Certificateless KEM and Hybrid Signcryption Schemes Revisited

S. Sharmila Deva Selvi, S. Sree Vivek * and C. Pandu Rangan*

Theoretical Computer Science Lab,
Department of Computer Science and Engineering,
Indian Institute of Technology Madras, India.
{sharmila,svivek,prangan}@cse.iitm.ac.in

Abstract. Often authentication and confidentiality are required as simultaneous key requirements in many cryptographic applications. The cryptographic primitive called signcryption effectively implements the same and while most of the public key based systems are appropriate for small messages, hybrid encryption (KEM-DEM) provides an efficient and practical way to securely communicate very large messages. Recently, Lippold et al. [13] proposed a certificateless KEM in the standard model and the first certificateless hybrid signcryption scheme was proposed by Fagen Li et al. [15]. The concept of certificateless hybrid signcryption has evolved by combining the ideas of signcryption based on tag-KEM and certificateless cryptography. In this paper, we show that [13] is not Type-I CCA secure and [15] is existentially forgeable. We also propose an improved certificateless hybrid signcryption scheme and formally prove the security of the improved scheme against both adaptive chosen ciphertext attack and existential forgery in the appropriate security models for certificateless hybrid signcryption.

Keywords. Certificateless Cryptography, Signcryption, Cryptanalysis, Hybrid Signcryption, Tag-KEM, Bilinear Pairing, Provable Security, Random Oracle Model.

1 Introduction

In 1984, Shamir [21] introduced the concept of identity based cryptography (IBC) and proposed the first identity based signature scheme. The idea of identity based cryptography is to enable a user to use any arbitrary string, that uniquely identifies him as his public key. Identity based cryptography serves as an efficient alternative to Public Key Infrastructure (PKI) because no certificate is needed to validate the public key of a user. Identity based cryptosystem makes use of a trusted third party, the private key generator (PKG), who is in possession of a master secret key which is used to derive the private key of any user in the system. Thus the private key of all the user in the system is known to the PKG, since it was generated by him. This is an inherent issue in IBC and is called as the *key escrow* problem. Certificateless Cryptography (CLC) was introduced by Al-Riyami at al. [1] to reduce the trust level of KGC (The trusted third party in CLC is the Key Generation Center) and thus to find an effective remedy to the key escrow problem in IBC. This can be achieved by splitting the private key into two parts; one is generated by the KGC and is known as the partial private key, other one is a user selected secret value. Any cryptographic operations can only be done with both these private key components or a combination of both. The public key is no longer the identity of the user in CLC but it is derived from the partial private key and the secret value of the corresponding user. The main challenge in building a CLC is to build a system that can resist two types of attacks namely Type-I and Type-II attacks (described later in the paper).

All the Certificateless Key Encapsulation Mechanism (CL-KEM) schemes [6], [14] till date are generic constructions, i.e. they combine a public key based encryption scheme and an identity based KEM and thus very in-efficient. Lippold et al. [13] proposed the first direct construction for a CCA secure CL-KEM in the standard model.

Simultaneous confidentiality and authentication of messages are often required for secure and authentic message transmission over an insecure channel like internet. Signcryption is the cryptographic primitive that offers both these properties with a very low cost when compared to encrypting and signing a message

* Work supported by Project No. CSE/05-06/076/DITX/CPAN on Protocols for Secure Communication and Computation sponsored by Department of Information Technology, Government of India

independently. Zheng [25] introduced the concept of signcryption in 1997 and many signcryption schemes were proposed till date [18, 17, 9, 7, 16, 5, 8, 4, 20], to name a few. Baek et al. in [3] gave the formal security model for signcryption and proved the security of [25] in the model. There are two different ways to construct signcryption schemes, one is public key signcryption and other is hybrid signcryption. In a public key signcryption scheme both encryption and signature are in public key setting. A few examples for this type of construct are schemes by An et al. [2], Malone-Mao [19] and Dodis et al.[12]. In a Hybrid signcryption scheme, the signature is in public key setting and encryption is in symmetric key setting, here an one-time secret key which is used in the symmetric key encryption of the message is encrypted by a public key encryption algorithm. The formal security model for a hybrid signcryption scheme was given by Dent [11] and Bj∅rstad [23]. Generation of secret key and encrypting it using a public key encryption scheme is called key encapsulation mechanism (KEM) and encrypting the message with the secret key and a symmetric key encryption scheme is called as data encryption mechanism (DEM). The definitions and formal treatment of KEM/DEM can be found in [10] and [22].

Our Contribution: Our contribution in this paper is three fold. First, we show that the CL-KEM in [13] is not CCA secure. To the best of our knowledge, there exists only one certificateless hybrid signcryption scheme (CLSC-TKEM) by Fagen Li et al. [15]. Fagen Li et al. have proposed the first CLSC-TKEM and proposed the security model for it. Next, we show that the scheme proposed in [15] is existentially forgeable. Finally, we propose an improved certificateless hybrid signcryption scheme and prove its security in the random oracle model.

2 Preliminaries

2.1 Bilinear Pairing

Let $\mathbb{G}_1$ be an additive cyclic group generated by P, with prime order q, and $\mathbb{G}_2$ be a multiplicative cyclic group of the same order. A bilinear pairing is a map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ with the following properties.

- ***Bilinearity.*** For all $P, Q, R \in \mathbb{G}_1$,
  - $\hat{e}(P + Q, R) = \hat{e}(P, R)\hat{e}(Q, R)$
  - $\hat{e}(P, Q + R) = \hat{e}(P, Q)\hat{e}(P, R)$
  - $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$ [Where $a, b \in_R \mathbb{Z}_q^*$]
- ***Non-Degeneracy.*** There exist $P, Q \in \mathbb{G}_1$ such that $\hat{e}(P, Q) \neq I_{\mathbb{G}_2}$, where $I_{\mathbb{G}_2}$ is the identity element of $\mathbb{G}_2$.
- ***Computability.*** There exists an efficient algorithm to compute $\hat{e}(P, Q)$ for all $P, Q \in \mathbb{G}_1$.

2.2 Computational Assumptions

In this section, we review the computational assumptions relevant to the protocol we propose.

Computation Diffie-Hellman Problem (CDH) Given $(P, aP, bP) \in \mathbb{G}_1^3$ for unknown $a, b \in \mathbb{Z}_q^*$, the CDH problem in $\mathbb{G}_1$ is to compute abP.

Definition. The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the CDH problem in $\mathbb{G}_1$ is defined as:

$$Adv_{\mathcal{A}}^{CDH} = Pr\left[\mathcal{A}(P, aP, bP) = abP \mid a, b \in \mathbb{Z}_q^*\right]$$

The *CDH Assumption* is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{CDH}$ is negligibly small.

Computational Bilinear Diffie-Hellman Problem (CBDH) Given $(P, aP, bP, cP) \in \mathbb{G}_1^4$ for unknown $a, b, c \in \mathbb{Z}_q^*$, the CBDH problem in $\mathbb{G}_1$ is to compute $\hat{e}(P, P)^{abc}$.

Definition. The advantage of any probabilistic polynomial time algorithm $\mathcal{A}$ in solving the CBDH problem in $\mathbb{G}_1$ is defined as:

$$Adv_{\mathcal{A}}^{CBDH} = Pr\left[\mathcal{A}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}) = 1 \mid a, b, c \in \mathbb{Z}_q^*\right]$$

The *CBDH Assumption* is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{CBDH}$ is negligibly small.

2.3 Certificateless Signcryption Tag-KEM (CLSC-TKEM)

A generic Certificateless Signcryption Tag-KEM scheme consists of the following seven probabilistic polynomial time algorithms:

- ***Setup*** (κ)***.*** Given a security parameter κ, the Key Generation Center (KGC) generates the public parameters *params* and master secret key *msk* of the system.
- ***Extract Partial Private Key*** (ID_A)***.*** Given an identity $ID_A \in_R \{0,1\}^*$ of a user A as input, the KGC computes the corresponding partial private key D_A and gives it to A in a secure way.
- ***Generate User Key*** (ID_A)***.*** Given an identity (ID_A) as input, this algorithm outputs a secret value x_A and a public key PK_A. This algorithm is executed by the user A to obtain his secret value which is used to generate his full private key and the corresponding public key which is published without certification.
- ***Set Private Key*** (D_A, x_A)***.*** The input to this algorithm are the partial private key D_A and the secret value x_A of a user A. This algorithm is executed by the user A to generate his full private key S_A.
- ***Sym*** $(ID_A, PK_A, S_A, ID_B, PK_B)$***.*** This is a symmetric key generation algorithm which takes the sender's identity ID_A, public key PK_A, private key S_A, the receiver's identity ID_B and public key PK_B as input. It is executed by the sender A in order to obtain the symmetric key K and an internal state information ω, which is not known to B.
- ***Encap*** (ω, τ)***.*** This is the key encapsulation algorithm which takes a state information ω, an arbitrary tag τ, the sender's identity ID_A, public key PK_A and private key S_A as input. This algorithm is executed by the sender A in order to obtain the encapsulation ψ. The values τ and ψ are sent to B.
- ***Decap*** $(\psi, \tau, ID_A, PK_A, ID_B, PK_B, S_B)$***.*** In order to obtain the encapsulated key K, the receiver B runs this algorithm. The input to this algorithm are the encapsulation ψ, a tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B, public key PK_B and private key S_B. The output of this algorithm is a key K or *invalid* with respect to the validity of ψ.

The consistency constraint we require is, if $(K, \omega) = Sym(ID_A, PK_A, S_A, ID_B, PK_B)$ and $\psi = Encap(\omega, \tau)$, then $K = Decap(\psi, \tau, ID_A, PK_A, ID_B, PK_B, S_B)$.

2.4 Security Model for CLSC-TKEM

The security notions for certificateless signcryption scheme was first formalized by Barbosa et al. in [4]. A CLSC scheme should satisfy indistinguishability against adaptive chosen ciphertext and identity attacks and existential unforgeability against adaptive chosen message and identity attacks. We describe below the security models to prove the *confidentiality* and *unforgeability* of a CLSC-TKEM scheme. These are the strongest security notions for this problem.

Confidentiality To prove the confidentiality of CLSC-TKEM scheme, we consider two games "IND-CLSC-TKEM-CCA2-I" and "IND-CLSC-TKEM-CCA2-II". A Type-I adversary $\mathcal{A}_I$ interacts with the challenger $\mathcal{C}$ in the IND-CLSC-TKEM-CCA2-I game and a Type-II adversary $\mathcal{A}_{II}$ interacts with the challenger $\mathcal{C}$ in the IND-CLSC-TKEM-CCA2-II game. A CLSC-TKEM scheme is indistinguishable against adaptive chosen ciphertext attacks (IND-CLSC-TKEM-CCA2), if no polynomially bounded adversaries $\mathcal{A}_I$ and $\mathcal{A}_{II}$ have non-negligible advantage in both IND-CLSC-TKEM-CCA2-I and IND-CLSC-TKEM-CCA2-II games between $\mathcal{C}$ and $\mathcal{A}_I$, $\mathcal{A}_{II}$ respectively:
IND-CLSC-TKEM-CCA2-I: The following is the interactive game between $\mathcal{C}$ and $\mathcal{A}_I$.

Setup: The challenger $\mathcal{C}$ runs this algorithm to generate the master public and private keys, *params* and *msk* respectively. $\mathcal{C}$ gives *params* to $\mathcal{A}_I$ and keeps the master private key *msk* secret from $\mathcal{A}_I$.
Phase 1: $\mathcal{A}_I$ performs a series of queries in an adaptive fashion in this phase. The queries allowed are given below:
- *Extract Partial Private Key queries:* $\mathcal{A}$ chooses an identity ID_i and gives it to $\mathcal{C}$. $\mathcal{C}$ computes the corresponding partial private key D_i and sends it to $\mathcal{A}_I$.
- *Extract Private Key queries:* $\mathcal{A}_I$ produces an identity ID_i and requests the corresponding full private key. If ID_i's public key has not been replaced then $\mathcal{C}$ responds with the full private key S_i. If $\mathcal{A}_I$ has already replaced ID_i's public key, then $\mathcal{C}$ does not provide the corresponding private key to $\mathcal{A}_I$.

- *Request Public Key queries:* $\mathcal{A}_I$ produces an identity ID_i to $\mathcal{C}$ and requests ID_i's public key. $\mathcal{C}$ responds by returning the public key PK_i for the user ID_i. (First by choosing a secret value if necessary).
- *Replace Public Key queries:* $\mathcal{A}_I$ can repeatedly replace the public key PK_i corresponding to the user identity ID_i with any value PK_i' of $\mathcal{A}_I$'s choice. The current value of the user's public key is used by $\mathcal{C}$ in any computations or responses to $\mathcal{A}_I$'s queries.
- *Symmetric Key Generation queries:* $\mathcal{A}_I$ produces a sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. The private key of the sender S_A is obtained from the corresponding list maintained by $\mathcal{C}$. $\mathcal{C}$ computes the symmetric key K and an internal state information ω, stores and keeps ω secret from the view of $\mathcal{A}_I$ and sends the symmetric key K to $\mathcal{A}_I$. It is to be noted that $\mathcal{C}$ may not be aware of the corresponding private key if the public key of ID_A is replaced. In this case $\mathcal{A}_I$ provides the private key of ID_A to $\mathcal{C}$.
- *Key Encapsulation queries:* $\mathcal{A}_I$ produces an arbitrary tag τ, the sender's identity ID_A and public key PK_A, The private key of the sender S_A is known to $\mathcal{C}$. $\mathcal{C}$ checks whether a corresponding ω value is stored previously. If ω exists then $\mathcal{C}$ computes the encapsulation ψ with ω and τ and deletes ω, else returns *invalid*.
- *Key Decapsulation queries:* $\mathcal{A}_I$ produces an encapsulation ψ, a tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B. The private key of the receiver S_B is obtained from the corresponding list maintained by $\mathcal{C}$. $\mathcal{C}$ returns the key K or *invalid* with respect to the validity of ψ. It is to be noted that $\mathcal{C}$ may not be aware of the corresponding private key if the public key of ID_B is replaced. In this case $\mathcal{A}_I$ provides the private key of ID_B to $\mathcal{C}$.

Challenge: At the end of *Phase 1* (which is decided by $\mathcal{A}_I$), $\mathcal{A}_I$ sends to $\mathcal{C}$, a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A}_I$ wishes to be challenged. Here, the private key of the receiver ID_{B^*} was not queried in **Phase 1**. Now, $\mathcal{C}$ computes $\langle K_1, \omega^* \rangle$ using $\mathrm{Sym}(ID_A, PK_A, S_A, ID_B, PK_B)$ and chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the CLSC-TKEM scheme. Now $\mathcal{C}$ chooses a bit $\delta \in_R \{0,1\}$ and sends K_δ to $\mathcal{A}_I$. When $\mathcal{A}_I$ receives K_δ, it generates an arbitrary tag τ^* and sends it to $\mathcal{C}$. $\mathcal{C}$ computes the challenge encapsulation ψ^* with ω^* and τ^* and sends ψ^* to $\mathcal{A}_I$.

Phase 2: $\mathcal{A}_I$ can perform polynomially bounded number of queries adaptively again as in *Phase 1* but it cannot make a partial private key extraction query on ID_{B^*} or cannot query for the decapsulation of ψ^*. If the public key of ID_{B^*} is replaced after the **Challenge**, $\mathcal{A}_I$ can ask for the decapsulation of ψ^*.

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

The advantage of $\mathcal{A}_I$ is defined as $Adv^{IND-CLSC-TKEM-CCA2-I}(\mathcal{A}_I) = |2Pr[\delta' = \delta] - 1|$, where $Pr[\delta' = \delta]$ denotes the probability that $\delta' = \delta$.

IND-CLSC-TKEM-CCA2-II: The following is the interactive game between $\mathcal{C}$ and $\mathcal{A}_{II}$.

Setup: The challenger $\mathcal{C}$ runs this algorithm to generate the master public and private keys, *params* and *msk* respectively. $\mathcal{C}$ gives both *params* and *msk* to $\mathcal{A}_{II}$.

Phase 1: $\mathcal{A}_{II}$ performs a series of queries in an adaptive fashion in this phase. The queries allowed are similar to that of the IND-CLSC-TKEM-CCA2-I game except that *Extract Partial Private Key queries:* is excluded because $\mathcal{A}_{II}$ can generate it on need basis as it knows *msk*.

Challenge: At the end of *Phase 1* (which is decided by $\mathcal{A}_{II}$), $\mathcal{A}_{II}$ sends to $\mathcal{C}$, a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A}_{II}$ wishes to be challenged. Here, the full private key of the receiver ID_{B^*} was not queried in **Phase 1**. Now, $\mathcal{C}$ computes $\langle K_1, \omega^* \rangle$ using $\mathrm{Sym}(ID_A, PK_A, S_A, ID_B, PK_B)$ and chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the CLSC-TKEM scheme. Now $\mathcal{C}$ chooses a bit $\delta \in_R \{0,1\}$ and sends K_δ to $\mathcal{A}_{II}$. When $\mathcal{A}_{II}$ receives K_δ, it generates an arbitrary tag τ^* and sends it to $\mathcal{C}$. $\mathcal{C}$ computes the challenge encapsulation ψ^* with ω^* and τ^* and sends ψ^* to $\mathcal{A}_{II}$.

Phase 2: $\mathcal{A}_{II}$ can perform ploynomially bounded number of queries adaptively again as in *Phase 1* but it cannot make a partial private key extraction query on ID_{B^*} or cannot query for the decapsulation of ψ^*. If the public key of ID_{B^*} is replaced after the **Challenge**, $\mathcal{A}_I$ can ask for the decapsulation of ψ^*.

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

The advantage of $\mathcal{A}_{II}$ is defined as $Adv^{IND-CLSC-TKEM-CCA2-II}(\mathcal{A}_{II}) = |2Pr[\delta' = \delta] - 1|$, where $Pr[\delta' = \delta]$ denotes the probability that $\delta' = \delta$.

Existential Unforgeability To prove the existential unforgeability of CLSC-TKEM scheme, we consider two games "EUF-CLSC-TKEM-CMA-I" and "EUF-CLSC-TKEM-CMA-II". A Type-I forger $\mathcal{F}_I$ interacts with the challenger $\mathcal{C}$ in the EUF-CLSC-TKEM-CMA-I game and a Type-II forger $\mathcal{F}_{II}$ interacts with $\mathcal{C}$ in the EUF-CLSC-TKEM-CMA-II game. A CLSC-TKEM scheme is existentially unforgeable against adaptive chosen message attack (EUF-CLSC-TKEM-CMA), if no polynomially bounded forgers $\mathcal{F}_I$ and $\mathcal{F}_{II}$ have non-negligible advantage in both EUF-CLSC-TKEM-CMA-I and EUF-CLSC-TKEM-CMA-II games between $\mathcal{C}$ and $\mathcal{F}_I$, $\mathcal{F}_{II}$ respectively:
EUF-CLSC-TKEM-CMA-I: The following is the interactive game between $\mathcal{C}$ and $\mathcal{F}_I$:

Setup: The challenger $\mathcal{C}$ runs this algorithm to generate the master public and private keys, *params* and *msk* respectively. $\mathcal{C}$ gives *params* to $\mathcal{F}_I$ and keeps the master private key *msk* secret from $\mathcal{F}_I$.

Training Phase: $\mathcal{F}_I$ performs a series of polynomially bounded number of queries in an adaptive fashion in this phase. The queries allowed are identical to the queries allowed in **Phase 1** of IND-CLSC-TKEM-CCA2-I game.

Forgery: At the end of the *Training Phase* (which is decided by $\mathcal{F}_I$), $\mathcal{F}_I$ sends to $\mathcal{C}$ an encapsulation $\langle \tau^*, \psi^*, ID_{A^*}, ID_{B^*} \rangle$, where ID_{A^*} is the sender identity and ID_{B^*} is the receiver identity. It is to be noted that the partial private key of the sender ID_{A^*} should not be queried and the public key of ID_{A^*} should not be replaced during the **Training Phase** simultaneously. In addition ψ^* should not be the response for any key encapsulation queries by $\mathcal{F}_I$ during the *Training Phase*.

$\mathcal{F}_I$ wins the game if the output of ***Decap***$(\psi^*, \tau^*, ID_{A^*}, PK_{A^*}, ID_{B^*}, PK_{B^*}, S_{B^*})$ is not *invalid*. The advantage of $\mathcal{F}_I$ is defined as the probability with which it wins the EUF-CLSC-TKEM-CMA-I game.

EUF-CLSC-TKEM-CMA-II: The following is the interactive game between $\mathcal{C}$ and $\mathcal{F}_{II}$:

Setup: The challenger $\mathcal{C}$ runs this algorithm to generate the master public and private keys, *params* and *msk* respectively. $\mathcal{C}$ gives both *params* and *msk* to $\mathcal{F}_{II}$.

Training Phase: $\mathcal{F}_{II}$ performs a series of polynomially bounded number of queries in an adaptive fashion in this phase. The queries allowed are identical to the queries allowed in **Phase 1** of IND-CLSC-TKEM-CCA2-II game.

Forgery: At the end of the *Training Phase* (which is decided by $\mathcal{F}_{II}$), $\mathcal{F}_{II}$ sends to $\mathcal{C}$ an encapsulation $\langle \tau^*, \psi^*, ID_{A^*}, ID_{B^*} \rangle$, where ID_{A^*} is the sender identity and ID_{B^*} is the receiver identity. It is to be noted that $\mathcal{F}_{II}$ should not query the secret value x_{A^*} of the sender ID_{A^*} and should not replace the public key of ID_{A^*} during the **Training Phase**. In addition ψ^* should not be the response for any key encapsulation queries by $\mathcal{F}_{II}$ during the *Training Phase*.

$\mathcal{F}_{II}$ wins the game if the output of ***Decap***$(\psi^*, \tau^*, ID_{A^*}, PK_{A^*}, ID_{B^*}, PK_{B^*}, S_{B^*})$ is not *invalid*. The advantage of $\mathcal{F}_{II}$ is defined as the probability with which it wins the EUF-CLSC-TKEM-CMA-II game.

3 Review and Attack of Lippold et al.'s CLSC-TKEM

In this section, we review the CL-KEM scheme of Lippold et al., presented in [13]. We show that the scheme in [13] is not CCA secure.

3.1 Review of the scheme

This scheme has the following five algorithms.

1. ***Setup:*** Given a security parameter κ, the KGC performs the following to setup the system:
   - Chooses two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order p.
   - It also chooses a bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$.
   - Chooses a generator $g \in_R \mathbb{G}_1$.
   - Chooses a suitable Water's hash function H, as described in [24].
   - Chooses $u_1, u_2, \alpha \in \mathbb{G}_1$, and computes $z = \hat{e}(g, \alpha)$.
   - The public parameters of the system are $params = \langle \kappa, \mathbb{G}_1, \mathbb{G}_2, p, g, H, u_1, u_2, z \rangle$ and α is the master secret key.

2. ***Identity-Based Key Derivation:*** Given the master secret key α and the identity $ID \in 0,1^n$, the KGC generates an ID-Based private key corresponding to the given identity as follows:
   – Chooses $s \in_R \mathbb{Z}_p^*$.
   – Returns $sk_{ID} = (\alpha H(ID)^s, g^s)$.
3. ***User Key Generation:*** This algorithm is executed by the user with identity ID, in order to generate his user secret value and the certificateless public key.
   – The user chooses a secret value $x_{ID} \in_R \mathbb{Z}_p^*$.
   – Computes the certificateless public key $\beta_{ID} = z^{x_{ID}}$.
   – Return $\langle x_{ID}, \beta_{ID} \rangle$.
4. ***Encapsulation:*** Given the public key β_{ID} of a user with identity ID and a message M, the sender generates an encryption key K and the corresponding encapsulation C as follows:
   – Chooses $r \in_R \mathbb{Z}_p^*$.
   – Computes $c_1 = g^r$ and $c_2 = H(ID)^r$.
   – Computes $t = TCR(c_1)$, where TCR is a Target Collision Resistant hash function.
   – Computes $c_3 = (u_1^t u_2)^r$.
   – Computes $K = \beta_{ID}^r = (z^x)^r \in \mathbb{G}_2$.
   – Computes $C = \langle c_1, c_2, c_3 \rangle \in \mathbb{G}_1^3$.
   – Returns (K, C). (Note that K is the key that is used in a symmetric data encapsulation mechanism (DEM) for the encryption of the message and is not a part of the ciphertext.)
5. ***Decapsulation:*** Given the secret keys $d_1 = \alpha H(ID)^s$, $d_2 = g^s$ and x_{ID}, and an encapsulation $C = \langle c_1, c_2, c_3 \rangle$, the receiver of the ciphertext executes this algorithm to recover the key K from C as follows:
   – Chooses $r_1, r_2 \in_R \mathbb{Z}_p^*$.
   – Computes $t = TCR(c_1)$.
   – Computes $K = \left(\frac{\hat{e}(c_1, d_1(u_1^t u_2)^{r_1} H(ID)^{r_2})}{\hat{e}(c_2, d_2 g^{r_2})\hat{e}(g^{r_1}, c_3)} \right)^{x_{ID}}$
   – Return the key K for data decapsulation.

3.2 Attack of Lippold et al.'s CL-KEM

In this section, we show that the CL-KEM by Lippold et al. [13] does not provide confidentiality.

Attack on Confidentiality: During the confidentiality game for Type-I adversary, the adversary is allowed to replace the public key of the receiver. The following attack is possible due to this liberalized constraint on the adversary.

Attack by Type-I adversary:

– Let ID^* be the target identity on which the adversary wishes to be challenged.
– The adversary chooses $x' \in_R \mathbb{Z}_p^*$.
– Replaces the public key of ID^* as $\beta'_{ID^*} = \hat{e}(g, g)^{x'}$.
– The adversary submits ID^* to the challenger as the challenge identity.

Upon receiving the challenge identity ID^*, the challenger generates the challenge encapsulation C^*, the encapsulation key K^* and sends it to the adversary. On receiving $C^* = \langle c_1^*, c_2^*, c_3^* \rangle$ and K^* from the challenger, the Type-I adversary computes the key K' from C^* as follows and distinguishes C^*:

$$\begin{aligned} K' &= \hat{e}(g^{x'}, c_1) \\ &= \hat{e}(g^{x'}, g^r) \\ &= \hat{e}(g, g)^{x' r} \\ &= (\beta'_{ID^*})^r \end{aligned}$$

Now, K' is the key corresponding to the encapsulation C^* generated by the challenger. The adversary compares K' with K^*, and returns $b' = 1$ if they are identical, $b' = 0$ otherwise.

4 Review and Attack of Fagen Li et al.'s CL-KEM

In this section we review the CLSC-TKEM scheme of Fagen Li et al, presented in [15]. We also show that the scheme in [15] is existentially forgeable.

4.1 Review of the scheme

This scheme has the following seven algorithms.

1. ***Setup:*** Given κ the security parameter, the KGC chooses two groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime order q, a bilinear map $\hat{e}: \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ and a generator $P \in_R \mathbb{G}_1$. It then chooses a master private key $s \in_R \mathbb{Z}_q^*$, sets a system-wide public key $P_{pub} = sP$ and chooses four cryptographic hash functions defined by $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$, $H_2 : \{0,1\}^* \rightarrow \{0,1\}^n$, $H_3 : \{0,1\}^* \rightarrow \mathbb{G}_1$ and $H_4 : \{0,1\}^* \rightarrow \mathbb{G}_1$. Here n is the key length of a DEM. The public parameters $Params$= $\langle \mathbb{G}_1, \mathbb{G}_2, P, \hat{e}, H_1, H_2, H_3, H_4, P_{pub} \rangle$.
2. ***Partial Private Key Extract:*** Given an identity $ID_A \in \{0,1\}^*$, the KGC does the following to extract the partial private key corresponding to ID_A:
   - Computes $Q_A = H_1(ID_A) \in \mathbb{G}_1$.
   - Sets the partial private key $D_A = sQ_A$.
3. ***Generate User Key:*** A user with identity ID_A chooses $x_A \in_R \mathcal{Z}_q^*$ and sets the public key $PK_A = x_A P$.
4. ***Set Private Key:*** The full private key of the user A is set to be $S_A = (x_A, D_A)$.
5. ***Sym*** $(ID_A, PK_A, S_A, ID_B, PK_B)$***.*** Given the sender's identity ID_A, public key PK_A, private key S_A, the receiver's identity ID_B and public key PK_B as input, the algorithm produces the symmetric key K as follows:
   - The sender A chooses $r \in_R \mathcal{Z}_q^*$,
   - Computes $U = rP$ and $T = \hat{e}(P_{pub}, Q_B)^r$,
   - Computes $K = H_2(U, T, r(PK_B), ID_B, PK_B)$,
   - Outputs K and a set $\omega = \langle r, U, ID_A, PK_A, S_A, ID_B, PK_B \rangle$
6. ***Encap*** (ω, τ)***.*** Given a state information ω and an arbitrary tag τ, the sender A obtains the encapsulation ψ by performing the following:
   - Computes $H = H_3(U, \tau, ID_A, PK_A)$.
   - Computes $H' = H_4(U, \tau, ID_A, PK_A)$.
   - Computes $W = D_A + rH + x_A H'$.
   - Output $\psi = \langle U, W \rangle$
7. ***Decap*** $(\psi, \tau, ID_A, PK_A, ID_B, PK_B, S_B)$***.*** Given the encapsulation ψ, a tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B, public key PK_B and private key S_B the key K is computed by the receiver B as follows:
   - Computes $H = H_3(U, \tau, ID_A, PK_A)$.
   - Computes $H' = H_4(U, \tau, ID_A, PK_A)$.
   - If $\hat{e}(P_{pub}, Q_A)\hat{e}(U, H)\hat{e}(PK_A, H') \stackrel{?}{=} \hat{e}(P, W)$, computes the value $T = \hat{e}(D_B, U)$ and outputs $K = H_2(U, T, x_B U, ID_B, PK_B)$, otherwise outputs *invalid*.

4.2 Attack of Fagen Li et al.'s CLSC-TKEM

We show that the CLSC-TKEM by Fagen Li et al. [15] is existentially forgeable in this section.

Attack on Unforgeability: Fagen Li et al. [15] have claimed that their scheme is existentially unforgeable against both Type-I and Type-II attacks. We show that the scheme does not resist both Type-I and Type-II attacks. It is to be noted that, in the unforgeability games, EUF-CLSC-TKEM-CMA-I and EUF-CLSC-TKEM-CMA-II the corresponding forgers $\mathcal{F}_I$ and $\mathcal{F}_{II}$ have access to the full private key of the receiver B, also, $\mathcal{F}_I$ is not allowed to extract the partial private keys of the sender A and $\mathcal{F}_{II}$ is not allowed to extract the user secret key or replace the sender A's public key. These constraints are maintained in order to ensure insider security.

Attack by Type-I forger $\mathcal{F}_I$: During the EUF-CLSC-TKEM-CMA-I game, the forger $\mathcal{F}_I$ interacts with the challenger $\mathcal{C}$ during the **Training Phase**. $\mathcal{F}_I$ has access to the various oracles offered by $\mathcal{C}$ in addition to it, $\mathcal{F}_I$ has access to the full private key of the receiver too.

- During the **Training Phase** $\mathcal{F}_I$ queries $\mathcal{C}$ for an encapsulation with ID_A as sender and ID_B as receiver with an arbitrary tag τ.
- Here, the private key of ID_A is not queried by $\mathcal{F}_I$ and the corresponding public key is not replaced.
- $\mathcal{C}$ responds with $\psi = \langle U, W \rangle$.

Now, $\mathcal{F}_I$ obtains a forged encapsulation from the encapsulation ψ received during the **Training Phase** for the same tag τ, by performing the following steps:

- Let ID_{B^*} be a user whose full private key S_{B^*} is known to $\mathcal{F}_I$.
- $\mathcal{F}_I$ computes a new key $K' = H_2(U, T', x_{B^*}U, ID_{B^*}, PK_{B^*})$, where $T' = \hat{e}(D_{B^*}, U)$.
- Now, $\psi^* = \langle U, W \rangle$ is a valid encapsulation of the key K' from the sender ID_A to a new receiver ID_{B^*}.

The correctness of the attack can be easily verified because ***Decap*** $(\psi^*, \tau, ID_A, PK_A, ID_{B^*}, PK_{B^*}, S_{B^*})$ passes the verification and yields a different key K' as follows.

- Compute $H = H_3(U, \tau, ID_A, PK_A)$ and $H' = H_4(U, \tau, ID_A, PK_A)$. It is to be noted that the computation of H and H' will yield the same value for both ciphertexts ψ and ψ^* because both the computations do not use the receiver identity and public key. Also, the value of U is same in both the ciphertexts.
- The validity check $\hat{e}(P_{pub}, Q_A)\hat{e}(U, H)\hat{e}(PK_A, H') \stackrel{?}{=} \hat{e}(P, W)$ also holds because this verification is also dependent on the sender's identity and public key alone and no receiver component is used explicitly or implicitly.

The value $T^* = \hat{e}(D_{B^*}, U)$ is computed and $K' = H_2(U, T', x_{B^*}U, ID_{B^*}, PK_{B^*})$ is output as the key. Thus ψ^* is a valid forgery with respect to the new key K'.

Attack by Type-II forger $\mathcal{F}_{II}$: The attack by Type-II forger is identical to that of the attack by the Type-I forger $\mathcal{F}_I$ because as mentioned above a Type-II forger $\mathcal{F}_{II}$ also has access to the full private key of the receiver B. The forgery can be done in a similar way as described in ***Attack by Type-I forger*** $\mathcal{F}_I$ because the attack does not involve the user secret value of the sender A, which is not available to the forger $\mathcal{F}_{II}$. ***Remark:*** We also point out that the same weakness holds for the generic Certificateless Hybrid Signcryption scheme proposed in [15]. The weakness is due to the lack of binding of the receiver identity to the signature component of the encapsulation ψ.

5 Improved CLSC-TKEM Scheme (ICLSC-TKEM)

In the preceding section we saw that the CLSC-TKEM scheme proposed by Fagen Li et al. does not withstand chosen message attack. The weakness of the scheme was due to the lack of binding between the receiver identity and the signature generated by the sender. This is the reason, for an encapsulation ψ to act as a valid encapsulation for different keys K_i (for $i = 1$ to n, where n is the number of forged keys) from a single sender to n different receivers. This weakness can be eliminated by making the following changes in the ***Sym***, ***Encap*** and ***Decap*** algorithms in Fagen Li et al's [15] scheme.

Sym $(ID_A, PK_A, S_A, ID_B, PK_B)$***.*** Given the sender's identity ID_A, public key PK_A, private key S_A, the receiver's identity ID_B and public key PK_B as input, the algorithm produces the symmetric key K as follows:

- The sender A chooses $r \in_R \mathcal{Z}_q^*$,
- Computes $U = rP$ and $T = \hat{e}(P_{pub}, Q_B)^r$,
- Computes $K = H_2(U, T, r(PK_B), ID_B, PK_B)$,
- Outputs K and a set $\omega = (r, U, T, ID_A, PK_A, S_A, ID_B, PK_B)$

Encap (ω, τ)***.*** Given a state information ω and an arbitrary tag τ, the sender A obtains the encapsulation ψ by performing the following:

- Computes $H = H_3(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$.
- Computes $H' = H_4(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$.
- Computes $W = D_A + rH + x_A H'$.
- Output $\psi = \langle U, W \rangle$

Now, it is not possible for $\mathcal{F}_I$ and $\mathcal{F}_{II}$ to generate different forged keys from a sender ID_A, whose secret key is not known to any receivers as the identity ID_B and the public key PK_B of the receiver is bound to the signature part of the encapsulation ψ which cannot be altered.

Decap $(\psi, \tau, ID_A, PK_A, ID_B, PK_B, S_B)$**.** Given the encapsulation ψ, a tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B, public key PK_B and private key S_B the key K is computed as follows:

- Computes the value $T = \hat{e}(D_B, U)$.
- Computes $H = H_3(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$.
- Computes $H' = H_4(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$.
- If $\hat{e}(P_{pub}, Q_A)\hat{e}(U, H)\hat{e}(PK_A, H') \stackrel{?}{=} \hat{e}(P, W)$ and outputs $K = H_2(U, T, x_B U, ID_B, PK_B)$, otherwise outputs *invalid*.

6 Security of the Improved CLSC-TKEM Scheme

In this section we provide the formal proof for the confidentiality and unforgeability of the improved CLSC-TKEM.

6.1 Type-I Confidentiality

Theorem 1. *If an IND-ICLSC-TKEM-CCA2-I adversary $\mathcal{A}_I$ has an advantage ϵ against the IND-ICLSC-TKEM-CCA2-I security of the ICLSC-TKEM scheme, asking q_{H_i} (i = 1, 2, 3, 4) hash queries to random oracles $\mathcal{O}_{H_i}$ (i = 1, 2, 3, 4), q_{ppk} partial private key extract queries and q_{fpk} private key extract queries, then there exist an algorithm $\mathcal{C}$ that solves the CBDH problem with the following advantage*

$$\epsilon' \geq \epsilon \left(1 - \frac{q_{ppk}}{q_{H_1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{ppk} + q_{fpk})}\right)\left(\frac{1}{q_{H_2}}\right)$$

Proof: A challenger $\mathcal{C}$ is challenged with an instance of the CBDH problem. Given $\langle P, aP, bP, cP\rangle \in \mathbb{G}_1$, $\mathcal{C}$ has to find out $\hat{e}(P, P)^{abc}$. Let $\mathcal{A}_I$ be the adversary who is capable of breaking the IND-ICLSC-TKEM-CCA2-I security of the ICLSC-TKEM scheme. $\mathcal{C}$ can make use of $\mathcal{A}_I$ to find the solution of the CBDH problem instance by playing the following interactive game with $\mathcal{A}_I$.

Setup: $\mathcal{C}$ sets the master public key $P_{pub} = aP$, designs the hash functions H_i (i =1 *to* 4) as random oracles $\mathcal{O}_{H_i}$ (i =1 *to* 4) respectively. In order to maintain the consistency between the responses to the hash queries, $\mathcal{C}$ maintains lists L_i (i =1 *to* 4) and to maintain the list of issued private keys and public keys, $\mathcal{C}$ maintains a list L_K. $\mathcal{C}$ gives the public parameters *params* to $\mathcal{A}_I$.

Phase 1: $\mathcal{A}_I$ performs a series of polynomially bounded number of queries in an adaptive fashion in this phase. The oracles and queries allowed are described below.

- $\mathcal{O}_{H_1}(ID_i)$: We will make a simplifying assumption that $\mathcal{A}_I$ queries the $\mathcal{O}_{H_1}$ oracle with distinct identities in each query. Without loss of generality, if the oracle query is repeated with an already queried identity, by definition the oracle consults the list L_1 and gives the same response. Thus, we assume that $\mathcal{A}_I$ asks q_{H_1} distinct queries for q_{H_1} distinct identities. Among this q_{H_1} identities, a random identity has to be selected by $\mathcal{C}$ as target identity and it is done as follows (Note that $\mathcal{A}_I$ should also choose this identity in the challenge phase).

  $\mathcal{C}$ selects a random index γ, where $1 \leq \gamma \leq q_{H_1}$. $\mathcal{C}$ does not reveal γ to $\mathcal{A}_I$. When $\mathcal{A}_I$ puts forth the γ^{th} query on ID_γ, $\mathcal{C}$ decides to fix ID_γ as target identity for the challenge phase. Moreover, $\mathcal{C}$ responds to $\mathcal{A}_I$ as follows:
  - If it is the γ^{th} query, then $\mathcal{C}$ sets $Q_\gamma = bP$ and stores the tuple $\langle ID_\gamma, Q_\gamma = bP, -\rangle$ in the list L_1. Here, $\mathcal{C}$ does not know b. $\mathcal{C}$ is simply using the bP value given in the instance of the CBDH problem.
  - For all other queries, $\mathcal{C}$ chooses $b_i \in_R Z_q^*$ and sets $Q_i = b_i P$ and stores $\langle ID_i, Q_i, b_i\rangle$ in the list L_1.

$\mathcal{C}$ returns Q_i to $\mathcal{A}$. (Note that as the identities are assumed to be distinct, for each query, we create distinct entry and add in the list L_1).

- $\mathcal{O}_{H_2}(U, T, r(PK_B), ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, r(PK_B), ID_B, PK_B, K\rangle$ exists in list L_2. If so, returns K to $\mathcal{A}_I$ else chooses $K \in_R \{0,1\}^n$, adds the tuple $\langle U, T, r(PK_B), ID_B, PK_B, K\rangle$ to the list L_2 and returns K to $\mathcal{A}_I$.
- $\mathcal{O}_{H_3}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A,\ PK_A,\ ID_B, PK_B, h_i, H\rangle$ exists in the list L_3. If so, returns H to $\mathcal{A}_I$ else performs the following:
  - If $ID_B \neq ID_\gamma$, $\mathcal{C}$ chooses $h_i \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i, H = h_iP\rangle$ to the list L_3 and returns H to $\mathcal{A}_I$.
  - If $ID_B = ID_\gamma$, $\mathcal{C}$ chooses $h_i \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i, H = h_iP_{pub}\rangle$ to the list L_3 and returns H to $\mathcal{A}_I$.
- $\mathcal{O}_{H_4}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h'_i, H'\rangle$ exists in the list L_4. If so, returns H' to $\mathcal{A}_I$ else chooses $h'_i \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h'_i, H' = h'_iP\rangle$ to the list L_4 and returns H' to $\mathcal{A}_I$.
- $\mathcal{O}_{ExtractPartialPrivateKey}(ID_i)$: On a request for the partial private key of a user with identity ID_i, $\mathcal{C}$ aborts if $ID_i = ID_\gamma$. Else, $\mathcal{C}$ retrieves the tuple $\langle ID_i, Q_i, b_i\rangle$ from list L_1, returns $D_i = b_iaP = aQ_i$ and adds the tuple $\langle ID_i, -, D_i, -\rangle$ to the list L_K.

  (**Note:** It is assumed throughout the confidentiality game, $\mathcal{A}_I$ queries $\mathcal{O}_{H_1}$ oracle with ID_i before querying any other oracles with ID_i as input.)
- $\mathcal{O}_{RequestPublicKey}(ID_i)$: $\mathcal{A}_I$ produces an identity ID_i to $\mathcal{C}$ and requests ID_i's public key. $\mathcal{C}$ checks in the list L_K for an tuple of the form $\langle ID_i, x_i, D_i, PK_i\rangle$. If an entry exists, then responds by returning the corresponding public key PK_i to $\mathcal{A}_I$. If it does not exist, $\mathcal{C}$ chooses $x_i \in_R \mathbb{Z}_q^*$, sets $PK_i = x_iP$, adds the tuple $\langle ID_i, x_i, -, PK_i\rangle$ (note that D_i is computed only after the $\mathcal{O}_{ExtractPartialPrivateKey}(ID_i)$ query) to the list L_K and sends the corresponding PK_i to $\mathcal{A}_I$.
- $\mathcal{O}_{ExtractPrivateKey}(ID_i)$: $\mathcal{A}_I$ produces an identity ID_i and requests the corresponding full private key. If ID_i's public key has not been replaced and if $ID_i \neq ID_\gamma$, then $\mathcal{C}$ responds with the full private key $S_i = \langle x_i, D_i\rangle$ retrieving it from the list L_K. If $\mathcal{A}_I$ has already replaced ID_i's public key, then $\mathcal{C}$ does not provide the corresponding private key to $\mathcal{A}_I$ (Note that $\mathcal{A}_I$ is allowed to replace the public key of any identity including ID_γ, thus $\mathcal{A}_I$ knows the user secret value corresponding to any identity, including ID_γ). If $ID_i = ID_\gamma$ then $\mathcal{C}$ aborts.
- $\mathcal{O}_{ReplacePublicKey}(PK'_i)$: In order to replace the public key PK_i of a user ID_i with any value PK'_i of $\mathcal{A}_I$'s choice, $\mathcal{C}$ updates the corresponding tuple in the list L_K as $\langle ID_i, -, D_i, PK'_i\rangle$. The current value of the user's public key is used by $\mathcal{C}$ in for computations or responses to any queries made by $\mathcal{A}_I$.
- $\mathcal{O}_{SymmetricKeyGeneration}$: $\mathcal{A}_I$ produces a sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. Now, $\mathcal{C}$ computes the symmetric key K and an internal state information ω, stores and keeps ω secret from the view of $\mathcal{A}_I$ and sends the symmetric key K to $\mathcal{A}_I$. It is to be noted that $\mathcal{C}$ can perform this even if $\mathcal{C}$ does not know the private key corresponding to the sender ID_A or the receiver ID_B because computing K does not involve the private key of either the sender or receiver.
- $\mathcal{O}_{KeyEncapsulation}$: $\mathcal{A}_I$ produces an arbitrary tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. The full private key of the sender $S_A = (x_A, D_A)$ is obtained from the list L_K. $\mathcal{C}$ checks whether a corresponding ω value is stored previously.
  - If ω does not exist, $\mathcal{C}$ returns *invalid*.
  - If a corresponding ω exists and $ID_A \neq ID_\gamma$, then $\mathcal{C}$ computes the encapsulation ψ with ω and τ as per the actual encapsulation algorithm, and deletes ω.
  - If a corresponding ω exists and $ID_A = ID_\gamma$, then $\mathcal{C}$ performs the following to compute ψ (It is to be noted that $\mathcal{C}$ does not know the private key corresponding to ID_γ, so it cooks up the encapsulation in a different way):

* Chooses $r, h_i, h'_i \in_R \mathcal{Z}_q^*$ and computes $U = rP - h_i^{-1}Q_A$, where $Q_A = bP$ is obtained from the list L_1.
* Computes $H = h_i P_{pub}$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i, H\rangle$ to the list L_3.
* Computes $H' = h'_i P$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h'_i, H'\rangle$ to the list L_4.
* Computes $W = rH + h'_i PK_A$ (This is possible because $\mathcal{C}$ knows the public key PK_A of the sender A which is $x_A P$).
* Outputs, $\psi = \langle U, W\rangle$ as the encapsulation.

We show that, $\psi = \langle U, W\rangle$ passes the verification done by $\mathcal{A}_I$ to validate the encapsulation, because the equality $\hat{e}(P_{pub}, Q_A)\ \hat{e}(U, H)\ \hat{e}(PK_A, H') \stackrel{?}{=} \hat{e}(P, W)$ holds.

Correctness:

$$\begin{aligned}
\hat{e}(P_{pub}, Q_A)\hat{e}(U, H)\hat{e}(PK_A, H') &= \hat{e}(aP, bP)\hat{e}(rP - h_i^{-1}Q_A, H)\hat{e}(x_A P, H') \\
&= \hat{e}(aP, bP)\hat{e}(rP, H)\hat{e}(-h_i^{-1}Q_A, H)\hat{e}(x_A P, H') \\
&= \hat{e}(aP, bP)\hat{e}(rP, h_i aP)\hat{e}(h_i^{-1}bP, h_i aP)^{-1}\hat{e}(x_A P, h'_i P) \\
&= \hat{e}(aP, bP)\hat{e}(rP, h_i aP)\hat{e}(aP, bP)^{-1}\hat{e}(x_A P, h'_i P) \\
&= \hat{e}(rP, h_i aP)\hat{e}(x_A P, h'_i P) \\
&= \hat{e}(P, h_i arP + x_A h'_i P) \\
&= \hat{e}(P, rH + h'_i PK_A) \\
&= \hat{e}(P, W)
\end{aligned}$$

– $\mathcal{O}_{KeyDecapsulation}$: $\mathcal{A}_I$ produces an encapsulation ψ, a tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. The private key of the receiver S_B is obtained from the list L_K. It is to be noted that $\mathcal{C}$ may not be aware of the corresponding private key if the public key of ID_B is replaced. In this case $\mathcal{C}$ obtains the private key of ID_B from $\mathcal{A}_I$.
  - If $ID_B \neq ID_\gamma$, then $\mathcal{C}$ computes the decapsulation of ψ as per the actual decapsulation algorithm.
  - If $ID_B = ID_\gamma$, then $\mathcal{C}$ computes K from ψ as follows:
    * Searches in the list L_3 and L_4 for entries of the type $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i, H\rangle$ and $\langle U, T, \tau, ID_A, PK_A, ID_B, PK_B, h'_i, H'\rangle$ respectively.
    * If entries H and H' exist then $\mathcal{C}$ checks whether the equality $\hat{e}(P_{pub}, Q_A)\ \hat{e}(U, H)\ \hat{e}(PK_A, H') \stackrel{?}{=} \hat{e}(P, W)$ holds.
    * If the above equality holds, then retrieves the corresponding value of T from the lists L_3 and L_4 (note that both the T values should be equal).
    * Now, $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, x_B U, ID_B, PK_B, K\rangle$ exists in the list L_2. If it exists output the corresponding K value as the decapsulation of ψ.

Challenge: At the end of *Phase 1*, $\mathcal{A}_I$ sends to $\mathcal{C}$, a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A}_I$ wishes to be challenged. Here, the full private key of the receiver ID_{B^*} was not queried in **Phase 1**. $\mathcal{C}$ aborts the game if $ID_{B^*} \neq ID_\gamma$, $\mathcal{C}$ performs the following to compute the challenge encapsulation ψ^*.
– Sets $U = cP$ and chooses $T \in_R \mathbb{G}_2$.
– Chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the ICLSC-TKEM scheme
– Computes $K_1 = H_2(U, T, x_B U, ID_B, PK_B)$.
– Sets $\omega^* = \langle -, U, U', T, ID_A, PK_A, S_A, ID_B, PK_B\rangle$.
– Now, $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends K_δ to $\mathcal{A}_I$.
– $\mathcal{A}_I$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$.
– Chooses $h_i, h'_i \in_R \mathbb{Z}_q^*$, stores the tuple $\langle U, \tau^*, T, ID_A, PK_A, ID_B, PK_B, h_i, H = h_i P\rangle$ to the list L_3 and $\langle U, \tau^*, T, ID_A, PK_A, ID_B, PK_B, h'_i, H' = h'_i P\rangle$ to the list L_4.
– Since $\mathcal{C}$ knows the private key of the sender, $\mathcal{C}$ computes $W = D_A + h_i cP + h'_i x_A P$.
– $\mathcal{C}$ now sends $\psi^* = \langle U, W\rangle$ to $\mathcal{A}_I$.

Phase II: $\mathcal{A}_I$ adaptively queries the oracles as in ***Phase I***, consistent with the constraints for Type-I adversary. Besides this it cannot query *decapsulation* on ψ^*.

Guess: Since $\mathcal{A}_I$ is capable of breaking the IND-ICLSC-TKEM-CCA2-I security of ICLSC-TKEM (which is assumed at the beginning of the proof), $\mathcal{A}_I$ should have queried $\mathcal{O}_{H_2}$ with $(U, T, x_B U, ID_B, PK_B)$ as inputs. it is to be noted that $T = \hat{e}(P_{pub}, Q_B)^r = \hat{e}(aP, bP)^c$. Therefore, if the list L_2 has q_{H_2} queries corresponding to the sender ID_A and receiver ID_B, one of the T's among q_{H_2} values stored in the list L_2, is the solution for the CBDH problem instance. Now, $\mathcal{C}$ chooses one T value uniformly at random from the q_{H_2} values from the list L_2 and outputs it as the solution for the CBDH instance.

Analysis: We now assess the probability of success of the challenger $\mathcal{C}$. The events in which $\mathcal{C}$ aborts the IND-ICLSC-TKEM-CCA2-I game are,

1. E_1 - when $\mathcal{A}_I$ queries the partial private key of the target identity ID_γ and $Pr[E_1] = \frac{q_{ppk}}{q_{H_1}}$.
2. E_2 - when $\mathcal{A}_I$ queries the full private key of the target identity ID_γ and $Pr[E_2] = \frac{q_{fpk}}{q_{H_1}}$.
3. E_3 - when $\mathcal{A}_I$ does not choose the target identity ID_γ as the receiver during the challenge and $Pr[E_3] = \left(1 - \frac{1}{q_{H_1} - (q_{ppk} + q_{fpk})}\right)$.

The probability that, $\mathcal{C}$ does not abort the IND-ICLSC-TKEM-CCA2-I game is given by

$$(Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3]) = \left(1 - \frac{q_{ppk}}{q_{H_1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{ppk} + q_{fpk})}\right)$$

The probability that, the T chosen randomly from L_2 by $\mathcal{C}$, being the solution to CBDHP is $\left(\frac{1}{q_{H_2}}\right)$. Therefore, the probability of $\mathcal{C}$ solving CBDHP is given by,

$$Pr[\mathcal{C}(P, aP, bP, cP) = \hat{e}(P,P)^{abc}] = \epsilon\left(1 - \frac{q_{ppk}}{q_{H_1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{ppk} + q_{fpk})}\right)\left(\frac{1}{q_{H_2}}\right)$$

Since ϵ is non-negligible, the probability of $\mathcal{C}$ solving CBDHP is also non-negligible. □

6.2 Type-II Confidentiality

Theorem 2. *If an IND-ICLSC-TKEM-CCA2-II adversary $\mathcal{A}_{II}$ has an advantage ϵ against the IND-ICLSC-TKEM-CCA2-II security of the ICLSC-TKEM scheme, asking q_{H_i} $(i = 1,\ 2,\ 3,\ 4)$ hash queries to random oracles $\mathcal{O}_{H_i}$ $(i = 1,\ 2,\ 3,\ 4)$, q_{fpk} full private key extract queries and q_{rpk} replace public key queries, then there exist an algorithm $\mathcal{C}$ that solves the CDH problem with the following advantage*

$$\epsilon' \geq \epsilon\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(1 - \frac{q_{rpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{fpk} + q_{rpk})}\right)\left(\frac{1}{q_{H_2}}\right)$$

Proof: A challenger $\mathcal{C}$ is challenged with an instance of the CDH problem say, given $\langle P, aP, bP\rangle \in \mathbb{G}_1$, $\mathcal{C}$ has to find out P^{ab}. Let $\mathcal{A}_{II}$ be the adversary who is capable of breaking the IND-ICLSC-TKEM-CCA2-II security of the ICLSC-TKEM scheme. $\mathcal{C}$ can make use of $\mathcal{A}_{II}$ to solve the CDH problem instance by playing the following interactive game, with $\mathcal{A}_{II}$.

Setup: $\mathcal{C}$ chooses $s \in_R \mathbb{Z}_q^*$, sets the master public key $P_{pub} = sP$ and gives s to $\mathcal{A}_{II}$. The hash functions H_i $(i = 1\ to\ 4)$ are viewed as random oracles $\mathcal{O}_{H_i}$ $(i = 1\ to\ 4)$ respectively. In order to maintain the consistency between the responses to the hash queries, $\mathcal{C}$ maintains lists L_i $(i = 1\ to\ 4)$ and to maintain the list of issued private keys and public keys, $\mathcal{C}$ maintains a list L_K. $\mathcal{C}$ gives the public parameters *params* to $\mathcal{A}_{II}$.

Phase 1: $\mathcal{A}_{II}$ performs a series of polynomially bounded number of queries in an adaptive fashion in this phase. The oracles and queries allowed are described below.

- $\mathcal{O}_{H_1}(ID_i)$: When $\mathcal{A}_{II}$ queries the $\mathcal{O}_{H_1}$ oracle, $\mathcal{C}$ chooses $b_i \in_R Z_q^*$ and sets $Q_i = b_iP$ and stores $\langle ID_i, Q_i, b_i\rangle$ in the list L_1. $\mathcal{C}$ returns Q_i to $\mathcal{A}$.
- $\mathcal{O}_{H_2}(U, T, r(PK_B), ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, r(PK_B), ID_B, PK_B, K\rangle$ exists in list L_2. If so, returns K to $\mathcal{A}_I$ else chooses $K \in_R \{0,1\}^n$, adds the tuple $\langle U, T, r(PK_B), ID_B, PK_B, K\rangle$ to the list L_2 and returns K to $\mathcal{A}_{II}$.

- $\mathcal{O}_{H_3}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A,\ PK_A,\ ID_B, PK_B, h_i, H\rangle$ exists in the list L_3. If so, returns H to $\mathcal{A}_I$ else chooses $h_i \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i, H = h_iP\rangle$ to the list L_3 and returns H to $\mathcal{A}_{II}$.
- $\mathcal{O}_{H_4}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i', H'\rangle$ exists in the list L_4. If so, returns H' to $\mathcal{A}_I$ else performs the following:
  - If $ID_A \neq ID_\gamma$, $\mathcal{C}$ chooses $h_i' \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i', H' = h_i'P\rangle$ to the list L_4 and returns H' to $\mathcal{A}_{II}$.
  - If $ID_A = ID_\gamma$, $\mathcal{C}$ chooses $h_i' \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i', H' = h_i'bP\rangle$ to the list L_4 and returns H' to $\mathcal{A}_{II}$.
- $\mathcal{O}_{ExtractPartialPrivateKey}(ID_i)$: On a request for the partial private key of a user with identity ID_i, $\mathcal{C}$ retrieves b_i corresponding to ID_i from list L_1 and returns $D_i = b_isP$.

  (**Note:** It is assumed throughout the confidentiality game, $\mathcal{A}_{II}$ queries $\mathcal{O}_{H_1}$ oracle with ID_i before querying any other oracles with ID_i as input.)
- $\mathcal{O}_{RequestPublicKey}(ID_i)$*:* During **Phase I**, the model requires the adversary $\mathcal{A}_{II}$ and the challenger $\mathcal{C}$ to work with a signcryption from user U_A to U_B, where U_A and U_B satisfy the following conditions:
  - The hash of the identity ID_A of the sender U_A should have been queried by $\mathcal{A}_{II}$ to $\mathcal{O}_{H_1}$.
  - At the same time, $\mathcal{A}_{II}$ should not know which of the identity that $\mathcal{A}_{II}$ has queried is selected by $\mathcal{C}$ for the challenge phase.

  In order to achieve the selection of identities satisfying the above conditions, we specify the Request Public Key oracle as follows:

  $\mathcal{A}_{II}$ will generate a series of user identities and send each of them to $\mathcal{C}$ and ask their corresponding public key. One of them will be chosen by $\mathcal{C}$, say the γ^{th} distinct user identity, as target identity for the challenge phase. However, $\mathcal{C}$ will not reveal the value of γ or ID_γ to $\mathcal{A}_{II}$. Moreover, $\mathcal{C}$ will choose γ randomly for each game. Thus, while the target identity is one of the identities chosen by $\mathcal{A}_{II}$, $\mathcal{A}_{II}$ will not know which one is that. From now on, ID_γ denotes the specific target identity selected by $\mathcal{C}$.

  The oracle description and the responses by $\mathcal{C}$ are described now. $\mathcal{A}_{II}$ produces an identity ID_i to $\mathcal{C}$ and requests ID_i's public key. $\mathcal{C}$ performs the following :
  - The response of $\mathcal{C}$ depends upon whether the query is γ^{th} query or not.

    Case 1: If the query is not the γ^{th} query then $\mathcal{C}$ proceeds as follows:
    * $\mathcal{C}$ checks in list L_K whether an entry of the type $\langle ID_i, x_i, D_i, PK_i\rangle$ already exists in the list. If a tuple appears in L_K, $\mathcal{C}$ retrieves PK_i from the tuple and returns it as the public key of ID_i to $\mathcal{A}_{II}$.
    * If no matching tuple exists for ID_i, $\mathcal{C}$ chooses $x_i \in_R \mathbb{Z}_q^*$, generates the public key $PK_i = x_iP$ corresponding to ID_i, stores $\langle ID_i, x_i, -, PK_i\rangle$ in L_K and sends PK_i to $\mathcal{A}_{II}$.

    Case 2: If the query is the γ^{th} query then $\mathcal{C}$ proceeds as follows:
    * Checks whether a tuple of the form $\langle ID_i, x_i, D_i, PK_i\rangle$, corresponding to ID_i exists in list L_K, if it is available then $\mathcal{C}$ retrieves PK_i from the tuple and returns it as the public key of ID_i to $\mathcal{A}_{II}$.
    * If the query is the γ^{th} query and no tuple corresponding to ID_i exists in list L_K, $\mathcal{C}$ secretly sets $ID_\gamma = ID_i$ (The target identity) and proceeds as follows:
      1. Sets $PK_\gamma = aP$.
      2. Adds $\langle ID_i, -, -, PK_i\rangle$ to the list L_K.
      3. Sends PK_i to $\mathcal{A}_{II}$.
- $\mathcal{O}_{ExtractPrivateKey}(ID_i)$*:* $\mathcal{A}_I$ produces an identity ID_i and requests the corresponding full private key. If ID_i's public key has not been replaced and if $ID_i \neq ID_\gamma$, then $\mathcal{C}$ responds with the full private key $S_i = \langle x_i, D_i\rangle$ retrieving it from the list L_K. If $\mathcal{A}_{II}$ has already replaced ID_i's public key, then $\mathcal{C}$ does not provide the corresponding private key to $\mathcal{A}_{II}$ (Note that $\mathcal{A}_{II}$ is allowed to replace the public key of any identity excluding ID_γ, thus $\mathcal{A}_{II}$ is allowed to know the user secret value corresponding to all the identity excluding ID_γ). $\mathcal{A}_{II}$ aborts if $ID_i = ID_\gamma$.

– $\mathcal{O}_{ReplacePublicKey}(PK_i')$: The public key PK_i corresponding to the identity ID_i can be replaced by another value PK_i' which is chosen by $\mathcal{A}_{II}$. Before replacing, $\mathcal{C}$ checks whether $ID_i = ID_\gamma$ if so, $\mathcal{C}$ aborts the game; else, $\mathcal{C}$ updates the corresponding tuple in the list L_K as $\langle ID_i, -, D_i, PK_i' \rangle$. Henceforth the current public key (i.e. replaced public key) is used by $\mathcal{C}$ for computations or responses to any queries made by $\mathcal{A}_{II}$.

– $\mathcal{O}_{SymmetricKeyGeneration}$: $\mathcal{A}_{II}$ produces a sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. Now, $\mathcal{C}$ computes the symmetric key K and an internal state information ω, stores and keeps ω secret from the view of $\mathcal{A}_{II}$ and sends the symmetric key K to $\mathcal{A}_{II}$. It is to be noted that $\mathcal{C}$ can perform this even if $\mathcal{C}$ does not know the private key corresponding to the sender ID_A or the receiver ID_B because computing K does not involve the private key of either the sender or receiver.

– $\mathcal{O}_{KeyEncapsulation}$: $\mathcal{A}_{II}$ produces an arbitrary tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. The full private key of the sender $S_A = (x_A, D_A)$ is obtained from the list L_K. $\mathcal{C}$ checks whether a corresponding ω value is stored previously.
  - If ω does not exist, $\mathcal{C}$ returns *invalid*.
  - If a corresponding ω exists and $ID_A \neq ID_\gamma$, then $\mathcal{C}$ computes the encapsulation ψ with ω and τ as per the actual encapsulation algorithm, and deletes ω.
  - If a corresponding ω exists and $ID_A = ID_\gamma$, then $\mathcal{C}$ performs the following to compute ψ (It is to be noted that $\mathcal{C}$ does not know the private key corresponding to ID_γ, so it cooks up the encapsulation in a different way):
    * Chooses $r, h_i, h_i' \in_R \mathcal{Z}_q^*$ and computes $U = rP - h_i^{-1} h_i' PK_A$, where PK_A is obtained from the list L_K.
    * Computes $H = h_i P$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i, H \rangle$ to the list L_3.
    * Computes $H' = h_i' P$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i', H' \rangle$ to the list L_4.
    * Computes $W = rH + b_i sP$ (This is possible because $\mathcal{C}$ knows the the master private key s).
    * Outputs, $\psi = \langle U, W \rangle$ as the encapsulation.

  We show that, $\psi = \langle U, W \rangle$ passes the verification done by $\mathcal{A}_I$ to validate the encapsulation, because the equality $\hat{e}(P_{pub}, Q_A)\ \hat{e}(U, H)\ \hat{e}(PK_A, H') \stackrel{?}{=} \hat{e}(P, W)$ holds.

  Correctness:
$$\begin{aligned}
\hat{e}(P_{pub}, Q_A)\hat{e}(U, H)\hat{e}(PK_A, H') &= \hat{e}(sP, b_iP)\hat{e}(rP - h_i^{-1}h_i'PK_A, H)\hat{e}(PK_A, H') \\
&= \hat{e}(sP, b_iP)\hat{e}(rP, H)\hat{e}(-h_i^{-1}h_i'PK_A, H)\hat{e}(PK_A, H') \\
&= \hat{e}(sP, b_iP)\hat{e}(rP, h_iP)\hat{e}(-h_i^{-1}h_i'PK_A, h_iP)\hat{e}(PK_A, h_i'P) \\
&= \hat{e}(sP, b_iP)\hat{e}(rP, h_iP)\hat{e}(-h_i'PK_A, P)\hat{e}(PK_A, h_i'P) \\
&= \hat{e}(sP, b_iP)\hat{e}(rP, h_iP) \\
&= \hat{e}(P, rh_iP + b_isP) \\
&= \hat{e}(P, W)
\end{aligned}$$

– $\mathcal{O}_{KeyDecapsulation}$: $\mathcal{A}_{II}$ produces an encapsulation ψ, a tag τ, the sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. The private key of the receiver S_B is obtained from the list L_K. It is to be noted that $\mathcal{C}$ may not be aware of the corresponding private key if the public key of ID_B is replaced. In this case $\mathcal{C}$ obtains the private key of ID_B from $\mathcal{A}_I$.
  - If $ID_B \neq ID_\gamma$, then $\mathcal{C}$ computes the decapsulation of ψ as per the actual decapsulation algorithm.
  - If $ID_B = ID_\gamma$, then $\mathcal{C}$ computes K from ψ as follows:
    * Searches in the list L_3 and L_4 for entries of the type $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, h_i, H \rangle$ and $\langle U, T, \tau, ID_A, PK_A, ID_B, PK_B, h_i', H' \rangle$ respectively.
    * If entries H and H' exist then $\mathcal{C}$ checks whether the equality $\hat{e}(P_{pub}, Q_A)\ \hat{e}(U, H)\ \hat{e}(PK_A, H') \stackrel{?}{=} \hat{e}(P, W)$ holds.
    * If the above equality holds, then retrieves the corresponding value of T from the lists L_3 and L_4 (note that both the T values should be equal).
    * Now, $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, U' = x_B U, ID_B, PK_B, K \rangle$ exists in the list L_2. If it exists then check whether $\hat{e}(U', P) \stackrel{?}{=} \hat{e}(U, bP)$. If the check holds then output the corresponding K value as the decapsulation of ψ.

Challenge: At the end of *Phase 1*, $\mathcal{A}_{II}$ sends to $\mathcal{C}$, a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A}_{II}$ wishes to be challenged. Here, the full private key of the receiver ID_{B^*} was not queried in **Phase 1**. $\mathcal{C}$ aborts the game if $ID_{B^*} \neq ID_\gamma$, $\mathcal{C}$ performs the following to compute the challenge encapsulation ψ^*.
- Sets $U = bP$ and computes $T = \hat{e}(D_B, U)$.
- Chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the ICLSC-TKEM scheme
- Chooses $U' \in_R \mathbb{G}_1$ and computes $K_1 = H_2(U, T, U', ID_B, PK_B)$.
- Sets $\omega^* = \langle -, U, U', T, ID_A, PK_A, S_A, ID_B, PK_B \rangle$.
- Now, $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends K_δ to $\mathcal{A}_{II}$.
- $\mathcal{A}_{II}$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$.
- Chooses $h_i, h'_i \in_R \mathbb{Z}_q^*$, stores the tuple $\langle U, \tau^*, T, ID_A, PK_A, ID_B, PK_B, h_i, H = h_i P \rangle$ to the list L_3 and $\langle U, \tau^*, T, ID_A, PK_A, ID_B, PK_B, h'_i, H' = h'_i P \rangle$ to the list L_4.
- Since $\mathcal{C}$ knows the private key of the sender, $\mathcal{C}$ computes $W = D_A + h_i bP + h'_i x_A P$.
- $\mathcal{C}$ now sends $\psi^* = \langle U, W \rangle$ to $\mathcal{A}_{II}$.

Phase II: $\mathcal{A}_{II}$ adaptively queries the oracles as in ***Phase I***, consistent with the constraints for Type-II adversary. Besides this it cannot query *decapsulation* on ψ^*.

Guess: Since $\mathcal{A}_{II}$ is capable of breaking the IND-ICLSC-TKEM-CCA2-II security of ICLSC-TKEM (which is assumed at the beginning of the proof), $\mathcal{A}_{II}$ should have queried $\mathcal{O}_{H_2}$ with (U, T, U', ID_B, PK_B) as inputs. it is to be noted that $U' = x_B U = abP$. Therefore, if the list L_2 has q_{H_2} queries corresponding to the sender ID_A and receiver ID_B, one of the U''s among q_{H_2} values stored in the list L_2, is the solution for the CDH problem instance. Now, $\mathcal{C}$ chooses one T value uniformly at random from the q_{H_2} values from the list L_2 and outputs it as the solution for the CDH instance.

Analysis: We now assess the probability of success of the challenger $\mathcal{C}$. The events in which $\mathcal{C}$ aborts the IND-ICLSC-TKEM-CCA2-II game are,

1. E_1 - when $\mathcal{A}_{II}$ queries the full private key of the target identity ID_γ and $Pr[E_1] = \dfrac{q_{fpk}}{q_{H_1}}$.
2. E_2 - when $\mathcal{A}_{II}$ requests to replace the public key of the target identity ID_γ and $Pr[E_2] = \dfrac{q_{rpk}}{q_{H_1}}$.
3. E_3 - when $\mathcal{A}_{II}$ does not choose the target identity ID_γ as the receiver during the challenge and $Pr[E_3] = \left(1 - \dfrac{1}{q_{H_1} - (q_{fpk} + q_{rpk})}\right)$.

The probability that, $\mathcal{C}$ does not abort the IND-ICLSC-TKEM-CCA2-II game is given by

$$(Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3]) = \left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(1 - \frac{q_{rpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{fpk} + q_{rpk})}\right)$$

The probability that, the U' chosen randomly from L_2 by $\mathcal{C}$, being the solution to CDH problem is $\left(\frac{1}{q_{H_2}}\right)$. Therefore, the probability of $\mathcal{C}$ solving the CDH instance is given by,

$$Pr[\mathcal{C}(P, aP, bP, cP) = \hat{e}(P, P)^{abc}] = \epsilon \left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(1 - \frac{q_{rpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{fpk} + q_{rpk})}\right)\left(\frac{1}{q_{H_2}}\right)$$

Since ϵ is non-negligible, the probability of $\mathcal{C}$ solving CDH problem is also non-negligible. □

6.3 Type-I Unforgeability

Theorem 3. *If there exists a forger $\mathcal{F}_I$ with an advantage ϵ against the EUF-ICLSC-TKEM-CMA-I security of the ICLSC-TKEM scheme, asking q_{H_i} (i = 1, 2, 3, 4) hash queries to random oracles $\mathcal{O}_{H_i}$ (i = 1, 2, 3, 4), q_{ppk} partial private key extract queries and q_{fpk} full private key extract queries, then there exist an algorithm $\mathcal{C}$ that solves the CDH problem with an advantage*

$$\epsilon' \geq \left(\epsilon - \frac{1}{2^{\kappa-1}}\right)\left(1 - \frac{q_{ppk}}{q_{H_1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{ppk} + q_{fpk})}\right)$$

Proof: A challenger $\mathcal{C}$ is challenged with an instance of the CDH problem say $\langle P, aP, bP \rangle \in \mathbb{G}_1$. Let $\mathcal{F}_I$ be a forger who is capable of breaking the EUF-ICLSC-TKEM-CMA-I security of the ICLSC-TKEM scheme. $\mathcal{C}$ can make use of $\mathcal{F}_I$ to compute the solution abP of the CDH instance by playing the following interactive game with $\mathcal{F}_I$.

Setup: $\mathcal{C}$ sets the master public key P_{pub} as aP, designs the hash functions H_i (i =1 *to* 4) as random oracles $\mathcal{O}_{H_i}$ (i =1 *to* 4) respectively. In order to maintain the consistency between the responses to the hash queries, $\mathcal{C}$ maintains lists L_i (i =1 *to* 4) and to maintain the list of issued private keys and public keys, $\mathcal{C}$ maintains a list L_K. $\mathcal{C}$ gives the public parameters *params* to $\mathcal{F}_I$.

Training Phase: $\mathcal{F}_I$ performs a series of polynomially bounded number of queries in an adaptive fashion in this phase. The oracles and queries allowed are described below.

- All the oracles are identical to that of the $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$, $\mathcal{O}_{H_3}$, $\mathcal{O}_{H_4}$, $\mathcal{O}_{ExtractPartialPrivateKey}$, $\mathcal{O}_{Request PublicKey}$, $\mathcal{O}_{ExtractPrivateKey}$, $\mathcal{O}_{ReplacePublicKey}$, $\mathcal{O}_{SymmetricKeyGeneration}$, $\mathcal{O}_{KeyEncapsulation}$ and $\mathcal{O}_{KeyDecapsulation}$ oracles in the IND-ICLSC-TKEM-CCA2-I game.

Forgery: At the end of the *Training Phase* (which is decided by $\mathcal{F}_I$), $\mathcal{F}_I$ sends to $\mathcal{C}$ an encapsulation $\langle \tau^*, \psi^* = \langle U, W \rangle, ID_\gamma, ID_B \rangle$, where ID_γ is the sender identity and ID_B is the receiver identity. It is to be noted that the partial private key of the sender ID_γ was not queried and the public key of ID_γ could be replaced during the **Training Phase**. In addition, ψ^* should not be the response for any key encapsulation queries by $\mathcal{F}_I$ during the *Training Phase*. If ψ^* is generated with the above restrictions, then $\mathcal{C}$ can obtain the solution for the CDH instance by performing the following steps.

- Retrieves the tuple $\langle U, \tau^*, T, ID_\gamma, PK_\gamma, ID_B, PK_B, h_i, H = h_i P \rangle$ from the list L_3 and $\langle U, \tau^*, T, ID_\gamma, PK_\gamma, ID_B, PK_B, h_i, H' = h'_i P \rangle$ from the list L_4.
- Output $W - h_i U - x_\gamma h'_i P = abP$ as the solution to the CDH problem instance.

Correctness:

$$\begin{aligned} W - h_i U - x_\gamma h'_i P &= D_\gamma + rH + x_\gamma H' - h_i U - x_\gamma h'_i P \\ &= D_\gamma + rH + x_\gamma H' - r h_i P - x_\gamma H' \\ &= D_\gamma + rH - rH \\ &= D_\gamma \\ &= abP \end{aligned}$$

Analysis: We now assess the probability of success of the challenger $\mathcal{C}$. The events in which $\mathcal{C}$ aborts the EUF-ICLSC-TKEM-CMA-I game are E_1, E_2 and E_3. (Events E_1, E_2 and E_3 are same as in IND-ICLSC-TKEM-CCA2-I proof). The probability that, $\mathcal{C}$ does not abort the EUF-ICLSC-TKEM-CMA-I game is given by

$$(Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3]) = \left(1 - \frac{q_{ppk}}{q_{H_1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{ppk} + q_{fpk})}\right)$$

The probability of $\mathcal{F}_I$ guessing the hash values corresponding to H_3 and H_4 oracles is $\frac{1}{2^\kappa} + \frac{1}{2^\kappa} = \frac{1}{2^{\kappa-1}}$. The probability with which $\mathcal{C}$ solves the CDH problem is

$$\epsilon' \geq \left(\epsilon - \frac{1}{2^{\kappa-1}}\right)\left(1 - \frac{q_{ppk}}{q_{H_1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{ppk} + q_{fpk})}\right)$$

Since ϵ is non-negligible, the probability of $\mathcal{C}$ solving CDH problem is also non-negligible. $\square$

6.4 Type-II Unforgeability

Theorem 4. *If there exists a forger $\mathcal{F}_I$ with an advantage ϵ against the EUF-ICLSC-TKEM-CMA-I security of the ICLSC-TKEM scheme, asking q_{H_i} (i = 1, 2, 3, 4) hash queries to random oracles $\mathcal{O}_{H_i}$ (i = 1, 2, 3, 4), q_{fpk} full private key extract queries and q_{fpk} replace public key queries, then there exist an algorithm $\mathcal{C}$ that solves the CDH problem with an advantage*

$$\epsilon' \geq \left(\epsilon - \frac{1}{2^{\kappa-1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(1 - \frac{q_{rpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{fpk} + q_{rpk})}\right)$$

Proof: A challenger $\mathcal{C}$ is challenged with an instance of the CDH problem say $\langle P, aP, bP \rangle \in \mathbb{G}_1$. Let $\mathcal{F}_{II}$ be a forger who is capable of breaking the EUF-ICLSC-TKEM-CMA-II security of the ICLSC-TKEM scheme. $\mathcal{C}$ can make use of $\mathcal{F}_{II}$ to compute the solution abP of the CDH instance by playing the following interactive game with $\mathcal{F}_{II}$.

Setup: $\mathcal{C}$ chooses $s \in_R \mathbb{Z}_q^*$ and sets the master public key $P_{pub} = sP$, designs the hash functions H_i (i =1 *to* 4) as random oracles $\mathcal{O}_{H_i}$ (i =1 *to* 4) respectively. In order to maintain the consistency between the responses to the hash queries, $\mathcal{C}$ maintains lists L_i (i =1 *to* 4) and to maintain the list of issued private keys and public keys, $\mathcal{C}$ maintains a list L_K. $\mathcal{C}$ gives the public parameters *params* and the master private key s to $\mathcal{F}_{II}$.

Training Phase: $\mathcal{F}_{II}$ performs a series of polynomially bounded number of queries in an adaptive fashion in this phase. The oracles and queries allowed are described below.

- All the oracles are identical to that of the $\mathcal{O}_{H_1}$, $\mathcal{O}_{H_2}$, $\mathcal{O}_{H_3}$, $\mathcal{O}_{H_4}$, $\mathcal{O}_{ExtractPartialPrivateKey}$, $\mathcal{O}_{Request\ PublicKey}$, $\mathcal{O}_{ExtractPrivateKey}$, $\mathcal{O}_{ReplacePublicKey}$, $\mathcal{O}_{SymmetricKeyGeneration}$, $\mathcal{O}_{KeyEncapsulation}$ and $\mathcal{O}_{KeyDecapsulation}$ oracles in the IND-ICLSC-TKEM-CCA2-II game.

Forgery: At the end of the *Training Phase* (which is decided by $\mathcal{F}_{II}$), $\mathcal{F}_{II}$ sends to $\mathcal{C}$ an encapsulation $\langle \tau^*, \psi^* = \langle U, W \rangle, ID_\gamma, ID_B \rangle$, where ID_γ is the sender identity and ID_B is the receiver identity. It is to be noted that the public key of the sender ID_γ was not replaced during the **Training Phase**. In addition, ψ^* should not be the response for any key encapsulation queries by $\mathcal{F}_I$ during the *Training Phase*. If ψ^* is generated with the above restrictions, then $\mathcal{C}$ can obtain the solution for the CDH instance by performing the following steps.

- Retrieves the tuple $\langle U, \tau^*, T, ID_\gamma, PK_\gamma, ID_B, PK_B, h_i, H = h_i P \rangle$ from the list L_3 and $\langle U, \tau^*, T, ID_\gamma, PK_\gamma, ID_B, PK_B, h_i, H' = h'_i P \rangle$ from the list L_4.
- Output $h_i^{'-1}(W - sb_\gamma P - h_i U) = abP$ as the solution to the CDH problem instance.

Correctness:

$$\begin{aligned} h_i^{'-1}(W - sb_\gamma P - h_i U) &= h_i^{'-1}(D_\gamma + rH + ah'_i bP - sb_\gamma P - h_i U) \\ &= h_i^{'-1}(b_\gamma sP + rh_i P + ah'_i bP - sb_\gamma P - rh_i P) \\ &= h_i^{'-1}(ah'_i bP) \\ &= abP \end{aligned}$$

Analysis: We now assess the probability of success of the challenger $\mathcal{C}$. The events in which $\mathcal{C}$ aborts the EUF-ICLSC-TKEM-CMA-II game are E_1, E_2 and E_3. (Events E_1, E_2 and E_3 are same as in IND-ICLSC-TKEM-CCA2-II proof). The probability that, $\mathcal{C}$ does not abort the EUF-ICLSC-TKEM-CMA-II game is given by

$$(Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3]) = \left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(1 - \frac{q_{rpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{fpk} + q_{rpk})}\right)$$

The probability of $\mathcal{F}_{II}$ guessing the hash values corresponding to H_3 and H_4 oracles is $\frac{1}{2^\kappa} + \frac{1}{2^\kappa} = \frac{1}{2^{\kappa-1}}$. The probability with which $\mathcal{C}$ solves the CDH problem is

$$\epsilon' \geq \left(\epsilon - \frac{1}{2^{\kappa-1}}\right)\left(1 - \frac{q_{fpk}}{q_{H_1}}\right)\left(1 - \frac{q_{rpk}}{q_{H_1}}\right)\left(\frac{1}{q_{H_1} - (q_{fpk} + q_{rpk})}\right)$$

Since ϵ is non-negligible, the probability of $\mathcal{C}$ solving CDH problem is also non-negligible. □

7 Conclusion

In this paper, we have showed that the only existing CL-KEM [13] proved in the standard model is not Type-I CCA secure and the only existing certificateless hybrid signcryption scheme of Fagen Li et al.'s [15] is existentially forgeable with respect to both Type-I and Type-II forgers. We have also proposed an improved certificateless hybrid signcryption scheme with the proper binding, that provides adequate security to the scheme. We have proved the improved certificateless hybrid signcryption scheme in the random oracle model.

References

1. Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In *Advances in Cryptology - ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 452–473. Springer, 2003.
2. Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002.
3. Joonsang Baek, Ron Steinfeld, and Yuliang Zheng. Formal proofs for the security of signcryption. In *Public Key Cryptography - PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 80–98. Springer, 2002.
4. Manuel Barbosa and Pooya Farshim. Certificateless signcryption. In *ACM Symposium on Information, Computer and Communications Security - ASIACCS 2008*, pages 369–372. ACM, 2008.
5. Paulo S. L. M. Barreto, Benoît Libert, Noel McCullagh, and Jean-Jacques Quisquater. Efficient and provably-secure identity-based signatures and signcryption from bilinear maps. In *Advances in Cryptology - ASIACRYPT 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 515–532. Springer, 2005.
6. Kamel Bentahar, Pooya Farshim, John Malone-Lee, and Nigel P. Smart. Generic constructions of identity-based and certificateless kems. *J. Cryptology*, 21(2):178–199, 2008.
7. Xavier Boyen. Multipurpose identity-based signcryption (a swiss army knife for identity-based cryptography). In *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2003.
8. Liqun Chen and John Malone-Lee. Improved identity-based signcryption. In *Public Key Cryptography - PKC 2005*, volume 3386 of *Lecture Notes in Computer Science*, pages 362–379. Springer, 2005.
9. Sherman S. M. Chow, Siu-Ming Yiu, Lucas Chi Kwong Hui, and K. P. Chow. Efficient forward and provably secure id-based signcryption scheme with public verifiability and public ciphertext authenticity. In *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 352–369. Springer, 2003.
10. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
11. Alexander W. Dent. Hybrid signcryption schemes with insider security. In *Information Security and Privacy - ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 253–266. Springer, 2005.
12. Yevgeniy Dodis, Michael J. Freedman, Stanislaw Jarecki, and Shabsi Walfish. Versatile padding schemes for joint signature and encryption. In *ACM Conference on Computer and Communications Security - CCS 2004*, pages 344–353. ACM, 2004.
13. Juan González Nieto Georg Lippold, Colin Boyd. Efficient certificateless kem in the standard model. Cryptology ePrint Archive, Report 2009/451 (Extended abstract of the paper accepted in ICISC-09), 2009. http://eprint.iacr.org/.
14. Qiong Huang and Duncan S. Wong. Generic certificateless key encapsulation mechanism. In *ACISP 2007*, volume 4586 of *Lecture Notes in Computer Science*, pages 215–229. Springer, 2007.
15. Fagen Li, Masaaki Shirase, and Tsuyoshi Takagi. Certificateless hybrid signcryption. In *Information Security Practice and Experience - ISPEC 2009*, volume 5451 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2009.
16. Benoît Libert and Jean-Jacques Quisquater. Efficient signcryption with key privacy from gap diffie-hellman groups. In *Public Key Cryptography - PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 187–200. Springer, 2004.
17. Benot Libert and Jean-Jacques Quisquater. A new identity based signcryption scheme from pairings. In *In IEEE Information Theory Workshop*, pages 155–158, 2003.
18. John Malone-Lee. Identity-based signcryption. Cryptology ePrint Archive, Report 2002/098, 2002.
19. John Malone-Lee and Wenbo Mao. Two birds one stone: Signcryption using rsa. In *Topics in Cryptology - CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2003.
20. S. Sharmila Deva Selvi, S. Sree Vivek, Deepanshu Shukla, and C. Pandu Rangan. Efficient and provably secure certificateless multi-receiver signcryption. In *Provable Security, ProvSec - 2008*, volume 5324 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2008.
21. Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology, CRYPTO - 1984*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.
22. Victor Shoup. Oaep reconsidered. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259. Springer, 2001.
23. T.E.Bj∅rstad. Provable security of signcryption. In *Masters Thesis*, http://www.nwo.no/tor/pdf/mscthesis.pdf. Norwegian University of Technology and Science, 2005.
24. Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.
25. Yuliang Zheng. Digital signcryption or how to achieve cost(signature & encryption) $<<$ cost(signature) + cost(encryption). In *Advances in Cryptology, CRYPTO - 1997*, volume 1294 of *Lecture Notes in Computer Science*, pages 165–179. Springer, 1997.