ITC 4/51

Information Technology and Control
Vol. 51 / No. 4 / 2022
pp. 638-660
DOI 10.5755/j01.itc.51.4.30691

Revocable Certificateless Public Key Encryption with Equality Test

Received 2022/02/10 | Accepted after revision 2022/08/22

http://dx.doi.org/10.5755/j01.itc.51.4.30691

HOW TO CITE: Tsai, T.-T., Lin, H.-Y., Tsai, H.-C. (2022). Revocable Certificateless Public Key Encryption with Equality Test. *Information Technology and Control*, 51(4), 638-660. http://dx.doi.org/10.5755/j01.itc.51.4.30691

Revocable Certificateless Public Key Encryption with Equality Test

Tung-Tso Tsai, Han-Yu Lin, Han-Ching Tsai

Department of Computer Science and Engineering, National Taiwan Ocean University, Keelung 202, Taiwan

Corresponding author: tttsai@mail.ntou.edu.tw

Traditional public key cryptography requires certificates as a link between each user's identity and her/his public key. Typically, public key infrastructures (PKI) are used to manage and maintain certificates. However, it takes a lot of resources to build PKI which includes many roles and complex policies. The concept of certificateless public key encryption (CL-PKC) was introduced to eliminate the need for certificates. Based on this concept, a mechanism called certificateless public key encryption with equality test (CL-PKEET) was proposed to ensure the confidentiality of private data and provide an equality test of different ciphertexts. The mechanism is suitable for cloud applications where users cannot only protect personal private data but also enjoy cloud services which test the equality of different ciphertexts. More specifically, any two ciphertexts can be tested to determine whether they are encrypted from the same plaintext. Indeed, any practical system needs to provide a solution to revoke compromised users. However, these existing CL-PKEET schemes do not address the revocation problem, and the related research is scant. Therefore, the aim of this article is to propose the *first* revocable CL-PKEET scheme called RCL-PKEET which can effectively remove illegal users from the system while maintaining the effectiveness of existing CL-PKEET schemes in encryption, decryption, and equality testing processes. Additionally, we formally demonstrate the security of the proposed scheme under the bilinear Diffie-Hellman assumption.

KEYWORDS: Revocable, certificateless, equality test, public key encryption, bilinear pairing.

1. Introduction

The 1970s saw new directions in cryptography called the public key cryptography (PKC) presented by Diffie and Hellman [10]. However, there exists a certificate management problem in PKC systems. To overcome the drawback of using certificates, Shamir [31] introduced a new notion called identity-based public key cryptography (ID-PKC) which eliminates the requirement of certificates since each user's public key is generated by her/his identities. Since ID-PKC systems have the key escrow problem, Al-Riyami and Paterson [1] presented the concept of certificateless public key cryptography (CL-PKC). A CL-PKC system has a KGC that is only responsible for producing each user's partial secret key. Each user takes the partial secret key and combines it with a secret value chosen by herself/himself to produce a full secret key. Obviously, the KGC cannot obtain the full secret key of any user due to the lack of secret value for each user.

Indeed, PKC or CL-PKC has also been applied to cloud computing since there exist the potential risks of privacy disclosure that the private data could be leaked to cloud servers. In order to protect private data and search encrypted data on the cloud, Yang *et al.* [42] introduced a new concept of the public key encryption with equality test (PKEET), which supports comparing whether two encrypted data (ciphertexts) are encrypted from the same message (plaintext). Until 2018, Qu *et al.* [28] proposed a mechanism called certificateless public key encryption with equality test (CL-PKEET) to ensure the confidentiality of private data and provide equality test of different ciphertexts. An important issue in all types of PKC is to offer a revocation mechanism to revoke compromised users (revoked users). Until now, these existing CL-PKEET schemes do not address the revocation problem, and the related research is scant. Therefore, the aim of this article is to propose the *first* revocable CL-PKEET scheme called RCL-PKEET.

1.1. Related Work

The PKC directions cause extensive discussions of the applications of cryptography such as the public key signature [12, 31], the public key encryption [28, 30], and the key agreement in public key systems [17, 42]. A fact we all know that PKC requires certificates as a link between each user's identity and her/his public key. Typically, public key infrastructures (PKI) are used to manage and maintain certificates. However, it takes a lot of resources to build PKI which includes many roles and complex policies. To overcome the drawback of using certificates, Shamir [32] introduced a new notion called identity-based public key cryptography (ID-PKC). Based on the new notion, Boneh and Franklin [5] employed bilinear pairings to present the first practical identity (ID)-based encryption (IBE) scheme. Afterward, related schemes such as ID-based signature [7-8, 16], hierarchical ID-based encryption [20, 25], ID-based broadcast encryption [9, 23], ID-based authentication [19, 22] have been studied and published. However, ID-PKC inheres in key escrow problem since the key generation center (KGC), a major role in ID-PKC, is used to produce each user's secret key in the sense that the KGC keeps the secret keys of all the users. In 2003, Al-Riyami and Paterson [1] presented the concept of CL-PKC to overcome the key escrow problem while eliminating the certificate requirement. After that, there has been a dramatic proliferation of research concerned with CL-PKC such as certificateless public key signature [2, 40], certificateless public key encryption [41, 46], certificateless public key agreement [24, 36].

To protect private data and search encrypted data on the cloud, a number of studies [3-4, 15, 38-39] of searching the encrypted data, namely public key encryption with keyword search (PEKS), were proposed. Unfortunately, the PEKS is only suitable for a user to search his/her encrypted data in the sense that PEKS cannot apply to multiple users' scenarios. To offer the search of encrypted data for multiple users, Yang *et al.* [43] introduced a new concept of the public key encryption with equality test (PKEET) which supports to compare whether two encrypted data (ciphertexts) are encrypted from the same message (plaintext). But PKEET still has the drawback of using certificates, Ma [26] combined the benefits of PKEET and ID-PKC to present a new mechanism called the identity-based public key encryption with equality test (ID-PKEET), which eliminates the requirement of certificates. As already mentioned above, ID-PKC appears the key escrow problem, and so does ID-PKEET. Based on the concept of CL-PKC, a mechanism called certificateless public key encryption with equality test (CL-PKEET) was proposed by Qu *et al.* [29] to ensure the confidentiality of private data and provide equality test of different ciphertexts. The mechanism, eliminating the requirement of certificates, does not have the key escrow problem and is suitable for cloud applications where users cannot only protect personal private data but also enjoy cloud services which test the equality of different ciphertexts. Two currently popular applications are Internet of Vehicles (IoV) and Industrial Internet of Things (IIoT). For the applications, two related literatures, namely CL-PKEET toward IoV [45] and CL-PKEET in IIoT [13], have been proposed.

All types of PKC need to offer a revocation mechanism to revoke compromised users (revoked users) such as PKC with revocation mechanism [44], ID-PKC with revocation mechanism [18, 35], CL-PKC with revocation mechanism [11, 34]. A revocation solution in traditional PKC is the certificate revocation list [14], but it is not suitable for ID-PKC or CL-PKC, since they do not have certificates. Boneh and Franklin [5] proposed a suggestion of revocation on an ID-PKC where every valid user can get a new secret key for each time period by secret channels, and a revoked user cannot get a new secret key. However, the solution is inefficient for multiple users since the cost of establishing secret channels is increased linearly with the number of users. Indeed, CL-PKC can adopt the solution to achieve a revocable CL-PKC (RCL-PKC), but the problem of inefficiency still exists. Tsai and Tseng [37] presented an efficient revocation method that uses public channels to revoke compromised users. Ma *et al.* [27] hired the efficient revocation method to propose revocable certificateless public key encryption with an outsourced semi-trusted cloud revocation agent.

1.2. Motivation

In fact, users can also be revoked in the existing ID-PKEET [26] and CL-PKEETs [13, 29, 45]. In these constructions, the KGC transmits secret keys to users through secure channels. The KGC can realize revocation by resending new secret keys to non-revoked users. As a result, the user who has not received the new secret key is the revoked user. However, such revocation requires a secure channel, and the establishment of this channel requires encryption and decryption procedures. In order to improve the efficiency of revoking users, we must remove the way of revoking users through secure channels. Therefore, we attempt to propose a new mechanism to revoke users through open channels.

1.3. Contribution and Organization

Until now, these existing CL-PKEET schemes do not address the revocation problem, and the related research is scant. Therefore, the aim of this article is to propose the *first* revocable CL-PKEET scheme called RCL-PKEET which can effectively remove illegal users from the system, while maintaining the effectiveness of existing CL-PKEET schemes in encryption, decryption, and equality testing processes. Additionally, we formally demonstrate the security of the proposed scheme under the bilinear Diffie-Hellman assumption.

The organization of this article is as follows. In Section 2, we give some preliminaries. In Section 3, we define the framework and security notions of RCL-PKEET. A concrete RCL-PKEET scheme is presented in Section 4. Section 5 analyzes the security of the RCL-PKEET scheme. We compare the performance with other existing schemes and draw a conclusion in Sections 6 and 7, respectively.

2. Preliminaries

In this section, we briefly describe the bilinear pairings and the bilinear Diffie-Hellman assumption which are used to construct our concrete scheme and analyze the security later. Let G_1, G_2, G_T be three multiplicative cyclic groups of large prime order q and two generators $P \in G_1$ and $Q \in G_2$. There is an asymmetric bilinear pairings e: $G_1 \times G_2 \to G_T$ satisfying three conditions as follows:

- Bilinear: for any $a, b \in Z_q^*$, $e(P^a, Q^b) = e(P, Q)^{ab}$.
- Non-degenerate: $e(P, Q) \neq 1$.
- Computable: the asymmetric bilinear pairings e is computable efficiently.

The bilinear Diffie-Hellman (BDH) assumption in the symmetric bilinear groups [5] was first presented in 2001. Boyen *et al.* [6] extended the BDH assumption from the symmetric bilinear groups to the asymmetric ones.

Bilinear Diffie-Hellman problem: let $\mathcal{G} = (q, G_1, G_2, G_T, e)$ defined as above, $P \in G_1$, $Q \in G_2$ be two generators, and $a, b, c \in Z_q^*$ be random numbers. Given $(P, P^a, P^c, Q, Q^a, Q^b) \in G_1^3 \times G_2^3$, compute $e(P, Q)^{abc} \in G_T$.

Definition 1. (*Bilinear Diffie-Hellman assumption*). Given an instance of bilinear Diffie-Hellman problem, no probabilistic polynomial time (PPT) adversary $\mathcal{A}$ computes $e(P, Q)^{abc}$ with non-negligible advantage which is defined as

$\Pr[\mathcal{A}(P, P^a, P^c, Q, Q^a, Q^b) = e(P, Q)^{abc}] < \epsilon$.

Note that the BDH assumption is based on solving the discrete logarithm problem which is to compute a by giving $P \in G_1$ and P^a, where a is a random value chosen in Z_q^*.

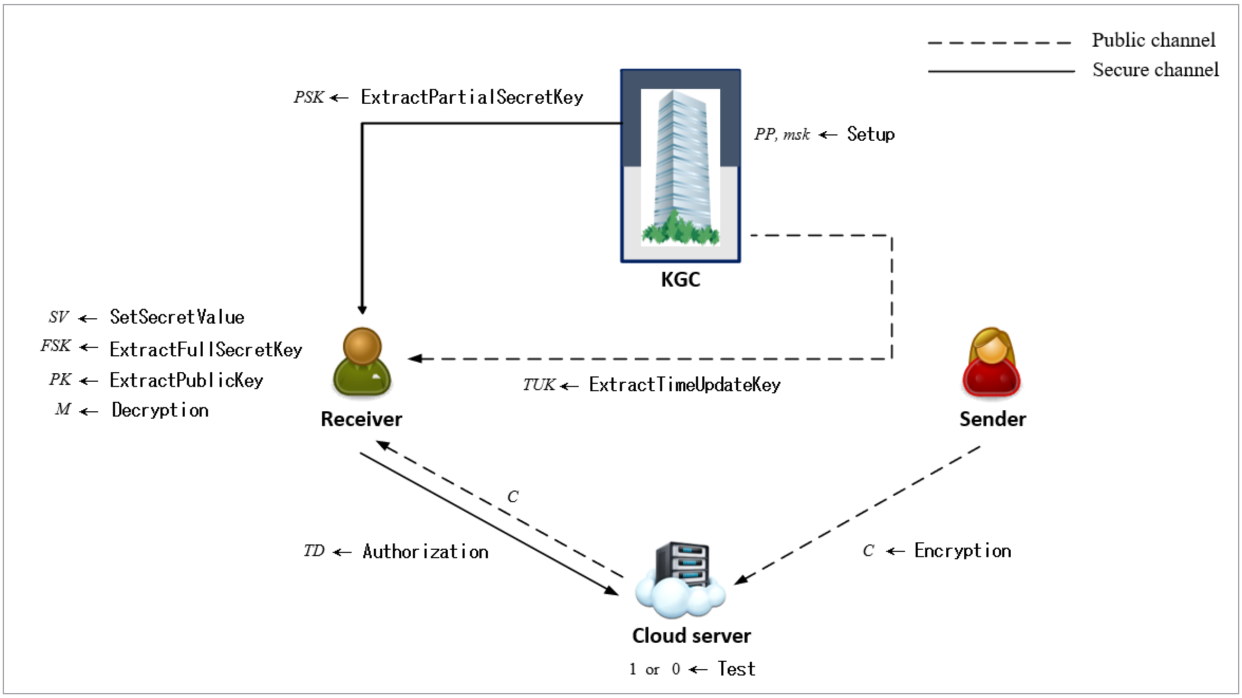
3. Framework and Security Notions of RCL-PKEET

3.1. Framework

This subsection formalizes the RCL-PKEET framework which is identical to CL-PKEET proposed by Qu *et al.* [29] except that it adds ExtractTimeUpdateKey algorithm. The proposed RCL-PKEET comprises three entities, namely the key generation center (KGC), the cloud server (CS) and users (senders and receivers), which are depicted in Figure 1. The KGC performs two tasks. One is responsible for creating a partial secret key *PSK* for each user, and the other for calculating the time update key *TUK* for each time period. Then, the *PSK* and the *TUK* are respectively issued via a secure channel and a public channel to each user. Each user selects a secret value *SV* and generates a full secret key *FSK* using *SV*, *PSK* and *TUK*, where the *FSK* is used to decrypt the associated ciphertext and produce the trapdoor *TD*. The *TD* of each user is transmitted to the CS, and then the CS can use it to compare whether the two ciphertexts are encrypted from the same plaintext. In the following, we first present the framework of RCL-PKEET which consists of ten algorithms:

- Setup(λ). Take a security parameter λ as input, and output system public parameters *PP* and a master secret key *msk*. This algorithm is run by a KGC to initially set up the system of RCL-PKEET.
- ExtractPartialSecretKey(*PP*, *ID*, *msk*). Take the public parameters *PP*, a user's identity $ID \in \{0, 1\}^*$ and the master secret key *msk* as input, and output the user's partial secret key *PSK*. This algorithm is run by the KGC once for the user and returns the *PSK* to the user via a secure channel.
- ExtractTimeUpdateKey(*PP*, *ID*, *t*, *msk*). Take the public parameters *PP*, a user's identity $ID \in \{0, 1\}^*$, a time period *t* and the master secret key *msk* as input, and output the user's time update key *TUK*. This algorithm is run by the KGC and returns the *TUK* to the user via a public channel.
- SetSecretValue(*PP*). Take the public parameters *PP* as input, and output a secret value *SV*. This algorithm is run by the user.
- ExtractFullSecretKey(*PP*, *PSK*, *TUK*, *SV*). Take the public parameters *PP*, a user's partial secret key

Figure 1
The framework of RCL-PKEET



PSK, the user's time update key *TUK*, and the user's secret value *SV* as input, and output the user's full secret key *FSK*. This algorithm is run by the user who can use the *FSK* to decrypt the associated ciphertext *C* or generate a trapdoor *TD*.

- ExtractPublicKey(*PP*, *SV*). Take the public parameters *PP* and a user's secret value *SV* as input, and output the user's public key *PK*. This algorithm is run by the user, and anyone can use the *PK* to generate the ciphertext *C*.
- Encryption(*PP*, *ID*, *t*, *PK*, *M*). Take the public parameters *PP*, a user's identity $ID \in \{0, 1\}^*$, a time period *t*, the user's public key *PK* and a message *M* as input, and output a ciphertext *C* or an error symbol $\perp$ to denote encryption failure. This algorithm is run by a sender.
- Decryption(*PP*,*FSK*,*C*). Take the public parameters *PP*, a user's full secret key *FSK*, and a ciphertext *C* as input, and output a corresponding message *M* or an error symbol $\perp$ to denote decryption failure. This algorithm is run by a receiver.
- Authorization(*PP*, *FSK*). Take the public parameters *PP* and a user's full secret key *FSK* as input, and output the user's trapdoor *TD*. This algorithm is run by the user who can authorize the cloud server to test ciphertexts with *TD*.
- Test(PP, C_ζ, TD_ζ, C_η, TD_η). Take the public parameters *PP*, two tuples (C_ζ, TD_ζ), (C_η, TD_η) as input, and output 1 if C_ζ and C_η are encrypted from the same message. Otherwise, output 0. Here, the ciphertext C_ζ and the trapdoor TD_ζ are from the user ζ, and the ciphertext C_η and the trapdoor TD_η are from the user η. This algorithm is run by a cloud server who has the trapdoors.

3.2. Security Notions

Before defining the security notions of RCL-PKEET, we discuss the types of adversaries. Four types of adversaries have been formally defined in CL-PKEET [29]. In addition to these four types, the types of adversaries of RCL-PKEET include the other two types named revoked users with and without the trapdoor. The six types of adversaries are detailed in the following way.

- Type-1 adversary: it is an outsider who is not a member in the system, but the adversary can replace the user's public key *PK* and obtain any user's time update key *TUK* from a public channel.
- Type-2 adversary: it is a malicious KGC who has the master secret key *msk*. The adversary can compute any user's partial secret key *PSK* and time update key *TUK*.
- Type-3 adversary: it was a member in the system, but now has been revoked by KGC. However, she/he still keeps own partial secret key *PSK* but cannot obtain the current time update key *TUK* from KGC.
- Type-4 adversary: besides the type-1 adversary's abilities, the type-4 adversary possesses the ability to obtain the trapdoor.
- Type-5 adversary: besides the type-2 adversary's abilities, the type-5 adversary possesses the ability to obtain the trapdoor.
- Type-6 adversary: besides the type-3 adversary's abilities, the type-6 adversary possesses the ability to obtain the trapdoor.

Then, we define two new security games, namely $G_{IND\text{-}CCA}$ and $G_{OW\text{-}CCA}$, to model our security notions. The two games $G_{IND\text{-}CCA}$ and $G_{OW\text{-}CCA}$ satisfy the IND-CCA and OW-CCA security notions, respectively. Assume that $\mathcal{A}$ is the adversary and $\mathcal{B}$ is the challenger in the security games. To simplify our description of security games, we present seven queries in advance before playing the security games. $\mathcal{A}$ may issue a number of queries many times to $\mathcal{B}$ as follows:

- *Partial secret key query(ID)*: $\mathcal{B}$ runs ExtractPartialSecretKey algorithm on ID, and forwards the resulting partial secret key PSK to $\mathcal{A}$.
- *Time update key query*(*ID*, *t*): $\mathcal{B}$ runs ExtractTimeUpdateKey algorithm on (*ID*, *t*), and forwards the resulting time update key *TUK* to $\mathcal{A}$.
- *Full secret key query*(*ID*, *t*): $\mathcal{B}$ runs ExtractFullSecretKey algorithm on (*ID*, *t*), and forwards the resulting full secret key *FSK* to $\mathcal{A}$.
- *Public key query(ID)*: $\mathcal{B}$ runs ExtractPublicKey algorithm on ID, and forwards the resulting public key PK to $\mathcal{A}$.
- *Replace public key query*(*ID*, *PK'*): after receiving this query with (*ID*, *PK'*) from $\mathcal{A}$, $\mathcal{B}$ replaces the public key of user *ID* with *PK'*.
- *Decryption query*(*ID*, *t*, *C*): $\mathcal{B}$ runs Decryption algorithm on (*ID*, *t*, *C*), and forwards the resulting message *M* to $\mathcal{A}$.
- *Authorization query*(*ID*, *t*): $\mathcal{B}$ runs Authorization

algorithm on (ID, t), and forwards the resulting trapdoor TD to $\mathcal{A}$.

We say that a RCL-PKEET scheme has the security of indistinguishability under chosen ciphertext attack (IND-CCA) if any PPT adversary $\mathcal{A}$ has no advantage in following security game $G_{IND\text{-}CCA}$ with a challenger $\mathcal{B}$. Define $Adv^{IND\text{-}CCA}_{RCL\text{-}PKEET}(\lambda)$ as $\mathcal{A}$'s advantage which is negligible. Note that the adversary includes the type-1, type-2 and type-3.

1. ***Setup***: $\mathcal{B}$ executes the Setup(λ) algorithm to generate the public parameters PP and the master secret key msk. Then PP is given to $\mathcal{A}$. If $\mathcal{A}$ is the type-2 adversary, $\mathcal{B}$ gives the master key msk to $\mathcal{A}$. Otherwise, the master key msk is kept by $\mathcal{B}$.
2. ***Phase* 1**: $\mathcal{A}$ may issue the *Partial secret key query*, *Time update key query*, *Full secret key query*, *Public key query*, *Replace public key query*, *Decryption query*, and *Authorization query* as mentioned above for many times. A restriction is that $\mathcal{A}$ should not issue the *Full secret key query* if the public key with the same identity ID has been replaced. Note that, if $\mathcal{A}$ is the type-2 adversary, $\mathcal{A}$ can compute the partial secret keys and time update keys by himself/herself without issuing *Partial secret key query* and *Time update key query*. However, the type-2 adversary cannot issue the *Replace public key query*.
3. ***Challenge***: $\mathcal{A}$ submits two messages M_0^*, M_1^*, a time period t^*, and an identity ID^* to $\mathcal{B}$. Three restrictions are given as the following:
   - If $\mathcal{A}$ is type-1 adversary, ID^* and t^* must not be issued in the *Partial secret key query*, *Full secret key query*, *Authorization query* and *Decryption query*.
   - If $\mathcal{A}$ is type-2 adversary, ID^* and t^* must not be issued in the *Full secret key query*, *Authorization query* and *Decryption query*.
   - If $\mathcal{A}$ is type-3 adversary, ID^* and t^* must not be issued in the *Time update key query*, *Full secret key query*, *Authorization query* and *Decryption query*.

$\mathcal{B}$ picks a random bit $b \in \{0, 1\}$, and then runs the Encryption(PP, ID^*, t^*, PK^*, M_b^*) algorithm to compute C^* as the challenge ciphertext. If C^* is invalid, outputs with failure symbol $\perp$. Otherwise, $\mathcal{B}$ sends C^* to $\mathcal{A}$.

4. ***Phase* 2**: $\mathcal{A}$ issues queries under the restrictions which are given above and $\mathcal{B}$ responds as in *Phase* 1.
5. ***Guess***: $\mathcal{A}$ submits a guess $b' \in \{0, 1\}$. $\mathcal{A}$ wins this game if $b = b'$. We define that the advantage of $\mathcal{A}$ is $Adv^{IND\text{-}CCA}_{RCL\text{-}PKEET}(\lambda) = |\Pr[b = b'] - 1/2|$.

We say that a RCL-PKEET scheme is one-way secure against the chosen ciphertext attack (OW-CCA) if any PPT adversary $\mathcal{A}$ has no advantage in following security game $G_{OW\text{-}CCA}$ with a challenger $\mathcal{B}$. Define $Adv^{OW\text{-}CCA}_{RCL\text{-}PKEET}(\lambda)$ as $\mathcal{A}$'s advantage which is negligible. Note that the adversary includes the type-4, type-5 and type-6.

1. ***Setup***: $\mathcal{B}$ executes the Setup(λ) algorithm to generate the public parameters PP and the master secret key msk. Then PP is given to $\mathcal{A}$. If $\mathcal{A}$ is the type-5 adversary, $\mathcal{B}$ gives the master key msk to $\mathcal{A}$. Otherwise, the master key msk is kept by $\mathcal{B}$.
2. ***Phase* 1**: $\mathcal{A}$ may issue the *Partial secret key query*, *Time update key query*, *Full secret key query*, *Public key query*, *Replace public key query*, *Decryption query*, and *Authorization query* as mentioned above for many times. A restriction is that $\mathcal{A}$ should not issue the *Full secret key query* if the public key with the same identity ID has been replaced. Note that, if $\mathcal{A}$ is the type-5 adversary, $\mathcal{A}$ can compute the partial secret keys and time update keys by himself/herself without issuing *Partial secret key query* and *Time update key query*. However, the type-5 adversary cannot issue the *Replace public key query*.
3. ***Challenge***: $\mathcal{A}$ submits an identity ID^*, and a time period t^* to $\mathcal{B}$. Three restrictions are given as the following:
   - If $\mathcal{A}$ is type-4 adversary, ID^* and t^* must not be issued in the *Partial secret key query*, *Full secret key query* and *Decryption query*.
   - If $\mathcal{A}$ is type-5 adversary, ID^* and t^* must not be issued in the *Full secret key query* and *Decryption query*.
   - If $\mathcal{A}$ is type-6 adversary, ID^* and t^* must not be issued in the *Time update key query*, *Full secret key query* and *Decryption query*.

$\mathcal{B}$ picks a random message M^*, and then runs the Encryption(PP, ID^*, t^*, PK^*, M^*) algorithm to compute C^* as the challenge ciphertext. Then $\mathcal{B}$ sends C^* to $\mathcal{A}$.

1. ***Phase* 2**: $\mathcal{A}$ issues queries under the restrictions which are given above and $\mathcal{B}$ responds as in *Phase* 1.
2. ***Guess***: $\mathcal{A}$ submits a guess M'. $\mathcal{A}$ wins this game if $M^* = M'$. We define that the advantage of $\mathcal{A}$ is $Adv^{OW\text{-}CCA}_{RCL\text{-}PKEET}(\lambda) = \Pr[M^* = M']$.

4. The RCL-PKEET Scheme

The concrete RCL-PKEET scheme is composed of ten algorithms and the details are presented as follows.

- Setup(λ). Take a security parameter λ as input and generate $\mathcal{G} = (q, G_1, G_2, G_T, e)$ as mentioned in section 2. Select two generators $P \in G_1$, $Q \in G_2$ and a master secret key $msk = s \in Z_q^*$, and then calculate $P_{pub} = P^s$. Pick eight hash functions H_1: $\{0,1\}^* \to G_2$, H_2: $\{0,1\}^* \to G_2$, H_3: $\{0,1\}^* \to G_2$, H_4: $\{0,1\}^* \to G_2$, H_5: $G_T \times G_1^2 \to \{0,1\}^{\lambda+l}$, H_6: $\{0,1\}^{\lambda} \to G_2$, H_7: $\{0,1\}^{\lambda+1} \to Z_q^*$, H_8: $G_T \to G_2$. Output public parameters $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$.
- ExtractPartialSecretKey(PP, ID, msk). Take public parameters PP, an identity ID and the master secret key msk as input. Output a partial secret key $PSK = (PSK_1, PSK_2) = (H_1(ID)^{msk}, H_2(ID)^{msk}) = (H_1(ID)^s, H_2(ID)^s)$.
- ExtractTimeUpdateKey(PP, ID, t, msk). Take public parameters PP, an identity ID, a time period t and the master secret key msk as input. Output a time update key $TUK = (TUK_1, TUK_2) = (H_3(ID, t)^{msk}, H_4(ID, t)^{msk}) = (H_3(ID, t)^s, H_4(ID, t)^s)$.
- SetSecretValue(PP). Take public parameters PP as input. Then, select a random value $x \in Z_q^*$ and output secret value $SV = x$.
- ExtractFullSecretKey(PP, PSK, TUK, SV). Take public parameters PP, a partial secret key PSK, a time update key TUK and a secret value SV as input. Output a full secret key $FSK = (FSK_1, FSK_2) = ((PSK_1 \cdot TUK_1)^{SV}, (PSK_2 \cdot TUK_2)^{SV}) = ((PSK_1 \cdot TUK_1)^x, (PSK_2 \cdot TUK_2)^x)$.
- ExtractPublicKey(PP, SV). Take public parameters PP and a secret value SV as input. Output a public key $PK = (PK_1, PK_2) = (P_{pub}{}^{SV}, Q^{SV}) = (P_{pub}{}^x, Q^x)$.
- Encryption(PP, ID, t, PK, M). Take public parameters PP, an identity ID, a time period t, the public key PK and a message M as input, where $M \in \{0,1\}^{\lambda}$, and $PK = (PK_1, PK_2)$.
  - Check whether $e(PK_1, Q) = e(P_{pub}, PK_2)$ holds. If not holds, the algorithm aborts with failure.
  - Choose $k \in \{0, 1\}^l$ and use the message M to calculate $R = H_7(M, k)$.
  - Randomly pick $\alpha \in Z_q^*$ and set a ciphertext C by computing (C_1, C_2, C_3, C_4) as follows:
  - $C_1 = P^R$, $C_2 = P^\alpha$, $C_3 = H_5(e(PK_1, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2) \oplus (M \,||\, k)$, $C_4 = H_6(M)^R \cdot H_8(e(PK_1, H_2(ID) \cdot H_4(ID, t))^\alpha)$.
- Decryption(PP, FSK, C). Take public parameters PP, a full secret key FSK, and a ciphertext C as input.
  - Obtain $M' \,||\, k'$ by computing $C_3 \oplus H_5(e(C_2, FSK_1), C_1, C_2)$.
  - Compute $R' = H_7(M', k')$.
  - Check if $C_1 = P^{R'}$ and $C_4 = H_6(M)^{R'} \cdot H_8(e(C_2, FSK_2))$ both hold, return M; otherwise, output failed.
- Authorization(PP, FSK). Take public parameters PP and a full secret key FSK as input. Output a trapdoor $TD = FSK_2$.
- Test(PP, C_ζ, TD_ζ, C_η, TD_η). Take public parameters PP, two ciphertext C_ζ, C_η and two trapdoor TD_ζ, TD_η as input, where $C_\zeta = (C_{\zeta1}, C_{\zeta2}, C_{\zeta3}, C_{\zeta4})$, $C_\eta = (C_{\eta1}, C_{\eta2}, C_{\eta3}, C_{\eta4})$.
  - Compute T_ζ and T_η as below.
    - $$T_\zeta = \frac{C_{\zeta4}}{H_8(e(C_{\zeta2}, TD_\zeta))} = \frac{H_6(M_\zeta)^{R_\zeta} \cdot H_8(e(PK_{\zeta1}, H_2(ID_\zeta) \cdot H_4(ID_\zeta, t_\zeta))^{\alpha_\zeta})}{H_8(e(P^{\alpha_\zeta}, (PSK_{\zeta2} \cdot TUK_{\zeta2})^{x_\zeta}))} = \frac{H_6(M_\zeta)^{R_\zeta} \cdot H_8(e(P_{pub}{}^{x_\zeta}, H_2(ID_\zeta) \cdot H_4(ID_\zeta, t_\zeta))^{\alpha_\zeta})}{H_8(e(P^{\alpha_\zeta}, H_2(ID_\zeta)^s \cdot H_4(ID_\zeta, t_\zeta)^s)^{x_\zeta}))} = \frac{H_6(M_\zeta)^{R_\zeta} \cdot H_8(e(P^{s.x_\zeta}, H_2(ID_\zeta) \cdot H_4(ID_\zeta, t_\zeta))^{\alpha_\zeta})}{H_8(e(P^{s.x_\zeta}, H_2(ID_\zeta) \cdot H_4(ID_\zeta, t_\zeta))^{\alpha_\zeta})} = H_6(M_\zeta)^{H_7(M_\zeta, k_\zeta)}$$
    - $$T_\eta = \frac{C_{\eta4}}{H_8(e(C_{\eta2}, TD_\eta))} = \frac{H_6(M_\eta)^{R_\eta} \cdot H_8(e(PK_{\eta1}, H_2(ID_\eta) \cdot H_4(ID_\eta, t_\eta))^{\alpha_\eta})}{H_8(e(P^{\alpha_\eta}, (PSK_{\eta2} \cdot TUK_{\eta2})^{x_\eta}))} = \frac{H_6(M_\eta)^{R_\eta} \cdot H_8(e(P_{pub}{}^{x_\eta}, H_2(ID_\eta) \cdot H_4(ID_\eta, t_\eta))^{\alpha_\eta})}{H_8(e(P^{\alpha_\eta}, H_2(ID_\eta)^s \cdot H_4(ID_\eta, t_\eta)^s)^{x_\eta}))} = \frac{H_6(M_\eta)^{R_\eta} \cdot H_8(e(P^{s.x_\eta}, H_2(ID_\eta) \cdot H_4(ID_\eta, t_\eta))^{\alpha_\eta})}{H_8(e(P^{s.x_\eta}, H_2(ID_\eta) \cdot H_4(ID_\eta, t_\eta))^{\alpha_\eta})} = H_6(M_\eta)^{H_7(M_\eta, k_\eta)}$$
  - Calculate $e(C_{\zeta1}, T_\eta)$ and $e(C_{\eta1}, T_\zeta)$ as below.
    - $$e(C_{\zeta1}, T_\eta) = e(P^{H_7(M_\zeta, k_\zeta)}, H_6(M_\eta)^{H_7(M_\eta, k_\eta)}) = e(P, H_6(M_\eta))^{H_7(M_\zeta, k_\zeta) \cdot H_7(M_\eta, k_\eta)}$$
    - $$e(C_{\eta1}, T_\zeta) = e(P^{H_7(M_\eta, k_\eta)}, H_6(M_\zeta)^{H_7(M_\zeta, k_\zeta)}) = e(P, H_6(M_\zeta))^{H_7(M_\zeta, k_\zeta) \cdot H_7(M_\eta, k_\eta)}$$

– Check $e(C_{\zeta 1}, T_\eta) = e(C_{\eta 1}, T_\zeta)$. If it holds, output 1; otherwise 0.

In the following, we state the rationality of the proposed RCL-PKEET. We first discuss the user revocation processes, and then prove that the revoked user cannot decrypt the associated ciphertext. The full secret key FSK of each user contains $PSK = (PSK_1, PSK_2)$, $TUK = (TUK_1, TUK_2)$ and SV, since $FSK = (FSK_1, FSK_2) = ((PSK_1 \cdot TUK_1)^{SV}, (PSK_2 \cdot TUK_2)^{SV})$. Among these keys, only TUK includes the current time period t due to $TUK = (TUK_1, TUK_2) = (H_3(ID, t)^{msk}, H_4(ID, t)^{msk}) = (H_3(ID, t)^s, H_4(ID, t)^s)$. As a result, TUK is used to revoke a user when stopping sending it to the user. Next, we prove that only non-revoked user with the current FSK can decrypt the associated ciphertext.

– According to the above Encryption algorithm, the ciphertext $C = (C_1, C_2, C_3, C_4)$, where
– $C_1 = P^R$, $C_2 = P^\alpha$, $C_3 = H_5(e(PK_1, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2) \oplus (M \,||\, k)$ and $C_4 = H_6(M)^R \cdot H_8(e(PK_1, H_2(ID) \cdot H_4(ID, t))^\alpha)$.
– Non-revoked user with $FSK = (FSK_1, FSK_2)$ can obatin M by computing

$$\begin{aligned}
&C_3 \oplus H_5(e(C_2, FSK_1), C_1, C_2)\\
&= H_5(e(PK_1, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2) \oplus (M||k)\\
&\quad \oplus H_5(e(C_2, FSK_1), C_1, C_2)\\
&= H_5(e(P_{pub}{}^{SV}, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2) \oplus (M||k)\\
&\quad \oplus H_5(e(P^\alpha, (PSK_1 \cdot TUK_1)^{SV}), C_1, C_2)\\
&= H_5(e(P_{pub}{}^{SV}, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2) \oplus (M||k)\\
&\quad \oplus H_5(e(P^\alpha, (H_1(ID)^s \cdot H_3(ID, t)^s)^{SV}), C_1, C_2)\\
&= H_5(e(P_{pub}{}^{SV}, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2) \oplus (M||k)\\
&\quad \oplus H_5(e(P^{s \cdot SV}, (H_1(ID) \cdot H_3(ID, t))^\alpha), C_1, C_2)\\
&= H_5(e(P_{pub}{}^{SV}, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2) \oplus (M||k)\\
&\quad \oplus H_5(e(P_{pub}{}^{SV}, H_1(ID) \cdot H_3(ID, t))^\alpha, C_1, C_2)\\
&= M||k
\end{aligned}$$

5. Security Proof

In this section, we propose a formal security proof for RCL-PKEET by using the technique [33]. Based on the assumed hard BDH problem, we give seven theorems to prove the security of the proposed RCL-PKEET scheme.

Theorem 1. *Assume that there exists PPT Type-1 adversary $\mathcal{A}_1$ against IND-CCA security for the proposed scheme in the random oracle model. Then, $\mathcal{A}_1$ has the advantage ϵ to break the scheme. By the ϵ from $\mathcal{A}_1$, we construct that a challenger $\mathcal{B}$ solves the BDH assumption with the advantage ϵ' and $\epsilon' \geq (1/q_{H5})\,[\in/(e(q_{PSK} + q_{FSK} + q_{Auth} + 1)) - q_D/q - q_{H8}/q]$. Suppose that the eight hash functions H_i $(1 \leq i \leq 8)$ are random oracles and then $\mathcal{A}_1$ can issue random oracle queries $q_{H_i}(1 \leq i \leq 8)$. Moreover, $\mathcal{A}_1$ also can issue Partial secret key queries q_{PSK}, Time update key queries q_{TUK}, Full secret key queries q_{FSK}, Public key queries q_{PK}, Replace public key queries q_{RPK}, Decryption queries q_D and Authorization queries q_{Auth} to the challenger $\mathcal{B}$.*

Proof. Assume that $(\mathcal{G}, P, P^a, P^c, Q, Q^a, Q^b)$ is an instance of the BDH problem where $\mathcal{G} = (q, G_1, G_2, G_T, e)$, and $\mathcal{B}$ would like to calculate the BDH solution $e(P, Q)^{abc}$. $\mathcal{B}$ acts as a challenger and interacts with the Type-1 adversary $\mathcal{A}_1$ to calculate $e(P, Q)^{abc}$ in the following $G_{IND\text{-}CCA}$ game:

1 ***Setup***: $\mathcal{B}$ sets $P_{pub} = P^a$ and selects eight collision-resistant hash functions H_i $(1 \leq i \leq 8)$ as random oracles. Then $\mathcal{B}$ outputs the public parameters PP to $\mathcal{A}_1$, where $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$. To keep the consistency between the random oracle queries and the corresponding responses, $\mathcal{B}$ needs to maintain the lists $L_{H1}, L_{H2}, \ldots, L_{H8}, L_{Key}$ as below, which are empty initially and the details of elements in the lists will be introduced later:
   – L_{H1} with items of the forms $[ID_i, \mu_i, cn]$,
   – L_{H2} with items of the forms $[ID_i, v_i, cn]$,
   – L_{H3} with items of the forms $[ID_i, t_i, \eta_i, cn]$,
   – L_{H4} with items of the forms $[ID_i, t_i, \zeta_i, cn]$,
   – L_{H5} with items of the forms $[W, C_1, C_2, \omega]$,
   – L_{H6} with items of the forms $[M, R]$,
   – L_{H7} with items of the forms $[M, k, \gamma]$,
   – L_{H8} with items of the forms $[N, S]$,
   – L_{Key} with items of the forms $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$.

Note that $\mathcal{B}$ maintains the list L_{Key} by the answer to the *Public key query*.

2 ***Phase 1***: $\mathcal{A}_1$ launches a series of queries to $\mathcal{B}$, and then $\mathcal{B}$ returns the corresponding answers as follows.

– H_1 *query*(ID_i): by completing the following steps, $\mathcal{B}$ can answer this query.

- If ID_i exists in L_{H1}, $\mathcal{B}$ searches the tuple $[ID_i, \mu_i, cn]$ by ID_i. Upon obtaining μ_i and cn from L_{H1}, compute:
  - If $cn = 0$, $\mathcal{B}$ returns Q^{μ_i}) to $\mathcal{A}_1$.
  - If $cn = 1$, $\mathcal{B}$ returns $Q^{b\mu_i}$ to $\mathcal{A}_1$.
- Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate μ_i, cn and store them in L_{H1}. Then repeat the above step to return Q^{μ_i} or $Q^{b\mu_i}$.

– H_2 *query*(ID_i): by completing the following steps, $\mathcal{B}$ can answer this query.

- If ID_i exists in L_{H2}, $\mathcal{B}$ searches the tuple $[ID_i, v_i, cn]$ by ID_i. Upon obtaining v_i and cn from L_{H2}, compute:
  - If $cn = 0$, $\mathcal{B}$ returns Q^{v_i} to $\mathcal{A}_1$.
  - If $cn = 1$, $\mathcal{B}$ returns Q^{bv_i} to $\mathcal{A}_1$.
- Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate v_i, cn and store them in L_{H2}. Then repeat the above step to return Q^{v_i} or Q^{bv_i}.

– H_3 *query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.

- If (ID_i, t_i) exists in L_{H3}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, \eta_i, cn]$ by (ID_i, t_i). Upon obtaining η_i and cn from L_{H3}, compute Q^{η_i} as the answer to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i). to generate η_i, cn and store them in L_{H3}. Then repeat the above step to return Q^{η_i}.

– H_4 *query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.

- If (ID_i, t_i) exists in L_{H4}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, \zeta_i, cn]$ by (ID_i, t_i). Upon obtaining ζ_i and cn from L_{H4}, compute Q^{ζ_i} as the answer to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate ζ_i, cn and store them in L_{H4}. Then repeat the above step to return Q^{ζ_i}.

– H_5 *query*(W, C_1, C_2): by completing the following steps, $\mathcal{B}$ can answer this query.

- If (W, C_1, C_2) exists in L_{H5}, $\mathcal{B}$ searches the tuple $[W, C_1, C_2, \omega]$ by (W, C_1, C_2), and returns ω to $\mathcal{A}_1$.
- Otherwise $\mathcal{B}$ randomly selects $\omega \in \{0, 1\}^{\lambda+l}$ as the answer to $\mathcal{A}_1$ and stores $[W, C_1, C_2, \omega]$ into L_{H5}.

– H_6 *query*(M): by completing the following steps, $\mathcal{B}$ can answer this query.

- If M exists in L_{H6}, $\mathcal{B}$ searches the tuple $[M, R]$ by M, and returns R to $\mathcal{A}_1$.
- Otherwise $\mathcal{B}$ randomly selects $R \in G_2$ as the answer to $\mathcal{A}_1$ and stores $[M, R]$ into L_{H6}.

– H_7 *query*(M, k): by completing the following steps, $\mathcal{B}$ can answer this query.

- If (M, k) exists in L_{H7}, $\mathcal{B}$ searches the tuple $[M, k, \gamma]$ by (M, k), and returns γ to $\mathcal{A}_1$.
- Otherwise $\mathcal{B}$ randomly selects $\gamma \in Z_q^*$ as the answer to $\mathcal{A}_1$ and stores $[M, k, \gamma]$ into L_{H7}.

– H_8 *query*(N): by completing the following steps, $\mathcal{B}$ can answer this query.

- If N exists in L_{H8}, $\mathcal{B}$ searches the tuple $[N, S]$ by N, and returns S to $\mathcal{A}_1$.
- Otherwise $\mathcal{B}$ randomly selects $S \in G_2$ as the answer to $\mathcal{A}_1$ and stores $[N, S]$ into L_{H8}.

– *Public key query*(ID_i, t_i): after receiving this query on (ID_i, t_i), $\mathcal{B}$ randomly selects $\mu_i, v_i, \eta_i, \zeta_i \in Z_q^*$, $cn \in \{0, 1\}$ with $\Pr[cn = 0] = \tau$, and then adds four tuples $[ID_i, \mu_i, cn]$, $[ID_i, v_i, cn]$, $[ID_i, t_i, \eta_i, cn]$, $[ID_i, t_i, \zeta_i, cn]$ into $L_{H1}, L_{H2}, L_{H3}, L_{H4}$ respectively.

- If $cn = 0$, $\mathcal{B}$ executes the SetSecretValue(PP) algorithm to get the secret value x_i, then computes $PSK_i = (PSK_{i,1}, PSK_{i,2}) = (Q^{a\mu_i}, Q^{av_i})$, $TUK_i = (TUK_{i,1}, TUK_{i,2}) = (Q^{a\eta_i}, Q^{a\zeta_i})$, $FSK_i = (FSK_{i,1}, FSK_{i,2}) = ((PSK_{i,1} \cdot TUK_{i,1})^{x_i}, (PSK_{i,2} \cdot TUK_{i,2})^{x_i})$ and $PK_i = (PK_{i,1}, PK_{i,2}) = (P_{pub}^{x_i}, Q^{x_i})$, adds an tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, 0]$ into L_{Key}, and returns PK_i to $\mathcal{A}_1$.
- Otherwise, $\mathcal{B}$ executes the SetSecretValue(PP) algorithm to get the secret value x_i, then computes $PK = (PK_1, PK_2) = (P_{pub}^{x_i}, Q^{x_i})$, $TUK_i = (TUK_{i,1}, TUK_{i,2}) = (Q^{a\eta_i}, Q^{a\zeta_i})$ adds an tuple $[ID_i, t_i, x_i, -, TUK_i, -, PK_i, 1]$ into L_{Key}, and returns PK_i to $\mathcal{A}_1$.

– *Partial secret key query*(ID_i): by completing the following steps, $\mathcal{B}$ can answer this query.

- If ID_i exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by ID_i, and execute the task as the following:
  - If $cn = 0$, $\mathcal{B}$ returns PSK_i to $\mathcal{A}_1$.
  - If $cn = 1$, $\mathcal{B}$ aborts the game.
- Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate

PSK_i, cn. Then repeat the above step to return PSK_i or abort the game.

- *Time update key query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If (ID_i, t_i) exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by (ID_i, t_i), and returns TUK_i to $\mathcal{A}_1$.
  - Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate TUK_i and returns TUK_i to $\mathcal{A}_1$.
- *Full secret key query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If (ID_i, t_i) exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by (ID_i, t_i), and execute the task as the following:
    - If $cn = 0$, $\mathcal{B}$ returns FSK_i to $\mathcal{A}_1$.
    - If $cn = 1$, $\mathcal{B}$ aborts the game.
  - Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate FSK_i, cn. Then repeat the above step to return FSK_i or abort the game.
- *Replace public key query*(ID_i, PK_i'): after receiving this query on (ID_i, PK_i'), $\mathcal{B}$ replaces the existing PK_i of the corresponding ID_i with PK_i'.
  - If it satisfies $e(PK_{i,1}', Q) = e(P_{pub}, PK_{i,2}')$, $\mathcal{B}$ keeps the change.
  - Otherwise $\mathcal{B}$ returns $\perp$ to $\mathcal{A}_1$.
- *Decryption query*(ID_i, t_i, C): after receiving this query on (ID_i, t_i, C) where $C = (C_1, C_2, C_3, C_4)$, $\mathcal{B}$ performs the following tasks.
  - If (ID_i, t_i) exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by (ID_i, t_i), and execute the task as the following:
    - If $cn = 0$, and the public key has not been replaced by $\mathcal{A}_1$, $\mathcal{B}$ uses FSK_i from the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, 0]$ in L_{Key} to execute the Decryption(PP, FSK_i, C) algorithm and returns the output to $\mathcal{A}_1$.
    - If $cn = 1$, $\mathcal{B}$ searches the tuple $[W, C_1, C_2, \omega]$ in L_{H5} by C_1, C_2, and calculates $M' \,||\, k' = C_3 \oplus \omega$. Next, (M', k') is used to search the tuple $[M, k, \gamma]$ in L_{H7}. After obtaining γ, compute the P^γ. Then retrieve the tuple $[M, R]$ in L_{H6} by M' to get R. If find the S in the tuple $[N, S]$ in L_{H8} such that $C_4 = R \cdot S$ holds, $\mathcal{B}$ will check whether $C_1 = P^\gamma$ holds. When both $C_1 = P^\gamma$ and $C_4 = R \cdot S$ holds, return M' to $\mathcal{A}_1$. $\mathcal{B}$ returns $\perp$ to $\mathcal{A}_1$ if $\mathcal{B}$ cannot search the tuple in L_{H5}.
  - Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate FSK_i, cn. Then repeat the above step to return M.
- *Authorization query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.
- If (ID_i, t_i) exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by (ID_i, t_i), and:
    - If $cn = 0$, $\mathcal{B}$ returns $FSK_{i,2}$ to $\mathcal{A}_1$, where $FSK_i = (FSK_{i,1}, FSK_{i,2})$.
    - If $cn = 1$, $\mathcal{B}$ aborts the game.
  - Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate FSK_i, cn. Then repeat the above step to return $FSK_{i,2}$ or abort the game.

3 ***Challenge***: $\mathcal{A}_1$ sends an identity ID^*, a time period t^* and two different messages $M_0^*, M_1^* \in \{0,1\}^\lambda$ to $\mathcal{B}$ for challenge. $\mathcal{B}$ uses (ID^*, t^*) as an input to produce *Public key query* and get the tuple $[ID^*, t^*, x^*, PSK^*, TUK^*, FSK^*, PK^*, cn]$ from L_{Key}.
- If $cn = 0$, $\mathcal{B}$ aborts the game.
- If $cn = 1$, $\mathcal{B}$ performs the following tasks:
  - Select $b \in \{0,1\}$, $k \in \{0,1\}^l$, $C_3^* \in \{0,1\}^{\lambda+l}$ and $C_4^* \in G_2$ at random.
  - Set $C_2^* = P^c$.
  - Obtain γ by H_7 *query*(M_b^*, k) and set $C_1^* = P^\gamma$.
  - Return $C^* = (C_1^*, C_2^*, C_3^*, C_4^*)$ to $\mathcal{A}_1$.

Based on the above construction, $H_5(e(P, Q)^{abcx^*\mu^*} \cdot e(P^{ac}, Q)^{x^*\eta^*}, C_1^*, C_2^*) = (M_b^* \,||\, k) \oplus C_3^*$ and $H_8(e(P, Q)^{abcx^*v^*} \cdot e(P^a, Q)^{cx^*\zeta^*}) = C_4^* / (H_6(M_b^*)^R)$, where $Q^{b\mu^*} = H_1(ID^*)$ and $Q^{bv^*} = H_2(ID^*)$.

4 ***Phase 2***: $\mathcal{A}_1$ launches a series of queries to $\mathcal{B}$ as in ***Phase 1***.

5 ***Guess***: eventually, $\mathcal{A}_1$ outputs $b' \in \{0,1\}$ as the guess bit. If $b' = b$, $\mathcal{A}_1$ wins the game; otherwise loses the game. $\mathcal{B}$ chooses a random tuple $[\sigma^*, C_1^*, C_2^*, \theta]$ from L_{H5} and outputs $(\sigma^*/e(P^{ac}, Q)^{x^*\eta^*})^{(x^*\mu^*)^{-1}} = e(P, Q)^{abc})$ as the solution to the BDH instance.

Analysis. We need to evaluate the simulation of the random oracles first. It is clear that H_1, H_2, H_3, H_4, H_6, and H_7 simulations are perfect due to their construction. $AskH_5^*$ is defined as the event that $H_5(e(P, Q)^{abcx^*\mu^*} \cdot e(P^{ac}, Q)^{x^*\eta^*}, C_1^*, C_2^*)$ has been issued by $\mathcal{A}_1$, $AskH_8^*$ is defined as the event that $H_8(e(P, Q)^{abcx^*v^*} \cdot e(P^a, Q)^{cx^*\zeta^*})$ has been issued by $\mathcal{A}_1$. We say that the simulation of H_5 is perfect if $AskH_5^*$ does not happen and the simulation of H_8 is perfect if $AskH_8^*$ does not happen too. Now we assess the simulation of the decryption oracle. $DecErr$ indicates an event in the valid ciphertext, and $\mathcal{B}$ cannot decrypt it exactly during the emulation and we get $\Pr[DecErr] \le q_D/q$.

Next, define $Abort$ as the event that the emulation is aborted by $\mathcal{B}$, and define $Evt = (AskH_5^* \vee AskH_8^* \vee DecErr)|\neg Abort$. $\mathcal{B}$ guess b with the advantage $\le 1/2$ if Evt does not occur due to the randomness of the outputs of H_5 and H_8. So $\Pr[b = b' \mid \neg Evt] \le 1/2$, we obtain

$$\begin{aligned}\Pr[b = b'] &= \Pr[b = b' \mid Evt]\Pr[Evt] + \Pr[b = b' \mid \neg Evt]\Pr[\neg Evt] \\ &\le \Pr[Evt] + (1/2)\Pr[\neg Evt] \\ &= \Pr[Evt] + (1/2)(1-\Pr[Evt]) \\ &= (1/2)\Pr[Evt] + 1/2.\end{aligned} \tag{1}$$

According to (1) and the sense of ϵ, the following equation can be obtained.

$$\begin{aligned}\epsilon &= \Pr[b = b'] - 1/2 \\ &\le \Pr[Evt] \\ &\le (\Pr[AskH_5^*] + \Pr[AskH_8^*] + \Pr[DecErr]) \\ &/ \Pr[\neg Abort].\end{aligned} \tag{2}$$

According to (2), we have:

$\Pr[AskH_5^*] \ge \epsilon \Pr[\neg Abort] - \Pr[DecErr] - \Pr[AskH_8^*]$.

Since $\Pr[\neg Abort] = \tau^{q_{PSK}+q_{FSK}+q_{Auth}}(1-\tau)$, we can obtain $\Pr[\neg Abort] \ge 1/e(q_{PSK}+q_{FSK}+q_{Auth}+1)$ when $\tau = 1-1/(q_{PSK}+q_{FSK}+q_{Auth}+1)$. We then have:

$$\Pr[AskH_5^*] \ge \epsilon/e(q_{PSK}+q_{FSK}+q_{Auth}+1) - q_D/q - q_{h_8}/q. \tag{3}$$

If $AskH_5^*$ occurs, $\mathcal{A}_1$ will distinguish the real one during the simulation and the challenge ciphertext C^* is invalid. Then $H_5(e(P, Q)^{abcx^*\mu^*} \cdot e(P^{ac}, Q)^{x^*\eta^*}, C_1^*, C_2^*)$ has been added in the L_{H5}. $\mathcal{B}$ can pick the right bit from the L_{H5} and wins the game. According to (3), the BDH problem can be solved by $\mathcal{B}$ with the following advantage

$$\begin{aligned}\epsilon' &\ge (1/q_{h_5})\Pr[AskH_5^*] \\ &\ge (1/q_{h_5})[\epsilon/e(q_{PSK}+q_{FSK}+q_{Auth}+1) - q_D/q - q_{h_8}/q].\end{aligned}$$

Theorem 2. *Assume that there exists PPT Type-2 adversary $\mathcal{A}_2$ against IND-CCA security for the proposed scheme in the random oracle model. Then, $\mathcal{A}_2$ has the advantage ϵ to break the scheme. By the ϵ from $\mathcal{A}_2$, we construct that a challenger $\mathcal{B}$ solves the BDH assumption with the advantage ϵ' and $\epsilon' \ge (1/q_{h_5})[\epsilon/(e(q_{FSK} + q_{Auth} + 1)) - q_D/q - q_{h_8}/q]$. Suppose that the eight hash functions $H_i (1 \le i \le 8)$ are random oracles and then $\mathcal{A}_2$ can issue random oracle queries $q_{h_i} (1 \le i \le 8)$. Moreover, $\mathcal{A}_2$ also can issue Full secret key queries q_{FSK}, Public key queries q_{PK}, Decryption queries q_D and Authorization queries q_{Auth} to challenger $\mathcal{B}$. Note that $\mathcal{A}_2$ is a malicious KGC so $\mathcal{A}_2$ cannot issue Partial secret key queries q_{PSK}, Time update key queries q_{TUK}, Replace public key queries q_{RPK}.*

Proof. Assume that $(\mathcal{G}, P, P^a, P^c, Q, Q^a, Q^b)$ is an instance of the BDH problem where $\mathcal{G} = (q, G_1, G_2, G_T, e)$, and $\mathcal{B}$ would like to calculate the BDH solution $e(P, Q)^{abc}$. $\mathcal{B}$ acts as a challenger and interacts with the Type-2 adversary $\mathcal{A}_2$ to calculate $e(P, Q)^{abc}$ in the following $G_{IND\text{-}CCA}$ game:

1 ***Setup***: $\mathcal{B}$ picks the master secret key $s \in Z_q^*$ at random and sets $P_{pub} = P^s$. Then select eight collision-resistant hash functions $H_i (1 \le i \le 8)$ as random oracles. Then $\mathcal{B}$ outputs the master secret key s and the public parameters PP to $\mathcal{A}_2$, where $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$. To keep the consistency between the random oracle queries and the corresponding responses, $\mathcal{B}$ needs to maintain the lists $L_{H1}, L_{H2}, ..., L_{H8}, L_{Key}$, which are similar to the proof of Theorem 1.

– *H_1-H_8 queries*: the queries are identical to the proof of Theorem 1.

– *Public key query*(ID_i, t_i): after receiving this query on (ID_i, t_i), $\mathcal{B}$ randomly selects $\mu_i, v_i, \eta_i, \zeta_i \in Z_q^*$, $cn \in \{0, 1\}$ with $\Pr[cn = 0] = \tau$, and then adds four tuples $[ID_i, \mu_i, cn]$, $[ID_i, v_i, cn]$, $[ID_i, t_i, \eta_i, cn]$, $[ID_i, t_i, \zeta_i, cn]$ into $L_{H1}, L_{H2}, L_{H3}, L_{H4}$ respectively.

  - If $cn = 0$, $\mathcal{B}$ executes the SetSecretValue(PP) algorithm to get the secret value x_i, then computes $PSK_i = (PSK_{i,1}, PSK_{i,2}) = (Q^{s\mu_i}, Q^{sv_i})$, $TUK_i = (TUK_{i,1}, TUK_{i,2}) = (Q^{s\eta_i}, Q^{s\zeta_i})$, $FSK_i = (FSK_{i,1}, FSK_{i,2}) = ((PSK_{i,1} \cdot TUK_{i,1})^{x_i}, (PSK_{i,2} \cdot TUK_{i,2})^{x_i})$ and $PK_i = (PK_{i,1}, PK_{i,2}) = (P_{pub}^{x_i}, Q^{x_i})$, adds an tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, 0]$ into L_{Key}, and returns PK_i to $\mathcal{A}_2$.
  - Otherwise, $\mathcal{B}$ selects $x_i' \in Z_q^*$ at random, then computes $PSK_i = (PSK_{i,1}, PSK_{i,2}) = (Q^{bs\mu_i}, Q^{bsv_i})$, $TUK_i = (TUK_{i,1}, TUK_{i,2}) = (Q^{s\eta_i}, Q^{s\zeta_i})$ and $PK_i = (PK_{i,1}, PK_{i,2}) = (P_{pab}^{ax_i'}, Q^{ax_i'})$ adds an tuple $[ID_i, t_i, x_i', PSK_i, TUK_i, -, PK_i, 1]$ into L_{Key}, and returns

PK_i to $\mathcal{A}_2$. Here, the secret value x_i is seen as ax_i' implicitly.

- *Full secret key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Decryption query*(ID_i, t_i, C): the query is identical to the proof of Theorem 1.
- *Authorization query*(ID_i, t_i): the query is identical to the proof of Theorem 1.

2 ***Phase 1***: $\mathcal{A}_2$ launches a series of queries to $\mathcal{B}$, and then $\mathcal{B}$ returns the corresponding answers as follows.

3 ***Challenge***: $\mathcal{A}_2$ sends an identity ID^*, a time period t^* and two different messages $M_0^*, M_1^* \in \{0, 1\}^\lambda$ to $\mathcal{B}$ for challenge. $\mathcal{B}$ uses (ID^*, t^*) as an input to produce *Public key query* and get the tuple $[ID^*, t^*, x^*, PSK^*, TUK^*, FSK^*, PK^*, cn]$ from L_{Key}.

- If $cn = 0$, $\mathcal{B}$ aborts the game.
- If $cn = 1$, $\mathcal{B}$ performs the following tasks:
  - Select $b \in \{0, 1\}$, $k \in \{0, 1\}^l$, $C_3^* \in \{0, 1\}^{\lambda+l}$ and $C_4^* \in G_2$ at random.
  - Set $C_2^* = P^c$.
  - Obtain γ by H_7 *query*(M_b^*, k) and set $C_1^* = P^\gamma$.
  - Return $C^* = (C_1^*, C_2^*, C_3^*, C_4^*)$ to $\mathcal{A}_2$.

Based on the above construction, $H_5(e(P, Q)^{abcx^*\mu^*s} \cdot e(P^{ac}, Q)^{x^*\eta^*s}, C_1^*, C_2^*) = (M_b^* \| k) \oplus C_3^*$ and $H_8(e(P, Q)^{abcx^*v^*s} \cdot e(P^a, Q)^{cx^*\zeta^*s}) = C_4^* / (H_6(M_b^*)^R)$, where $Q^{b\mu^*} = H_1(ID^*)$ and $Q^{bv^*} = H_2(ID^*)$.

4 ***Phase 2***: $\mathcal{A}_2$ launches a series of queries to $\mathcal{B}$ as in ***Phase 1***.

5 ***Guess***: eventually, $\mathcal{A}_2$ outputs $b' \in \{0, 1\}$ as the guess bit. If $b' = b$, $\mathcal{A}_2$ wins the game; otherwise loses the game. $\mathcal{B}$ chooses a random tuple $[\sigma^*, C_1^*, C_2^*, \theta]$ from L_{H5} and outputs $(\sigma^*/e(P^{ac}, Q)^{x^*\eta^*s})^{(x^*\mu^*s)^{-1}} = e(P, Q)^{abc})$ as the solution to the BDH instance.

Analysis. We need to evaluate the simulation of the random oracles first. It is clear that H_1, H_2, H_3, H_4, H_6, and H_7 simulations are perfect due to their construction. $AskH_5^*$ is defined as the event that $H_5(e(P, Q)^{abcx^*\mu^*s} \cdot e(P^{ac}, Q)^{x^*\eta^*s}, C_1^*, C_2^*)$ has been issued by $\mathcal{A}_1$, $AskH_8^*$ is defined as the event that $H_8(e(P, Q)^{abcx^*v^*s} \cdot e(P^a, Q)^{cx^*\zeta^*s})$ has been issued by $\mathcal{A}_1$. We say that the simulation of H_5 is perfect if $AskH_5^*$ does not happen and the simulation of H_8 is perfect if $AskH_8^*$ does not happen too. Now we assess the simulation of the decryption oracle. $DecErr$ indicates an event in the valid ciphertext, and $\mathcal{B}$ cannot decrypt it exactly during the emulation and we get $\Pr[DecErr] \le q_D/q$.

Next, define $Abort$ as the event that the emulation is aborted by $\mathcal{B}$, and define $Evt = (AskH_5^* \vee AskH_8^* \vee DecErr) | \neg Abort$. $\mathcal{B}$ guess b with the advantage $\le 1/2$ if Evt does not occur due to the randomness of the outputs of H_5 and H_8. So $\Pr[b = b' | \neg Evt] \le 1/2$, we obtain

$$\begin{aligned}\Pr[b = b'] &= \Pr[b = b' | Evt]\Pr[Evt] + \Pr[b = b' | \neg Evt]\Pr[\neg Evt]\\ &\le \Pr[Evt] + (1/2)\Pr[\neg Evt]\\ &= \Pr[Evt] + (1/2)(1 - \Pr[Evt])\\ &= (1/2)\Pr[Evt] + 1/2.\end{aligned} \tag{4}$$

According to (4) and the sense of ϵ, the following equation can be obtained.

$$\begin{aligned}\epsilon &= \Pr[b = b'] - 1/2\\ &\le \Pr[Evt]\\ &\le (\Pr[AskH_5^*] + \Pr[AskH_8^*] + \Pr[DecErr]) / \Pr[\neg Abort].\end{aligned} \tag{5}$$

According to (5), we have:

$\Pr[AskH_5^*] \ge \epsilon \Pr[\neg Abort] - \Pr[DecErr] - \Pr[AskH_8^*]$.

Since $\Pr[\neg Abort] = \tau^{q_{FSK} + q_{Auth}}(1 - \tau)$, we can obtain $\Pr[\neg Abort] \ge 1/e(q_{FSK} + q_{Auth} + 1)$ when $\tau = 1 - 1/(q_{FSK} + q_{Auth} + 1)$. We then have:

$$\Pr[AskH_5^*] \ge \epsilon/e(q_{FSK} + q_{Auth} + 1) - q_D/q - q_{h_8}/q. \tag{6}$$

If $AskH_5^*$ occurs, $\mathcal{A}_2$ will distinguish the real one during the simulation and the challenge ciphertext C^* is invalid. Then $H_5(e(P, Q)^{abcx^*\mu^*s} \cdot e(P^{ac}, Q)^{x^*\eta^*s}, C_1^*, C_2^*)$ has been added in the L_{H5}. $\mathcal{B}$ can pick the right bit from the L_{H5} and wins the game. According to (6), the BDH problem can be solved by $\mathcal{B}$ with the following advantage

$$\begin{aligned}\epsilon' &\ge (1/q_{h_5})\Pr[AskH_5^*]\\ &\ge (1/q_{h_5})[\epsilon/e(q_{FSK} + q_{Auth} + 1) - q_D/q - q_{h_8}/q].\end{aligned}$$

Theorem 3. *Assume that there exists PPT Type-3 adversary $\mathcal{A}_3$ against IND-CCA security for the proposed scheme in the random oracle model. Then, $\mathcal{A}_3$ has the advantage ϵ to break the scheme. By the ϵ from $\mathcal{A}_3$, we construct that a challenger $\mathcal{B}$ solves the BDH assumption with the advantage ϵ' and $\epsilon' \ge (1/q_{h_5})$ $[\epsilon/(e(q_{TUK} + q_{FSK} + q_{Auth} + 1)) - q_D/q - q_{h_8}/q]$. Suppose that the eight* *hash functions H_i $(1 \le i \le 8)$ are random oracles and then $\mathcal{A}_3$ can issue random oracle queries q_{H_i} $(1 \le i \le 8)$. Moreover, $\mathcal{A}_3$ also can issue Partial secret key queries q_{PSK}, Time update key queries q_{TUK}, Full secret key queries q_{FSK}, Public key queries q_{PK}, Replace public key queries q_{RPK}, Decryption queries q_D and Authorization queries q_{Auth} to challenger $\mathcal{B}$.*

Proof. Assume that $(\mathcal{G}, P, P^a, P^c, Q, Q^a, Q^b)$ is an instance of the BDH problem where $\mathcal{G} = (q, G_1, G_2, G_T, e)$, and $\mathcal{B}$ would like to calculate the BDH solution $e(P, Q)^{abc}$. $\mathcal{B}$ acts as a challenger and interacts with the Type-3 adversary $\mathcal{A}_3$ to calculate $e(P, Q)^{abc}$ in the following $G_{IND\text{-}CCA}$ game:

1 ***Setup***: $\mathcal{B}$ sets $P_{pub} = P^a$ and selects eight collision-resistant hash functions H_i $(1 \le i \le 8)$ as random oracles. Then $\mathcal{B}$ outputs the public parameters PP to $\mathcal{A}_3$, where $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$. To keep the consistency between the random oracle queries and the corresponding responses, $\mathcal{B}$ needs to maintain the lists $L_{H1}, L_{H2}, ..., L_{H8}, L_{Key}$, which are similar to the proof of Theorem 1.

2 ***Phase 1***: $\mathcal{A}_3$ launches a series of queries to $\mathcal{B}$, and then $\mathcal{B}$ returns the corresponding answers as follows.

- H_1 *query*(ID_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If ID_i exists in L_{H1}, $\mathcal{B}$ searches the tuple $[ID_i, \mu_i, cn]$ by ID_i. Upon obtaining μ_i and cn from L_{H1}, compute Q^{μ_i} as the answer to $\mathcal{A}_3$.
  - Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate μ_i, cn and store them in L_{H1}. Then repeat the above step to return Q^{μ_i}.
- H_2 *query*(ID_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If ID_i exists in L_{H2}, $\mathcal{B}$ searches the tuple $[ID_i, v_i, cn]$ by ID_i. Upon obtaining v_i and cn from L_{H2}, compute Q^{v_i} as the answer to $\mathcal{A}_3$.
  - Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate v_i, cn and store them in L_{H2}. Then repeat the above step to return Q^{v_i}.
- H_3 *query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If (ID_i, t_i) exists in L_{H3}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, \eta_i, cn]$ by (ID_i, t_i). Upon obtaining η_i and cn from L_{H3}, compute:
    - If $cn = 0$, $\mathcal{B}$ returns Q^{η_i} to $\mathcal{A}_3$.
    - If $cn = 1$, $\mathcal{B}$ returns $Q^{b\eta_i}$ to $\mathcal{A}_3$.
  - Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate v_i, cn and store them in L_{H3}. Then repeat the above step to return Q^{η_i} or $Q^{b\eta_i}$.
- H_4 *query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If (ID_i, t_i) exists in L_{H4}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, \zeta_i, cn]$ by (ID_i, t_i). Upon obtaining ζ_i and cn from L_{H4}, compute:
    - If $cn = 0$, $\mathcal{B}$ returns Q^{ζ_i} to $\mathcal{A}_3$.
    - If $cn = 1$, $\mathcal{B}$ returns $Q^{b\zeta_i}$ to $\mathcal{A}_3$.
  - Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate ζ_i, cn and store them in L_{H4}. Then repeat the above step to return Q^{ζ_i} or $Q^{b\zeta_i}$.
- H_5-H_8 *queries*: the query is identical to the proof of Theorem 1.
- *Public key query*(ID_i, t_i): after receiving this query on (ID_i, t_i), $\mathcal{B}$ randomly selects $\mu_i, v_i, \eta_i, \zeta_i \in Z_q^*$, $cn \in \{0, 1\}$ with $\Pr[cn = 0] = \tau$, and then adds four tuples $[ID_i, \mu_i, cn]$, $[ID_i, v_i, cn]$, $[ID_i, t_i, \eta_i, cn]$, $[ID_i, t_i, \zeta_i, cn]$ into $L_{H1}, L_{H2}, L_{H3}, L_{H4}$ respectively.
  - If $cn = 0$, $\mathcal{B}$ executes the SetSecretValue(PP) algorithm to get the secret value x_i, then computes $PSK_i = (PSK_{i,1}, PSK_{i,2}) = (Q^{a\mu_i}, Q^{av_i})$, $TUK_i = (TUK_{i,1}, TUK_{i,2}) = (Q^{a\eta_i}, Q^{a\zeta_i})$, $FSK_i = (FSK_{i,1}, FSK_{i,2}) = ((PSK_{i,1} \cdot TUK_{i,1})^{x_i}, (PSK_{i,2} \cdot TUK_{i,2})^{x_i})$ and $PK_i = (PK_{i,1}, PK_{i,2}) = (P_{pub}^{x_i}, Q^{x_i})$ adds an tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, 0]$ into L_{Key}, and returns PK_i to $\mathcal{A}_3$.
  - Otherwise, $\mathcal{B}$ executes the SetSecretValue(PP) algorithm to get the secret value x_i, then computes $PK = (PK_1, PK_2) = (P_{pub}^{x_i}, Q^{x_i})$, $PSK_i = (PSK_{i,1}, PSK_{i,2}) = (Q^{a\mu_i}, Q^{av_i})$ adds an tuple $[ID_i, t_i, x_i, PSK_i, -, -, PK_i, 1]$ into L_{Key}, and returns PK_i to $\mathcal{A}_3$.
- *Partial secret key query*(ID_i) : by completing the following steps, $\mathcal{B}$ can answer this query.
  - ID_i exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by ID_i, and returns PSK_i to $\mathcal{A}_3$.
  - Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate PSK_i and returns PSK_i to $\mathcal{A}_3$.

- *Time update key query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If (ID_i, t_i) exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by (ID_i, t_i), and execute the task as the following:
    - If $cn = 0$, $\mathcal{B}$ returns TUK_i to $\mathcal{A}_3$.
    - If $cn = 1$, $\mathcal{B}$ aborts the game.
  - Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate PSK_i, cn. Then repeat the above step to return TUK_i or abort the game.
- *Full secret key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Replace public key query*(ID_i, PK_i'): the query is identical to the proof of Theorem 1.
- *Decryption query*(ID_i, t_i, C): the query is identical to the proof of Theorem 1.
- *Authorization query*(ID_i, t_i): the query is identical to the proof of Theorem 1.

3 ***Challenge***: $\mathcal{A}_3$ sends an identity ID^*, a time period t^* and two different messages $M_0^*, M_1^* \in \{0,1\}^{\lambda}$ to $\mathcal{B}$ for challenge. $\mathcal{B}$ usess (ID^*, t^*) as an input to produce *Public key query* and get the tuple $[ID^*, t^*, x^*, PSK^*, TUK^*, FSK^*, PK^*, cn]$ from L_{Key}.

- If $cn = 0$, $\mathcal{B}$ aborts the game.
- If $cn = 1$, $\mathcal{B}$ performs the following tasks:
  - Select $b \in \{0,1\}$, $k \in \{0,1\}^{l}$, $C_3^* \in \{0,1\}^{\lambda+l}$ and $C_4^* \in G_2$ at random.
  - Set $C_2^* = P^c$.
  - Obtain γ by H_7 *query*(M_b^*, k) and set $C_1^* = P^{\gamma}$.
  - Return $C^* = (C_1^*, C_2^*, C_3^*, C_4^*)$ to $\mathcal{A}_3$.

Based on the above construction, $H_5(e(P^{ac}, Q)^{x^*\mu^*}\cdot e(P, Q)^{abcx^*\eta^*}, C_1^*, C_2^*) = (M_b^* \| k)\oplus C_3^*$ and $H_8(e(P^{ac}, Q)^{x^*\nu^*}\cdot e(P, Q)^{abcx^*\zeta^*}) = C_4^* / (H_6(M_b^*)^R)$, where $Q^{b\eta^*} = H_3(ID^*)$ and $Q^{b\zeta^*} = H_4(ID^*)$.

4 ***Phase 2***: $\mathcal{A}_3$ launches a series of queries to $\mathcal{B}$ as in ***Phase 1***.

5 ***Guess***: eventually, $\mathcal{A}_3$ outputs $b' \in \{0,1\}$ as the guess bit. If $b' = b$, $\mathcal{A}_3$ wins the game; otherwise loses the game. $\mathcal{B}$ chooses a random tuple $[\sigma^*, C_1^*, C_2^*, \theta]$ from L_{H5} and outputs $(\sigma^*/e(P^{ac}, Q)^{x^*\mu^*})^{(x^*\eta^*)^{-1}} = e(P, Q)^{abc}$ as the solution to the BDH instance.

Analysis. We need to evaluate the simulation of the random oracles first. It is clear that H_1, H_2, H_3, H_4, H_6, and H_7 simulations are perfect due to their construction. $AskH_5^*$ is defined as the event that $H_5(e(P^{ac}, Q)^{x^*\mu^*}\cdot e(P, Q)^{abcx^*\eta^*}, C_1^*, C_2^*)$ has been issued by $\mathcal{A}_3$, $AskH_8^*$ is defined as the event that $H_8(e(P^{ac}, Q)^{x^*\nu^*}\cdot e(P, Q)^{abcx^*\zeta^*})$ has been issued by $\mathcal{A}_3$. We say that the simulation of H_5 is perfect if $AskH_5^*$ does not happen and the simulation of H_8 is perfect if $AskH_8^*$ does not happen too. Now we assess the simulation of the decryption oracle. *DecErr* indicates an event in the valid ciphertext, and $\mathcal{B}$ cannot decrypt it exactly during the emulation and we get $\Pr[DecErr] \le q_D/q$.

Next, define *Abort* as the event that the emulation is aborted by $\mathcal{B}$, and define $Evt = (AskH_5^* \vee AskH_8^* \vee DecErr)|\neg Abort$. $\mathcal{B}$ guess b with the advantage $\le 1/2$ if Evt does not occur due to the randomness of the outputs of H_5 and H_8. So $\Pr[b = b' \mid \neg Evt] \le 1/2$, we obtain

$$
\begin{aligned}
&\Pr[b = b'] = \Pr[b = b' \mid Evt]\Pr[Evt] + \Pr[b = b' \mid \neg Evt]\Pr[\neg Evt]\\
&\qquad \le \Pr[Evt] + (1/2)\Pr[\neg Evt]\\
&\qquad = \Pr[Evt] + (1/2)(1-\Pr[Evt])\\
&\qquad = (1/2)\Pr[Evt] + 1/2.
\end{aligned} \tag{7}
$$

According to (7) and the sense of ϵ, the following equation can be obtained.

$$
\begin{aligned}
&\epsilon = \Pr[b = b'] - 1/2\\
&\le \Pr[Evt]\\
&\le (\Pr[AskH_5^*] + \Pr[AskH_8^*] + \Pr[DecErr])\\
&/ \Pr[\neg Abort].
\end{aligned} \tag{8}
$$

According to (8), we have:

$\Pr[AskH_5^*] \ge \epsilon \Pr[\neg Abort] - \Pr[DecErr]$

$-\ \Pr[AskH_8^*]$.

Since $\Pr[\neg Abort] = \tau^{q_{TUK}+q_{FSK}+q_{Auth}}(1-\tau)$, we can obtain $\Pr[\neg Abort] \ge 1/e(q_{TUK} + q_{FSK} + q_{Auth} + 1)$ when $\tau = 1 - 1/(q_{TUK} + q_{FSK} + q_{Auth} + 1)$. We then have:

$$
\Pr[AskH_5^*] \ge \epsilon/e(q_{TUK}+q_{ful}+q_{Auth}+1) - q_D/q - q_{h_8}/q. \tag{9}
$$

If $AskH_5^*$ occurs, $\mathcal{A}_3$ will distinguish the real one during the simulation and the challenge ciphertext C^* is invalid. Then $H_5(e(P^{ac}, Q)^{x^*\mu^*}\cdot e(P, Q)^{abcx^*\eta^*}, C_1^*, C_2^*)$ has been added in the L_{H5}. $\mathcal{B}$ can pick the right bit from the L_{H5} and wins the game. According to (9), the BDH problem can be solved by $\mathcal{B}$ with the following advantage

$\epsilon' \ge (1/q_{h_5})\Pr[AskH_5^*]$

$\ge (1/q_{h_5})\,[\epsilon/e(q_{TUK}+q_{FSK}+q_{Auth}+1) - q_D/q - q_{h_8}/q]$.

Theorem 4. *Assume that there exists PPT Type-4 adversary $\mathcal{A}_4$ against OW-CCA security for the proposed scheme in the random oracle model. Then, $\mathcal{A}_4$ has the advantage ϵ to break the scheme. By the ϵ from $\mathcal{A}_4$, we construct that a challenger $\mathcal{B}$ solves the BDH assumption with the advantage ϵ' and $\epsilon' \geq (1/q_{h_5})\ [(\epsilon - 1/2^{\lambda})/(e(q_{PSK} + q_{FSK} + 1)] - q_D/q$. Suppose that the eight hash functions $H_i\,(1 \leq i \leq 8)$ are random oracles and then $\mathcal{A}_4$ can issue random oracle queries $q_{H_i}(1 \leq i \leq 8)$. Moreover, $\mathcal{A}_4$ also can issue Partial secret key queries q_{PSK}, Time update key queries q_{TUK}, Full secret key queries q_{FSK}, Public key queries q_{PK}, Replace public key queries q_{RPK}, Decryption queries q_D and Authorization queries q_{Auth} to challenger $\mathcal{B}$.*

Proof. Assume that $(\mathcal{G}, P, P^a, P^c, Q, Q^a, Q^b)$ is an instance of the BDH problem where $\mathcal{G} = (q, G_1, G_2, G_T, e)$, and $\mathcal{B}$ would like to calculate the BDH solution $e(P, Q)^{abc}$. $\mathcal{B}$ acts as a challenger and interacts with the Type-4 adversary $\mathcal{A}_4$ to calculate $e(P, Q)^{abc}$ in the following $G_{OW\text{-}CCA}$ game:

1 ***Setup***: $\mathcal{B}$ sets $P_{pub} = P^a$ and selects eight collision-resistant hash functions $H_i\,(1 \leq i \leq 8)$ as random oracles. Then $\mathcal{B}$ outputs the public parameters PP to $\mathcal{A}_4$, where $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$. To keep the consistency between the random oracle queries and the corresponding responses, $\mathcal{B}$ needs to maintain the lists $L_{H1}, L_{H2}, ..., L_{H8}, L_{Key}$, which are similar to the proof of Theorem 1.

2 ***Phase 1***: $\mathcal{A}_4$ launches a series of queries to $\mathcal{B}$, and then $\mathcal{B}$ returns the corresponding answers as follows.

- H_1 *query*(ID_i): the query is identical to the proof of Theorem 1.
- H_2 *query*(ID_i): after receiving this query on ID_i, $\mathcal{B}$ does the following.
  - If ID_i exists in L_{H2}, $\mathcal{B}$ searches the tuple $[ID_i, v_i, cn]$ by ID_i. Upon obtaining v_i and cn from L_{H2}, compute Q^{v_i} as the answer to $\mathcal{A}_4$.
  - Otherwise $\mathcal{B}$ picks a time period t_i at random and makes *Public key query* on (ID_i, t_i) to generate v_i, cn and store them in L_{H2}. Then repeat the above step to return Q^{v_i}.
- H_3-H_8 *queries*: the queries are identical to the proof of Theorem 1.
- *Public key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Partial secret key query*(ID_i) : the query is identical to the proof of Theorem 1.
- *Time update key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Full secret key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Replace public key query*(ID_i, PK_i'): the query is identical to the proof of Theorem 1.
- *Decryption query*(ID_i, t_i, C): after receiving this query on (ID_i, t_i, C) where $C = (C_1, C_2, C_3, C_4)$, $\mathcal{B}$ performs the following tasks.
  - If (ID_i, t_i) exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by (ID_i, t_i), and execute the task as the following:
    - If $cn = 0$, and the public key has not been replaced by $\mathcal{A}_4$, $\mathcal{B}$ uses FSK_i from the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, 0]$ in L_{Key} to execute the Decryption(PP, FSK_i, C) algorithm and returns the output to $\mathcal{A}_4$.
    - If $cn = 1$, $\mathcal{B}$ searches the tuple $[W, C_1, C_2, \omega]$ in L_{H5} by C_1, C_2, and calculates $M' \,||\, k' = C_3 \oplus \omega$. Next, (M', k') is used to search the tuple $[M, k, \gamma]$ in L_{H7}. After obtaining γ, compute the P^{γ}. Then retrieve the tuple $[ID_i, v_i, cn]$ in L_{H2} by ID_i and research the tuple $[ID_i, t_i, \zeta_i, cn]$ in L_{H4} by (ID_i, t_i) to compute $FSK_2 \cdot TUK_2 = Q^{a(v_i+\zeta_i)x_i}$. If find the S in the tuple $[e(C_2, Q^{a(v_i+\zeta_i)x_i}), S]$ in L_{H8} such that $C_4 = R \cdot S$ holds, $\mathcal{B}$ will check whether $C_1 = P^{\gamma}$ holds. When both $C_1 = P^{\gamma}$ and $C_4 = R \cdot S$ holds, return M' to $\mathcal{A}_4$. $\mathcal{B}$ returns $\perp$ to $\mathcal{A}_4$ if $\mathcal{B}$ cannot search the tuple in L_{H5}.
  - Otherwise, $\mathcal{B}$ sends a *Public key query* to (ID_i, t_i) to generate FSK_i, cn. Then repeat the above step to return M.
- *Authorization query*(ID_i, t_i): by completing the following steps, $\mathcal{B}$ can answer this query.
  - If (ID_i, t_i) exists in L_{Key}, $\mathcal{B}$ searches the tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, cn]$ by (ID_i, t_i), and returns $FSK_{i,2}$ to $\mathcal{A}_4$, where $FSK_i = (FSK_{i,1}, FSK_{i,2})$.
  - Otherwise $\mathcal{B}$ retrieves the tuple $[ID_i, v_i, cn]$ in L_{H2} by ID_i to compute $FSK_{i,2} = Q^{av_ix_i}$ and returns $FSK_{i,2}$ to $\mathcal{A}_4$.

3 ***Challenge***: $\mathcal{A}_4$ sends an identity ID^*, a time period t^* to $\mathcal{B}$ for challenge. $\mathcal{B}$ selects $M^* \in \{0, 1\}^{\lambda}$ at random and uses (ID^*, t^*) as an input to produce *Public key*

query and get the tuple $[ID^*, t^*, x^*, PSK^*, TUK^*, FSK^*, PK^*, cn]$ from L_{Key}.

- If $cn = 0$, $\mathcal{B}$ aborts the game.
- If $cn = 1$, $\mathcal{B}$ performs the following tasks:
  - Select $k \in \{0,1\}^l$, $C_3^* \in \{0,1\}^{\lambda+l}$ at random.
  - Set $C_2^* = P^c$.
  - Obtain γ by H_7 $query(M^*, k)$ and set $C_1^* = P^\gamma$.
  - Obtain R and S by H_6 $query(M^*)^\gamma$ and H_8 $query(e(C_2^*, Q^{a(v_i^*+\zeta_i^*)x_i^*}))$, respectively.
  - Set $C_4^* = R \cdot S$.
  - Return $C^* = (C_1^*, C_2^*, C_3^*, C_4^*)$ to $\mathcal{A}_4$.

Based on the above construction, $H_5(e(P, Q)^{abcx^*\mu^*} \cdot e(P^{ac}, Q)^{x^*\eta^*}, C_1^*, C_2^*) = (M^* \| k) \oplus C_3^*$, where $Q^{b\mu^*} = H_1(ID^*)$.

4. ***Phase 2***: $\mathcal{A}_4$ launches a series of queries to $\mathcal{B}$ as in ***Phase 1***.
5. ***Guess***: eventually, $\mathcal{A}_4$ outputs $M' \in \{0, 1\}^\lambda$ as the guess bit. If $M' = M$, $\mathcal{A}_4$ wins the game; otherwise loses the game. $\mathcal{B}$ chooses a random tuple $[\sigma^*, C_1^*, C_2^*, \theta]$ from L_{H5} and outputs $(\sigma^*/e(P^{ac}, Q)^{x^*\eta^*})^{(x^*\mu^*)^{-1}} = e(P, Q)^{abc})$ as the solution to the BDH instance.

Analysis. We need to evaluate the simulation of the random oracles first. It is clear that H_1, H_2, H_3, H_4, H_6, H_7 and H_8 simulations are perfect due to their construction. $AskH_5^*$ is defined as the event that $H_5(e(P, Q)^{abcx^*\mu^*} \cdot e(P^{ac}, Q)^{x^*\eta^*}, C_1^*, C_2^*)$ has been issued by $\mathcal{A}_4$. We say that the simulation of H_5 is perfect if $AskH_5^*$ does not happen. Now we assess the simulation of the decryption oracle. $DecErr$ indicates an event in the valid ciphertext, and $\mathcal{B}$ cannot decrypt it exactly during the emulation and we get $\Pr[DecErr] \le q_D/q$.

Next, define $Abort$ as the event that the emulation is aborted by $\mathcal{B}$, and define $Evt = (AskH_5^* \vee DecErr) | \neg Abort$. $\mathcal{B}$ guess M with the advantage $\le 1/2^\lambda$ if Evt does not occur due to the randomness of the outputs of H_5. So $\Pr[M = M' | \neg Evt] \le 1/2^\lambda$, we obtain

$$
\begin{aligned}
\Pr[M = M'] &= \Pr[M = M' | Evt]\Pr[Evt] \\
&\quad + \Pr[M = M' | \neg Evt]\Pr[\neg Evt] \\
&\le \Pr[Evt] + (1/2^\lambda) \Pr[\neg Evt] \\
&= \Pr[Evt] + (1/2^\lambda)(1 - \Pr[Evt]) \\
&= (1 - 1/2^\lambda)\Pr[Evt] + (1/2^\lambda).
\end{aligned} \tag{10}
$$

According to (10) and the sense of ϵ, the following equation can be obtained.

$$
\begin{aligned}
\epsilon &= \Pr[M = M'] \\
&\le (1 - 1/2^\lambda)\Pr[Evt] + (1/2^\lambda) \\
&\le (1 - 1/2^\lambda)(\Pr[AskH_5^*] + \Pr[DecErr]) \\
&\quad / \Pr[\neg Abort] + (1/2^\lambda).
\end{aligned} \tag{11}
$$

According to (11), we have:

$\Pr[AskH_5^*] \ge [(\epsilon - 1/2^\lambda)/(1 - 1/2^\lambda)]\Pr[\neg Abort] - \Pr[DecErr]$

Since $\Pr[\neg Abort] = \tau^{q_{PSK} + q_{FSK}}(1 - \tau)$, we can obtain $\Pr[\neg Abort] \ge 1/e(q_{PSK} + q_{FSK} + 1)$ when $\tau = 1 - 1/(q_{PSK} + q_{FSK} + 1)$. We then have:

$$\Pr[AskH_5^*] \ge [(\epsilon - 1/2^\lambda)/e(q_{PSK} + q_{FSK} + 1)] - q_D/q. \tag{12}$$

If $AskH_5^*$ occurs, $\mathcal{A}_4$ will distinguish the real one during the simulation and the challenge ciphertext C^* is invalid. Then $H_5(e(P, Q)^{abcx^*\mu^*} \cdot e(P^{ac}, Q)^{x^*\eta^*}, C_1^*, C_2^*)$ has been added in the L_{H5}. $\mathcal{B}$ can pick the right bit from the L_{H5} and wins the game. According to (12), the BDH problem can be solved by $\mathcal{B}$ with the following advantage

$\epsilon' \ge (1/q_{h_5})\Pr[AskH_5^*]$

$\ge (1/q_{h_5})[(\epsilon - 1/2^\lambda)/e(q_{PSK} + q_{FSK} + 1)] - q_D/q.$

Theorem 5. *Assume that there exists PPT Type-5 adversary $\mathcal{A}_5$ against OW-CCA security for the proposed scheme in the random oracle model. Then, $\mathcal{A}_5$ has the advantage ϵ to break the scheme. By the ϵ from $\mathcal{A}_5$, we construct that an algorithm challenger $\mathcal{B}$ solves the BDH assumption with the advantage ϵ' and $\epsilon' \ge (1/q_{h_5})[(\epsilon - 1/2^\lambda)/(e(q_{FSK} + 1)] - q_D/q$. Suppose that the eight hash functions H_i $(1 \le i \le 8)$ are random oracles and then $\mathcal{A}_5$ can issue random oracle queries $q_{H_i}(1 \le i \le 8)$. Moreover, $\mathcal{A}_5$ also can issue Full secret key queries q_{FSK}, Public key queries q_{PK}, Decryption queries q_D and Authorization queries q_{Auth} to challenger $\mathcal{B}$. Note that $\mathcal{A}_5$ is a malicious KGC so $\mathcal{A}_5$ cannot issue Partial secret key queries q_{PSK}, Time update key queries q_{TUK}, Replace public key queries q_{RPK}.*

Proof. Assume that $(\mathcal{G}, P, P^a, P^c, Q, Q^a, Q^b)$ is an instance of the BDH problem where $\mathcal{G} = (q, G_1, G_2, G_T, e)$, and $\mathcal{B}$ would like to calculate the BDH solution $e(P, Q)^{abc}$. $\mathcal{B}$ acts as a challenger and interacts with the Type-5 adversary $\mathcal{A}_5$ to calculate $e(P, Q)^{abc}$ in the following $G_{OW\text{-}CCA}$ game:

1. ***Setup***: $\mathcal{B}$ picks the master secret key $s \in Z_q^*$ at random and sets $P_{pub} = P^s$. Then select eight collision-resistant hash functions H_i $(1 \le i \le 8)$ as ran-

dom oracles. Then $\mathcal{B}$ outputs the master secret key s and the public parameters PP to $\mathcal{A}_5$, where $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$. To keep the consistency between the random oracle queries and the corresponding responses, $\mathcal{B}$ needs to maintain the lists $L_{H1}, L_{H2}, ..., L_{H8}, L_{Key}$, which are similar to the proof of Theorem 1.

2 ***Phase 1***: $\mathcal{A}_4$ launches a series of queries to $\mathcal{B}$, and then $\mathcal{B}$ returns the corresponding answers as follows.

- H_1 *query*(ID_i): the query is identical to the proof of Theorem 1.
- H_2 *query*(ID_i): the query is identical to the proof of Theorem 4.
- H_3-H_8 *queries*: the queries are identical to the proof of Theorem 1.
- *Public key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Public key query*(ID_i, t_i): after receiving this query on (ID_i, t_i), $\mathcal{B}$ randomly selects $\mu_i, v_i, \eta_i, \zeta_i \in Z_q^*$, $cn \in \{0, 1\}$ with $\Pr[cn = 0] = \tau$, and then adds four tuples $[ID_i, \mu_i, cn]$, $[ID_i, v_i, cn]$, $[ID_i, t_i, \eta_i, cn]$, $[ID_i, t_i, \zeta_i, cn]$ into $L_{H1}, L_{H2}, L_{H3}, L_{H4}$ respectively.
  - If $cn = 0$, $\mathcal{B}$ executes the SetSecretValue(PP) algorithm to get the secret value x_i, then computes $PSK_i = (PSK_{i,1}, PSK_{i,2}) = (Q^{s\mu_i}, Q^{sv_i})$, $TUK_i = (TUK_{i,1}, TUK_{i,2}) = (Q^{s\eta_i}, Q^{s\zeta_i})$, $FSK_i = (FSK_{i,1}, FSK_{i,2}) = ((PSK_{i,1} \cdot TUK_{i,1})^{X_i}, (PSK_{i,2} \cdot TUK_{i,2})^{X_i})$ and $PK_i = (PK_{i,1}, PK_{i,2}) = (P_{pub}^{X_i}, Q^{X_i})$, adds an tuple $[ID_i, t_i, x_i, PSK_i, TUK_i, FSK_i, PK_i, 0]$ into L_{Key}, and returns PK_i to $\mathcal{A}_5$.
  - Otherwise, $\mathcal{B}$ selects $x_i' \in Z_q^*$ at random, then computes $PSK_i = (PSK_{i,1}, PSK_{i,2}) = (Q^{bs\mu_i}, Q^{sv_i})$, $TUK_i = (TUK_{i,1}, TUK_{i,2}) = (Q^{s\eta_i}, Q^{s\zeta_i})$ and $PK_i = (PK_{i,1}, PK_{i,2}) = (P_{pub}^{ax_i'}, Q^{ax_i'})$ adds an tuple $[ID_i, t_i, x_i', PSK_i, TUK_i, -, PK_i, 1]$ into L_{Key}, and returns PK_i to $\mathcal{A}_5$. Here, the secret value x_i is seen as ax_i' implicitly.
- *Full secret key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Decryption query*(ID_i, t_i, C): the query is identical to the proof of Theorem 4.
- *Authorization query*(ID_i, t_i): the query is identical to the proof of Theorem 4.

3 ***Challenge***: $\mathcal{A}_5$ sends an identity ID^*, a time period t^* to $\mathcal{B}$ for challenge. $\mathcal{B}$ selects $M^* \in \{0, 1\}^{\lambda}$ at random and uses (ID^*, t^*) as an input to produce *Public key query* and get the tuple $[ID^*, t^*, x^*, PSK^*, TUK^*, FSK^*, PK^*, cn]$ from L_{Key}.

- If $cn = 0$, $\mathcal{B}$ aborts the game.
- If $cn = 1$, $\mathcal{B}$ performs the following tasks:
  - Select $k \in \{0, 1\}^l$, $C_3^* \in \{0, 1\}^{\lambda+l}$ at random.
  - Set $C_2^* = P^c$.
  - Obtain γ by H_7 *query*(M^*, k) and set $C_1^* = P^{\gamma}$.
  - Obtain R and S by H_6 *query*$(M^*)^{\gamma}$ and H_8 *query*$(e(C_2^*, Q^{a(v_i^* + \zeta_i^*)x_i^*}))$ respectively.
  - Set $C_4^* = R \cdot S$.
  - Return $C^* = (C_1^*, C_2^*, C_3^*, C_4^*)$ to $\mathcal{A}_5$.

Based on the above construction, $H_5(e(P, Q)^{abcx^*\mu^*s} \cdot e(P^{ac}, Q)^{x^*\eta^*s}, C_1^*, C_2^*) = (M^* \| k) \oplus C_3^*$, where $Q^{b\mu^*} = H_1(ID^*)$.

4 ***Phase 2***: $\mathcal{A}_5$ launches a series of queries to $\mathcal{B}$ as in ***Phase 1***.

5 ***Guess***: eventually, $\mathcal{A}_5$ outputs $M' \in \{0, 1\}^{\lambda}$ as the guess bit. If $M' = M$, $\mathcal{A}_5$ wins the game; otherwise loses the game. $\mathcal{B}$ chooses a random tuple $[\sigma^*, C_1^*, C_2^*, \theta]$ from L_{H5} and outputs $(\sigma^*/e(P^{ac}, Q)^{x^*\eta^*s})^{(x^*\mu^*s)^{-1}} = e(P, Q)^{abc})$ as the solution to the BDH instance.

Analysis. We need to evaluate the simulation of the random oracles first. It is clear that $H_1, H_2, H_3, H_4, H_6, H_7$ and H_8 simulations are perfect due to their construction. $AskH_5^*$ is defined as the event that $H_5(e(P, Q)^{abcx^*\mu^*s} \cdot e(P^{ac}, Q)^{x^*\eta^*s}, C_1^*, C_2^*)$ has been issued by $\mathcal{A}_5$. We say that the simulation of H_5 is perfect if $AskH_5^*$ does not happen. Now we assess the simulation of the decryption oracle. $DecErr$ indicates an event in the valid ciphertext, and $\mathcal{B}$ cannot decrypt it exactly during the emulation and we get $\Pr[DecErr] \le q_D/q$.

Next, define $Abort$ as the event that the emulation is aborted by $\mathcal{B}$, and define $Evt = (AskH_5^* \vee DecErr) | \neg Abort$. $\mathcal{B}$ guess M with the advantage $\le 1/2^{\lambda}$ if Evt does not occur due to the randomness of the outputs of H_5. So $\Pr[M = M' | \neg Evt] \le 1/2^{\lambda}$, we obtain

$$
\begin{aligned}
&\Pr[M = M'] = \Pr[M = M' | Evt]\Pr[Evt] \\
&+ \Pr[M = M' | \neg Evt]\Pr[\neg Evt] \\
&\le \Pr[Evt] + (1/2^{\lambda}) \Pr[\neg Evt] \\
&= \Pr[Evt] + (1/2^{\lambda}) (1 - \Pr[Evt]) \\
&= (1 - 1/2^{\lambda})\Pr[Evt] + (1/2^{\lambda}).
\end{aligned}
\tag{13}
$$

According to (13) and the sense of ϵ, the following equation can be obtained.

$$\begin{aligned}\epsilon &= \Pr[M = M'] \\ &\leq (1-1/2^{\lambda})\Pr[Evt] + (1/2^{\lambda}) \\ &\leq (1-1/2^{\lambda})\,(\Pr[AskH_5^{*}] + \Pr[DecErr]) \\ &\quad / \Pr[\neg Abort] + (1/2^{\lambda}).\end{aligned} \tag{14}$$

According to (14), we have:

$\Pr[AskH_5^{*}] \geq [(\epsilon - 1/2^{\lambda})/(1-1/2^{\lambda})]\Pr[\neg Abort]$
$- \Pr[DecErr]$

Since $\Pr[\neg Abort] = \tau^{q_{FSK}}(1-\tau)$, we can obtain $\Pr[\neg Abort] \geq 1/e(q_{FSK}+1)$ when $\tau = 1 - 1/(q_{FSK}+1)$. We then have:

$$\Pr[AskH_5^{*}] \geq [(\epsilon - 1/2^{\lambda})/e(q_{FSK}+1)] - q_D/q. \tag{14}$$

If $AskH_5^{*}$ occurs, $\mathcal{A}_5$ will distinguish the real one during the simulation and the challenge ciphertext C^* is invalid. Then $H_5(e(P, Q)^{abcx^*\mu^*s} \cdot e(P^{ac}, Q)^{x^*\eta^*s}, C_1^{*}, C_2^{*})$ has been added in the L_{H5}. $\mathcal{B}$ can pick the right bit from the L_{H5} and wins the game. According to (15), the BDH problem can be solved by $\mathcal{B}$ with the following advantage

$$\begin{aligned}\epsilon' &\geq (1/q_{h_5})\Pr[AskH_5^{*}] \\ &\geq (1/q_{h_5})\,[(\epsilon - 1/2^{\lambda})/e(q_{FSK}+1)] - q_D/q.\end{aligned}$$

Theorem 6. *Assume that there exists PPT Type-6 adversary $\mathcal{A}_6$ against OW-CCA security for the proposed scheme in the random oracle model. Then, $\mathcal{A}_6$ has the advantage ϵ to break the scheme. By the ϵ from $\mathcal{A}_6$, we construct that a challenger $\mathcal{B}$ solves the BDH assumption with the advantage ϵ' and $\epsilon' \geq (1/q_{h_5})\,[(\epsilon - 1/2^{\lambda})/(e(q_{TUK} + q_{FSK} + 1)] - q_D/q$. Suppose that the eight hash functions $H_i\,(1 \leq i \leq 8)$ are random oracles and then $\mathcal{A}_6$ can issue random oracle queries $q_{H_i}(1 \leq i \leq 8)$. Moreover, $\mathcal{A}_6$ also can issue Partial secret key queries q_{PSK}, Time update key queries q_{TUK}, Full secret key queries q_{FSK}, Public key queries q_{PK}, Replace public key queries q_{RPK}, Decryption queries q_D and Authorization queries q_{Auth} to challenger $\mathcal{B}$.*

Proof. Assume that $(\mathcal{G}, P, P^a, P^c, Q, Q^a, Q^b)$ is an instance of the BDH problem where $\mathcal{G} = (q, G_1, G_2, G_T, e)$, and $\mathcal{B}$ would like to calculate the BDH solution $e(P, Q)^{abc}$. $\mathcal{B}$ acts as a challenger and interacts with the Type-6 adversary $\mathcal{A}_6$ to calculate $e(P, Q)^{abc}$ in the following $G_{OW\text{-}CCA}$ game:

1. ***Setup***: $\mathcal{B}$ sets $P_{pub} = P^a$ and selects eight collision-resistant hash functions $H_i\,(1 \leq i \leq 8)$ as random oracles. Then $\mathcal{B}$ outputs the public parameters PP to $\mathcal{A}_6$, where $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$. To keep the consistency between the random oracle queries and the corresponding responses, $\mathcal{B}$ needs to maintain the lists $L_{H1}, L_{H2}, ..., L_{H8}, L_{Key}$, which are similar to the proof of Theorem 1.
2. ***Phase 1***: $\mathcal{A}_6$ launches a series of queries to $\mathcal{B}$, and then $\mathcal{B}$ returns the corresponding answers as follows.

- H_1-H_3 *queries*(ID_i): the queries are identical to the proof of Theorem 3.
- H_4-H_8 *queries*: the queries are identical to the proof of Theorem 1.
- *Public key query*(ID_i, t_i): the query is identical to the proof of Theorem 3.
- *Partial secret key query*(ID_i) : the query is identical to the proof of Theorem 3.
- *Time update key query*(ID_i, t_i): the query is identical to the proof of Theorem 3.
- *Full secret key query*(ID_i, t_i): the query is identical to the proof of Theorem 1.
- *Replace public key query*(ID_i, PK_i'): the query is identical to the proof of Theorem 1.
- *Decryption query*(ID_i, t_i, C): the query is identical to the proof of Theorem 4.
- *Authorization query*(ID_i, t_i): the query is identical to the proof of Theorem 4.

3. ***Challenge***: $\mathcal{A}_6$ sends an identity ID^*, a time period t^* to $\mathcal{B}$ for challenge. $\mathcal{B}$ selects $M^* \in \{0, 1\}^{\lambda}$ at random and uses (ID^*, t^*) as an input to produce *Public key query* and get the tuple $[ID^*, t^*, x^*, PSK^*, TUK^*, FSK^*, PK^*, cn]$ from L_{Key}.

- If $cn = 0$, $\mathcal{B}$ aborts the game.
- If $cn = 1$, $\mathcal{B}$ performs the following tasks:
  - Select $k \in \{0, 1\}^l$, $C_3^{*} \in \{0, 1\}^{\lambda+l}$ at random.
  - Set $C_2^{*} = P^c$.
  - Obtain γ by H_7 *query*(M^*, k) and set $C_1^{*} = P^{\gamma}$.
  - Obtain R and S by H_6 *query*$(M^*)^{\gamma}$ and H_8 *query*$(e(C_2^{*}, Q^{a(v_i^* + \zeta_i^*)x_i^*}))$ respectively.
  - Set $C_4^{*} = R \cdot S$.
  - Return $C^* = (C_1^{*}, C_2^{*}, C_3^{*}, C_4^{*})$ to $\mathcal{A}_6$.

Based on the above construction, $H_5(e(P, Q)^{x^*\mu^*} \cdot e(P^{ac}, Q)^{abcx^*\eta^*}, C_1^{*}, C_2^{*}) = (M^* \,||\, k) \oplus C_3^{*}$, where $Q^{b\eta^*} = H_3(ID^*)$.

4. ***Phase 2***: $\mathcal{A}_6$ launches a series of queries to $\mathcal{B}$ as in ***Phase 1***.

5 ***Guess***: eventually, $\mathcal{A}_6$ outputs $M' \in \{0,1\}^\lambda$ as the guess bit. If $M' = M$, $\mathcal{A}_6$ wins the game; otherwise loses the game. $\mathcal{B}$ chooses a random tuple $[\sigma^*, C_1^*, C_2^*, \theta]$ from L_{H5} and outputs $(\sigma^*/e(P^{ac}, Q)^{x^*\mu^*})^{(x^*\eta^*)^{-1}} = e(P, Q)^{abc}$) as the solution to the BDH instance.

Analysis. We need to evaluate the simulation of the random oracles first. It is clear that H_1, H_2, H_3, H_4, H_6, H_7 and H_8 simulations are perfect due to their construction. $AskH_5^*$ is defined as the event that $H_5(e(P, Q)^{x^*\mu^*} \cdot e(P^{ac}, Q)^{abcx^*\eta^*}, C_1^*, C_2^*)$ has been issued by $\mathcal{A}_6$. We say that the simulation of H_5 is perfect if $AskH_5^*$ does not happen. Now we assess the simulation of the decryption oracle. $DecErr$ indicates an event in the valid ciphertext, and $\mathcal{B}$ cannot decrypt it exactly during the emulation and we get $\Pr[DecErr] \leq q_D/q$.

Next, define $Abort$ as the event that the emulation is aborted by $\mathcal{B}$, and define $Evt = (AskH_5^* \vee DecErr)|\neg Abort$. $\mathcal{B}$ guess M with the advantage $\leq 1/2^\lambda$ if Evt does not occur due to the randomness of the outputs of H_5. So $\Pr[M = M' | \neg Evt] \leq 1/2^\lambda$, we obtain

$$\begin{aligned} \Pr[M = M'] &= \Pr[M = M' | Evt]\Pr[Evt] \\ &+ \Pr[M = M' | \neg Evt]\Pr[\neg Evt] \\ &\leq \Pr[Evt] + (1/2^\lambda)\Pr[\neg Evt] \\ &= \Pr[Evt] + (1/2^\lambda)(1 - \Pr[Evt]) \\ &= (1 - 1/2^\lambda)\Pr[Evt] + (1/2^\lambda). \end{aligned} \tag{16}$$

According to (16) and the sense of ϵ, the following equations can be obtained.

$$\begin{aligned} \epsilon &= \Pr[M = M'] \\ &\leq (1 - 1/2^\lambda)\Pr[Evt] + (1/2^\lambda) \\ &\leq (1 - 1/2^\lambda)(\Pr[AskH_5^*] \\ &+ \Pr[DecErr]) / \Pr[\neg Abort] + (1/2^\lambda). \end{aligned} \tag{17}$$

According to (17), we have:

$\Pr[AskH_5^*] \geq [(\epsilon - 1/2^\lambda)/(1 - 1/2^\lambda)]\Pr[\neg Abort]$
$- \Pr[DecErr]$

Since $\Pr[\neg Abort] = \tau^{q_{TUK} + q_{FSK}}(1 - \tau)$, we can obtain $\Pr[\neg Abort] \geq 1/e(q_{TUK} + q_{FSK} + 1)$ when $\tau = 1 - 1/(q_{TUK} + q_{FSK} + 1)$. We then have:

$$\Pr[AskH_5^*] \geq [(\epsilon - 1/2^\lambda)/e(q_{TUK} + q_{FSK} + 1)] - q_D/q. \tag{18}$$

If $AskH_5^*$ occurs, $\mathcal{A}_6$ will distinguish the real one during the simulation and the challenge ciphertext C^* is invalid. Then $H_5(e(P, Q)^{x^*\mu^*} \cdot e(P^{ac}, Q)^{abcx^*\eta^*}, C_1^*, C_2^*)$ has been added in the L_{H5}. $\mathcal{B}$ can pick the right bit from the L_{H5} and wins the game. According to (18), the BDH problem can be solved by $\mathcal{B}$ with the following advantage

$$\begin{aligned} \epsilon' &\geq (1/q_{h_5})\Pr[AskH_5^*] \\ &\geq (1/q_{h_5})\,[(\epsilon - 1/2^\lambda)/e(q_{TUK} + q_{FSK} + 1)] - q_D/q. \end{aligned}$$

Theorem 7. *Assume that there exists PPT adversary $\mathcal{A}$ (all types adversary) against the security of brute force attacks for the proposed scheme. Then, $\mathcal{A}$ has the negligible advantage to break the scheme.*

Proof. As mentioned in Section 4 (The RCL-PKEET scheme), the master secret key is $msk = s \in Z_q^*$ and the public parameters is $PP = (\mathcal{G}, P, Q, P_{pub}, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8)$. We can ensure that the PPT adversary $\mathcal{A}$ cannot break the system to obtain the master secret key from the public parameters, since only P_{pub} and msk are related and $P_{pub} = P^s$. Calculating $msk = s$ from P_{pub} and P is a problem of discrete logarithm that the PPT adversary $\mathcal{A}$ cannot solve in the polynomial time. In fact, the user's partial secret key PSK and time update key TUK are also designed based on the discrete logarithm problem, where $PSK = (PSK_1, PSK_2) = (H_1(ID)^{msk}, H_2(ID)^{msk}) = (H_1(ID)^s, H_2(ID)^s)$ and $TUK = (TUK_1, TUK_2) = (H_3(ID, t)^{msk}, H_4(ID, t)^{msk}) = (H_3(ID, t)^s, H_4(ID, t)^s)$. Therefore, we believe that the proposed scheme can withstand brute force attacks.

6. Comparsions

In this section, the computation cost, the communication cost and functionalities of our proposed RCL-PKEET scheme, the existing IBEET scheme [26], CL-PKEET schemes [13, 29, 45] and RCL-PKE scheme [37] are compared. For the computation cost in the procedures of encryption, decryption and equality test and communication cost in piublic key, ciphertext and trapdoor, we first define several notations as below.

- T_{pair}: the cost of computing a bilinear pairing.
- T_{exp}: the cost of computing an exponentiation.
- T_{hash}: the cost of computing a hash function.
- $|G_1|$: the size of a point in G_1.
- $|G_2|$: the size of a point in G_2.
- $|Z_q|$: the bit length in Z_q.
- $|PK|$: the bit length of public key.
- $|CT|$: the bit length of ciphertext.

$|TD|$: the bit length of trapdoor.

Table 1 lists the cost of T_{pair}, T_{exp} and T_{hash} in the simulation experiences [21] where the CPU is Intel Core i7-8550U with 1.80 Ghz processor. In addition, F_q, G_1 and G_2 are selective parameters, where F_q is a finite field composed of the sets of integers $\{0, 1, ..., q - 1\}$, $q \in \{0, 1\}^{256}$ is a prime number and G_1, G_2 are groups of order 224 bits prime number over F_q.

Table 1

The cost of T_{pair}, T_{exp} and T_{hash}

	T_{pair}	T_{exp}	T_{hash}
The executing time	7.8351 ms	0.4746 ms	0.0126 ms

Table 2 compares our RCL-PKEET scheme with other existing schemes in terms of encryption, decryption, equality test and three functionalities. Although our scheme may be slower than the existing IBEET scheme in the procedures of encryption and decryption, our scheme possesses the ability to solve the key escrow problem and provide the efficient revocation mechanism. Similarly, the overall efficiency of the existing RCL-PKE scheme is better than that of our RCL-PKEET scheme. However, our RCL-PKEET scheme has the functionality of the equality test but the existing RCL-PKE scheme does not. Compared with the existing CL-PKEET schemes, our RCL-PKEET scheme provides the efficient revocation mechanism while retaining the performance in the procedures of encryption, decryption and equality test. Obviously, our RCL-PKEET scheme solves key escrow problem and possesses the functionalities of equality test and revocation mechanism.

Table 3 compares our RCL-PKEET scheme with other existing schemes in terms of the bit length of public key, ciphertext and trapdoor. Obviously, the communication cost of our scheme is close to other existing schemes.

Table 3

Comparison of communication cost

	|PK|	|CT|	|TD|
IBEET [26]	$\|G_1\|$	$4\|G_1\|+\|Z_q\|$	$\|G_1\|$
CL-PKEET [29]	$\|G_1\|+\|G_2\|$	$2\|G_1\|+\|G_2\|+\|Z_q\|$	$\|G_2\|$
CL-PKEET [45]	$2\|G_1\|$	$3\|G_1\|+2\|Z_q\|$	$\|G_1\|$
CL-PKEET [13]	$3\|G_1\|$	$3\|G_1\|+\|Z_q\|$	$\|G_1\|$
RCL-PKE [37]	$\|G_1\|$	$\|G_1\|+2\|Z_q\|$	-
Our RCL-PKEET	$\|G_1\|+\|G_2\|$	$2\|G_1\|+\|G_2\|+\|Z_q\|$	$\|G_2\|$

Table 2

Comparison between our proposed scheme and other existing schemes

Schemes	Encryption	Decryption	Equality test	With equality test	Without key escrow problem	With revocation mechanism
IBEET [26]	$2T_{pair}+6T_{exp}+3T_{hash}$ (18.5556ms)	$2T_{pair}+2T_{exp}+2T_{hash}$ (16.6446ms)	$4T_{pair}+2T_{exp}+2T_{hash}$ (32.3148ms)	Yes	No	No
CL-PKEET [29]	$4T_{pair}+5T_{exp}+6T_{hash}$ (33.7890ms)	$2T_{pair}+2T_{exp}+4T_{hash}$ (16.6698ms)	$4T_{pair}+2T_{hash}$ (31.3656ms)	Yes	Yes	No
CL-PKEET [45]	$2T_{pair}+5T_{exp}+8T_{hash}$ (18.144ms)	$2T_{pair}+2T_{exp}+4T_{hash}$ (16.6698ms)	$4T_{pair}+2T_{hash}$ (31.3656ms)	Yes	Yes	No
CL-PKEET [13]	$4T_{pair}+5T_{exp}+6T_{hash}$ (33.789ms)	$2T_{pair}+2T_{exp}+2T_{hash}$ (16.6446ms)	$4T_{pair}+2T_{hash}$ (31.3656ms)	Yes	Yes	No
RCL-PKE [37]	$T_{pair}+3T_{exp}+5T_{hash}$ (9.3219ms)	$T_{pair}+2T_{exp}+3T_{hash}$ (8.8221ms)	-	No	Yes	Yes
Our RCL-PKEET	$4T_{pair}+5T_{exp}+8T_{hash}$ (33.8142ms)	$2T_{pair}+2T_{exp}+4T_{hash}$ (16.6698ms)	$4T_{pair}+2T_{hash}$ (31.3656ms)	**Yes**	**Yes**	**Yes**

7. Conclusions

In this article, we defined the framework of RCL-PKEET and formalized two security models which include six types of adversaries. Based on the framework, we presented the *first* RCL-PKEET scheme which possesses an efficient revocation mechanism. In addition, we demonstrated the proposed scheme is provably secure under the BDH assumption. Compared with the existing CL-PKEET scheme, the proposed scheme can efficiently revoke compromised users from the system while retaining the performance in the procedures of encryption, decryption and equality test.

Acknowledgement

This research was partially supported by Ministry of Science and Technology, Taiwan, under contract no. MOST 110-2222-E-019-001-MY2 and MOST 110-2221-E-019-041-MY3.

References

1. Al-Riyami, S. S., Paterson, K. G. Certificateless Public Key Cryptography. In: ASIACRYPT'03, 2003, LNCS 2894, 452-473. https://doi.org/10.1007/978-3-540-40061-5_29
2. Ali, I., Gervais, M., Ahene, E., Li, F. A Blockchain-based Certificateless Public Key Signature Scheme for Vehicle-to-Infrastructure Communication in VANETs. Journal of Systems Architecture, 2019, 99, 101636-101652. https://doi.org/10.1016/j.sysarc.2019.101636
3. Baek, J., Safavi-Naini, R., Susilo, W. Public Key Encryption with Keyword Search Revisited. In: ICCSA'08, 2008, LNCS 5072, 1249-1259. https://doi.org/10.1007/978-3-540-69839-5_96
4. Boneh, D., Crescenzo, G. D., Ostrovsky, R., Persiano, G. Public Key Encryption with Keyword Search. In: EUROCRYPT'04, 2004, LNCS 3027, 506-522. https://doi.org/10.1007/978-3-540-24676-3_30
5. Boneh, D., Franklin, M. Identity-based Encryption from the Weil Pairing. In: CRYPTO'01, 2001, LNCS 2139, 213-229. https://doi.org/10.1007/3-540-44647-8_13
6. Boyen, X., Mei, Q., Waters, B. Direct Chosen Ciphertext Security from Identity-based Techniques. In: Computer and Communications Security'05, 2005, ACM Press, 320-329. https://doi.org/10.1145/1102120.1102162
7. Chang, J., Wang, H., Wang, F., Zhang, A., Ji, Y. RKA Security for Identity-based Signature Scheme. lEEE Access, 2020, 8, 17833-17841. https://doi.org/10.1109/ACCESS.2020.2967904
8. Choon, J. C., Cheon, J. H. An Identity-based Signature from Gap Diffie-Hellman Groups. In: PKC'03, 2003, LNCS 2567, 18-30. https://doi.org/10.1007/3-540-36288-6_2
9. Delerablée, C. Identity-based Broadcast Encryption with Constant Size Ciphertexts and Private Keys. In: ASIACRYPT'07, 2007, LNCS 4833, 200-215. https://doi.org/10.1007/978-3-540-76900-2_12
10. Diffie, W., Hellman, M. New Directions in Cryptography. lEEE Transactions on Information Theory, 1976, 22(6), 644-654. https://doi.org/10.1109/TIT.1976.1055638
11. Du, H., Wen, Q., Zhang, S. A Provably-secure Outsourced Revocable Certificateless Signature Scheme Without Bilinear Pairings. IEEE Access, 2018, 6, 73846-73855. https://doi.org/10.1109/ACCESS.2018.2880875
12. Elgamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. IEEE Transactions on Information Theory, 1985, 31(4), 469-472. https://doi.org/10.1109/TIT.1985.1057074
13. Elhabob, R., Zhao, Y., Sella, I., Xiong, H. An Efficient Certificateless Public Key Cryptography with Authorized Equality Test in IIoT. Journal of Ambient Intelligence and Humanized Computing, 2020, 11(3), 1065-1083. https://doi.org/10.1007/s12652-019-01365-4
14. Housley, R., Polk, W., Ford, W., Solo, D. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280, 2002, IETF. https://doi.org/10.17487/rfc3280
15. Hsu, S. T., Yang, C. C., Hwang, M. S. A Study of Public Key Encryption with Keyword Search. International Journal of Network Security, 2013, 15(2), 71-79.
16. Hung, Y. H., Tsai, T. T., Tseng, Y. M., Huang, S. S. Strongly Secure Revocable ID-based Signature Without Random Oracles. Information Technology and Control, 2014, 43(3), 264-276. https://doi.org/10.5755/j01.itc.43.3.5718
17. Iriyama, S., Jimbo, K., Regoli, M. New Subclass Framework and Concrete Examples of Strongly Asymmetric Public Key Agreement. MDPI Applied Sciences, 2021, 11(12), 5540-5571. https://doi.org/10.3390/app11125540

18. Jia, X., He, D., Zeadally, S., Li, L. Efficient Revocable ID-based Signature with Cloud Revocation Server. IEEE Access, 2017, 5, 2945-2954. https://doi.org/10.1109/ACCESS.2017.2676021

19. Kumar, V., Ahmad, M., Kumar, P. An Identity-based Authentication Framework for Big Data Security. In: ICCCN'18, 2018, LNNS 46, 63-71. https://doi.org/10.1007/978-981-13-1217-5_7

20. Langrehr, R., Pan, J. Hierarchical Identity-based Encryption with Tight Multi-challenge Security. In: PKC'20, 2020, LNCS 12110, 153-183. https://doi.org/10.1007/978-3-030-45374-9_6

21. Li, Y., Cheng, Q., Liu, X., Li, X. A Secure Anonymous Identity-based Scheme in New Authentication Architecture for Mobile Edge Computing. IEEE Systems Journal, 2021, 15(1), 935-946. https://doi.org/10.1109/JSYST.2020.2979006

22. Li, H., Dai, Y., Tian, L., Yang, H. Identity-based Authentication for Cloud Computing. In: CloudCom'09, 2009, LNCS 5931, 157-166. https://doi.org/10.1007/978-3-642-10665-1_14

23. Li, J., Yu, Q., Zhang, Y. Identity-based Broadcast Encryption with Continuous Leakage Resilience. Information Sciences, 2018, 429, 177-193. https://doi.org/10.1016/j.ins.2017.11.008

24. Lin, H. Y. Secure Certificateless Two-party Key Agreement with Short Message. Information Technology and Control, 2016, 45(1), 71-76. https://doi.org/10.5755/j01.itc.45.1.12595

25. Liu, W., Liu, J., Wu, Q., Qin, B. Hierarchical Identity-based Broadcast Encryption. In: ACISP'14, 2014, LNCS 8544, 242-257. https://doi.org/10.1007/978-3-319-08344-5_16

26. Ma, S. Identity-based Encryption with Outsourced Equality Test in Cloud Computing. Information Sciences, 2016, 328, 389-402. https://doi.org/10.1016/j.ins.2015.08.053

27. Ma, M., Shi, G., Shi, X., Su, M., Li, F. Revocable Certificateless Public Key Encryption with Outsourced Semi-trusted Cloud Revocation Agent. lEEE Access, 2020, 8, 148157-148168. https://doi.org/10.1109/ACCESS.2020.3015893

28. Mehibel, N., Hamadouche, M. A New Algorithm for a Public Key Cryptosystem Using Elliptic Curve. In: EECS'17, 2017, IEEE, 17-22. https://doi.org/10.1109/EECS.2017.12

29. Qu, H., Yan, Z., Lin, X. J., Zhang, Q., Sun, L. Certificateless Public Key Encryption with Equality Test. Information Sciences, 2018, 462, 76-92. https://doi.org/10.1016/j.ins.2018.06.025

30. Raghunandan, K. R., Ganesh, A., Surendra, S., Bhavya, K. Key Generation Using Generalized Pell's Equation in Public Key Cryptography Based on the Prime Fake Modulus Principle to Image Encryption and Its Security Analysis. Cybernetics and Information Technologies, 2020, 20(3), 86-101. https://doi.org/10.2478/cait-2020-0030

31. Rivest, R. L., Shamir, A., Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 1978, 21(2), 120-126. https://doi.org/10.1145/359340.359342

32. Shamir, A. Identity-based Cryptosystems and Signature Schemes. In: CRYPTO'84, 1984, LNCS 196, 47-53. https://doi.org/10.1007/3-540-39568-7_5

33. Sun, Y., Zhang, F., Baek, J. Strongly secure Certificateless Public Key Encryption Without Pairing. In: CANS'07, 2007, LNCS 4856, 194-208. https://doi.org/10.1007/978-3-540-76969-9_13

34. Sun, Y., Zhang, Z., Shen, L. A Revocable Certificateless Signature Scheme Without Pairing. In: ICCCS'16, 2016, LNCS 10039, 355-364. https://doi.org/10.1007/978-3-319-48671-0_32

35. Takayasu, A., Watanabe, Y. Lattice-based Revocable Identity-based Encryption with Bounded Decryption Key Exposure Resistance. In: ACISP'17, 2017, LNCS 10342, 184-204. https://doi.org/10.1007/978-3-319-60055-0_10

36. Tedeschi, P., Sciancalepore, S., Eliyan, A., Pietro, R. D. LiKe: Lightweight Certificateless Key Agreement for Secure IoT Communications. IEEE Internet of Things Journal, 2020, 7(1), 621-638. https://doi.org/10.1109/JIOT.2019.2953549

37. Tsai, T. T., Tseng, Y. M. Revocable Certificateless Public Key Encryption. lEEE Systems Journal, 2015, 9(3), 824-833. https://doi.org/10.1109/JSYST.2013.2289271

38. Wu, T. Y., Chen, C. M., Wang, K. H., Meng, C., Wang, E. A Provably Secure Certificateless Public Key Encryption with Keyword Search. Journal of the Chinese Institute of Engineers, 2019, 42(1), 20-28. https://doi.org/10.1080/02533839.2018.1537807

39. Wu, T. Y., Chen, C. M., Wang, K. H., Wu, J. M. T. Security Analysis and Enhancement of a Certificateless Searchable Public Key Encryption Scheme for IIoT Environments. IEEE Access, 2019, 7, 49232-49239. https://doi.org/10.1109/ACCESS.2019.2909040

40. Wu, J. D., Tseng, Y. M., Huang, S. S. Leakage-resilient Certificateless Signature Under Continual Leakage Model. Information Technology and Control, 2018, 47(2), 363-386. https://doi.org/10.5755/j01.itc.47.2.17847

41. Wu, J. D., Tseng, Y. M., Huang, S. S., Chou, W. C. Leakage-resilient Certificateless Key Encapsulation Scheme. Informatica, 2018, 29(1), 125-155. https://doi.org/10.15388/Informatica.2018.161

42. Yang, Z., Lai, J., Sun, Y., Zhou, J. A Novel Authenticated Key Agreement Protocol with Dynamic Credential for WSNs. ACM Transactions on Sensor Networks, 2019, 15(2), 1-27. https://doi.org/10.1145/3303704

43. Yang, G., Tan, C. H., Huang, Q., Wong, D. S. Probabilistic Public Key Encryption with Equality Test. In: The Cryptographers' Track at the RSA Conference'10, 2010, LNCS 5985, 119-131. https://doi.org/10.1007/978-3-642-11925-5_9

44. Zhang, J., Mao, J. Efficient Public Key Encryption with Revocable Keyword Search in Cloud Computing. Cluster Computing, 2016, 19, 1211-1217. https://doi.org/10.1007/s10586-016-0584-7

45. Zhao, Y., Hou, Y., Chen, Y., Kumar, S., Deng, F. An Efficient Certificateless Public Key Encryption with Equality Test Toward Internet of Vehicles. T Transactions on Emerging Telecommunications Technologies, 33(5), e3812. https://doi.org/10.1002/ett.3812

46. Zhou, Y., Yang, B. Continuous Leakage-Resilient Certificateless Public Key Encryption with CCA Security. Knowledge-Based Systems, 2017, 136, 27-36. https://doi.org/10.1016/j.knosys.2017.08.019