= \hat{e}(P_{pub}, Q_{ID_s^*})\hat{e}(U^*, t^*P)\hat{e}(aP, l^*bP) \\
\hat{e}(P, l^*abP) &= \hat{e}(P, W^* - D_{ID_s^*} - t^*U^*)
\end{aligned}$$

and thus $\mathcal{C}$ can compute

$$abP = (W^* - D_{ID_s^*} - t^*U)/l^*.$$

Let us now analyze the probability that $\mathcal{C}$ succeeds in solving the CDH problem instance. For this to happen, the simulation must run until the end of the game, the adversary must pick a specific identity as ID_j^*, and it must query the hash functions H_3 and H_4 to properly construct the forgery. The probability that $\mathcal{F}$ is able to produce a forgery without querying both hash functions is upper bounded by $2/2^k$.

The probability that $\mathcal{C}$ aborts the simulation is related with the following events:

- $\mathcal{F}$ places a full private key extraction on ID_j.
- $\mathcal{C}$ wants to simulate a key encapsulation query and this leads to an inconsistency in the H_3 simulation.

Note that if $\mathcal{F}$ places the first fatal query, then it could not possibly use ID_j as the sender identity in the forgery it produces at the end of the game, so we can pinpoint the probability that $\mathcal{C}$ does not abort the simulation due to these events and $\mathcal{F}$ picks the only useful case for solving CDH as $1/q_T$. Note that the maximum length of the list L_K is q_T, as stated in the Lemma

The latter fatal event occurs if $\mathcal{C}$'s simulation triggers a collision in its simulation of H_3. Since the maximum size of L_3 is $q_{SK} + q_D + q_3 + 1$, we can upper bound the probability that this occurs as $q_{SK}(q_{SK} + q_D + q_3 + 1)/2^k$. The result follows. □