Breaking and Re-Building a Certificateless Hybrid Signcryption Scheme

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 counterparts in many cryptographic applications. Both these functionalities are efficiently achieved simultaneously by the cryptographic primitive called signcryption. On the other hand hybrid encryption (KEM-DEM) provides an efficient and practical way to securely communicate very large messages. Recently, the first certificateless hybrid signcryption scheme was proposed by Fagen Li et al.. The concept of certificateless hybrid signcryption evolved by combining the ideas of signcryption based on tag-KEM and certificateless cryptography. Fagen Li et al. claimed that their scheme is secure against adaptive chosen ciphertext attack and it is existentially unforgeable. In this paper, we show that their scheme is existentially forgeable and also provide an improved certificateless hybrid signcryption scheme. We formal 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

Simultaneous confidentiality and authentication of messages are often required in secure and authentic message transmission over an insecure channel like internet. Signcryption is the cryptographic primitive that offers both these properties concurrently with a very low cost when compared to encrypting and signing a message independently. Zheng [21] introduced the concept of signcryption in 1997, subsequently signcryption, which was considered to be an important and useful primitive for secure message transmission and got the attention of crypto researchers. As a result many signcryption schemes were proposed till date, [15, 14, 8, 6, 13, 5, 7, 4, 17] to name a few. Zheng's [21] scheme was not proven to be secure. Baek et al. in [3] gave the formal security model for signcryption and proved the security of [21] in the model.

In 1984, Shamir [18] 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 an 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) based systems because no certificate is needed to validate the public key of an user. Identity based cryptosystem makes use of a trusted third party the private key generator (PKG) who is in procession 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] inorder 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. 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 an user selected secret value. An effective signcryption or unsigncryption 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

* 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

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).

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 [16] and Dodis et al.[11]. 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 [10] and Bjrstad [20]. 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 [9] and [19].

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 q. 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 related to bilinear maps that are relevant to the protocol we discuss.

Computation Diffie-Hellman Problem (CDHP) 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.

Decisional Bilinear Diffie-Hellman Problem (DBDHP) Given $(P, aP, bP, cP, \alpha) \in \mathbb{G}_1^4 \times \mathbb{G}_2$ for unknown $a, b, c \in \mathbb{Z}_q^*$, the DBDH problem in $\mathbb{G}_1$ is to decide if $\alpha = \hat{e}(P, P)^{abc}$.

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

$$Adv_{\mathcal{A}}^{DBDH} = |Pr\left[\mathcal{A}(P, aP, bP, cP, \hat{e}(P, P)^{abc}) = 1\right] - Pr\left[\mathcal{A}(P, aP, bP, cP, \alpha) = 1\right]|$$

The *DBDH Assumption* is that, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage $Adv_{\mathcal{A}}^{DBDH}$ 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 an 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 an 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.
- ***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.
- ***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 to 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 (IND-CLSC-TKEM-CCA2) and existential unforgeability against adaptive chosen message and identity attacks (EUF-CLSC-TKEM-CMA). 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-CCA-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 can request 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 for an user 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 obtained from the corresponding list maintained by $\mathcal{C}$. $\mathcal{C}$ 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 $\text{Sym}(ID_A, PK_A, S_A, ID_B, PK_B)$ 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 $b \in_R \{0, 1\}$ and sends K_b to $\mathcal{A}_I$. When $\mathcal{A}_I$ receives K_b, 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 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: $\mathcal{A}_I$ outputs a bit b' and wins the game if $b' = b$.

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

IND-CLSC-TKEM-CCA-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-CCA-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 $\text{Sym}(ID_A, PK_A, S_A, ID_B, PK_B)$ 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 $b \in_R \{0, 1\}$ and sends K_b to $\mathcal{A}_{II}$. When $\mathcal{A}_{II}$ receives K_b, 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 $Adv^{IND-CLSC-TKEM-CCA2-II}(\mathcal{A}_{II}) = |2Pr[b' = b] - 1|$, where $Pr[b' = b]$ denotes the probability that $b' = b$.

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 the challenger $\mathcal{C}$ 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 be 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 Fagen Li et al.'s CLSC-TKEM

In this section we review the CLSC-TKEM scheme of Fagen Li et al, presented in [12]. We also show that [12] does not provide both confidentiality and unforgeability.

3.1 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 $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$ and a generator $P \in_R \mathbb{G}_1$. It then chooses a master private key $s \in_R \mathbb{Z}_q^*$, a sets a system-wide public key $P_{pub} = sP$ and chooses four cryptographic hash functions defined as $H_1 : \{0,1\}^* \to \mathbb{G}_1$, $H_2 : \{0,1\}^* \to \{0,1\}^n$, $H_3 : \{0,1\}^* \to \mathbb{G}_1$ and $H_4 : \{0,1\}^* \to \mathbb{G}_1$. 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 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:*** An 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 an 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 $\omega = (r, U, ID_A, PK_A, S_A, ID_B, PK_B)$
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 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*.

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

We launch an attack on the scheme to show the weakness in unforgeability of Fagen Li et al.'s [12] CLSC-TKEM.

Attack on Unforgeability: Fagen Li et al. [12] 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. 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 and are not allowed to extract the full private keys of the sender A inorder to ensure insider security. The weakness of the system is observed due to this constraint.

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 $\mathcal{C}$ also as mentioned above $\mathcal{F}_I$ 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 performs the following steps:

- Let ID_{B^*} be an user for which $\mathcal{F}_I$ knows the full private key S_{B^*}.
- $\mathcal{F}_I$ 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.

- The computation of $H = H_3(U, \tau, ID_A, PK_A)$, $H' = H_4(U, \tau, ID_A, PK_A)$ will output the same value because it depends only upon the sender identity and public key.
- 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.

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. The forgery can be done in a similar way as described in ***Attack by Type-I forger*** $\mathcal{F}_I$.

4 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 easily countered by making the following changes in the ***Sym***, ***Encap*** and ***Decap*** algorithms in Fagen Li et al's [12] 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 $W = D_S + rH + x_A H'$.
- 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*.

5 Security of the Improved ICLSC-TKEM Scheme

In this section we provide the formal proof for the unforgeability of the improved CLSC-TKEM. The proof for confidentiality will be appended soon.

5.1 Type-I Unforgeability

Theorem 1. *The improved certificateless signcryption scheme ICLSC-TKEM is EUF-ICLSC-TKEM-CMA-I secure in the random oracle model, if the CDH problem is intractable in* $\mathbb{G}_1$.

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. Inorder to maintain the consistancy 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.

- $\mathcal{O}_{H_1}(ID_i)$: To respond to this query, $\mathcal{C}$ checks whether a tuple $\langle ID_i, b_iP, coin, b_i\rangle$ already exists in the list L_1. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding b_iP. Otherwise, $\mathcal{C}$ chooses a random coin $coin_R\{0,1\}$, chooses $b_i \in_R \mathbb{Z}_q^*$, if $coin = 1$ adds the tuple $\langle ID_i, Q_i = b_iP, coin = 1, b_i\rangle$ to the list L_1 else adds $\langle ID_i, Q_i = b_ibP, coin = 0, b_i\rangle$ and returns Q_i to $\mathcal{F}_I$.
- $\mathcal{O}_{H_2}(U, T, r(PK_B), ID_B, PK_B)$: 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 $\mathcal{F}_I$ else chooses $K \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, T, r(PK_B), ID_B, PK_B, K\rangle$ to the list L_2 and returns K to $\mathcal{F}_I$.
- $\mathcal{O}_{H_3}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l, H\rangle$ exists in the list L_3. If so, returns H to $\mathcal{F}_I$ else chooses $l \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l, H = lP\rangle$ to the list L_3 and returns H to $\mathcal{F}_I$.
- $\mathcal{O}_{H_4}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l', H'\rangle$ exists in the list L_4. If so, returns H' to $\mathcal{F}_I$ else chooses $l' \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l', H' = l'P\rangle$ to the list L_4 and returns H' to $\mathcal{F}_I$.
- $\mathcal{O}_{ExtractPartialPrivateKey}$: $\mathcal{F}_I$ chooses an identity ID_i and gives it to $\mathcal{C}$. Now, $\mathcal{C}$ checks whether the value $coin = 1$ for ID_i in list L_1, if so searches for a tuple of the form $\langle ID_i, x_i, D_i, PK_i\rangle$ in the list L_K. If it is present then $\mathcal{C}$ responds with the corresponding D_i value else, retrieves the corresponding b_i value from the list L_1, computes $D_i = b_iaP$, updates the list L_K with the tuple $\langle ID_i, -, D_i, -\rangle$ and sends D_i to $\mathcal{F}_I$. If $coin = 0$, $\mathcal{C}$ aborts the game.
- $\mathcal{O}_{RequestPublicKey}$: $\mathcal{F}_I$ 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 entry of the type $\langle ID_i, x_i, D_i, PK_i\rangle$. If an entry exists, then responds by returning the corresponding public key PK_i to $\mathcal{F}_I$. If it does not exist, $\mathcal{C}$ chooses $x_i \in_R \mathbb{Z}_q^*$, sets $PK_i = x_iP$, computes D_i, adds the tuple $\langle ID_i, x_i, D_i, PK_i\rangle$ to the list L_K and sends the corresponding PK_i to $\mathcal{F}_I$.
- $\mathcal{O}_{ExtractPrivateKey}$: $\mathcal{F}_I$ produces an identity ID_i and requests the corresponding full private key. If ID_i's public key has not been replaced and the entry for $coin = 1$, corresponding to ID_i in the list L_1 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 $\mathcal{F}_I$ has already replaced ID_i's public key or the correspondinf value of $coin = 0$, then $\mathcal{C}$ does not provide the corresponding private key to $\mathcal{F}_I$.
- $\mathcal{O}_{ReplacePublicKey}$: Inorder to replace the public key PK_i of an user ID_i with any value PK_i' of $\mathcal{F}_I$'s choice, $\mathcal{C}$ updates the corresponding tuple in the list L_K as $\langle ID_i, -, S_i, PK_i'\rangle$. The current value of the user's public key is used by $\mathcal{C}$ in for computations or responses to any queries made by $\mathcal{F}_I$.
- $\mathcal{O}_{SymmetricKeyGeneration}$: $\mathcal{F}_I$ produces a sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. If the value of $coin = 1$ in list L_1 for the identity ID_A, the private key S_A of the sender is obtained from the list L_K and the algorithm works in the normal way. If $coin = 0$, $\mathcal{C}$ computes the symmetric key K and an internal state information ω, stores and

keeps ω secret from the view of $\mathcal{F}_I$ and sends the symmetric key K to $\mathcal{F}_I$. 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 $\mathcal{F}_I$ provides the secret value of ID_A to $\mathcal{C}$.

- $\mathcal{O}_{KeyEncapsulation}$: $\mathcal{F}_I$ 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 private key of the sender S_A is obtained from the list L_K. $\mathcal{C}$ checks whether a corresponding ω value is stored previously.
  - If ω exists and the value of $coin = 1$, corresponding to ID_A in the list L_1, then $\mathcal{C}$ computes the encapsulation ψ with ω and τ as per the actual encapsulation algorithm, and deletes ω.
  - If ω exists and the value of $coin = 0$, corresponding to ID_A in the list L_1, then $\mathcal{C}$ performs the following to compute ψ:
    * Chooses $r \in_R \mathcal{Z}_q^*$ and computes $U = raP$.
    * Sets $H = -r^{-1}b_AbP$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, r, H = -r^{-1}b_AbP\rangle$ to the list L_3, where b_AbP is retrieved from the list L_1.
    * Sets $H' = l'P$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l', H' = l'P\rangle$ to the list L_3.
    * Computes $W = l'x_AP$ (This is possible $\mathcal{C}$ knows the public key PK_A of the sender A which is x_AP).
    * Outputs, $\psi = \langle U, W\rangle$ as the encapsulation.

    We show that, $\psi = \langle U, W\rangle$ passes the verification done inorder to validate the encapsulation by $\mathcal{F}_I$ 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:

$$\begin{aligned}\hat{e}(P_{pub}, Q_A)\hat{e}(U, H)\hat{e}(PK_A, H') &= \hat{e}(aP, b_AbP)\hat{e}(raP, -r^{-1}b_AbP)\hat{e}(x_AP, l'P)\\ &= \hat{e}(aP, b_AbP)\hat{e}(raP, r^{-1}b_AbP)^{-1}\hat{e}(x_AP, l'P)\\ &= \hat{e}(P, b_AabP)\hat{e}(P, b_AabP)^{-1}\hat{e}(P, x_Al'P)\\ &= \hat{e}(P, x_Al'P)\\ &= \hat{e}(P, W)\end{aligned}$$

  - Else, if ω doesnot exist, $\mathcal{C}$ returns *invalid*.

- $\mathcal{O}_{KeyDecapsulation}$: $\mathcal{F}_I$ 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 $\mathcal{F}_I$ provides the private key of ID_B to $\mathcal{C}$.
  - $\mathcal{C}$ returns the key K by computing it as per the Decap algorithm if the value of $coin = 1$ corresponding to ID_B in the list L_1.
  - If the value of $coin = 0$, corresponding to ID_B in the list L_1, $\mathcal{C}$ computes K from ψ as follows:
    * Searches in the list L_3 and L_4 for entries of the type $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l, H\rangle$ and $\langle U, T, \tau, ID_A, PK_A, ID_B, PK_B, l', H'\rangle$ 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 T value 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 $\langle U, T, x_BU, ID_B, PK_B, K\rangle$ exists in the list L_2. If it exists output the corresponding K value as the decapsulation of ψ.

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*. If ψ^* is generated with the above restrictions, then $\mathcal{C}$ can obtain the solution for the CDH instance by performing the following steps.

–

–

5.2 Type-II Unforgeability

Theorem 2. *The improved certificateless signcryption scheme ICLSC-TKEM is EUF-ICLSC-TKEM-CMA-II secure in the random oracle model, if the CDH problem is intractable in* $\mathbb{G}_1$.

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. Inorder to maintain the consistancy 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.

- $\mathcal{O}_{H_1}(ID_i)$: To respond to this query, $\mathcal{C}$ checks whether a tuple $\langle ID_i, a_iP, coin, a_i\rangle$ already exists in the list L_1. If a tuple of this form exists, $\mathcal{C}$ returns the corresponding a_iP. Otherwise, $\mathcal{C}$ chooses a random coin $coin_R\{0,1\}$, chooses $a_i \in_R \mathbb{Z}_q^*$, if $coin = 1$ adds the tuple $\langle ID_i, Q_i = a_iP, coin = 1, a_i\rangle$ to the list L_1 else adds $\langle ID_i, Q_i = a_iaP, coin = 0, a_i\rangle$ and returns Q_i to $\mathcal{F}_{II}$.
- $\mathcal{O}_{H_2}(U, T, r(PK_B), ID_B, PK_B)$: Identical to that of $\mathcal{O}_{H_2}$ query in the EUF-ICLSC-TKEM-CMA-I game.
- $\mathcal{O}_{H_3}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: Identical to that of $\mathcal{O}_{H_3}$ query in the EUF-ICLSC-TKEM-CMA-I game.
- $\mathcal{O}_{H_4}(U, \tau, T, ID_A, PK_A, ID_B, PK_B)$: To respond to this query, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l', H'\rangle$ exists in the list L_4. If so, returns H' to $\mathcal{F}_{II}$ else, chooses $l' \in_R \mathbb{Z}_q^*$, adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l', H' = l'P\rangle$ to the list L_4 and returns H' to $\mathcal{F}_I$.
- $\mathcal{O}_{ExtractPartialPrivateKey}$: $\mathcal{F}_I$ chooses an identity ID_i and gives it to $\mathcal{C}$. Now, $\mathcal{C}$ checks whether the value $coin = 1$ for ID_i in list L_1, if so searches for a tuple of the form $\langle ID_i, x_i, D_i, PK_i\rangle$ in the list L_K. If it is present then $\mathcal{C}$ responds with the corresponding D_i value else, retrieves the corresponding b_i value from the list L_1, computes $D_i = b_iaP$, updates the list L_K with the tuple $\langle ID_i, -, D_i, -\rangle$ and sends D_i to $\mathcal{F}_I$. If $coin = 0$, $\mathcal{C}$ aborts the game.
- $\mathcal{O}_{RequestPublicKey}$: $\mathcal{F}_I$ 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 entry of the type $\langle ID_i, x_i, D_i, PK_i\rangle$. If an entry exists, then responds by returning the corresponding public key PK_i to $\mathcal{F}_I$. If it does not exist, $\mathcal{C}$ chooses $x_i \in_R \mathbb{Z}_q^*$, sets $PK_i = x_iP$, computes D_i, adds the tuple $\langle ID_i, x_i, D_i, PK_i\rangle$ to the list L_K and sends the corresponding PK_i to $\mathcal{F}_I$.
- $\mathcal{O}_{ExtractPrivateKey}$: $\mathcal{F}_I$ produces an identity ID_i and requests the corresponding full private key. If ID_i's public key has not been replaced and the entry for $coin = 1$, corresponding to ID_i in the list L_1 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 $\mathcal{F}_I$ has already replaced ID_i's public key or the correspondinf value of $coin = 0$, then $\mathcal{C}$ does not provide the corresponding private key to $\mathcal{F}_I$.
- $\mathcal{O}_{ReplacePublicKey}$: Inorder to replace the public key PK_i of an user ID_i with any value PK_i' of $\mathcal{F}_I$'s choice, $\mathcal{C}$ updates the corresponding tuple in the list L_K as $\langle ID_i, -, S_i, PK_i'\rangle$. The current value of the user's public key is used by $\mathcal{C}$ in for computations or responses to any queries made by $\mathcal{F}_I$.
- $\mathcal{O}_{SymmetricKeyGeneration}$: $\mathcal{F}_I$ produces a sender's identity ID_A, public key PK_A, the receiver's identity ID_B and public key PK_B to $\mathcal{C}$. If the value of $coin = 1$ in list L_1 for the identity ID_A, the private key S_A of the sender is obtained from the list L_K and the algorithm works in the normal way. If $coin = 0$, $\mathcal{C}$ computes the symmetric key K and an internal state information ω, stores and

keeps ω secret from the view of $\mathcal{F}_I$ and sends the symmetric key K to $\mathcal{F}_I$. 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 $\mathcal{F}_I$ provides the secret value of ID_A to $\mathcal{C}$.

- $\mathcal{O}_{KeyEncapsulation}$: $\mathcal{F}_I$ 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 private key of the sender S_A is obtained from the list L_K. $\mathcal{C}$ checks whether a corresponding ω value is stored previously.
  - If ω exists and the value of $coin = 1$, corresponding to ID_A in the list L_1, then $\mathcal{C}$ computes the encapsulation ψ with ω and τ as per the actual encapsulation algorithm, and deletes ω.
  - If ω exists and the value of $coin = 0$, corresponding to ID_A in the list L_1, then $\mathcal{C}$ performs the following to compute ψ:
    * Chooses $r \in_R \mathcal{Z}_q^*$ and computes $U = raP$.
    * Sets $H = -r^{-1}b_AbP$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, r, H = -r^{-1}b_AbP\rangle$ to the list L_3, where b_AbP is retrieved from the list L_1.
    * Sets $H' = l'P$ and adds the tuple $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l', H' = l'P\rangle$ to the list L_3.
    * Computes $W = l'x_AP$ (This is possible $\mathcal{C}$ knows the public key PK_A of the sender A which is x_AP).
    * Outputs, $\psi = \langle U, W\rangle$ as the encapsulation.

    We show that, $\psi = \langle U, W\rangle$ passes the verification done inorder to validate the encapsulation by $\mathcal{F}_I$ 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:

$$\begin{aligned}\hat{e}(P_{pub}, Q_A)\hat{e}(U, H)\hat{e}(PK_A, H') &= \hat{e}(aP, b_AbP)\hat{e}(raP, -r^{-1}b_AbP)\hat{e}(x_AP, l'P)\\ &= \hat{e}(aP, b_AbP)\hat{e}(raP, r^{-1}b_AbP)^{-1}\hat{e}(x_AP, l'P)\\ &= \hat{e}(P, b_AabP)\hat{e}(P, b_AabP)^{-1}\hat{e}(P, x_Al'P)\\ &= \hat{e}(P, x_Al'P)\\ &= \hat{e}(P, W)\end{aligned}$$

  - Else, if ω doesnot exist, $\mathcal{C}$ returns *invalid*.

- $\mathcal{O}_{KeyDecapsulation}$: $\mathcal{F}_I$ 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 $\mathcal{F}_I$ provides the private key of ID_B to $\mathcal{C}$.
  - $\mathcal{C}$ returns the key K by computing it as per the Decap algorithm if the value of $coin = 1$ corresponding to ID_B in the list L_1.
  - If the value of $coin = 0$, corresponding to ID_B in the list L_1, $\mathcal{C}$ computes K from ψ as follows:
    * Searches in the list L_3 and L_4 for entries of the type $\langle U, \tau, T, ID_A, PK_A, ID_B, PK_B, l, H\rangle$ and $\langle U, T, \tau, ID_A, PK_A, ID_B, PK_B, l', H'\rangle$ 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 T value 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 $\langle U, T, x_BU, ID_B, PK_B, K\rangle$ exists in the list L_2. If it exists output the corresponding K value as the decapsulation of ψ.

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*. If ψ^* is generated with the above restrictions, then $\mathcal{C}$ can obtain the solution for the CDH instance by performing the following steps.

–

–

6 Conclusion

In this paper, we have cryptanalized the certificateless hybrid signcryption scheme of Fagen Li et al.'s [12]. We have showed attacks on unforgeability of the scheme by both Type-I and Type-II forgers. We have also provided a possible fix for Fagen Li et al.'s scheme with the proper binding, that provides adequate security to the scheme. The proof of the improved scheme follows from the security proofs by Fagen Li et al.

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. 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.
7. Liqun Chen and John Malone-Lee. Improved identity-based signcryption. volume 3386 of *Lecture Notes in Computer Science*, pages 362–379. Springer, 2005.
8. 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.
9. 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.
10. Alexander W. Dent. Hybrid signcryption schemes with insider security. In *ACISP*, volume 3574 of *Lecture Notes in Computer Science*, pages 253–266. Springer, 2005.
11. 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.
12. Fagen Li, Masaaki Shirase, and Tsuyoshi Takagi. Certificateless hybrid signcryption. In *ISPEC*, volume 5451 of *Lecture Notes in Computer Science*, pages 112–123. Springer, 2009.
13. 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.
14. Benot Libert and Jean-Jacques Quisquater. A new identity based signcryption scheme from pairings. In *In IEEE Information Theory Workshop*, pages 155–158, 2003.
15. John Malone-Lee. Identity-based signcryption. Cryptology ePrint Archive, Report 2002/098, 2002.
16. John Malone-Lee and Wenbo Mao. Two birds one stone: Signcryption using rsa. In *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 211–225. Springer, 2003.
17. S. Sharmila Deva Selvi, S. Sree Vivek, Deepanshu Shukla, and C. Pandu Rangan. Efficient and provably secure certificateless multi-receiver signcryption. In *ProvSec*, volume 5324 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2008.
18. 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.
19. Victor Shoup. Oaep reconsidered. In *CRYPTO*, volume 2139 of *Lecture Notes in Computer Science*, pages 239–259. Springer, 2001.
20. T.E.Bjrstad. Provable security of signcryption. In *Masters Thesis*, http://www.nwo.no/tor/pdf/mscthesis.pdf. Norwegian University of Technology and Science, 2005.
21. 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.