# An Efficient Certificateless Proxy Re-Encryption Scheme without Pairing

S. Sharmila Deva Selvi, Arinjita Paul and C. Pandu Rangan

Theoretical Computer Science Lab,
Department of Computer Science and Engineering,
Indian Institute of Technology Madras, Chennai, India.
{sharmila,arinjita,prangan}@cse.iitm.ac.in

**Abstract.** Proxy re-encryption (PRE) is a cryptographic primitive introduced by Blaze, Bleumer and Strauss [4] to provide delegation of decryption rights. PRE allows re-encryption of a ciphertext intended for Alice (delegator) to a ciphertext for Bob (delegatee) via a semi-honest proxy, who should not learn anything about the underlying message. In 2003, Al-Riyami and Patterson introduced the notion of certificateless public key cryptography which offers the advantage of identity-based cryptography without suffering from the key escrow problem. The existing certificateless PRE (CLPRE) schemes rely on costly bilinear pairing operations. In ACM ASIA-CCS SCC 2015, Srinivasan *et al*. proposed the first construction of a certificateless PRE scheme without resorting to pairing in the random oracle model. However, in this work, we demonstrate a flaw in the CCA-security proof of their scheme. Also, we present the first construction of a CLPRE scheme without pairing which meets CCA security under the computational Diffie-Hellman hardness assumption in the random oracle model.

**Keywords:** Proxy Re-Encryption, Pairing-less, Public Key, Certificateless, Unidirectional.

## 1 Introduction

Due to segregation of data ownership and storage, security remains as one of the major concerns in the public cloud scenario. In order to protect the stored data from illegal access and usage, users encrypt their data with their public keys before storing it in the cloud. To enable sharing of the stored data, a naive approach would be that a user Alice shares her secret key with a legitimate user Bob. However, this would compromise the privacy of Alice. As a solution towards providing delegation of decryption rights, Blaze *et al*. [4] in 1998 proposed the concept of proxy re-encryption, which allows a proxy server with special information (re-encryption key) to translate a ciphertext for Alice into another ciphertext (with the same message) for Bob, without learning any information about the underlying plaintext. Besides, this approach also offloads the costly burden of secure data sharing from Alice to the resource-abundant proxy. As Alice delegates her decryption rights to Bob via a proxy-server, Alice is termed as the "delegator" and Bob as the "delegatee". Ever since, PRE has found a lot of applications such as encrypted email forwarding, distributed file systems, digital rights management(DRM) of Apple's iTunes, outsourced filtering of encrypted spam and content distribution [2,3,5,16].

Based on the direction of the delegation, PRE schemes are classified into bidirectional and unidirectional schemes. In unidirectional schemes, a proxy can re-encrypt ciphertexts from Alice to Bob but not from Bob to Alice, while in the bidirectional schemes, the proxy is allowed to re-encrypt ciphertexts in both the directions. PRE schemes are also classified into single-hop and multihop schemes. In a single-hop scheme, a proxy cannot re-encrypt ciphertexts that have been re-encrypted once. In a multi-hop scheme, the proxy can further re-encrypt the re-encrypted ciphertexts. In this paper, we focus on single-hop unidirectional PRE schemes.

Several PRE constructions have been proposed in the literature, either in the Public Key Infrastructure (PKI) or identity based (IBE) setting. The schemes in the PKI setting entrusts a third party called the Certification Authority(CA) to assure the authenticity of a user's public key by digitally signing it and issuing Digital Certificates. However, the overhead involved in the revocation, storage and distribution of certificates has long been a concern, which makes public key cryptography inefficient. As a solution to the authenticity problem, Identity-based cryptography was introduced by Shamir in 1984 [12], which involves a trusted third party called the Private Key Generator (PKG) to generate the secret keys of all the users. Yet again, due to the unconditional trust placed on the PKG, identity based cryptography suffers from the *key-escrow problem*. To avoid both the certificate management problem in the PKI setting and the key-escrow problem in the ID-based setting, certificateless cryptography was introduced in 2003 by Al-Riyami and Patterson [1]. Certificateless cryptography splits the task of key-generation of the user between a semi-trusted entity called the Key Generation Center (KGC) and the user himself. This approach no longer relies on the use of certificates for the authenticity of keys and hence does not suffer from the certificate management problem. Also, the KGC does

not have access to the secret keys of the users, which addresses the key-escrow problem inherent in IBE setting. Thus, certificateless cryptography setting enjoys the benefits of both PKI-based and ID-based cryptography.
In this paper, we study proxy re-encryption in the light of certificateless public key cryptography. Consider the following scenario that motivates the need for proxy re-encryption in the *certificateless* setting. Suppose Alice stores her encrypted data in the cloud, which provides services to billions of users. We note that the number of cloud users is large and therefore, management of certificates for public key authenticity is an overhead. This makes proxy re-encryption in the PKI setting unfit for cloud services. On the other hand, entrusting a third party PKG with the power to generate the secret keys of the users makes the cloud vulnerable to key-escrow problem. A malicious PKG can decrypt the confidential data of the users, due to which PRE in IBE setting is highly impractical. Certificateless PRE affirmatively solves both the certificate management problem and key-escrow problem in the above scenario.
The existing certificateless PRE scheme [9] in the literature is based on bilinear pairing. Note that bilinear pairing is an expensive computation when compared to modular exponentiation operation in finite fields. In this work, we propose the first pairing-free unidirectional single-hop CCA-secure CL-PRE scheme in the random oracle model.

### 1.1 Related Work and Contribution

While several schemes achieving PRE have been proposed in the literature, a majority of these schemes are either in the Public Key Infrastructure (PKI) setting or Identity-Based (IBE) setting. Certificateless public key cryptography introduced in [1] offers the advantage of identity-based cryptography without suffering from its inherent key-escrow problem. In 2010, Sur *et al.* [14] introduced the notion of certificateless proxy re-encryption (CL-PRE) and proposed a CCA secure CL-PRE scheme in the random oracle model. However, in 2013, their scheme was shown to be vulnerable to chosen ciphertext attack by Zheng *et al.* [17]. In 2013, Guo *et al.* [9] proposed a CL-PRE scheme in the random oracle model based on bilinear pairing which satisfies RCCA-security, a weaker notion of security. In 2014, Yang, Xu and Zhang [15] proposed a CCA-secure CL-PRE scheme without pairing in the random oracle model, which was later shown to be vulnerable to chain collusion attack in [13]. In 2015, Srinivasan *et al.* [13] proposed a CCA-secure unidirectional certificateless PRE scheme without pairing under the computational Diffie-Hellman assumption in the random oracle model. In this paper, we expose a critical weakness present in the security proof of the scheme and provide a potential fix to make the scheme provably secure.
Another major contribution of this work is we propose an efficient pairing-free unidirectional single-hop certificateless proxy re-encryption scheme in the random oracle model. As stated, all the existing CL-PRE schemes are vulnerable to attacks except for [9]. The CL-PRE scheme due to Guo *et al.* [9] is based on bilinear pairing which is an expensive operation as compared to modular exponentiation operations in finite fields. Besides, their scheme [9] satisfies a weaker notion of security, namely the RCCA-security and is based on *q-weak Decisional Bilinear Assumption*. Our scheme satisfies CCA security against both the Type-I and Type-II adversaries and is based on a much standard assumption called the Computational Diffie Hellman ($CDH$) assumption.

## 2 Definition and Security Model

### 2.1 Definition

We describe the syntactical definition of unidirectional single-hop certificateless proxy re-encryption and its security notion. We follow the model given in [13] for a certificateless proxy re-encryption scheme. A PRE scheme consists of the following ten algorithms:

- **Setup**$(1^\lambda)$: A probabilistic polynomial time (PPT) algorithm run by the Key Generation Center(KGC), which takes the unary encoding of the security parameter $\lambda$ as input and outputs the public parameters $params$ and the master secret key $msk$.
- **PartialKeyExtract**$(msk, ID_i, params)$: A PPT algorithm run by the KGC which takes the master secret key $msk$, user identity $ID_i$ and public parameters $params$ as input, and outputs the partial public key and partial secret key pair $(PPK_i, PSK_i)$.
- **UserKeyGen**$(ID_i, params)$: A PPT algorithm run by the user, which takes the identity $ID_i$ of the user and the public parameters $params$ as input, and outputs the user generated public key and secret key pair $(USK_i, UPK_i)$.
- **SetPrivateKey**$(ID_i, PSK_i, USK_i, params)$: A PPT algorithm run by the user, which takes as input the identity $ID_i$ of the user, partial secret key $PSK_i$, user generated secret key $USK_i$ and public parameters $params$, and outputs the full secret key $SK_i$ of the user.
- **SetPublicKey**$(ID_i, PPK_i, PSK_i, UPK_i, USK_i, params)$: A PPT algorithm run by the user, which takes as input the the identity $ID_i$ of the user, partial public key $PPK_i$, partial secret key $PSK_i$, user generated public key $UPK_i$, user generated secret key $USK_i$ and public parameters $params$, and outputs the full public key $PK_i$ of the user.

- **Re-KeyGen**$(ID_i, ID_j, SK_i, PK_j, params)$: A PPT algorithm run by the user (delegator) with identity $ID_i$ which takes as input the identity $ID_i$ of the delegator, identity $ID_j$ of the delegatee, the full secret key $SK_i$ of $ID_i$, full public key $PK_j$ of $ID_j$ and public parameters $params$, and outputs a re-encryption key $RK_{i\to j}$ or an error symbol $\perp$.
- **Encrypt**$(ID_i, PK_i, m, params)$: A PPT algorithm run by the sender which takes as input the identity $ID_i$ of the receiver, full public key $PK_i$ of $ID_i$, a message $m \in \mathcal{M}$ and the public parameters $params$, and outputs the ciphertext $C$ or an error symbol $\perp$. Note that $C$ is termed as the first level ciphertext.
- **Re-Encrypt**$(ID_i, ID_j, C, RK_{i\to j}, params)$: A PPT algorithm run by the proxy which takes the identities $ID_i, ID_j$, a first level ciphertext $C$ encrypted under identity $ID_i$, a re-encryption key $RK_{i\to j}$ and public parameters $params$ as input, and outputs a ciphertext $D$ or an error symbol $\perp$. Note that $D$ is termed as the second-level ciphertext.
- **Decrypt**$(ID_i, SK_i, C, params)$: A deterministic algorithm run by the receiver (delegator) which takes the identity $ID_i$, secret key $SK_i$ of identity $ID_i$, first-level ciphertext $C$ and public parameters $params$ as input, and outputs the message $m \in \mathcal{M}$ or an error symbol $\perp$.
- **Re-Decrypt**$(ID_j, SK_j, D, params)$: A deterministic algorithm run by the receiver (delegatee) which takes the identity $ID_j$, secret key $SK_j$ of identity $ID_j$, a second-level ciphertext $D$ and public parameters $params$ as input, and outputs the message $m \in \mathcal{M}$ or an error symbol.

The consistency of a CL-PRE scheme for any given public parameters $params$ and full public-private key pairs $\{(PK_i, SK_i), (PK_j, SK_j)\}$ is defined as follows:

1. Consistency between encryption and decryption; i.e.,

$$\textbf{Decrypt}(ID_i, SK_i, C, params) = m,\ \forall m \in \mathcal{M},$$

   where $C =$**Encrypt**$(ID_i, PK_i, m, params)$.

2. Consistency between encryption, proxy re-encryption and decryption; i.e.,

$$\textbf{Re-Decrypt}(ID_j, SK_j, D, params) = m,\ \forall m \in \mathcal{M},$$

   where $D =$**Re-Encrypt**$(ID_i, ID_j, C, RK_{i\to j}, params)$ and $C =$**Encrypt**$(ID_i, PK_i, m, params)$.

### 2.2 Security Model

Due to the existence of two types of ciphertexts in a PRE scheme namely *first level* and *second level* ciphertexts, it is essential to prove the security for both levels [10]. Again, there exists two types of adversaries specific to CL-PRE: *Type-I* adversary and *Type-II* adversary. The Type I adversary models an attacker who can replace the public keys of the users by fake keys of its choice because of the absence of authenticating information for public keys [1]. However, the security proof demonstrates that the adversary cannot learn anything useful from this attack as it cannot derive the partial keys and in turn the full private keys needed for decryption without the cooperation of the KGC (who possesses the master secret key). The Type-II adversary models the semi-trusted KGC, who possesses the master secret key and tries to break the security of the system by eavesdropping or making decryption queries. Note that, the KGC is restrained from replacing the public keys of the users.
The security of a CL-PRE scheme is modelled in the form of a security game between the two entities : the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$. $\mathcal{A}$ can adaptively query the oracles as listed below which $\mathcal{C}$ answers and simulates an environment running CL-PRE for $\mathcal{A}$. $\mathcal{C}$ maintains a list $P_{current}$ of the public keys to keep a track of the replaced public keys. $P_{current}$ consists of tuples of the form $\langle ID_i, PK_i, \hat{PK_i}\rangle$, where $\hat{PK_i}$ denotes the current value of the public key. To begin with, $\hat{PK_i}$ is assigned the value of the initial public key $\hat{PK_i} = PK_i$. $\mathcal{A}$ can make queries to the following oracles which are answered by $\mathcal{C}$:

- Public Key Extract$(\mathcal{O}_{pe}(ID_i))$: Given an $ID_i$ as input, compute the partial public key and secret key pair: $(PPK_i, PSK_i) = PartialKeyExtract(msk, ID_i, params)$, the user public key and secret key pair: $(USK_i, UPK_i) = UserKeyGen(ID_i, params)$, the full public key $PK_i = SetPublicKey(ID_i, PPK_i, PSK_i, UPK_i, USK_i, params)$. Return $PK_i$.
- Partial Key Extract$(\mathcal{O}_{ppe}(ID_i))$: Given an $ID_i$ as input, compute $(PPK_i, PSK_i) = PartialKeyExtract(msk, ID_i, params)$ and return $(PPK_i, PSK_i)$.
- User Key Extract$(\mathcal{O}_{ue}(ID_i))$: Given an $ID_i$ as input, compute $(UPK_i, USK_i) = UserKeyGen(ID_i, params)$ and return $(USK_i, UPK_i)$.
- Re-Key Generation$(\mathcal{O}_{rk}(ID_i, ID_j))$: Compute $RK_{i\to j} = Re\text{-}KeyGen(ID_i, ID_j, SK_i, PK_j, params)$ and return $RK_{i\to j}$.

– Re-Encryption($\mathcal{O}_{re}(ID_i, ID_j, C)$): Given a first-level ciphertext $C$ and two identities $ID_i, ID_j$ as inputs, compute $RK_{i\to j} = Re\text{-}KeyGen(ID_i, ID_j, SK_i, PK_j, params)$ and compute the second level ciphertext as $D = Re\text{-}Encrypt(ID_i, ID_j, C, RK_{i\to j}, params)$.
– Decryption($\mathcal{O}_{dec}(ID_i, C)$): Given a first level ciphertext $C$ encrypted under the public key of $ID_i$ as input, compute the decryption of the ciphertext to obtain $m \in \mathcal{M}$. Return $m$ or return $\perp$ if the ciphertext is invalid.
– Re-Decryption($\mathcal{O}_{redec}(ID_i, C)$): Given a second level ciphertext $D$ re-encrypted under the public key $ID_j$ as input, compute the decryption of the ciphertext to obtain $m \in \mathcal{M}$. Return $m$ or return $\perp$ if the ciphertext is invalid.
– Public Key Replacement($\mathcal{O}_{rep}(ID_i, PK_i)$): Replace the value of the third component $\hat{PK}_i$ in the $PK_{current}$ list with the new value $PK_i$, provided $PK_i$ is a valid public key.

**Security against Type-I adversary $\mathcal{A}_I$**

The Type-I adversary models an outside attacker without access to the master secret key, trying to learn some information about the underlying plaintext, given the ciphertext. We consider separate security models for the first level and second level ciphertexts against $\mathcal{A}_I$.

**First Level Ciphertext Security:** We consider the following security game where $\mathcal{A}_I$ interacts with the challenger $\mathcal{C}$ in following stages.

- Initialization: $\mathcal{C}$ runs the $Setup(\lambda)$ algorithm to generate the public parameters $params$ and the master secret key $msk$. It sends $params$ to $\mathcal{A}_I$ while keeping $msk$ secret.
- Phase 1: The challenger $\mathcal{C}$ sets up the list of corrupt and honest users, initialises $\hat{PK}_i$ to $PK_i$ for all the users in the public key list $P_{current}$. $\mathcal{A}_I$ issues several queries to the above stated oracles simulated by $\mathcal{C}$, with the restriction that $\mathcal{A}_I$ cannot make partial key extract queries ($\mathcal{O}_{ppe}$) or user key extract queries ($\mathcal{O}_{ue}$) of the users whose public keys have been replaced as it is unreasonable to expect $\mathcal{C}$ to respond to such queries for the public keys replaced by $\mathcal{A}_I$ [1].
- Challenge: Once $\mathcal{A}_I$ decides that phase 1 is over, it outputs two equal length messages $m_0$ and $m_1$ in $\mathcal{M}$ and the target identity $ID_{ch}$, with the following adversarial constraints:
  – $ID_{ch}$ should not be a corrupt user.
  – $\mathcal{A}_I$ must not query the partial key extract oracle ($\mathcal{O}_{ppe}$) or user key extract oracle ($\mathcal{O}_{ue}$) of $ID_{ch}$ at any point in time.
  – $\mathcal{A}_I$ must not query $\mathcal{O}_{rk}(ID_{ch}, ID_i)$, where $ID_i$ is a corrupt user.
  – If $\mathcal{A}_I$ replaces the public key of $ID_{ch}$, it should not query the partial key extract oracle ($\mathcal{O}_{ppe}$) for $ID_{ch}$.
  On receiving $\{m_0, m_1\}$, $\mathcal{C}$ picks $\delta \in \{0, 1\}$ at random and generates a challenge ciphertext $C^* = Encrypt(ID_{ch}, \hat{PK}_{ch}, m_\delta, params)$ and gives to $\mathcal{A}_I$.
- Phase 2: $\mathcal{A}_I$ issues the queries to the oracles similar to Phase 1, with the same adversarial constraint as mentioned in Phase 1 and the added constraints on the target identity $ID_{ch}$ as mentioned in the Challenge phase. Additionally, there are other constraints as below:
  – $\mathcal{A}_I$ cannot query $\mathcal{O}_{dec}(ID_{ch}, C^*)$, for the same public key of $ID_{ch}$ that was used to initially encrypt $m_\delta$.
  – $\mathcal{A}_I$ cannot query the re-decryption oracle $\mathcal{O}_{redec}(ID_i, C)$ if $(ID_i, C)$ is a challenge derivative[1].
  – $\mathcal{A}_I$ cannot query $\mathcal{O}_{re}(ID_i, ID_j, C)$, if $(ID_i, C)$ is a challenge derivative and $ID_j$ is a corrupt user.
  – $\mathcal{A}_I$ cannot query $\mathcal{O}_{rk}(ID_{ch}, ID_j)$, if $ID_j$ is a corrupt user.
- Guess: $\mathcal{A}_I$ outputs its guess $\delta' \in \{0, 1\}$.

We define the advantage of $\mathcal{A}_I$ in winning the game as:

$$Adv^{IND-CLPRE-CCA}_{A_I, first} = 2|Pr\lceil \delta' = \delta \rfloor - \tfrac{1}{2}|$$

where the probability is over the random coin tosses performed by $\mathcal{C}$ and $\mathcal{A}_I$. The scheme is said to be $(t, \epsilon)IND - CLPRE - CCA$ secure for the first level ciphertext against Type-I adversary $\mathcal{A}_I$ if for all $t$-time adversary $\mathcal{A}_I$ that makes $q_{pe}$ queries to $\mathcal{O}_{pe}$, $q_{ppe}$ queries to $\mathcal{O}_{ppe}$, $q_{ue}$ queries to $\mathcal{O}_{ue}$, $q_{re}$ queries to $\mathcal{O}_{re}$, $q_{rk}$ queries to $\mathcal{O}_{rk}$, $q_{dec}$ queries to $\mathcal{O}_{dec}$, $q_{redec}$ queries to $\mathcal{O}_{redec}$ and $q_{rep}$ queries to $\mathcal{O}_{rep}$, the advantage of $\mathcal{A}_I$ is $Adv^{IND-CLPRE-CCA}_{A_I, first} \leq \epsilon$.

[1] The definition of challenge derivative $(ID_i, C)$ is adopted from [6] as stated below:
∗ Reflexitivity: $(ID_i, C)$ is a challenge derivative of itself.
∗ Derivative by re-encryption: $(ID_j, C')$ is a challenge derivative of $(ID_i, C)$ if $C' \leftarrow \mathcal{O}_{re}(ID_i, ID_j, C)$.
∗ Derivative by re-encryption key: $(ID_j, C')$ is a challenge derivative of $(ID_i, C)$ if $RK_{i\to j} \leftarrow \mathcal{O}_{rk}(ID_i, ID_j)$ and $C' = Re - Encrypt(ID_i, ID_j, C, RK_{i\to j}, params)$.

**Second Level Ciphertext Security:** We consider the following security game for security of the second level ciphertext against Type-I adversary $\mathcal{A}_I$, where $\mathcal{A}_I$ interacts with the challenger $\mathcal{C}$ in following stages.

- Initialization: $\mathcal{C}$ runs the $Setup(\lambda)$ algorithm to generate the public parameters $params$ and the master secret key $msk$. It sends the $params$ to $\mathcal{A}_I$ while keeping $msk$ secret.
- Phase 1: The challenger $\mathcal{C}$ sets up the list of corrupt and honest users, initialises $\hat{PK}_i$ to $PK_i$ for all the users and updates the public key list $P_{current}$. $\mathcal{A}_I$ issues several queries to the above stated oracles simulated by $\mathcal{C}$ with the restriction that it cannot make partial key extract queries ($\mathcal{O}_{ppe}$) or user key extract queries ($\mathcal{O}_{ue}$) of the users whose public keys have already been replaced.
- Challenge: $\mathcal{A}_I$ outputs two messages $m_0$ and $m_1$ in $\mathcal{M}$ where $|m_0| = |m_1|$, the target identity $ID_{ch}$, and the delegator's identity $ID_{del}$ with the adversarial constraints as follows:
  - $ID_{ch}$ should not be a corrupt user.
  - $\mathcal{A}_I$ must not query the partial key extract oracle ($\mathcal{O}_{ppe}$) or user key extract oracle ($\mathcal{O}_{ue}$) of $ID_{ch}$ at any point in time.
  - If $\mathcal{A}_I$ replaces the public key of $ID_{ch}$, it should not query the partial key extract oracle ($\mathcal{O}_{ppe}$) for $ID_{ch}$.
  - $\mathcal{A}_I$ must not query $\mathcal{O}_{rk}(ID_{del}, ID_{ch})$.
  - $\mathcal{A}_I$ must not query $\mathcal{O}_{rk}(ID_{ch}, ID_i)$, where $ID_i$ is a corrupt user.

  On receiving $\{m_0, m_1\}$, $\mathcal{C}$ picks $\delta \in \{0,1\}$ at random and generates a challenge ciphertext $D^* = Re{-}Encrypt(ID_{del}, ID_{ch}, Encrypt(ID_{ch}, \hat{PK}_{ch}, m_\delta, params), RK_{ID_{del} \to ID_{ch}}, params)$ and gives to $\mathcal{A}_I$.
- Phase 2: $\mathcal{A}_I$ issues the queries to the oracles similar to Phase 1, with the same adversarial constraint as mentioned in Phase 1 and constraints on the target identity $ID_{ch}$ mentioned in the Challenge phase. Additionally, $\mathcal{A}_I$ cannot query $\mathcal{O}_{redec}(ID_{ch}, C^*)$, for the same public key of $ID_{ch}$ that was used to initially encrypt $m_\delta$.
- Guess: $\mathcal{A}_I$ outputs its guess $\delta' \in \{0,1\}$.

We define the advantage of $\mathcal{A}_I$ in winning the game as:

$$Adv_{A_I,second}^{IND-CLPRE-CCA} = 2|Pr\lceil \delta' = \delta \rfloor - \tfrac{1}{2}|$$

where the probability is over the random coin tosses performed by $\mathcal{C}$ and $\mathcal{A}_I$. The scheme is said to be $(t, \epsilon)IND - CLPRE - CCA$ secure for the second level ciphertext against Type-I adversary $\mathcal{A}_I$ if for all $t$-time adversary $\mathcal{A}_I$ that makes $q_{pe}$ queries to $\mathcal{O}_{pe}$, $q_{ppe}$ queries to $\mathcal{O}_{ppe}$, $q_{ue}$ queries to $\mathcal{O}_{ue}$, $q_{re}$ queries to $\mathcal{O}_{re}$, $q_{rk}$ queries to $\mathcal{O}_{rk}$, $q_{dec}$ queries to $\mathcal{O}_{dec}$, $q_{redec}$ queries to $\mathcal{O}_{redec}$ and $q_{rep}$ queries to $\mathcal{O}_{rep}$, the advantage of $\mathcal{A}_I$ is $Adv_{A_I,second}^{IND-CLPRE-CCA} \leq \epsilon$.

### Security against Type-II adversary $\mathcal{A}_{II}$

The Type-II adversary models an *honest-but-curious KGC* who has access to the master secret key $msk$, but is not allowed to replace the public keys of the users. We consider separate security models for the first level and second level ciphertexts.

**First Level Ciphertext Security:** We consider the following security game where $\mathcal{A}_{II}$ interacts with the challenger $\mathcal{C}$ as follows.

- Initialization: $\mathcal{C}$ runs the $Setup(\lambda)$ algorithm to generate the public parameters $params$ and the master secret key $msk$. It sends both $params$ and $msk$ to $\mathcal{A}_{II}$.
- Phase 1: The challenger $\mathcal{C}$ maintains the list of honest and corrupt users and initialises $\hat{PK}_i$ to $PK_i$ for all the users in the public key list $P_{current}$. $\mathcal{A}_{II}$ issues several queries to the above stated oracles simulated by $\mathcal{C}$ with the restriction that it cannot make partial key extract queries ($\mathcal{O}_{ppe}$) or user key extract queries ($\mathcal{O}_{ue}$) of the users whose public keys have been replaced.
- Challenge: Once $\mathcal{A}_{II}$ decides that phase 1 is over, it outputs two equal length messages $\{m_0, m_1\}$ in $\mathcal{M}$ and the target identity $ID_{ch}$, with the adversarial constraints as follows:
  - $ID_{ch}$ should not be a corrupt user.
  - $\mathcal{A}_{II}$ must not replace the public key of $ID_{ch}$.
  - $\mathcal{A}_{II}$ must have not queried $\mathcal{O}_{rk}(ID_{ch}, ID_i)$, where $ID_i$ is a corrupt user.

  On receiving $\{m_0, m_1\}$, $\mathcal{C}$ selects $\delta \in \{0,1\}$ at random, generates a challenge ciphertext $C^* = Encrypt(ID_{ch}, \hat{PK}_{ch}, m_\delta, params)$ and gives $C^*$ to $\mathcal{A}_{II}$.
- Phase 2: $\mathcal{A}_{II}$ issues the queries to the oracles similar to Phase 1, with the same adversarial constraints as mentioned in Phase 1 and the constraints on the target identity $ID_{ch}$ as mentioned in the Challenge phase. Additionally, there are other constraints as below:

- $\mathcal{A}_{II}$ cannot query $\mathcal{O}_{dec}(ID_{ch}, C^*)$, for the same public key of $ID_{ch}$ that was used to initially encrypt $m_\delta$.
- $\mathcal{A}_{II}$ cannot query the re-decryption oracle $\mathcal{O}_{redec}(ID, C)$ if $(ID, C)$ is a challenge derivative.
- $\mathcal{A}_{II}$ cannot query $\mathcal{O}_{re}(ID_i, ID_j, C)$, if $(ID_i, C)$ is a challenge derivative and $ID_j$ is a corrupt user.
- $\mathcal{A}_{II}$ cannot query $\mathcal{O}_{rk}(ID_{ch}, ID_j)$, if $ID_j$ is a corrupt user.

– Guess: $\mathcal{A}_{II}$ outputs its guess $\delta' \in \{0, 1\}$.

We define the advantage of $\mathcal{A}_{II}$ in winning the game as:

$$Adv_{A_{II},first}^{IND-CLPRE-CCA} = 2|Pr\lceil\delta' = \delta\rfloor - \tfrac{1}{2}|$$

where the probability is over the random coin tosses performed by $\mathcal{C}$ and $\mathcal{A}_{II}$. The scheme is said to be $(t, \epsilon)IND - CLPRE - CCA$ secure for the first level ciphertext against Type-II adversary $\mathcal{A}_{II}$ if for all $t$-time adversary $\mathcal{A}_{II}$ that makes $q_{pe}$ queries to $\mathcal{O}_{pe}$, $q_{ppe}$ queries to $\mathcal{O}_{ppe}$, $q_{ue}$ queries to $\mathcal{O}_{ue}$, $q_{re}$ queries to $\mathcal{O}_{re}$, $q_{rk}$ queries to $\mathcal{O}_{rk}$, $q_{dec}$ queries to $\mathcal{O}_{dec}$, $q_{redec}$ queries to $\mathcal{O}_{redec}$ and $q_{rep}$ queries to $\mathcal{O}_{rep}$, the advantage of $\mathcal{A}_{II}$ is $Adv_{A_{II},first}^{IND-CLPRE-CCA} \leq \epsilon$.

**Second Level Ciphertext Security:** We consider the following security game where $\mathcal{A}_{II}$ interacts with the challenger $\mathcal{C}$ in the following stages.

- Initialization: $\mathcal{C}$ runs the $Setup(\lambda)$ algorithm to generate the public parameters $params$ and the master secret key $msk$. It sends both $params$ and $msk$ to $\mathcal{A}_{II}$.
- Phase 1: The challenger $\mathcal{C}$ sets up the list of corrupt and honest users, initialises $\hat{PK}_i$ to $PK_i$ for all the users and updates the public key list $P_{current}$. $\mathcal{A}_{II}$ issues several queries to the above stated oracles simulated by $\mathcal{C}$ with the restriction that it cannot make partial key extract queries ($\mathcal{O}_{ppe}$) or user key extract queries ($\mathcal{O}_{ue}$) of the users whose public keys have been replaced. Also, $\mathcal{A}_{II}$ cannot place queries to $\mathcal{O}_{ppe}$ as it already has access to $msk$ and can generate the partial keys itself.
- Challenge: Once $\mathcal{A}_{II}$ decides that phase 1 is over, it outputs two messages $m_0$ and $m_1$ in $\mathcal{M}$ where $|m_0| = |m_1|$, the target identity $ID_{ch}$, and the delegator's identity $ID_{del}$ with the adversarial constraints as follows:
  - $ID_{ch}$ should not be a corrupt user.
  - $\mathcal{A}_{II}$ must not query the user key extract oracle ($\mathcal{O}_{ue}$) of $ID_{ch}$ at any point in time.
  - $\mathcal{A}_{II}$ must not replace the public key of $ID_{ch}$.
  - $\mathcal{A}_{II}$ must not query $\mathcal{O}_{rk}(ID_{del}, ID_{ch})$.
  - $\mathcal{A}_{II}$ must have not queried $\mathcal{O}_{rk}(ID_{ch}, ID_i)$, where $ID_i$ is a corrupt user.

  On receiving $\{m_0, m_1\}, \mathcal{C}$ picks $\delta \in \{0, 1\}$ at random and generates a challenge ciphertext $D^* = Re{-}Encrypt(ID_{del}, ID_{ch}, Encrypt(ID_{ch}, \hat{PK}_{ch}, m_\delta, params), RK_{ID_{del} \to ID_{ch}}, params)$ and gives to $\mathcal{A}_{II}$.
- Phase 2: $\mathcal{A}_{II}$ issues the queries to the oracles similar to Phase 1, with the same adversarial constraint as mentioned in Phase 1 and the added constraint on the target identity $ID_{ch}$ as mentioned in the Challenge phase. Additionally, $\mathcal{A}_{II}$ cannot query $\mathcal{O}_{redec}(ID_{ch}, C^*)$, for the same public key of $ID_{ch}$ that was used to initially encrypt $m_\delta$.

– Guess: $\mathcal{A}_{II}$ outputs its guess $\delta' \in \{0, 1\}$.

We define the advantage of $\mathcal{A}_{II}$ in winning the game as:

$$Adv_{A_{II},second}^{IND-CLPRE-CCA} = 2|Pr\lceil\delta' = \delta\rfloor - \tfrac{1}{2}|$$

where the probability is over the random coin tosses performed by $\mathcal{C}$ and $\mathcal{A}_{II}$. The scheme is said to be $(t, \epsilon)IND - CLPRE - CCA$ secure for the second level ciphertext against Type-II adversary $\mathcal{A}_{II}$ if for all $t$-time adversary $\mathcal{A}_{II}$ that makes $q_{pe}$ queries to $\mathcal{O}_{pe}$, $q_{ppe}$ queries to $\mathcal{O}_{ppe}$, $q_{ue}$ queries to $\mathcal{O}_{ue}$, $q_{re}$ queries to $\mathcal{O}_{re}$, $q_{rk}$ queries to $\mathcal{O}_{rk}$, $q_{dec}$ queries to $\mathcal{O}_{dec}$, $q_{redec}$ queries to $\mathcal{O}_{redec}$ and $q_{rep}$ queries to $\mathcal{O}_{rep}$, the advantage of $\mathcal{A}_{II}$ is $Adv_{A_{II},second}^{IND-CLPRE-CCA} \leq \epsilon$.

**Hardness Assumption**

We state the computational hardness assumption we use to prove the security of our scheme. Let $\mathbb{G}$ be a cyclic group with a prime order $q$.

**Definition 1.** ***Computational Diffie-Hellman (CDH) assumption*** *: The Computational Diffie-Hellman (CDH) assumption for group $\mathbb{G}$ says that, given the elements $\{P, aP, bP\} \in \mathbb{G}$, there exists no probabilistic polynomial-time adversary which can compute $abP \in \mathbb{G}$ with a non-negligible advantage, where $P$ is a generator of $\mathbb{G}$ and $a, b \in_R \mathbb{Z}_q^*$.*

## 3 Analysis of a Certificateless PRE Scheme by Srinivasan *et al.*[13]

We review the scheme due to Srinivasan *et al.* [13] and point out the weakness of the scheme in this section.

### 3.1 Review of the scheme

- **Setup**$(1^\lambda)$:
  - Choose two large primes $p$ and $q$ such that $q|p-1$ and the security parameter $\lambda$ defines the bit length of $q$. Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_p^*$ of order $q$ and $g$ is a generator of $\mathbb{G}$.
  - Pick $x \in_R \mathbb{Z}_q^*$ and compute $y = g^x$.
  - Choose the following cryptographic hash functions:

$$\begin{aligned} H &: \mathbb{G} \to \mathbb{Z}_q^*, \\ H_1 &: \{0,\ 1\}^* \times \mathbb{G} \to \mathbb{Z}_q^*, \\ H_2 &: \{0,\ 1\}^* \times \mathbb{G}^3 \to \mathbb{Z}_q^*, \\ H_3 &: \mathbb{G} \to \{0,\ 1\}^{l_0+l_1}, \\ H_4 &: \{0,\ 1\}^{l_0} \times \{0,\ 1\}^{l_1} \to \mathbb{Z}_q^*, \\ H_5 &: \mathbb{G}^2 \times \{0,\ 1\}^{l_0+l_1} \to \mathbb{Z}_q^*, \\ H_6 &: \{0,\ 1\}^* \times \mathbb{G}^2 \to \mathbb{Z}_q^* \end{aligned}$$

    Here $l_0 = \log q$ and $l_1$ is determined by the security parameter $\lambda$. The message space $\mathcal{M}$ is set to $\{0,\ 1\}^{l_0}$.
  - Return the 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 is $msk = x$.

- **PartialKeyExtract**$(msk, ID, params)$:
  - Pick $s_1, s_2, s_3 \in_R \mathbb{Z}_q^*$ 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 $S_3 = s_3 + xH_2(ID, Q_1, Q_2, Q_3)$.
  - Return the partial public key $PPK = (Q_1, Q_2, Q_3, S_3)$ and the partial secret key $PSK = (S_1, S_2)$.

- **UserKeyGen**$(ID, params)$:
  - Pick $z_1, z_2 \in_R \mathbb{Z}_q^*$ and compute $(g^{z_1}, g^{z_2})$.
  - Return $USK = (U_1, U_2) = (z_1, z_2)$ and $UPK = (P_1, P_2) = (g^{z_1}, g^{z_2})$.

- **SetPublicKey**$(ID, PPK, PSK, UPK, USK, params)$:
  - Pick $t_1, t_2 \in_R \mathbb{Z}_q^*$. Compute $T_1 = g^{t_1}$ and $T_2 = g^{t_2}$.
  - Compute $\mu_1 = t_1 + S_1 H_6(ID, P_1, T_1)$ and $\mu_2 = t_2 + S_2 H_6(ID, P_2, T_2)$.
  - Return the full public key of the user as $PK = (P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2)$.

- **Public Verify**$(ID, PK, params)$:
  - Parse $PK$ as $(P_1, P_2, Q_1, Q_2, Q_3, S_3, T_1, T_2, \mu_1, \mu_2)$.
  - Compute $R_1 = Q_1 \cdot y^{H_1(ID,Q_1)}$ and $R_2 = Q_2 \cdot y^{H_1(ID,Q_2)}$.
  - 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 all the above checks are satisfied, return *success*, else return $failure$.

- **SetPrivateKey**$(ID, PSK, USK, params)$:
  - Output the full secret key of the identity $ID$ as $SK = (U_1, U_2, S_1, S_2)$.

- **Re-KeyGen**$(ID_i, ID_j, SK_i, PK_j, params)$:
  - 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 verify the validity of the public key of $ID_j$ by checking if $Public\ Verify(ID_j, PK_j, params) = success$. If the check fails, return $\perp$.
  - 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 $\alpha = H(X)$.
  - Select $h \in_R \{0,1\}^{l_0}$ and $\pi \in_R \{0,1\}^{l_1}$. 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-encryption key $RK_{i \to j} = (rk, V, W)$.

- **Encrypt**$(ID_i, PK_i, m, params)$:
  - Parse $PK_i$ as $(P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{i,1}, T_{i,2}, \mu_{ji,1}, \mu_{i,2})$ and verify the validity of the public key $PK_i$ by checking if $Public\ Verify(ID_i, PK_i, params) = success$. If the check fails, output $\perp$.
  - Compute $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$ and $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$. 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 $Z = (X(Y)^{\alpha})$.
  - Select $u \in_R \mathbb{Z}_q^*$ and $\omega \in_R \{0,1\}^{l_1}$. Compute $r = H_4(m, \omega)$.
  - Compute $D = (Z)^u,\ E = Z^r,\ F = H_3(g^r) \oplus (m||\omega)$ and $s = u + rH_5(D, E, F)$.
  - Return the ciphertext $C = (D, E, F, s)$ as the first level ciphertext.

- **Re-Encrypt**$(ID_i, ID_j, C, RK_{i \to j}, params)$:
  - Parse $RK_{i \to j}$ as $(rk, V, W)$ and $C = (D, E, F, s)$. Check the validity of the ciphertext by computing $Z$ as shown in $Encrypt(ID_i, PK_i, m, params)$ and performing the following checks.

$$(Z)^s \stackrel{?}{=} D \cdot E^{H_5(D,E,F)} \tag{1}$$

    If the check fails, return $\perp$.
  - Else, compute $E' = R^{rk}$.
  - Output $D = (E', F, V, W)$ as the second level ciphertext.

- **Decrypt**$(ID_i, SK_i, C, params)$:
  - Obtain the public key $PK_i$ 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 $(D, E, F, s)$. Check the validity of the ciphertext by checking if equation 1 holds. If the check fails, output $\perp$.
  - Else, compute $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$ and $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$. 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 $(m||\omega) = F \oplus H_3(E^{\frac{1}{K}})$. Output $m$ if $E \stackrel{?}{=} (Z)^{H_4(m,\omega)}$ holds. Else, return $\perp$.

- **Re-Decrypt**$(ID_j, SK_j, D, params)$:
  - Parse $D$ 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 $R_{j,1} = Q_{j,1}(y^{H_1(ID_j, Q_{j,1})})$, $X_1 = P_{j,1}(R_{j,1}^{H(P_{j,1})})$.
  - Compute $(h||\pi) = W \oplus H_3(V^{\frac{1}{U_{j,1} + H(P_{j,1})S_{j,1}}})$ and $(m||\omega) = F \oplus H_3(E'^{1/h})$.
  - Output $m$ if $V \stackrel{?}{=} (X)_1^{H_4(h,\pi)},\ E' \stackrel{?}{=} g^{h(H_4(m,\omega))}$. Else, return $\perp$.

### 3.2 Our Attack

In this section, we highlight the flaw in the security reduction of the CLPRE scheme due to Srinivasan *et al.* [13]. We demonstrate that the simulation of the random oracles does not comply with the real system due to which, the adversary can distinguish the simulation of the challenger from the real system. Note that the flaw is observed in the proof for both $Type - I$ and $Type - II$ adversary and we refer to both the two types of adversaries as $\mathcal{A}$ in general. Consider that the adversary constructs a first level dummy ciphertext $C_d = (D, E, F, s)$ in the following way under a public key $PK_i$. Let us denote this technique to construct dummy ciphertexts as $Encrypt_{fake}$.

- Compute $Z$ as given in the $Encrypt(ID_i, PK_i, m, params)$ algorithm in the scheme.
- Select $u \in_R \mathbb{Z}_q^*$ and compute $D = (Z)^u$.
- Pick $r \in_R \mathbb{Z}_q^*$ and compute $E = (Z)^r$.
- Choose $F \in_R \{0,1\}^{l_0 + l_1}$.
- Compute $s = u + rH_5(D, E, F) \mod q$.

Note that the computation of $F$ and $r$ in $C_d$ using $Encrypt_{fake}$ violates the definition of the $Encrypt(ID_i, PK_i, m, params)$ algorithm. But $C_d$ clears the ciphertext validity check of equation (1). In fact,

$$\begin{aligned} RHS &= D \cdot E^{H_5(D,E,F)} \\ &= (Z)^u \cdot (Z)^{r \cdot H_5(D,E,F)} \\ &= (Z)^s \\ &= LHS. \end{aligned}$$

The decryption algorithm $Decrypt(ID_i, SK_i, C_d, params)$ detects the ciphertext $C_d$ as invalid and returns $\perp$. However, the $ReEncrypt(ID_i, ID_j, C_d, RK_{i \to j}, params)$ algorithm **accepts** $C_d$ as a valid ciphertext. We use this knowledge to construct a distinguisher for the simulated environment from the real system described stepwise as follows:

1. After the *Challenge* phase, $\mathcal{A}$ generates a dummy ciphertext $C_1 = (D_1, E_1, F_1, s)$ under the target identity $PK_{ch}$ using $Encrypt_{fake}$ as shown:
   - Compute $Z_{ch}$ as shown in the $Encrypt(ID_i, PK_i, m, params)$ scheme.
   - Select $u_1 \in_R \mathbb{Z}_q^*$ and compute $D_1 = (Z_{ch})^{u_1}$.
   - Pick $r_1 \in_R \mathbb{Z}_q^*$ and compute $E_1 = (Z_{ch})^{r_1}$.
   - Choose $F_1 \in_R \{0,1\}^{l_0+l_1}$.
   - Compute $s_1 = u_1 + r_1 H_5(D_1, E_1, F_1) \mod q$.
2. $\mathcal{A}$ generates another dummy ciphertext $C_2 = (D_2, E_2, F_2, s_2)$ in the same way described above considering random values $r_2 \in_R \mathbb{Z}_q^*$ and $F_2 \in_R \{0,1\}^{l_0+l_1}$ similarly.
3. $\mathcal{A}$ queries the re-encryption oracle $\mathcal{O}_{renc}(ID_{ch}, ID_j, C_1, RK_{ch\to j})$. As per $\mathcal{O}_{renc}$, $\mathcal{C}$ searches the $H_4$ list for a tuple of the form $(\langle m, \omega\rangle, r)$ such that $E_1 = (Z_{ch}^r)$. If no such tuple exists, $\mathcal{O}_{renc}$ outputs $\perp$. Note that, on an output $\perp$, $\mathcal{A}$ can distinguish between the simulation and the real system, since $C_1$ is a valid ciphertext as per the definition of $ReEncrypt(ID_{ch}, ID_j, C, RK_{ch\to j}, params)$ algorithm and should produce a valid second level ciphertext $D_1$.
   - If $\mathcal{O}_{renc}$ returns $\perp$, $\mathcal{A}$ aborts.
   - Else, $\mathcal{O}_{renc}$ computes $D_1 = (E_1', F_1, V_1, W_1)$ as specified in the scheme and output $D_1$.
4. Similarly, $\mathcal{A}$ queries the re-encryption oracle $\mathcal{O}_{renc}(ID_{ch}, ID_j, C_2, RK_{ch\to j})$. As per $\mathcal{O}_{renc}$, $\mathcal{C}$ searches the $H_4$ list for a tuple of the form $(\langle m, \omega\rangle, r)$ such that $E_2 = (Z_{ch}^r)$. If no such tuple exists, $\mathcal{O}_{renc}$ outputs $\perp$. Note that, on an output $\perp$, $\mathcal{A}$ can distinguish between the simulation and the real system, since $C_1$ is a valid ciphertext as per the definition of $ReEncrypt(ID_{ch}, ID_j, C, RK_{ch\to j}, params)$ algorithm and should produce a valid second level ciphertext $D_2$.
   - If $\mathcal{O}_{renc}$ returns $\perp$, $\mathcal{A}$ aborts.
   - Else, $\mathcal{O}_{renc}$ computes $D_2 = (E_2', F_2, V_2, W_2)$ as specified in the scheme and output $D_2$.
5. On receiving $D_1$ and $D_2$, $\mathcal{A}$ computes $T_1 = {E_1'}^{\frac{1}{r_1}}$ and $T_2 = {E_2'}^{\frac{1}{r_2}}$.
6. If $T_1 \stackrel{?}{=} T_2$ does not hold, $ReEncrypt(ID_{ch}, ID_j, C, RK_{ch\to j}, params) \neq \mathcal{O}_{renc}$ and $\mathcal{A}$ learns that it is not the real system and aborts. Else, if $T_1 \stackrel{?}{=} T_2$ holds, $\mathcal{A}$ cannot distinguish between the simulated environment and the real system.

Hence, we have pointed out that the adversary (both Type I and Type II) will be able distinguish between the simulation run by the Challenger $\mathcal{C}$ and the real system. This makes the security proof incomplete and makes the scheme provably insecure.

### 3.3 A Possible Fix

The flaw in the scheme can be fixed by modifying the encryption algorithm **Encrypt**$(ID_i, PK_i, m, params)$ along with additional ciphertext validity checks in both the **Re-Encrypt** and the **Decrypt** algorithm. The modified scheme is shown below.

- **Setup**$(1^\lambda)$: The *Setup* algorithm remains the same as in [13] described in Section 3.1. Add another cryptographic hash function to the existing public parameters as defined:

$$\tilde{H} : \mathbb{G}^4 \times \{0,1\}^{l_0+l_1} \to \mathbb{G}$$

  Return the public parameters $params = (p, q, \mathbb{G}, g, y, \tilde{H}, H, H_1, H_2, H_3, H_4, H_5, H_6)$ and the master secret key is $msk = x$, which is generated as decribed in Section 3.1.

- The **PartialKeyExtract**, **UserKeyGen**, **SetPublicKey**, **Public Verify**, **SetPrivateKey** algorithm remains the same described in Section 3.1.

- **Encrypt**$(ID_i, PK_i, m, params)$:
  - Parse $PK_i$ as $(P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{i,1}, T_{i,2}, \mu_{ji,1}, \mu_{i,2})$ and verify the validity of the public key $PK_i$ by checking if **Public Verify**$(ID_i, PK_i) = success$. If the check fails, output $\perp$.
  - Compute $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$ and $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$. 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 $Z = (X(Y)^\alpha)$.
  - Select $u \in_R \mathbb{Z}_q^*$ and $\omega \in_R \{0,1\}^{l_1}$. Compute $r = H_4(m, \omega)$.
  - Compute $D = (Z)^u,\ E = Z^r$.
  - Compute $\bar{D} = \tilde{H}(X, Y, D, E, F)^u,\ \bar{E} = \tilde{H}(X, Y, D, E, F)^r$.
  - Compute $F = H_3(g^r) \oplus (m||\omega)$ and $s = u + rH_5(E, \bar{E}, F)$.

– Return the ciphertext $C = (E, \bar{E}, F, s)$ as the first level ciphertext.

- **Re-Encrypt**$(ID_i, ID_j, C, RK_{i\to j}, params)$:On input a re-encryption key $RK_{i\to j} = (RK_{i\to j}^{\langle 1\rangle}, V, W)$, an original ciphertext $C_i = (E, \bar{E}, F, s)$ encrypted under public key $PK_i = (P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{i,1}, T_{i,2}, \mu_{i,1}, \mu_{i,2})$, re-encrypt $C$ into a ciphertext $D$ under the public key $PK_j = (P_{j,1}, P_{j,2}, Q_{j,1}, Q_{j,2}, Q_{j,3}, S_{j,3}, T_{j,1}, T_{j,2}, \mu_{j,1}, \mu_{j,2})$ as follows:
  – Compute $R_{i,1} = Q_{i,1}(y^{H_1(ID_i, Q_{i,1})})$ and $R_{i,2} = Q_{i,2}(y^{H_1(ID_i, Q_{i,2})})$. 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 $Z = (X(Y)^{\alpha})$.
  – Compute $D$ and $\bar{D}$ as follows:

$$\begin{aligned} D &= (Z)^s \cdot (E^{H_5(E,\bar{E},F)})^{-1} \\ &= Z^u \cdot Z^{r\cdot H_5(E,\bar{E},F)} \cdot Z^{-r\cdot H_5(E,\bar{E},F)} \\ &= (Z)^u. \end{aligned}$$

$$\begin{aligned} \bar{D} &= \tilde{H}(X,Y,D,E,F)^s \cdot (\bar{E}^{H_5(E,\bar{E},F)})^{-1} \\ &= \tilde{H}(X,Y,D,E,F)^u \cdot \tilde{H}(X,Y,D,E,F)^{r\cdot H_5(E,\bar{E},F)} \cdot \tilde{H}(X,Y,D,E,F)^{-r\cdot H_5(E,\bar{E},F)} \\ &= \tilde{H}(X,Y,D,E,F)^u. \end{aligned}$$

  – Check the validity of the ciphertext by performing the following checks.

$$(Z)^s \stackrel{?}{=} D \cdot E^{H_5(E,\bar{E},F)} \tag{2}$$

$$\tilde{H}(X,Y,D,E,F)^s \stackrel{?}{=} \bar{D} \cdot \bar{E}^{H_5(E,\bar{E},F)} \tag{3}$$

  If the check fails, return $\perp$.
  – Else, parse $RK_{i\to j}$ as $(rk, V, W)$ compute $E' = R^{rk}$.
  – Output $D = (E', F, V, W)$ as the second level ciphertext.

- **Decrypt**$(ID_i, SK_i, C, params)$:
  – Obtain the public key $PK_i$ corresponding to $ID_i$ and parse it as $(P_{i,1}, P_{i,2}, Q_{i,1}, Q_{i,2}, Q_{i,3}, S_{i,3}, T_{i,1}, T_{i,2}, \mu_{ji,1}, \mu_{i,2})$. Parse $SK_i$ as $(U_{i,1}, U_{i,2}, S_{i,1}, S_{i,2})$ and $C$ as $(D, E, F, s)$. Check if the ciphertext is well-formed by computing the values of $D$ and $\bar{D}$ and checking if equations 2 and 3 holds. If they do not hold, return $\perp$.
  – Else, compute $R_{i,1}, R_{i,2}, X, Y, \alpha, Z, K$ and retrieve $m$ as described in the $Decrypt(ID_i, SK_i, C, params)$ algorithm in Section 3.1.

- **Re-Decrypt**$(ID_j, SK_j, D, params)$: Same as described in the original scheme.

# 4 Our Proposed Unidirectional CCA-secure CL-PRE Scheme

## 4.1 Our Scheme

- **Setup**$(1^\lambda)$: Given $\lambda$ as the security parameter, choose a group $\mathbb{G}$ of prime order $q$. Let $P$ be a generator of $\mathbb{G}$. Pick $s \in_R \mathbb{Z}_q^*$ and compute $P_{pub} = sP$. Choose cryptographic hash functions :

$$\begin{aligned} \tilde{H} &: \{0,1\}^{l_{ID}} \times \mathbb{G}^2 \times \{0,1\}^{l_0+l_1} \to \mathbb{G} \\ H_1 &: \{0,1\}^{l_{ID}} \times \mathbb{G}^2 \to \mathbb{Z}_q^* \\ H_2 &: \mathbb{Z}_q^* \times \mathbb{Z}_q^* \to \mathbb{Z}_q^* \\ H_3 &: \mathbb{G} \to \mathbb{Z}_q^* \\ H_4 &: \{0,1\}^{l_0} \times \{0,1\}^{l_1} \to \mathbb{Z}_q^* \\ H_5 &: \mathbb{G}^2 \to \{0,1\}^{l_0+l_1} \\ H_6 &: \mathbb{G}^2 \times \{0,1\}^{l_0+l_1} \to \mathbb{Z}_q^* \end{aligned}$$

  where $\{0,1\}^{l_0}$ is the size of the message space $\mathcal{M}$, $l_1$ is determined by the security parameter $\lambda$ and $\{0,1\}^{l_{ID}}$ is the size of the identity of a user.
  Return the public parameters $params = (\mathbb{G}, q, P, P_{pub}, \tilde{H}, H_1, H_2, H_3, H_4, H_5, H_6)$ and master secret key $msk = s$.

- **PartialKeyExtract**$(msk, ID_i, params)$:
  - Choose $x_i, y_i \in_R \mathbb{Z}_q^*$.
  - Compute $X_i = x_i P,\ Y_i = y_i P$.
  - Compute $q_i = H_1(ID_i, X_i, Y_i)$.
  - Compute $d_i = (x_i + q_i s) \mod q$.
  - Return the Partial Public Key $PPK_i = (X_i, Y_i, d_i)$ and the Partial Private Key $PSK_i = y_i$.

- **UserKeyGen**$(ID_i, params)$:
  - Pick $z_i \in_R \mathbb{Z}_q^*$.
  - Compute $Z_i = z_i P$.
  - Return the user private key-public key pair $(USK_i, UPK_i) = (z_i, Z_i)$.

- **SetPrivateKey**$(ID_i, PSK_i, USK_i, params)$: Set the full secret key as $SK_i = \langle z_i, y_i \rangle$.

- **SetPublicKey**$(ID_i, PPK_i, PSK_i, UPK_i, USK_i, params)$: Set the full public key as $PK_i = \langle X_i, Y_i, Z_i, d_i \rangle$.

- **PublicVerify**$(ID_i, PK_i, params)$: We additionally provide public verifiability of the public keys of each user. This is done by the following check:

$$d_i P \stackrel{?}{=} X_i + H_1(ID_i, X_i, Y_i) \cdot Ppub \tag{4}$$

  In fact,

$$\begin{aligned} RHS &= X_i + H_1(ID_i, X_i, Y_i) \cdot Ppub \\ &= x_i P + H_1(ID_i, X_i, Y_i) \cdot sP \\ &= (x_i + s \cdot H_1(ID_i, X_i, Y_i))P \\ &= LHS. \end{aligned}$$

  If the check is satisfied, return *valid*, else return *invalid*.

  *Remark 1.* Our public key verification algorithm $PublicVerify(ID_i, PK_i, params)$ ensures the validity of the public keys, since an adversary can replace the public keys with false keys of its choice.

- **Re-KeyGen**$(ID_i, ID_j, SK_i, PK_j, params)$:
  - Pick $\alpha_{ij}^{(1)}, \beta_{ij}^{(1)} \in_R \mathbb{Z}_q^*$.
  - Compute $\alpha_{ij}^{(2)}$ such that $\alpha_{ij}^{(1)} \cdot \alpha_{ij}^{(2)} = y_i \mod q$.
  - Compute $\beta_{ij}^{(2)}$ such that $\beta_{ij}^{(1)} \cdot \beta_{ij}^{(2)} = z_i \mod q$.
  - Compute $v_{ij} = H_2(\alpha_{ij}^{(2)} || \beta_{ij}^{(2)})$.
  - Compute $V_{ij} = v_{ij} \cdot Y_j$ and $W_{ij} = H_3(v_{ij} P) \oplus (\alpha_{ij}^{(2)} || \beta_{ij}^{(2)})$.
  - Return $RK_{i \to j} = (\alpha_{ij}^{(1)}, \beta_{ij}^{(1)}, V_{ij}, W_{ij})$.

- **Encrypt**$(ID_i, PK_i, m, params)$:
  - Check the validity of the public key of identity $ID_i$ by checking if **PublicVerify(**$ID_i, PK_i, params$**)=***valid*.
  - If *invalid*, return $\perp$.
  - Else, pick $\sigma \in_R \{0,1\}^{l_1},\ u \in_R \mathbb{Z}_q^*$.
  - Compute $r = H_4(m, \sigma) \in \mathbb{Z}_q^*$.
  - Compute the ciphertext $C = (C_1, C_2, C_3, C_4)$ where:
    - Compute $C_1 = rP \in \mathbb{G}$.
    - Compute $\overline{C}_1 = uP \in \mathbb{G}$.
    - Compute $C_2 = r\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) \in \mathbb{G}$.
    - Compute $\overline{C}_2 = u\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) \in \mathbb{G}$.
    - Compute $C_3 = H_5(rY_i, rZ_i) \oplus (m || \sigma) \in \{0,1\}^{l_0 + l_1}$.
    - Compute $C_4 = u + rH_6(C_1, C_2, C_3) \in \mathbb{Z}_q^*$.

* Return $C = (C_1, C_2, C_3, C_4)$.

- **Re-Encrypt**$(ID_i, ID_j, C, RK_{i\to j}, params)$:To verify that $C$ is well-formed, parse $C$ to obtain $(C_1, C_2, C_3, C_4)$ and compute $\overline{C}_1$ and $\overline{C}_2$ as given:

$$\begin{aligned}\overline{C}_1 &= C_4P - H_6(C_1, C_2, C_3) \cdot C_1 \\ &= uP + H_6(C_1, C_2, C_3)rP - H_6(C_1, C_2, C_3) \cdot C_1 \\ &= uP.\end{aligned}$$

$$\begin{aligned}\overline{C}_2 &= C_4 \cdot \tilde{H}(ID_i, C_1, \overline{C}_1, C_3) - H_6(C_1, C_2, C_3) \cdot C_2 \\ &= u\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) + rH_6(C_1, C_2, C_3)\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) - H_6(C_1, C_2, C_3) \cdot C_2 \\ &= u\tilde{H}(ID_i, C_1, \overline{C}_1, C_3).\end{aligned}$$

We verify if the ciphertext is well-formed by performing the following checks:

$$C_4 \cdot P \stackrel{?}{=} \overline{C}_1 + H_6(C_1, C_2, C_3) \cdot C_1 \tag{5}$$

$$C_4 \cdot \tilde{H}(ID_i, C_1, \overline{C}_1, C_3) \stackrel{?}{=} \overline{C}_2 + H_6(C_1, C_2, C_3) \cdot C_2 \tag{6}$$

If verification is successful, do the following computation:
* Parse the re-encryption key $RK_{i\to j}$ to obtain $(\alpha_{ij}^{(1)}, \beta_{ij}^{(1)}, V_{ij}, W_{ij})$.
* Compute $D_1 = \alpha_{ij}^{(1)} \cdot C_1$.
* Compute $D_2 = \beta_{ij}^{(1)} \cdot C_1$.
* Return the re-encrypted ciphertext as $D = (D_1, D_2, D_3, D_4, D_5) = (D_1, D_2, C_3, V_{ij}, W_{ij})$.

- **Decrypt**$(ID_i, SK_i, C, params)$: Verify that $C$ is a valid ciphertext by checking if equations 5 and 6 holds. If satisfied, compute:

$$(m||\sigma) = C_3 \oplus H_5(y_i \cdot C_1, z_i \cdot C_1) \tag{7}$$

- **Re-Decrypt**$(ID_j, SK_j, D, params)$:
  * Compute $(\alpha_{ij}^{(2)}||\beta_{ij}^{(2)}) = W_{ij} \oplus H_3(\frac{1}{y_j}V_{ij})$.
  * Check if $V_{ij} \stackrel{?}{=} H_2(\alpha_{ij}^{(2)}||\beta_{ij}^{(2)}) \cdot Y_j$.
  * If satisfied, output the plaintext as :

$$(m||\sigma) = C_3 \oplus H_5\big(\alpha_{ij}^{(2)} \cdot D_1, \beta_{ij}^{(2)} \cdot D_2\big) \tag{8}$$

### 4.2 Correctness

- Correctness of Ciphertext Verification from equation 5:

$$\begin{aligned}RHS &= \overline{C}_1 + H_6(C_1, C_2, C_3) \cdot C_1 \\ &= uP + H_6(C_1, C_2, C_3) \cdot rP \\ &= (u + r \cdot H_6(C_1, C_2, C_3))P \\ &= C_4 \cdot P \\ &= LHS.\end{aligned}$$

- Correctness of Ciphertext Verification from equation 6:

$$\begin{aligned}RHS &= \overline{C}_2 + H_6(C_1, C_2, C_3) \cdot C_2 \\ &= u\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) + rH_6(C_1, C_2, C_3)\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) \\ &= (u + rH_6(C_1, C_2, C_3))\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) \\ &= C_4 \cdot \tilde{H}(ID_i, C_1, \overline{C}_1, C_3) \\ &= LHS.\end{aligned}$$

- Consistency between Encryption and Decryption from equation 7:

$$\begin{aligned} RHS &= C_3 \oplus H_5(y_i \cdot C_1, z_i \cdot C_1) \\ &= H_5(r \cdot Y_i, r \cdot Z_i) \oplus (m||\sigma) \oplus H_5(y_i \cdot rP, z_i \cdot rP) \\ &= H_5(r \cdot Y_i, r \cdot Z_i) \oplus (m||\sigma) \oplus H_5(r \cdot Y_i, r \cdot Z_i) \\ &= LHS. \end{aligned}$$

- Consistency between Re-Encryption and Re-Decryption from equation 8:

$$\begin{aligned} RHS &= C_3 \oplus H_5(\alpha_{ij}^{(2)} \cdot D_1, \beta_{ij}^{(2)} \cdot D_2) \\ &= H_5(r \cdot Y_i, r \cdot Z_i) \oplus (m||\sigma) \oplus H_5(\alpha_{ij}^{(2)} \cdot \alpha_{ij}^{(2)} C_1, \beta_{ij}^{(2)} \cdot \beta_{ij}^{(1)} C_1) \\ &= H_5(r \cdot Y_i, r \cdot Z_i) \oplus (m||\sigma) \oplus H_5(y_i \cdot C_1, z_i \cdot C_1) \\ &= H_5(r \cdot Y_i, r \cdot Z_i) \oplus (m||\sigma) \oplus H_5(r \cdot Y_i, r \cdot Z_i) \\ &= LHS. \end{aligned}$$

### 4.3 Security Proof

**First-level Ciphertext Security against Type I adversary :**

**Theorem 1.** *Our proposed scheme is CCA-secure against Type-I adversary for the first level ciphertext under the CDH assumption and the $EUF-CMA$ security of Schnorr signature scheme [11]. If a $(t, \epsilon) IND-CLPRE-CCA$ Type-I adversary $\mathcal{A}_I$ with an advantage $\epsilon$ breaks the IND-CLPRE-CCA security of the given scheme, $\mathcal{C}$ can solve the CDH problem with advantage $\epsilon'$ within time $t'$ where:*

$$\epsilon' \geq \frac{1}{q_{H_5}} \left( \frac{(1-\omega)^{1+q_{rk}} \epsilon}{e(q_{ppe}+1)} - \frac{q_{H_4}}{2^{l_0+l_1}} - \frac{q_{H_6}}{2^{l_0+l_1}} - \frac{q_{\tilde{H}}}{2^{l_0+l_1}} - q_{dec} \Big( \frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q} \Big) \right)$$

*where $\omega$ is the advantage of an attacker against the EUF-CMA security game of the Schnorr signature scheme and $e$ is the base of the natural logarithm. Time taken by $\mathcal{C}$ to solve the CDH problem is:*

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

*where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$. We denote the time taken for exponentiation operation in group $\mathbb{G}$ as $t_{exp}$.*

*Proof.* If a Type-I adversary $\mathcal{A}_I$ for the first level ciphertext with access to the random oracles $\tilde{H}, H_1, H_2, H_3, H_4, H_5$, $H_6$ breaks the IND-CLPRE-CCA security of the given scheme, we can construct an algorithm $\mathcal{C}$ which breaks the CDH problem given the instance $(P, aP, bP)$.

- **Initialisation:** $\mathcal{C}$ sets $P_{pub} = sP$, where $sP$ is the public key of the Schnorr Signature Scheme. $\mathcal{C}$ maintains a list $P_{current}$ of the public keys to keep a track of the replaced public keys. $P_{current}$ consists of tuples of the form $\langle ID_i, PK_i, \hat{PK_i} \rangle$, where $\hat{PK_i}$ denotes the current value of the public key of $ID_i$. $\mathcal{C}$ maintains a list of all the keys in the following lists:
  1. $L_{pub}$ : Maintains a public key list with tuples of the form $\langle ID_i, PK_i, c_i \rangle$. Note that $c_i$ is used to identify the corrupt and honest public keys respectively, as explained later.
  2. $L_{part}$ : Maintains a partial key list with tuples of the form $\langle ID_i, PPK_i, PSK_i, c_i \rangle$.
  3. $L_{user}$ : Maintains a user key list with tuples of the form $\langle ID_i, UPK_i, USK_i, c_i \rangle$.
  4. $L_{priv}$ : Maintains a private key list with tuples of the form $\langle ID_i, SK_i, c_i \rangle$.
  5. $L_{rekey}$: Maintains a list of re-encryption keys $RK_{i \to j}$, consisting of tuples of the form $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V_{ij}, W_{ij} \rangle$.
  6. $L_{schnorr-PK}$ : Maintains a Schnorr public key list with tuples of the form $\langle ID_i, X_i, Y_i, m_i, \psi_i \rangle$.

- **Phase 1:** $\mathcal{C}$ interacts with $\mathcal{A}_I$ in the following ways:
  - *Oracle Queries*:
  -$\tilde{H}(ID_i, C_1, \overline{C}_1, C_3)$ *Oracle*: $\mathcal{C}$ responds to the queries of $\mathcal{A}_I$ by maintaining a list $L_{\tilde{H}}$ with tuples of the form $\langle ID_i \in \{0,1\}^{l_{ID}},\ C_1 \in \mathbb{G},\ \overline{C}_1 \in \mathbb{G},\ C_3 \in \mathbb{Z}_q^*,\ h_i \in \mathbb{Z}_q^*,\ \gamma_i \in \mathbb{G} \rangle$. If the tuple $\langle ID_i, C_1, \overline{C}_1, C_3, h_i, \gamma_i \rangle$ already exists in $L_{\tilde{H}}$, retrieve and return $\tilde{H}(ID_i, C_1, \overline{C}_1, C_3) = \gamma_i$. Else, $\mathcal{C}$ retrieves the tuple $\langle ID_i, PK_i, c_i \rangle$ from $L_{pub}$ list and computes $\tilde{H}(ID_i, C_1, \overline{C}_1, C_3)$ according to the following cases:

· If $c_i = 0$, $\mathcal{C}$ picks $h_i \in_R \mathbb{Z}_q^*$ and sets $\gamma_i = h_i P$.
· If $c_i = 1$, $\mathcal{C}$ picks $h_i \in_R \mathbb{Z}_q^*$ and sets $\gamma_i = h_i aP$.

*-$H_1(ID_i, X_i, Y_i)$ Oracle*: $\mathcal{C}$ responds to the queries of $\mathcal{A}_I$ by maintaining a list $L_{H_1}$ with tuples of the form $\langle ID_i \in \{0,1\}^*,\ X_i \in \mathbb{G},\ Y_i \in \mathbb{G},\ k \in \mathbb{Z}_q^* \rangle$. If the tuple $\langle ID_i, X_i, Y_i, k \rangle$ already exists in $L_{H_1}$, retrieve and return the value $k$. Else do the following:
· Check for the existence of the tuple $\langle ID_i, PK_i, \hat{PK_i} \rangle$ in the list $P_{current}$ and the tuple $\langle ID_i, PK_i, c_i \rangle$ in $L_{pub}$.
· If $\hat{PK_i} = PK_i$ in $P_{current}$, pick $k \in_R \mathbb{Z}_q^*$ and return $H_1(ID_i, X_i, Y_i) = k$.
· Else if $\hat{PK_i} \neq PK_i$ in $P_{current}$ and $c_i = 1 \in L_{pub}$, select $m_i \in \mathbb{Z}_q^*$. Check if there exists a tuple $\langle ID_i, X_i, Y_i, m_i, \psi_i \rangle$ in list $L_{schnorr-pk}$. If exists, repeat by considering fresh value of $m_i$.
· Query the Schnorr hash $H_{Schnorr}$ for $\psi_i = H_{Schnorr}(m_i, X_i)$.
· Update list $L_{schnorr-pk}$ with the tuple $\langle ID_i, X_i, Y_i, m_i, \psi_i \rangle$.
· Return $k = \psi_i$ as the value of $H_1(ID_i, X_i, Y_i)$ and store tuple $\langle ID_i, X_i, Y_i, k \rangle$ in $L_{H_1}$.

*-$H_2(\alpha_{ij}^{(2)} || \beta_{ij}^{(2)})$ Oracle*: $\mathcal{C}$ maintains a list $L_{H_2}$ with tuples of the form $\langle \alpha_{ij}^{(2)} \in \mathbb{Z}_q^*,\ \beta_{ij}^{(2)} \in \mathbb{Z}_q^*,\ e \in \mathbb{Z}_q^* \rangle$. If the tuple $\langle \alpha_{ij}^{(2)}, \beta_{ij}^{(2)}, e \rangle$ already exists in $L_{H_2}$, retrieve and return the value $e$. Else, choose $e \leftarrow \mathbb{Z}_q^*$, set $H_2(\alpha_{ij}^{(2)} || \beta_{ij}^{(2)}) = e$, store tuple $\langle \alpha_{ij}^{(2)}, \beta_{ij}^{(2)}, e \rangle$ to $L_{H_2}$ and return $e$.

*-$H_3(R)$ Oracle*: $\mathcal{C}$ maintains a list $L_{H_3}$ with tuples of the form $\langle R \in \mathbb{G},\ n \in \mathbb{Z}_q^* \rangle$. If the tuple $\langle R, n \rangle$ already exists in $L_{H_3}$, retrieve and return the value $n$. Else, choose $n \leftarrow \mathbb{Z}_q^*$, set $H_3(R) = n$, store the tuple $\langle R, n \rangle$ to $L_{H_3}$ and return $n$.

*-$H_4(m, \sigma)$ Oracle*: $\mathcal{C}$ maintains a list $L_{H_4}$ with tuples of the form $\langle m \in \{0,1\}^{l_0},\ \sigma \in \{0,1\}^{l_1},\ r \in \mathbb{Z}_q^* \rangle$. If the tuple $\langle m, \sigma, r \rangle$ already exists in $L_{H_4}$, retrieve and return the value $r$. Else, choose $r \leftarrow \mathbb{Z}_q^*$, set $H_4(m, \sigma) = r$, store the tuple $\langle m, \sigma, r \rangle$ to $L_{H_4}$ and return $r$.

*-$H_5(S, U)$ Oracle*: $\mathcal{C}$ maintains a list $L_{H_5}$ with tuples of the form $\langle S \in \mathbb{G},\ U \in \mathbb{G},\ u \in \mathbb{Z}_q^* \rangle$. If the tuple $\langle S, U, u \rangle$ already exists in $L_{H_5}$, retrieve and return the value $u$. Else, choose $u \leftarrow \mathbb{Z}_q^*$, set $H_5(S, U) = u$, store the tuple $\langle S, U, u \rangle$ to $L_{H_5}$ and return $u$.

*-$H_6(C_1, C_2, C_3)$ Oracle*: $\mathcal{C}$ maintains a list $L_{H_6}$ with tuples of the form $\langle C_1 \in \mathbb{G},\ C_2 \in \mathbb{G},\ C_3 \in \{0,1\}^{l_0+l_1},\ t \in \mathbb{Z}_q^* \rangle$. If the tuple $\langle C_1, C_2, C_3, t \rangle$ already exists in $L_{H_6}$, retrieve and return the value $t$. Else, choose $t \leftarrow \mathbb{Z}_q^*$, set $H_6(C_1, C_2, C_3) = t$, store the tuple $\langle C_1, C_2, C_3, t \rangle$ to $L_{H_6}$ and return $t$.

∗ *Public Key Extract Query* ($\mathcal{O}_{pe}(ID_i)$): $\mathcal{C}$ generates the keys using Coron's coin-tossing technique [8] by tossing a biased coin $c_i$ which takes the value $c_i \in \{0,1\}$. The probability that the coin takes the value 0 is $Pr[c_i = 0] = \theta$, which is to be determined later. The Coron's technique is used to implant the hard problem instance into the *partial keys of the honest users*.
If $c_i = 0$, $\mathcal{C}$ chooses $x_i, y_i, d_i \in_R \mathbb{Z}_q^*$. It computes $X_i = x_i \cdot P_{pub} + d_i P$, $Y_i = y_i P$, and sets $H_1(ID_i, X_i, Y_i) = -x_i$. It sets the user secret key by picking $z_i \in \mathbb{Z}_q^*$ and setting $Z_i = z_i P$. Note that the public key generated by $\mathcal{C}$ satisfies the public verifiability in equation 4 as:

$$\begin{aligned} RHS &= X_i + H_1(ID_i, X_i, Y_i) \cdot P_{Pub} \\ &= x_i \cdot sP + d_i P + (-x_i) \cdot sP \\ &= d_i P \\ &= LHS. \end{aligned}$$

If $c_i = 1$, $\mathcal{C}$ chooses $x_i, \bar{y}_i, d_i \in_R \mathbb{Z}_q^*$. It computes $X_i = x_i \cdot P_{pub} + d_i P$, $Y_i = y_i P$ where $y_i = a\bar{y}_i$ and sets $H_1(ID_i, X_i, Y_i) = -x_i$. It sets the user secret key by picking $z_i \in \mathbb{Z}_q^*$ and setting $Z_i = z_i P$. It is easy to follow that, similar to the keys generated for $c_i = 0$, the public key generated by $\mathcal{C}$ satisfies the public verifiability.
Update the above given lists $L_{pub}, L_{part}, L_{user}, L_{priv}$ with the new key values by adding tuple $\langle ID_i, PK_i, \hat{PK_i} = PK_i \rangle$ to list $P_{current}$, $\langle ID_i, PK_i = (X_i, Y_i, Z_i, d_i), c_i \rangle$ to public key list $L_{pub}$, tuple $\langle ID_i, PPK_i, PSK_i, c_i \rangle$ to partial key list $L_{part}$, tuple $\langle ID_i, UPK_i, USK_i, c_i \rangle$ to user key list $L_{user}$ and tuple $\langle ID_i, SK_i, c_i \rangle$ to private key list $L_{priv}$.

* *Partial Key Extract Query* ($\mathcal{O}_{ppe}(ID_i)$): When $\mathcal{A}_I$ queries for the partial key for user $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ searches the list $L_{part}$ for the existence of a tuple of the form $\langle ID_i, PPK_i, PSK_i, c_i\rangle$. If found, $\mathcal{C}$ checks the value of the coin $c_i$. If $c_i = 1$, $\mathcal{C}$ aborts and reports failure. If $c_i = 0$, return the partial keys $(PPK_i, PSK_i)$ to $\mathcal{A}_I$. If such a tuple does not exist, $\mathcal{C}$ generates the partial keys using the Coron's coin tossing technique as described in the *Public Key Extract Query* $\mathcal{O}_{pe}$.
  If $c_i = 0$, $\mathcal{C}$ calls the *Public Key Extract Query* oracle to generate the keys which updates the lists $L_{pub}, L_{part}, L_{user}$ and $L_{priv}$. $\mathcal{C}$ returns $(PPK_i, PSK_i)$.
  If $c_i = 1$, $\mathcal{C}$ aborts and reports failure.

* *User Key Extract Query* ($\mathcal{O}_{ue}(ID_i)$): When $\mathcal{A}_I$ queries for the user key for user $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ searches the list $L_{user}$ to check the existence of a tuple $\langle ID_i, UPK_i, USK_i, c_i\rangle$ and returns the user keys $(USK_i, UPK_i)$ to $\mathcal{A}_I$. If such a tuple does not exist, $\mathcal{C}$ sets the user secret key by calling the *Public Key Extract Query* oracle to generate the keys which updates the lists $L_{pub}, L_{part}, L_{user}$ and $L_{priv}$. $\mathcal{C}$ returns $(USK_i, UPK_i)$.

* *Re − Key Generation Query* ($\mathcal{O}_{rk}(ID_i, ID_j)$): When $\mathcal{A}_I$ sends a re-key generation query from user $ID_i$ to $ID_j$, $\mathcal{C}$ computes the re-encryption key according to the following cases:
  - Check if the re-encryption key $RK_{i\to j}$ already exists by searching for a tuple $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V_{ij}, W_{ij}\rangle$ in $L_{rekey}$. If present, return $RK_{i\to j}$.
  - If $c_i = 0$, compute $RK_{i\to j}$ by running the *Re-KeyGen*$(ID_i, ID_j, SK_i, PK_j, params)$ algorithm. Add to list $L_{rekey}$ the tuple $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V_{ij}, W_{ij}\rangle$.
  - If $(c_i = 1 \wedge c_j = 1)$, choose $\alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)} \in_R \mathbb{Z}_q^*$. Retrieve the user secret key $USK_i = z_i$ from the list $L_{user}$ and compute $\beta_{ij}^{(2)}$ such that $\beta_{ij}^{(1)} \cdot \beta_{ij}^{(2)} = z_i \mod q$. Compute $v_{ij} = H_2(\alpha_{ij}^{(2)}||\beta_{ij}^{(2)})$, $V_{ij} = v_{ij} \cdot Y_j$, $W_{ij} = H_3(v_{ij}P) \oplus (\alpha_{ij}^{(2)}||\beta_{ij}^{(2)})$. Return $RK_{i\to j}$ to $\mathcal{A}_I$ and add the tuple $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V_{ij}, W_{ij}, \rangle$ in list $L_{rekey}$.

  *Remark 2.* Note that querying for a re-encryption key for $(c_i = 1 \wedge c_j = 0)$ is a violation of our security model. This is because, the corrupt user $ID_j$ receives $\alpha_{ij}^{(1)}$ as a part of the re-encryption key $RK_{i\to j}$ and can also acquire $\alpha_{ij}^{(2)}$ from $V_{ij}$ and $W_{ij}$ as demonstrated in the *Re − Decrypt*$(ID_j, SK_j, D, params)$ algorithm. Consequently, with the possession of $\alpha_{i1}$ and $\alpha_{i2}$, user $ID_j$ can compute the private key of the honest user $ID_i$ as $y_i = \alpha_{i1} \cdot \alpha_{i2}$, which compromises the privacy of $ID_i$.

* *Re − Encryption Query* ($\mathcal{O}_{re}(ID_i, ID_j, C)$): When $\mathcal{A}_I$ queries decryption of a first-level ciphertext $C$ from $ID_i$ to $ID_j$, the Challenger $\mathcal{C}$ first computes $\overline{C}_1$ and $\overline{C}_2$ and checks if equations 5 and 6 hold to verify the well-formedness of the ciphertext $C$. If the validation fails, it outputs $\perp$. Else, $\mathcal{C}$ performs re-encryption according to the following cases:
  - If $c_i = 0$ or $(c_i = 1 \wedge c_j = 1)$, $\mathcal{C}$ computes the second level ciphertext $D$ by obtaining the re-encryption keys $RK_{i\to j}$ by running the *Re − Key Generation* oracle $\mathcal{O}_{rk}$ and calling the *Re-Encrypt*$(ID_i, ID_j, C, RK_{i\to j}, params)$ algorithm.
  - If $(c_i = 1 \wedge c_j = 0)$, $\mathcal{C}$ picks $\alpha_{ij}^{(1)}$, $\beta_{ij}^{(1)} \in_R \mathbb{Z}_q^*$ and sets $\alpha_{ij}^{(2)} = \bar{y}_i \cdot (\alpha_{ij}^{(1)})^{-1}$. $\mathcal{C}$ retrieves the user secret key $USK_i = z_i$ from the list $L_{user}$ and compute $\beta_{ij}^{(2)}$ such that $\beta_{ij}^{(1)} \cdot \beta_{ij}^{(2)} = z_i \mod q$. $\mathcal{C}$ computes $v_{ij} = H_2(\alpha_{ij}^{(2)}||\beta_{ij}^{(2)})$, $V_{ij} = v_{ij} \cdot Y_j$ and $W_{ij} = H_3(v_{ij}P) \oplus (\alpha_{ij}^{(2)}||\beta_{ij}^{(2)})$ and updates $L_{rekey}$ with $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V_{ij}, W_{ij}, \rangle$. $\mathcal{C}$ computes $D_1 = \alpha_{ij}^{(1)} h_i^{-1} C_2 = \alpha_{ij}^{(1)} raP$, $D_2 = \beta_{ij}^{(1)} C_1$, $D_3 = C_3$, $D_4 = V_{ij}$, $D_5 = W_{ij}$. $\mathcal{C}$ returns the second-level ciphertext $D = (D_1, D_2, D_3, D_4)$ to $\mathcal{A}_I$.

* *Decryption Query* ($\mathcal{O}_{dec}(ID_i, C)$): For decrypting a first level ciphertext $C$ under identity $ID_i$ whose public key has not been replaced (if replaced, $\mathcal{A}$ should provide with value of $sk_i = z_i$ of $ID_i$), the Challenger $\mathcal{C}$ first computes $\overline{C}_1$ and $\overline{C}_2$ and checks if equations 5 and 6 holds to verify the well-formedness of the ciphertext $C$. If it holds and $c_i = 0$, $\mathcal{C}$ runs the *Decrypt*$(ID_i, SK_i, C, params)$ algorithm to retrieve the plaintext $m$. Else, if $c_i = 1$, $\mathcal{C}$ retrieves the value of $sk_i = z_i$ of $ID_i$ from $L_{user}$. $\mathcal{C}$ computes $\bar{y}_i h_i^{-1} C_2 = \bar{y}_i h_i^{-1} r h_i aP = rY_i$ to obtain the value of $rY_i$ and retrieves the plaintext $m$ by computing $C_3 \oplus H_5(rY_i, z_i C_1) = (m||\sigma)$. It computes $r = H_4(m, \sigma)$, computes the values of $\overline{C}_1$ and $\overline{C}_2$ as shown in *Re-Encrypt*$(ID_i, ID_j, C, RK_{i\to j}, params)$ algorithm and performs the following checks:
  - $C_1 \stackrel{?}{=} rP$.

· $C_2 = r\tilde{H}(ID_i, C_1, \overline{C}_1, C_3)$.
· $C_3 \stackrel{?}{=} H_5(rY_i, rZ_i) \oplus (m||\sigma)$.
If all the checks are satisfied, $\mathcal{C}$ returns $m$.

* *Re − Decryption Query* ($\mathcal{O}_{redec}(ID_j, C)$): For decrypting a second level ciphertext under identity $ID_j$, $\mathcal{C}$ checks if $c_j = 0$ and the public key has not been replaced. If so, it runs the $Re{-}Decrypt(ID_j, SK_j, D, params)$ algorithm to retrieve the plaintext. Else, if $c_j = 1$, $\mathcal{C}$ searches lists $L_{H_2}$ and $L_{H_3}$ for tuples $\langle \alpha_{ij}^{(2)}, \beta_{ij}^{(2)}, e\rangle$ and $\langle R, n\rangle$ respectively such that: $D_4 \stackrel{?}{=} e \cdot Y_j$, $D_5 \stackrel{?}{=} n \oplus (\alpha_{ij}^{(2)}||\beta_{ij}^{(2)})$ and $R \stackrel{?}{=} eP$. Check the re-key list $L_{rk}$ for the existence of a tuple of the form $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V = D_4, W = D_5\rangle$. Compute $C_1 = (\alpha_{ij}^{(1)})^{-1} \cdot D_1$ and check $L_{H_4}$ and $L_{H_5}$ for tuples $\langle m, \sigma, r\rangle$ and $\langle rY_i, rZ_i, u\rangle$ respectively such that:

$$C_1 \stackrel{?}{=} rP$$
$$C_3 \stackrel{?}{=} u \oplus (m||\sigma).$$

If all the checks are satisfied, $\mathcal{C}$ returns $m$.

* *Public Key Replacement Query* ($\mathcal{O}_{rep}(ID_i, PK_i)$): When $\mathcal{A}_I$ wants to replace the public key of identity $ID_i$, $\mathcal{C}$ checks if the public key is valid by verifying equation 4. If the check fails, it outputs $\perp$.

- **Challenge:** Once $\mathcal{A}_I$ decides that Phase-1 is over, it outputs two messages $m_0, m_1$ and the target identity $ID_{ch}$ on which it wishes to be challenged. $\mathcal{C}$ checks if $c_{ch} = 1$, else it aborts. It tosses a coin and chooses $\delta \in \{0, 1\}$ uniformly at random. It then simulates the challenge ciphertext for $m_\delta$ in the following steps:
  1. Pick $\sigma^* \in_R \{0,1\}^{l_1}$, $\bar{u}, f \in_R \mathbb{Z}_q^*$ and implicitly define $H_4(m_\delta, \sigma) = b$.
  2. Compute $C_1^* = bP = rP$.
  3. Set $u \triangleq \bar{u} - bf$. Compute $\overline{C}_1^* = \bar{u}P - fbP = uP$.
  4. C intends to set $\tilde{H}(ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*) = \gamma_{ch} = h_{ch}P$, where $h_{ch} \in_R \mathbb{Z}_q^*$ and include the tuple $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma_{ch}\rangle$ to $L_{\tilde{H}}$ list. $\mathcal{C}$ checks if a tuple of the form $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h'_{ch}, \gamma_{ch}\rangle$ or $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma'_{ch}\rangle$ already exists in the $L_{\tilde{H}}$ list. If it exists, including $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma_{ch}\rangle$ to the $L_{\tilde{H}}$ list is incorrect and hence GOTO step 1 and re-compute with fresh random values. If no such tuple exists in list $L_{\tilde{H}}$, $\mathcal{C}$ stores the tuple $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma_{ch}\rangle$ in $L_{\tilde{H}}$ list.
  5. Compute $C_2^* = h_{ch}bP = bh_{ch}P = r \cdot \tilde{H}(ID_{ch}, C_1, \overline{C}_1, C_3)$.
  6. Pick $C_3^* \in_R \{0,1\}^{l_0, l_1}$ and define $H_5(y_{ch}^- \cdot abP, z_{ch}bP) = C_3^* \oplus (m_\delta||\sigma^*)$
  7. Check in $L_{H_6}$ if the tuple $\langle C_1^*, C_2^*, C_3^*, t\rangle$ already exists. If so, GOTO step 1 and recompute with fresh random values.
  8. Define $H_6(C_1^*, C_2^*, C_3^*) = f$ . Set $C_4^* = \bar{u}$. From the definition of $u$, we have $C_4^* = u + bf = u + rH_6(C_1^*, C_2^*, C_3^*)$.

  Note that the challenge ciphertext is identically distributed to the real ciphertext generated by the encryption algorithm. Hence, the challenge ciphertext is a valid ciphertext.

- **Phase-2:** The adversary $\mathcal{A}_I$ continues to query the oracles maintained by $\mathcal{C}$ with the restrictions stated in the security model.

- **Guess:** The adversary $\mathcal{A}_I$ eventually produces its guess $\delta' \in \{0, 1\}$ to $\mathcal{C}$. $\mathcal{C}$ randomly picks a tuple $\langle S, U, u\rangle$ from $H_5$ list and outputs $\bar{y}_{ch}S$ as the solution to the CDH instance.

- **Probability Analysis:** Our analysis closely follows the probability analysis of the simulations given in [7]. We first analyse the probability with which the Challenger $\mathcal{C}$ aborts during the simulation, which can occur during the *partial key extraction query*, $re - key$ *generation query* or the *challenge* phase. In the context of these two occurrences, $\mathcal{C}$ does not abort in case of the following two events:
  – $E_1$ : In the *partial key extract* query, $c_i = 0$.
  – $E_2$ : In the $re - key$ *generation* query, the public keys of the users $ID_i$ and $ID_j$ have not been replaced.
  – $E_3$ : In the *Challenge* phase, $c_{ch} = 1$ and the public keys of the target user $ID_{ch}$ has not been replaced.

  Note that, a valid public key replacement indicates that the Schnorr signature has been compromised. We denote the advantage of an attacker in breaking the Schnorr signature as $\omega$. Therefore, the probability that $\mathcal{C}$

does not abort $Pr[\neg abort] = \theta^{q_{ppe}}(1-\theta)(1-\omega)^{1+q_{rk}}$, which has a maximum value at $\theta_{OPT} = \frac{q_{ppe}}{q_{ppe}+1}$. Using $\theta_{OPT}$, we obtain:

$$Pr[\neg abort] = \frac{(1-\omega)^{1+q_{rk}}}{e(q_{ppe}+1)}.$$

Next, we analyse the simulation of the random oracles. The simulation of the random oracles takes place perfectly unless the following events take place:

- $E_{H_4^*}$: Event that $(m_\delta, \sigma^*)$ was queried to the $H_4$ hash function.
- $E_{H_5^*}$: Event that $(y_{ch}^{-}abP, z_{ch}bP)$ was queried to $H_5$.
- $E_{H_6^*}$: Event that $(C_1^*, C_2^*, C_3^*)$ was queried to $H_6$ before the Challenge phase.
- $E_{\tilde{H}^*}$: Event that $(ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*)$ was queried to $\tilde{H}$ before the Challenge phase.

We derive the probabilities of the above events. Since $C_3^*$ is chosen at random from $\{0,1\}^{l_0+l_1}$, we have $Pr[E_{H_6^*}] \leq \frac{q_{H_6}}{\{0,1\}^{l_0+l_1}}$. Similarly, we have $Pr[E_{\tilde{H}^*}] \leq \frac{q_{\tilde{H}}}{\{0,1\}^{l_0+l_1}}$.

We further analyse the simulation of the re-decryption oracle. Note that the simulation of the decryption oracle runs correctly unless valid ciphertexts are rejected, which occurs when $\mathcal{A}_I$ queries the re-decryption oracle without querying $H_4$ and $H_5$. Let $E_{valid}$ denote the event that the ciphertext is a valid ciphertext. Let $E_{H_4}$ denote the event that $(m, \sigma)$ has been queried to $H_4$ and $E_5$ denote the event that $(rY_i, rZ_i)$ has been queried to $H_5$. We have $Pr[E_{valid}|(\neg E_{H_4} \vee \neg E_{H_5})] \leq Pr[E_{valid}|\neg E_{H_4}] + Pr[E_{valid}|\neg E_{H_5}]$.

$$\begin{aligned}
Pr[E_{valid}|\neg E_{H_4}] &= Pr[E_{valid} \wedge E_{H_5}|\neg E_{H_4}] + Pr[E_{valid} \wedge \neg E_{H_5}|\neg E_{H_4}] \\
&\leq Pr[E_{valid} \wedge E_{H_5}|\neg E_{H_4}] + Pr[E_{valid}|\neg E_{H_5} \wedge \neg E_{H_4}] \\
&= \frac{Pr[E_{valid} \wedge E_{H_5} \wedge E_{H_4}]}{Pr[\neg E_{H_4}]} + \frac{1}{q} \\
&\leq \frac{Pr[E_{H_5}]}{Pr[\neg E_{H_4}]} + \frac{1}{q} \\
&\leq \frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{1}{q}.
\end{aligned}$$

With a similar analysis, we obtain $Pr[E_{valid}|\neg E_{H_5}] \leq \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{1}{q}$.

We obtain:

$$\begin{aligned}
Pr[E_{valid}|(\neg E_{H_4} \vee \neg E_{H_5})] &\leq Pr[E_{valid}|\neg E_{H_4}] + Pr[E_{valid}|\neg E_{H_5}] \\
&\leq \frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}.
\end{aligned}$$

Let us denote $E_{dec}$ denote that the event $E_{valid}|(\neg E_{H_4} \vee \neg E_{H_5})$ occurs during the entire simulation, and we obtain:

$$Pr[E_{dec}] \leq q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)$$

Let $E_{er}$ denote the event $(E_{H_4^*} \vee E_{H_5^*} \vee E_{H_6^*} \vee E_{\tilde{H}^*} \vee E_{dec})|\neg abort$. If $E_{er}$ does not occur, the adversary $\mathcal{A}_I$ does not gain any advantage greater than $\frac{1}{2}$ in guessing $\delta$ due to the randomness in the output of $H_5$ oracle. Therefore, $Pr[\delta' = \delta|\neg E_{er}] = \frac{1}{2}$. Note that:

$$\begin{aligned}
Pr[\delta' = \delta] &= Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] + Pr[\delta' = \delta|E_{er}]Pr[E_{er}] \\
&\leq 1/2 Pr[\neg E_{er}] + Pr[E_{er}] = 1/2 + 1/2 Pr[E_{er}].
\end{aligned}$$

Also,

$$Pr[\delta' = \delta] \geq Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] \geq 1/2 - 1/2 Pr[E_{er}].$$

By the definition of the advantage of IND-CLPRE-CCA adversary, we have the advantage:

$$\begin{aligned}
\epsilon &= |2Pr[\delta' = \delta] - 1| \\
&\leq Pr[E_{er}] = Pr[(E_{H_4^*} \vee E_{H_5^*} \vee E_{H_6^*} \vee E_{\tilde{H}^*} \vee E_{dec})|\neg abort] \\
&\leq \frac{Pr[E_{H_4^*}] + Pr[E_{H_5^*}] + Pr[E_{H_6^*}] + Pr[E_{\tilde{H}^*}] + Pr[E_{dec}]}{Pr[\neg abort].}
\end{aligned}$$

Therefore, we obtain the following bound on $Pr[E_{H_5^*}]$ as:

$$\begin{aligned} Pr[E_{H_5^*}] &\geq Pr[\neg abort] \cdot \epsilon - Pr[E_{H_4^*}] - Pr[E_{H_6^*}] - Pr[E_{\tilde{H}^*}] - Pr[E_{dec}] \\ &\geq \frac{(1-\omega)^{1+q_{rk}}\epsilon}{e(q_{ppe}+1)} - \frac{q_{H_4}}{2^{l_0+l_1}} - \frac{q_{H_6}}{2^{l_0+l_1}} - \frac{q_{\tilde{H}}}{2^{l_0+l_1}} - q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big) \end{aligned}$$

Note that, if event $E_{H_5^*}$ occurs, then the challenger $\mathcal{C}$ solves the $CDH$ instance with advantage:

$$\begin{aligned} \epsilon' &\geq \frac{1}{q_{H_5}} Pr[E_{H_5^*}] \\ &\geq \frac{1}{q_{H_5}}\Bigg(\frac{(1-\omega)^{1+q_{rk}}\epsilon}{e(q_{ppe}+1)} - \frac{q_{H_4}}{2^{l_0+l_1}} - \frac{q_{H_6}}{2^{l_0+l_1}} - \frac{q_{\tilde{H}}}{2^{l_0+l_1}} - q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)\Bigg) \end{aligned}$$

We bound the time taken by $\mathcal{C}$ using $t'$:

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$ and $t_{exp}$ denotes the time taken for exponentiation operation in group $\mathbb{G}$.

□

**Second-level Ciphertext Security against Type I adversary** :

**Theorem 2.** *Our proposed scheme is CCA-secure against Type-I adversary for the second level ciphertext under the CDH assumption and the $EUF-CMA$ security of the Schnorr signature scheme. If a $(t,\epsilon)IND-CLPRE-CCA$ Type-I adversary $\mathcal{A}_I$ with an advantage $\epsilon$ breaks the IND-CLPRE-CCA security of the given scheme, $\mathcal{C}$ can solve the CDH problem with advantage $\epsilon'$ within time $t'$ where:*

$$\epsilon' \geq \frac{1}{q_{H_5}}\Bigg(\frac{2(1-\omega)^{2+q_{rk}}\epsilon}{e(q_{ppe}+2)^2} - q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)\Bigg)$$

*where $\omega$ is the advantage of an attacker against the EUF-CMA security game of the Schnorr signature scheme and $e$ is the base of the natural logarithm. Time taken by $\mathcal{C}$ to solve the CDH problem is:*

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

*where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$. We denote the time taken for exponentiation operation in group $\mathbb{G}$ as $t_{exp}$.*

*Proof.* If a Type-I adversary $\mathcal{A}_I$ for the second level ciphertext with access to the random oracles $\tilde{H}, H_1, H_2, H_3, H_4, H_5, H_6$ breaks the IND-CLPRE-CCA security of the given scheme, we can construct an algorithm $\mathcal{C}$ which breaks the CDH problem given the instance $(P, aP, bP)$.

- **Initialisation:** $\mathcal{C}$ sets $P_{pub} = sP$, where $sP$ is the public key of the Schnorr Signature. Similar to **Theorem 1**, $\mathcal{C}$ maintains a list of all the keys in the following lists $L_{pub}$, $L_{part}$, $L_{user}$, $L_{priv}$ and $L_{schnorr-PK}$.

- **Phase 1:** $\mathcal{C}$ interacts with $\mathcal{A}_I$ in the following ways:
  - *Oracle Queries*:
    $\mathcal{C}$ responds to the hash function queries of $\mathcal{A}_I$ in the same way as it responded in the first level ciphertext security.

  * *Public Key Extract Query* ($\mathcal{O}_{pe}$): $\mathcal{C}$ generates the keys using Coron's coin-tossing technique [8] by tossing a biased coin $c_i$ which takes the value $c_i \in \{0,1\}$. The probability that the coin takes the value 0 is $Pr[c_i = 0] = \theta$, which is to be determined later.
    For $c_i = 0$ and $c_i = 1$, $\mathcal{C}$ sets the values of the keys in the same manner as shown in the first level ciphertext security game and initialises the above lists with the key values.

* *Partial Key Extract Query* ($\mathcal{O}_{ppe}$): When $\mathcal{A}_I$ queries for a partial key for an $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ responds exactly as shown in the first level ciphertext security proof.

* *User Key Extract Query* ($\mathcal{O}_{ue}$): When $\mathcal{A}_I$ queries for a user key for an $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ responds exactly as in the first level ciphertext security game.

* *Decryption Query* ($\mathcal{O}_{dec}$): Given a decryption query for a user with $ID_i$ where $c_i = 0$, $\mathcal{C}$ runs the $Decrypt(ID_i, SK_i, C, params)$ algorithm to retrieve and return the plaintext $m$. For an $ID_i$ such that $c_i = 1$, $\mathcal{C}$ responds exactly as shown in the first level ciphertext security game.

The *Re-Key Generation* query $\mathcal{O}_{rk}$, *Re-Encryption* query $\mathcal{O}_{re}$, *Re-Decryption* query $\mathcal{O}_{redec}$ and the *Public Key Replacement* query $\mathcal{O}_{rep}$ are handled in the exact manner as demonstrated for the first level ciphertext security game.

- **Challenge:** Once $\mathcal{A}_I$ decides that Phase-1 is over, it outputs two messages $m_0, m_1$, the delegator identity $ID_{del}$ (whose public keys have not been replaced) the target identity $ID_{ch}$ (whose public keys have not been replaced) on which it wishes to be challenged. (We note that if the public keys are replaced, the simulation would have aborted in the public key replacement query $\mathcal{O}_{rep}$ and compromised Schnorr Signature Scheme.) $\mathcal{C}$ checks if $c_{ch} = 1$ and $c_{del} = 1$ else it aborts. It tosses a coin and chooses $\delta \in \{0, 1\}$ uniformly at random. It retrieves the value of $sk_{ch} = z_{ch}$ of $ID_{ch}$ from $L_{user}$ and generates the re-encryption key $RK_{del \to ch}$ by picking $\alpha^{(1)}_{del-ch}, \alpha^{(2)}_{del-ch}, \beta^{(1)}_{del-ch} \in_R \mathbb{Z}^*_q$ and computing $\beta^{(2)}_{del-ch}$ such that $\beta^{(1)}_{del-ch} \cdot \beta^{(2)}_{del-ch} = z_{ch}$. $\mathcal{C}$ computes the values $v_{del \to ch} = H_2(\alpha^{(2)}_{del-ch} || \beta^{(2)}_{del-ch})$, $V_{del \to ch} = v_{del \to ch} \cdot Y_{ch}$, $W_{del \to ch} = H_3(v_{del \to ch} P) \oplus (\alpha^{(2)}_{del-ch} || \beta^{(2)}_{del-ch})$. It simulates the challenge ciphertext for $m_\delta$ as shown stepwise:
  1. Pick $\sigma^* \in_R \{0,1\}^{l_1}$, $u \in_R \mathbb{Z}^*_q$ and implicitly define $H_4(m_\delta, \sigma) = b$.
  2. Compute $D^*_1 = \alpha^{(1)}_{del-ch} bP = \alpha^{(1)}_{del-ch} \cdot C_1$.
  3. Compute $D^*_2 = \beta^{(1)}_{del-ch} bP = \beta^{(1)}_{del-ch} \cdot C_1$.
  4. Pick $D^*_3 \in_R \{0,1\}^{l_0, l_1}$ and define $H_5(\bar{y_{del}} \cdot abP, z_{del} bP) = D^*_3 \oplus (m_\delta || \sigma)$.
  5. Set $D^*_4 = V_{del \to ch}$.
  6. Set $D^*_5 = W_{del \to ch}$

  Note that the challenge ciphertext $D^* = (D^*_1, D^*_2, D^*_3, D^*_4, D^*_5)$ is identically distributed to the real ciphertext generated by the encryption algorithm. Hence, the challenge ciphertext is a valid ciphertext.

- **Phase-2:** The adversary $\mathcal{A}_I$ continues to query the oracles maintained by $\mathcal{C}$ with the restrictions stated in the security model.

- **Guess:** The adversary $\mathcal{A}_I$ eventually produces its guess $\delta' \in \{0, 1\}$. $\mathcal{C}$ randomly picks a tuple $\langle S, U, u \rangle$ from $H_5$ list and outputs $\bar{y}_{del} S$ as the solution to the CDH instance.

- **Probability Analysis:** We first analyse the probability with which the Challenger $\mathcal{C}$ aborts during the simulation, which can occur during the *partial key extraction query*, *re − key generation query* or the *Challenge* phase. In the context of these three occurrences, $\mathcal{C}$ does not abort in case of the following three events:
  – $E_1$ : In the *partial key extract* query, $c_i = 0$.
  – $E_2$ : In the *re − key generation* query, the public keys of the users $ID_i$ and $ID_j$ have not been replaced.
  – $E_3$ : In the *Challenge* phase, $c_{ch} = 1$, $c_{del} = 1$ and the public keys of the target user $ID_{ch}$ and delegator $ID_{del}$ have not been replaced.

  The probability that $\mathcal{C}$ does not abort $Pr[\neg abort] = \theta^{q_{ppe}}(1-\theta)^2(1-\omega)^{2+q_{rk}}$, which has a maximum value at $\theta_{OPT} = \frac{q_{ppe}}{q_{ppe}+2}$. Using $\theta_{OPT}$, we obtain:

$$Pr[\neg abort] = \frac{2(1-\omega)^{2+q_{rk}}}{e(q_{ppe}+2)^2}.$$

  We analyse the simulation of the random oracles. The simulation of the random oracles occur perfectly unless the following event take place:
  – $E_{H^*_5}$: Event that $(\bar{y_{ch}} abP, z_{ch} bP)$ was queried to $H_5$.

  The analysis of the simulation of the re-decryption oracle remains the same as shown for the first level ciphertext security. The probability of the decryption oracle rejecting valid ciphertexts throughout the entire simulation is denoted by $E_{dec}$ and from previous calculations, we obtain:

$$Pr[E_{dec}] \leq q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)$$

Let $E_{er}$ denote the event $(E_{H_5^*} \vee E_{dec})|\neg abort$. If $E_{er}$ does not occur, the adversary $\mathcal{A}_I$ does not gain any advantage greater than $\frac{1}{2}$ in guessing $\delta$ due to the randomness in the output of $H_5$ oracle. Therefore, $Pr[\delta' = \delta|\neg E_{er}] = \frac{1}{2}$. Note that:

$$\begin{aligned} Pr[\delta' = \delta] &= Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] + Pr[\delta' = \delta|E_{er}]Pr[E_{er}] \\ &\leq 1/2Pr[\neg E_{er}] + Pr[E_{er}] = 1/2 + 1/2Pr[E_{er}]. \end{aligned}$$

Also,

$$Pr[\delta' = \delta] \geq Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] \geq 1/2 - 1/2Pr[E_{er}].$$

By the definition of the advantage of IND-CLPRE-CCA adversary, we have the advantage:

$$\begin{aligned} \epsilon &= |2Pr[\delta' = \delta] - 1| \\ &\leq Pr[E_{er}] = Pr[E_{H_5^*} \vee E_{dec})|\neg abort] \\ &\leq \frac{Pr[E_{H_5^*}] + Pr[E_{dec}]}{Pr[\neg abort].} \end{aligned}$$

Therefore, we obtain the following bound on $Pr[E_{H_5^*}]$ as:

$$\begin{aligned} Pr[E_{H_5^*}] &\geq Pr[\neg abort] \cdot \epsilon - Pr[E_{dec}] \\ &\geq \left( \frac{2(1-\omega)^{2+q_{rk}}\epsilon}{e(q_{ppe}+2)^2} - q_{dec}\Big( \frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q} \Big) \right) \end{aligned}$$

Note that, if event $E_{H_5^*}$ occurs, then the challenger $\mathcal{C}$ solves the $CDH$ instance with advantage:

$$\begin{aligned} \epsilon' &\geq \frac{1}{q_{H_5}} Pr[E_{H_5^*}] \\ &\geq \frac{1}{q_{H_5}} \left( \frac{2(1-\omega)^{2+q_{rk}}\epsilon}{e(q_{ppe}+2)^2} - q_{dec}\Big( \frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q} \Big) \right) \end{aligned}$$

We bound the time taken by $\mathcal{C}$ using $t'$:

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$ and $t_{exp}$ denotes the time taken for exponentiation operation in group $\mathbb{G}$.

□

**First-level Ciphertext Security against Type II adversary** :

**Theorem 3.** *Our proposed scheme is CCA-secure against Type-II adversary for the first level ciphertext under the CDH assumption and the $EUF-CMA$ security of the Schnorr signature scheme. If a $(t, \epsilon)IND-CLPRE-CCA$ Type-II adversary $\mathcal{A}_{II}$ with an advantage $\epsilon$ breaks the IND-CLPRE-CCA security of the given scheme, $\mathcal{C}$ can solve the CDH problem with advantage $\epsilon'$ within time $t'$ where:*

$$\epsilon' \geq \frac{1}{q_{H_2}} \left( \frac{(1-\omega)^{1+q_{rk}}\epsilon}{e(q_{ppe}+q_{ue}+1)} - \frac{q_{H_4}}{2^{l_0+l_1}} - \frac{q_{H_6}}{2^{l_0+l_1}} - \frac{q_{\tilde{H}}}{2^{l_0+l_1}} - q_{dec}\Big( \frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q} \Big) \right)$$

*where $\omega$ is the advantage of an attacker against the EUF-CMA security game of the Schnorr signature scheme and $e$ is the base of the natural logarithm. Time taken by $\mathcal{C}$ to solve the CDH problem is:*

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

*where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$. We denote the time taken for exponentiation operation in group $\mathbb{G}$ as $t_{exp}$.*

*Proof.* If a Type-II adversary $\mathcal{A}_{II}$ for the first level ciphertext with access to the random oracles $\tilde{H}, H_1, H_2, H_3, H_4, H_5$, $H_6$ breaks the IND-CLPRE-CCA security of the given scheme, we can construct an algorithm $\mathcal{C}$ which breaks the CDH problem given the instance $(P, aP, bP)$.

- **Initialisation:** $\mathcal{C}$ chooses $\eta \in \mathbb{Z}_q^*$ and sets $P_{pub} = \eta P$. $\mathcal{C}$ maintains a list of all the keys in the lists $L_{pub}, L_{part}, L_{user}, L_{priv}$ and $L_{schnorr-PK}$ exactly the same way as shown for the first level ciphertext security game against $\mathcal{A}_I$.

- **Phase 1:** $\mathcal{C}$ interacts with $\mathcal{A}_{II}$ in the following ways:
  - *OracleQueries*:
    $\mathcal{C}$ responds to the oracle queries of $\mathcal{A}_{II}$ in the same manner as shown in the first level ciphertext security game for Type-I adversary.

  * *Public Key Extract Query* ($\mathcal{O}_{pe}$): $\mathcal{C}$ generates the keys using Coron's coin-tossing technique [8] by tossing a biased coin $c_i$ which takes the value $c_i \in \{0,1\}$. The probability that the coin takes the value 0 is $Pr[c_i = 0] = \theta$, which is to be determined later. The Coron's technique is used to inject the hard problem instance into the *partial keys* and *user keys* of the honest users.
    If $c_i = 0$, $\mathcal{C}$ chooses $x_i, y_i, d_i \in_R \mathbb{Z}_q^*$. It computes $X_i = x_i \cdot P_{pub} + d_i P$, $Y_i = y_i P$, and sets $H_1(ID_i, X_i, Y_i) = -x_i$. It sets the user secret key by picking $z_i \in \mathbb{Z}_q^*$ and setting $Z_i = z_i P$. Note that the public key generated by $\mathcal{C}$ satisfies the public verifiability in equation 4 as:

$$\begin{aligned} RHS &= X_i + H_1(ID_i, X_i, Y_i) \cdot P_{Pub} \\ &= x_i \cdot \eta P + d_i P + (-x_i) \cdot \eta P \\ &= d_i P \\ &= LHS. \end{aligned}$$

    If $c_i = 1$, $\mathcal{C}$ picks $x_i, \bar{y}_i, \bar{z}_i, d_i \in_R \mathbb{Z}_q^*$ and computes $X_i = x_i P_{pub} + d_i P = x_i \eta P + d_i P$, $Z_i = z_i P = \bar{z}_i a P$, $Y_i = y_i P = \bar{y}_i z_i P = \bar{y}_i \bar{z}_i a P$. $\mathcal{C}$ sets $H_1(ID_i, X_i, Y_i, T_i) = -x_i$. Note that, the public keys set by $\mathcal{C}$ satisfies the public verifiability from equation 4.
    Update the above given lists $L_{pub}, L_{part}, L_{user}, L_{priv}$ with the new key values by adding tuple $\langle ID_i, PK_i, \hat{PK}_i = PK_i \rangle$ to list $P_{current}$, $\langle ID_i, PK_i = (X_i, Y_i, Z_i, d_i), c_i \rangle$ to public key list $L_{pub}$, tuple $\langle ID_i, PPK_i, PSK_i, c_i \rangle$ to partial key list $L_{part}$, tuple $\langle ID_i, UPK_i, USK_i, c_i \rangle$ to user key list $L_{user}$ and tuple $\langle ID_i, SK_i, c_i \rangle$ to private key list $L_{priv}$.

  * *Partial Key Extract Query* ($\mathcal{O}_{ppe}$): When $\mathcal{A}_{II}$ queries for a partial key for an $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ responds exactly the same way as shown in the first level ciphertext security game.

  * *User Key Extract Query* ($\mathcal{O}_{ue}$):When $\mathcal{A}_I$ queries for the user key for an $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ searches the list $L_{user}$ for the existence of a tuple of the form $\langle ID_i, UPK_i, USK_i, c_i \rangle$. If found, $\mathcal{C}$ checks the value of the coin $c_i$. If $c_i = 1$, $\mathcal{C}$ aborts and reports failure. If $c_i = 0$, return the user keys $(USK_i, UPK_i)$ to $\mathcal{A}_{II}$. If such a tuple does not exist, $\mathcal{C}$ generates the user keys using the Coron's coin tossing technique as described in the *Public Key Extract Query*.
    If $c_i = 0$, $\mathcal{C}$ calls the *Public Key Extract Query* oracle to generate the keys which updates the lists $L_{pub}, L_{part}, L_{user}$ and $L_{priv}$. $\mathcal{C}$ returns $(USK_i, UPK_i)$.
    If $c_i = 1$, $\mathcal{C}$ aborts and reports failure.

  * *Re − Key Generation Query* ($\mathcal{O}_{rk}$): When $\mathcal{A}_{II}$ sends a re-key generation query to $\mathcal{C}$ from user $ID_i$ to $ID_j$, $\mathcal{C}$ computes the re-encryption key according to the following cases:
    · If $c_i = 0$, compute $RK_{i \to j}$ by running the *Re-KeyGen* algorithm. Add to list $L_{rekey}$ the tuple $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V_{ij}, W_{ij} \rangle$.
    · If $(c_i = 1 \wedge c_j = 1)$, choose $\alpha_{ij}^{(1)}, \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)} \in_R \mathbb{Z}_q^*$. Compute $v_{ij} = H_2(\alpha_{ij}^{(2)} || \beta_{ij}^{(2)})$, $V_{ij} = v_{ij} \cdot Y_j$, $W_{ij} = H_3(v_{ij} P) \oplus (\alpha_{ij}^{(2)} || \beta_{ij}^{(2)})$. Return $RK_{i \to j}$ to $\mathcal{A}_{II}$ and add the tuple $\langle ID_i, ID_j, \alpha_{ij}^{(1)}, \alpha_{i2} \alpha_{ij}^{(2)}, \beta_{ij}^{(1)}, \beta_{ij}^{(2)}, V_{ij}, W_{ij} \rangle$ in list $L_{rekey}$.

  * *Re − Encryption Query* ($\mathcal{O}_{re}$): When $\mathcal{A}_{II}$ queries decryption of a first-level ciphertext $C$ from $ID_i$ to $ID_j$, the Challenger $\mathcal{C}$ first computes $\overline{C}_1$ and $\overline{C}_2$ and checks if equations 5 and 6 hold to verify the well-formedness of the ciphertext $C$. If validation fails, it outputs $\perp$. Else, $\mathcal{C}$ performs re-encryption according to the following cases:

· If $c_i = 0$ or $(c_i = 1 \ \wedge \ c_j = 1)$, $\mathcal{C}$ computes the second level ciphertext $D$ by obtaining the re-encryption keys $RK_{i\to j}$ by running the $Re-Key\ Generation$ oracle $\mathcal{O}_{rk}$ and calling the $Re\text{-}Encrypt(ID_i, ID_j, C, RK_{i\to j}, params)$ algorithm.

· If $(c_i = 1 \ \wedge \ c_j = 0)$, $\mathcal{C}$ picks $\alpha_{ij}^{(1)},\ \beta_{ij}^{(1)} \in_R \mathbb{Z}_q^*$ and sets $\alpha_{ij}^{(2)} = \bar{y}_i \cdot (\alpha_{ij}^{(1)})^{-1}$ and $\beta_{ij}^{(2)} = \bar{z}_i \cdot (\beta_{ij}^{(1)})^{-1}$. $\mathcal{C}$ computes $D_1 = \alpha_{ij}^{(1)} \bar{z}_i h_i^{-1} C_2 = \alpha_{ij}^{(1)} \bar{z}_i raP$, $D_2 = \beta_{ij}^{(1)} h_i^{-1} C_2 = \beta_{ij}^{(1)} raP$, $D_3 = C_3, D_4 = V_{ij}, D_5 = W_{ij}$ and returns the second-level ciphertext $D = (D_1, D_2, D_3, D_4, D_5)$ to $\mathcal{A}_I$.

∗ *Decryption Query* ($\mathcal{O}_{dec}$): For decrypting a first level ciphertext $C$ under identity $ID_i$ (its public key has not been replaced), $\mathcal{C}$ first checks for the validity of the ciphertext. If valid and $c_i = 0$, it runs the *Decrypt* algorithm to retrieve the plaintext $m$. Else, if $c_i = 1$, $\mathcal{C}$ computes $rY_i = \bar{y}_i \bar{z}_i h_i^{-1} C_2 = \bar{y}_i \bar{z}_i h_i^{-1} rh_i aP$ and $rZ_i = \bar{z}_i h_i^{-1} C_2 = \bar{z}_i h_i^{-1} rh_i aP$. It compute the value of $(m||\sigma) = C_3 \oplus H_5(rY_i, rZ_i)$. It computes $r = H_4(m, \sigma)$, computes the values of $\overline{C}_1$ and $\overline{C}_2$ as shown in Re-Encryption algorithm and performs the following checks:
· $C_1 \stackrel{?}{=} rP$.
· $C_2 = r\tilde{H}(ID_i, C_1, \overline{C}_1, C_3)$.
· $C_3 \stackrel{?}{=} H_5(rY_i, rZ_i) \oplus (m||\sigma)$.
If all the checks are satisfied, $\mathcal{C}$ returns $m$.

∗ *Re − Decryption Query* ($\mathcal{O}_{redec}$): $\mathcal{C}$ responds to the queries in the exact manner as shown in the first level ciphertext security game against Type I adversary.

∗ *Public Key Replacement Query* ($\mathcal{O}_{rep}$): $\mathcal{C}$ $\mathcal{C}$ responds to the queries in the exact manner as shown in the first level ciphertext security game against Type I adversary.

- **Challenge:** Once $\mathcal{A}_{II}$ decides that Phase-1 is over, it outputs two messages $m_0, m_1$ and the target identity $ID_{ch}$ on which it wishes to be challenged. $\mathcal{C}$ checks if $c_{ch} = 1$, else it aborts. It tosses a coin and chooses $\delta \in \{0, 1\}$ uniformly at random. It then simulates the challenge ciphertext for $m_\delta$ in the following steps:
  1. Pick $\sigma^* \in_R \{0,1\}^{l_1},\ \bar{u}, f \in_R \mathbb{Z}_q^*$ and implicitly define $H_4(m_\delta, \sigma) = b$.
  2. Compute $C_1^* = bP = rP$.
  3. Set $u \triangleq \bar{u} - bf$. Compute $\overline{C}_1^* = \bar{u}P - fbP = uP$.
  4. C intends to set $\tilde{H}(ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*) = \gamma_{ch} = h_{ch}P$, where $h_{ch} \in_R \mathbb{Z}_q^*$ and include the tuple $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma_{ch}\rangle$ to $L_{\tilde{H}}$ list. $\mathcal{C}$ checks if a tuple of the form $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h'_{ch}, \gamma_{ch}\rangle$ or $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma'_{ch}\rangle$ already exists in the $L_{\tilde{H}}$ list. If it exists, including $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma_{ch}\rangle$ to the $L_{\tilde{H}}$ list is incorrect and hence GOTO step 1 and re-compute with fresh random values. If no such tuple exists in list $L_{\tilde{H}}$, $\mathcal{C}$ sets $\tilde{H}(ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*) = \gamma_{ch} = h_{ch}P$ and include $\langle ID_{ch}, C_1^*, \overline{C}_1^*, C_3^*, h_{ch}, \gamma_{ch}\rangle$ in $L_{\tilde{H}}$ list.
  5. Compute $C_2^* = h_{ch}bP = r \cdot \tilde{H}(ID_{ch}, C_1, \overline{C}_1, C_3)$.
  6. Pick $C_3^* \in_R \{0,1\}^{l_0, l_1}$ and define $H_5(\bar{z}_{ch}\bar{y}_{ch} \cdot abP, \bar{y}_{ch} \cdot abP) = C_3^* \oplus (m_\delta||\sigma^*)$
  7. Check in $L_{H_6}$ if the tuple $\langle C_1^*, C_2^*, C_3^*, t\rangle$ already exists. If so, GOTO step 1 and recompute with fresh random values.
  8. Define $H_6(C_1^*, C_2^*, C_3^*) = f$ . Set $C_4^* = \bar{u}$. From the definition of $u$, we have $C_4^* = u + rH_6(C_1^*, C_2^*, C_3^*)$.

  Note that the challenge ciphertext is identically distributed to the real ciphertext generated by the encryption algorithm. Hence, the challenge ciphertext is a valid ciphertext.

- **Phase-2:** The adversary $\mathcal{A}_{II}$ continues to query the oracles maintained by $\mathcal{C}$ with the restrictions stated in the security model.

- **Guess:** The adversary $\mathcal{A}_{II}$ eventually produces its guess $\delta' \in \{0, 1\}$ to $\mathcal{C}$. $\mathcal{C}$ randomly picks a tuple $\langle S, U, u\rangle$ from $H_5$ list and outputs $\bar{z}_{ch}\bar{y}_{ch}S$ as the solution to the CDH instance.

- **Probability Analysis:** We first analyse the probability with which the Challenger $\mathcal{C}$ aborts during the simulation, which can occur during the *partial key extraction query*, the *user key extract* query, the $re - key\ generation\ query$ or the *challenge* phase. In the context of these four occurrences, $\mathcal{C}$ does not abort in case of the following four events:
  – $E_1$ : In the *partial key extract* query, $c_i = 0$.
  – $E_2$ : In the *user key extract* query, $c_i = 0$.
  – $E_3$ : In the $re - key\ generation$ query, the public keys of the users $ID_i$ and $ID_j$ have not been replaced.

– $E_4$ : In the *Challenge* phase, $c_i = 1$ and the public keys of the target user $ID_{ch}$ has not been replaced.

The probability that $\mathcal{C}$ does not abort $Pr[\neg abort] = \theta^{q_{ppe}}\theta^{q_{ue}}(1-\theta)(1-\omega)^{1+q_{rk}}$, which has a maximum value at $\theta_{OPT} = \frac{q_{ppe}+q_{ue}}{q_{ppe}+q_{ue}+1}$. Using $\theta_{OPT}$, we obtain:

$$Pr[\neg abort] \geq \frac{(1-\omega)^{1+q_{rk}}}{e(q_{ppe}+q_{ue}+1)}.$$

The analysis of the simulation of the random oracles is exactly the same as shown for the first level ciphertext security. The simulation of the random oracles occur perfectly unless the events $E_{H_4^*}$, $E_{H_5^*}$, $E_{H_6^*}$ or $E_{\tilde{H}^*}$ take place as described for the first level ciphertext security analysis. exactly as shown for the first level ciphertext security against $\mathcal{A}_I$: From previous calculations, we have $Pr[E_{H_6^*}] \leq \frac{q_{H_6}}{\{0,1\}^{l_0+l_1}}$ and $Pr[E_{\tilde{H}^*}] \leq \frac{q_{\tilde{H}}}{\{0,1\}^{l_0+l_1}}$.

The analysis of the simulation of the re-decryption oracle is the same as shown for the first level ciphertext security against $\mathcal{A}_I$. The probability that valid ciphertexts are rejected atleast once in the entire simulation is derived previously as shown:

$$Pr[E_{dec}] \leq q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)$$

Let $E_{er}$ denote the event $(E_{H_4^*} \vee E_{H_5^*} \vee E_{H_6^*} \vee E_{\tilde{H}^*} \vee E_{dec})|\neg abort$. If $E_{er}$ does not occur, the adversary $\mathcal{A}_{II}$ does not gain any advantage greater than $\frac{1}{2}$ in guessing $\delta$ due to the randomness in the output of $H_5$ oracle. Therefore, $Pr[\delta' = \delta|\neg E_{er}] = \frac{1}{2}$. Note that:

$$\begin{aligned} Pr[\delta' = \delta] &= Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] + Pr[\delta' = \delta|E_{er}]Pr[E_{er}] \\ &\leq 1/2 Pr[\neg E_{er}] + Pr[E_{er}] = 1/2 + 1/2 Pr[E_{er}]. \end{aligned}$$

Also,

$$Pr[\delta' = \delta] \geq Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] \geq 1/2 - 1/2 Pr[E_{er}].$$

By the definition of the advantage of IND-CLPRE-CCA adversary, we have the advantage:

$$\begin{aligned} \epsilon &= |2Pr[\delta' = \delta] - 1| \\ &\leq Pr[E_{er}] = Pr[E_{H_4^*} \vee E_{H_5^*} \vee E_{H_6^*} \vee E_{\tilde{H}^*} \vee E_{dec})|\neg abort] \\ &\leq \frac{Pr[E_{H_4^*}] + Pr[E_{H_5^*}] + Pr[E_{H_6^*}] + Pr[E_{\tilde{H}^*}] + Pr[E_{dec}]}{Pr[\neg abort].} \end{aligned}$$

Therefore, we obtain the following bound on $Pr[E_{H_5^*}]$ as:

$$\begin{aligned} Pr[E_{H_5^*}] &\geq Pr[\neg abort] \cdot \epsilon - Pr[E_{H_4^*}] + Pr[E_{H_6^*}] + Pr[E_{\tilde{H}^*}] + Pr[E_{dec}] \\ &\geq \frac{(1-\omega)^{1+q_{rk}}\epsilon}{e(q_{ppe}+q_{ue}+1)} - \frac{q_{H_4}}{2^{l_0+l_1}} - \frac{q_{H_6}}{2^{l_0+l_1}} - \frac{q_{\tilde{H}}}{2^{l_0+l_1}} - q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big) \end{aligned}$$

Note that, if event $E_{H_5^*}$ occurs, then the challenger $\mathcal{C}$ solves the $CDH$ instance with advantage:

$$\begin{aligned} \epsilon' &\geq \frac{1}{q_{H_5}} Pr[E_{H_5^*}] \\ &\geq \frac{1}{q_{H_5}}\Bigg(\frac{(1-\omega)^{1+q_{rk}}\epsilon}{e(q_{ppe}+q_{ue}+1)} - \frac{q_{H_4}}{2^{l_0+l_1}} - \frac{q_{H_6}}{2^{l_0+l_1}} - \frac{q_{\tilde{H}}}{2^{l_0+l_1}} - q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)\Bigg) \end{aligned}$$

We bound the time taken by $\mathcal{C}$ using $t'$:

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$ and $t_{exp}$ denotes the time taken for exponentiation operation in group $\mathbb{G}$. □

**Second-level Ciphertext Security against Type II adversary** :

**Theorem 4.** *Our proposed scheme is CCA-secure against Type-II adversary for the second level ciphertext under the DDH assumption and the $EUF-CMA$ security of the Schnorr signature scheme. If a $(t,\epsilon)IND-CLPRE-CCA$ Type-II adversary $\mathcal{A}_{II}$ with an advantage $\epsilon$ breaks the IND-CLPRE-CCA security of the given scheme, $\mathcal{C}$ can solve the CDH problem with advantage $\epsilon'$ within time $t'$ where:*

$$Pr[E_{H_5^*}] \geq \frac{2(1-\omega)^{2+q_{rk}}}{e(q_{ppe}+q_{ue}+2)^2} - q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)$$

*where $\omega$ is the advantage of an attacker against the EUF-CMA security game of the Schnorr signature scheme and $e$ is the base of the natural logarithm. Time taken by $\mathcal{C}$ to solve the CDH problem is:*

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

*where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$. We denote the time taken for exponentiation operation in group $\mathbb{G}$ as $t_{exp}$.*

*Proof.* If a Type-II adversary $\mathcal{A}_{II}$ for the second level ciphertext with access to the random oracles $\tilde{H}, H_1, H_2, H_3, H_4, H_5, H_6$ breaks the IND-CLPRE-CCA security of the given scheme, we can construct an algorithm $\mathcal{C}$ which breaks the CDH problem given the instance $(P, aP, bP)$.

- **Initialisation:** $\mathcal{C}$ picks $\eta \in_R \mathbb{Z}_q^*$ and compute $P_{pub} = \eta P$. Similar to **Theorem 1**, $\mathcal{C}$ maintains a list of all the keys in the following lists $L_{pub}$, $L_{part}$, $L_{user}$, $L_{priv}$, and $L_{schnorr-PK}$.

- **Phase 1:** $\mathcal{C}$ interacts with $\mathcal{A}_{II}$ in the following ways:
  - *OracleQueries*:
    $\mathcal{C}$ responds to the hash function queries of $\mathcal{A}_{II}$ in the same way as it responded in the first level ciphertext security.

  * *Public Key Extract Query* ($\mathcal{O}_{pe}$): $\mathcal{C}$ generates the keys using Coron's coin-tossing technique [8] by tossing a biased coin $c_i$ which takes the value $c_i \in \{0,1\}$. For $c_i = 0$ and $c_i = 1$, $\mathcal{C}$ sets the values of the keys in the same manner as shown in the first level ciphertext security proof against $\mathcal{A}_{II}$ and initialises the above lists with the key values respectively.

  * *Partial Key Extract Query* ($\mathcal{O}_{ppe}$): When $\mathcal{A}_{II}$ queries for a partial key for an $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ responds exactly as shown in the first level ciphertext security proof.

  * *User Key Extract Query* ($\mathcal{O}_{ue}$): When $\mathcal{A}_{II}$ queries for a user key for an $ID_i$ whose public keys have not been replaced, $\mathcal{C}$ responds exactly as in the first level ciphertext security proof.

  * *Decryption Query* ($\mathcal{O}_{dec}$): Given a decryption query for a user with $ID_i$ where $c_i = 0$, $\mathcal{C}$ runs the *Decrypt*($ID_i$, $SK_i, C, params$) algorithm to retrieve the plaintext $m$. For an $ID_i$ such that $c_i = 1$, given the re-encryption keys for all users,$\mathcal{C}$ can re-encrypt the first level ciphertext $C$ to a second level ciphertext $D$ under a user $ID_j$ such that $c_j = 0$ and then re-decrypt it.

  The re-key generation query $\mathcal{O}_{rk}$, re-encryption query $\mathcal{O}_{re}$, re-decryption query $\mathcal{O}_{redec}$ and the public key replacement queries are handled in the exact manner as demonstrated for the first level ciphertext security proof.

- **Challenge:** Once $\mathcal{A}_{II}$ decides that Phase-1 is over, it outputs two messages $m_0, m_1$, the delegator identity $ID_{del}$ (whose public keys have not been replaced) the target identity $ID_{ch}$(whose public keys have not been replaced) on which it wishes to be challenged. $\mathcal{C}$ checks if $c_{ch} = 1$ and $c_{del} = 1$ else it aborts. It tosses a coin and chooses $\delta \in \{0,1\}$ uniformly at random. It generates the re-encryption key $RK_{del \to ch}$ by picking $\alpha_{del-ch}^{(1)}$, $\alpha_{del-ch}^{(1)}$, $\beta_{del-ch}^{(1)}$, $\beta_{del-ch}^{(1)} \in_R \mathbb{Z}_q^*$ and computing the values $v_{del-ch} = H_2(\alpha_{del-ch}^{(2)} || \beta_{del-chij}^{(2)})$, $V_{del-ch} = v_{del-ch} \cdot Y_{ch}$, $W_{del-ch} = H_3(v_{del-ch}P) \oplus (\alpha_{del-ch}^{(2)} || \beta_{del-ch}^{(2)})$. It simulates the challenge ciphertext for $m_\delta$ as shown stepwise:
  1. Pick $\sigma^* \in_R \{0,1\}^{l_1}$, $u \in_R \mathbb{Z}_q^*$ and implicitly define $H_4(m_\delta, \sigma) = b$.
  2. Compute $D_1^* = \alpha_{del-ch}^{(1)} bP = \alpha_{del-ch}^{(1)} \cdot C_1$.
  3. Compute $D_2^* = \beta_{del-ch}^{(1)} bP = \beta_{del-ch}^{(1)} \cdot C_1$.
  4. Pick $D_3^* \in_R \{0,1\}^{l_0, l_1}$ and define $H_5(\overline{y}_{del}\overline{z}_{del} \cdot abP, \overline{z}_{del} \cdot abP) = D^3 \oplus (m_\delta || \sigma^*)$.
  5. Set $D_4^* = V_{del \to ch}$.

6. Set $D_5^* = W_{del \to ch}$

Note that the challenge ciphertext $D^* = (D_1^*, D_2^*, D_3^*, D_4^*, D_5^*)$ is identically distributed to the real ciphertext generated by the encryption algorithm. Hence, the challenge ciphertext is a valid ciphertext.

- **Phase-2:** The adversary $\mathcal{A}_{II}$ continues to query the oracles maintained by $\mathcal{C}$ with the restrictions stated in the security model.

- **Guess:** The adversary $\mathcal{A}_{II}$ eventually produces its guess $\delta' \in \{0,1\}$. $\mathcal{C}$ randomly picks a tuple $\langle S, U, u \rangle$ from $H_5$ list and outputs $\overline{y}_{del}\overline{z}_{del}S$ as the solution to the CDH instance.

- **Probability Analysis:** We first analyse the probability with which the Challenger $\mathcal{C}$ aborts during the simulation, which can occur during the *partial key extraction query*,the *user key extract query*, the $re-key$ *generation query* or the *challenge* phase. In the context of these four occurrences, $\mathcal{C}$ does not abort in case of the following four events:
  - $E_1$ : In the *partial key extract* query, $c_i = 0$.
  - $E_2$ : In the *user key extract* query, $c_i = 0$.
  - $E_3$ : In the $re-key$ *generation* query, the public keys of the users $ID_i$ and $ID_j$ have not been replaced.
  - $E_4$ : In the *Challenge* phase, $c_{ch} = 1$, $c_{del} = 1$ and the public keys of the target user $ID_{ch}$ and delegator $ID_{del}$ have not been replaced.

  The probability that $\mathcal{C}$ does not abort $Pr[\neg abort] = \theta^{q_{ppe}}\theta^{q_{ue}}(1-\theta)^2(1-\omega)^{2+q_{rk}}$, which has a maximum value at $\theta_{OPT} = \frac{q_{ppe}+q_{ue}}{q_{ppe}+q_{ue}+2}$. Using $\theta_{OPT}$, we obtain:

$$Pr[\neg abort] = \frac{2(1-\omega)^{2+q_{rk}}}{e(q_{ppe}+q_{ue}+2)^2}.$$

  We analyse the simulation of the random oracles. The simulation of the random oracles occur perfectly unless the following event take place:
  - $E_{H_5^*}$: Event that $(y_{ch}^{-}abP, z_{ch}bP)$ was queried to $H_5$.

  The analysis of the simulation of the re-decryption oracle remains the same as shown for the first level ciphertext security against $\mathcal{A}_{II}$. The probability of the decryption oracle rejecting valid ciphertexts throughout the entire simulation is denoted by $E_{dec}$ and from previous calculations, we obtain:

$$Pr[E_{dec}] \leq q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big)$$

  Let $E_{er}$ denote the event $(E_{H_5^*} \vee E_{dec})|\neg abort$. If $E_{er}$ does not occur, the adversary $\mathcal{A}_I$ does not gain any advantage greater than $\frac{1}{2}$ in guessing $\delta$ due to the randomness in the output of $H_5$ oracle. Therefore, $Pr[\delta' = \delta|\neg E_{er}] = \frac{1}{2}$. Note that:

$$\begin{aligned} Pr[\delta' = \delta] &= Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] + Pr[\delta' = \delta|E_{er}]Pr[E_{er}] \\ &\leq 1/2 Pr[\neg E_{er}] + Pr[E_{er}] = 1/2 + 1/2 Pr[E_{er}]. \end{aligned}$$

  Also,

$$Pr[\delta' = \delta] \geq Pr[\delta' = \delta|\neg E_{er}]Pr[\neg E_{er}] \geq 1/2 - 1/2 Pr[E_{er}].$$

  By the definition of the advantage of IND-CLPRE-CCA adversary, we have the advantage:

$$\begin{aligned} \epsilon &= |2Pr[\delta' = \delta] - 1| \\ &\leq Pr[E_{er}] = Pr[E_{H_5^*} \vee E_{dec})|\neg abort] \\ &\leq \frac{Pr[E_{H_5^*}] + Pr[E_{dec}]}{Pr[\neg abort].} \end{aligned}$$

  Therefore, we obtain the following bound on $Pr[E_{H_5^*}]$ as:

$$\begin{aligned} Pr[E_{H_5^*}] &\geq Pr[\neg abort] \cdot \epsilon - Pr[E_{dec}] \\ &\geq \frac{2(1-\omega)^{2+q_{rk}}}{e(q_{ppe}+q_{ue}+2)^2} - q_{dec}\Big(\frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q}\Big) \end{aligned}$$

Note that, if event $E_{H_5^*}$ occurs, then the challenger $\mathcal{C}$ solves the $CDH$ instance with advantage:

$$\epsilon' \geq \frac{1}{q_{H_5}} Pr[E_{H_5^*}]$$
$$\geq \frac{1}{q_{H_5}} \Big( \frac{2(1-\omega)^{2+q_{rk}}}{e(q_{ppe}+q_{ue}+2)^2} - q_{dec} \Big( \frac{q_{H_5}/q}{1-(q_{H_4}/(2^{l_0+l_1}))} + \frac{q_{H_4}/(2^{l_0+l_1})}{1-(q_{H_5}/q)} + \frac{2}{q} \Big) \Big)$$

We bound the time taken by $\mathcal{C}$ using $t'$:

$$t' \leq t + (T_q)O(1) + (T_{\mathcal{O}})t_{exp}$$

where $T_q = q_{\tilde{H}} + q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_{H_6}$, $T_{\mathcal{O}} = 4t_{pe} + 4t_{ppe} + 4t_{ue} + 2t_{rk} + 8t_{re} + 8t_{dec} + 6t_{redec}$ and $t_{exp}$ denotes the time taken for exponentiation operation in group $\mathbb{G}$.

□

### 4.4 Efficiency Comparison

We give a comparison of the efficiency of our proposed CL-PRE scheme with [13] with the suggested fix as described in Section 3.3. In Table 1, we show the computational efficiency of our scheme and the modified scheme by comparing the time taken by the different algorithms in our protocols. Note that we use $t_{exp}$ to denote the time required for exponentiation in a group. The comparison reveals that our scheme is more efficient than the existing scheme with our suggested fix.

| **Scheme** | Modified CLPRE scheme of Srinivasan *et al.* [13] | Our CLPRE Scheme |
|---|---|---|
| Setup | $t_{exp}$ | $t_{exp}$ |
| PartialKeyExtract | $3t_{exp}$ | $2t_{exp}$ |
| UserKeyGen | $2t_{exp}$ | $t_{exp}$ |
| SetPublicKey | $2t_{exp}$ | – |
| PublicVerify | $8t_{exp}$ | $2t_{exp}$ |
| Re-KeyGen | $5t_{exp}$ | $2t_{exp}$ |
| Encrypt | $10t_{exp}$ | $4t_{exp}$ |
| Re-Encrypt | $10t_{exp}$ | $6t_{exp}$ |
| Decrypt | $11t_{exp}$ | $6t_{exp}$ |
| Re-Decrypt | $6t_{exp}$ | $4t_{exp}$ |

Table 1: Efficiency comparison of the scheme [13] with the suggested fix with our CL-PRE scheme indicates that our scheme is more efficient.

## 5 Conclusion

Although several CL-PRE schemes were proposed in the literature, to the best of our knowledge, only one scheme [9] has reported the certificateless property without any known attacks to the scheme. However, the scheme is based on costly bilinear pairing operation and satisfies a weaker notion of security, termed as RCCA security. Recently, Srinivasan *et al.* [13] proposed a CL-PRE scheme without resorting to bilinear pairing in the random oracle model. However, we demonstrated that their security proof is flawed by presenting a concrete attack. We then presented a unidirectional CL-PRE scheme which is pairing-free and satisfies CCA-security against both the Type-I and Type-II adversaries for the first and second level ciphertexts. We remark that a potential fix to [13] is also suggested in our paper but our proposed algorithm is more efficient as noted from our efficiency comparison. Our work affirmatively resolves the problems faced by PKI-based and IB-based PRE schemes by proposing an efficient pairing-free certificateless Proxy Re-encryption scheme.

## References


1. Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In *Advances in Cryptology - ASIACRYPT 2003, 9th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, November 30 - December 4, 2003, Proceedings*, pages 452–473, 2003.

2. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *IN NDSS*, 2005.
3. 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.
4. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 127–144. Springer, 1998.
5. Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–222. Springer, 2004.
6. Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient unidirectional proxy re-encryption. In *Progress in Cryptology - AFRICACRYPT 2010, Third International Conference on Cryptology in Africa, Stellenbosch, South Africa, May 3-6, 2010. Proceedings*, pages 316–332, 2010.
7. Sherman SM Chow, Jian Weng, Yanjiang Yang, and Robert H Deng. Efficient unidirectional proxy re-encryption. In *International Conference on Cryptology in Africa*, pages 316–332. Springer, 2010.
8. Jean-Sébastien Coron. On the exact security of full domain hash. In *Annual International Cryptology Conference*, pages 229–235. Springer, 2000.
9. Hui Guo, Zhenfeng Zhang, Jiang Zhang, and Cheng Chen. Towards a secure certificateless proxy re-encryption scheme. In *Provable Security - 7th International Conference, ProvSec 2013, Melaka, Malaysia, October 23-25, 2013. Proceedings*, pages 330–346, 2013.
10. Benoît Libert and Damien Vergnaud. Tracing malicious proxies in proxy re-encryption. In *International Conference on Pairing-Based Cryptography*, pages 332–353. Springer, 2008.
11. Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.
12. Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 47–53, 1984.
13. Akshayaram Srinivasan and C. Pandu Rangan. Certificateless proxy re-encryption without pairing: Revisited. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing, SCC@ASIACCS '15, Singapore, Republic of Singapore, April 14, 2015*, pages 41–52, 2015.
14. Chul Sur, Chae Duk Jung, Youngho Park, and Kyung Hyune Rhee. Chosen-ciphertext secure certificateless proxy re-encryption. In *Communications and Multimedia Security, 11th IFIP TC 6/TC 11 International Conference, CMS 2010, Linz, Austria, May 31 - June 2, 2010. Proceedings*, pages 214–232, 2010.
15. Kang Yang, Jing Xu, and Zhenfeng Zhang. Certificateless proxy re-encryption without pairings. In *Information Security and Cryptology - ICISC 2013 - 16th International Conference, Seoul, Korea, November 27-29, 2013, Revised Selected Papers*, pages 67–88, 2013.
16. Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 261–270. ACM, 2010.
17. Yulin Zheng, Shaohua Tang, Chaowen Guan, and Min-Rong Chen. Cryptanalysis of a certificateless proxy re-encryption scheme. In *2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies, Xi'an, Shaanxi, China, September 9-11, 2013*, pages 307–312, 2013.