pCLSC-TKEM: a Pairing-free Certificateless Signcryption-tag Key Encapsulation Mechanism for a Privacy-Preserving IoT

Seung-Hyun Seo*, Jongho Won, Elisa Bertino****

*Department of Mathematics, Korea University Sejong Campus 2511 Sejong-ro, Sejong City 30019, KOREA.

**Computer Science Department, Purdue University, West Lafayette, IN, 47907, USA.

E-mail: `crypto77@korea.ac.kr`, `won12@purdue.edu`, `bertino@purdue.edu`

Received 1 October 2015; received in revised form 15 March 2016; accepted 1 May 2016

Abstract. Certificateless Signcryption Tag Key Encapsulation Mechanism (CLSC-TKEM) is an effective method for simultaneously providing key encapsulation and a digital signature on the tag. It has applications in several security services such as communication confidentiality, integrity, authentication, and non-repudiation. Moreover, because CLSC-TKEM is based on certificateless public key cryptography (CL-PKC), it has the advantage of not requiring public key certificates. In addition it does not suffer from the key escrow problem which is instead a major drawback of identity-based public key cryptography (ID-PKC). Unfortunately, current constructions of CLSC-TKEM rely on the use of bilinear pairing-based operations that are computationally very expensive for small IoT devices. In this paper, we present a new construction of CLSC-TKEM that does not require bilinear pairing operations. We refer to our new construction on pairing-free Certificateless Signcryption Tag Key Encapsulation Mechanism (pCLSC-TKEM). We also provide a simple construction for pairing-free certificateless hybrid signcryption by combining pCLSC-TKEM with a data encapsulation mechanism (DEM). We provide a security model for pCLSC-TKEM. Then, we prove that our pCLSC-TKEM is secure against both an adaptively chosen ciphertext attack and existential forgery in the random oracle model. We have implemented our pCLSC-TKEM construction and previous pairing-based CLSC-TKEM constructions in order to compare their performance. Our experimental results demonstrate that pCLSC-TKEM is much more efficient that previous pairing-based CLCS-TKEM constructions.

Keywords. Certificateless Public Key Cryptography, Key Encapsulation Mechanism, Signcryption, Elliptic Curve Cryptography

*This work was supported by Purdue Cyber Center, by the US National Science Foundation under grant CNS-1111512, and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIP) (No.2015R1C1A1A01052491).

1 Introduction

The development of powerful sensor technology and embedded systems is making possible a tight integration between the physical world and the cyber world, resulting in what is called the Internet-of-Things (IoT). IoT will enable ground-breaking applications, including personalized and preventive healthcare, smart grid and smart manufacturing, smart cities, environment protection, precision agriculture, and emergency management. However as in many such applications, sensors and embedded devices are deployed to collect and transmit data from the environments, privacy is a critical issue. Recent projects and analyses, such as Open Web Application Security Project (OWASP) IoT Top 10 Vulnerabilities Project [17], have shown that IoT systems have insufficient authentication, lack of encryption, and also raise major privacy concerns. Addressing privacy in IoT is challenging and requires developing and combining different security techniques to assure data confidentiality. However, even though many applicable such techniques exist, they need to be tailored to small heterogeneous devices, such as sensors and embedded devices.

One critical fundamental data protection technique is represented by signcryption. Signcryption [25] is considered an efficient approach for supporting both message encryption and authentication compared to methods involving independently manage encryption and signing. However, when signcryption is directly applied to messages transmitting large amounts of data, its performance decreases. Inspired by the concept of hybrid encryption, Dent [8] proposed the signcryption key encapsulation mechanism (SC-KEM) in order to improve the practical use of signcryption. By using the SC-KEM construction, a random session key is first encapsulated by a signcryption key encapsulation algorithm, then the data is encrypted by this session key using a symmetric encryption algorithm, and finally both the encapsulated session key and ciphertext are transmitted over the insecure channel.

Then, based on the SC-KEM construction, Bjørstad et al. [4] proposed the concept of signcryption tag-KEM (SC-TKEM), where a tag can be a random string used to refresh a key or a ciphertext. The most notable feature of SC-TKEM is that it combines key encapsulation and digital signatures into one efficient algorithm. In comparison to SC-KEM, SC-TKEM has the added functionality of guaranteeing the authenticity and integrity of the tag. By combining SC-TKEM with a data encapsulation mechanism (DEM), one can easily construct a hybrid signcryption scheme. However, initial approaches [4, 7, 8] to constructing SC-KEM and SC-TKEM rely on traditional public key cryptography (PKC). Therefore those approaches require a certificate management infrastructure for issuing, distributing, storing and revoking certificates.

In order to eliminate the need for certificate management infrastructures, recent schemes have been proposed that apply Certificateless Public Key Cryptography (CL-PKC) [1] to the hybrid encryption techniques [13, 12, 20]. Lippold et al. [13] proposed a direct construction for certificateless key encapsulation (CL-KEM) that uses a certificateless public key encryption scheme as a KEM. Li et al. [12] proposed a certificateless signcryption tag-KEM (CLSC-TKEM) scheme that combines the ideas of SC-TKEM and CL-PKC. The main advantage of CLSC-TKEM is to eliminate the overhead resulting from the certificate management and the key escrow problem, while maintaining the benefits of SC-TKEM. In CL-PKC, each user's complete private key is a combination of a partial private key generated by a Key Generation Center (KGC) and an additional secret value generated by the user. Compared to identity-based PKC (ID-PKC) [22], the advantage of CL-PKC is that the KGC is no longer susceptible to the key escrow problem, because the KGC is not responsible for the complete user private key. In CLSC-TKEM, the cryptographic operations, such as key encapsulation based on CL-PKC, can only be performed when a valid user holds both the partial private key and the secret value. Moreover, the special structure of CL-PKC allows a user to perform the key decapsulation operation of CLSC-TKEM without having to verify the public key of the sender via a public key certificate. CLSC-TKEM is thus an efficient scheme, since the key encapsulation and the digital signature functionality are supported without the need of a certificate management infrastructure.

Several CL-KEM [13] and CLSC-TKEMs [12, 20] have been proposed. However a major common drawback of all these schemes is that they are constructed based on bilinear pairing operations. CL-KEM [13] requires 3 pairing operations for the decapsulation algorithm. CLSC-TKEMs [12, 20] require 1 and 5 pairing operations for the encapsulation algorithm and decapsulation algorithm, respectively. Despite recent advances in implementation techniques, the computational costs required for pairing operations are still considerably higher in comparison to standard operations such as point multiplication in elliptic curves (EC) or modular exponentiation in the finite field. For example, a currently available implementation of NanoECC [24] libraries, that uses the MIRACL library, takes around 17.93s to compute one pairing operation and around 1.27s to compute one ECC point multiplication on a MICA2 (8MHz) mote. Therefore, schemes based on pairing operations are impractical for application in emerging IoT environments where devices have inherent resource constraints. Therefore, in order to improve the efficiency of current CLSC-TKEMs so to be suitable for resource-constrained IoT devices, it is necessary to investigate how the CLSC-TKEM can be constructed without the use of bilinear paring operations.

1.1 Our Contributions

Privacy is a critical issue in IoT. In order to address privacy, data confidentiality is critical which in turn requires suitable data encryption mechanisms [11]. Such mechanisms must be highly efficient in order to run on a large variety of devices with limited computing capacity. In this paper, to address such requirement, we propose a novel pairings-free Certificateless Signcryption Tag-Key Encapsulation Mechanism (pCLSC-TKEM) that does not use bilinear pairing operations. We also present a simple construction for a pairing-free certificateless hybrid signcryption (pCL-HSC) scheme by combining pCLSC-TKEM with a data encapsulation mechanism (DEM). Then, we provide the formal security proof of our pCLSC-TKEM against both an adaptively chosen ciphertext attack and existential forgery. Since our pCLSC-TKEM is designed based on the elliptic curve cryptography (ECC), it inherits all the advantages of using ECC keys defined on an additive group with 160-bit length, which provides the same security of RSA keys with 1024-bit length. In order to show the efficiency of our scheme, we implemented both our scheme and current pairing-based certificateless hybrid encryption techniques such as CL-KEM [13] and CLSC-TKEMs [12, 20] on a PC, a Raspberry Pi 2 model B [21], and an Android smartphone platform and performed an experimental comparison among all the implemented schemes. Compared to previous schemes, pCLSC-TKEM dramatically reduces the computational overhead by eliminating the pairing operations. On the PC platform, the measured running time of our pCLSC-TKEM is at least 5.6 times faster than the running times of other schemes for RSA 1024-bit security. Our scheme is also at least 3.3 times faster than other schemes for RSA 3072-bit security. In the Raspberry Pi 2 platform, our scheme is at least 5.9 and 3.5 times faster than other schemes for RSA 1024-bit and 3072-bit security, respectively. Moreover, on the Android smartphone platform, our pCLSC-TKEM is at least 2.6 times faster for RSA 1024-bit security and at least 15.3 times faster for RSA 2048-bit security than the other schemes, respectively. pCLSC-TKEM is significantly more efficient when sharing symmetric encryption keys for secure end-to-end communications, and in supporting non-repudiation services.

1.2 Organization

The remainder of this paper is organized as follows: In Section 2, we review previous work related to pCLSC-TKEM. In Section 3, we briefly present an overview of ECC and then introduce the definition of pCLSC-TKEM and DEM, and a security model for pCLSC-TKEM. In Section 4, we introduce our pairing-free Certificateless Signcryption Tag-KEM and show a construction of pairing-free Certificateless Hybrid Signcryption. In Section 5, we provide a formal security proof of pCLSC-TKEM. In Section 6, we report the experimental results and performance measurements of pCLSC-TKEM, including the comparison with other schemes. In Section 7, we outline our conclusions.

2 Related Work

In this section, we thoroughly review previous work related to our pCLSC-TKEM. Beginning from its introduction by Zheng [25], the signcryption scheme has been considered as a very efficient method for supporting both message encryption and digital signatures compared to conventional approaches based on the independent encryption and signature of the message. In order to enhance the efficiency and practical use of signcryption, Dent [7, 8] introduced the concept of the hybrid signcryption scheme. The hybrid signcryption scheme consists of a signcryption KEM and a signcryption DEM like a usual KEM-DEM construction of the hybrid technique. However, the unsigncryption algorithm by Dent is complex due to the need to verify the link between the message and the encapsulated key. He also proposed the use of a signcryption KEM as a key agreement mechanism for two parties. However, the general signcryption KEM does not provide any form of authentication and does not guarantee key freshness. So, it is susceptible to known key attacks. Later on, Bjørstad et al. [4] proposed a signcryption tag-KEM, where a tag (e.g., a random string used to refresh a key) is taken as input. Signcryption tag-KEMs may be combined with a regular DEM (e.g. standard symmetric encryption algorithm such as AES) to build a hybrid signcryption scheme. Signcryption tag-KEMs also automatically support the transmission of the associated data along with messages through the tag. Since the encapsulation acts as a signature on the input tag, the authenticity and integrity of both the ciphertext and the associated data is guaranteed. Due to the nature of the components that provide authentication and freshness, the signcryption tag-KEM can be utilized as a practical and simple key agreement protocol. So far, several hybrid signcryption schemes and signcryption tag-KEMs have been proposed [7, 8, 4, 2, 9, 18]. A formal security model for tag-KEM was given by Dent [7]. However, since these approaches rely on a traditional PKI they require the management of certificates. Identity-based Public Key Cryptography (ID-PKC) [22] eliminates the dependency on explicit certificates. However, it suffers from the key escrow problem because the Key Generation Center (KGC) stores the private keys of all the users. In order to address these drawbacks, Al-Riyami et al. [1] introduced Certificateless Public Key Cryptography (CL-PKC), that separates the user's private key into two parts: one is a partial private key generated by the KGC, and the other one is a secret value selected by the user. CL-PKC is able to address 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 signature, can be performed.

More recently, researchers have investigated the application of CL-PKC to hybrid techniques [13, 12, 20]. Lippold et al. [13] first presented a direct construction for certificateless key encapsulation (CL-KEM). They use a certificateless public key encryption scheme as a KEM and claimed that their CL-KEM is secure against chosen ciphertext attacks (CCA). However, Selvi et al. identified several security weaknesses of the Lippold et al.'s scheme in [20]. Li et al. [12] were the first to construct a hybrid signcryption scheme which is truly certificateless by using a 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. [12] claimed that their scheme is secure against adaptive chosen ciphertext attacks and that it is existentially unforgeable. However, such schemes were proven to be existentially forgeable and therefore the definition of the generic scheme is insufficient. Selvi et al. [20] showed the security weaknesses of Li et al.'s scheme [12] and presented an improved certificateless hybrid signcryption scheme. However, existing certificateless based hybrid techniques such as CL-KEM [13] and CLSC-TKEM [12, 20] employ bilinear pairing operations. Despite recent advances in implementation techniques, the computational costs required for pairing operations are still considerably higher in comparison to standard operations such as ECC point multiplication. In this paper, we present the first concrete construction of CLSC-TKEM without bilinear pairing operations. In Section 6, we show a performance comparison between our construction and existing pairing-based certificateless hybrid techniques. By removing the pairing based operations, we dramatically reduce the computational overheads of CLSC-TKEM in comparison to [13, 12, 20].

3 Preliminaries

3.1 Elliptic Curve Cryptography

The security of Elliptic Curve Cryptography (ECC) [14] is based on the hardness of solving the Discrete Logarithm Problem (DLP) called Elliptic Curve Discrete Logarithm Problem (ECDLP). The primary benefit of ECC is a smaller key size, reducing storage and transmission requirements. For example, a 160-bit ECC is considered to be as secure as 1024-bit RSA key. 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:

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

where $a, b, x, y \in F_q$ and $4a^3 + 27b^2 \bmod q \neq 0$. A point $P(x, y)$ is an elliptic curve point if it satisfies equation 1. 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 defined by equation (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 $nP = O$.

3.2 Definitions

[Definition 1] The Pairing-free Certificateless Signcryption Tag-KEM (pCLSC-TKEM) is a 8-tuple pCLSC-TKEM=(SetUp, SetUserKey, PartialPrivateKeyExtract, SetPrivateKey, SetPublicKey, SymmetricKeyGen, Encapsulation, Decapsulation). The description of each probabilistic polynomial time algorithm is as follows.

1. SetUp: The Key Generation Center (KGC) runs this algorithm, which takes a security parameter k as input and returns the system parameters `params` and a master secret key `msk` for the KGC. We assume that `params` are publicly available to all users whereas `msk` is kept secret by the KGC.

2. SetUserKey: This algorithm is run by each user A to generate a secret value and the corresponding public value for oneself. It takes `params` and an identity ID_A of the user A as inputs, and returns the user's secret value x_A and a corresponding public value P_A.

3. PartialPrivateKeyExtract: The KGC runs this algorithm to generate the partial private key of the users. This algorithm takes `params`, `msk`, an identity ID_A and a public value P_A of A as inputs. It returns the partial private key d_A and the partial public key R_A of user A. The KGC runs this algorithm for each user, and we assume that the partial private key is securely transmitted to the user.

4. SetPrivateKey: This algorithm is run by each user A to generate the full private key. It takes `params`, the partial private key d_A and the secret value x_A of ID_A as inputs. It returns the full private key sk_A for A.

5. 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, a user's public value P_A, a partial private key R_A as the inputs. It returns the full public key pk_A to A as the output.

6. SymmetricKeyGen: This algorithm is run by the sender A to obtain the symmetric key K and an internal state information ω, which is not known to the 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. It returns to A the symmetric key K and ω.

7. Encapsulation: This algorithm is executed by the sender A to obtain the encapsulation φ. It takes a state information ω corresponding to K and an arbitrary tag τ as inputs. It returns the encapsulation of K, φ.

8. Decapsulation: This algorithm is executed by the receiver B to obtain the key K encapsulated in φ. It takes the encapsulation φ, a tag τ, 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 inputs. It returns the symmetric key K or a symbol $\perp$ indicating an invalid encapsulation with respect to the validity of φ as the output.

The following consistency constraint is defined. Let $(K, \omega) =$ SymmetricKeyGen(`params`, $ID_A, pk_A, sk_A, ID_B, pk_B$) and $\varphi =$ Encapsulation(ω, τ). Then $K =$ Decapsulation(`params`, $ID_A, pk_A, ID_B, pk_B, sk_B, \varphi, \tau$).

[Definition 2] Data Encapsulation Mechanism (DEM) is a symmetric encryption scheme, $\mathsf{DEM} = (\mathsf{ENC}, \mathsf{DEC})$, which consists of the following algorithms [4].

1. ENC: A symmetric encryption algorithm that takes a symmetric key $K \in \mathcal{K}$ and a message $m \in \{0,1\}^*$ as inputs, and returns a ciphertext $c \in \{0,1\}^*$, where $\mathcal{K}$ is the given key space of DEM.

2. DEC: A symmetric decryption algorithm which takes a symmetric key $K \in \mathcal{K}$ and a ciphertext $c \in \{0,1\}^*$ as inputs, and returns a message $m = \mathsf{DEC}_K(c)$.

For soundness, the symmetric encryption algorithm and decryption algorithm should be each other's inverse under a fixed key K. That is, $m = \mathsf{DEC}_K(\mathsf{ENC}_K(m))$. For the purposes of this paper, it is only required that a DEM is secure with respect to indistinguishability against passive attackers [4].

[Definition 3] The One-sided Gap Diffie-Hellman Problem (OGDH) [10] for a group G_q with a generator P and a fixed point Q is defined as follows: for $x, y \in \mathsf{Z}_q^*$, given Q, R, compute xyP by accessing an One-sided Decision Diffie-Hellman (ODDH) Oracle, where $Q = xP$ and $R = yP$.

[Definition 4] The One-sided Decision Diffie-Hellman Oracle (ODDH) [10] for a group G_q with a generator P and a fixed point Q is an oracle that for any $R', S' \in \mathsf{G}_q$ correctly answers the question: Is $z' \equiv xy'$ (mod p), where $x, y', z' \in \mathsf{Z}_q^*$ are integers such that $Q = xP, R' = y'P, S' = z'P$?

[Definition 5] The Elliptic Curve Discrete Logarithm Problem (ECDLP) is defined as follows: given a random instance P, Q, find a number $x \in \mathsf{Z}_q^*$ such that $Q = xP$.

The security of ECC depends on the hardness of solving ECDLP.

3.3 Security Model for Certificateless Signcryption Tag-KEM

Barbosa et al. [3] were the first to formalize the security model for certificateless signcryption scheme. Then, Selvi et al. [20] defined the security model for the certificateless signcryption tag-KEM (CLSC-TKEM). We adopted the security model of CLSC-TKEM as that of pCLSC-TKEM. A pCLSC-TKEM must satisfy confidentiality, that is, indistinguishability against adaptive chosen ciphertext and identity attacks (IND-CCA2), and unforgeability, that is, existential unforgeability against adaptive chosen messages and identity attacks (EUF-CMA). In order to prove the confidentiality and the unforgeability of pCLSC-TKEM, we must consider two types of adversaries: Type I and Type II. A Type I adversary is a model of an attacker which is a user of the system but does not possess the KGC's master secret key. However, the attacker is able to adaptively replace a users' public keys with a valid public keys of its choosing. A Type II adversary models an honest-but-curious KGC which has knowledge of the KGC's master secret key. However, it is unable to replace the users' public keys. For the confidentiality, we consider two games "IND-pCLSC-TKEM-CCA2-I" and "IND-pCLSC-TKEM-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 the two games "EUF-pCLSC-TKEM-CMA-I" and "EUF-pCLSC-TKEM-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. Note that the "Challenger $\mathcal{C}$" keeps a history of the "query-answers" while interacting with adversaries or forgers. Now we describe these games below.

3.3.1 Confidentiality

A pCLSC-TKEM satisfies indistinguishability against chosen ciphertext and identity attacks (IND-pCLSC-TKEM-CCA2), if there are no polynomially bounded adversaries $\mathcal{A}_{\mathcal{I}}$ and $\mathcal{A}_{\mathcal{II}}$ that have non-negligible advantages in both IND-pCLSC-TKEM-CCA2-I and IND-pCLSC-TKEM-CCA2-II games between $\mathcal{C}$ and $\mathcal{A}_{\mathcal{I}}$, $\mathcal{A}_{\mathcal{II}}$, respectively.

1. **IND-pCLSC-TKEM-CCA2-I Game**: This is the game in which a Type I adversary $\mathcal{A}_{\mathcal{I}}$ interacts with a challenger $\mathcal{C}$. The challenger $\mathcal{C}$ runs the $\mathsf{SetUp()}$ algorithm to generate the public parameters `params` and the master private key `msk` respectively. $\mathcal{C}$ gives `params` to $\mathcal{A}_{\mathcal{I}}$ while keeping `msk` secret.

   Phase I: In Phase I, $\mathcal{A}_{\mathcal{I}}$ may perform a polynomially bounded number of the following queries in an adaptive fashion.

   - **Extract-Partial-Private-Key queries**: $\mathcal{A}_{\mathcal{I}}$ chooses an identity ID_U for an user U. $\mathcal{C}$ first computes $(x_U, P_U) \leftarrow \mathsf{SetUserKey}(\mathtt{params}, ID_U)$ and then computes $D_U \leftarrow \mathsf{PartialPrivateKeyExtract}(\mathtt{params}, \mathtt{msk}, ID_U, P_U)$. Then $\mathcal{C}$ sends D_U to $\mathcal{A}_{\mathcal{I}}$.
   - **Extract-Secret-Value queries**: $\mathcal{A}_{\mathcal{I}}$ chooses an identity ID_U and requests the secret value of ID_U. $\mathcal{C}$ computes $(x_U, P_U) \leftarrow \mathsf{SetUserKey}(\mathtt{params}, ID_U)$ and sends x_U to $\mathcal{A}_{\mathcal{I}}$. The adversary cannot query any identity for which the corresponding public key has been replaced. That is, If $\mathcal{A}_{\mathcal{I}}$ has already replaced the public key of U, then $\mathcal{C}$ does not provide the corresponding secret value to $\mathcal{A}_{\mathcal{I}}$.
   - **Request-Public-Key queries**: $\mathcal{A}_{\mathcal{I}}$ presents an identity ID_U to $\mathcal{C}$ and requests ID_U's full public key. $\mathcal{C}$ returns the full public key pk_U for user ID_U by computing $pk_U \leftarrow \mathsf{SetPublicKey}(\mathtt{params}, ID_U, P_U, R_U)$.
   - **Public-Key-Replacement queries**: $\mathcal{A}_{\mathcal{I}}$ may replace the public key pk_U corresponding to the user identity ID_U with any value pk'_U of $\mathcal{A}_{\mathcal{I}}$'s choice.
   - **Symmetric Key Generation queries**: $\mathcal{A}_{\mathcal{I}}$ chooses a sender's identity ID_A and a receiver's identity ID_B. $\mathcal{C}$ obtains the private key of the sender, sk_A from the corresponding "query-answer" list. Then, $\mathcal{C}$ computes the symmetric key K and an internal state information ω by running $(K, \omega) \leftarrow \mathsf{SymmetricKeyGen}(\mathtt{params}, ID_A, pk_A, sk_A, ID_B, pk_B)$ and stores ω while keeping the ω secret from the view of $\mathcal{A}_{\mathcal{I}}$. Finally, $\mathcal{C}$ sends the symmetric key K to $\mathcal{A}_{\mathcal{I}}$. It is to be noted that $\mathcal{C}$ may not obtain the sender's secret value if the associated public value of the sender A is replaced. In this case, $\mathcal{A}_{\mathcal{I}}$ is required to provide the secret value of A to $\mathcal{C}$. We do not allow queries where $ID_A = ID_B$.
   - **Key Encapsulation queries**: $\mathcal{A}_{\mathcal{I}}$ produces an arbitrary tag τ for sender A. $\mathcal{C}$ checks whether there exists a corresponding ω value. If ω had been previously stored, then $\mathcal{C}$ computes $(\varphi) \leftarrow \mathsf{Encapsulation}(\omega, \tau)$, deletes ω and returns φ to $\mathcal{A}_{\mathcal{I}}$. Otherwise, $\mathcal{C}$ returns $\perp$ and terminates.

- **Key Decapsulation queries**: $\mathcal{A_I}$ produces an encapsulation φ, a tag τ, the sender's identity ID_A, the public key pk_A, the receiver's identity ID_B and the public key pk_B. $\mathcal{C}$ obtains the receiver's private key sk_B from the corresponding "query-answer" list. $\mathcal{C}$ returns the key K by computing $K \leftarrow \mathsf{Decapsulation}(\texttt{params}, ID_A, pk_A, ID_B, pk_B, sk_B, \varphi, \tau)$. It is to be noted that $\mathcal{C}$ may not be aware of the corresponding secret value if the associated public value of ID_B is replaced. In this case $\mathcal{A_I}$ requires to provide the secret value of B to $\mathcal{C}$. We do not allow the queries where $ID_A = ID_B$.

Challenge: At the end of Phase I decided by $\mathcal{A_I}$, $\mathcal{A_I}$ generates a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A_I}$ wishes to be challenged. Here, it is to be noted that ID_{B^*} must not be queried to extract a full private key sk_{B^*} in Phase 1. It is also to be noted that ID_{B^*} may not be equal to an identity for which both the public key has been replaced and the partial private key has been extracted. Now, $\mathcal{C}$ computes $(K_1, \omega^*) \leftarrow \mathsf{SymmetricKeyGen}(\texttt{params}, ID_{A^*}, pk_{A^*}, sk_{A^*}, ID_{B^*}, pk_{B^*})$ and chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the pCLSC-TKEM scheme. Now $\mathcal{C}$ chooses a bit $\delta \in_R \{0,1\}$ and sends K_δ to $\mathcal{A_I}$. $\mathcal{A_I}$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$. $\mathcal{C}$ computes $(\varphi^*) \leftarrow \mathsf{Encapsulation}(\omega^*, \tau^*)$ and sends φ^* to $\mathcal{A_I}$ as a challenge encapsulation.

Phase II: $\mathcal{A_I}$ can perform a polynomially bounded number of queries adaptively as in Phase I. However, $\mathcal{A_I}$ may not extract the full private key for ID_{B^*}. If the public key of ID_{B^*} has been replaced before the challenge phase, $\mathcal{A_I}$ may not extract the partial private key for ID_{B^*}. Moreover, $\mathcal{A_I}$ may not make a key decapsulation query on (K_δ, φ^*) under ID_{A^*} and ID_{B^*}, unless the public key $pk_{ID_{A^*}}$ or $pk_{ID_{B^*}}$ has been replaced after the challenge phase.

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

The advantage of $\mathcal{A_I}$ is defined as $Adv(\mathcal{A_I}) = |2Pr[\delta' = \delta] - 1|$, where $Pr[\delta' = \delta]$ denotes the probability that $\delta' = \delta$.

2. **IND-pCLSC-TKEM-CCA2-II Game**: This is the game in which a Type II adversary $\mathcal{A_{II}}$ interacts with a challenger $\mathcal{C}$. The challenger $\mathcal{C}$ runs the $\mathsf{SetUp}()$ algorithm to generate the public parameters `params` and the master private key `msk`, respectively. $\mathcal{C}$ gives both `params` and `msk` to $\mathcal{A_{II}}$.

   Phase I: In Phase I, $\mathcal{A_{II}}$ performs a series of queries in an adaptive fashion. The queries allowed are similar to the queries in the IND-pCLSC-TKEM-CCA2-I game except that Extract-Partial-Private-Key queries are not included here, because $\mathcal{A_{II}}$ can generate a partial private key by itself because it knows `msk`.

   Challenge: At the end of Phase I decided by $\mathcal{A_{II}}$, $\mathcal{A_{II}}$ generates a sender identity ID_{A^*} and a receiver identity ID_{B^*} on which $\mathcal{A_{II}}$ wishes to be challenged. Here, it is to be noted that ID_{B^*} must not be queried to extract a full private key sk_{B^*} in Phase I. Now, $\mathcal{C}$ computes $(K_1, \omega^*) \leftarrow \mathsf{SymmetricKeyGen}(\texttt{params}, ID_{A^*}, pk_{A^*}, sk_{A^*}, ID_{B^*}, pk_{B^*})$ and chooses $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the pCLSC-TKEM scheme. Now $\mathcal{C}$ chooses a bit $\delta \in_R \{0,1\}$ and sends K_δ to $\mathcal{A_{II}}$. $\mathcal{A_{II}}$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$. $\mathcal{C}$ computes $(\varphi^*) \leftarrow \mathsf{Encapsulation}(\omega^*, \tau^*)$ and sends φ^* to $\mathcal{A_{II}}$ as a challenge encapsulation.

Phase II: $\mathcal{A}_{\mathcal{II}}$ can perform a polynomially bounded number of queries adaptively, again as in Phase I. However, $\mathcal{A}_{\mathcal{II}}$ may not make Extract-full-Private-Key queries on ID_{B^*}. Moreover, $\mathcal{A}_{\mathcal{I}}$ may not make a key decapsulation query on (K_δ, φ^*) under ID_{A^*} and ID_{B^*}, unless the public key $pk_{ID_{A^*}}$ or $pk_{ID_{B^*}}$ has been replaced after the challenge phase.

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

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

3.3.2 Existential Unforgeability

A pCLSC-TKEM scheme satisfies existential unforgeability against an adaptively chosen message attack (EUF-pCLSC-TKEM-CMA), if no polynomially bounded forgers $\mathcal{F}_{\mathcal{I}}$ and $\mathcal{F}_{\mathcal{II}}$ have a non-negligible advantage in both "EUF-pCLSC-TKEM-CMA-I" and "EUF-pCLSC-TKEM-CMA-II" games between $\mathcal{C}$ and $\mathcal{F}_{\mathcal{I}}$, $\mathcal{F}_{\mathcal{II}}$ respectively:

1. **EUF-pCLSC-TKEM-CMA-I Game**: This is the game in which a Type I Forger $\mathcal{F}_{\mathcal{I}}$ interacts with a challenger $\mathcal{C}$. The challenger $\mathcal{C}$ runs the SetUp() algorithm to generate the public parameters `params` and the master private key `msk` respectively. $\mathcal{C}$ gives `params` to $\mathcal{F}_{\mathcal{I}}$ while keeping the `msk` secret.

   Training Phase: $\mathcal{F}_{\mathcal{I}}$ may make a 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 the queries allowed in Phase I of the IND-pCLSC-TKEM-CCA2-I game.

   Forgery: A the end of the Training Phase, $\mathcal{F}_{\mathcal{I}}$ produces an encapsulation $\langle \tau^*, \varphi^*, ID_{A^*}, ID_{B^*} \rangle$ on a arbitrary tag τ^*, where ID_{A^*} is the sender identity and ID_{B^*} is the receiver identity. Then, $\mathcal{F}_{\mathcal{I}}$ sends $\langle \tau^*, \varphi^*, ID_{A^*}, ID_{B^*} \rangle$ to $\mathcal{C}$. During the Training Phase, the partial private key for ID_{A^*} must not be queried and the public key for ID_{A^*} must not be replaced simultaneously. Moreover φ^* must not be returned by the key encapsulation oracle on the input $(\tau^*, \omega^*, ID_{A^*}, ID_{B^*})$ during the Training Phase. If the output of Decapsulation(`params`, $ID_{A^*}, pk_{A^*}, ID_{B^*}, pk_{B^*}, sk_{B^*}, \varphi^*, \tau^*$) is valid, $\mathcal{F}_{\mathcal{I}}$ wins the game. The advantage of $\mathcal{F}_{\mathcal{I}}$ is defined as the probability with which it wins the EUF-pCLSC-TKEM-CMA-I game.

2. **EUF-pCLSC-TKEM-CMA-II Game**: This is the game in which a Type II Forger $\mathcal{F}_{\mathcal{II}}$ interacts with a challenger $\mathcal{C}$. The challenger $\mathcal{C}$ runs the SetUp() algorithm to generate the public parameters `params` and the master private key `msk`, respectively. $\mathcal{C}$ gives `params` and the master private key `msk` to $\mathcal{F}_{\mathcal{II}}$.

   Training Phase: $\mathcal{F}_{\mathcal{II}}$ may make a 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 the queries allowed in Phase I of the IND-pCLSC-TKEM-CCA2-II game.

Forgery: A the end of the Training Phase, $\mathcal{F}_{\mathcal{II}}$ produces an encapsulation $\langle \tau^*, \varphi^*, ID_{A^*}, ID_{B^*} \rangle$ on a arbitrary tag τ^*, where ID_{A^*} is the sender identity and ID_{B^*} is the receiver identity. Then, $\mathcal{F}_{\mathcal{I}}$ sends $\langle \tau^*, \varphi^*, ID_{A^*}, ID_{B^*} \rangle$ to $\mathcal{C}$. During the Training Phase, the secret value x_{A^*} for ID_{A^*} must not be queried and the public key for ID_{A^*} must not be replaced, simultaneously. Moreover, φ^* must not be returned by the key encapsulation oracle on the input $(\tau^*, \omega^*, ID_{A^*}, ID_{B^*})$ during the Training Phase. If the output of Decapsulation($\texttt{params}, ID_{A^*}, pk_{A^*}, ID_{B^*}, pk_{B^*}, sk_{B^*}, \varphi^*, \tau^*$) is valid, $\mathcal{F}_{\mathcal{II}}$ wins the game. The advantage of $\mathcal{F}_{\mathcal{II}}$ is defined as the probability with which it wins the EUF-pCLSC-TKEM-CMA-II game.

4 Pairing-free Certificateless Signcryption Tag KEM

In this section, we present a novel pairing-free Certificateless Signcryption Tag KEM (pCLSC-TKEM). Then, we provide an example of how to construct a certificateless hybrid signcryption by combining pCLSC-TKEM with a DEM.

4.1 A Concrete pCLSC-TKEM

The pCLSC-TKEM scheme consists of the following seven algorithms.

1. **Setup**

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

   (a) 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.

   (b) Choose the master key $x \in \mathsf{Z}_q^*$ uniformly at random and compute the system public key $P_{pub} = xP$.

   (c) Choose cryptographic hash functions $H_0 : \{0,1\}^* \times G_q^2 \to \{0,1\}^*$, $H_1 : G_q^3 \times \{0,1\}^* \times G_q \to \{0,1\}^n$, $H_2 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \to \mathsf{Z}_q^*$, and $H_3 : G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \times \{0,1\}^* \times G_q \to \mathsf{Z}_q^*$. Here, n is the key length of a DEM.

   (d) 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 secret.

2. **SetUserKey**

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

3. **PartialPrivateKeyExtract**

   This algorithm takes the KGC's master secret key, the identity of an entity and the system parameter as inputs. 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 KGC. KGC the execute the following steps:

   (a) Choose $r_A \in \mathsf{Z}_q^*$ uniformly at random and compute $R_A = r_A P$.

(b) 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 its private key by checking whether $d_A P = R_A + H_0(ID_A, R_A, P_A)P_{pub}$ holds.

4. **SetPrivateKey**

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

5. **SetPublicKey**

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

6. **SymmetricKeyGen**

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

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

7. **Encapsulation**

   Given a state information ω and an arbitrary tag τ, the sender A obtains the encapsulation φ by performing the following steps:

   (a) 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 $W = d_A + l_A \cdot H + x_A \cdot H'$
   (b) Output $\varphi = (U, V, W)$.

8. **Decapsulation**

   Given the encapsulation φ, a tag τ, 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:

   (a) Compute $T = d_B \cdot V \; (= (r_B + xH_0(ID_B, R_B, P_B)) \cdot s_A P \bmod q = s_A \cdot H_0(ID_B, R_B, P_B) \cdot P_{pub} + s_A \cdot R_B \bmod q)$.
   (b) Compute $H = H_2(U, \tau, T, ID_A, P_A, ID_B, P_B)$ and $H' = H_3(U, \tau, T, ID_A, P_A, ID_B, P_B)$.
   (c) 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 $K = H_1(U, T, x_B \cdot V, ID_B, P_B)$. Otherwise, output invalid.
   The correctness of the above equation is as follows:

$$\begin{aligned} W \cdot P &= (d_A + l_A \cdot H + x_A \cdot H') \cdot P \\ &= d_A \cdot P + l_A \cdot P \cdot H + x_A \cdot P \cdot H' \\ &= (r_A + xH_0(ID_A, R_A, P_A)) \cdot P + U \cdot H + H' \cdot P_A \\ &= R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A \end{aligned}$$

4.2 A Construction of Pairing-Free Certificateless Hybrid Signcryption

If we combine the pCLSC-TKEM scheme with a DEM, we can construct a pairing-free certificateless hybrid signcryption (pCL-HSC) scheme. Such pCL-HSC scheme consists of the following eight algorithms: Setup, SetUser-Key, PartialPrivateKeyExtract, SetPrivateKey, SetPublic-Key, SymmetricKeyGen, Signcryption and Unsigncryption. Except for the Signcryption and Unsigncryption algorithms, all the algorithms are the same as the algorithms of pCLSC-TKEM (see Section 4.1). Thus, in this section we only describe the Signcryption and Unsigncryption algorithms. The Signcryption algorithm consists of the SymmetricKey-Gen and the Encapsulation algorithms of pCLSC-TKEM, and the ENC algorithm of DEM. The Unsigncryption algorithm consists of the Decapsulation algorithm of pCLSC-TKEM, and the DEC algorithm of DEM.

- **Signcryption** Given the parameter Ω, a message $m \in \{0,1\}^*$, the sender A's identity ID_A, the full public key pk_A, the full private key sk_A, the receiver B's identity ID_B and the full public key pk_B as inputs, the following steps are executed:

  1. Compute $(K, \omega) \leftarrow \mathsf{SymmetricKeyGen}(\texttt{params}, ID_A, pk_A, sk_A, ID_B, pk_B)$.
  2. Compute $c \leftarrow \mathsf{ENC}_K(m)$.
  3. Compute $\varphi \leftarrow \mathsf{Encapsulation}(\omega, c)$.
  4. Output the ciphertext (φ, c).

- **Unsigncryption** Given the parameter Ω, a ciphertext (φ, c), the sender A's identity ID_A, full public key pk_A, the receiver B's identity ID_B, full public key pk_B and full private key sk_B as inputs, the following steps are executed:

  1. Compute $K \leftarrow \mathsf{Decapsulation}(\texttt{params}, ID_A, pk_A, ID_B, pk_B, sk_B, \varphi, c)$.
  2. If K is invalid, then output $\perp$ and stop. Otherwise, compute $m \leftarrow \mathsf{DEC}_K(c)$.
  3. Output the message m.

5 Security Analysis

In this section, we provide the formal security proof for the confidentiality and the existential unforgeability of our pCLSC-TKEM.

5.1 Confidentiality

Theorem 1. *In the random oracle model, the pCLSC-TKEM is IND-CCA2 secure under the assumption that the One-sided Gap Diffie-Hellman (OGDH) problem is intractable.*
The Theorem 1 is proved based on Lemmas 1 and 2.

Lemma 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-pCLSC-TKEM-CCA2-I security of the pCLSC-TKEM with advantage a non-negligible ε, asking q_{ppri} extract-partial-private-key queries, q_{sv} extract-secret-value queries and q_{H_i} random oracle queries to H_i $(0 \leq i \leq 3)$, then an algorithm $\mathcal{C}$ exists that solves the OGDH problem with the following advantage ε'

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

Proof. A challenger $\mathcal{C}$ is challenged with an instance of the OGDH (One-sided Gap Diffie-Hellman) problem. The OGDH for a group G_q with a generator P and a fixed second point $Q(= aP)$ has as input $R(= bP) \in G_q$ and computes for the point $S(= cP) \in G$ such that $c = ab$ (mod q), by accessing a ODDH (One-sided Decision Diffie-Hellman) oracle. Here, the ODDH oracle solves the OGDH problem for a group G_q with a generator P and a fixed second point Q as input $R', S' \in G_q$ and decides whether $c' = ab' \bmod q$, where $a, b', c' \in Z_q^*$ such that $Q = aP, R' = b'P, S' = c'P$. Let $\mathcal{A}_{\mathcal{I}}$ be an adversary who is able to break the IND-pCLSC-TKEM-CCA2-I security of the pCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{A}_{\mathcal{I}}$ to compute the solution abP of the OGDH instance by playing the following interactive game with $\mathcal{A}_{\mathcal{I}}$. To solve the OGDH problem, $\mathcal{C}$ sets the master private/public key pair as $(x, P_{pub} = xP)$, 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. $\mathcal{C}$ 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 fix ID_t as the target identity for the challenge phase.

Phase 1: $\mathcal{A}_{\mathcal{I}}$ may make a series of polynomially bounded numbers 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 $e_t, x_t \in_R Z_q^*$ and sets $H_0(ID_t, R_t, P_t) = -e_t$, $R_t = e_t P_{pub} + aP$ and $P_t = x_t P$. Here, $\mathcal{C}$ does not know a. $\mathcal{C}$ uses the aP given in the instance of the OGDH problem. $\mathcal{C}$ inserts $\langle ID_t, R_t, P_t, -e_t\rangle$ into the list L_0 and $\langle ID_t, \perp, x_t, R_t, P_t\rangle$ into the list L_k.
- If $ID_i \neq ID_t$, $\mathcal{C}$ 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 $\langle ID_i, R_i, P_i, -e_i\rangle$ into the list L_0 and $\langle ID_i, d_i, x_i, R_i, P_i\rangle$ into 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 $\langle ID_i, R_i, P_i, -e_i\rangle$, $\mathcal{C}$ 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 $\langle ID_i, R_i, P_i, -e_i\rangle$ into the list L_0.

H_1 **queries**: When $\mathcal{A}_{\mathcal{I}}$ submits a H_1 query with $(V_i, T_i,$
$Y_i, ID_i, P_i)$, $\mathcal{C}$ checks whether the ODDH oracle returns 1 when queried with the tuple (aP, V_i, T_i). If the ODDH oracle returns 1, $\mathcal{C}$ outputs T_i and stop. Then $\mathcal{C}$ goes through the list L_1 with entries $\langle V_i, *, Y_i, ID_i, P_i, l_i\rangle$, for different values of l_i, such that the ODDH oracle returns 1 when queried on the tuple (aP, V_i, T_i). Note that in this case $ID_i = ID_t$. If such a tuple exists, it returns l_i and replaces the symbol $*$ with T_i. Otherwise, $\mathcal{C}$ chooses $l \in_R \{0,1\}^n$ and updates the list L_1, which is initially empty, with a tuple containing the input and return values. $\mathcal{C}$ then returns l to $\mathcal{A}_{\mathcal{I}}$.

H_2 **queries**: $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ exists in the list L_2. If it exists, $\mathcal{C}$ returns $H = h_i$ to $\mathcal{A}_{\mathcal{I}}$. Otherwise, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ to the list L_3 and returns $H = h_i$ to $\mathcal{A}_{\mathcal{I}}$.

H_3 **queries**: $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_i\rangle$ exists in the list L_3. If it exists, $\mathcal{C}$ returns $H' = h'_i$ to $\mathcal{A}_{\mathcal{I}}$. Otherwise, $\mathcal{C}$ chooses $h'_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_i\rangle$ to the list L_3 and returns $H' = h'_i$ to $\mathcal{A}_{\mathcal{I}}$.

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

- If $ID_i = ID_t$, $\mathcal{C}$ aborts the execution.
- If $ID_i \neq ID_t$, $\mathcal{C}$ 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 $d_iP = R_i + H_0(ID_i, R_i, P_i)P_{pub}$.

Extract-Secret-Value queries: $\mathcal{A_I}$ 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 $ID_i \neq ID_t$, then $\mathcal{C}$ responds with x_i by retrieving from the list L_k. If $\mathcal{A_I}$ has already replaced the public key, $\mathcal{C}$ does not provide the corresponding secret value to $\mathcal{A_I}$. If $ID_i = ID_t$, $\mathcal{C}$ aborts.
Request-Public-Key queries: $\mathcal{A_I}$ produces ID_i to $\mathcal{C}$ and requests a public key of the user with ID_i. $\mathcal{C}$ checks in the list 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: $\mathcal{A_I}$ 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 $\mathcal{A_I}$.
Symmetric Key Generation queries: $\mathcal{A_I}$ 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 $\mathcal{C}$. For each query (ID_A, ID_B), $\mathcal{C}$ proceeds as follows:

- If $ID_A \neq ID_t$, $\mathcal{C}$ computes the full private key sk_A corresponding to ID_A by executing the Extract-Partial-Private-Key query and Extract-Secret-Value query algorithm. Then, $\mathcal{C}$ gets the symmetric key K and an internal state information ω by running the actual SymmetricKeyGen algorithm. $\mathcal{C}$ stores ω and overwrite any previous value. $\mathcal{C}$ sends the symmetric key K to $\mathcal{A_I}$.
- If $ID_A = ID_t$ (and hence $ID_B \neq ID_t$), $\mathcal{C}$ chooses $r_1, r_2, h_t, h_t' \in_R Z_q^*$ and computes $U = r_1P - {h_t}^{-1} \cdot aP$, $V = r_2P$, $T = r_2 \cdot H_0(ID_B, R_B, P_B)P_{pub} + r_2 \cdot R_B \bmod q$ and $K = H_1(U, T, r_2 \cdot P_B, ID_B, P_B)$. Note that $\omega = (r_1, r_2, h_t, h_t', U, V, T, ID_A, pk_A, ID_B, pk_B)$.
- $\mathcal{C}$ goes through the list L_1 looking for an entry (V, T, Y, ID_B, P_B, k) for some k such that ODDH(P_B, V, Y)=1, where P_B is obtained by calling the Request-Public-Key query oracle on ID_B. If such an entry exists, it computes $K \leftarrow l$. Otherwise it uses a random l and updates the list L_1 with $(V, T, *, ID_B, P_B, l)$. $\mathcal{C}$ stores ω and sends the symmetric key K to $\mathcal{A_I}$.

Key Encapsulation queries: $\mathcal{A_I}$ produces an arbitrary tag τ, 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 them to $\mathcal{C}$. The full private key of the sender $sk_A = (d_A, x_A)$ is obtained from the list L_k. $\mathcal{C}$ checks whether a corresponding ω value has been stored previously.

- If ω does not exist, $\mathcal{C}$ returns an invalid reply.
- If a corresponding ω exists and $ID_A \neq ID_t$, then $\mathcal{C}$ computes φ with ω and τ by using the actual Encapsulation algorithm, and deletes ω.
- If a corresponding ω exists and $ID_A = ID_t$, then $\mathcal{C}$ computes φ by performing the following steps. Note that ω is $(r_1, r_2, h_t, h_t', U, V, T, ID_A, pk_A, ID_B, pk_B)$ and $\mathcal{C}$ does not know the private key corresponding to ID_t. So $\mathcal{C}$ should perform the encapsulation in a different way.:

1. Set $H = h_t$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t \rangle$ to the list L_2.
2. Set $H' = h'_t$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_t \rangle$ to the list L_3.
3. Compute $W = h_t \cdot r_1 + h'_t \cdot x_A$.
4. Output $\varphi = (U, V, W)$ as the encapsulation.

We show that $\mathcal{A}_{\mathcal{I}}$ can pass the verification of $\varphi = (U, V, W)$ 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:

$$\begin{aligned} & R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A \\ & = aP + e_t P_{pub} + (-e_t) \cdot P_{pub} + h_t \cdot (r_1 P - {h_t}^{-1} \cdot aP) + h'_t \cdot P_A \\ & = h_t \cdot r_1 P + h'_t \cdot P_A \\ & = W \cdot P \end{aligned}$$

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

- If $ID_B \neq ID_t$, then $\mathcal{C}$ computes the decapsulation of φ by using the actual Decapsulation algorithm.
- If $ID_B = ID_t$, then $\mathcal{C}$ computes K from φ as follows:
  1. Searches in the list L_2 and L_3 for entries of the type $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t \rangle$ and $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_t \rangle$ respectively.
  2. If entries $H = h_t$ and $H' = h'_t$ 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.
  3. 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.
  4. $\mathcal{C}$ goes through L_1 and looks for a tuple of the form $\langle V, T, Y, ID_B, P_B, l \rangle$ such that the ODDH oracle returns 1 when queried on the (aP, V, T). If such entry exists, the corresponding $K \leftarrow l$ value is returned as the decapsulation of φ.
  5. If $\mathcal{C}$ reaches this point of execution, it put the entry $\langle V, *, Y, ID_B, P_B, l \rangle$ for a random l on the list L_1 and returns $K \leftarrow l$. The symbol $*$ denotes an unknown value of Point. Note that the identity component of all entries with a $*$ is a receiver identity ID_B.

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

1. Choose $r \in_R Z_q^*$ and compute $U^* = rP$.
2. Set $V^* = bP$ and choose $T^* \in_R G_q$. Here, $\mathcal{C}$ does not know b. $\mathcal{C}$ uses the bP given in the instance of the OGDH problem.
3. Choose $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the pCLSC-TKEM.
4. Compute K_1 by executing Symmetric Key Generation queries.

5. Set $\omega^* = \langle -, U^*, V^*, T^*, ID_{A^*}, P_{A^*}, R_{A^*}, sk_{A^*}, ID_{B^*}, P_{B^*}, R_{B^*} \rangle$.

6. $\mathcal{C}$ chooses a bit $\delta \in_R \{0,1\}$ and sends K_δ to $\mathcal{A_I}$.

7. $\mathcal{A_I}$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$.

8. Choose $h_i, h_i' \in_R \mathsf{Z}_q^*$, store the tuple $\langle U, \tau^*, T, ID_A, P_A, ID_B, P_B, h_i \rangle$ to the list L_2 and $\langle U, \tau^*, T, ID_A, P_A, ID_B, P_B, h_i' \rangle$ to the list L_3.

9. Since $\mathcal{C}$ knows the private key of the sender, $\mathcal{C}$ computes $W^* = d_{A^*} + r \cdot h_i + x_{A^*} \cdot h_i'$.

10. $\mathcal{C}$ sends $\varphi^* = \langle U^*, V^*, W^* \rangle$ 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 φ^*.
Guess: Since $\mathcal{A_I}$ is able to break the IND-pCLSC-TKEM-CCA2-I security of pCLSC-TKEM (which is assumed at the beginning of the proof), $\mathcal{A_I}$ should have asked a H_1 query with $(V^*, T^*, x_{B^*} \cdot V^*, ID_{B^*}, P_{B^*})$ as inputs. It is to be noted that $T^* = b \cdot H_0(ID_B, R_B, P_B) \cdot P_{pub} + b \cdot R_{B^*} = b \cdot (-e_t) \cdot P_{pub} + b \cdot (e_t \cdot P_{pub} + aP) = ab \cdot P$. 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 OGDH 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 OGDH instance.
Analysis: In order to assess the probability of success of the challenger, $\mathcal{C}$ lets E_1, E_2 and E_3 be the events in which $\mathcal{C}$ aborts the IND-pCLSC-TKEM-CCA2-I game.

- E_1 is an event in which $\mathcal{A_I}$ 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 in which $\mathcal{A_I}$ 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 in which $\mathcal{A_I}$ 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-pCLSC-TKEM-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 OGDH problem is $\frac{1}{q_{H_1}}$. So, the probability that $\mathcal{C}$ finds the OGDH 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 ε is non-negligible.

Lemma 2. Suppose that the hash functions $H_i (i = 0, 1, 2, 3)$ are random oracles. If there exists an adversary $\mathcal{A_{II}}$ against the IND-pCLSC-TKEM-CCA2-II security of the pCLSC-TKEM with advantage a non-negligible ε, asking q_{sv} extract-secret-value queries, q_{pkR} public key replacement queries and q_{H_i} random oracle queries to H_i $(0 \le i \le 3)$, then there exist an algorithm $\mathcal{C}$ that solves the OGDH problem with the following advantage ε'.
$\varepsilon' \ge \varepsilon \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot (\frac{1}{q_{H_0} - q_{sv} - q_{pkR}}) \cdot (\frac{1}{q_{H_1}})$
Proof. A challenger $\mathcal{C}$ is challenged with an instance of the OGDH problem. Let $\mathcal{A_{II}}$ be an adversary who is able to break the IND-pCLSC-TKEM-CCA2-II security of the pCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{A_{II}}$ to compute the solution abP of the OGDH instance by playing

the following interactive game with $\mathcal{A}_{\mathcal{II}}$. To solve the OGDH, $\mathcal{C}$ chooses $s \in_R \mathsf{Z}_q^*$, 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. $\mathcal{C}$ 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 $\mathcal{A}_{\mathcal{II}}$. 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 of the 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 $1 \leq t \leq q_{H_0}$ and fixes ID_t as the target identity for the challenge phase.
Phase 1: $\mathcal{A}_{\mathcal{II}}$ 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:
Create(ID_i) queries: When $\mathcal{A}_{\mathcal{II}}$ submits a Create(ID_i) query, $\mathcal{C}$ responds as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ chooses $a_t, l_t \in_R Z_q^*$ and sets $H_0(ID_t, R_t, P_t) = l_t$, computes $R_t = a_tP$, $d_t = a_t + l_t \cdot s$ and the public key as $P_t = aP$. Here, $\mathcal{C}$ does not know a. $\mathcal{C}$ uses the aP given in the instance of the OGDH problem. $\mathcal{C}$ inserts $\langle ID_t, R_t, P_t, l_t\rangle$ into the list L_0 and $\langle ID_t, d_t, \perp, R_t, P_t\rangle$ into the list L_k.

- If $ID_i \neq ID_t$, $\mathcal{C}$ picks $a_i, x_i, l_i \in_R Z_q^*$, then sets $H_0(ID_i, R_i, P_i) = l_i$, computes $R_i = a_iP$, $d_i = a_i + l_i \cdot s$ and the public key as $P_i = x_iP$. $\mathcal{C}$ inserts $\langle ID_i, R_i, P_i, l_i\rangle$ into the list L_0 and $\langle ID_i, d_i, x_i, R_i, P_i\rangle$ into the list L_k.

H_0 **queries**: When $\mathcal{A}_{\mathcal{II}}$ submits a H_0 query with ID_i, $\mathcal{C}$ searches the list L_0. If there is a tuple $\langle ID_i, R_i, P_i, l_i\rangle$, $\mathcal{C}$ 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 $\langle ID_i, R_i, P_i, l_i\rangle$ into the list L_0.
H_1 **queries**: When $\mathcal{A}_{\mathcal{II}}$ submits a H_1 query with $(V_i, T_i, Y_i, ID_i, P_i)$, $\mathcal{C}$ checks whether the ODDH (One-sided Decision Diffie-Hellman) oracle returns 1 when queried with the tuple (aP, V_i, Y_i). If the ODDH oracle returns 1, $\mathcal{C}$ outputs Y_i and stop. Then $\mathcal{C}$ goes through the list L_1 with entries $\langle V_i, T_i, *, ID_i, P_i, l_i\rangle$, for different values of l_i, such that the ODDH oracle returns 1 when queried on the tuple (aP, V_i, Y_i). Note that in this case $ID_i = ID_t$. If such a tuple exists, it returns l_i and replaces the symbol $*$ with Y_i. Otherwise, $\mathcal{C}$ chooses $l \in_R \{0,1\}^n$ and updates the list L_1, which is initially empty, with a tuple containing the input and return values. $\mathcal{C}$ then returns l to $\mathcal{A}_{\mathcal{II}}$.
H_2 **queries**: When $\mathcal{A}_{\mathcal{II}}$ submits a H_2 query with ID_i, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ exists in the list L_2. If it exists, $\mathcal{C}$ returns $H = h_i$ to $\mathcal{A}_{\mathcal{II}}$. Otherwise, $\mathcal{C}$ chooses $h_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_i\rangle$ to L_2 and returns $H = h_i$ to $\mathcal{A}_{\mathcal{II}}$.
H_3 **queries**: When $\mathcal{A}_{\mathcal{II}}$ submits a H_3 query with ID_i, $\mathcal{C}$ checks whether a tuple of the form $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_i\rangle$ exists in the list L_3. If it exists, $\mathcal{C}$ returns $H' = h'_i$ to $\mathcal{A}_{\mathcal{II}}$. Otherwise, $\mathcal{C}$ chooses $h'_i \in_R Z_q^*$, adds the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_i\rangle$ to L_3 and returns $H' = h'_i$ to $\mathcal{A}_{\mathcal{II}}$.
Extract-Partial-Private-Key queries: When $\mathcal{A}_{\mathcal{II}}$ asks a Extract-Partial-Private-Key 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.
Extract-Secret-Value queries: If $\mathcal{A}_{\mathcal{II}}$ asks a Extract-Secret-Value query for ID_i, $\mathcal{C}$ answers as follows:

- If $ID_i = ID_t$, $\mathcal{C}$ aborts.

- If $ID_i \neq ID_t$, $\mathcal{C}$ looks for the tuple $\langle ID_i, d_i, x_i, R_i, P_i \rangle$ in the list L_k. If such tuple exists in 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 $\mathcal{A_{II}}$ 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 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 $\mathcal{A_{II}}$ 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 $\mathcal{A_{II}}$. The current public key (i.e. replaced public key) is used by $\mathcal{C}$ for computations or responses to any queries made by $\mathcal{A_{II}}$.
Symmetric Key Generation queries: $\mathcal{A_{II}}$ 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) to $\mathcal{C}$. For each query (ID_A, ID_B), $\mathcal{C}$ proceeds as follows:

- If $ID_A \neq ID_t$, $\mathcal{C}$ computes the full private key sk_A corresponding to ID_A by executing the Extract-Partial-Private-Key query and Extract-Secret-Value query algorithm. Then, $\mathcal{C}$ gets the symmetric key K and an internal state information ω by running the actual SymmetricKeyGen algorithm. $\mathcal{C}$ stores ω and overwrite any previous value. $\mathcal{C}$ sends the symmetric key K to $\mathcal{A_{II}}$.

- If $ID_A = ID_t$ (and hence $ID_B \neq ID_t$), $\mathcal{C}$ chooses $r_1, r_2, h_t, h'_t \in_R Z_q^*$ and computes $U = r_1P - {h_t}^{-1} \cdot h'_t \cdot P_A$, where P_A is obtained from the list L_k, $V = r_2P$, $T = r_2 \cdot H_0(ID_B, R_B, P_B)P_{pub} + r_2 \cdot R_B \bmod q$ and $K = H_1(U, T, r_2 \cdot P_B, ID_B, P_B)$. Note that ω is $\omega = (r_1, r_2, h_t, h'_t, U, V, T, ID_A, d_A, pk_A, ID_B, pk_B)$.

- $\mathcal{C}$ goes through the list L_1 looking for an entry (V, T, Y, ID_B, P_B, k) for some k such that ODDH(P_B, V, Y)=1, where P_B is obtained by calling the Request-Public-Key query oracle on ID_B. If such an entry exists, it computes $K \leftarrow l$. Otherwise it uses a random l and updates the list L_1 with $(V, T, *, ID_B, P_B, l)$. $\mathcal{C}$ stores ω and sends the symmetric key K to $\mathcal{A_{II}}$.

Key Encapsulation queries: $\mathcal{A_{II}}$ produces an arbitrary tag τ, 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 ω value is stored previously.

- If ω does not exist, $\mathcal{C}$ returns invalid.

- If a corresponding ω exists and $ID_A \neq ID_t$, then $\mathcal{C}$ computes φ with ω and τ by using the actual Encapsulation algorithm, and deletes ω. Here, $\mathcal{C}$ gets the full private key of the sender $sk_A = (d_A, x_A)$ from the list L_k.

- If a corresponding ω exists and $ID_A = ID_t$, then $\mathcal{C}$ computes φ by performing the following steps. Note that ω is $(r_1, r_2, h_t, h'_t, U, V, T, ID_A, d_A, pk_A, ID_B, pk_B)$ and $\mathcal{C}$ does not know the secret value x_A corresponding to ID_A. So $\mathcal{C}$ should perform the encapsulation in a different way:

  1. Set $H = h_t$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t \rangle$ to the list L_2.
  2. Set $H' = h'_t$ and add the tuple $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_t \rangle$ to the list L_3.

3. Compute $W = d_A + r_1H$, where d_A is obtained from the list L_k.
4. Output $\varphi = (U, V, W)$ as the encapsulation.

We show that $\mathcal{A}_{\mathcal{II}}$ can pass the verification of $\varphi = (U, V, W)$ 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:
$R_A + H_0(ID_A, R_A, P_A) \cdot P_{pub} + H \cdot U + H' \cdot P_A$
$= a_tP + l_t \cdot sP + h_t \cdot (r_1P - {h_t}^{-1} \cdot h'_t \cdot P_A) + h'_t \cdot P_A$
$= (a_t + l_t \cdot s) \cdot P + h_t \cdot r_1P - h'_t \cdot P_A + h'_t \cdot P_A$
$= d_A \cdot P + h_t \cdot r_1P$
$= (d_A + r_1H) \cdot P$
$= W \cdot P$

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

- If $ID_B \neq ID_t$, then $\mathcal{C}$ computes the decapsulation of φ 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 φ as follows:
  1. Searches lists L_2 and L_3 for entries of the type $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h_t\rangle$ and $\langle U, \tau, T, ID_A, P_A, ID_B, P_B, h'_t\rangle$, respectively.
  2. If entries $H = h_t$ and $H' = h'_t$ 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.
  3. If the above equality holds, then retrieves the corresponding value of T from lists L_2 and L_3. Both the T values should be equal.
  4. $\mathcal{C}$ goes through L_1 and looks for a tuple $\langle V, T, Y, ID_B, P_B, l\rangle$ such that ODDH(P_B, $V, Y) = 1$. If such an entry exists, then $\mathcal{C}$ found the correct value of Y and outputs $K \leftarrow l$.
  5. If $\mathcal{C}$ reaches this point of execution, it places the entry $\langle U, T, *, ID_B, P_B, l\rangle$ for a random l on list L_1 and returns the corresponding $K \leftarrow l$ value as the decapsulation of φ. The symbol $*$ denotes an unknown value of Y.

Challenge: At the end of Phase I, $\mathcal{A}_{\mathcal{II}}$ 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 $ID_{B*} \neq ID_t$. Otherwise, $\mathcal{C}$ performs the following to compute the challenge encapsulation φ^*.

1. Choose $r \in_R Z_q^*$ and compute $U^* = rP$.
2. Set $V^* = bP$ and compute $T^* = d_{B*} \cdot V$, where $\mathcal{C}$ knows the partial private key for ID_{B*}, d_{B*}. Here, $\mathcal{C}$ uses the bP given in the instance of the OGDH problem.
3. Choose $K_0 \in_R \mathcal{K}$, where $\mathcal{K}$ is the key space of the pCLSC-TKEM.
4. Choose $Y^* \in_R G_q$ and compute $K_1 = H_1(V^*, T^*, Y^*, ID_{B*}, P_{B*})$.
5. Set $\omega^* = \langle -, V^*, Y^*, T^*, ID_{A*}, P_{A*}, R_{A*}, x_{A*}, d_{A*}, ID_{B*}, P_{B*}, R_{B*}\rangle$.
6. $\mathcal{C}$ chooses a bit $\delta \in_R \{0, 1\}$ and sends K_δ to $\mathcal{A}_{\mathcal{II}}$.

7. $\mathcal{A}_{\mathcal{II}}$ generates an arbitrary tag τ^* and sends it to $\mathcal{C}$.

8. Choose $h_i, h'_i \in_R \mathsf{Z}_q^*$, store the tuple $\langle U^*, \tau^*, T^*, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h_i\rangle$ to the list L_2 and $\langle U, \tau^*, T, ID_{A*}, P_{A*}, ID_{B*}, P_{B*}, h'_i\rangle$ to the list L_3.

9. Since $\mathcal{C}$ knows the private key of the sender ID_{A*}, $\mathcal{C}$ computes $W = d_{A*} + r \cdot h_i + x_{A*} \cdot h'_i$.

10. $\mathcal{C}$ sends $\varphi^* = \langle U^*, V^*, W^*\rangle$ 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. Other than this, it cannot query decapsulation on φ^*.
Guess: Since $\mathcal{A}_{\mathcal{II}}$ is able to break the IND-pCLSC-TKEM-CCA2-II security of pCLSC-TKEM (which is assumed at the beginning of the proof), $\mathcal{A}_{\mathcal{II}}$ should have asked a H_1 query with $(V^*, T^*, Y^*, ID_{B*}, P_{B*})$ as inputs. Since ID_{B*} is a target identity ID_t, $P_{B*} = aP$. Here, aP was given as the instance of the OGDH problem and $\mathcal{C}$ does not know a. Thus, computing $Y^* = x_{B*}V^* = abP$ is to find abP when $\langle P, aP(= P_{B*}), bP(= V)\rangle \in G_q$ 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 Y^* stored in the list L_1 is the solution for the OGDH problem instance. $\mathcal{C}$ chooses one Y^* value uniformly at random from the q_{H_1} values from the list L_1 and outputs it as the solution for the OGDH 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-pCLSC-TKEM-CCA2-II game.

- E_1 is an event in which $\mathcal{A}_{\mathcal{II}}$ 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 in which $\mathcal{A}_{\mathcal{II}}$ 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 in which $\mathcal{A}_{\mathcal{II}}$ 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-pCLSC-TKEM-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 Y^* from L_1 and Y^* is the solution of OGDH problem is $\frac{1}{q_{H_1}}$. So, the probability that $\mathcal{C}$ finds the OGDH 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 ε is non-negligible.

5.2 Existential Unforgeability

Theorem 2. *In the random oracle model, the pCLSC-TKEM is EUF-pCLSC-TKEM-CMA secure under the assumption that the Elliptic Curve Discrete Logarithm Problem (ECDLP) problem is intractable.*
The Theorem 2 is proved based on Lemma 3 and 4.

Lemma 3. Suppose that the hash functions $H_i (i = 0, 1, 2, 3)$ are random oracles. If there exists a forger $\mathcal{F}_{\mathcal{I}}$ against the EUF-pCLSC-TKEM-CMA-I security of the pCLSC-TKEM with advantage a non-negligible ε, 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} extract-partial-private-key queries and q_{sv} extract-secret-value queries, then there exists an algorithm $\mathcal{C}$ that solves the ECDLP (Elliptic Curve Discrete Logarithm Problem) with the following advantage ε'.

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

Proof. A challenger $\mathcal{C}$ is challenged with an instance of the ECDLP. To solve the ECDLP, given $\langle P, bP \rangle \in G_q$, $\mathcal{C}$ must find b. Let $\mathcal{F_I}$ be a forger who is able to break the EUF-pCLSC-TKEM-CMA-I security of the pCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{F_I}$ to compute the solution b of the ECDLP instance by playing the following interactive game with $\mathcal{F_I}$. To solve the ECDLP, $\mathcal{C}$ sets the master private/public key pair as $(x, P_{pub} = xP)$, 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. Then $\mathcal{C}$ 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 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, Extract-Partial-Private-Key queries, Extract-Secret-Value queries, Public-Key-Replacement queries, Symmetric Key Generation queries, Key Encapsulation queries and Key Decapsulation queries in IND-pCLSC-TKEM-CCA2-I game.

Forgery: Eventually, $\mathcal{F_I}$ returns a valid encapsulation $\langle \tau, \varphi = (U, V, W), ID_A, ID_B \rangle$ on a arbitrary tag τ, 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 $\langle \tau, \varphi^* = (U, V, W^*), ID_A, ID_B \rangle$ with different h_i^* such that $h_i^* \neq h_i$ on the same τ as done in the forking lemma [19]. Thus, we can get $W \cdot P = R_A - e_t \cdot P_{pub} + h_i \cdot U + h_i' \cdot P_A$ and $W^* \cdot P = R_A - e_t \cdot P_{pub} + h_i^* \cdot U + h_i' \cdot P_A$. Let $U = bP$. Then if we subtract these two equations, we obtain following value.

$W^* \cdot P - W \cdot P = h_i^* \cdot U - h_i \cdot U$
$\Rightarrow (W^* - W)P = (h_i^* - h_i) \cdot U$
$\Rightarrow (W^* - W)P = (h_i^* - h_i) \cdot bP$
$\Rightarrow (W^* - W) = b \cdot (h_i^* - h_i)$
$\Rightarrow (W^* - W) \cdot (h_i^* - h_i)^{-1} = b$

Therefore, $\mathcal{F_I}$ solves the ECDLP as $b = \frac{W^* - W}{h_i^* - h_i}$ using the algorithm $\mathcal{C}$ for a given random instance $\langle P, bP \rangle \in G_q$.

Analysis: In order to assess the probability of success of the challenger $\mathcal{C}$. We assume that $\mathcal{F_I}$ 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 $\mathcal{F_I}$ never repeats H_i $(0 \leq i \leq 3)$ a 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 $\mathcal{F_I}$ queries the partial private key of the target identity ID_t is $\frac{q_{ppri}}{q_{H_0}}$.
- The probability that $\mathcal{F_I}$ asks to query the set secret value of the ID_t is $\frac{q_{sv}}{q_{H_0}}$.

So, the success probability that $\mathcal{C}$ can win the EUF-pCLSC-TKEM-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 ECDLP is non-negligible, because ε is non-negligible.

Lemma 4. Suppose that the hash functions $H_i (i = 0, 1, 2, 3)$ are random oracles. If there exists a forger $\mathcal{F_{II}}$ against the EUF-pCLSC-TKEM-CMA-II security of the pCLSC-TKEM with advantage a non-negligible ε, 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} extract-secret-value queries and q_{pkR} public key replacement queries, then there exist an algorithm $\mathcal{C}$ that solves the ECDLP with the following advantage ε'.
$\varepsilon' \geq q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot \varepsilon$

Proof. A challenger $\mathcal{C}$ is challenged with an instance of the ECDLP. Given $\langle P, aP \rangle \in G_q$, $\mathcal{C}$ must find a. Let $\mathcal{F_{II}}$ be a forger who is able to break the EUF-pCLSC-TKEM-CMA-II security of the pCLSC-TKEM. $\mathcal{C}$ can utilize $\mathcal{F_{II}}$ to compute the solution a of the ECDLP instance by playing the following interactive game with $\mathcal{F_{II}}$. To solve the ECDLP, $\mathcal{C}$ 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. Then $\mathcal{C}$ 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 $\mathcal{F_{II}}$. 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 of issued private keys and public keys. $\mathcal{C}$ can simulate the challenger's execution of each phase of the formal Game.
Training Phase: $\mathcal{F_{II}}$ 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, Extract-Partial-Private-Key queries, Extract-Secret-Value queries, Public-Key-Replacement queries, Symmetric Key Generation queries, Key Encapsulation queries and Key Decapsulation queries in IND-pCLSC-TKEM-CCA2-II game.
Forgery: Eventually, $\mathcal{F_{II}}$ returns a valid encapsulation $\langle \tau, \varphi = (U, V, W), ID_A, ID_B \rangle$ on an arbitrary tag τ to $\mathcal{C}$, where the target identity ID_t is the sender identity ID_A and ID_B is the receiver identity. 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 $\langle \tau, \varphi^* = (U, V, W^*), ID_t, ID_B \rangle$ with different $h_i^{'*}$ such that $h_i^{'*} \neq h_i'$ on the same τ as done in the forking lemma [19]. Thus, we can get $W \cdot P = R_t - l_t \cdot P_{pub} + h_t \cdot U + h_t' \cdot P_t$ and $W^* \cdot P = R_t - l_t \cdot P_{pub} + h_t \cdot U + h_t^{'*} \cdot P_t$. Note that $P_t = aP$. Then if we subtract these two equations, we obtain following value.
$W^* \cdot P - W \cdot P = h_i^{'*} \cdot P_t - h_i' \cdot P_t$
$\Rightarrow (W^* - W)P = (h_i^{'*} - h_i') \cdot aP$

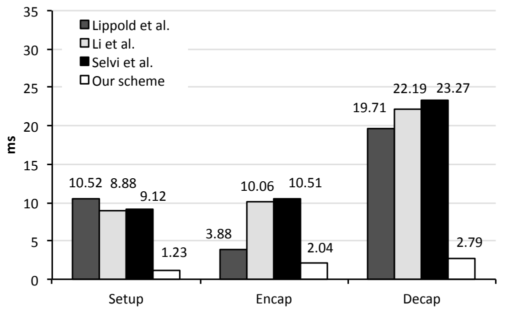
$\Rightarrow (W^* - W)P = (h_i^* - h_i')] \cdot aP$
$\Rightarrow (W^* - W) = (h_i^* - h_i') \cdot a$
$\Rightarrow (W^* - W) \cdot (h_i^* - h_i')^{-1} = a$
Therefore, $\mathcal{F}_{\mathcal{II}}$ solves the ECDLP as $a = \frac{W^*-W}{h_i^*-h_i'}$ using the algorithm $\mathcal{C}$ for a given random instance $\langle P, aP \rangle \in G_q$.
Analysis: In order to assess the probability of success of the challenger $\mathcal{C}$. We assume that $\mathcal{F}_{\mathcal{II}}$ 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 $\mathcal{F}_{\mathcal{II}}$ 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 $\mathcal{F}_{\mathcal{II}}$ queries the secret value of the target identity ID_t is $\frac{q_{sv}}{q_{H_0}}$.
- The probability that $\mathcal{F}_{\mathcal{II}}$ asks to replace the public key of the ID_t is $\frac{q_{pkR}}{q_{H_0}}$.

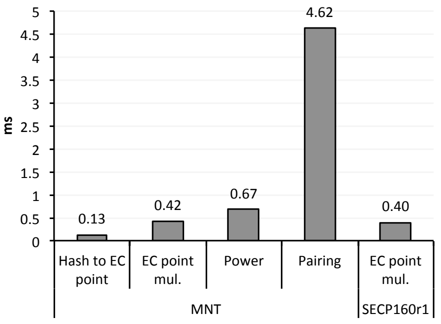
So, the success probability that $\mathcal{C}$ can win the EUF-pCLSC-TKEM-CMA-II game is $\varepsilon' \geq q_E \cdot (1 - \frac{q_{H_0} \cdot q_C}{q}) \cdot (1 - \frac{q_{H_2}^2}{q}) \cdot (1 - \frac{q_{H_3}^2}{q}) \cdot (1 + \frac{1}{q}) \cdot (\frac{1}{q_C}) \cdot (1 - \frac{q_{sv}}{q_{H_0}}) \cdot (1 - \frac{q_{pkR}}{q_{H_0}}) \cdot \varepsilon$ Therefore, the probability that $\mathcal{C}$ computes the solution of ECDLP is non-negligible, because ε is non-negligible.

6 Performance Evaluation

In this section, we evaluate the performance of our pCLSC-TKEM. We focus on the computational overhead of pCLSC-TKEM and compare it with the overhead of other three schemes: Lippold et al.'s pairing-based CL-KEM [13], Li et al.'s pairing-based CLSC-TKEM [12], and Selvi et al.'s CLSC-TKEM [20]. We thus implemented all these three schemes in addition to implementing pCLSC-TKEM. As in most real-world applications of the client-server model, the server possesses powerful computational resources, and the client has relatively limited computing power, we implemented the four schemes on three different platforms: a Raspberry Pi 2 model B, an Android smartphone (acting as clients) and a PC (acting as a server). Then, we compared pCLSC-TKEM with the previous pairing based CL-KEM [13] and CLSC-TKEMs [12, 20] with respect to each of the following steps: Setup; the Encap step including symmetric key generation and key encapsulation; and the Decap step including key decapsulation. The computational overhead of Selvi et al.'s CLSC-TKEM is similar to that of Li et al.'s CLSC-TKEM, because the scheme by Selvi et al. is a modified version of Li et al.'s scheme with improvements to resolve the security weaknesses of the latter. The difference between the two is in the input parameters of the hash functions.

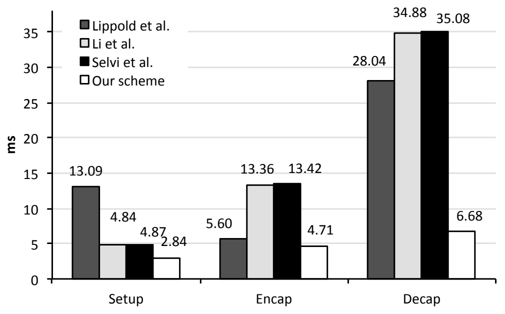


(a) Computation time comparison

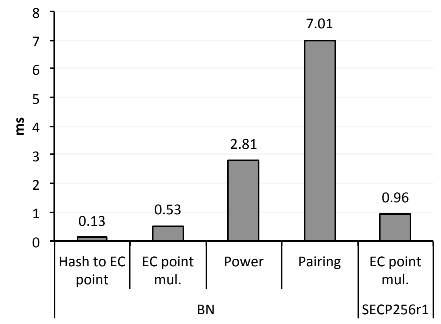


(b) Computation times of expensive operations

Figure 1: PC (1024-bit RSA security level)



(a) Computation time comparison



(b) Computation times of expensive operations

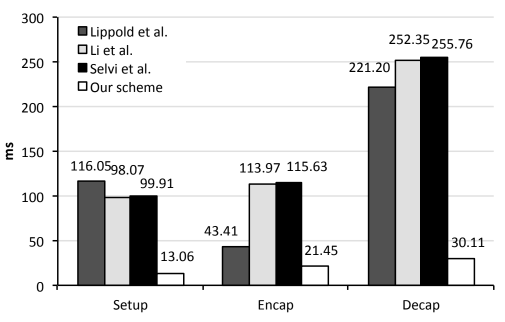
Figure 2: PC (3072-bit RSA security level)

For the PC and Raspberry Pi 2 platform, predetermined parameters and elliptic curves are used to achieve security levels at 1024/3072-bit RSA. For the Android smartphone platform, the parameters for 1024/2048-bit RSA security levels were chosen. Such security levels were chosen since each library for the three platforms supports 1024/3072-bit RSA security level or 1024/2048-bit RSA security level for both PBC and ECC. Then, we evaluated the computational overhead of each scheme according to these differing security levels.

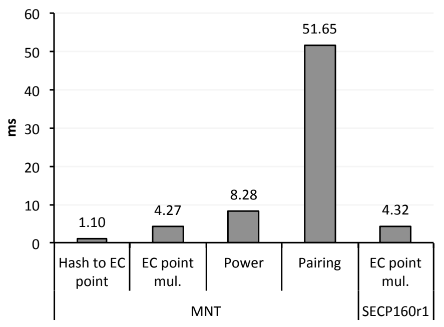
6.1 Experimental Results in the PC platform and Raspberry Pi 2

We implemented three pairing-based schemes and our scheme using MIRACL library (ver. 7.1.0) [5]. MIRACL is an up-to-date cryptographic library supporting pairing-based cryptography (PBC) as well as elliptic curve cryptography (ECC). For the setting of 1024-bit RSA security level, we utilized a MNT curve with embedding degree k=6 for the pairing parameter and SECP160R1 for the elliptic curve parameter. SECP160R1 defines EC parameters on a 160-bit prime field.

For the experiments on the PC platform, we utilized a Linux machine running 64-bit GNU Linux kernel version 3.5.0-23 with 1.8 GHz Intel Core i5 and 4GB memory. Fig. 1(a) compares the computation times of the four schemes at each step. Our scheme is 5.6, 6.8 and 7.1 times faster than the Lippold et al.'s scheme, the Li et al.'s scheme and the Selvi et al.'s

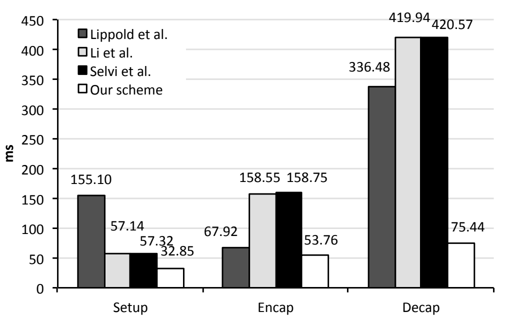


(a) Computation time comparison

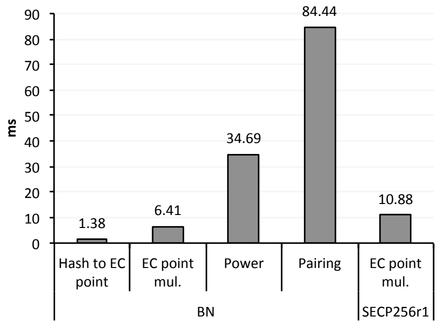


(b) Computation times of expensive operations

Figure 3: Raspberry Pi 2 (1024-bit RSA security level)



(a) Computation time comparison



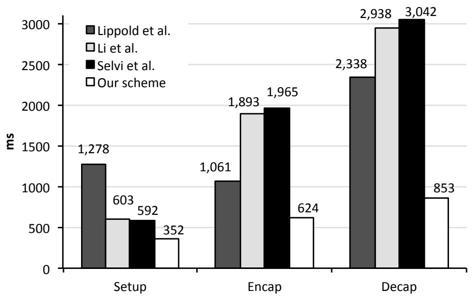
(b) Computation times of expensive operations

Figure 4: Raspberry Pi 2 (3072-bit RSA security level)

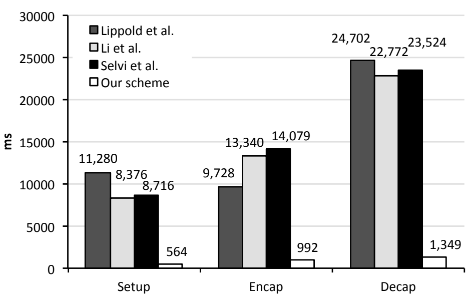
scheme in the total computation time.

In addition, the same experiments were performed on the Raspberry Pi 2 running Raspbian Linux (version 4.1) with 900 MHz quad-core ARM Cortex-A7 (32-bit) and 1GB RAM. Raspberry Pi [21] is a popular platform for IoT applications because it offers a complete Linux in a tiny platform for a very low cost. Fig. 3(a) shows the computation times of the four schemes at each step. Our scheme is 5.9, 7.2 and 7.3 times faster in the total computation time than the Lippold et al.'s scheme, the Li et al.'s scheme and the Selvi et al.'s scheme in the total computation time. Notice that our scheme takes only 30.11ms for the decapsulation, whereas three other schemes take more than 200ms (see Fig. 3(a)). Considering that 100ms is the limit for having the user feel that the system is reacting instantaneously [15], our pairing-free scheme has a big advantage over other pairing-based schemes for the real IoT applications.

The computation times of Li et al.'s scheme and the Selvi et al.'s scheme are similar because Selvi et al.'s scheme simply inserts more parameters into the hash functions. Fig. 1(b) and Fig. 3(b) show the computation times of each expensive operations on the PC and the Raspberry Pi 2, respectively. They provide an explanation of the improved performance of our pairing-free scheme. In the pairing-based schemes, the computational overhead is mainly due to the pairing operation. This is in contrast to our pCLSC-TKEM scheme which mainly relies on EC point scalar multiplication. For example, in the PC platform, EC point scalar multiplication under the SECP160R1 is 11.6 times faster than the pairing operation



(a) 1024-bit RSA security level



(b) 2048-bit RSA security level

Figure 5: Computation time comparison on an Android smart phone

under the MNT curve.

For 3072-bit security, we adopt a BN curve with embedding degree k=12 for the pairing parameter and SECP256R1 for the elliptic curve parameter. SECP256R1 defines EC parameters on a 256-bit prime field. In the PC platform, our scheme is 3.3, 3.7 and 3.7 times faster (see Fig. 2(a)) than Lippold et al.'s, Li et al.'s and Selvi et al.'s schemes with respect to total computation time, respectively. In the Raspberry Pi 2 platform, our scheme is 3.5, 3.9 and 3.9 times faster (see Fig. 4(a)) than Lippold et al.'s, Li et al.'s and Selvi et al.'s schemes with respect to total computation time, respectively.

As shown in Fig. 2(b) and Fig. 4(b), the EC point multiplication under SECP256R1 is much faster than exponentiation (power) and pairing operations under the BN curve. One interesting point is that EC point multiplication under SECP256R1 is 1.8 times slower than EC point scalar multiplication under the BN curve in the PC platform, which means the computation time of our scheme can be reduced if the BN curve is used instead of SECP256R1.

6.2 Experimental Results in the Android SmartPhone Platform

The four schemes were also implemented on a Samsung Galaxy S4 Zoom that runs on Android 4.2 with ARM cortex A9 processor (32-bit, 1.5GHz) and 512 MB memory. We utilized the JPBC library [6] 2.0.0 for PBC and the JECC library [23] 1.1 for ECC. For the setting of 1024-bit RSA security level, we utilized type a for the pairing parameter and SECP160R1 for the elliptic curve parameter. Type a has the base field size of 512 bits and the embedding degree of the curve is 2. For the setting of 2048-bit RSA security level, we utilized type a1 for the pairing parameter and SECP224R1 for the elliptic curve parameter. Type a1 has the base field size of 1024 bits and the embedding degree of the curve is 2.

Under the 1024-bit RSA security level, our scheme only takes 0.62 and 0.85 seconds for Encap and Decap, respectively (see Fig. 5(a)), and is more than twice faster than pairing-based schemes. Considering that even the most state-of-the-art consumer mobile devices are equipped with ARM cortex series, these results provide strong evidence that our pairing-free scheme is practical in real-world applications.

In 2048-bit RSA security level, the advantage of our pairing-free scheme over the pairing-based schemes becomes more apparent. Our pCLSC-TKEM requires significantly less CPU time (see Fig. 5(b)), which also implies that our scheme consumes less battery power in smartphones. In addition, considering that NIST recommends that cryptographic functions adopt 2048-bit RSA security from 2011 to 2030 [16], our pairing-free CLSC-TKEM will be advantageous over pairing-based schemes in current and in future applications.

7 Conclusions

Confidentiality and authentication are prerequisite to protect privacy-sensitive data generated by IoT devices from unauthorized accesses. To address this requirement, in this paper, we developed a new construction for the certificateless signcryption tag key encapsulation (CLSC-TKEM) without utilizing bilinear pairing operations. We refer to this novel scheme as pairing-free CLSC-TKEM (pCLSC-TKEM). Then, we propose a simple construction for pairing-free CL-HSC by combining the pCLSC-TKEM with the DEM. The proposed pCLSC-TKEM efficiently executes both key encapsulation and digital signature, while providing security against both an adaptively chosen ciphertext attack and an existential forgery of Type I and II adversaries. In order to measure the performance and applicability of the proposed scheme, we implemented the pCLSC-TKEM and previously defined pairing-based schemes on various platforms such as a PC, Android smartphones and Raspberry Pi - a platform widely used for IoT applications. Our performance comparison with the previous approaches shows that our pCLSC-TKEM is an ideal building block for securing the privacy of data being transmitted in emerging distributed systems and applications enabled by IoT technology.

References

[1] Al-Riyami S, Paterson K (2003) Certificateless public key cryptography. In: Advances in Cryptology - ASIACRYPT 2003, Lecture Notes in Computer Science, vol 2894, Springer, pp 452–473

[2] An JH, Dodis Y, Rabin T (2002) On the security of joint signature and encryption. In: EUROCRYPT 2002, Springer, Lecture Notes in Computer Science, vol 2332, pp 83–107

[3] Barbosa M, Farshim P (2008) Certificateless signcryption. In: Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS '08, pp 369–372

[4] Bjørstad TE, Dent AW (2006) Building better signcryption schemes with tag-kems. In: Proceedings of 9th International Conference of Public-Key Cryptography, PKC 2006, Springer-Verlag, Lecture Notes in Computer Science, pp 491–507

[5] Certivox (2014) Miracl cryptographic sdk. URL http://www.certivox.com/miracl/

[6] De Caro A, Iovino V (2011) JPBC: Java pairing based cryptography. In: Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011, IEEE, Kerkyra, Corfu, Greece, June 28 - July 1, pp 850–855, URL http://gas.dia.unisa.it/projects/jpbc/

[7] Dent AW (2005) Hybrid signcryption schemes with insider security. In: Proceedings of the 10th Australasian Conference, ACISP 2005, Springer-Verlag, Lecture Notes in Computer Science, pp 253–266

[8] Dent AW (2005) Hybrid signcryption schemes with outsider security. In: Proceedings of the 8th International Security Consernce, ISC 2005, Springer-Verlag, Lecture Notes in Computer Science, pp 203–217

[9] Dodis Y, Freedman MJ, Jarecki S, Walfish S (2004) Versatile padding schemes for joint signature and encryption. In: Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS '04, pp 344–353

[10] Koblitz N, Menezes A (2009) Intractable problem in cryptography. In: Proceedings of the 9th International Conference on Finite Field and Their Applications, Contemporary Mathematics, pp 279–300

[11] Kolias C, Stavrou A, Voas J, Bojanova I and Kuhn R (2016) Learning Internet-of-Things Security Hands-On, in IEEE Security & Privacy, vol. 14, no. 1, pp. 37-46, Jan.-Feb

[12] Li F, Shirase M, Takagi T (2009) Certificateless hybrid signcryption. In: Information Security Practice and Experience, Lecture Notes in Computer Science, vol 5451, Springer Berlin Heidelberg, pp 112–123

[13] Lippold G, Boyd C, Nieto JMG (2010) Efficient certificateless kem in the standard model. In: Proceedings of the 12th International Conference on Information Security and Cryptology, Springer-Verlag, Berlin, Heidelberg, ICISC'09, pp 34–46, URL http://dl.acm.org/citation.cfm?id=1883749.1883753

[14] Miller VS (1985) Use of elliptic curves in cryptography. In: Proceedings of Crypto 85, Springer-Verlag, pp 417–426

[15] Nielsen Norman Group URL http://www.nngroup.com/articles/response-times-3-important-limits/

[16] NIST (2007) URL http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

[17] The Open Web Application Security Project URL https://www.owasp.org/index.php/OWASP_Internet_of_Things_Project

[18] Paterson KG, Schuldt JCN, Stam M, Thomson S (2011) On the joint security of encryption and signature, revisited. In: Proceedings of the 17th ASIACRYPT'11, ASIACRYPT'11, pp 161–178

[19] Pointcheval D, Stern J (2000) Security arguments for digital signatures and blind signatures. JOURNAL OF CRYPTOLOGY 13:361–396

[20] Selvi S, Vivek S, Rangan C (2010) Certificateless kem and hybrid signcryption schemes revisited. In: Information Security, Practice and Experience, Lecture Notes in Computer Science, vol 6047, Springer, pp 294–307

[21] Raspberry Pi URL https://www.raspberrypi.org

[22] Shamir A (1985) Identity-based cryptosystems and signature schemes. In: Proceedings of CRYPTO 84 on Advances in cryptology, pp 47–53

[23] Sorensen TB, Kragh T, Erlandsen MK (2013) URL http://jecc.sourceforge.net

[24] Szczechowiak P, Oliveira LB, Scott M, Collier M, Dahab R (2008) Nanoecc: Testing the limits of elliptic curve cryptography in sensor networks. In: Proceedings of the 5th European Conference on Wireless Sensor Networks, EWSN'08, pp 305–320

[25] Zheng Y (1997) Digital signcryption or how to achieve cost(signature & encryption) cost(signature) + cost(encryption). In: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology, Springer-Verlag, CRYPTO '97, pp 165–179