**CERIAS Tech Report 2013-10**

**Elliptic Curve Cryptography based Certificateless Hybrid Signcryption Scheme without Pairing**

by Seung-Hyun Seo and Elisa Bertino
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

# Elliptic Curve Cryptography based Certificateless Hybrid Signcryption Scheme without Pairing

Seung-Hyun Seo and Elisa Bertino

*Dept. of Computer Science, Purdue University,*
*West Lafayette, IN 47907 US*
*seo29@purdue.edu, bertino@purdue.edu*

## 1 Introduction

Signcryption is a scheme that provides confidentiality and authentication while keeping costs low in comparison to independent encryption and message signing. Since Zheng [13] introduced the concept of signcryption, a variety of schemes have been presented in [6–11]. We can divide the schemes in two ways to construct the signcryption scheme such as a public signcryption and a hybrid signcryption. In the public signcryption scheme, the process of encryption and signing are performed utilizing the public key operation. However, in the hybrid signcryption scheme, only the signing process uses the public key operation while the symmetric key setting is used for the encryption. That is, we can construct the hybrid signcryption scheme by combining two methods: (1) an asymmetric part, takes a private and a public key as the input and outputs a suitably sized random symmetric key and then performs an encapsulation of the key, (2) the symmetric part takes a message and a symmetric key as the input and outputs an authenticated encryption of the message. Thus, a hybrid signcryption approach can efficiently encapsulate new keys and securely transmit data for various applications such as Advanced Metering Infrastructures (AMIs) and Wireless Sensor Networks (WSNs).

The hybrid signcryption scheme has been proposed in [1–5] and its formal security model was presented in [2] However, since these approaches rely on traditional PKI using a certificate trusted by CA, they require the management of certificates. Although Identity-based Public Key Cryptography (ID-PKC)[16] was introduced to eliminate the dependency from explicit certificates, it suffers from a key escrow problem because the Key Generation Center (KGC) stores the private keys of all users. In order to resolve these drawbacks, Al-Riyami et al. [12] introduced certificateless public key cryptography (CL-PKC), that splits the user's private key into two parts: one is a partial private key generator by the KGC, and the other one is a secret value selected by the user. CL-PKC is able to overcome the key escrow problem because the KGC is unable to access the user's secret value. Only when a valid user holds both the partial private key and the secret value, the cryptographic operations such as decryption or digital signing based on CL-PKC can be performed.

Recently, Li et al.[15] first constructed a hybrid signcryption scheme which was truly certificateless by using certificateless signcryption tag-KEM and a DEM. The concept of certificateless hybrid signcryption evolved by combining the ideas of signcryption based on tag-KEM and certificateless cryptography. Li et al.[15] claimed that their scheme is secure against adaptive chosen ciphertext attack and it is existentially unforgeable. However, such scheme is existentially forgeable and the definition of the generic scheme is insufficient. Selvi et al.[14] showed the security weaknesses of Li et al.[15]'s scheme and presented an improved certificateless hybrid signcryption scheme. However, Li et al. [15] and Selvi et al. [14] used a scheme based on bilinear pairings. In spite of the recent advances in implementation techniques, the computational cost required for pairing operation is still considerably higher in comparison to standard operations such as ECC point multiplication. For example, TinyTate, which uses TinyECC as the underlying library, takes around 31s to compute one pairing operation on the MICAz(8MHz) mote. NanoECC, which uses the MIRACL library, takes around 17.93s to compute one pairing operation and around 1.27s to compute one ECC point multiplication on the MICA2(8MHz) mote [20]. Thus, such schemes based on pairing operations are not applicable to security mechanisms for AMIs and WSNs. In this technical report, we propose a certificateless hybrid signcryption (CL-HSC) scheme without pairing operations. We present the formal security model of our CL-HSC scheme. Then, we provide the security proof of our CL-HSC scheme against both adaptive chosen ciphertext attack and existential forgery in the appropriate security models for certificateless hybrid signcryption. Since our CL-HSC scheme does not depend on the pairing-based operation, it reduces the computational overhead. It is also adopted to utilize ECC (Elliptic Curve Cryptography). Thus, we take the benefit of ECC keys defined on an additive group with a 160-bit length as secure as the RSA keys with 1024-bit length.

The remainder of this report is organized as follows: In Section 2, we briefly describe the overview of elliptic curve cryptography and computational assumptions. In Section 3, we provide the definition of CL-HSC and security models for CL-HSC. In Section 4, we introduced out ECC based CL-HSC scheme without pairing operations. In Section 5, we provide formal security proof of our scheme, and conclude in Section 6.

## 2 Preliminaries

### 2.1 Elliptic Curve Cryptography

The ECC was proposed by Miller [17] and Koblitz [18], and its security is based on the difficulty of solving the ECDLP. Any cryptosystem based on ECC provides high security with small key size, for example, a 160-bit ECC is considered to be as secured as 1024-bit RSA key [19]. Let $F_q$ be the field of integers of modulo a large prime number q. A non-singular elliptic curve $E_q(a, b)$ over $F_q$ is defined by the following equation:

$$y^2 \bmod q = (x^3 + ax + b) \bmod q \ (1),$$

where $a, b, x, y \in F_q$ and $\triangle = (4a^3 + 27b^2) \bmod q \neq 0$. A point $P(x, y)$ is an elliptic curve point if it satisfies Equation (1), and the point $Q(x, -y)$ is called the negative of $P$, i.e. $Q = -P$ Let $P(x_1, y_1)$ and $Q(x_2, y_2)(P \neq Q)$ be two points in Equation (1), the line l (tangent line to Equation (1) if $P = Q$) joining the points $P$ and $Q$ intersects the curve (1) at $-R(x_3, -y_3)$ and the reflection of $-R$ with respect to $x$-axis is the point $R(x_3, y_3)$, i.e. $P + Q = R$. The points $E_q(a, b)$ together with a point $O$ (called point at infinity) form an additive cyclic group $G_q$, that is, $G_q = (x, y) : a, b, x, y \in F_q$ and $(x, y) \in E_q(a, b) \cup O$ of prime order $q$. The scalar point multiplication on the group $G_q$ can be computed as follows: $kP = P + P + \ldots + P$ ($k$ times). A point $P$ has order $n$ if $n$ is the smallest positive integer such that $nP = O$.

### 2.2 Computational Assumptions

**Definition 1. Elliptic Curve Computational Diffie-Hellman Problem (EC-CDH)** is defined as follows: Let $\mathcal{A}^{EC-CDH}$ be an adversary. $\mathcal{A}^{EC-CDH}$ tries to solve the following problem: Given a random instance $(P, aP, bP) \in G_q$, compute $abP$. We define $\mathcal{A}^{EC-CDH}$'s advantage in solving the $EC - CDH$ by **Adv(**$\mathcal{A}^{EC-CDH}$**)**=Pr[$\mathcal{A}^{EC-CDH}(P, aP, bP) = abP$].

## 3 Certificateless Hybrid Signcryption Scheme Without Pairing

In this section, we present the Certificateless Hybrid Signcryption (CL-HSC) scheme.

*3.1 Definition of CL-HSC scheme*

The generic certificateless hybrid signcryption scheme is a 8-tuple CL-HSC=(SetUp, SetSecretValue, PartialPrivateKeyExtract, SetPrivateKey, SetPublicKey, SymmetricKeyGen, Encapsulation, Decapsulation). The description of each probabilistic polynomial time algorithm is as follows.

- SetUp: The Key Generation Center (KGC) runs this algorithm, which takes a security parameter $k$ as input and returns system parameters `params` and a master secret key `msk` of KGC. We assume that `params` are publicly available to all users whereas the `msk` is kept secret by KGC.
- SetSecretValue: This algorithm is run by each user $A$ to generate a secret value for oneself. It takes `params` and an identity $ID_A$ of the user $A$ as inputs and outputs user's secret value $x_A$ and a corresponding public value $P_A$.
- PartialPrivateKeyExtract: The KGC runs this algorithm to generate the partial private key of the users. This algorithm takes `params`, `msk` and an identity $ID_A$ of the user $A$ as inputs, and then it outputs the partial private key $D_A$ of $ID_A$. The KGC runs this algorithm for each user, and we assume upthat the partial private key is distributed securely to the user.
- SetPrivateKey: This algorithm is run by each user $A$ to generate the full private key. It takes `params`, partial private key $D_A$ and the secret value $x_A$ of $ID_A$ as inputs, and then it outputs the full private key $sk_A$ for $A$.
- SetPublicKey: This algorithm is run by each user $A$ to generate the full public key. It takes `params` and a user's secret value $x_A$ as inputs and then returns the full public key $pk_A$ to $A$ as output.
- SymmetricKeyGen: This symmetric key generation algorithm is run by the sender $A$ to obtain the symmetric key $K$ and an internal state information $\omega$, which is not known to a receiver $B$. It takes the sender's identity $ID_A$, a full public key $pk_A$, a full private key $sk_A$, the receiver's identity $ID_B$ and a full public key $pk_B$ as inputs and then returns the symmetric key $K$ and $\omega$ to $A$ as output.
- Encapsulation: This key encapsulation algorithm is executed by the sender $A$ to obtain the encapsulation $\varphi$. It takes a state information $\omega$ corresponding to $K$, an arbitrary tag $\tau$, the sender's identity $ID_A$, a full public key $pk_A$ and a full private key $sk_A$ as input. The $\tau$ and $\varphi$ are sent to the receiver $B$.
- Decapsulation: This key decapsulation algorithm is executed by the receiver $B$ to obtain the key $K$ encapsulated in $\varphi$. It takes the encapsulation $\varphi$, a tag $\tau$, the sender's identity $ID_A$, a full public key $pk_A$, the receiver's identity $ID_B$, a full public key $pk_B$ and a full private key $sk_B$ as input and then returns the symmetric key $K$ or invalid with respect to the validity of $\psi$ as

output.

The consistency constraint is if $(K, \omega) = \mathsf{SymmetricKeyGen}(ID_A, pk_A, sk_A, ID_B, pk_B)$ and $\varphi = \mathsf{Encapsulation}(\omega, \tau)$, then $K = \mathsf{Decapsulation}(\varphi, \tau, ID_A, pk_A, ID_B, pk_B, sk_B)$.

### *3.2 Security Model for CL-HSC*

Barbosa et al. [6] firstly formalized the security notion for certificateless signcryption (CL-SC) scheme. Then, Selvi et al. [14] provided the security model for certificateless hybrid signcryption (CL-HSC) scheme. A CL-HSC scheme must satisfy confidentiality (indistinguishability against adaptive chosen ciphertext and identity attacks (IND-CCA2)) and unforgeability (existential unforgeability against adaptive chosen messages and identity attacks (EUF-CMA)). In order to prove the confidentiality and the unforgeability of CL-HSC scheme, we have to consider two types of adversaries, Type I and Type II. A Type I adversary models an attacker which is a common user of the system without the possession of the $KGC$'s master secret key. But it is able to adaptively replace users' public keys with valid public keys of its choice. A Type II adversary models an honest-but-curious $KGC$ who knows the $KGC$'s master secret key. But it cannot replace users' public keys. For the confidentiality, we consider two games "IND-CL-HSC-CCA2-I" and "IND-CL-HSC-CCA2-II" where a Type I adversary $\mathcal{A}_{\mathcal{I}}$ and a Type II adversary $\mathcal{A}_{\mathcal{II}}$ interact with their "Challenger $\mathcal{C}$" in these two games, respectively. For the unforgeability, we consider two games "EUF-CL-HSC-CMA-I" and "EUF-CL-HSC-CMA-II" where a Type I forger $\mathcal{F}_{\mathcal{I}}$ and a Type II forger $\mathcal{F}_{\mathcal{II}}$ interact with their "Challenger $\mathcal{C}$" in these two games, respectively. Now we describe these games below.

#### *3.2.1 Confidentiality*

A CL-HSC scheme is indistinguishable against chosen ciphertext and identity attacks (IND-CL-HSC-CCA2), if no polynomially bounded adversaries $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{II}}$ have non-negligible advantage in both IND-CL-HSC-CCA2-I and IND-CL-HSC-CCA2-II games between $\mathcal{C}$ and $\mathcal{A}_{\mathcal{I}}$, $\mathcal{A}_{\mathcal{II}}$ respectively:

**IND-CL-HSC-CCA2-I**: The following is the interactive game between $\mathcal{C}$ and $\mathcal{A}_{\mathcal{I}}$. The challenger $\mathcal{C}$ runs this algorithm to generate the master public and private keys, `params` and `msk` respectively. $\mathcal{C}$ gives `params` to $\mathcal{A}_{\mathcal{I}}$ and keeps the master private key `msk` secret from $\mathcal{A}_{\mathcal{I}}$.

**Phase 1**: $\mathcal{A}_{\mathcal{I}}$ performs a series of queries in an adaptive fashion in this phase. The queries allowed are given below:

- **Partial-Private-Key-Extract queries**: $\mathcal{A}_{\mathcal{I}}$ 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 $\mathcal{A}_{\mathcal{I}}$.

- **Set-Secret-Value queries**: $\mathcal{A}_{\mathcal{I}}$ 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 $sk_i$. If $\mathcal{A}_{\mathcal{I}}$ has already replaced $ID_i$'s public key, then $\mathcal{C}$ does not provide the corresponding private key to $\mathcal{A}_{\mathcal{I}}$.
- **Set-Public-Key queries**: $\mathcal{A}_{\mathcal{I}}$ 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 $ID_i$.
- **Public-Key-Replacement queries**: $\mathcal{A}_{\mathcal{I}}$ can repeatedly replace the public key $pk_i$ corresponding to the user identity $ID_i$ with any value $pk_i'$ of $\mathcal{A}_{\mathcal{I}}$'s choice. The current value of the user's public key is used by $\mathcal{C}$ in any computations or responses to $\mathcal{A}_{\mathcal{I}}$'s queries.
- **Symmetric Key Generation queries**: $\mathcal{A}_{\mathcal{I}}$ 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 $sk_A$ is obtained from the corresponding list maintained by $\mathcal{C}$. $\mathcal{C}$ computes the symmetric key $K$ and an internal state information $\omega$, stores and keeps $\omega$ secret from the view of $\mathcal{A}_{\mathcal{I}}$ and sends the symmetric key $K$ to $\mathcal{A}_{\mathcal{I}}$. 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 $\mathcal{A}_{\mathcal{I}}$ provides the private key of $ID_A$ to $\mathcal{C}$.
- **Key Encapsulation queries**: $\mathcal{A}_{\mathcal{I}}$ produces an arbitrary tag $\tau$, the sender's identity $ID_A$ and public key $pk_A$. The private key of the sender $sk_A$ is known to $\mathcal{C}$. $\mathcal{C}$ checks whether a corresponding $\omega$ value is stored previously. If $\omega$ exists then $\mathcal{C}$ computes the encapsulation $\varphi$ with $\omega$ and $\tau$ and deletes $\omega$, else returns invalid.
- **Key Decapsulation queries**: $\mathcal{A}_{\mathcal{I}}$ produces an encapsulation $\varphi$, a tag $\tau$, 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 $sk_B$ is obtained from the corresponding list maintained by $\mathcal{C}$. $\mathcal{C}$ returns the key $K$ or invalid with respect to the validity of $\varphi$. 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 $\mathcal{A}_{\mathcal{I}}$ provides the private key of $ID_B$ to $\mathcal{C}$.

**Challenge**: At the end of Phase 1 decided by $\mathcal{A}_{\mathcal{I}}$, $\mathcal{A}_{\mathcal{I}}$ sends to $\mathcal{C}$, a sender identity $ID_{A^*}$ and a receiver identity $ID_{B^*}$ on which $\mathcal{A}_{\mathcal{I}}$ wishes to be challenged. Here, the private key of the receiver $ID_{B^*}$ was not queried in Phase 1. Now, $\mathcal{C}$ computes $(K_1, \omega^*)$ using $\mathsf{SymmetricKeyGen}(ID_A, pk_A, sk_A, ID_B, pk_B)$ and chooses $K_0 \in_R \mathcal{K}$, where $K$ is the key space of the CL-HSC scheme. Now $\mathcal{C}$ chooses a bit $\delta \in_R \{0,1\}$ and sends $K_\delta$ to $\mathcal{A}_{\mathcal{I}}$. $\mathcal{A}_{\mathcal{I}}$ generates an arbitrary tag $\tau^*$ and sends it to $\mathcal{C}$. $\mathcal{C}$ computes the challenge encapsulation $\varphi^*$ with $\omega^*$ and $\tau^*$ and sends $\varphi^*$ to $\mathcal{A}_{\mathcal{I}}$.

**Phase II**: $\mathcal{A}_{\mathcal{I}}$ can perform polynomially bounded number of queries adaptively again as in Phase 1 but it cannot make a partial-private-key extract query on $ID_{B^*}$ or cannot query for the key decapsulation of $\varphi^*$.

**Guess**: $\mathcal{A}_{\mathcal{I}}$ outputs a bit $\delta$ and wins the game if $\delta = \delta$.

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

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

**Phase 1**: $\mathcal{A}_{\mathcal{II}}$ performs a series of queries in an adaptive fashion in this phase. The queries allowed are similar to that of IND-CL-HSC-CCA2-I game except that **Partial-Private-Key-Extract queries** is not included, because $\mathcal{A}_{\mathcal{II}}$ can generate it on need basis as it knows `msk`.

**Challenge**: At the end of Phase 1 decided by $\mathcal{A}_{\mathcal{II}}$, $\mathcal{A}_{\mathcal{II}}$ sends to $\mathcal{C}$, a sender identity $ID_{A^*}$ and a receiver identity $ID_{B^*}$ on which $\mathcal{A}_{\mathcal{II}}$ wishes to be challenged. Here, the private key of the receiver $ID_{B^*}$ was not queried in Phase 1. Now, $\mathcal{C}$ computes $(K_1, \omega^*)$ using $\mathsf{SymmetricKeyGen}(ID_A, pk_A, sk_A, ID_B, pk_B)$ and chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the CL-HSC scheme. Now $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends $K_\delta$ to $\mathcal{A}_{\mathcal{II}}$. $\mathcal{A}_{\mathcal{II}}$ generates an arbitrary tag $\tau^*$ and sends it to $\mathcal{C}$. $\mathcal{C}$ computes the challenge encapsulation $\varphi^*$ with $\omega^*$ and $\tau^*$ and sends $\varphi^*$ to $\mathcal{A}_{\mathcal{I}}$.

**Phase II**: $\mathcal{A}_{\mathcal{II}}$ can perform polynomially bounded number of queries adaptively again as in Phase 1 but it cannot make a set-secret-value query on $ID_{B^*}$, cannot make a public-key-replacement query on $ID_{B^*}$ or cannot query for the key decapsulation of $\varphi^*$.

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

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

#### *3.2.2 Existential Unforgeability*

A CL-HSC scheme is existentially unforgeable against adaptive chosen message attack (EUF-CL-HSC-CMA), if no polynomially bounded forgers $\mathcal{F}_{\mathcal{I}}$ and $\mathcal{F}_{\mathcal{II}}$ have non-negligible advantage in both "EUF-CL-HSC-CMA-I" and "EUF-CL-HSC-CMA-II" games between $\mathcal{C}$ and $\mathcal{F}_{\mathcal{I}}$, $\mathcal{F}_{\mathcal{II}}$ respectively:

**EUF-CL-HSC-CMA-I**: The following is the interactive game between $\mathcal{C}$ and $\mathcal{F_I}$. 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**: $\mathcal{F_I}$ may make a series of polynomially bounded number of queries to random oracles $H_i (0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:
All the oracles and queries needed in the training phase are identical to those of queries allowed in **Phase 1** of IND-CL-HSC-CCA2-I game.

**Forgery**: A the end of the **Training Phase**, $\mathcal{F_I}$ sends to $\mathcal{C}$ an encapsulation $\langle \tau^*, \omega^*, ID_{A^*}, ID_{B^*} \rangle$ on a arbitrary tag $\tau^*$, where $ID_{A^*}$ is the sender identity and $ID_{B^*}$ is the receiver identity. During the **Training Phase**, the partial private key of the sender $ID_{A^*}$ must not be queried and the public key of the sender $ID_{A^*}$ must not be replaced, simultaneously. Moreover $\omega^*$ must not be the response for any key encapsulation queries by $\mathcal{F_I}$ during the **Training Phase**.

If the output of $\mathsf{Decapsulation}(\omega^*, \tau^*, ID_{A^*}, pk_{A^*}, ID_{B^*}, pk_{B^*}, sk_{B^*})$ is valid, $\mathcal{F_I}$ wins the game. The advantage of $\mathcal{F_I}$ is defined as the probability with which it wins the EUF-CL-HSC-CMA-I game.

**EUF-CL-HSC-CMA-II**: The following is the interactive game between $\mathcal{C}$ and $\mathcal{F_{II}}$. The challenger $\mathcal{C}$ runs this algorithm to generate the master public and private keys, `params` and `msk` respectively. $\mathcal{C}$ gives `params` and the master private key `msk` to $\mathcal{F_I}$.

**Training Phase**: $\mathcal{F_{II}}$ may make a series of polynomially bounded number of queries to random oracles $H_i (0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:
All the oracles and queries needed in the training phase are identical to those of queries allowed in **Phase 1** of IND-CL-HSC-CCA2-II game.

**Forgery**: A the end of the **Training Phase**, $\mathcal{F_{II}}$ sends to $\mathcal{C}$ an encapsulation $\langle \tau^*, \omega^*, ID_{A^*}, ID_{B^*} \rangle$ on a arbitrary tag $\tau^*$, where $ID_{A^*}$ is the sender identity and $ID_{B^*}$ is the receiver identity. During the **Training Phase**, the secret value $x_{A^*}$ of the sender $ID_{A^*}$ must not be queried and the public key of the sender $ID_{A^*}$ must not be replaced, simultaneously. Moreover $\omega^*$ must not be the response for any key encapsulation queries by $\mathcal{F_{II}}$ during the **Training Phase**.

If the output of $\mathsf{Decapsulation}(\omega^*, \tau^*, ID_{A^*}, pk_{A^*}, ID_{B^*}, pk_{B^*}, sk_{B^*})$ is valid, $\mathcal{F_{II}}$ wins the game. The advantage of $\mathcal{F_{II}}$ is defined as the probability with which it wins the EUF-CL-HSC-CMA-II game.

# 4 ECC based Certificateless Hybrid Signcryption Scheme Without Pairing

## *4.1 Setup*

This algorithm takes a security parameter $k \in \mathbb{Z}^+$ as input, and returns list of system parameter $\Omega$ and KGC's master private key $msk$. Given $k$, KGC performs the following steps:

(1) Choose a $k$-bit prime $q$ and determine the tuple $\{F_q, E/F_q, G_q, P\}$, where the point $P$ is the generator of $G_q$.
(2) Choose the master key $x \in \mathbb{Z}_q^*$ uniformly at random and compute the system public key $P_{pub} = xP$.
(3) Choose cryptographic hash functions $H_0 : \{0,1\}^* \times G_q^2 \to \mathbb{Z}_q^*$, $H_1 : \{0,1\}^* \times G_q^2 \to \mathbb{Z}_q^*$ and $H_2 : \{0,1\}^* \times G_q^2 \to \mathbb{Z}_q^*$.
(4) Publish $\Omega = \{F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3\}$ as the system's parameter and keep the master key $x$ is secret.

## *4.2 Set Secret Value*

The entity **A** with an identity $ID_A$ chooses $x_A \in \mathbb{Z}_q^*$ uniformly at random as his secret value and generates the corresponding public key as $P_A = x_A P$.

## *4.3 Partial Private key Extract*

This algorithm takes KGC's master secret key, identity of an entity and the system parameter as input. Then, it returns the partial private key of the entity. In order to obtain the partial private key, the entity **A** sends $(ID_A, P_A)$ to the KGC and then KGC does as follows:

(1) Choose $r_A \in \mathbb{Z}_q^*$ uniformly at random and compute $R_A = r_A P$.
(2) Compute $d_A = r_A + xH_0(ID_A, R_A, P_A) \bmod q$.

The partial private key of the entity **A** is $d_A$. The entity can validate his private key by checking whether $d_A P = R_A + H_0(ID_A, R_A, P_A)P_{pub}$ holds.

## *4.4 Set Private Key*

The entity **A** takes the pair $sk_A = (d_A, x_A)$ as his full private key.

### 4.5 Set Public Key

The entity **A** takes the pair $pk_A = (P_A, R_A)$ as his full public key.

### 4.6 Symmetric Key Generation

Given the sender(entity **A**)'s identity $ID_A$, full public key $pk_A$, full private key $sk_A$, the receiver's identity $ID_B$ and full public key $pk_B$ as input, the sender executes this symmetric key generation algorithm to obtain the symmetric key $K$ as follows:

(1) Choose $l_A \in \mathbb{Z}_q^*$ uniformly at random and compute $U = l_A P$.
(2) Compute $T = l_A \cdot H_0(ID_B, R_B, P_B)P_{pub} + l_A \cdot R_B \bmod q$ and $K = H_1(U, T, l_A \cdot P_B, ID_B, P_B)$.
(3) Output $K$ and the intermediate information $\omega = (l_A, U, T, ID_A, pk_A, sk_A, ID_B, pk_B)$.

### 4.7 Encapsulation

Given a state information $\omega$ and an arbitrary tag $\tau$, the sender **A** obtains the encapsulation $\varphi$ by performing the following:

(1) Compute $H = H_2(U, \tau, T, ID_A, P_A, ID_B, P_B)$, $H\ = H_3(U, \tau, T, ID_A, P_A, ID_B, P_B)$ and $W = d_A + l_A \cdot H + x_A \cdot H$
(2) Output $\varphi = (U, W)$.

### 4.8 Decapsulation

Given the encapsulation $\varphi$, a tag $\tau$, the sender's identity $ID_A$, full public key $pk_A$, the receiver's identity $ID_B$, full public key $pk_B$ and full private key $sk_B$, the key $K$ is computed as follows:

(1) Compute $T = d_B \cdot U$ $(= (r_B + xH_0(ID_B, R_B, P_B)) \cdot l_A P \bmod q = l_A \cdot H_0(ID_B, R_B, P_B)P_{pub} + l_A \cdot R_B \bmod q)$.
(2) Compute $H = H_2(U, \tau, T, ID_A, P_A, ID_B, P_B)$ and $H\ = H_3(U, \tau, T, ID_A, P_A, ID_B, P_B)$.
(3) If $W \cdot P = R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H\ \cdot P_A$, output $K = H_1(U, T, x_B \cdot U, ID_B, P_B)$. Otherwise, output invalid.

## 5 Security Analysis

### *5.1 Type-I Confidentiality*

**Theorem 1.** Suppose that the hash functions $H_i(i = 0, 1, 2, 3)$ are random oracles. If there exists an adversary $\mathcal{A}_{\mathcal{I}}$ against the IND-CL-HSC-CCA2-I security of the CL-HSC scheme with advantage a non-negligible $\varepsilon$, asking $q_{ppri}$ partial-private-key queries, $q_{sv}$ set-secret-value queries and $q_{H_i}$ random oracle queries to $H_i$ $(0 \leq i \leq 3)$, then there exist an algorithm $\mathcal{C}$ that solves the $EC-CDH$ problem with the following advantage $\varepsilon$

$$\varepsilon \ \geq \varepsilon \cdot (1 - \frac{q_{ppri}}{q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (\frac{1}{q_{H_0} - q_{ppri} - q_{sv}}) \cdot (\frac{1}{q_{H_1}})$$

*Proof.* A challenger $\mathcal{C}$ is challenged with an instance of the EC-CDH problem. Given $\langle P, aP, bP\rangle \in G_q$, $\mathcal{C}$ must find $abP$. Let $\mathcal{A}_{\mathcal{I}}$ be an adversary who is able to break the IND-CL-HSC-CCA2-I security of the CL-HSC scheme. $\mathcal{C}$ can utilize $\mathcal{A}_{\mathcal{I}}$ to compute the solution $abP$ of the EC-CDH instance by playing the following interactive game with $\mathcal{A}_{\mathcal{I}}$. To solve the EC-CDH problem, $\mathcal{C}$ sets the master private/public key pair as $(x = a, P_{pub} = aP)$, where $P$ is the generator of the group $G_q$ and the hash functions $H_i(0 \leq i \leq 3)$ are treated as random oracles. The $\mathcal{C}$ sends the system parameters $\Omega = \{F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3\}$ to $\mathcal{A}_{\mathcal{I}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i(0 \leq i \leq 3)$). It also maintains a list of issued private keys and public keys in $L_k$. $\mathcal{C}$ can simulate the Challenger's execution of each phase of the formal Game. Let $\mathcal{C}$ select a random index $t$, where $1 \leq t \leq q_{H_0}$ and fixes $ID_t$ as the target identity for the challenge phase.

**Phase 1**: $\mathcal{A}_{\mathcal{I}}$ may make a series of polynomially bounded number of queries to random oracles $H_i(0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:

**Create(**$ID_i$: When $\mathcal{A}_{\mathcal{I}}$ submits a Create($ID_i$) query to $\mathcal{C}$, $\mathcal{C}$ responds as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ chooses $e_i, x_i \in_R Z_q^*$ and sets $H_0(ID_i, R_i, P_i) = -e_i$, $R_i = e_i P_{pub} + bP$ and $P_i = x_i P$. Here, $\mathcal{C}$ does not know $b$. $\mathcal{C}$ uses the $bP$ given in the instance of the EC-CDH problem. $\mathcal{C}$ inserts $(ID_i, R_i, P_i, e_i)$ to the list $L_0$ and $(ID_i, \perp, x_i, R_i, P_i)$ to the list $L_k$.
- If $ID_i = ID_t$, $\mathcal{C}$ picks $e_i, b_i, x_i \in_R Z_q^*$, then sets $H_0(ID_i, R_i, P_i) = -e_i$, $R_i = e_i P_{pub} + b_i P$ and computes the public key as $P_i = x_i P$. $d_i = b_i$ and it satisfies the equation $d_i P = R_i + H_0(ID_i, R_i, P_i)P_{pub}$. $\mathcal{C}$ inserts $(ID_i, R_i, P_i, e_i)$ to the list $L_0$ and $(ID_i, d_i, x_i, R_i, P_i)$ to the list $L_k$.

$H_0$ **queries**: When $\mathcal{A}_{\mathcal{I}}$ submits a $H_0$ query with $ID_i$, $\mathcal{C}$ searches the list $L_0$. If there is a tuple $ID_i, R_i, P_i, e_i)$, $\mathcal{C}$ responds with the previous value $e_i$. Otherwise, $\mathcal{C}$ chooses $e_i \in_R Z_q^*$ and returns $e_i$ as the answer. Then, $\mathcal{C}$ inserts $(ID_i, R_i, P_i, l_i)$ to the list $L_0$.

$H_1$ **queries**: $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, l_A \cdot P_B, ID_B, P_B \rangle$ exists in list $L_1$. If it exists, $\mathcal{C}$ returns $K$ to $\mathcal{A}_{\mathcal{I}}$. Otherwise, it chooses $K \in_R \{0,1\}^n$ and adds the tuple $\langle U, T, l_A \cdot P_B, ID_B, P_B, K \rangle$ to the $L_1$, then returns $K$ to $\mathcal{A}_{\mathcal{I}}$.

$H_2$ **queries**: $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \rangle$ exists in the list $L_2$. If it exists, $\mathcal{C}$ returns $H$ to $\mathcal{A}_{\mathcal{I}}$. Otherwise, $\mathcal{C}$ performs the following steps.

- If $ID_B = ID_t$, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H = h_i P \rangle$ to the $L_2$ and returns $H$ to $\mathcal{A}_{\mathcal{I}}$.
- If $ID_B = ID_t$, $\mathcal{C}$ picks $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H = h_i P_{pub} \rangle$ to the $L_2$ and returns $H$ to $\mathcal{A}_{\mathcal{I}}$.

$H_3$ **queries**: $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \rangle$ exists in the list $L_3$. If it exists, $\mathcal{C}$ returns $H$ to $\mathcal{A}_{\mathcal{I}}$. Otherwise, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H = h_i P \rangle$ to the $L_3$ and returns $H$ to $\mathcal{A}_{\mathcal{I}}$.

**Partial-Private-Key-Extract queries**: In order to respond to the query for the partial private key of a user with $ID_i$, $\mathcal{C}$ performs as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ aborts the execution.
- If $ID_i = ID_t$, $\mathcal{C}$ retrieves the tuple $\langle ID_i, d_i, x_i, R_i, P_i \rangle$ from $L_k$, returns $(d_i, R_i)$ which satisfies the equation $d_i P = R_i + H_0(ID_i, R_i, P_i) P_{pub}$.

**Set-Secret-Value queries**: $\mathcal{A}_{\mathcal{I}}$ produces $ID_i$ to $\mathcal{C}$ and requests a secret value of the user with $ID_i$. If the public key of $ID_i$ has not been replaced and $ID_i = ID_t$, then $\mathcal{C}$ responds with $x_i$ by retrieving from the $L_k$. If $\mathcal{A}_{\mathcal{I}}$ has already replaced the public key of $ID_i$, $\mathcal{C}$ does not provide the corresponding secret value to $\mathcal{A}_{\mathcal{I}}$. If $ID_i = ID_t$, $\mathcal{C}$ aborts.

**Set-Public-Key queries**: $\mathcal{A}_{\mathcal{I}}$ produces $ID_i$ to $\mathcal{C}$ and requests a public key of the user with $ID_i$. $\mathcal{C}$ checks in the $L_k$ for a tuple of the form $\langle ID_i, d_i, x_i, R_i, P_i \rangle$. If it exists, $\mathcal{C}$ returns the corresponding public key $(R_i, P_i)$. Otherwise, $\mathcal{C}$ recalls Create$(ID_i)$ query to obtain $(R_i, P_i)$ and returns $(R_i, P_i)$ as the answer.

**Public-Key-Replacement queries**: $\mathcal{A}_{\mathcal{I}}$ chooses values $(R_i, P_i)$ to replace the public key $(R_i, P_i)$ of a user $ID_i$. $\mathcal{C}$ updates the corresponding tuple in the list $L_k$ as $\langle ID_i, -, -, R_i, P_i \rangle$. The current value of the user's public key is used by $\mathcal{C}$ for computations or responses to any queries made by $\mathcal{A}_{\mathcal{I}}$.

**Symmetric Key Generation queries**: $\mathcal{A}_\mathcal{I}$ produces a sender's identity $ID_A$, public key $(R_A, P_A)$, the receiver's identity $ID_B$ and public key $(R_B, P_B)$ to $\mathcal{C}$. $\mathcal{C}$ computes the symmetric key $K$ and an internal state information $\omega$, stores and keeps $\omega$ secret from the view of $\mathcal{A}_\mathcal{I}$ and sends the symmetric key $K$ to $\mathcal{A}_\mathcal{I}$. $\mathcal{C}$ can perform this step 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 utilize the private key of either the sender or receiver.

**Key Encapsulation queries**: $\mathcal{A}_\mathcal{I}$ produces an arbitrary tag $\tau$, the sender's identity $ID_A$, public key $(R_A, P_A)$, the receiver's identity $ID_B$ and public key $(R_B, P_B)$ and sends to $\mathcal{C}$. The full private key of the sender $(d_A, x_A)$ is obtained from the list $L_k$. $\mathcal{C}$ checks whether a corresponding $\omega$ value has been stored previously.

- If $\omega$ does not exist, $\mathcal{C}$ returns invalid.
- If a corresponding $\omega$ exists and $ID_i = ID_t$, then $\mathcal{C}$ computes $\varphi$ with $\omega$ and $\tau$ by using the actual Encapsulation algorithm, and deletes $\omega$.
- If a corresponding $\omega$ exists and $ID_i = ID_t$, then $\mathcal{C}$ computes $\varphi$ by performing the following steps. ($\mathcal{C}$ does not know the private key corresponding to $ID_t$, so it should perform the encapsulation in a different way.):
  · Choose $r, h_i, h_{i} \in_R Z_q^*$ and compute $U = rP - (h_i P_{pub})^{-1} \cdot (R_A - e_i P_{pub})$, where $R_A - e_i P_{pub} = bP$, $R_A$ and $e_i$ are obtained from the list $L_0$.
  · Compute $H = h_i P_{pub}$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \rangle$ to the list $L_2$.
  · Compute $H \ = h_{i} P$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_{i}, H \ \rangle$ to the list $L_3$.
  · Compute $W = rH + h_{i} P_A$.
  · Output $\varphi = (U, W)$ as the encapsulation.

  We show that $\mathcal{A}_\mathcal{I}$ can pass the verification of $\varphi = (U, W)$ to validate the encapsulation, because the equality $W \cdot P = R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H \ \cdot P_A$ holds as follows:

  $$\begin{aligned}
  & R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H \ \cdot P_A \\
  &= bP + eP_{pub} + (-e_i) \cdot P_{pub} + H \cdot (rP - (h_i P_{pub})^{-1} \cdot (R_A - e_i P_{pub})) + H \ \cdot P_A \\
  &= bP + e_i P_{pub} - e_i P_{pub} + (h_i P_{pub}) \cdot (rP - (h_i P_{pub})^{-1} \cdot bP) + h_{i} P \cdot P_A \\
  &= bP + h_i P_{pub} \cdot rP - bP + h_{i} P \cdot P_A \\
  &= (rH + h_{i} P_A) \cdot P \\
  &= W \cdot P
  \end{aligned}$$

**Key Decapsulation queries**: $\mathcal{A}_\mathcal{I}$ produces an encapsulation $\varphi$, a tag $\tau$, the sender's identity $ID_A$, public key $(R_A, P_A)$, the receiver's identity $ID_B$ and public key $(R_B, P_B)$ to $\mathcal{C}$. The full private key of the receiver $(d_B, x_B)$ is obtained from the list $L_k$.

- If $ID_i = ID_t$, then $\mathcal{C}$ computes the decapsulation of $\varphi$ by using the actual Decapsulation algorithm.

- If $ID_i = ID_t$, then $\mathcal{C}$ computes $K$ from $\varphi$ as follows:
  · Searches in the list $L_2$ and $L_3$ for entries of the type $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H\rangle$ and $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \ \rangle$ respectively.
  · If entries $H$ and $H$ exist then $\mathcal{C}$ checks whether the equality $W \cdot P = R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H \ \cdot P_A$ holds.
  · If the above equality holds, the corresponding value of $T$ is retrieved from the lists $L_2$ and $L_3$. Both the $T$ values should be equal.
  · $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, x_B \cdot U, ID_B, P_B, K\rangle$ exists in the list $L_1$. If it exists the corresponding $K$ value is output as the decapsulation of $\varphi$.

**Challenge**: At the end of Phase 1, $\mathcal{A_I}$ sends a sender identity $ID_{A*}$ and a receiver identity $ID_{B*}$ on which $\mathcal{A_I}$ wishes to be challenged to $\mathcal{C}$. Here, the full private key of the receiver $ID_{B*}$ was not queried in Phase 1. $\mathcal{C}$ aborts the game if $ID_{B*} = ID_t$. Otherwise, $\mathcal{C}$ performs the following to compute the challenge encapsulation $\varphi^*$.

- Set $U = cP$ and choose $T \in_R G_q$.
- Choose $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the CL-HSC scheme.
- Compute $K_1 = H_1(U, T, x_B \cdot U, ID_B, P_B)$.
- Set $\omega^* = \langle -, U, U \ , T, ID_A, P_A, R_A, x_A, d_A, ID_B, P_B, R_B\rangle$.
- $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends $K_\delta$ to $\mathcal{A_I}$.
- $\mathcal{A_I}$ generates an arbitrary tag $\tau^*$ and sends it to $\mathcal{C}$.
- Choose $h_i, h_i \in_R \mathbb{Z}_q^*$, store the tuple $\langle U, \tau^*, T, ID_A, P_A, ID_B, P_B, h_i, H = h_iP\rangle$ to the list $L_2$ and $\langle U, \tau^*, T, ID_A, P_A, ID_B, P_B, h_i, H \ = h_iP\rangle$ to the list $L_3$.
- Since $\mathcal{C}$ knows the private key of the sender, $\mathcal{C}$ computes $W = d_A + h_i \cdot cP + h_i x_A P$.
- $\mathcal{C}$ sends $\omega^* = \langle U, W\rangle$ 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 it cannot query decapsulation on $\omega^*$.

**Guess**: Since $\mathcal{A_I}$ is able to break the IND-CL-HSC-CCA2-I security of CL-HSC (which is assumed at the beginning of the proof), $\mathcal{A_I}$ should have asked a $H_1$ query with $(U, T, x_B \cdot U, ID_B, P_B)$ as inputs. $T = r \cdot H_0(ID_B, R_B, P_B) \cdot P_{pub} + r \cdot R_B = c \cdot (-e_i) \cdot aP + c \cdot (e_i \cdot aP + bP) = c \cdot bP$. Therefore, if the list $L_1$ has $q_{H_1}$ queries corresponding to the sender $ID_A$ and receiver $ID_B$, one of the $T$'s among $q_{H_1}$ values stored in the list $L_1$ is the solution for the $EC - CDH$ problem instance. $\mathcal{C}$ chooses one $T$ value uniformly at random from the $q_{H_1}$ values from the list $L_1$ and outputs it as the solution for the $EC - CDH$ instance.

**Analysis**: In order to assess the probability of success of the challenger $\mathcal{C}$, Let $E_1$, $E_2$ and $E_3$ be the events in which $\mathcal{C}$ aborts the IND-CL-HSC-CCA2-I game.

- $E_1$ is an event when $\mathcal{A}_{\mathcal{I}}$ queries the partial private key of the target identity $ID_t$. The probability of $E_1$ is $Pr[E_1] = \frac{q_{ppri}}{q_{H_0}}$.
- $E_2$ is an event when $\mathcal{A}_{\mathcal{I}}$ asks to query the set secret value of the target identity $ID_t$. The probability of $E_2$ is $Pr[E_2] = \frac{q_{sv}}{q_{H_0}}$.
- $E_3$ is an event when $\mathcal{A}_{\mathcal{I}}$ does not choose the target identity $ID_t$ as the receiver during the challenge. The probability of $E_3$ is $Pr[E_3] = 1 - \frac{1}{q_{H_0} - q_{ppri} - q_{sv}}$.

Thus, the probability that $\mathcal{C}$ does not abort the IND-CL-HSC-CCA2-I game is

$$Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3] = (1 - \frac{q_{ppri}}{q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (\frac{1}{q_{H_0} - q_{ppri} - q_{sv}})$$

The probability that $\mathcal{C}$ randomly chooses the $T$ from $L_1$ and $T$ is the solution of $EC-CDH$ is $\frac{1}{q_{H_1}}$. So, the probability that $\mathcal{C}$ finds the $EC-CDH$ instance is as follows:

$$Pr[\mathcal{C}(P, aP, bP) = abP] = \varepsilon \cdot (1 - \frac{q_{ppri}}{q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (\frac{1}{q_{H_0} - q_{ppri} - q_{sv}}) \cdot (\frac{1}{q_{H_1}})$$

Therefore, the $Pr[\mathcal{C}(P, aP, bP) = abP]$ is non-negligible, because $\varepsilon$ is non-negligible.

### *5.2 Type-II Confidentiality*

**Theorem 2.** Suppose that the hash functions $H_i (i = 0, 1, 2, 3)$ are random oracles. If there exists an adversary $\mathcal{A}_{\mathcal{II}}$ against the IND-CL-HSC-CCA2-II security of the CL-HSC scheme with advantage a non-negligible $\varepsilon$, asking $q_{sv}$ set-secret-value queries, $q_{pkR}$ public key replacement queries and $q_{H_i}$ random oracle queries to $H_i$ $(0 \leq i \leq 3)$, then there exist an algorithm $\mathcal{C}$ that solves the $EC - CDH$ problem with the following advantage $\varepsilon$

$$\varepsilon \ \geq \varepsilon \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot (\frac{1}{q_{H_0} - q_{sv} - q_{pkR}}) \cdot (\frac{1}{q_{H_1}})$$

*Proof.* A challenger $\mathcal{C}$ is challenged with an instance of the EC-CDH problem. Given $\langle P, aP, bP \rangle \in G_q$, $\mathcal{C}$ must find $abP$. Let $\mathcal{A}_{\mathcal{II}}$ be an adversary who is able to break the IND-CL-HSC-CCA2-II security of the CL-HSC scheme. $\mathcal{C}$ can utilize $\mathcal{A}_{\mathcal{II}}$ to compute the solution $abP$ of the EC-CDH instance by playing the following interactive game with $\mathcal{A}_{\mathcal{II}}$. To solve the EC-CDH, $\mathcal{C}$ chooses $s \in_R$

$\mathbb{Z}_q^*$, sets the master public key $P_{pub} = sP$, where $P$ is the generator of the group $G_q$ and the hash functions $H_i (0 \le i \le 3)$ are treated as random oracles. The $\mathcal{C}$ sends the system parameter $\Omega = \{F_q, E/F_q, G_q, P, P_{pub} = sP, H_0, H_1, H_2, H_3\}$ and the master private key $s$ to $\mathcal{A_{II}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i (0 \le i \le 3)$). It also maintains a list $L_k$ to maintain the list of issued private keys and public keys. $\mathcal{C}$ can simulate the Challenger's execution of each phase of the formal Game. Let $\mathcal{C}$ select a random index $t$, where $1 \le t \le q_{H_0}$ and fixes $ID_t$ as the target identity for the challenge phase.

**Phase 1**: $\mathcal{A_{II}}$ may make a series of polynomially bounded number of queries to random oracles $H_i (0 \le i \le 3)$ at any time and $\mathcal{C}$ responds as follows:

**Create($ID_i$) queries**: When $\mathcal{A_{II}}$ submits a Create($ID_i$) query to $\mathcal{C}$, $\mathcal{C}$ responds as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ chooses $a_i, l_i \in_R Z_q^*$ and sets $H_0(ID_i, R_i, P_i) = l_i$, computes $R_i = a_i P$, $d_i = a_i + l_i \cdot s$ and the public key as $P_i = aP$. Here, $\mathcal{C}$ does not know $a$. $\mathcal{C}$ uses the $aP$ given in the instance of the EC-CDH problem. $\mathcal{C}$ inserts $(ID_i, R_i, P_i, l_i)$ to the list $L_0$ and $(ID_i, d_i, \perp, R_i, P_i)$ to the list $L_k$.
- If $ID_i = ID_t$, $\mathcal{C}$ picks $a_i, x_i, l_i \in_R Z_q^*$, then sets $H_0(ID_i, R_i, P_i) = l_i$, computes $R_i = a_i P$, $d_i = a_i + l_i \cdot s$ and the public key as $P_i = x_i P$. $\mathcal{C}$ inserts $(ID_i, R_i, P_i, l_i)$ to the list $L_0$ and $(ID_i, d_i, x_i, R_i, P_i)$ to the list $L_k$.

$H_0$ **queries**: When $\mathcal{A_{II}}$ submits a $H_0$ query with $ID_i$, $\mathcal{C}$ searches the list $L_0$. If there is a tuple $ID_i, R_i, P_i, l_i)$, $\mathcal{C}$ responds with the previous value $l_i$. Otherwise, $\mathcal{C}$ chooses $l_i \in_R Z_q^*$ and returns $l_i$ as the answer. Then, $\mathcal{C}$ inserts $(ID_i, R_i, P_i, l_i)$ to the list $L_0$.

$H_1$ **queries**: When $\mathcal{A_{II}}$ submits a $H_1$ query with $ID_i$, $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, r \cdot P_B, ID_B, P_B \rangle$ exists in list $L_1$. If it exists, $\mathcal{C}$ returns $K$ to $\mathcal{A_{II}}$. Otherwise, it chooses $K \in_R \{0,1\}^n$ and adds the tuple $\langle U, T, r \cdot P_B, ID_B, P_B, K \rangle$ to the list $L_1$, then returns $K$ to $\mathcal{A_{II}}$.

$H_2$ **queries**: When $\mathcal{A_{II}}$ submits a $H_2$ query with $ID_i$, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \rangle$ exists in the list $L_2$. If it exists, $\mathcal{C}$ returns $H$ to $\mathcal{A_{II}}$. Otherwise, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H = h_i P \rangle$ to the $L_2$ and returns $H$ to $\mathcal{A_{II}}$.

$H_3$ **queries**: When $\mathcal{A_{II}}$ submits a $H_3$ query with $ID_i$, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \rangle$ exists in the list $L_3$. If it exists, $\mathcal{C}$ returns $H$ to $\mathcal{A_{II}}$. Otherwise, $\mathcal{C}$ performs the following:

- If $ID_A = ID_t$, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H = h_i P \rangle$ to the $L_3$ and returns $H$ to $\mathcal{A_{II}}$.
- If $ID_A = ID_t$, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H =$

$h_i \cdot bP\rangle$ to the $L_3$ and returns $H$ to $\mathcal{A}_{\mathcal{II}}$. Here, $\mathcal{C}$ knows $bP$ but does not know $b$. $\mathcal{C}$ uses the $bP$ given in the instance of the EC-CDH problem.

**Partial-Private-Key-Extract queries**: When $\mathcal{A}_{\mathcal{II}}$ asks a Partial-Private-Key-Extract query for $ID_i$, $\mathcal{C}$ checks whether the corresponding partial private key for $ID_i$, $d_i$ exists in the list $L_k$. If it exists, $\mathcal{C}$ returns $d_i$ to $\mathcal{A}_{\mathcal{II}}$. Otherwise, $\mathcal{C}$ recalls Create($ID_i$) query to obtain $d_i$ and returns $d_i$ as the answer.

**Set-Secret-Value queries**: If $\mathcal{A}_{\mathcal{II}}$ asks a Set-Secret-Value query for $ID_i$, $\mathcal{C}$ answers as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ aborts.
- If $ID_i = ID_t$, $\mathcal{C}$ looks from the tuple $(ID_i, d_i, x_i, R_i, P_i)$ in the list $L_k$. If such tuple exists in the $L_k$, $\mathcal{C}$ returns $x_i$. Otherwise, $\mathcal{C}$ recalls Create($ID_i$) query to obtain $x_i$ and returns $x_i$ as the answer.

**Set-Public-Key queries**: When $\mathcal{A}_{\mathcal{II}}$ asks Set-Public-Key query for $ID_i$, $\mathcal{C}$ searches the list $L_k$. If the public key for $ID_i$, $(R_i, P_i)$ is found in the $L_k$, $\mathcal{C}$ returns $(R_i, P_i)$ as the answer. Otherwise, $\mathcal{C}$ executes a Create($ID_i$) query to obtain $(R_i, P_i)$ and then returns $(R_i, P_i)$ as the answer.

**Public-Key-Replacement queries**: When $\mathcal{A}_{\mathcal{II}}$ asks Public-Key-Replacement query for $ID_i$, $\mathcal{C}$ checks whether $ID_i = ID_t$. If $ID_i = ID_t$, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ updates the corresponding tuple in the list $L_k$ as $\langle ID_i, -, -, R_i, P_i\rangle$, where $(R_i, P_i)$ is chosen by $\mathcal{A}_{\mathcal{II}}$. The current public key(i.e. replaced public key) is used by $\mathcal{C}$ for computations or responses to any queries made by $\mathcal{A}_{\mathcal{II}}$.

**Symmetric Key Generation queries**: $\mathcal{A}_{\mathcal{II}}$ produces a sender's identity $ID_A$, public key $(R_A, P_A)$, the receiver's identity $ID_B$ and public key $(R_B, P_B)$ then sends to $\mathcal{C}$. Now, $\mathcal{C}$ computes the symmetric key $K$ and an internal state information $\omega$, stores and keeps $\omega$ secret from the view of $\mathcal{A}_{\mathcal{II}}$ and sends the symmetric key $K$ to $\mathcal{A}_{\mathcal{II}}$. $\mathcal{C}$ can perform this step without knowing the private key corresponding to the sender $ID_A$ or the receiver $ID_B$, because computing $K$ does not utilize the private key of either the sender or the receiver.

**Key Encapsulation queries**: $\mathcal{A}_{\mathcal{II}}$ produces an arbitrary tag $\tau$, the sender's identity $ID_A$, public key $(R_A, P_A)$, the receiver's identity $ID_B$ and public key $(R_B, P_B)$ then sends to $\mathcal{C}$. $\mathcal{C}$ checks whether a corresponding $\omega$ value is stored previously.

- If $\omega$ does not exist, $\mathcal{C}$ returns invalid.
- If a corresponding $\omega$ exists and $ID_A = ID_t$, then $\mathcal{C}$ computes $\varphi$ with $\omega$ and $\tau$ by using the actual Encapsulation algorithm, and deletes $\omega$. Here, $\mathcal{C}$ gets the full private key of the sender $(d_A, x_A)$ from the list $L_k$.
- If a corresponding $\omega$ exists and $ID_A = ID_t$, then $\mathcal{C}$ computes $\varphi$ by performing the following steps. ($\mathcal{C}$ does not know the secret value $x_t$ corresponding

to $ID_t$, so it should perform the encapsulation in a different way.):

· Choose $r, h_i, h_i \in_R Z_q^*$ and compute $U = rP - h_i^{-1} \cdot P^{-1} \cdot R_A - h_i^{-1} \cdot h_i \cdot P_A$, where $R_A$ and $P_A$ are obtained from the list $L_k$.

· Compute $H = h_i P$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \rangle$ to the list $L_2$.

· Compute $H \ = h_i P$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \ \rangle$ to the list $L_3$.

· Compute $W = rH + l_i \cdot s$, where $l_i$ is obtained from the list $L_0$ and $\mathcal{C}$ knows the master private key $s$.

· Output $\varphi = (U, W)$ as the encapsulation.

We show that $\mathcal{A}_{\mathcal{II}}$ can pass the verification of $\varphi = (U, W)$ to validate the encapsulation, because the equality $W \cdot P = R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H \ \cdot P_A$ holds as follows:

$R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H \ \cdot P_A$
$= R_A + l_i \cdot sP + h_i P \cdot (rP - h_i^{-1} \cdot P^{-1} \cdot R_A - h_i^{-1} \cdot h_i \cdot P_A) + h_i P \cdot P_A$
$= R_A + l_i \cdot sP + h_i P \cdot rP - R_A - h_i P \cdot P_A + h_i P \cdot P_A$
$= l_i \cdot sP + h_i P \cdot rP$
$= l_i \cdot sP + H \cdot rP$
$= W \cdot P$

**Key Decapsulation queries**: $\mathcal{A}_{\mathcal{II}}$ produces an encapsulation $\varphi$, a tag $\tau$, the sender's identity $ID_A$, public key $(R_A, P_A)$, the receiver's identity $ID_B$ and public key $(R_B, P_B)$ to $\mathcal{C}$.

- If $ID_B = ID_t$, then $\mathcal{C}$ computes the decapsulation of $\varphi$ by using the actual Decapsulation algorithm. Here, the full private key of the receiver $(d_B, x_B)$ is obtained from the list $L_k$.
- If $ID_B = ID_t$, then $\mathcal{C}$ computes $K$ from $\varphi$ as follows:

  · Searches in the list $L_2$ and $L_3$ for entries of the type $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \rangle$ and $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i, H \ \rangle$ respectively.

  · If entries $H$ and $H$ exist then $\mathcal{C}$ checks whether the equality $W \cdot P = R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H \ \cdot P_A$ holds.

  · If the above equality holds, then retrieves the corresponding value of $T$ from the lists $L_2$ and $L_3$. Both the T values should be equal.

  · $\mathcal{C}$ checks whether a tuple of the form $\langle U, T, U \ (= x_B \cdot U), ID_B, P_B, K \rangle$ exists in the list $L_1$. If it exists, then $\mathcal{C}$ checks whether $U \ \cdot P = P_B \cdot U$. If the check holds then $\mathcal{C}$ outputs the corresponding $K$ value as the decapsulation of $\varphi$.

**Challenge**: At the end of Phase 1, $\mathcal{A}_{\mathcal{II}}$ sends a sender identity $ID_{A*}$ and a receiver identity $ID_{B*}$ on which $\mathcal{A}_{\mathcal{II}}$ wishes to be challenged to $\mathcal{C}$. Here, the secret value of the receiver $ID_{B*}$ was not queried in Phase 1. $\mathcal{C}$ aborts the game if $ID_{B*} = ID_t$. Otherwise, $\mathcal{C}$ performs the following to compute the challenge encapsulation $\varphi^*$.

- Set $U = bP$ and computes $T = d_{B*} \cdot U$, where $\mathcal{C}$ knows the partial private key for $ID_{B*}$, $d_{B*}$. Here, $\mathcal{C}$ does not know $b$. $\mathcal{C}$ uses the $bP$ given in the instance of the EC-CDH problem.
- Choose $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the CL-HSC scheme.
- Choose $U \ \in_R G_q$ and compute $K_1 = H_1(U, T, U\ , ID_{B*}, P_{B*})$.
- Set $\omega^* = \langle -, U, U\ , T, ID_{A*}, P_{A*}, R_{A*}, x_{A*}, d_{A*}, ID_{B*}, P_{B*}, R_{B*} \rangle$.
- $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends $K_\delta$ to $\mathcal{A}_{\mathcal{II}}$.
- $\mathcal{A}_{\mathcal{II}}$ generates an arbitrary tag $\tau^*$ and sends it to $\mathcal{C}$.
- Choose $h_i, h_i \in_R \mathbb{Z}_q^*$, store the tuple $\langle U, \tau^*, T, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h_i, H = h_i P \rangle$ to the list $L_2$ and $\langle U, \tau^*, T, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h_i, H\ = h_i P \rangle$ to the list $L_3$.
- Since $\mathcal{C}$ knows the private key of the sender $ID_{A*}$, $\mathcal{C}$ computes $W = d_{A*} + h_i \cdot bP + h_i x_{A*} P$.
- $\mathcal{C}$ sends $\omega^* = \langle U, W \rangle$ to $\mathcal{A}_{\mathcal{II}}$.

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

**Guess**: Since $\mathcal{A}_{\mathcal{II}}$ is able to break the IND-CL-HSC-CCA2-II security of CL-HSC (which is assumed at the beginning of the proof), $\mathcal{A}_{\mathcal{II}}$ should have asked a $H_1$ query with $(U, T, U\ , ID_{B*}, P_{B*})$ as inputs. Since $ID_{B*}$ is a target identity $ID_t$, $P_{B*} = aP$. Here, $aP$ was given as the instance of the EC-CDH problem and $\mathcal{C}$ does not know $a$. Thus, computing $U\ = x_{B*} U = abP$ is to find $abP$ when $\langle P, aP(= P_{B*}), bP(= U) \rangle \in G_q$ are given. Therefore, if the list $L_1$ has $q_{H_1}$ queries corresponding to the sender $ID_{A*}$ and receiver $ID_{B*}$, one of the $q_{H_1}$ values of $U$ stored in the list $L_1$ is the solution for the $EC-CDH$ problem instance. $\mathcal{C}$ chooses one $U$ value uniformly at random from the $q_{H_1}$ values from the list $L_1$ and outputs it as the solution for the $EC - CDH$ instance.

**Analysis**: In order to assess the probability of success of the challenger $\mathcal{C}$, let $E_1$, $E_2$ and $E_3$ be the events in which $\mathcal{C}$ aborts the IND-CL-HSC-CCA2-II game.

- $E_1$ is an event when $\mathcal{A}_{\mathcal{II}}$ queries the secret value of the target identity $ID_t$. The probability of $E_1$ is $Pr[E_1] = \frac{q_{sv}}{q_{H_0}}$.
- $E_2$ is an event when $\mathcal{A}_{\mathcal{II}}$ asks to replace the public key of the target identity $ID_t$. The probability of $E_2$ is $Pr[E_2] = \frac{q_{pkR}}{q_{H_0}}$.
- $E_3$ is an event when $\mathcal{A}_{\mathcal{II}}$ does not choose the target identity $ID_t$ as the receiver during the challenge. The probability of $E_3$ is $Pr[E_3] = 1 - \frac{1}{q_{H_0} - q_{sv} - q_{pkR}}$.

Thus, the probability that $\mathcal{C}$ does not abort the IND-CL-HSC-CCA2-II game is

$$Pr[\neg E_1 \wedge \neg E_2 \wedge \neg E_3] = (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot (\frac{1}{q_{H_0} - q_{sv} - q_{pkR}})$$

The probability that $\mathcal{C}$ randomly chooses the $U$ from $L_1$ and $U$ is the solution of $EC-CDH$ is $\frac{1}{q_{H_1}}$. So, the probability that $\mathcal{C}$ finds the $EC-CDH$ instance is as follows:

$$Pr[\mathcal{C}(P, aP, bP) = abP] = \varepsilon \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot (\frac{1}{q_{H_0} - q_{sv} - q_{pkR}}) \cdot (\frac{1}{q_{H_1}})$$

Therefore, the $Pr[\mathcal{C}(P, aP, bP) = abP]$ is non-negligible, because $\varepsilon$ is non-negligible.

### *5.3 Type-I Unforgeability*

**Theorem 3.** Suppose that the hash functions $H_i (i = 0, 1, 2, 3)$ are random oracles. If there exists a forger $\mathcal{F_I}$ against the EUF-CL-HSC-CMA-I security of the CL-HSC scheme with advantage a non-negligible $\varepsilon$, asking $q_C$ create $(ID_i)$ queries, $q_E$ key-encapsulation queries, $q_{H_i}$ random oracle queries to $H_i$ $(0 \leq i \leq 3)$, $q_{ppri}$ partial-private-key queries and $q_{sv}$ set-secret-value queries, then there exists an algorithm $\mathcal{C}$ that solves the $EC-CDH$ problem with the following advantage $\varepsilon$

$$\varepsilon \ \geq q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{ppri}}{q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot \varepsilon$$

*Proof.* A challenger $\mathcal{C}$ is challenged with an instance of the EC-CDH problem. Given $\langle P, aP, cP \rangle \in G_q$, $\mathcal{C}$ must find $acP$. Let $\mathcal{F_I}$ be a forger who is able to break the EUF-CL-HSC-CMA-I security of the CL-HSC scheme. $\mathcal{C}$ can utilize $\mathcal{F_I}$ to compute the solution $abP$ of the EC-CDH instance by playing the following interactive game with $\mathcal{F_I}$. To solve the EC-CDH problem, $\mathcal{C}$ sets the master private/public key pair as $(x = a, P_{pub} = aP)$, where $P$ is the generator of the group $G_q$ and the hash functions $H_i (0 \leq i \leq 3)$ are treated as random oracles. The $\mathcal{C}$ sends the system parameter $\Omega = \{F_q, E/F_q, G_q, P, P_{pub}, H_0, H_1, H_2, H_3\}$ to $\mathcal{F_I}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i (0 \leq i \leq 3)$). It also maintains a list $L_k$ to maintain the list of issued private keys and public keys. $\mathcal{C}$ can simulate the Challenger's execution of each phase of the formal game.

**Training Phase**: $\mathcal{F_I}$ may make a series of polynomially bounded number of queries to random oracles $H_i (0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:
All the oracles and queries needed in the training phase are identical to those of the **Create($ID_i$) queries**, $H_0$ **queries**, $H_1$ **queries**, $H_2$ **queries**, $H_3$ **queries**,

**Partial-Private-Key-Extract queries**, **Set-Secret-Value queries**, **Public-Key-Replacement queries**, **Symmetric Key Generation queries**, **Key Encapsulation queries** and **Key Decapsulation queries** in IND-CL-HSC-CCA2-I game.

**Forgery**: Eventually, $\mathcal{F}_{\mathcal{I}}$ returns a valid encapsulation $\langle \tau, \omega = (U, W), ID_A, ID_B \rangle$ on a arbitrary tag $\tau$, where $ID_A$ is the sender identity and $ID_B$ is the receiver identity, to $\mathcal{C}$. If $ID_A = ID_t$, $\mathcal{C}$ aborts the execution of this game. Otherwise, $\mathcal{C}$ searches the list $L_2$ and outputs another valid encapsulation $\langle \tau, \omega^* = (U, W^*), ID_A, ID_B \rangle$ with different $h_i^*$ such that $h_i^* = h_i$ on the same $\tau$ as done in forking lemma [21]. Thus, we can get $W \cdot P = R_t - e_t \cdot P_{pub} + U \cdot h_i P_{pub} + P_t \cdot h_i P$ and $W^* \cdot P = R_t - e_t \cdot P_{pub} + U \cdot h_i^* P_{pub} + P_t \cdot h_i P$. Let $U = cP$ and $P_{pub} = aP$. Then if we subtract these two equations, we get following value.
$W^* \cdot P - W \cdot P = U \cdot h_i^* P_{pub} - U \cdot h_i P_{pub}$
$\Rightarrow (W^* - W)P = U \cdot (h_i^* - h_i)P_{pub}$
$\Rightarrow (W^* - W)P = cP \cdot (h_i^* - h_i)P_{pub}$
$\Rightarrow (W^* - W) = c \cdot (h_i^* - h_i)P_{pub}$
$\Rightarrow (W^* - W) \cdot (h_i^* - h_i)^{-1} = c \cdot aP$
Therefore, $\mathcal{F}_{\mathcal{I}}$ solve the $EC - CDH$ problem as $acP = \frac{W^* - W}{h_i^* - h_i}$ using the algorithm $\mathcal{C}$ for given a random instance $\langle P, aP, cP \rangle \in G_q$.

**Analysis**: In order to assess the probability of success of the challenger $\mathcal{C}$. We assume that $\mathcal{F}_{\mathcal{I}}$ can ask $q_C$ create $(ID_i)$ queries, $q_E$ key-encapsulation queries and $q_{H_i}$ random oracle queries to $H_i$ $(0 \leq i \leq 3)$. We also assume that $\mathcal{F}_{\mathcal{I}}$ never repeats $H_i$ $(0 \leq i \leq 3)$ query with the same input.

- The success probability of the Create$(ID_i)$ query execution is $(1 - \frac{q_{H_0}}{q})^{q_C} \geq 1 - \frac{q_{H_0} \cdot q_C}{q}$.
- The success probability of the $H_2$ query execution is $(1 - \frac{q_{H_2}}{q})^{q_{H_2}} \geq 1 - \frac{q_{H_2}^2}{q}$.
- The success probability of the $H_3$ query execution is $(1 - \frac{q_{H_3}}{q})^{q_{H_3}} \geq 1 - \frac{q_{H_3}^2}{q}$.
- The success probability of the key encapsulation query execution is $\frac{q_E}{(1 - \frac{1}{q})} \geq q_E \cdot (1 + \frac{1}{q})$.
- The probability that $ID_i = ID_t$ is $\frac{1}{q_C}$.
- The probability that $\mathcal{F}_{\mathcal{I}}$ queries the partial private key of the target identity $ID_t$ is $\frac{q_{ppri}}{q_{H_0}}$.
- The probability that $\mathcal{F}_{\mathcal{I}}$ asks to query the set secret value of the target identity $ID_t$ is $\frac{q_{sv}}{q_{H_0}}$.

Thus, the success probability that $\mathcal{C}$ can win the EUF-CL-HSC-CMA-I game is

$$\varepsilon \ \geq q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{ppri}}{q_{H_0}}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot \varepsilon$$

. Therefore, the probability that $\mathcal{C}$ computes the solution of $EC - CDH$ problem is non-negligible, because $\varepsilon$ is non-negligible.

### 5.4 *Type-II Unforgeability*

**Theorem 4.** Suppose that the hash functions $H_i (i = 0, 1, 2, 3)$ are random oracles. If there exists a forger $\mathcal{F}_{\mathcal{II}}$ against the EUF-CL-HSC-CMA-II security of the CL-HSC scheme with advantage a non-negligible $\varepsilon$, asking $q_C$ create $(ID_i)$ queries, $q_E$ key-encapsulation queries, $q_{H_i}$ random oracle queries to $H_i$ $(0 \leq i \leq 3)$, $q_{sv}$ set-secret-value queries and $q_{pkR}$ public key replacement queries, then there exist an algorithm $\mathcal{C}$ that solves the $EC - CDH$ problem with the following advantage $\varepsilon$

$$\varepsilon \geq q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot \varepsilon$$

*Proof.* A challenger $\mathcal{C}$ is challenged with an instance of the EC-CDH problem. Given $\langle P, aP, bP \rangle \in G_q$, $\mathcal{C}$ must find $abP$. Let $\mathcal{F}_{\mathcal{II}}$ be a forger who is able to break the EUF-CL-HSC-CMA-II security of the CL-HSC scheme. $\mathcal{C}$ can utilize $\mathcal{F}_{\mathcal{II}}$ to compute the solution $abP$ of the EC-CDH instance by playing the following interactive game with $\mathcal{F}_{\mathcal{II}}$. To solve the EC-CDH, $\mathcal{C}$ chooses $s \in_R \mathbb{Z}_q^*$, sets the master public key $P_{pub} = sP$, where $P$ is the generator of the group $G_q$ and the hash functions $H_i (0 \leq i \leq 3)$ are treated as random oracles. The $\mathcal{C}$ sends the system parameter $\Omega = \{F_q, E/F_q, G_q, P, P_{pub} = sP, H_0, H_1, H_2, H_3\}$ and the master private key $s$ to $\mathcal{F}_{\mathcal{II}}$. In order to avoid the inconsistency between the responses to the hash queries, $\mathcal{C}$ maintains lists $L_i (0 \leq i \leq 3)$). It also maintains a list $L_k$ to maintain the list of issued private keys and public keys. $\mathcal{C}$ can simulate the Challenger's execution of each phase of the formal Game.

**Training Phase**: $\mathcal{F}_{\mathcal{II}}$ may make a series of polynomially bounded number of queries to random oracles $H_i (0 \leq i \leq 3)$ at any time and $\mathcal{C}$ responds as follows:
All the oracles and queries needed in the training phase are identical to those of the **Create($ID_i$) queries**, $H_0$ **queries**, $H_1$ **queries**, $H_2$ **queries**, $H_3$ **queries**, **Partial-Private-Key-Extract queries**, **Set-Secret-Value queries**, **Public-Key-Replacement queries**, **Symmetric Key Generation queries**, **Key Encapsulation queries** and **Key Decapsulation queries** in IND-CL-HSC-CCA2-II game.

**Forgery**: Eventually, $\mathcal{F}_{\mathcal{II}}$ returns a valid encapsulation $\langle \tau, \omega = (U, W), ID_t, ID_B \rangle$ on a arbitrary tag $\tau$, where $ID_t$ is the sender identity and $ID_B$ is the receiver

identity, to $\mathcal{C}$. The public key of the sender $ID_t$ should not be replaced during the training phase. The secret value of the target identity $ID_t$ should not be queried during the training phase. $\mathcal{C}$ searches the list $L_3$ and outputs another valid encapsulation $\langle \tau, \omega^* = (U, W^*), ID_t, ID_B \rangle$ with different $h_i^{'*}$ such that $h_i^{'*} = h_i$ on the same $\tau$ as done in forking lemma[21]. Thus, we can get $W \cdot P = R_t - e_t \cdot P_{pub} + U \cdot h_i P + P_t \cdot h_i bP$ and $W^* \cdot P = R_t - e_t \cdot P_{pub} + U \cdot h_i P + P_t \cdot h_i^{'*} bP$. Then if we subtract these two equations, we get following value.
$W^* \cdot P - W \cdot P = P_t \cdot h_i^{'*} bP - P_t \cdot h_i^{'} bP$
$\Rightarrow (W^* - W)P = P_t \cdot (h_i^{'*} - h_i) \cdot bP$
$\Rightarrow (W^* - W)P = aP \cdot (h_i^* - h_i)] \cdot bP$
$\Rightarrow (W^* - W) = a \cdot (h_i^* - h_i) \cdot bP$
$\Rightarrow (W^* - W) \cdot (h_i^* - h_i)^{-1} = a \cdot bP$
Therefore, $\mathcal{F_{II}}$ solve the $EC - CDH$ problem as $abP = \frac{W^* - W}{h_i^* - h_i}$ using the algorithm $\mathcal{C}$ for given a random instance $\langle P, aP, bP \rangle \in G_q$.

**Analysis**: In order to assess the probability of success of the challenger $\mathcal{C}$. We assume that $\mathcal{F_{II}}$ can ask $q_C$ create $(ID_i)$ queries, $q_E$ key-encapsulation queries, $q_{H_i}$ random oracle queries to $H_i$ $(0 \leq i \leq 3)$, $q_{sv}$ set-secret-value queries and $q_{pkR}$ public key replacement queries. We also assume that $\mathcal{F_{II}}$ never repeats $H_i$ $(0 \leq i \leq 3)$ query with the same input.

- The success probability of the Create$(ID_i)$ query execution is $(1 - \frac{q_{H_0}}{q})^{q_C} \geq 1 - \frac{q_{H_0} \cdot q_C}{q}$.
- The success probability of the $H_2$ query execution is $(1 - \frac{q_{H_2}}{q})^{q_{H_2}} \geq 1 - \frac{q_{H_2}^2}{q}$.
- The success probability of the $H_3$ query execution is $(1 - \frac{q_{H_3}}{q})^{q_{H_3}} \geq 1 - \frac{q_{H_3}^2}{q}$.
- The success probability of the key encapsulation query execution is $\frac{q_E}{(1 - \frac{1}{q})} \geq q_E \cdot (1 + \frac{1}{q})$.
- The probability that $ID_i = ID_t$ is $\frac{1}{q_C}$.
- The probability that $\mathcal{F_{II}}$ queries the secret value of the target identity $ID_t$ is $\frac{q_{sv}}{q_{H_0}}$.
- The probability that $\mathcal{F_{II}}$ asks to replace the public key of the target identity $ID_t$ is $\frac{q_{pkR}}{q_{H_0}}$.

Thus, the success probability that $\mathcal{C}$ can win the EUF-CL-HSC-CMA-II game is

$$\varepsilon \ \geq q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot \varepsilon$$

. Therefore, the probability that $\mathcal{C}$ computes the solution of $EC - CDH$ problem is non-negligible, because $\varepsilon$ is non-negligible.

## 6 Conclusions

In this report we first proposed a CL-HSC (Certificateless Hybrid Signcryption) scheme without pairing operations and provided its formal security. Our CL-HSC scheme is satisfied with confidentiality and unforgeability against Type I and Type II adversaries. Our scheme is also adopted to use ECC (Elliptic Curve Cryptography). Thus, our scheme has the benefit of ECC keys defined on an additive group with a 160-bit length as secure as the RSA keys with 1024-bit length. Therefore, our CL-HSC scheme can be practically applied into encryption key management for Advanced Metering Infrastructures or Wireless Sensor Networks.

## References


[1] A.W.Dent, *Hybrid signcryption schemes with outsider security*, Information Security and Privacy, 2005.

[2] A.W.Dent, *Hybrid signcryption schemes with insider security*, ACISP 2005, pp. 253-266, 2005.

[3] J.H.An, Y.Dodis and T.Rabin, *On the security of joint signature and encryption*, EUROCRYPT 2002, pp. 83-107, 2002.

[4] Y.Dodis, M.J.Freedman, S.Jarecki and S.Walfish, *Versatile padding schemes for joint signature and encryption*, CCS 2004, pp. 344-353, 2004.

[5] J.Malone-Lee and W.Mao, *Two birds one stone: Signcryption using RSA*, CT-RSA 2003, pp. 211-225, 2003.

[6] Barreto, P.S.L.M., Libert, B., McCullagh, N. and Quisquater, J.-J., *Efficient and provably-secure identity-based signatures and signcryption from bilinear maps*, ASIACRYPT 2005, pp. 515-532, 2005.

[7] Chen, L. and Malone-Lee, J., *Improved identity-based signcryption*, PKC 2005, pp. 362-379, 2005.

[8] Barbosa, M. and Farshim, P., *Certificateless signcryption*, ASIACCS 2008, pp. 369-372, 2008.

[9] Chow, S.S.M., Yiu, S.-M., Hui, L.C.K. and Chow, K.P., *Efficient forward and provably secure id-based signcryption scheme with public verifiability and public ciphertext authenticity*, ICISC 2003, pp. 352-369, 2004.

[10] Libert, B. and Quisquater, J.-J., *Efficient signcryption with key privacy from gap diffiehellman groups*, PKC 2004, pp. 187-200, 2004.

[11] Selvi, S.S.D., Vivek, S.S., Shukla, D. and Rangan Chandrasekaran, P., *Efficient and provably secure certificateless multi-receiver signcryption*, ProvSec 2008, pp. 52-67, 2008.

[12] S. Al-Riyami and K. Paterson, *Certificateless public key cryptography*, ASIACRYPT 2003, Vol. 2894, pp. 452-473, 2013.

[13] Y. Zheng, *Digital signcryption or how to achieve cost (Signature and encryption) << cost(Signature) + cost(Encryption)*, Crypto 1997, Vol. 1294, pp. 165-179, 1997.

[14] S. Sharmila Deva Selvi, S. Sree Vivek and C. P.Rangan, *Certificateless KEM and Hybrid Signcryption Scehems Revisited*, ISPEC 2010, pp. 294-307, 2010.

[15] F. Li, M. Shirase and T. Takagi, *Certificateless hybrid signcryption*, ISPEC 2009, pp. 112-123, 2009.

[16] A. Shamir, *Identity-based cryptosystems and signature schemes*, CRYPTO 84, pp. 47-53, 1985.

[17] V.S.Miller, *Use of elliptic curves in cryptography*, Crypto 85, pp. 417-426, 1985.

[18] N.Koblitz, *Elliptic curve cryptosystem*, Journal of Mathematics of Computations, Vol. 48, no. 177, pp. 203-209, 1987.

[19] D.Hankerson, A.Menezes and S.Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag, 2004.

[20] P. Szczechowiak, L. B. Oliveira, M. Scott, M. Collier, R. Dahab, *NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks*, WSN, LNCS 4913, pp. 305-320, 2008.

[21] D. Pointcheval and J. Stern, *Security arguments for digital signatures and blind signatures*, Journal of Cryptology, vol. 13, no. 3, pp. 361-396, 2000.