

Certificateless Proxy Re-Encryption Without Pairing

Akshayaram Srinivasan ^{*} C. Pandu Rangan [†]

November 14, 2014

Abstract

Proxy Re-Encryption was introduced by Blaze, Bleumer and Strauss to efficiently solve the problem of delegation of decryption rights. In proxy re-encryption, a semi-honest proxy transforms a ciphertext intended for Alice to a ciphertext of the same message for Bob without learning anything about the underlying message. From its introduction, several proxy re-encryption schemes in the Public Key Infrastructure (PKI) and Identity (ID) based setting have been proposed. In practice, systems in the public key infrastructure suffer from the certificate management problem and those in identity based setting suffer from the key escrow problem. Certificateless Proxy Re-encryption schemes enjoy the advantages provided by ID-based constructions without suffering from the key escrow problem.

In this work, we construct the *first* unidirectional, single-hop CCA-secure certificateless proxy re-encryption scheme *without pairing* by extending the PKI based construction of Chow et al. proposed in 2010. We prove its security in the random oracle model under the Computational Diffie-Hellman (CDH) assumption. Prior to this work, the only secure certificateless proxy re-encryption scheme is due to Guo et al. proposed in 2013 using bilinear pairing. The construction proposed in this work is more efficient than that system and satisfies stronger security properties. We also show that the recently proposed construction of Yang et al. is insecure with respect to the security model considered in this work.

^{*}Theoretical Computer Science Lab, Dept. of Computer Science and Engg, Indian Institute of Technology, Madras. Email: akshayram1993@gmail.com

[†]Theoretical Computer Science Lab, Dept. of Computer Science and Engg, Indian Institute of Technology, Madras. Email: prangan55@gmail.com

1 Introduction

Proxy Re-Encryption (PRE) allows a semi-trusted proxy to transform ciphertexts intended for Alice (delegator) to a ciphertext of the same message for Bob (delegatee). The main goal of proxy re-encryption is to ensure that the proxy does not “learn” any information about the underlying message. PRE systems can be classified into two types based on the direction of transformation: *unidirectional* and *bidirectional*. In an unidirectional PRE scheme, the proxy has the ability to transform ciphertexts from Alice to Bob but not in the other direction. However, in a bidirectional setting the proxy can transform ciphertexts in both directions. An unidirectional PRE scheme implies a bidirectional one, as a bidirectional scheme can be obtained by trivially combining two unidirectional schemes. Blaze et al. [BBS98] also gave another method to classify PRE schemes: multi-hop, i.e., the ciphertext can be transformed from Alice to Bob to Charlie and so on; and single-hop, i.e., the ciphertext can be transformed only once.

From its introduction, proxy re-encryption has found a variety of applications such as distributed file systems [AFGH06], simplification of key distribution [BBS98], multicast [CLH05], DRM of Apple’s iTunes [Smi], privacy for public transportation [HBCDF06], telemedical system [GZZC13] and secure e-mail forwarding [BBS98].

Most of the PRE constructions found in literature are either in the public key setting or in the identity (ID) based setting. In the public key setting, there is a need to authenticate the public key of users. For this purpose, we rely on a trusted third party called as Certifying Authority (CA) to issue certificates proving the authenticity of a public key. Certificate management is a costly and a cumbersome process that inherently makes public key cryptography inefficient. Identity based setting removes the problem of certificate management but brings in a new problem. The Private Key Generator (PKG) generates the secret keys of all users and hence can decrypt all their messages. Thus, PKG must be unconditionally trusted and this leads to the *key escrow problem*. We highlight an use case where proxy re-encryption systems in public key setting as well in ID-based setting fail to provide an useful solution.

Consider a cloud storage provider with millions of users storing their data on its servers. The users wish that their data remains confidential and hence encrypt their data before storing it on the cloud. Consider an user Alice who wishes to share some of her data with another user Bob but doesn’t want to share her secret key with him. A naive approach taken by the Alice would be to download all her data, decrypt it and then re-encrypt it under Bob’s public key. This solution is highly inefficient if the data stored by Alice is of the order of hundreds of gigabytes and the computing power of Alice is very small (like a smart phone or a personal computer). Proxy re-encryption provides an efficient way to solve the above problem. Alice can give the cloud service provider with a re-encryption key which can be used to transform the encrypted data stored under Alice’s public key to data encrypted under Bob’s public key. Since the number of users is in the order of millions, proxy re-encryption schemes in the public key setting would be highly inefficient. Also, finding an authority (to issue certificates) which is unconditionally trusted by all the users is impractical. In the ID-based setting, a third party computes the secret keys of all the users and it could read all the data stored on the server. This may not be preferred by the users if they wish to store highly confidential or personal data. In such a scenario, certificateless proxy re-encryption provides an efficient solution by avoiding the certificate management problem in the PKI based setting and the key escrow problem in the ID-based setting.

Certificateless cryptography was introduced by Al-Riyami and Patterson in 2003 [ARP03]. In the certificateless setting, public key as well as the secret key of an user consists of two parts: one generated by the user himself and the other generated by a semi-trusted party called as the Key Generation Center (KGC). The public key of an user does not require a certificate and hence systems designed in this setting do not suffer from the certificate management problem. Also, the KGC does not have any information about the secret value generated by the user and hence cannot decrypt any ciphertexts. Thus, key escrow problem is also avoided. Informally, a scheme is secure in the certificateless setting if it is secure against two types of adversaries namely Type-I and Type-II. A Type-I adversary models an outside attacker (one who is different from KGC and receiver). And, Type-II adversary models an honest-but-curious KGC who tries to break the confidentiality of the scheme.

1.1 Related Work

1.1.1 Certificateless Public Key Cryptography

Since the introduction of certificateless public key cryptography by Al-Riyami and Patterson in 2003 [ARP03], a variety of schemes satisfying various notions of security have been proposed. Baek, Safavi-Naini and Susilo [BSNS05] proposed the first CCA-secure certificateless public key encryption without pairing in a weakened security model where the adversary is not allowed to replace the public key of the target identity. In 2007, Sun, Zhang and Baek [SZB07] strengthened the scheme and allowed the adversary to replace the public key of the target identity but still disallowed him to obtain the partial key of the target identity. In 2006, Libert and Quisquater [LQ06] proposed generic construction of certificateless encryption from an identity based encryption scheme. In the same year, Chow, Boyd and Nieto [CBN06] proposed a generic construction for security mediated certificateless encryption which provides instant revocation. Dent [Den08] provides an excellent survey of certificateless encryption schemes and ranks the different notions of security for a certificateless public key encryption scheme against an outside attacker as well as passive key generation center.

1.1.2 Proxy Re-Encryption

In 1998, Blaze, Bleumer and Strauss [BBS98] introduced the notion of proxy re-encryption and proposed a bidirectional CPA-secure scheme. The first unidirectional CPA-secure proxy re-encryption scheme was proposed by Ateniese et al. in [AFGH06]. In 2007, Canetti and Hohenberger [CH07] proposed the first bidirectional multihop Replayable Chosen Ciphertext (RCCA) secure scheme in the standard model. In 2008, Libert and Vergnaud [LV08] proposed the first unidirectional single hop scheme which is RCCA secure in the standard model. All the above mentioned constructions used bilinear pairing. In 2008, Weng, Deng, Liu and Chen [WDLC10, DWLC08] proposed a bidirectional CCA-secure (in the random oracle model) PRE scheme without pairing. In 2010, Chow, Weng, Yang and Deng [CWYD10] proposed an efficient CCA-secure (in the random oracle model) unidirectional proxy re-encryption scheme without pairing. Recently, Lu, Lin, Shao and Liang in [LLSL14] proposed a RCCA-secure (in the random oracle model) bidirectional proxy re-encryption scheme satisfying some additional properties like constant ciphertext size (irrespective of the number of transformations done) and master secret security (introduced in [AFGH06]).

To solve the certificate management problem in PRE, Green and Ateniese [GA07] introduced the concept of ID based proxy re-encryption scheme (IB-PRE) and proposed a multi hop IB-PRE scheme that is CCA secure in the random oracle model. Chu and Tzeng [CT07] proposed the first CCA secure ID based proxy re-encryption scheme without random oracles. In the above constructions, the ciphertext size and complexity of the decryption algorithm grew linearly in the number of transformations done to a ciphertext. Recently, Liang et al. in [LCT⁺14] proposed a CCA-secure multi hop ID based proxy re-encryption scheme in the standard model having constant ciphertext and computational complexity. IB-PRE systems solve the issue of certificate management but brings in the key escrow problem. Hence, the notion of certificateless proxy re-encryption scheme was introduced.

1.1.3 Certificateless Proxy Re-Encryption

In 2010, Sur, Jung, Park and Rhee [SJPR10] gave a construction of a certificateless proxy re-encryption and proposed a CLPRE scheme using bilinear pairing. However, Zheng et al. in 2013 [ZTGC13] gave a concrete attack on their scheme and proved that their scheme was not CCA secure as claimed. In 2013, Guo, Zhang, Zhang and Chen [GZZC13] proposed a certificateless proxy re-encryption which was RCCA - secure in the random oracle model using bilinear pairings. Recently, Yang, Xu and Zhang [YXZ14] proposed a certificateless proxy re-encryption scheme without pairing and claimed it to be CCA-secure. We show that their construction is not CCA-secure with respect to the security model considered in this work.

1.2 Our Contributions

In this paper, we propose the *first pairing-less* unidirectional, single-hop, CCA secure certificateless proxy re-encryption scheme by extending the PKI-based scheme of Chow et al. [CWYD10]. Prior to this work, the only known secure certificateless proxy re-encryption scheme was proposed by Guo et al. in 2013. We highlight several advantages enjoyed by our construction when compared to theirs.

Firstly, we propose a CCA-secure certificateless proxy re-encryption scheme whereas their construction could satisfy only a weaker notion of security namely, RCCA (Replayable Chosen Ciphertext). Secondly, the Type-I adversary considered in their work is weak when compared to the Type-I adversary we consider. In particular, their Type-I adversary is only allowed to adaptively corrupt the users in a specific way (i.e by replacing the public key and then querying the partial key) and cannot have access to a strong decryption oracle defined in [Den08]. In our work, Type-I adversary can adaptively corrupt users in any arbitrary manner and can access the strong decryption oracle. The Type-II adversary we consider is also significantly stronger than the one considered by them. Their Type-II adversary cannot replace the public key of any user and does not have access to a strong decryption oracle. On the other hand, we provide with our Type-II adversary the ability to perform both the tasks. We also compare the efficiency of their system with ours and our proposed construction outperforms theirs in almost every aspect.

Another major contribution of this work is the cryptanalysis of Yang et al.’s certificateless proxy re-encryption construction [YXZ14]. We observe that their proposed construction does not satisfy the confidentiality property of the challenge ciphertext in our security model. We give a brief overview of their construction and give a concrete attack on their system. The attack we demonstrate is inspired by the “chain-collusion attack” described in [SC09, CWYD10].

2 Framework for CL-PRE

In this section, we describe the framework for a *single hop, unidirectional* CL-PRE scheme. We use the formulation given by Baek, Safavi-Naini and Susilo [BSNS05, SZB07, Den08] for a certificateless encryption system. In this formulation, the KGC provides a partial public key and a partial secret key to the user which are combined with his own secret values to obtain his full public and secret key. This is a slight deviation from the original formulation of Al-Riyami and Patterson which allows the public key of an user to be generated independent of the partial secret key given by the KGC. Though one might view this as a slight disadvantage as the user can obtain encrypted messages only after receiving the partial public key and the partial secret key from the KGC, it has been shown in [Den08] that only schemes following the Baek, Safavi-Naini and Susilo formulation can resist “denial of decryption attacks”. A CL-PRE scheme consists of the following algorithms:

- **Setup(1^k)** : This is a probabilistic polynomial time (PPT) algorithm run by the Key Generation Center (KGC). It takes the unary encoding of security parameter 1^k as input and outputs a set of public parameters denoted by $params$ and a master secret key msk . $params$ also includes a description of a message space \mathcal{M} of finite size.
- **PartialKeyExtract($params, msk, ID$)** : This is a PPT algorithm run by the KGC. It takes public parameters $params$, the master secret key msk , and the user’s identity ID as inputs and outputs partial public key ppk and partial secret key psk .
- **UserKeyGen($ID, params$)** : This is a PPT algorithm run by the user. It takes the user’s identity ID , and $params$ as inputs and outputs a secret key sk , and a public key pk . This algorithm can be run independent of the partial key extract.
- **SetPublicKey($ID, params, pk, sk, psk, ppk$)** : This is a PPT algorithm run by the user. It takes the user’s identity ID , $params$, user generated public key pk and secret key sk , the partial secret key psk and the partial public key ppk to return the user’s full public key PK .

- **SetPrivateKey**($ID, params, sk, psk$): This is a polynomial time algorithm run by the user. It takes the user's identity ID , $params$, user generated secret key sk and the partial secret key psk to return the user's full secret key SK .
- **Re – KeyGen**($ID_i, ID_j, params, SK_i, PK_j$): This is a PPT algorithm run by the user with identity ID_i ¹. It takes ID_i, ID_j , the public parameters $params$, the full secret key SK_i of ID_i , full public key PK_j of ID_j and outputs either a re-encryption key $rk_{i \rightarrow j}$ or an error symbol \perp .
- **Encrypt**($ID_i, params, PK_i, m$): This is a PPT algorithm run by the sender. This algorithm takes the identity of the receiver ID_i , the public parameters $params$, the full public key PK_i of ID_i and a message m from \mathcal{M} as inputs. It outputs a *first level*² ciphertext C' or an error symbol \perp .
- **Re – Encrypt**($ID_i, ID_j, params, C', rk_{i \rightarrow j}$): This is a PPT algorithm run by the proxy. It takes user's identities ID_i, ID_j , $params$, a first level ciphertext C' , and a re-encryption key $rk_{i \rightarrow j}$ as inputs. It outputs a *second level*³ ciphertext C'' or an error symbol \perp .
- **Decrypt1**($ID_i, params, C', SK_i$): This is a deterministic algorithm run by the receiver. It takes in the receiver's identity ID_i , the public parameters $params$, a first level ciphertext C' and the receiver's full secret key SK_i as input and outputs a message m from \mathcal{M} or an error symbol \perp .
- **Decrypt2**($ID_i, params, C'', SK_i$): This is a deterministic algorithm run by the receiver. It takes in the receiver's identity ID_i , the public parameters $params$, a second level ciphertext C'' and the receiver's full secret key SK_i as input and outputs a message m from \mathcal{M} or an error symbol \perp .

Correctness: The algorithms stated above must satisfy the following correctness requirements. For all $k \in \mathbb{N}$, if $(params, msk) \leftarrow \text{Setup}(1^k)$ and PK_i, PK_j are the full public keys corresponding to identities ID_i, ID_j with full secret keys SK_i, SK_j then for all $m \in \mathcal{M}$:

- $\text{Decrypt1}(ID_i, params, \text{Encrypt}(ID_i, params, PK_i, m), SK_i) = m$
- If $rk_{i \rightarrow j} = \text{Re – KeyGen}(ID_i, ID_j, params, SK_i, PK_j)$, and $C'' = \text{Re – Encrypt}(ID_i, ID_j, params, \text{Encrypt}(ID_i, params, PK_i, m), rk_{i \rightarrow j})$, then $\text{Decrypt2}(ID_j, params, C'', SK_j) = m$

3 Security Model

An adversary attacking a CL-PRE scheme can be of two types: Type-I or Type-II. A Type-I adversary models an attacker from the outside (i.e anyone except the KGC) who is trying to gain some information about the underlying message in the ciphertext. A Type-II adversary models an honest but a curious KGC⁴ who tries to break the confidentiality of the scheme. For a scheme to be secure, we must argue that the adversary cannot gain any protected information unless he holds the full secret key.

We allow a Type-I adversary to adaptively corrupt users by obtaining the private keys of users⁵. We also allow the Type-I adversary to adaptively replace public keys of users with public key of his choice and to have access to a strong decryption oracle (notation as in [Den08]). If an adversary has replaced the public key corresponding to an identity, then the strong decryption oracle returns the correct decryption of a ciphertext with the secret key that inverts the current value of the public key. A Type-II adversary can also adaptively corrupt users and replace public key of identities except that of the challenge identity. Type-II adversary

¹It is suggested in [GZZC13] and [CH07] that Re-KeyGen for a unidirectional proxy re-encryption scheme need not involve the delegatee or the proxy. It is run by the delegator locally and the result is then sent to the proxy.

²A first level ciphertext can be re-encrypted by proxy

³A second level ciphertext cannot be re-encrypted by proxy

⁴This modeled by giving the Type-2 adversary with the master secret key

⁵In our security model, an adversary can obtain the full secret key of the user by querying partial key extract oracle and the user key extract oracle. Once the adversary obtains both partial key as well as user generated key for a particular identity, he can compute the full secret key of the user by running the **SetPrivateKey** algorithm. We note that this is equivalent to providing the adversary with a secret key extract oracle.

also has access to a strong decryption oracle. Both Type-I and Type-II adversaries have access to public key extraction, Re-Encryption, Re-Key generation oracles.

The security of any CL-PRE scheme is proved by means of an interactive game between a challenger and an adversary who tries to break the confidentiality of the scheme. In the confidentiality game between an adversary A and the challenger (or the simulator) C , the adversary has the access to various oracles which are simulated by C . Before setting up the oracles, C sets up a list CI of corrupted identities which is initially empty and a list of current public keys called as CPK . A does not possess the full secret key of an identity ID which is not in CI list. When A obtains the full secret key corresponding to an identity ID , C adds ID to the CI list. CPK consists of tuples of the form $(\langle ID_i \rangle, PK_i, \hat{PK}_i)$ where ID_i denotes user's identity, PK_i denotes the full public key returned by SetPublicKey algorithm, and \hat{PK}_i denotes the current public key provided by the adversary. C initializes the CPK list with $PK_i = \hat{PK}_i$ for all identities ID_i . Whenever the adversary replaces the existing public key of an identity ID_i with a new public key \hat{PK}_i , C replaces the third component of the tuple corresponding to ID_i in the CPK list with \hat{PK}_i . We now describe the oracles that A can access.

- **Public Key Extract** (\mathcal{O}_{pke}): On giving ID as input, \mathcal{O}_{pke} computes $(pk, sk) \leftarrow \text{UserKeyGen}(ID, params)$, $(ppk, psk) \leftarrow \text{PartialKeyExtract}(params, msk, ID)$, $PK \leftarrow \text{SetPublicKey}(ID, params, pk, sk, psk, ppk)$ and returns PK .
- **Partial Key Extract** (\mathcal{O}_{pex}): On giving ID as input, \mathcal{O}_{pex} computes $(ppk, psk) \leftarrow \text{PartialKeyExtract}(params, msk, ID)$ and returns (ppk, psk) . If the adversary has already obtained the User Key of ID , then the challenger adds ID to the list CI .
- **User Key Extract** (\mathcal{O}_{uke}): On giving ID as input, \mathcal{O}_{uke} computes $(pk, sk) \leftarrow \text{UserKeyGen}(params, ID)$ and returns (pk, sk) . If the adversary has already obtained the Partial key of ID , then challenger adds ID to the corrupted identities list CI .
- **Re-Key Generation** (\mathcal{O}_{rkg}): On giving (ID_i, ID_j) as input, \mathcal{O}_{rkg} returns $rk_{i \rightarrow j} = \text{Re-KeyGen}(ID_i, ID_j, params, SK_i, \hat{PK}_j)$ where \hat{PK}_j is the current public key of ID_j .
- **Re-Encryption Oracle** (\mathcal{O}_{renc}): On giving (ID_i, ID_j, C') as inputs where C' is a first level ciphertext, \mathcal{O}_{renc} computes $rk_{i \rightarrow j} = \text{Re-KeyGen}(ID_i, ID_j, params, SK_i, \hat{PK}_j)$ where \hat{PK}_j is the current public key of ID_j and returns a second level ciphertext $C'' = \text{Re-Encrypt}(ID_i, ID_j, params, C', rk_{i \rightarrow j})$.
- **Strong Decrypt-1 Oracle** (\mathcal{O}_{dec1}): On giving (ID, C') as inputs, where C' is a first level ciphertext, \mathcal{O}_{dec1} computes the decryption of the ciphertext using the private key that inverts the current value of the public key and returns m from \mathcal{M} or an error symbol.
- **Strong Decrypt-2 Oracle** (\mathcal{O}_{dec2}): On giving (ID, C'') as inputs, where C'' is a second level ciphertext, \mathcal{O}_{dec2} computes the decryption of the ciphertext using the private key that inverts the current value of the public key and returns m from \mathcal{M} or an error symbol.
- **Public Key Replacement Oracle** (\mathcal{O}_{pkr}): On giving (ID_i, \hat{PK}_i) as input, \mathcal{O}_{pkr} replaces the public key of ID_i with \hat{PK}_i by replacing the third component of the tuple corresponding to ID_i in the CPK list with \hat{PK}_i .

3.1 Security against a Type-I adversary

We first consider the security game between the challenger and a Type-I adversary. A Type-I adversary denoted by A_I has the access to all the eight oracles mentioned above.

Now, we describe the CCA security game between the challenger C and a Type-I adversary A_I . The game proceeds in 3 phases and the description of each phase is given below.

- **Initialization:** C sets up the list CI of corrupted identities and the current public key list CPK . It initializes $CI = \emptyset$ and CPK with $PK_i = \hat{PK}_i$ for all identities ID_i .

- **Phase-1:** C runs the algorithm $\text{Setup}(k)$, where k is the security parameter and computes the public parameters $params$ and the master secret key msk . It gives $params$ to A_I and keeps the msk to itself. A_I can query any of the above mentioned oracles but it cannot query the user key or the partial secret key of an identity for which it has replaced the public key ⁶. That is, it cannot query $\mathcal{O}_{uke}(ID_i)$ or $\mathcal{O}_{pex}(ID_i)$ if $PK_i \neq \hat{PK}_i$. The challenger responds to the other oracle queries correctly and consistently.
- **Challenge Phase:** Once A_I decides that the Phase-1 of the security game is over, it outputs (ID^*, M_0, M_1) where ID^* is the target identity, M_0, M_1 are messages from \mathcal{M} such that $|M_0| = |M_1|$ (where $|\cdot|$ denotes length). It is required that ID^* satisfies the following conditions.
 - ID^* does not belong to the CI list.
 - A_I has not queried the partial key of ID^* in Phase-1.
 - A_I has not issued a query $\mathcal{O}_{rkg}(ID^*, ID)$, where ID is in CI list.

Remark 1 Our assumption that the adversary does not obtain the partial keys for the challenge identity is a reasonable one. This is achieved in a system where the KGC delivers the partials keys through a confidential channel. According to the nomenclature given in [Den08], our model is termed as Strong Type I*.

C chooses $\delta \in \{0, 1\}$ uniformly at random and computes $C^* = \text{Encrypt}(ID^*, params, \hat{PK}^*, M_\delta)$, where \hat{PK}^* denotes the current value of the public key of the target identity.

- **Phase-2:** Before we describe the phase-2 of the security game we define a *challenge derivative* (similar to [SJPR10]). Informally, a challenge derivative is the challenge ciphertext or a ciphertext obtained by re-encrypting the challenge ciphertext. Formally,
 - (ID^*, C^*) is a challenge derivative.
 - If (ID_i, C_i) is a challenge derivative and A_I has issued a re-encryption query $\mathcal{O}_{renc}(ID_i, ID_j, C_i)$ to obtain the ciphertext C_j , then (ID_j, C_j) is a challenge derivative.
 - If (ID_i, C_i) is a challenge derivative and A_I has issued a re-key generation query $\mathcal{O}_{rkg}(ID_i, ID_j)$ to obtain the re-key $rk_{i \rightarrow j}$ and the ciphertext $C_j = \text{Re-Encrypt}(ID_i, ID_j, params, C_i, rk_{i \rightarrow j})$, then (ID_j, C_j) is a challenge derivative.

A_I can continue to query the above oracles but as in Phase-1 it cannot query:

- $\mathcal{O}_{uke}(ID_i)$ or $\mathcal{O}_{pex}(ID_i)$ if $PK_i \neq \hat{PK}_i$.
- $\mathcal{O}_{pex}(ID^*)$.

In addition, the adversary is not allowed to make the following queries which allows it to trivially win the game.

- $\mathcal{O}_{uke}(ID)$ and $\mathcal{O}_{pex}(ID)$ ⁷ if there exists a challenge derivative (ID, C) .
- $\mathcal{O}_{dec1}(ID^*, C^*)$ unless the public key of ID^* which was used to create the challenge has been replaced.
- $\mathcal{O}_{dec2}(ID, C)$ if (ID, C) is a challenge derivative.
- $\mathcal{O}_{renc}(ID_i, ID_j, C_i)$, if ID_j is in CI list and (ID_i, C_i) is a challenge derivative.
- $\mathcal{O}_{rkg}(ID^*, ID_j)$, if ID_j is in CI list.

⁶It is unreasonable to expect the challenger to provide secret keys for public keys replaced by the adversary

⁷Adversary is allowed to query any one of the oracles. But is not allowed to query both.

- **Guess:** A_I finally outputs a guess δ' of δ . A_I is said to win the game if $\delta = \delta'$.

We define the advantage that A_I has over this game as

$$Adv_{CLPRE, A_I}^{IND-CLPRE-CCA} = 2|Pr[\delta = \delta'] - 1/2|$$

where the probability is over the random coin tosses performed by the C , A_I and the oracles.

Definition 1 A unidirectional, single hop scheme is said to be $(t, \varepsilon, q_{pke}, q_{uke}, q_{pex}, q_{rkg}, q_{renc}, q_{dec1}, q_{dec2}, q_{pkr})$ secure against Type-I adversary, if for any t time Type-I adversary who makes at most q_{pke} queries to \mathcal{O}_{pke} , at most q_{uke} queries to \mathcal{O}_{uke} , at most q_{pex} queries to \mathcal{O}_{pex} , at most q_{rkg} queries to \mathcal{O}_{rkg} , at most q_{renc} queries to \mathcal{O}_{renc} , at most q_{dec1} queries to \mathcal{O}_{dec1} , at most q_{dec2} queries to \mathcal{O}_{dec2} , at most q_{pkr} queries to \mathcal{O}_{pkr} , we have $Adv_{CLPRE, A_I}^{IND-CLPRE-CCA} \leq \varepsilon$. \diamond

3.2 Security against a Type-II adversary

We now consider the notion of security against a Type-II adversary. Since a Type-II adversary models an honest yet curious KGC, a Type-II adversary is provided with the master secret key msk . Since the Type-II adversary has access to msk , it can compute the partial key corresponding to arbitrary identities of its choice. Thus, we do not provide a Type-II adversary explicitly with a partial key extract oracle. A Type-II adversary can replace the public key of arbitrary identities except the challenge identity. A Type-II adversary also has access to a Strong Decryption oracle. The adversary can query all the oracles that are mentioned except \mathcal{O}_{pke} . We note that an identity ID will belong to the corrupted list if the adversary has queried for the user key for that identity as the adversary can compute the partial secret key by himself.

We now describe the game between a Type-II adversary A_{II} and the challenger C . As before, the game proceeds in 3 phases.

- **Initialization:** C sets up the list CI of corrupted identities and the current public key list CPK . It initializes $CI = \emptyset$ and CPK with $PK_i = \hat{PK}_i$ for all identities ID_i .
- **Phase-1:** The C runs the algorithm $\text{Setup}(k)$, where k is the security parameter and computes the public parameter $params$ and the master secret key msk . It gives $params$ and msk to A_{II} . As in the previous case, it cannot query $\mathcal{O}_{uke}(ID_i)$ or $\mathcal{O}_{pex}(ID_i)$ if $PK_i \neq \hat{PK}_i$.
- **Challenge Phase:** Once A_{II} decides that the Phase-1 of the security game is over, it outputs (ID^*, M_0, M_1) where ID^* is the target identity, M_0, M_1 are messages from \mathcal{M} such that $|M_0| = |M_1|$ (where $|\cdot|$ denotes length). It is required that ID^* satisfies the following conditions.
 - ID^* does not belong to the CI list.
 - A_{II} has not queried $\mathcal{O}_{rkg}(ID^*, ID)$, where ID is in CI list.
 - $PK^* = \hat{PK}^*$. That is, A_{II} has not replaced the public key corresponding to the challenge identity. This is a standard requirement in the security model for a Type-II adversary.

C chooses $\delta \in \{0, 1\}$ uniformly at random and computes $C^* = \text{Encrypt}(ID^*, params, PK^*, M_\delta)$.

- **Phase-2:** A_{II} can continue to query the above mentioned oracles in this phase but as in Phase-1 is disallowed to query:
 - $\mathcal{O}_{uke}(ID_i)$ or $\mathcal{O}_{pex}(ID_i)$ if $PK_i \neq \hat{PK}_i$

In addition, it is not allowed to make the following queries which trivially allow it to win the game:

- $\mathcal{O}_{uke}(ID)$ and $\mathcal{O}_{pex}(ID)$ ⁸ if there exists a challenge derivative (ID, C) .

⁸Again, adversary is allowed to query one of the oracles but not both

- $\mathcal{O}_{dec1}(ID^*, C^*)$ if $PK^* = P\hat{K}^*$.
- $\mathcal{O}_{dec2}(ID, C)$ if (ID, C) is a challenge derivative.
- $\mathcal{O}_{renc}(ID_i, ID_j, C_i)$, if ID_j is in CI list and (ID_i, C_i) is a challenge derivative.
- $\mathcal{O}_{rkg}(ID^*, ID_j)$, if ID_j is in CI list.

- **Guess:** A_{II} finally outputs a guess δ' of δ . The adversary is said to win the game if $\delta = \delta'$.

We define the advantage that the adversary A_{II} has over this game as

$$Adv_{CLPRE, A_{II}}^{IND-CLPRE-CCA} = 2|Pr[\delta = \delta'] - 1/2|$$

where the probability is over the random coin tosses performed by C , A_{II} and the oracles.

Definition 2 A unidirectional, single hop scheme is said to be $(t, \varepsilon, q_{pke}, q_{uke}, q_{rkg}, q_{renc}, q_{dec1}, q_{dec2}, q_{pkr})$ secure against Type-II adversary, if for any t time Type-II adversary who makes at most q_{pke} queries to \mathcal{O}_{pke} , at most q_{uke} queries to \mathcal{O}_{uke} , at most q_{rkg} queries to \mathcal{O}_{rkg} , at most q_{renc} queries to \mathcal{O}_{renc} , at most q_{dec1} queries to \mathcal{O}_{dec1} , at most q_{dec2} queries to \mathcal{O}_{dec2} , at most q_{pkr} queries to \mathcal{O}_{pkr} , we have $Adv_{CLPRE, A_{II}}^{IND-CLPRE-CCA} \leq \varepsilon$. \diamond

4 Cryptanalysis of Yang et al.'s CL-PRE scheme

We first give an overview of Yang et al.'s CL-PRE scheme and later describe our Chosen Ciphertext attack against the confidentiality of their construction.

4.1 Yang et al.'s CL-PRE scheme

- **Setup(1^k) :**
 - Generate a k -bit prime number q and a group \mathbb{G} of order q . Pick a random generator $g \in \mathbb{G}$.
 - Randomly pick a $x \in_R \mathbb{Z}_q^*$ and compute $y = g^x$.
 - Choose cryptographic hash functions $H_1 : \{0, 1\}^* \times \mathbb{G} \rightarrow \mathbb{Z}_q^*$, $H_2 : \mathbb{G} \rightarrow \{0, 1\}^{n+n'}$ for some $n, n' \in \mathbb{N}$, $H_3 : \{0, 1\}^* \rightarrow \mathbb{Z}_q^*$, $H_4 : \mathbb{G} \rightarrow \mathbb{Z}_q^*$, $H_5 : \{0, 1\}^* \rightarrow \mathbb{Z}_q^*$ and $H_6 : \{0, 1\}^* \rightarrow \mathbb{Z}_q^*$.
 - Output the public parameters $params = \{G, g, q, y, n, n', H_1, H_2, H_3, H_4, H_5, H_6\}$ and the master secret key $msk = x$.
- **PartialKeyExtract($params, msk, ID_A$) :** Pick a random $s_A \in_R \mathbb{Z}_q^*$ and compute $\omega_A = g^{s_A}$ and $t_A = s_A + xH_1(ID_A, \omega_A)$. Return the partial public key $ppk = \omega_A$ and the partial secret key $psk = t_A$.
- **UserKeyGen($ID_A, params$) :** Pick random $z_A, v_A \in_R \mathbb{Z}_q^*$ and compute $\mu_A = g^{z_A}$ and $\varphi_A = g^{v_A}$. Return $pk = (\mu_A, \varphi_A)$ and $sk = (z_A, v_A)$.
- **SetPublicKey($ID_A, params, pk, ppk$) :** Set the public key to be $PK = (\omega_A, \mu_A, \varphi_A)$.
- **SetPrivateKey($ID_A, params, sk, psk$) :** Set the private key $SK = (t_A, z_A, v_A)$.
- **Re – KeyGen($ID_A, ID_B, params, PK_A, SK_A, PK_B$) :** Parse PK_A as $(\omega_A, \mu_A, \varphi_A)$, SK_A as (t_A, z_A, v_A) and PK_B as $(\omega_B, \mu_B, \varphi_B)$. Compute $\gamma_B = \omega_B y^{H_1(ID_B, \omega_B)}$ and $X_{AB} = H_3(\gamma_B^{v_A}, \varphi_B^{v_A}, ID_A, PK_A, ID_B, PK_B)$. Return $RK_{A \rightarrow B} = (t_A H_4(\mu_A) + z_A) X_{AB}$.
- **Encrypt($ID_A, params, PK_A, m$) :** Parse PK_A as $(\omega_A, \mu_A, \varphi_A)$. Compute $\gamma_A = \omega_A y^{H_1(ID_A, \omega_A)}$ and $Y_A = \gamma_A^{H_4(\mu_A)} \mu_A$. Pick a random $\sigma \in \{0, 1\}^{n'}$ and compute $r = H_5(m, \sigma, ID_A, pk_A)$. Pick a random $\hat{r} \in \mathbb{Z}_q^*$ and compute $C_A = (c_1, c_2, c_3, c_4)$ such that $c_1 = g^{\hat{r}}$, $c_2 = g^r$, $c_3 = (m || \sigma) \oplus H_2(Y_A^r)$ and $c_4 = \hat{r} + rH_6(c_1, c_2, c_3)$. Return C_A .

- **Re – Encrypt**($ID_A, params, C_A, RK_{A \rightarrow B}$) : Parse C_A as (c_1, c_2, c_3, c_4) and check whether $g^{c_4} = c_1 \cdot c_2^{H_6(c_1, c_2, c_3)}$. If not return \perp . Else, compute $c'_2 = c_2^{RK_{A \rightarrow B}}$ and $c'_3 = c_3$ and return $C'_B = (c'_2, c'_3)$.
- **Decrypt1**($ID_A, params, C_A, SK_A$) : Parse C_A as (c_1, c_2, c_3, c_4) and SK_A as (t_A, z_A, v_A) . Then compute:
 - Check if $g^{c_4} = c_1 \cdot c_2^{H_6(c_1, c_2, c_3)}$. If not, return \perp .
 - Compute $m || \sigma = c_3 \oplus H_2(c_2^{t_A H_4(\mu_A) + z_A})$.
 - Check if $c_2 = g^{H_5(m, \sigma, ID_A, pk_A)}$.
- **Decrypt2**($ID_B, params, C'_B, SK_B, PK_A$) : Parse C'_B as (c'_2, c'_3) , PK_A as $(\omega_A, \mu_A, \varphi_A)$ and SK_B as (t_B, z_B, v_B) . Then compute:
 - Compute $X_{AB} = H_3(\varphi_A^{t_B}, \varphi_A^{v_B}, ID_A, PK_A, ID_B, PK_B)$.
 - Compute $m || \sigma = c'_3 \oplus H_2(c'_2^{1/X_{AB}})$.
 - Compute $r' = H_5(m, \sigma, ID_A, pk_A)$, $\gamma_A = \omega_A y^{H_1(ID_A, \omega_A)}$ and $Y_A = \gamma_A^{H_4(\mu_A)} \mu_A$.
 - If $Y_A^{X_{AB} r'} = c'_2$ then return m . Else, return \perp .

4.2 Attack against confidentiality

We now show that the construction of Yang et al. is insecure in the security model which we consider in this work. In particular, Yang et al. scheme is insecure once the adversary is given access to the user key extract oracle. We illustrate this weakness with a concrete attack against their construction. In the attack, we demonstrate how to obtain the secret key of an uncorrupted user and how to break the confidentiality of the construction using the secret key.

We describe a Type-I adversary \mathcal{A} against the CCA-security challenger for the above construction. The adversary works as follows:

- \mathcal{A} chooses ID_A as the challenge identity. In addition, it chooses two more identities, ID_B and ID_C .
- \mathcal{A} obtains the challenge ciphertext C_A^* .
- The adversary makes the following queries to their respective oracles:
 - \mathcal{A} queries for the user keys for ID_B to \mathcal{O}_{uke} and obtains (z_B, v_B) .
 - \mathcal{A} queries for the user keys of ID_C to \mathcal{O}_{uke} and obtains (z_C, v_C) .
 - \mathcal{A} queries for the partial keys for ID_C to \mathcal{O}_{pex} to obtain t_C . The adversary now has the full secret key of ID_C and hence ID_C is added to the corrupted list.
 - It now queries the re-key oracle \mathcal{O}_{rkg} to obtain the re-key $RK_{B \rightarrow C}$.
 - \mathcal{A} queries the re-encryption oracle \mathcal{O}_{renc} to re-encrypt the challenge ciphertext C_A^* from ID_A to ID_B and obtains C_B^* .

We note that all the queries made by the adversary are valid and allowed by our security model.

- Now, the adversary is in possession of the following information:
 - The full secret key of ID_C : (t_C, z_C, v_C) .
 - The user key of ID_B : (z_B, v_B) .
 - Re-key from ID_B to ID_C : $RK_{B \rightarrow C}$.
 - Re-encryption of challenge ciphertext to ID_B : $C_B^* = (c'_2, c'_3)$.
- \mathcal{A} then computes the partial key for ID_B as follows:

- It computes $X_{BC} = H_3((\varphi_B)^{t_C}, (\varphi_B)^{v_C}, ID_B, PK_B, ID_C, PK_C)$ using the knowledge of secret key of C .
- It then computes $y_B = RK_{B \rightarrow C}/X_{BC}$. Note that $y_B = t_B H_4(\mu_B) + z_B$.
- It then computes $t_B = (y_B - z_B)/H_4(\mu_B)$.
- Now, \mathcal{A} has access to the full secret key (t_B, z_B, v_B) of ID_B . It then decrypts C_B^* and obtains the challenge message m . Thus, \mathcal{A} wins the game with probability 1.

5 Complexity assumptions

For a prime q , let \mathbb{Z}_q denote the set $\{0, 1, 2, \dots, q-1\}$ and let \mathbb{Z}_q^* denote $\mathbb{Z}_q - \{0\}$. For a finite set X , $x \in_R X$ means x is chosen uniformly at random from set X .

Definition 3 Let \mathbb{G} be a cyclic multiplicative group of prime order q and g is a generator of \mathbb{G} . The Computational Diffie-Hellman (CDH) problem in \mathbb{G} asks to compute g^{ab} , given (g, g^a, g^b) without the knowledge of a or b where $a, b \in_R \mathbb{Z}_q^*$. \diamond

Definition 4 For an adversary A , we define the advantage in solving the CDH problem as $Adv_A^{CDH} = Pr[A(g, g^a, g^b) = g^{ab}]$, where the probability is taken over the random choices of a, b and the random bits consumed by the adversary A . We say the (t, ε) -CDH assumption holds, if there exists no adversary who runs in time t and has an advantage at least ε in solving CDH. \diamond

Bao et al. [BDZ03] introduced a variant of the CDH problem which we term as Modified Computational Diffie-Hellman (M-CDH) problem which is formally defined below. It was also shown by [BDZ03] that the M-CDH problem and the CDH problem are equivalent in the same group.

Definition 5 Let \mathbb{G} be a cyclic multiplicative group of prime order q and g is a generator of \mathbb{G} . The Modified Computational Diffie-Hellman (M-CDH) problem in \mathbb{G} asks to compute $g^{b/a}$, given (g, g^a, g^b) without the knowledge of a or b where $(a, b) \in_R \mathbb{Z}_q^{*2}$. \diamond

Definition 6 For an adversary A , we define the advantage in solving the M-CDH problem as $Adv_A^{M-CDH} = Pr[A(g, g^a, g^b) = g^{b/a}]$, where the probability is taken over the random choices of a, b and the random bits consumed by the adversary A . We say the (t, ε) -M-CDH assumption holds, if there exists no adversary who runs in time t and has an advantage at least ε in solving M-CDH. \diamond

Lemma 1 [BDZ03] The M-CDH problem is equivalent to the CDH problem in the same group.

6 Our Scheme

Our Certificateless Proxy Re-encryption scheme extends the PKI-based scheme of Chow et al. [CWYD10] to the certificateless setting using the "token-controlled encryption" technique explained in section 3.

- **Setup**(1^k):
 - Choose two large primes p and q such that $q|p-1$ and the bit length of q is the security parameter k . Let \mathbb{G} be a sub group of \mathbb{Z}_p^* of order q and g is a generator of \mathbb{G} .
 - Select $x \in_R \mathbb{Z}_q^*$ and compute $y = g^x$.
 - Choose cryptographic Hash functions: $H : \mathbb{G} \rightarrow \mathbb{Z}_q^*$, $H_1 : \{0, 1\}^* \times \mathbb{G} \rightarrow \mathbb{Z}_q^*$, $H_2 : \{0, 1\}^* \times \mathbb{G}^3 \rightarrow \mathbb{Z}_q^*$, $H_3 : \mathbb{G} \rightarrow \{0, 1\}^{l_0+l_1}$, $H_4 : \{0, 1\}^{l_0} \times \{0, 1\}^{l_1} \rightarrow \mathbb{Z}_q^*$, $H_5 : \mathbb{G} \times \mathbb{G} \times \{0, 1\}^{l_0+l_1} \rightarrow \mathbb{Z}_q^*$, $H_6 : \{0, 1\}^* \times \mathbb{G}^2 \rightarrow \mathbb{Z}_q^*$. Here, l_0, l_1 are determined by the security parameter k . Set the message space \mathcal{M} to be $\{0, 1\}^{l_0}$.

- Output public parameters $params = (p, q, \mathbb{G}, g, y, H, H_1, H_2, H_3, H_4, H_5, H_6)$ and the master secret key $msk = x$ and the message space \mathcal{M} .
 - **PartialKeyExtract**($params, msk, ID$) :
 - Choose $s_1, s_2, s_3 \in_R \mathbb{Z}_q^*$ and compute $Q_1 = g^{s_1}, Q_2 = g^{s_2}, Q_3 = g^{s_3}, .$
 - Compute $S_1 = s_1 + xH_1(ID, Q_1), S_2 = s_2 + xH_1(ID, Q_2)$ and $S_3 = s_3 + xH_2(ID, Q_1, Q_2, Q_3).$
 - Output partial secret key $psk = (S_1, S_2)$ and the partial public key $ppk = (Q_1, Q_2, Q_3, S_3).$
 - **UserKeyGen**($ID, params$) :
 - Choose $z_1, z_2 \in_R \mathbb{Z}_q^*$ and compute $(g^{z_1}, g^{z_2}).$
 - Output $sk = (U_1, U_2) = (z_1, z_2)$ and $pk = (P_1, P_2) = (g^{z_1}, g^{z_2}).$
 - **SetPublicKey**($ID, params, pk, sk, ppk, psk$) :
 - Choose $t_1, t_2 \in_R \mathbb{Z}_q^*$ and compute $T_1 = g^{t_1}$ and $T_2 = g^{t_2}.$
 - Compute $\mu_1 = t_1 + S_1H_6(ID, P_1, T_1)$ and $\mu_2 = t_2 + S_2H_6(ID, P_2, T_2).$
 - Output the full public key of the user $PK = (P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2).$
- In a certificateless setting, public key of any user must satisfy the property of *public verifiability*. The validity of the public key of an user can be checked by running the algorithm **PublicKeyVerification**(ID, PK) which is given by
- Parse PK as $(P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2).$
 - Compute $R_1 = Q_1.y^{H_1(ID, Q_1)}$ and $R_2 = Q_2.y^{H_1(ID, Q_2)}.$
 - Check if $g^{\mu_1} \stackrel{?}{=} (T_1)(R_1)^{H_6(ID, P_1, T_1)}, g^{\mu_2} \stackrel{?}{=} (T_2)(R_2)^{H_1(ID, P_2, T_2)}, g^{S_3} \stackrel{?}{=} (Q_3)(y^{H_2(ID, Q_1, Q_2, Q_3)}).$
 - If any of the above check fails, output "failure". Else, output "success".
- **SetPrivateKey**($ID, params, sk, psk$) :
 - Output the full secret key of the identity ID as $SK = (U_1, U_2, S_1, S_2).$
 - **Re – KeyGen**($ID_i, ID_j, params, SK_i, PK_j$) :
 - Parse SK_i as $(U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$ and PK_j as $(P_{j,1}, P_{j,2}, Q_{j,1}, Q_{j,2}, Q_{j,3}, S_{j,3}, T_{j,1}, T_{j,2}, \mu_{j,1}, \mu_{j,2})$ and check the validity of public key of ID_j with **PublicKeyVerification**(ID_j, PK_j). If the check fails, output \perp .
 - Compute $R_{j,1} = Q_{j,1}(y^{H_1(ID_j, Q_{j,1})}), X_1 = P_{j,1}(R_{j,1}^{H(P_{j,1})}), X = P_{j,1}(P_{j,2})^{H(P_{j,1})}$ and $\alpha = H(X).$
 - Pick $h \in_R \{0, 1\}^{l_0}$ and $\pi \in_R \{0, 1\}^{l_1}$ and compute $v = H_4(h, \pi).$
 - Compute $V = (X_1)^v, W = H_3(g^v) \oplus (h||\pi)$ and $rk = \frac{h}{U_{i,1} + H(P_{i,1})U_{i,2} + \alpha(S_{i,1} + H(R_{i,1})S_{i,2})}.$
 - Output the re key as $rk_{i \rightarrow j} = (rk, V, W).$
 - **Encrypt**($ID_i, params, PK_i, m$) :
 - Parse PK_i as $(P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{i,1}, T_{i,2}, \mu_{i,1}, \mu_{i,2})$ and check the validity of PK_i using **PublicKeyVerification**(ID_i, PK_i). If the check fails, output \perp .
 - Compute $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$ and $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})}).$ Also, compute $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$ and $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}.$ Compute $\alpha = H(X).$ Set $Z = (X(Y)^\alpha)$
 - Pick $u \in_R \mathbb{Z}_q^*$ and $w \in_R \{0, 1\}^{l_1}.$ Compute $r = H_4(m, w).$

- Compute $D = (Z)^u$, $E = Z^r$, $F = H_3(g^r) \oplus (m||w)$ and $S = u + rH_5(D, E, F)$.
- Output $C' = (D, E, F, S)$ as the first level ciphertext.

Remark 2 In order to avoid recomputing $R_{i,1}, R_{i,2}, X, Y, Z, \alpha$ for each message transmission, they can be computed once and stored locally.

The validity of a first level ciphertext can be verified by $\text{CiphertextVerification}(ID_i, C')$ which is explained below

- Compute Z as above.
- Check if $(Z)^S \stackrel{?}{=} D.E^{H_5(D,E,F)}$.
- If the check fails output "failure". Else, output "success"
- **Re – Encrypt** $(ID_i, ID_j, params, C', rk_{i \rightarrow j})$:
 - Parse $rk_{i \rightarrow j}$ as (rk, V, W) and C' as (D, E, F, S) and check ciphertext validity by executing $\text{CiphertextVerification}(ID_i, C')$. If the check fails output \perp .
 - Compute $E' = E^{rk}$.
 - Output (E', F, V, W) as the second level ciphertext.
- **Decrypt1** $(ID_i, params, C', SK_i)$:
 - Obtain the public key corresponding to ID_i and parse it as $(P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{i,1}, T_{i,2}, \mu_{i,1}, \mu_{i,2})$. Parse SK_i as $(U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$ and C' as (D, E, F, S) and check for ciphertext validity using $\text{CiphertextVerification}(ID_i, C')$.
 - Compute $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$ and $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$. Also, compute $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$ and $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$. Compute $\alpha = H(X)$ and set $Z = (X(Y)^\alpha)$. Set $K = U_{i,1} + H(P_{i,1})U_{i,2} + \alpha(S_{i,1} + H(R_{i,1})S_{i,2})$
 - Compute $(m||w) = F \oplus H_3(E^{\frac{1}{K}})$. Return m if $E = (Z)^{H_4(m,w)}$ holds. Otherwise return \perp .

Remark 3 Again, $R_{i,1}, R_{i,2}, X, Y, Z, \alpha$ can be precomputed and stored locally to avoid re-computation for every decryption.

- **Decrypt2** $(ID_j, params, C'', SK_j)$:
 - Parse C'' as (E', F, V, W) , PK_j as $(P_{j,1}, P_{j,2}, Q_{j,1}, Q_{j,2}, Q_{j,3}, S_{j,3}, T_{j,1}, T_{j,2}, \mu_{j,1}, \mu_{j,2})$ and SK_j as $(U_{j,1}, U_{j,2}, S_{j,1}, S_{j,2})$.
 - Compute $R_{j,1} = Q_{j,1}(y^{H_1(ID_j, Q_{j,1})})$ and $X_1 = P_{j,1}(R_{j,1}^{H(P_{j,1})})$.
 - Compute $(h||\pi) = W \oplus H_3(V^{\overline{U_{j,1} + H(P_{j,1})S_{j,1}}})$ and $(m||w) = F \oplus H_3(E'^{1/h})$. Output m if $V = (X_1)^{H_4(h,\pi)}$, $E' = g^{h(H_4(m,w))}$. Otherwise return \perp .

Remark 4 $R_{j,1}, X_1$ can be computed once and stored locally.

6.1 Correctness

To make the notations less cumbersome, let $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$ and $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$. Let $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$ and $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$. Let $\alpha = H(X)$ and $Z = (X(Y)^\alpha)$. Let $K = U_{i,1} + H(P_{i,1})U_{i,2} + \alpha(S_{i,1} + H(R_{i,1})S_{i,2})$

- **Correctness of PublicKeyVerification:** If $PK = (P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2)$ is a valid public key output by the SetPublicKey algorithm then, $(P_1)(R_1)^{H_1(ID, P_1)} = (g^{U_1})(g^{S_1})^{H_1(ID, P_1)} = g^{\mu_1}$. We can similarly verify that $(P_2)(R_2)^{H_1(ID, P_2)} = (g^{U_2})(g^{S_2})^{H_1(ID, P_2)} = g^{\mu_2}$ and $(Q_3)(y^{H_2(ID, Q_1, Q_2, Q_3)}) = g^{S_3}(g^x)^{H_2(ID, Q_1, Q_2, Q_3)} = g^{S_3}$.
- **Correctness of CiphertextVerification:** If C' is a properly generated first level ciphertext then, $C' = (D, E, F, S) = (Z^u, Z^r, H_3(g^r) \oplus (m||w), u + rH_5(D, E, F))$. Therefore, $D.E^{H_5(D, E, F)} = (Z)^{u+rH_5(D, E, F)} = (Z)^S$
- **Correctness of Decrypt1:** If C' is a correctly generated first level ciphertext, then $C' = (D, E, F, S) = (Z^u, Z^r, H_3(g^r) \oplus (m||w), u + rH_5(D, E, F))$. We can see that $E = Z^r = (g^{U_{i,1} + H(P_{i,1})U_{i,2} + \alpha(S_{i,1} + H(R_{i,1})S_{i,2})})^r = g^{Kr}$. Hence, $F \oplus H_3(E^{\frac{1}{K}}) = F \oplus H_3(g^r) = m||w$.
- **Correctness of Decrypt2:** If $C'' = (E', F, V, W)$ is a correctly generated second level ciphertext, then $E' = E^{rk} = g^{rK(h/K)} = g^{rh}$. and $V = g^{v \cdot (U_{j,1} + H(P_{j,1})S_{j,1})}$. Hence, $W \oplus H_3(V^{\frac{1}{U_{j,1} + H(P_{j,1})S_{j,1}}}) = W \oplus H_3(g^v) = (h||\pi)$. Therefore, $F \oplus H_3(E'^{\frac{1}{h}}) = F \oplus H_3(g^r) = (m||w)$.

7 Security

We prove that our scheme is secure against both Type-I and Type-II adversaries.

7.1 Security against Type-I adversary

If there exists a Type-I adversary who breaks our scheme with non-negligible probability we show how to construct an adversary that solves the M-CDH problem with some non-negligible probability.

Theorem 1 Suppose $H, H_1, H_2, H_3, H_4, H_5, H_6$ are random oracles and there exists a $(t, \varepsilon, q_{pke}, q_{uke}, q_{pex}, q_{rkg}, q_{renc}, q_{dec1}, q_{dec2}, q_{pkr})$ IND-CLPRE-CCA adversary A_I against our scheme making at most q_H queries to H and at most q_{H_i} queries to random oracles H_i where $1 \leq i \leq 6$, then there exists a PPT algorithm C which solves the M-CDH problem with advantage

$$\varepsilon' \geq (1/q_{H_3}) \left(\frac{\varepsilon(1-\nu)^{q_{rkg}+1}}{e(1+q_{pex}+q_{rkg})} - \left(\frac{q_{H_4}}{2^{l_0+l_1}} + \frac{q_{H_5}}{2^{l_0+l_1}} + \frac{q_{renc}}{q} + q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q \right) \right) \right)$$

where ν is the advantage an attacker may have over the EUF-CMA security game of the Schnorr signature scheme and which runs in time t'

$$t' \leq t + (T_1)O(1) + (T_2)t_{exp}$$

where $T_1 = q_H + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{pke} + q_{uke} + q_{pex} + q_{rkg} + q_{renc} + q_{dec1} + q_{dec2} + q_{pkr}$, $T_2 = 10q_{pke} + 10q_{uke} + 10q_{pex} + 5q_{rkg} + 8q_{renc} + 7q_{dec1} + 9q_{dec2}$ and t_{exp} is the time taken for exponentiation in group G .

Proof. We now describe the challenger C which interacts with the adversary A_I and solves the M-CDH problem. C is given an instance of the M-CDH problem (g, g^a, g^b) . It sets $y = g^a$ and implicitly defines the master secret key as a . C maintains a list CI of corrupted identities which is initially empty. The challenger also maintains a list of current public keys called as CPK list consisting of tuples of the form (ID_i, PK_i, \hat{PK}_i) where PK_i denotes the full public key returned by SetPublicKey algorithm, and \hat{PK}_i denotes the current public key. The list is initialized with $PK_i = \hat{PK}_i$ for all identities ID_i . A_I requests the access to several oracles which are to be simulated by C . We now describe the confidentiality game between the adversary A_I and the challenger C .

7.1.1 Phase-1

In phase-1 of the game, A_I requests the access to several oracles which are simulated by the challenger. We now describe how C answers A_I 's oracle queries.

- **H queries:** C maintains a H -list of tuples $(\langle A \rangle, \alpha)$. On receiving a query, C searches H -list for $(\langle A \rangle, \alpha)$. If found, outputs α . Else, chooses $\alpha \in_R \mathbb{Z}_q^*$ and returns α . C adds $(\langle A \rangle, \alpha)$ to the H -list.
- **H_1 queries:** C maintains a H_1 -list of tuples $(\langle ID, Q \rangle, e_0)$. On receiving a query, C searches H_1 -list for $(\langle ID, Q \rangle, e_0)$. If found, outputs e_0 . Else, chooses $e_0 \in_R \mathbb{Z}_q^*$ and returns e_0 . C adds $(\langle ID, Q \rangle, e_0)$ to the H_1 -list.
- **H_2 queries:** C maintains a H_2 -list of tuples $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_1)$. On receiving a query, C searches H_2 -list for $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_1)$. If found, outputs e_1 . Else, chooses $e_1 \in_R \mathbb{Z}_q^*$ and returns e_1 . C adds $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_1)$ to the H_2 -list.
- **H_3 queries:** C maintains a H_3 -list of tuples $(\langle A \rangle, h)$. On receiving a query, C searches H_3 -list for $(\langle A \rangle, h)$. If found, outputs h . Else, chooses $h \in_R \{0, 1\}^{l_0+l_1}$ and returns h . It adds $(\langle A \rangle, h)$ to the H_3 -list.
- **H_4 queries:** C maintains a H_4 -list of tuples $(\langle m, w \rangle, r)$. On receiving a query, C searches H_4 -list for $(\langle m, w \rangle, r)$. If found, outputs r . Else, chooses $r \in_R \mathbb{Z}_{q^*}$ and returns r . It adds $(\langle m, w \rangle, r)$ to the H_4 -list.
- **H_5 queries:** C maintains a H_5 -list of tuples $(\langle A, B, C \rangle, p)$. On receiving a query, C searches H_5 -list for $(\langle A, B, C \rangle, p)$. If found, outputs p . Else, chooses $p \in_R \mathbb{Z}_{q^*}$ and gives p as output. It adds $(\langle A, B, C \rangle, p)$ to the H_5 -list.
- **H_6 queries:** C maintains a H_6 -list of tuples $(\langle ID, A, B \rangle, p)$. On receiving a query, C searches H_6 -list for $(\langle ID, A, B \rangle, p)$. If found, outputs p . Else, chooses $p \in_R \mathbb{Z}_{q^*}$ and gives p as output. It adds $(\langle ID, A, B \rangle, p)$ to the H_6 -list.
- **Public Key Extract \mathcal{O}_{pke} :** C maintains a public key list of tuples $(\langle ID \rangle, P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2, coin)$. On receiving a query on ID , C searches the public key list for key value ID . If found, then it outputs the corresponding the public key. Else, using the technique outlined in [Cor00] it tosses a biased coin and picks $coin \in \{0, 1\}$ such that $Pr[coin = 0] = \theta$ where θ is to be determined later.
 If **coin=0**, C chooses $(U_1, U_2) = (z_1, z_2) \in_R \mathbb{Z}_q^*$ and computes $(P_1, P_2) = (g^{z_1}, g^{z_2})$. It then chooses $(S_1, S_2, S_3, e_1, e_2, e_3) \in \mathbb{Z}_q^*$ and computes $Q_1 = g^{S_1}y^{-e_1}$, $Q_2 = g^{S_2}y^{-e_2}$, $Q_3 = g^{S_3}y^{-e_3}$. It then checks in the H_1 list if there exists a tuple $(\langle ID, Q_1 \rangle, f_1)$. If yes, then it re-chooses $(S_1, e_1) \in_R \mathbb{Z}_q^*$ and re-computes Q_1 . It then adds $(\langle ID, Q_1 \rangle, e_1)$ to the H_1 list. Similarly, it checks in the H_1 list if there exists a tuple $(\langle ID, Q_2 \rangle, f_2)$. If yes, then it re-chooses $(S_2, e_2) \in_R \mathbb{Z}_q^*$ and re-computes Q_2 . It then adds $(\langle ID, Q_2 \rangle, e_3)$ to the H_1 list. Likewise, it searches the H_2 list for a tuple $(\langle ID, Q_1, Q_2, Q_3 \rangle, f_3)$. If found, it re-chooses $(S_3, e_3) \in_R \mathbb{Z}_q^*$ and then re-computes Q_3 . It finally adds, $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_3)$ to the H_2 list. It then chooses $t_1, t_2 \in_R \mathbb{Z}_q^*$ and computes $T_1 = g^{t_1}$ and $T_2 = g^{t_2}$. It then computes $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$ and $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$. It adds $(\langle ID, coin \rangle, (Q_1, Q_2, Q_3, S_3), (S_1, S_2))$ to the partial key list, $(\langle ID \rangle, P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2, coin)$ to the public key list, $(\langle ID, coin \rangle, (P_1, P_2), (U_1, U_2))$ to the user key list and $(\langle ID, coin \rangle, U_1, U_2, S_1, S_2)$ to the private key list. It returns $(P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2)$ as the public key.

Proposition 1 The public key computed by public key extract for the case $coin = 0$ is identically distributed to the public key computed by the **SetPublicKey** algorithm and additionally it passes the **PublicKeyVerification** test.

Proof The public key generated above is given by:

$$(g^{z_1}, g^{z_2}, g^{S_1} y^{-e_1}, g^{S_2} y^{-e_2}, g^{S_3} y^{-e_3}, S_3, g^{t_1}, g^{t_2}, \mu_1, \mu_2)$$

where $z_1, z_2, S_1, S_2, S_3, e_1, e_2, e_3, t_1, t_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$ and $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$.

The public key output by **SetPublicKey** is given by:

$$(g^{z_1}, g^{z_2}, g^{s_1}, g^{s_2}, g^{s_3}, S_3, g^{t_1}, g^{t_2}, \mu_1, \mu_2)$$

where $z_1, z_2, s_1, s_2, s_3, t_1, t_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $S_3 = s_3 + x H_2(ID, Q_1, Q_2, Q_3)$, $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$, $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$.

It is easy to see that both the public keys are identically distributed under assumption that H_2 behaves as a random oracle.

We now show that the public key generated by public key extract passes the **PublicKeyVerification** test.

- We note that $R_1 = Q_1 \cdot y^{H_1(ID, Q_1)} = g^{S_1} y^{-e_1} \cdot y^{e_1} = g^{S_1}$ and similarly $R_2 = Q_2 \cdot y^{H_1(ID, Q_2)} = g^{S_2}$.
- $g^{\mu_1} = g^{t_1} \cdot g^{S_1 \cdot (H_6(ID, P_1, T_1))} = (T_1)(R_1)^{H_6(ID, P_1, T_1)}$. Similarly, one can verify $g^{\mu_2} = (T_2)(R_2)^{H_1(ID, P_2, T_2)}$.
- $(Q_3)(y^{H_2(ID, Q_1, Q_2, Q_3)}) = g^{S_3} y^{-e_3} \cdot y^{e_3} = g^{S_3}$.

Thus, the above generated public key passes the public key verification test. ■

If **coin=1**, C chooses $(s_1, s_2, \mu_1, \mu_2, \beta_1, \beta_2, z_1, z_2) \in \mathbb{Z}_q^*$ and computes $P_1 = g^{z_1}$ and $P_2 = g^{z_2}$. Let $r = H(P_1)$ and let $X = P_1(P_2)^r$. Let $\alpha = H(X)$. It then sets $Q_1 = (g^a)^{s_1} \cdot g^{-\frac{z_1 + r z_2}{\alpha}}$ and $Q_2 = (g^a)^{s_2}$. It computes $R_1 = Q_1 \cdot y^{H_1(ID, Q_1)}$ and $R_2 = Q_2 \cdot y^{H_1(ID, Q_2)}$. It then sets $T_1 = g^{\mu_1} (R_1)^{-\beta_1}$ and $T_2 = g^{\mu_2} (R_2)^{-\beta_2}$. It also sets $H_6(ID, P_1, T_1) = \beta_1$ and $H_6(ID, P_2, T_2) = \beta_2$ (If there already exists tuples in the H_6 list then it re-chooses the appropriate variables and re-computes T_1, T_2). It then chooses $S_3, e_3 \in_R \mathbb{Z}_q^*$ and computes $Q_3 = g^{S_3} y^{-e_3}$. If there exists a tuple $(\langle ID, Q_1, Q_2, Q_3 \rangle, f_3)$ in the H_2 list it re-chooses $(S_3, e_3) \in_R \mathbb{Z}_q^*$ and re-computes Q_3 . It adds $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_3)$ to the H_2 list. It then adds $(\langle ID \rangle, P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2, coin)$ to the public key list, $(\langle ID, coin \rangle, z_1, z_2, s_1, s_2)$ to the private key list (C adds this so that it can generate a valid challenge ciphertext), $(\langle ID, coin \rangle, z_1, z_2)$ to the user key list and returns $(P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2)$ as the public key.

Proposition 2 The public key computed by public key extract for the case $coin = 1$ is identically distributed to the public key computed by the **SetPublicKey** algorithm and additionally it passes the **PublicKeyVerification** test.

Proof The public key generated above is given by:

$$(g^{z_1}, g^{z_2}, (g^a)^{s_1} \cdot g^{-\frac{z_1 + r z_2}{\alpha}}, (g^a)^{s_2}, g^{S_3} y^{-e_3}, S_3, g^{\mu_1} (R_1)^{-\beta_1}, g^{\mu_2} (R_2)^{-\beta_2}, \mu_1, \mu_2)$$

where $z_1, z_2, s_1, s_2, S_3, e_3, \beta_1, \beta_2, \mu_1, \mu_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $R_1 = Q_1 \cdot y^{H_1(ID, Q_1)}$ and $R_2 = Q_2 \cdot y^{H_1(ID, Q_2)}$.

The public key output by **SetPublicKey** is given by:

$$(g^{z_1}, g^{z_2}, g^{s_1}, g^{s_2}, g^{s_3}, S_3, g^{t_1}, g^{t_2}, \mu_1, \mu_2)$$

where $z_1, z_2, s_1, s_2, s_3, t_1, t_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $S_3 = s_3 + x H_2(ID, Q_1, Q_2, Q_3)$, $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$, $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$.

It is easy to see that both the public keys are identically distributed under assumption that H_2 and H_6 behave as a random oracles.

We now show that the public key generated by public key extract passes the **PublicKeyVerification** test.

- $(T_1)(R_1)^{H_6(ID, P_1, T_1)} = g^{\mu_1}(R_1)^{-\beta_1} \cdot (R_1)^{\beta_1} = g^{\mu_1}$. Similarly, one can verify $g^{\mu_2} = (T_2)(R_2)^{H_1(ID, P_2, T_2)}$.
- $(Q_3)(y^{H_2(ID, Q_1, Q_2, Q_3)}) = g^{S_3} y^{-e_3} \cdot y^{e_3} = g^{S_3}$.

Thus, the above generated public key passes the public key verification test. ■

- **Partial Key Extract \mathcal{O}_{pex} :** C maintains a list of partial keys of tuples $(\langle ID, coin \rangle, (Q_1, Q_2, Q_3, S_3), (S_1, S_2))$. When A_I queries the partial key of ID (for which it has not yet replaced the public key), C first searches the partial key list with key value ID . If found and $coin = 0$, then returns the corresponding partial key. If $coin = 1$, it aborts and reports failure. If such a tuple is not found, it queries \mathcal{O}_{pke} with ID_i . It then searches the partial key list for $(\langle ID, coin \rangle, (Q_1, Q_2, Q_3, S_3), (S_1, S_2))$. If $coin = 0$, it outputs $((Q_1, Q_2, Q_3, S_3), (S_1, S_2))$ as the partial public keys and the partial secret key respectively. If $coin = 1$, it aborts and report failure.
- **User Key Extract \mathcal{O}_{uke} :** C maintains a list of user keys of tuples $(\langle ID, coin \rangle, (P_1, P_2), (U_1, U_2))$. When A_I queries the user key of ID (for which it has not yet replaced the public key), C first searches in the user key list for a tuple with key value ID . If found, it returns the corresponding user key. If such a tuple is not found, it queries \mathcal{O}_{pke} with ID_i . It then searches the user key list for $(\langle ID, coin \rangle, (U_1, U_2))$. We note that such a tuple is guaranteed to exist. It returns $(P_1, P_2), (U_1, U_2)$.
- **Re-Key Generation \mathcal{O}_{rkg} :** C maintains a Re-Key list comprising of tuples $(\langle ID_i, ID_j \rangle, rk, V, W, h, \tau)$. When C receives a re-key query from ID_i to ID_j , it first searches for a tuple in the Re-Key list with key value $\langle ID_i, ID_j \rangle$. If found, then the corresponding re-key is given to the adversary. Else, it recovers tuple $(\langle ID_j \rangle, (P_{j,1}, P_{j,2}, Q_{j,1}, Q_{j,2}, Q_{j,3}, S_{j,3}, T_{j,1}, T_{j,2}, \mu_{j,1}, \mu_{j,2}, coin))$. It picks $h \in_R \{0, 1\}^{l_0}$ and $\pi \in_R \{0, 1\}^{l_1}$ and computes $v = H_4(h, \pi)$. It then computes $R_{j,1} = Q_{j,1}(y^{H_1(ID_j, Q_{j,1})})$, $X_1 = P_{j,1}(R_{j,1}^{H(P_{j,1})})$, $X = P_{j,1}(P_{j,2})^{H(P_{j,1})}$ and $\alpha = H(X)$. It then sets $V = (X_1)^v$ and $W = H_3(g^v) \oplus (h||\pi)$. The first component of the re-key are constructed as per the following cases:
 - If $coin_i = 0$ and $PK_i = \hat{PK}_i$ in the CPK list, then C retrieves the private key corresponding to ID_i from private key list and parses it as (U_1, U_2, S_1, S_2) . It computes $rk = \frac{h}{U_{i,1} + H(P_{i,1})U_{i,2} + \alpha(S_{i,1} + H(R_{i,1})S_{i,2})}$. It defines $\tau = 1$ and adds $(\langle ID_i, ID_j \rangle, rk, V, W, h, \tau)$ to the Re-Key list. It outputs (rk, V, W) as the re-key.
 - If $(coin_i = 0$ and $PK_i \neq \hat{PK}_i$ in the CPK list or $coin_i = 1)$ and $(ID_j$ does not belong to the CI list), then it chooses $rk \in_R \mathbb{Z}_q^*$ and defines $\tau = 0$. It adds $(\langle ID_i, ID_j \rangle, rk_{i \rightarrow j}^{<1>}, V, W, h, \tau)$ to the Re-Key list and outputs (rk, V, W) as the re-key.
 - Else, it aborts and reports failure.
- **Re-Encryption Oracle \mathcal{O}_{renc} :** When A_I queries for a re-encryption of a first level ciphertext $C' = (D, E, F, S)$ from ID_i and ID_j , then C checks the validity of C' . If the check fails, it outputs \perp . It then recovers tuple $(\langle ID_i \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{j,1}, T_{j,2}, \mu_{i,1}, \mu_{i,2}, coin)$ from public key list. If $(coin_i = 0$ and $PK_i \neq \hat{PK}_i$ in the CPK list or $coin_i = 1)$ and $ID_j \in CI$ list does not hold, then it queries $\mathcal{O}_{rkg}(ID_i, ID_j)$ to obtain the re-key (rk, V, W) and returns $Re - Encrypt(ID_i, ID_j, params, C, rk_{i \rightarrow j})$. Else, it computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$, $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$, $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$, $\alpha = H(X_1)$ and $Z = (X_1(X_2)^\alpha)$. It also computes $X_1 = P_{j,1}(R_{j,1}^{H(P_{j,1})})$. It then searches for $(\langle m, w \rangle, r)$ in the H_4 list such that $E = Z^r$. If such a tuple is not found, it outputs \perp . It then picks $h \in_R \{0, 1\}^{l_0}$ and $\pi \in_R \{0, 1\}^{l_1}$ and computes $v = H_4(h, \pi)$. It sets $V = (X_1)^v$, $W = H_3(g^v) \oplus (h||\pi)$ and $E' = g^{rh}$. It outputs (E', F, V, W) as the re-encrypted ciphertext.
- **Strong Decrypt-1 Oracle \mathcal{O}_{dec1} :** When A_I queries for the decryption of a first level cipher text C' under the identity ID_i , it recovers $(\langle ID_i \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, \mu_{i,1}, \mu_{i,2}, coin)$ from the public key list. If $coin_i = 0$ and $PK_i = \hat{PK}_i$ in the CPK list, it retrieves $(\langle ID_i, coin \rangle, U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$

from the private key list and sets $SK_i = (U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$. It returns $\text{Decrypt1}(ID_i, params, C', SK_i)$. Otherwise, C checks for the validity of C' . If the check fails, it outputs \perp . It computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$, $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$, $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$, $\alpha = H(X)$ and $Z = (X_1(X_2)^\alpha)$ using the current values in the Public key. It then searches the H_4 list for tuple $(\langle m, w \rangle, r)$ and H_3 list for tuple $(\langle A \rangle, h)$ such that $E = Z^r$, $A = g^r$ and $h \oplus (m||w) = F$. If found, it returns m . Otherwise, it outputs \perp .

- **Strong Decrypt-2 Oracle \mathcal{O}_{dec2} :** When A_I queries for the decryption of a second level ciphertext C'' under the identity ID_i , it recovers $(\langle ID_i \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, \mu_{i,1}, \mu_{i,2}, coin)$ from the public key list. If $coin_i = 0$ and $PK_i = \hat{P}\hat{K}_i$ in the CPK list, it retrieves the tuple $(\langle ID_i, coin \rangle, U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$ from the private key list and sets $SK_i = (U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$. It returns $\text{Decrypt2}(ID_i, params, C'', SK_i)$. Else, it computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$, $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$, $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$, $\alpha = H(X)$ and $Z = (X_1(X_2)^\alpha)$. It then searches in the Re-Key list for a tuple $(\langle ID_j, ID_i \rangle, rk, V, W, h, 0)$. If found, it computes $E = E'^{\frac{1}{rk}}$ and searches H_4 list for tuple $(\langle m, w \rangle, r)$ and H_3 list for tuple $(\langle A \rangle, h)$ such that $E = Z^r$, $A = g^r$, $h \oplus (m||w) = F$. If found, it returns m to adversary. Otherwise, outputs \perp . Else, it computes $X_1 = P_{i,1}(R_{i,1})^{H(P_{i,1})}$ and searches in the H_4 list for tuples $(\langle m, w \rangle, r)$, and $(\langle h, \pi \rangle, v)$, H_3 list for the tuples $(\langle A \rangle, l)$, $(\langle R \rangle, k)$ such that $X_1^v = V$, $R = g^v$, $k \oplus (h||\pi) = W$, $E' = g^{rh}$, $A = g^r$, $l \oplus (m||w) = F$. If found, it returns m to the adversary. Otherwise, it return \perp .
- **Public Key Replacement Oracle \mathcal{O}_{pkr} :** When A_I wants to replace the public key corresponding to an identity ID_i with a new value $\hat{P}\hat{K}_i$, C checks validity of $\hat{P}\hat{K}_i$. If the check fails output \perp . Otherwise, update the CPK list with the new value of the public key corresponding to the identity ID_i .

7.1.2 Challenge

A_I outputs two messages M_0, M_1 and an identity ID_i^* on which it wishes to be challenged. It is required that the adversary has not queried the partial key corresponding to the identity ID_i^* . C recovers the tuple $(\langle ID_i^* \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, \mu_{i,1}, \mu_{i,2}, coin)$ from the public key list. If $coin = 0$ or $PK_i \neq \hat{P}\hat{K}_i$, then C aborts. Else, it tosses a coin and chooses $p \in \{0, 1\}$ uniformly at random. It computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i^*, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i^*, Q_{i,2})})$. It retrieves s_1 and s_2 from the private key list. It sets $K = (s_1 + H_1(ID_i^*, Q_{i,1}))\alpha + H(R_{i,1})\alpha(s_2 + H_1(ID_i^*, Q_{i,2}))$. It chooses $e^*, f^* \in_R \mathbb{Z}_q^*$ and computes $D = (g^a)^{f^*K}(g^b)^{-e^*K}$, $E = (g^b)^K$. It picks $F^* \in_R \{0, 1\}^{l_0+l_1}$ and implicitly defines $H_5(D, E, F^*) = e^*$. It then, pick $w^* \in_R \{0, 1\}^{l_1}$ and implicitly defines $H_4(M_p, w^*) = b/a$ and $H_3(g^{b/a}) = F^* \oplus (M_p||w^*)$. It returns $C^* = (D, E, F^*, f^*)$.

Proposition 3 The challenge ciphertext is identically distributed to the real ciphertext output by Encrypt algorithm.

Proof Let us define $u = f^* - e^*(b/a)$ and $r = b/a$. We can easily verify that $Z = X(Y)^\alpha = (g^a)^K$.

We note that,

$$E = (g^b)^K = ((g^a)^K)^{b/a} = (Z)^r$$

$$D = (g^a)^{f^*K}(g^b)^{-e^*K} = ((g^a)^K)^{f^* - e^*(b/a)} = (Z)^u$$

$$S = f^* = f^* - e^*(b/a) + e^*(b/a) = u + rH_5(D, E, F^*)$$

Hence, we can conclude that (D, E, F^*, f^*) is a valid ciphertext. ■

7.1.3 Phase-2

The adversary continues to query any of the above mentioned oracles with the restrictions defined in the security model and the challenger responds as in phase 1.

7.1.4 Guess

Finally, the adversary outputs a guess p' of p . If $p = p'$, the challenger chooses a random tuple $(\langle A \rangle, h)$ in the H_3 list and outputs A as the solution to the M-CDH problem.

We now analyse the probability that the challenger solves the M-CDH problem.

7.1.5 Analysis

We first analyze the simulation of the random oracles. It is clear that the simulations of H, H_1, H_2, H_6 are perfect. Let $Ask_{H_3}^*$ denote the event that $(g^{b/a})$ was queried to H_3 . Let $Ask_{H_4}^*$ be the event that adversary queried H_4 with (M_p, w^*) . Let $Ask_{H_5}^*$ be the event that the adversary queried H_5 with (D, E, F^*) before the challenge phase. As long as $Ask_{H_3}^*, Ask_{H_4}^*, Ask_{H_5}^*$ did not occur the simulations of these oracles are perfect.

It is easy to see that challenger's response to the public key extract queries of the adversary are correct. Let $Abort$ be the event that the challenger aborts before the guess phase. The challenger can abort the game in Secret key extract or partial key extract or in re-key extract or in challenge phase.

The simulation of \mathcal{O}_{rkg} is perfect except for the case where $(coin_i = 0$ and $PK_i \neq \hat{PK}_i$ in the CPK list or $coin_i = 1$) and ID_j does not belong to the CI list, in which $rk_{i \rightarrow j}^{<1>}$ is randomly chosen. If $Abort$ does not happen then we show (like in [CWYD10]) that this is computationally indistinguishable from the real world. First, we note that (V, W) is an encryption of h under the hashed-ElGamal scheme and the adversary does not possess SK_j . So, if the adversary can distinguish between the correct re-key and a randomly chosen one with non-negligible advantage then he can in fact distinguish whether h or h' was hidden in the cipher text (V, W) . Thus, this adversary can be used to break the security of hashed ElGamal system based on CDH assumption. Hence, if $Abort$ does not happen then the simulation of \mathcal{O}_{rkg} is perfect.

Next, we analyse the simulation of re-encryption queries. The simulation is perfect as long as the adversary can submit valid ciphertexts without querying the hash function H_4 . We denote this event be $ReErr$.

The simulation of \mathcal{O}_{dec1} and \mathcal{O}_{dec2} is perfect as long as we don't reject valid ciphertexts. This can happen when the adversary can query the oracles with valid cipher text without querying the hash functions H_3 and H_4 .

We now estimate the probabilities associated with each of the above described events.

$$Pr[Ask_{H_5}^*] \leq \frac{q_{H_5}}{2^{l_0+l_1}}, \text{ as } F^* \text{ is chosen uniformly at random.}$$

$$Pr[\neg Abort] \geq \theta^{q_{pex}+q_{rkg}}(1-\theta)(1-\nu)^{q_{rkg}+1} \text{ which is maximized at } \theta^* = \frac{q_{pex} + q_{rkg}}{1 + q_{pex} + q_{rkg}}$$

Using θ^* , we get

$$Pr[\neg Abort] \geq \frac{(1-\nu)^{q_{rkg}+1}}{e(1 + q_{pex} + q_{rkg})}$$

$$Pr[ReErr] \leq \frac{q_{renc}}{q}, \text{ due to randomness of } H_4 \text{'s output}$$

We now estimate the probability that a valid ciphertext gets rejected by the Strong Decrypt-1 or Strong Decrypt-2 oracles. Let $Valid$ denote the event that the ciphertext is valid. Let Ask_{H_4} denote the event that (m, w) has been queried to H_4 and Ask_{H_3} denote the event that (g^r) has been queried to H_3 .

$$Pr[Valid | \neg Ask_{H_3}] = Pr[Valid \wedge Ask_{H_4} | \neg Ask_{H_3}] + Pr[Valid \wedge \neg Ask_{H_4} | \neg Ask_{H_3}]$$

$$\begin{aligned}
&\leq Pr[Valid \wedge Ask_{H_4} | \neg Ask_{H_3}] + Pr[Valid | \neg Ask_{H_4} \wedge \neg Ask_{H_3}] \\
&= \frac{Pr[Valid \wedge Ask_{H_4} \wedge \neg Ask_{H_3}]}{Pr[\neg Ask_{H_3}]} + Pr[Valid | \neg Ask_{H_4} \wedge \neg Ask_{H_3}] \\
&\leq \frac{Pr[Ask_{H_4}]}{Pr[\neg Ask_{H_3}]} + Pr[Valid | \neg Ask_{H_4} \wedge \neg Ask_{H_3}] \\
&\leq \frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + 1/q
\end{aligned}$$

Similarly, we can show that

$$Pr[Valid | \neg Ask_{H_4}] \leq \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 1/q$$

Hence, we have

$$\begin{aligned}
Pr[Valid | \neg Ask_{H_3} \vee \neg Ask_{H_4}] &\leq Pr[Valid | \neg Ask_{H_3}] + Pr[Valid | \neg Ask_{H_4}] \\
&\leq \frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q
\end{aligned}$$

Then, $DErr$ is the event that $Valid | \neg(Ask_{H_3} \wedge Ask_{H_4})$ happens at least once during the entire simulation. Hence,

$$Pr[DErr] \leq q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q \right)$$

Let $Query$ be the event $Ask_{H_3^*} \cup Ask_{H_4^*} \cup Ask_{H_5^*} \cup ReErr \cup DErr | \neg Abort$. Clearly, if $Query$ does not happen during the simulation then due to randomness of H_3 's output the adversary does not have any advantage which is greater than $1/2$ in guessing p . Hence, $Pr[p' = p | \neg Query] = 1/2$

$$Pr[p' = p] = Pr[p' = p | \neg Query]Pr[\neg Query] + Pr[p' = p | Query]Pr[Query]$$

$$\leq 1/2Pr[\neg Query] + Pr[Query] \leq 1/2 + Pr[Query]$$

$$Pr[p' = p] \geq Pr[p' = p | \neg Query]Pr[\neg Query] \geq 1/2 - 1/2Pr[Query]$$

We have

$$\begin{aligned}
\varepsilon &= 2|Pr[p' = p] - 1/2| \leq Pr[Query] \\
&\leq \frac{Pr[Ask_{H_3^*}] + Pr[(Ask_{H_4^*}) + Pr[Ask_{H_5^*}] + Pr[ReErr] + Pr[DErr]]}{Pr[\neg Abort]}
\end{aligned}$$

Thus,

$$\begin{aligned}
Pr[Ask_{H_3^*}] &\geq \varepsilon(Pr[\neg Abort]) - (Pr[(Ask_{H_4^*}) + Pr[Ask_{H_5^*}] + Pr[ReErr] + Pr[DErr]]) \\
&\geq \frac{\varepsilon(1 - \nu)^{q_{rk_g}+1}}{e(1 + q_{pex} + q_{rk_g})} - \left(\frac{q_{H_4}}{2^{l_0+l_1}} + \frac{q_{H_5}}{2^{l_0+l_1}} + \frac{q_{renc}}{q} + q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q \right) \right)
\end{aligned}$$

If $Ask_{H_3^*}$ occurs then C will be able to solve the M-CDH problem with advantage ε' given by

$$\begin{aligned}
\varepsilon' &\geq (1/q_{H_3})Pr[Ask_{H_3^*}] \\
&\geq (1/q_{H_3}) \left(\frac{\varepsilon(1 - \nu)^{q_{rk_g}+1}}{e(1 + q_{pex} + q_{rk_g})} - \left(\frac{q_{H_4}}{2^{l_0+l_1}} + \frac{q_{H_5}}{2^{l_0+l_1}} + \frac{q_{renc}}{q} + q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q \right) \right) \right)
\end{aligned}$$

Let $T_1 = q_H + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{pke} + q_{uke} + q_{pex} + q_{rk_g} + q_{renc} + q_{dec1} + q_{dec2} + q_{pkr}$ and $T_2 = 10q_{pke} + 10q_{uke} + 10q_{pex} + 5q_{rk_g} + 8q_{renc} + 7q_{dec1} + 9q_{dec2}$. From the construction of C we can bound its running time t' by,

$$t' \leq t + (T_1)O(1) + (T_2)t_{exp}$$

7.2 Security against Type-II adversary

We now show that our scheme is secure against a Type-II adversary.

Theorem 2 Suppose $H, H_1, H_2, H_3, H_4, H_5$ are random oracles and there exists a $(t, \varepsilon, q_{pke}, q_{uke}, q_{pex}, q_{rkg}, q_{renc}, q_{dec1}, q_{dec2}, q_{pkr})$ IND-CLPRE-CCA adversary A_{II} against our scheme making at most q_H queries to H and at most q_{H_i} queries to random oracles H_i where $1 \leq i \leq 5$, then there exists a PPT algorithm C which solves the M-CDH problem with advantage

$$\varepsilon' \geq (1/q_{H_3}) \left(\frac{\varepsilon(1-\nu)^{q_{rkg}}}{e(1+q_{uke}+q_{rkg})} - \left(\frac{q_{H_4}}{2^{l_0+l_1}} + \frac{q_{H_5}}{2^{l_0+l_1}} + \frac{q_{renc}}{q} + q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1-q_{H_3}/q} + \frac{q_{H_3}/q}{1-q_{H_4}/2^{l_0+l_1}} + 2/q \right) \right) \right)$$

where ν is the advantage an attacker may have over the EUF-CMA security game of the Schnorr signature scheme and which runs in time t'

$$t' \leq t + (T_1)O(1) + (T_2)t_{exp}$$

where $T_1 = q_H + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{pke} + q_{uke} + q_{rkg} + q_{renc} + q_{dec1} + q_{dec2} + q_{pkr}$, $T_2 = 10q_{pke} + 10q_{uke} + 10q_{pex} + 5q_{rkg} + 8q_{renc} + 7q_{dec1} + 9q_{dec2}$ and t_{exp} is the time taken for exponentiation in group G .

Proof. We describe the challenger C which interacts with a Type-II adversary A_{II} and solves the M-CDH problem. C is given an instance of the M-CDH problem (g, g^a, g^b) . It chooses $x \in_R \mathbb{Z}_q^*$ and sets $y = g^x$. It gives x as the master secret key to A_{II} . C maintains a list CI of corrupted identities which is initially empty. The challenger also maintains a list of current public keys called as CPK list consisting of tuples of the form (ID_i, PK_i, \hat{PK}_i) where PK_i denotes the full public key returned by SetPublicKey algorithm, and \hat{PK}_i denotes the current public key. The list is initialized with $PK_i = \hat{PK}_i$ for all identities ID_i . A_{II} requests the access to several oracles which are to be simulated by C . We now describe the confidentiality game between the adversary A_{II} and the challenger C .

7.2.1 Phase-1

In phase-1 of the game, A_{II} requests the access to several oracles which are simulated by the challenger. We now describe how C answers A_{II} 's oracle queries.

- **H queries:** C maintains a H -list of tuples $(\langle A \rangle, \alpha)$. On receiving a query, C searches H -list for $(\langle A \rangle, \alpha)$. If found, outputs α . Else, chooses $\alpha \in_R \mathbb{Z}_q^*$ and returns α . C adds $(\langle A \rangle, \alpha)$ to the H -list.
- **H_1 queries:** C maintains a H_1 -list of tuples $(\langle ID, Q \rangle, e_0)$. On receiving a query, C searches H_1 -list for $(\langle ID, Q \rangle, e_0)$. If found, outputs e_0 . Else, chooses $e_0 \in_R \mathbb{Z}_q^*$ and returns e_0 . C adds $(\langle ID, Q \rangle, e_0)$ to the H_1 -list.
- **H_2 queries:** C maintains a H_2 -list of tuples $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_1)$. On receiving a query, C searches H_2 -list for $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_1)$. If found, outputs e_1 . Else, chooses $e_1 \in_R \mathbb{Z}_q^*$ and returns e_1 . C adds $(\langle ID, Q_1, Q_2, Q_3 \rangle, e_1)$ to the H_2 -list.
- **H_3 queries:** C maintains a H_3 -list of tuples $(\langle A \rangle, h)$. On receiving a query, C searches H_3 -list for $(\langle A \rangle, h)$. If found, outputs h . Else, chooses $h \in_R \{0, 1\}^{l_0+l_1}$ and returns h . It adds $(\langle A \rangle, h)$ to the H_3 -list.
- **H_4 queries:** C maintains a H_4 -list of tuples $(\langle m, w \rangle, r)$. On receiving a query, C searches H_4 -list for $(\langle m, w \rangle, r)$. If found, outputs r . Else, chooses $r \in_R \mathbb{Z}_{q^*}$ and returns r . It adds $(\langle m, w \rangle, r)$ to the H_4 -list.
- **H_5 queries:** C maintains a H_5 -list of tuples $(\langle A, B, C \rangle, p)$. On receiving a query, C searches H_5 -list for $(\langle A, B, C \rangle, p)$. If found, outputs p . Else, chooses $p \in_R \mathbb{Z}_{q^*}$ and gives p as output. It adds $(\langle A, B, C \rangle, p)$ to the H_5 -list.

- **H_6 queries:** C maintains a H_6 -list of tuples $(\langle ID, A, B \rangle, p)$. On receiving a query, C searches H_6 -list for $(\langle ID, A, B \rangle, p)$. If found, outputs p . Else, chooses $p \in_R \mathbb{Z}_q^*$ and gives p as output. It adds $(\langle ID, A, B \rangle, p)$ to the H_5 -list.
- **Compute Partial Key:** A_{II} computes the partial private key $(S_{i,1}, S_{i,2})$ and the partial public key $(Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3})$ for any ID_i of its choice. C maintains a list of partial keys computed by A in a partial key list $(\langle ID_i \rangle, (Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}), (S_{i,1}, S_{i,2}))$
- **Public Key Extract \mathcal{O}_{pke} :** C maintains a public key list of tuples $(\langle ID \rangle, P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2, coin)$. On receiving a query on ID , C searches the public key list for key value ID . If found, then it outputs the corresponding the public key. Else, it tosses a biased coin and picks $coin \in \{0, 1\}$ such that $Pr[coin = 0] = \theta$ where θ is to be determined later.

If **coin=0**, C chooses $(U_1, U_2) = (z_1, z_2) \in_R \mathbb{Z}_q^*$ and computes $(P_1, P_2) = (g^{z_1}, g^{z_2})$. It retrieves the partial keys corresponding to ID from the partial key list. It then chooses $t_1, t_2 \in_R \mathbb{Z}_q^*$ and computes $T_1 = g^{t_1}$ and $T_2 = g^{t_2}$. It then computes $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$ and $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$. It then adds $(\langle ID \rangle, P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2, coin)$ to the public key list, $(\langle ID, coin \rangle, U_1, U_2, S_1, S_2)$ to the private key list and $(\langle ID, coin \rangle, (P_1, P_2), (U_1, U_2))$ to the user key list. It returns $(P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2)$ as the public key.

Proposition 4 The public key computed by public key extract for the case $coin = 0$ is identically distributed to the public key computed by the **SetPublicKey** algorithm and additionally it passes the **PublicKeyVerification** test.

Proof The public key generated above is given by:

$$(g^{z_1}, g^{z_2}, g^{s_1}, g^{s_2}, g^{s_3}, S_3, g^{t_1}, g^{t_2}, \mu_1, \mu_2)$$

where $z_1, z_2, s_1, s_2, s_3, t_1, t_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $S_3 = s_3 + xH_2(ID, Q_1, Q_2, Q_3)$, $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$, $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$.

The public key output by **SetPublicKey** is given by:

$$(g^{z_1}, g^{z_2}, g^{s_1}, g^{s_2}, g^{s_3}, S_3, g^{t_1}, g^{t_2}, \mu_1, \mu_2)$$

where $z_1, z_2, s_1, s_2, s_3, t_1, t_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $S_3 = s_3 + xH_2(ID, Q_1, Q_2, Q_3)$, $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$, $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$.

It is easy to see that both the public keys are identically distributed.

Since the above public key is generated in exact same way as that of a **SetPublicKey** algorithm, it is easy to see that it passes the public key verification test. \blacksquare

If **coin=1**, C retrieves the partial key corresponding to ID from the partial key list. It computes $R_1 = Q_1 \cdot y^{H_1(ID, Q_1)}$ and $R_2 = Q_2 \cdot y^{H_1(ID, Q_2)}$. Let $r = H(R_1)$. It then chooses $z_1, z_2, \alpha \in_R \mathbb{Z}_q^*$ and computes $P_1 = (g^a)^{z_1} g^{-\alpha(S_1 + S_2 H(R_1))}$ and $P_2 = (g^a)^{z_2}$. It computes $X = P_1(P_2)^{H(P_1)}$ and sets $H(X) = \alpha$. It then chooses $t_1, t_2 \in_R \mathbb{Z}_q^*$ and computes $T_1 = g^{t_1}$ and $T_2 = g^{t_2}$. It then computes $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$ and $\mu_2 = t_2 + S_2 H_1(ID, P_2, T_2)$. It then adds $(\langle ID \rangle, P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2, coin)$ to the public key list, $(\langle ID, coin \rangle, z_1, z_2, S_1, S_2)$ to the private key list, $(\langle ID, coin \rangle, (P_1, P_2), (z_1, z_2))$ to the user key list. It returns $(P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2)$ as the public key.

Proposition 5 The public key computed by public key extract for the case $coin = 1$ is identically distributed to the public key computed by the **SetPublicKey** algorithm and additionally it passes the **PublicKeyVerification** test.

Proof The public key generated above is given by:

$$((g^a)^{z_1} g^{-\alpha(S_1+S_2H(R_1))}, (g^a)^{z_2}, g^{s_1}, g^{s_2}, g^{s_3}, S_3, g^{t_1}, g^{t_2}, \mu_1, \mu_2)$$

where $z_1, z_2, s_1, s_2, s_3, t_1, t_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $S_3 = s_3 + xH_2(ID, Q_1, Q_2, Q_3)$, $\mu_1 = t_1 + S_1H_6(ID, P_1, T_1)$, $\mu_2 = t_2 + S_2H_1(ID, P_2, T_2)$.

The public key output by `SetPublicKey` is given by:

$$(g^{z_1}, g^{z_2}, g^{s_1}, g^{s_2}, g^{s_3}, S_3, g^{t_1}, g^{t_2}, \mu_1, \mu_2)$$

where $z_1, z_2, s_1, s_2, s_3, t_1, t_2$ are chosen uniformly and independently at random from \mathbb{Z}_q^* and $S_3 = s_3 + xH_2(ID, Q_1, Q_2, Q_3)$, $\mu_1 = t_1 + S_1H_6(ID, P_1, T_1)$, $\mu_2 = t_2 + S_2H_1(ID, P_2, T_2)$.

It is easy to see that both the public keys are identically distributed.

We now show that the public key generated by public key extract passes the `PublicKeyVerification` test.

- $(T_1)(R_1)^{H_6(ID, P_1, T_1)} = g^{t_1 + S_1 \cdot H_6(ID, P_1, T_1)} = g^{\mu_1}$. Similarly, one can verify $g^{\mu_2} = (T_2)(R_2)^{H_1(ID, P_2, T_2)}$.
- $(Q_3)(y^{H_2(ID, Q_1, Q_2, Q_3)}) = g^{s_3 + x \cdot H_2(ID, Q_1, Q_2, Q_3)} = g^{S_3}$.

Thus, the above generated public key passes the public key verification test. ■

- **User Key Extract** \mathcal{O}_{uke} : C maintains a list of user keys of tuples $(\langle ID, coin \rangle, (P_1, P_2), (U_1, U_2))$. On receiving a query for an identity ID (for which A_I has not replaced the public key), C first searches for a tuple with key value ID . If found, and $coin = 0$ then it returns the corresponding user key. If $coin = 1$ then C aborts and reports failure. If such a tuple is not found then, it queries \mathcal{O}_{pke} with ID_i . It searches the private key list for a tuple of the form $(\langle ID, coin \rangle, (P_1, P_2), (U_1, U_2))$. If $coin = 0$, it outputs $(P_1, P_2), (U_1, U_2)$ and adds ID_i to the list CI . Else, it aborts the game and report failure.
- **Re-Key Generation** \mathcal{O}_{rkg} : C maintains a Re-Key list comprising of tuples $(\langle ID_i, ID_j \rangle, rk_{i \rightarrow j}^{<1>}, V, W, h, \tau)$. When C receives a re-key query from ID_i to ID_j , it first searches for a tuple in the Re-Key list with key value $\langle ID_i, ID_j \rangle$. If found, then the corresponding re-key is given to the adversary. Else, it recovers tuple $(\langle ID_j \rangle, (P_{j,1}, P_{j,2}, Q_{j,1}, Q_{j,2}, Q_{j,3}, S_{j,3}, T_{j,1}, T_{j,2}, \mu_{j,1}, \mu_{j,2}, coin))$. It picks $h \in_R \{0, 1\}^{l_0}$ and $\pi \in_R \{0, 1\}^{l_1}$ and computes $v = H_4(h, \pi)$. It then computes $R_{j,1} = Q_{j,1}(y^{H_1(ID_j, Q_{j,1})})$, $X_1 = P_{j,1}(R_{j,1}^{H(P_{j,1})})$, $X = P_{j,1}(P_{j,2})^{H(P_{j,1})}$ and $\alpha = H(X)$. It then sets $V = (X_1)^v$ and $W = H_3(g^v) \oplus (h || \pi)$. The first component of the re-key are constructed as per the following cases:
 - If $coin_i = 0$ and $PK_i = \hat{PK}_i$ in the CPK list, then C retrieves the private key corresponding to ID_i from private key list and parses it as (U_1, U_2, S_1, S_2) . It computes $rk = \frac{h}{\overline{U_{i,1} + H(P_{i,1})U_{i,2} + \alpha(S_{i,1} + H(R_{i,1})S_{i,2})}}$. It defines $\tau = 1$ and adds $(\langle ID_i, ID_j \rangle, rk, V, W, h, \tau)$ to the Re-Key list. It outputs (rk, V, W) as the re-key.
 - If $(coin_i = 0$ and $PK_i \neq \hat{PK}_i$ in the CPK list or $coin_i = 1)$ and $(ID_j$ does not belong to the CI list), then it chooses $rk \in_R \mathbb{Z}_q^*$ and defines $\tau = 0$. It adds $(\langle ID_i, ID_j \rangle, rk_{i \rightarrow j}^{<1>}, V, W, h, \tau)$ to the Re-Key list and outputs (rk, V, W) as the re-key.
 - Else, it aborts and reports failure.
- **Re-Encryption Oracle** \mathcal{O}_{renc} : When A_{II} queries for a re-encryption of a first level ciphertext $C' = (D, E, F, S)$ from ID_i and ID_j , then C checks the validity of C' . If the check fails, it outputs \perp . It then recovers tuple $(\langle ID_i \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{j,1}, T_{j,2}, \mu_{i,1}, \mu_{i,2}, coin)$ from public key list. If $(coin_i = 0$ and $PK_i \neq \hat{PK}_i$ in the CPK list or $coin_i = 1)$ and $ID_j \in CI$ list does not hold, then it queries $\mathcal{O}_{rkg}(ID_i, ID_j)$ to obtain the re-key (rk, V, W) and returns $Re - Encrypt(ID_i, ID_j, params, C, rk_{i \rightarrow j})$. Else, it computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$, $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$, $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$, $\alpha = H(X_1)$ and $Z = (X_1(X_2)^\alpha)$. It also computes

$X_1 = P_{j,1}(R_{j,1}^{H(P_{j,1})})$. It then searches for $(\langle m, w \rangle, r)$ in the H_4 list such that $E = Z^r$. If such a tuple is not found, it outputs \perp . It then picks $h \in_R \{0, 1\}^{l_0}$ and $\pi \in_R \{0, 1\}^{l_1}$ and computes $v = H_4(h, \pi)$. It sets $V = (X_1)^v$, $W = H_3(g^v) \oplus (h||\pi)$ and $E' = g^{r^h}$. It outputs (E', F, V, W) as the re-encrypted ciphertext.

- **Strong Decrypt-1 Oracle \mathcal{O}_{dec1} :** When A_{II} queries for the decryption of a first level cipher text C' under the identity ID_i , it recovers $(\langle ID_i \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, \mu_{i,1}, \mu_{i,2}, coin)$ from the public key list. If $coin_i = 0$ and $PK_i = \hat{P}\hat{K}_i$ in the CPK list, it retrieves $(\langle ID_i, coin \rangle, U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$ from the private key list and sets $SK_i = (U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$. It returns $\text{Decrypt1}(ID_i, params, C', SK_i)$. Otherwise, C checks for the validity of C' . If the check fails, it outputs \perp . It computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$, $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$, $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$, $\alpha = H(X)$ and $Z = (X_1(X_2)^\alpha)$. It then searches the H_4 list for tuple $(\langle m, w \rangle, r)$ and H_3 list for tuple $(\langle A \rangle, h)$ such that $E = Z^r$, $A = g^r$ and $h \oplus (m||w) = F$. If found, it returns m . Otherwise, it outputs \perp .
- **Strong Decrypt-2 Oracle \mathcal{O}_{dec2} :** When A_{II} queries for the decryption of a second level ciphertext C'' under the identity ID_i , it recovers $(\langle ID_i \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, \mu_{i,1}, \mu_{i,2}, coin)$ from the public key list. If $coin_i = 0$ and $PK_i = \hat{P}\hat{K}_i$ in the CPK list, it retrieves the tuple $(\langle ID_i, coin \rangle, U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$ from the private key list and sets $SK_i = (U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$. It returns $\text{Decrypt2}(ID_i, params, C'', SK_i)$. Else, it computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$, $X = P_{i,1}(P_{i,2})^{H(P_{i,1})}$, $Y = R_{i,1}(R_{i,2})^{H(R_{i,1})}$, $\alpha = H(X)$ and $Z = (X_1(X_2)^\alpha)$. It then searches in the Re-Key list for a tuple $(\langle ID_j, ID_i \rangle, rk, V, W, h, 0)$. If found, it computes $E = E'^{\frac{1}{rk}}$ and searches H_4 list for tuple $(\langle m, w \rangle, r)$ and H_3 list for tuple $(\langle A \rangle, h)$ such that $E = Z^r$, $A = g^r$, $h \oplus (m||w) = F$. If found, it returns m to adversary. Otherwise, outputs \perp . Else, it computes $X_1 = P_{i,1}(R_{i,1})^{H(P_{i,1})}$ and searches in the H_4 list for tuples $(\langle m, w \rangle, r)$, and $(\langle h, \pi \rangle, v)$, H_3 list for the tuples $(\langle A \rangle, l)$, $(\langle R \rangle, k)$ such that $X_1^v = V$, $R = g^v$, $k \oplus (h||\pi) = W$, $E' = g^{r^h}$, $A = g^r$, $l \oplus (m||w) = F$. If found, it returns m to the adversary. Otherwise, it return \perp .
- **Public Key Replacement Oracle \mathcal{O}_{pkr} :** When A_{II} wants to replace the public key corresponding to an identity ID_i with a new value $\hat{P}\hat{K}_i$, C checks validity of $\hat{P}\hat{K}_i$. If the check fails output \perp . Otherwise, update the CPK list with the new value of the public key corresponding to the identity ID_i .

7.2.2 Challenge

A_I outputs two messages M_0, M_1 and an identity ID_i^* on which it wishes to be challenged. It is required that the adversary has not queried the partial key and the secret key corresponding to the identity ID_i^* or replaced the public key corresponding to ID_i^* . C recovers the tuple $(\langle ID_i^* \rangle, P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, \mu_{i,1}, \mu_{i,2}, coin)$ from the public key list. If $coin = 0$, then C aborts. Else, it tosses a coin and chooses $p \in \{0, 1\}$ uniformly at random. It computes $R_{i,1} = Q_{i,1}(y^{H_1(ID_i^*, Q_{i,1})})$, $R_{i,2} = Q_{i,2}(y^{H_1(ID_i^*, Q_{i,2})})$. It retrieves z_1 and z_2 from the private key list. It sets $K = z_1 + H(P_{i,1})z_2$. It chooses $e^*, f^* \in_R \mathbb{Z}_q^*$ and computes $D = (g^a)^{f^*K} (g^b)^{-e^*K}$, $E = (g^b)^K$. It picks $F^* \in_R \{0, 1\}^{l_0+l_1}$ and implicitly defines $H_5(D, E, F^*) = e^*$. It then, pick $w^* \in_R \{0, 1\}^{l_1}$ and implicitly defines $H_4(M_p, w^*) = b/a$ and $H_3(g^{b/a}) = F^* \oplus (M_p||w^*)$. It returns $C^* = (D, E, F^*, w^*)$.

Proposition 6 The challenge ciphertext generated as above is identically distributed to the real ciphertext generated by Encrypt algorithm.

Proof Let us define $u = f^* - e^*(b/a)$ and $r = b/a$. It is easy to see that $Z = X(Y)^\alpha = (g^a)^K$. Now,

$$E = (g^b)^K = ((g^a)^K)^{b/a} = (Z)^r$$

$$D = (g^a)^{f^*K} (g^b)^{-e^*K} = ((g^a)^K)^{f^* - e^*(b/a)} = (Z)^u$$

$$S = f^* = f^* - e^*(b/a) + e^*(b/a) = u + rH_5(D, E, F^*)$$

Hence, we can conclude that (D, E, F^*, f^*) is a valid ciphertext. ■

7.2.3 Phase-2

The adversary continues to query any of the above mentioned oracles with the restrictions defined in the security model and the challenger responds as in phase 1.

7.2.4 Guess

Finally, the adversary outputs a guess p' of p . If $p = p'$, the challenger chooses a random tuple $(\langle A \rangle, h)$ in the H_3 list and outputs A as the solution to the M-CDH problem.

We now analyse the success probability of the adversary in solving the M-CDH problem.

Analysis

We first analyze the simulation of the random oracles. It is clear that the simulations of H, H_1, H_2, H_6 are perfect. Let $Ask_{H_3}^*$ denote the event that $(g^{b/a})$ was queried to H_3 . Let $Ask_{H_4}^*$ be the event that adversary queried H_4 with (M_p, w^*) . Let $Ask_{H_5}^*$ be the event that the adversary queried H_5 with (D, E, F^*) before the challenge phase. As long as $Ask_{H_3}^*, Ask_{H_4}^*, Ask_{H_5}^*$ did not occur the simulations of these oracles are perfect.

Let $Abort$ be the event that the challenger aborts before the guess phase. The challenger can abort the game in Secret key extract or partial key extract or in re-key extract or in challenge phase.

The simulation of \mathcal{O}_{rk_g} is perfect except for the case where $(coin_i = 0$ and $PK_i \neq \tilde{PK}_i$ in the CPK list or $coin_i = 1)$ and ID_j does not belong to the CI list, in which rk is randomly chosen. If $Abort$ does not happen exactly like in the previous case we can argue that the adversary cannot distinguish between the cases when rk is randomly chosen from the one in which it is properly generated.

Let $REErr$ be the event that A_{II} can submit valid ciphertexts for re-encryption without querying H_4 .

We now estimate the probabilities associated with each of the above described events.

$$Pr[Ask_{H_5}^*] \leq \frac{q_{H_5}}{2^{l_0+l_1}}, \text{ as } F^* \text{ is chosen uniformly at random.}$$

$$Pr[\neg Abort] \geq \theta^{q_{uke}+q_{rkg}}(1-\theta)(1-\nu)^{q_{rkg}} \text{ which is maximized at } \theta^* = \frac{q_{uke} + q_{rkg}}{1 + q_{uke} + q_{rkg}}$$

Using θ^* , we get

$$Pr[\neg Abort] \geq \frac{(1-\nu)^{q_{rkg}}}{e(1 + q_{uke} + q_{rkg})}$$

Exactly as in the previous case we can prove the following inequalities.

$$Pr[REErr] \leq \frac{q_{renc}}{q}, \text{ due to randomness of } H_4 \text{'s output}$$

Let $Valid$ denote the event that the ciphertext is valid. Let Ask_{H_4} denote the event that (m, w) has been queried to H_4 and Ask_{H_3} denote the event that (g^r) has been queried to H_3 .

$$Pr[Valid | \neg Ask_{H_3}] \leq \frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + 1/q$$

Similarly, we can show that

$$Pr[Valid|\neg Ask_{H_4}] \leq \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 1/q$$

Hence, we have

$$Pr[Valid|\neg Ask_{H_3} \vee \neg Ask_{H_4}] \leq \frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q$$

Then, $DErr$ is the event that $Valid|\neg(Ask_{H_3} \wedge Ask_{H_4})$ happens at least once during the entire simulation. Hence,

$$Pr[DErr] \leq q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q \right)$$

Let $Query$ be the event $Ask_{H_3^*} \cup Ask_{H_4^*} \cup Ask_{H_5^*} \cup ReErr \cup DErr|\neg Abort$. Clearly, if $Query$ does not happen during the simulation then due to randomness of H_3 's output the adversary does not have any advantage which is greater than $1/2$ in guessing p . Hence, $Pr[p' = p|\neg Query] = 1/2$

$$Pr[p' = p] = Pr[p' = p|\neg Query]Pr[\neg Query] + Pr[p' = p|Query]Pr[Query]$$

$$\leq 1/2Pr[\neg Query] + Pr[Query] \leq 1/2 + Pr[Query]$$

$$Pr[p' = p] \geq Pr[p' = p|\neg Query]Pr[\neg Query] \geq 1/2 - 1/2Pr[Query]$$

We have

$$\begin{aligned} \varepsilon &= 2|Pr[p' = p] - 1/2| \leq Pr[Query] \\ &\leq \frac{Pr[Ask_{H_3^*}] + Pr[(Ask_{H_4^*}) + Pr[Ask_{H_5^*}] + Pr[ReErr] + Pr[DErr]}{Pr[\neg Abort]} \end{aligned}$$

Thus,

$$\begin{aligned} Pr[Ask_{H_3^*}] &\geq \varepsilon(Pr[\neg Abort]) - (Pr[(Ask_{H_4^*}) + Pr[Ask_{H_5^*}] + Pr[ReErr] + Pr[DErr]) \\ &\geq \frac{\varepsilon(1 - \nu)^{q_{rkg}}}{e(1 + q_{uke} + q_{rkg})} - \left(\frac{q_{H_4}}{2^{l_0+l_1}} + \frac{q_{H_5}}{2^{l_0+l_1}} + \frac{q_{renc}}{q} + q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q \right) \right) \end{aligned}$$

If $Ask_{H_3^*}$ occurs then C will be able to solve the M-CDH problem with advantage ε' given by

$$\begin{aligned} \varepsilon' &\geq (1/q_{H_3})Pr[Ask_{H_3^*}] \\ &\geq (1/q_{H_3}) \left(\frac{\varepsilon(1 - \nu)^{q_{rkg}}}{e(1 + q_{uke} + q_{rkg})} - \left(\frac{q_{H_4}}{2^{l_0+l_1}} + \frac{q_{H_5}}{2^{l_0+l_1}} + \frac{q_{renc}}{q} + q_d \left(\frac{q_{H_4}/(2^{l_0+l_1})}{1 - q_{H_3}/q} + \frac{q_{H_3}/q}{1 - q_{H_4}/2^{l_0+l_1}} + 2/q \right) \right) \right) \end{aligned}$$

Let $T_1 = q_H + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{pke} + q_{uke} + q_{rkg} + q_{renc} + q_{dec1} + q_{dec2} + q_{pkr}$ and $T_2 = 8q_{pke} + 10q_{uke} + 5q_{rkg} + 8q_{renc} + 7q_{dec1} + 9q_{dec2}$. From the construction of C we can bound its running time t' by,

$$t' \leq t + (T_1)O(1) + (T_2)t_{exp}$$

8 Efficiency Comparison

We compare the efficiency of our scheme and the one proposed in [GZZC13] which is the only scheme (to the best of our knowledge) for which there are no known attacks. t_{exp} denotes the time for exponentiation and t_{bp} denotes the time for bilinear pairing computation in a group. According to the results in [Sco05, BKLS02] the time taken for pairing computation is more than twice that of modular exponentiation. In our scheme, we assume that $R_1, R_2, X_1, X_2, X_3, X'$ are precomputed and PublicKeyVerification, CiphertextVerification is done as a pre-processing step.

Algorithms	CL-PRE in [GZZC13]	Our scheme
Setup	$3t_{exp} + t_{bp}$	t_{exp}
UserSecretKeyGen	t_{exp}	$2t_{exp}$
SetPublicKey	0	$2t_{exp}$
PartialKeyExtract	$3t_{exp}$	$3t_{exp}$
ReKeyGen	$3t_{exp}$	$2t_{exp}$
Encryption	$4t_{exp}$	$3t_{exp}$
Re-Encryption	$2t_{bp}$	t_{exp}
Decrypt-1	$2t_{exp} + t_{bp}$	$2t_{exp}$
Decrypt-2	$4t_{exp} + t_{bp}$	$4t_{exp}$

References

- [AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.
- [ARP03] Sattam S Al-Riyami and Kenneth G Paterson. Certificateless public key cryptography. In *Advances in Cryptology-ASIACRYPT 2003*, pages 452–473. Springer, 2003.
- [BBS98] Matt Blaze, Gerit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology-EUROCRYPT'98*, pages 127–144. Springer, 1998.
- [BDZ03] Feng Bao, Robert H Deng, and Huafei Zhu. Variations of diffie-hellman problem. In *Information and Communications Security*, pages 301–312. Springer, 2003.
- [BKLS02] Paulo SLM Barreto, Hae Y Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in cryptology-CRYPTO 2002*, pages 354–369. Springer, 2002.
- [BSNS05] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Certificateless public key encryption without pairing. In *Information Security*, pages 134–148. Springer, 2005.
- [CBN06] Sherman SM Chow, Colin Boyd, and Juan Manuel González Nieto. Security-mediated certificateless cryptography. In *Public Key Cryptography-PKC 2006*, pages 508–524. Springer, 2006.
- [CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 185–194. ACM, 2007.
- [CLH05] Yun-Peng Chiu, Chin-Laung Lei, and Chun-Ying Huang. Secure multicast using proxy encryption. In *Information and Communications Security*, pages 280–290. Springer, 2005.
- [Cor00] Jean-Sébastien Coron. On the exact security of full domain hash. In *Advances in Cryptology-CRYPTO 2000*, pages 229–235. Springer, 2000.

- [CT07] Cheng-Kang Chu and Wen-Guey Tzeng. Identity-based proxy re-encryption without random oracles. In *Information Security*, pages 189–202. Springer, 2007.
- [CWYD10] Sherman SM Chow, Jian Weng, Yanjiang Yang, and Robert H Deng. Efficient unidirectional proxy re-encryption. In *Progress in Cryptology–AFRICACRYPT 2010*, pages 316–332. Springer, 2010.
- [Den08] Alexander W Dent. A survey of certificateless encryption schemes and security models. *International Journal of Information Security*, 7(5):349–377, 2008.
- [DWLC08] Robert H Deng, Jian Weng, Shengli Liu, and Kefei Chen. Chosen-ciphertext secure proxy re-encryption without pairings. In *Cryptology and Network Security*, pages 1–17. Springer, 2008.
- [GA07] Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In *Applied Cryptography and Network Security*, pages 288–306. Springer, 2007.
- [GZZC13] Hui Guo, Zhenfeng Zhang, Jiang Zhang, and Cheng Chen. Towards a secure certificateless proxy re-encryption scheme. In *Provable Security*, pages 330–346. Springer, 2013.
- [HBCDF06] Thomas S. Heydt-Benjamin, Hee-Jin Chae, Benessa Defend, and Kevin Fu. Privacy for public transportation. In *Privacy Enhancing Technologies*, pages 1–19, 2006.
- [LCT⁺14] Kaitai Liang, Cheng-Kang Chu, Xiao Tan, Duncan S Wong, Chunming Tang, and Jianying Zhou. Chosen-ciphertext secure multi-hop identity-based conditional proxy re-encryption with constant-size ciphertexts. *Theoretical Computer Science*, 539:87–105, 2014.
- [LLSL14] Rongxing Lu, Xiaodong Lin, Jun Shao, and Kaitai Liang. Rcca-secure multi-use bidirectional proxy re-encryption with master secret security. In *Provable Security*, pages 194–205. Springer, 2014.
- [LQ06] Benoît Libert and Jean-Jacques Quisquater. On constructing certificateless cryptosystems from identity based encryption. In *Public Key Cryptography–PKC 2006*, pages 474–490. Springer, 2006.
- [LV08] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *Public Key Cryptography–PKC 2008*, pages 360–379. Springer, 2008.
- [SC09] Jun Shao and Zhenfu Cao. Cca-secure proxy re-encryption without pairings. In *Public Key Cryptography–PKC 2009*, pages 357–376. Springer, 2009.
- [Sco05] Michael Scott. Computing the tate pairing. In *Topics in Cryptology–CT-RSA 2005*, pages 293–304. Springer, 2005.
- [SJPR10] Chul Sur, Chae Duk Jung, Youngho Park, and Kyung Hyune Rhee. Chosen-ciphertext secure certificateless proxy re-encryption. In *Communications and Multimedia Security*, pages 214–232. Springer, 2010.
- [Smi] T Smith. Dvd jon: buy drm-less tracks from apple itunes (2005).
- [SZB07] Yinxia Sun, Futai Zhang, and Joonsang Baek. Strongly secure certificateless public key encryption without pairing. In *Cryptology and Network Security*, pages 194–208. Springer, 2007.
- [WDLC10] Jian Weng, Robert H Deng, Shengli Liu, and Kefei Chen. Chosen-ciphertext secure bidirectional proxy re-encryption schemes without pairings. *Information Sciences*, 180(24):5077–5089, 2010.
- [YXZ14] Kang Yang, Jing Xu, and Zhenfeng Zhang. Certificateless proxy re-encryption without pairings. In *Information Security and Cryptology–ICISC 2013*, pages 67–88. Springer, 2014.

- [ZTGC13] Yulin Zheng, Shaohua Tang, Chaowen Guan, and Min-Rong Chen. Cryptanalysis of a certificateless proxy re-encryption scheme. In *Emerging Intelligent Data and Web Technologies (EIDWT), 2013 Fourth International Conference on*, pages 307–312. IEEE, 2013.