A Provably-Secure Unidirectional Proxy Re-Encryption Scheme Without Pairing in the Random Oracle Model

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) enables delegation of decryption rights by entrusting a proxy server with special information, that allows it to transform a ciphertext under one public key into a ciphertext of the same message under a different public key. It is important to note that, the proxy which performs the re-encryption learns nothing about the message encrypted under either public keys. Due to its transformation property, proxy re-encryption schemes have practical applications in distributed storage, encrypted email forwarding, Digital Rights Management (DRM) and cloud storage. From its introduction, several proxy re-encryption schemes have been proposed in the literature, and a majority of them have been realized using bilinear pairing. In Africacrypt 2010, the first PKI-based collusion resistant CCA secure PRE scheme without pairing was proposed in the random oracle model. In this paper, we point out an important weakness in the scheme. We also present the first collusion-resistant pairing-free unidirectional proxy re-encryption scheme which meets CCA security under a variant of the computational Diffie-Hellman hardness assumption in the random oracle model.

Keywords: Proxy Re-Encryption, Random Oracle Model, Chosen Ciphertext Security, provably secure, unidirectional.

1 Introduction

Encryption is one of the fundamental cryptographic primitives for scenarios requiring confidentiality. Proxy re-encryption is an important primitive that allows a third party termed as *proxy server*, to transform the ciphertext of a user into a ciphertext of another user without learning anything about the underlying message. Consider the email forwarding scenario, where Alice with public key PK_{Alice} is on a vacation and wishes her mail server to forward all her encrypted emails to Bob with public key PK_{Bob}. Here, Alice is not interested in sharing her private key SK_{Alice} with either the mail server or Bob as that would compromise her private key. As pointed out by Mambo and Okamoto in [18], this is a common situation in practice where a data encrypted under PK_{Alice} is required to be encrypted under PK_{Bob}. When Alice is online, the encrypted message $E_{PK_{Alice}}(m)$ can be decrypted by Alice using SK_{Alice} to extract the message m, followed by encrypting it under PK_{Bob} to obtain the ciphertext $E_{PK_{Bob}}(m)$. But in applications like encrypted email forwarding (described above), secure distributed file systems and outsourced filtering of encrypted spam, when the owner of SK_{Alice} is not online, proxy re-encryption efficiently solves the problem of delegation of decryption rights with the involvement of an untrusted party called proxy. Here, Alice provides a secret information to the proxy called Re-Encryption Key (but not her private key SK_{Alice}) that allows it to transform $E_{PK_{Alice}}(m)$ to $E_{PK_{Bob}}(m)$. Since Alice delegates her decryption rights to Bob, Alice is termed as *delegator* and Bob as *delegatee*. The important property to note here is, the proxy learns nothing about m or SK_{Alice}.
A PRE scheme can be unidirectional or bidirectional. In a bidirectional scheme, the re-encryption key ($RK_{Alice \to Bob}$) allows re-encryption in both directions, i.e., the transformation of a ciphertext $E_{PK_{Alice}}(m)$ under the public key of Alice to a ciphertext $E_{PK_{Bob}}(m)$ under the public key of Bob and also $E_{PK_{Bob}}(m)$ to $E_{PK_{Alice}}(m)$. In a unidirectional scheme, the re-encryption key $RK_{Alice \to Bob}$ allows the transformation of the ciphertext only in one direction, i.e., $E_{PK_{Alice}}(m)$ to $E_{PK_{Bob}}(m)$ but not vice versa. Again, the re-encryption algorithm can be single-hop or multi-hop. In a single-hop scheme, the re-encrypted ciphertext cannot be re-encrypted any further. In a multi-hop scheme, the re-encrypted ciphertext can be re-encrypted multiple times. We focus on unidirectional single-hop PRE schemes in this paper.
PRE can be used in many applications, including encrypted email forwarding, distributed file systems, secure certified email mailing lists, the DRM of Apple's iTunes, access control and privacy for public transportation [1,2,21,14,22].

[⋆] Postdoctoral researcher supported by Project No. CCE/CEP/22/VK&CP/CSE/14-15 on Information Security & Awareness(ISEA) Phase-II by Ministry of Electronics & Information Technology, Government of India.

[⋆⋆] Work partially supported by Project No. CCE/CEP/22/VK&CP/CSE/14-15 on ISEA-Phase II.

In 1998, Blaze, Bleumer and Strauss [5] proposed the first Elgamal-based bidirectional proxy re-encryption scheme. However, their scheme is *transitive* and not *collusion resistant*. In a transitive PRE scheme, given the re-encryption keys $RK_{Alice \rightarrow Bob}$ and $RK_{Bob \rightarrow Carol}$, the proxy can compute $RK_{Alice \rightarrow Carol}$. Also, collusion resistance is an important property of PRE, which prevents a colluding proxy and the delegatees to extract the delegator's private key. Besides, bidirectionality may not be always desirable. Dodis and Ivan [15] proposed a CCA security model for PRE and designed the first unidirectional PRE scheme. However, in their protocol, the decryption key of the delegatee Bob requires a part of the the private key of the delegator Alice.
In 2005, Ateniese *et al.* [1,2] gave the first construction of a unidirectional PRE scheme based on bilinear maps. Their scheme is non-transitive and collusion resistant. But, their scheme only offers chosen plaintext security which is insufficient for many practical applications.
In 2007, Canetti and Hohenberger [6] proposed the security notion of PRE satisfying chosen-ciphertext attack and presented a bidirectional CCA secure PRE scheme using bilinear pairing satisfying the same in the standard model. In 2008, Libert and Vergnaud [17] presented a single-hop unidirectional PRE scheme in the standard model using pairing. Their scheme is secure against replayable chosen-ciphertext attack ($RCCA$). $RCCA$ security is a weaker variant of the CCA security in the sense that a harmless mauling of the challenge ciphertext is tolerated. Green and Ateniese [13] also proposed a pairing based CCA-secure PRE scheme for ID-based cryptosystems.
Note that, despite recent advances in implementation techniques, bilinear pairing takes more than twice the time taken by modular exponentiation computation [4] and is an expensive operation. Weng *et al.* [10] proposed a CCA secure pairing-free bidirectional PRE scheme, but their scheme is not collusion resilient [23]. Subsequently, Shao and Cao [20] proposed a unidirectional PRE scheme without pairing, which was later shown to be vulnerable to CCA attack by Chow *et al.* [8]. In Africacrypt 2010, Chow *et al.* [8] proposed a CCA secure PRE scheme that does not use bilinear pairing. Their scheme is unidirectional and resists collusion attack. This is the only scheme that offers these properties without pairing. However, in this work, we expose a critical weakness in the security proof of the scheme.

2 Our Contributions

Although several PRE schemes have been proposed in the literature, a majority of the schemes relies on costly bilinear pairing operations. As stated by Chow *et al.*[8], removing pairing operations from PRE constructions is one of the open problems left by [6]. Weng *et al.* [10] proposed the first CCA secure pairing-free PRE scheme, which was however shown to be vulnerable to *collusion attack* [23]. Note that collusion resistance is an important property in the PRE setting, which prevents a colluding proxy and malicious delegatees from recovering the private key of the delegator. Collusion resistance, also termed as *delegator secret security* is a desirable property in many practical scenarios such as secure cloud services. In 2010, Chow *et al.*[8] proposed the first construction of a collusion-resistant CCA secure pairing-free PRE scheme. However, in our work, we point out a major weakness in the security proof of the scheme by Chow *et al.*, which clearly shows that the scheme is not provably secure. Further, we justify that a trivial fix to the scheme is not possible. We also provide the first construction of a CCA-secure collusion-resistant pairing-free unidirectional single-hop proxy re-encryption scheme under the Computational Diffie-Hellman(CDH) and the Divisible Computational Diffie-Hellman(DCDH) hardness assumptions in the random oracle model. To the best of our knowledge, ours is the first construction of pairing-free PRE that affirmatively satisfies the collusion-resistance property.

3 Definition and Security Model

We provide the definitions and security notions of unidirectional proxy re-encryption systems in this section.

3.1 A Generic Model for Single-hop Unidirectional Proxy Re-Encryption Scheme

A single-hop unidirectional PRE scheme consists of the following algorithms described below:

- **Setup**(λ): The setup algorithm is run with the security parameter λ as input, and it returns the public parameters $PARAMS$.
- **KeyGen**$(U_i, PARAMS)$: The key generation algorithm takes the user information U_i and public parameters $PARAMS$ as inputs, and returns the private key SK_i and its corresponding public key PK_i for a user U_i.
- **ReKeyGen**$(SK_i, PK_i, PK_j, PARAMS)$: The re-encryption key (re-key) generation algorithm takes the private key SK_i and public key PK_i of the delegator, the public key of the delegatee PK_j and the public parameters $PARAMS$ as inputs and returns the re-encryption key $RK_{i \rightarrow j}$. This algorithm is run by the user U_i.
- **Encrypt**$(m, PK_i, PARAMS)$: The encryption algorithm takes a message $m \in \mathcal{M}$, public key PK_i of the receiver and the public parameters $PARAMS$ as inputs, and returns the ciphertext σ_i, which is an encryption of m under PK_i. The generated ciphertext σ_i is termed *original ciphertext*, which can be further re-encrypted.

– **ReEncrypt**$(\sigma_i, PK_i, PK_j, RK_{i\to j}, PARAMS)$: The re-encryption algorithm takes a ciphertext σ_i (encryption of a message $m \in \mathcal{M}$ under PK_i), the public key of delegator PK_i, the public key of the delegatee PK_j, the re-encryption key $RK_{i\to j}$ and the public parameters $PARAMS$ as inputs, and returns the re-encrypted ciphertext $\hat{\sigma_j}$ which is an encryption of m under PK_j. This algorithm is run by the proxy who does not learn anything about the message m. The re-encrypted ciphertext $\hat{\sigma_j}$ is termed *transformed ciphertext.*
– **Encrypt**$_1(m, PK_i, PARAMS)$: The encryption algorithm takes a message $m \in \mathcal{M}$, public key PK_i of the receiver and the public parameters $PARAMS$ as inputs, and returns a non-transformable ciphertext $\hat{\sigma}_i$, which is an encryption of m under PK_i. The generated ciphertext $\hat{\sigma}_i$ is termed *non-transformable* since it cannot be further re-encrypted.
– **Decrypt**$(\sigma_i, PK_i, SK_i, PARAMS)$: The decryption algorithm takes the ciphertext σ_i (original,transformed or non-transformable) which is an encryption of a message $m \in \mathcal{M}$ under PK_i, the public key PK_i, the private key SK_i and the public parameters $PARAMS$ as inputs. It returns the message $m \in \mathcal{M}$ if σ_i is a valid encryption of m under PK_i, or the error symbol $\perp$ otherwise.

3.2 Security Model

We adopt the game based definitions of ciphertext security of a single-hop unidirectional PRE scheme from the security model of Chow *et al.* [8]. The security of a unidirectional PRE scheme is modelled in the form of a security game between two entities : the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$. $\mathcal{C}$ simulates an environment running PRE for $\mathcal{A}$, who can adaptively query the oracles (to be listed later) which $\mathcal{C}$ answers. Our security model is based on the *Knowledge of Secret Key* (KOSK) model [6,16]. The challenger computes and provides the public keys of the honest users (HU) and the public/private key pairs of the corrupt users (CU) beforehand and the adversary cannot determine which parties are to be compromised adaptively. $\mathcal{C}$ executes the key generation algorithm n_h times and n_c times to generate the public/private key pairs of the honest and corrupt users respectively. Due to the presence of three types of ciphertexts: original, transformed and non-transformable ciphertexts, it is essential to prove the security for all the three types. Hence, we consider separate security models for the original, transformed and non-transformable ciphertexts in our scheme.

Original ciphertext security The original ciphertext security model involves an adversary $\mathcal{A}$ challenged with an original ciphertext under the target public key PK_T. The security of the scheme is shown by a game between an adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ as demonstrated below:

– **Phase I**: $\mathcal{C}$ takes the security parameter λ as input, runs the algorithm $Setup(\lambda)$ and gives the resulting system parameters $PARAMS$ to $\mathcal{A}$. $\mathcal{C}$ runs the KeyGen algorithm and provides the public keys of the honest users and the public/private key pairs of the corrupt users to $\mathcal{A}$. Additionally, $\mathcal{A}$ can adaptively query the following oracles provided by $\mathcal{C}$.
  - $\mathcal{O}_{ReKeyGen}(PK_i, PK_j)$: $\mathcal{C}$ runs the algorithm **ReKeyGen**$(SK_i, PK_i, PK_j, PARAMS)$ and returns the re-encryption key $RK_{i\to j}$ to $\mathcal{A}$.
  - $\mathcal{O}_{ReEncrypt}(\sigma_i, PK_i, PK_j)$: $\mathcal{C}$ runs the algorithm **ReEncrypt**$(\sigma_i, PK_i, PK_j,$ $ReKeyGen(SK_i, PK_i, PK_j, PARAMS), PARAMS)$ and returns the re-encrypted ciphertext $\hat{\sigma_j}$ to $\mathcal{A}$.
  - $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$ or $\mathcal{O}_{Decrypt}(\hat{\sigma_i}, PK_i)$: $\mathcal{C}$ runs the algorithm **Decrypt**$(\sigma_i, PK_i, SK_i, PARAMS)$ and returns the result to $\mathcal{A}$. Here, $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$ is a query to decrypt the original ciphertexts and $\mathcal{O}_{Decrypt}(\hat{\sigma_i}, PK_i)$ to decrypt the transformed/non-transformable ciphertexts.

– **Challenge Phase**: After taking sufficient training, $\mathcal{A}$ provides a target public key PK_T ($T \in HU$), and two messages $m_0, m_1 \in \mathcal{M}$ of equal length. $\mathcal{C}$ flips a random coin $\delta \in \{0,1\}$, sets the challenge ciphertext to be $\sigma_T = Encrypt(m_\delta, PK_T, PARAMS)$ and provides σ_T as the challenge ciphertext to $\mathcal{A}$.

– **Phase II**: $\mathcal{A}$ adaptively issues queries to the oracles simulated by $\mathcal{C}$ and $\mathcal{C}$ responds as in **Phase I**. However, $\mathcal{A}$ is restricted from placing the following queries to $\mathcal{C}$, which trivially lets $\mathcal{A}$ decrypt the challenge ciphertext σ_T:
  - $\mathcal{O}_{ReKeyGen}(PK_T, PK_j)$ is allowed only if $PK_j \in HU$.
  - $\mathcal{A}$ cannot issue a re-encryption query $\mathcal{O}_{ReEncrypt}(\sigma_i, PK_i, PK_j)$ where $PK_j \in CU$, if (σ_i, PK_i) is a *challenge derivative* (See Definition 1) of (σ_T, PK_T).
  - $\mathcal{A}$ cannot issue a decryption query $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$ if (σ_i, PK_i) is a *challenge derivative* of (σ_T, PK_T).
  - $\mathcal{A}$ cannot issue a corrupted key generation query on user U_T to obtain the target private key SK_T.
  - $\mathcal{A}$ cannot issue decryption queries on σ_T under PK_T.

Definition 1. ***(Challenge Derivative for Chosen-Ciphertext Security)****. A challenge derivative* (σ_i, PK_i) *in the CCA setting is defined below. The definition is adopted from [8]:*

- ***Reflexitivity:*** (σ_i, PK_i) *is a challenge derivative of itself.*
- ***Derivative by re-encryption:*** $(\hat{\sigma_j}, PK_j)$ *is a challenge derivative of* (σ_i, PK_i) *if* $\hat{\sigma_j} \leftarrow \mathcal{O}_{ReEncrypt}$ (σ_i, PK_i, PK_j).
- ***Derivative by re-encryption key:*** $(\hat{\sigma}_j, PK_j)$ *is a challenge derivative of* (σ_i, PK_i) *if* $RK_{i\to j} \leftarrow \mathcal{O}_{ReKeyGen}(PK_i, PK_j)$ *and* $\hat{\sigma_j} = \textit{Re-Encrypt}(\sigma_i, PK_i, PK_j, RK_{i\to j}, PARAMS)$.

– **Guess**: Finally, $\mathcal{A}$ outputs a guess $\delta' \in \{0, 1\}$ and wins the game if $\delta' = \delta$.

Transformed ciphertext security In a single-hop PRE scheme, it is essential to also prove the security for the transformed ciphertext as the ciphertexts cannot be further re-encrypted in a single-hop environment. The adversary $\mathcal{A}$ is challenged with a transformed ciphertext, and does not have access to its corresponding original ciphertext. $\mathcal{A}$ is challenged with a ciphertext $\hat{\sigma}_T$ (re-encryption of $\sigma_{i'}$ under the public key $PK_{i'}$ to $\hat{\sigma}_T$ under PK_T) re-encrypted using the re-key $RK_{i'\to T}$. The security for the transformed ciphertext remains unaffected by the fact whether $U_{i'}$ is a corrupt user or not. The security of the scheme is shown by a game between an adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ and is demonstrated below:

– **Phase I**: $\mathcal{C}$ takes the security parameter λ as input and runs the algorithm $Setup(\lambda)$ and gives the resulting system parameters $PARAMS$ to $\mathcal{A}$. $\mathcal{C}$ runs the KeyGen algorithm and provides the public keys of the honest users and the public/private key pairs of the corrupt users to $\mathcal{A}$. Additionally, $\mathcal{A}$ can adaptively query the re-key generation oracle $\mathcal{O}_{ReKeyGen}(PK_i, PK_j)$, re-encryption oracle $\mathcal{O}_{ReEncrypt}(\sigma_i, PK_i, PK_j)$ and decryption oracles $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$ provided by $\mathcal{C}$.

– **Challenge Phase**: After taking sufficient training, $\mathcal{A}$ provides the delegator's public key $PK_{i'}$, the delegatee's (target) public key PK_T $(T \in HU)$, and two messages $m_0, m_1 \in \mathcal{M}$ of equal length. $\mathcal{C}$ flips a random coin $\delta \in \{0, 1\}$ and sets the challenge ciphertext to be $\hat{\sigma_T} = ReEncrypt(m_\delta, PK_i', PK_T, RK_{i'\to T}, PARAMS)$ and provides $\hat{\sigma}_T$ as challenge ciphertext to $\mathcal{A}$.

– **Phase II**: $\mathcal{A}$ adaptively issues queries and $\mathcal{C}$ responds as in **Phase I**. However, the adversary $\mathcal{A}$ is subjected to the following restrictions during this phase :
  1. If $PK_i' \in CU$, $\mathcal{A}$ cannot query $RK_{i'\to T}$.
  2. If $\mathcal{A}$ has already obtained $RK_{i'\to T}$ where $PK_i' \in CU$, PK_i' cannot be the delegator in the challenge phase.
  3. $\mathcal{A}$ cannot issue a corrupted key generation query on user U_T to obtain the target private key SK_T.
  4. $\mathcal{A}$ cannot issue decryption queries on $\hat{\sigma_T}$ under PK_T.

– **Guess**: Finally, $\mathcal{A}$ outputs a guess $\delta' \in \{0, 1\}$ and wins the game if $\delta' = \delta$.

Nontransformable Ciphertext Security Non-transformable ciphertexts are the encryptions of very sensitive information that should not be further encrypted. The $Encrypt_1$ algorithm produces non-transformable ciphertexts $\hat{\sigma}_i$, indistinguishable from re-encrypted ciphertexts generated by the $ReEncypt$ algorithm. The security of the scheme is shown by a game between an adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ and is demonstrated below:

- **Phase I**: $\mathcal{C}$ runs the algorithm $Setup(\lambda)$ and gives the system parameters $PARAMS$ to $\mathcal{A}$. $\mathcal{C}$ runs the KeyGen algorithm and provides the public keys of the honest users and the public/private key pairs of the corrupt users to $\mathcal{A}$. Additionally, $\mathcal{A}$ can adaptively query the re-key generation oracle $\mathcal{O}_{ReKeyGen}(PK_i, PK_j)$ and decryption oracles $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$ provided by $\mathcal{C}$.

- **Challenge Phase**: After taking sufficient training, $\mathcal{A}$ provides the target public key PK_T $(T \in HU)$ and two messages $m_0, m_1 \in \mathcal{M}$ of equal length. $\mathcal{C}$ flips a random coin $\delta \in \{0, 1\}$, and sets the challenge ciphertext to be $\hat{\sigma_T} = Encrypt_1(m_\delta, PK_T, PARAMS)$ and provides $\hat{\sigma}_T$ as challenge ciphertext to $\mathcal{A}$.

- **Phase II**: $\mathcal{A}$ adaptively issues queries and $\mathcal{C}$ responds as in **Phase I**. However, the adversary $\mathcal{A}$ is subjected to the following restrictions during this phase :
  1. $\mathcal{A}$ cannot issue a corrupted key generation query on user U_T to obtain the target private key SK_T.
  2. $\mathcal{A}$ cannot issue decryption queries on $\hat{\sigma_T}$ under PK_T.

- **Guess**: Finally, $\mathcal{A}$ outputs a guess $\delta' \in \{0, 1\}$ and wins the game if $\delta' = \delta$.

We refer to the above adversary $\mathcal{A}$ against all the three types of ciphertexts as *IND-PRE-CCA* adversary. $\mathcal{A}$'s advantage in attacking the PRE scheme is defined as :

$$Adv_{PRE,\mathcal{A}}^{IND-PRE-CCA} = |2Pr[\delta' - \delta] - 1|,$$

where, the probability is over the random coin tosses performed by $\mathcal{C}$ and $\mathcal{A}$. Given a t-time *IND-PRE-CCA* adversary $\mathcal{A}$ who makes at most q_{RK} re-encryption key generation queries, q_{RE} re-encryption queries and q_d decryption queries, a single-hop unidirectional PRE scheme is defined as (t, ϵ)-*IND-PRE-CCA* secure if the advantage of $\mathcal{A}$ is $Adv_{PRE,\mathcal{A}}^{IND-PRE-CCA} \leq \epsilon$.

Delegator Secret Security

Delegator secret security or collusion-resistance prevents a colluding dishonest proxy and delegatees to derive the delegator's private key in full [8]. This attack is also captured by the transformed/non-transformable ciphertext security, since the challenger provides the adversary with all the re-encryption keys, which makes decryption of ciphertexts encrypted under the delegator's public key easy when the adversary can derive the delegator's private key completely. We adopt the delegator secret security model from the definition in Chow *et al.*[7]. The security is defined by the following game between the challenger $\mathcal{C}$ and the adversary $\mathcal{A}$ as demonstrated below:

– **Setup**: $\mathcal{C}$ takes the security parameter λ as input, runs *Setup*(λ) and gives the resulting system parameters $PARAMS$ to $\mathcal{A}$.

– **Queries**: $\mathcal{A}$ issues the following queries adaptively to $\mathcal{C}$:
  - Uncorrupted-Key Generation: $\mathcal{C}$ runs the KeyGen($U_i, PARAMS$) algorithm to generate the public/private key pair (PK_i, SK_i) of user U_i and returns PK_i to $\mathcal{A}$.
  - Corrupted-Key Generation: $\mathcal{C}$ runs the KeyGen($U_i, PARAMS$) algorithm to generate the public/private key pair (PK_i, SK_i) of user U_i and returns (PK_i, SK_i) to $\mathcal{A}$.
  - Re-encryption Key Generation (PK_i, PK_j): $\mathcal{C}$ generates the public/private key pairs(corrupted or uncorrupted) of users U_i and U_j and runs the $ReKeyGen(SK_i, PK_i, PK_j, PARAMS)$ algorithm to generate the re-encryption key $RK_{i \to j}$ from users U_i to U_j.

– **Output**: $\mathcal{A}$ returns SK_i^* as the private key of public key PK_i^*. $\mathcal{A}$ wins the game if SK_i^* is a valid private key of an uncorrupted user with public key PK_i^*.

We refer to the above adversary $\mathcal{A}$ as the DSK adversary. The advantage of $\mathcal{A}$ in attacking the delegator secret security of the scheme is defined as:

$$Adv_{PRE,\mathcal{A}}^{DSK} = Pr[\mathcal{A}\ wins].$$

where, the probability is over the random coin tosses performed by $\mathcal{C}$ and $\mathcal{A}$. Given a t-time DSK adversary $\mathcal{A}$ who makes at most q_{RK} re-encryption key generation queries, a single-hop unidirectional PRE scheme is defined as (t, ϵ)-DSK secure if the advantage of $\mathcal{A}$ is $Adv_{PRE,\mathcal{A}}^{DSK} \leq \epsilon$.

Complexity Assumptions

We define the complexity assumptions used in the proof of security of our PRE scheme.

Definition 2. ***Computational Diffie Hellman Assumption (CDH):*** *Let $\mathbb{G}$ be a cyclic multiplicative group of prime order q. The Computational Diffie Hellman problem in $\mathbb{G}$ is, given $(g, g^a, g^b) \in \mathbb{G}^3$, compute g^{ab}, where $a, b \leftarrow \mathbb{Z}_q^*$.*

Definition 3. ***Divisible Computational Diffie Hellman Assumption (DCDH):*** *Let $\mathbb{G}$ be a cyclic multiplicative group of prime order q. The Divisible Computational Diffie Hellman problem in $\mathbb{G}$ is, given $(g, g^a, g^b) \in \mathbb{G}^3$, compute $g^{b/a}$, where $a, b \leftarrow \mathbb{Z}_q^*$.*

Definition 4. ***Discrete Logarithm Assumption (DL):*** *Let $\mathbb{G}$ be a cyclic multiplicative group of prime order q. The Discrete Logarithm problem in $\mathbb{G}$ is, given $(g, g^a) \in \mathbb{G}^2$, compute a, where $a \leftarrow \mathbb{Z}_q^*$.*

4 Analysis of a Unidirectional PRE Scheme by Chow *et al.*[8]

We review the scheme due to Chow *et al.* [8] and point out the weakness of the scheme in this section.

4.1 Review of the scheme

- **Setup**(λ): Choose two primes p and q such that $q|p-1$ and the security parameter λ defines the bit-length of q. Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_q^*$ with order q and let g be a generator of the group $\mathbb{G}$. Choose four hash functions:

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

  The hash functions H_1, H_2, H_3 are modelled as random oracles in the security proof reduction. Here l_0 and l_1 are security parameters determined by λ, and the message space $\mathcal{M}$ is $\{0,1\}^{l_0}$.
  Return the public parameters $PARAM = (q, \mathbb{G}, g, H_1, H_2, H_3, H_4, l_0, l_1)$.

- **KeyGen**$(U_i, PARAMS)$: To generate the private key (SK_i) and the corresponding public key (PK_i) of user U_i:
  - Pick $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and set $SK_i = (x_{i,1}, x_{i,2})$.
  - Compute $PK_i = (PK_{i,1}, PK_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$.

- **ReKeyGen**$(SK_i, PK_i, PK_j, PARAMS)$: On input of the private key of user U_i: $SK_i = (x_{i,1}, x_{i,2})$ and public key $PK_i = (PK_{i,1}, PK_{i,2})$ and user j's public key $PK_j = (PK_{j,1}, PK_{j,2})$, generate the re-encryption key $RK_{i\to j}$ as shown:
  - Pick $h \in_R \{0,1\}^{l_0}$, $\pi \in_R \{0,1\}^{l_1}$.
  - Compute $v = H_1(h, \pi)$.
  - Compute $V = PK_{j,2}^{v}$ and $W = H_2(g^v) \oplus (h||\pi)$.
  - Define $RK_{i\to j}^{\langle 1\rangle} = \frac{h}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}$.
  - Return $RK_{i\to j} = (RK_{i\to j}^{\langle 1\rangle}, V, W)$.

- **Encrypt**$(m, PK_i, PARAMS)$: To encrypt a message $m \in \mathcal{M}$ under the public key PK_i:
  - Pick $u \in_R \mathbb{Z}_q^*$.
  - Compute $D = \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^u$.
  - Pick $\omega \in_R \{0,1\}^{l_1}$.
  - Compute $r = H_1(m, \omega)$.
  - Compute $E = \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^r$ and $F = H_2(g^r) \oplus (m||\omega)$.
  - Compute $s = u + r \cdot H_3(D, E, F) \mod q$.
  - Output the ciphertext $\sigma_i = (D, E, F, s)$.

- **ReEncrypt**$(\sigma_i, PK_i, PK_j, RK_{i\to j}, PARAMS)$: On input of an original ciphertext $\sigma_i = (D, E, F, s)$ encrypted under the public key of the delegator $PK_i = (PK_{i,1}, PK_{i,2})$, the public key of the delegatee $PK_j = (PK_{j,1}, PK_{j,2})$, the re-encryption key $RK_{i\to j} = (RK_{i\to j}^{\langle 1\rangle}, V, W)$, re-encrypt σ_i into a ciphertext $\hat{\sigma_j}$ under PK_j as follows:
  - Check if the following condition holds to satisfy the well-formedness of ciphertexts:

$$\left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^s \stackrel{?}{=} D \cdot E^{H_3(D,E,F)} \tag{1}$$

    If it does not hold, return $\perp$.
  - Else, compute $\hat{E} = E^{RK_{i\to j}^{\langle 1\rangle}} = g^{rh}$.
  - Output $\hat{\sigma_j} = (\hat{E}, F, V, W) = (g^{rh}, H_2(g^r) \oplus (m||\omega), PK_{j,2}^{v}, H_2(g^v) \oplus (h||\pi))$.

- **Encrypt**$_1(m, PK_i, PARAMS)$: To generate a non-transformable ciphertext under public key PK_i of a message $m \in \mathcal{M}$:
  - Pick $h \in_R \{0,1\}^{l_0}$ and $\pi \in_R \{0,1\}^{l_1}$.
  - Compute $v = H_1(h, \pi)$.
  - Compute $V = PK_{j,2}^{v}$ and $W = H_2(g^v) \oplus (h||\pi)$.
  - Pick $\omega \in_R \{0,1\}^{l_1}$ and compute $r = H_1(m, \omega)$.
  - Compute $\hat{E} = (g^r)^h$ and $F = H_2(g^r) \oplus (m||\omega)$.
  - Output the non-transformable ciphertext $\hat{\sigma_j} = (\hat{E}, F, V, W)$.

– **Decrypt**$(\sigma_i, PK_i, SK_i, PARAMS)$: On input of a ciphertext σ_i, public key PK_i and its corresponding private key $SK_i = (x_{i,1}, x_{i,2})$,decrypt according to two cases:
  - Original Ciphertext $\sigma_i = (D, E, F, s)$:
    - ∗ If equation (1) does not hold, return $\perp$.
    - ∗ Otherwise, compute:

$$(m||\omega) = F \oplus H_2(E^{\frac{1}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}}). \tag{2}$$

    - ∗ Return m if $E \overset{?}{=} \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{H_1(m,\omega)}$ holds; else return $\perp$.
  - Transformed /Non-transformable Ciphertext $\hat{\sigma_i} = (\hat{E}, F, V, W)$:
    - ∗ Compute $(h||\pi) = W \oplus H_2(V^{1/SK_{i,2}})$ and $(m||\omega) = F \oplus H_2(\hat{E}^{1/h})$.
    - ∗ Return m if $V \overset{?}{=} PK_{i,2}^{H_1(h,\pi)}$ and $\hat{E} \overset{?}{=} g^{H_1(m,\omega)\cdot h}$ holds; else return $\perp$.

4.2 Our Attack

In this section, we point out the weakness of the scheme by Chow *et al.* [8]. We show that the simulation of the oracles defined in the security proof of the scheme is not consistent with the real algorithm. This allows the adversary to distinguish the simulation run by the challenger from the real system. We demonstrate this flaw by considering the validity of the ciphertexts with respect to the *ReEncrypt* and *Decrypt* algorithm in the simulation and in the real system. To make it simple, we consider PK_T as the public key of the target user in the challenge phase and the attack is posed in **Phase-II** after the challenge phase is over. We re-encrypt a ciphertext σ_T under PK_T into a ciphertext $\hat{\sigma}_j$ under PK_j (PK_j is corrupt) and further decrypt $\hat{\sigma}_j$. All the computations hereafter are done using PK_T and PK_j. Before we explain the flaw, let us partially review the re-encryption and decryption oracles of the scheme in [7] which is relevant to our attack. We note that the re-encryption keys are maintained in a list R^{list} in the form of tuples $\langle PK_T, PK_j, (RK^{\langle 1 \rangle}_{T\to j}, V, W), h, \tau\rangle$. In order to respond to the random oracle queries of the adversary, the challenger maintains list H_1^{list} with tuples $\langle m, \omega, r\rangle$ such that $H_1(m, \omega) = r$ and list H_2^{list} with tuples $\langle R, \beta\rangle$ such that $H_2(R) = \beta$. The description of the re-encryption and decryption oracles are as follows:

– Re-encryption Oracle $(\mathcal{O}_{RE}(PK_T, PK_j, \sigma_T))$:
  - Parse σ_T to obtain (D, E, F, s). Note that σ_T (U_T is honest) is to be re-encrypted to $\hat{\sigma}_j$ (U_j is corrupt).
  - Search for tuple $\langle m, \omega, r\rangle$ in $H_1{}^{list}$ such that $(PK_{T,1}^{H_4(PK_{T,2})} PK_{T,2})^r \overset{?}{=} E$. If no such tuple exists, return $\perp$.
  - Else, retrieve tuple $\langle PK_T, PK_j, (*, V, W), h, '-'\rangle$ in list R^{list}.
  - If found, compute $\hat{E} = E^{r\cdot h}$.
  - Else, prepare the partial re-encryption key as follows:
    - ∗ Pick $h \in_R \{0,1\}^{l_0}$, $\pi \in_R \{0,1\}^{l_1}$. Compute $v = H_1(h, \pi)$.
    - ∗ Compute $V = PK_{j,2}^v$ and $W = H_2(g^v) \oplus (h||\pi)$.
    - ∗ Update list R^{list} with tuple $\langle PK_i, PK_j, (\perp, V, W), h, '-'\rangle$, define $\hat{E} = E^{r\cdot h}$.
  - Return $\hat{\sigma}_j = (\hat{E}, F, V, W)$ to $\mathcal{A}$.

– Decryption Oracle $(\mathcal{O}_{Dec}(PK_T, \sigma_T))$ or $(\mathcal{O}_{Dec}(PK_j, \hat{\sigma}_j))$:
  - To decrypt an original ciphertext $\sigma_T = (D, E, F, s)$ under the public key PK_T (uncorrupt), search lists H_1^{list} and H_2^{list} for tuples $\langle m, \omega, r\rangle$ and $\langle R, \beta\rangle$ such that:

$$(PK_{T,1}^{H_4(PK_{i,2})} PK_{T,2})^r \overset{?}{=} E,\ \beta \oplus (m||\omega) \overset{?}{=} F,\ R \overset{?}{=} g^r. \tag{3}$$

    If yes, return m to the adversary. Else, return $\perp$.
  - To decrypt a transformed ciphertext $\hat{\sigma}_j = (\hat{E}, F, V, W)$ under public key PK_j (corrupt), run algorithm *Decrypt*$(\hat{\sigma}_j, PK_j, SK_j, PARAMS)$ and return the result to $\mathcal{A}$.

First, we encrypt a message m under PK_T. Let us consider two forms of ciphertext $\sigma_{Real} = \langle D_{Real}, E_{Real}, F_{Real}, s_{Real}\rangle$ and $\sigma_{Fake} = \langle D_{Fake}, E_{Fake}, F_{Rand}, s_{Fake}\rangle$. σ_{Real} is the ciphertext obtained from the encryption algorithm **Encrypt** (i.e., encryption of m under PK_T by executing the **Encrypt**$(m, PK_T, PARAMS)$). σ_{Fake} is a cooked-up ciphertext that can pass the verification tests of *ReEncrypt* algorithm but not the *Decrypt* algorithm. We list down the steps to construct σ_{Real} and σ_{Fake}.

– **Construction of** σ_{Real}: σ_{Real} is the encryption of a message m under the public key PK_T obtained from the real system, i.e., $\sigma_{Real} \leftarrow$ Encrypt$(m, PK_T, PARAMS)$. The construction of σ_{Real} is as follows:

- Pick $u_{Real} \in_R \mathbb{Z}_q^*$.
- Compute $D_{Real} = \Big((PK_{T,1})^{H_4(PK_{T,2})} PK_{T,2}\Big)^{u_{Real}}$.
- Pick $\omega_{Real} \in_R \{0,1\}^{l_1}$.
- Compute $r_{Real} = H_1(m, \omega_{Real})$.
- Compute $E_{Real} = \Big((PK_{T,1})^{H_4(PK_{T,2})} PK_{T,2}\Big)^{r_{Real}}$.
- Compute $F_{Real} = H_2(g^{r_{Real}}) \oplus (m||\omega_{Real})$.
- Compute $s_{Real} = u_{Real} + r_{Real} H_3(D_{Real}, E_{Real}, F_{Real}) \mod q$.
- Output the ciphertext $\sigma_{Real} = (D_{Real}, E_{Real}, F_{Real}, s_{Real})$. It is clear that σ_T passes the ciphertext verification test of equation (1).

– **Construction of** σ_{Fake}: σ_{Fake} is a cooked-up ciphertext that clears the verification tests of the *ReEncrypt* algorithm but fails against the verification of the *Decrypt* algorithm. We denote the algorithm for the construction of σ_{Fake} as $\mathbf{Encrypt}_{Fake}(m, PK_T)$, which is as follows:
  - Pick $u_{Fake} \in_R \mathbb{Z}_q^*$.
  - Compute $D_{Fake} = \Big((PK_{T,1})^{H_4(PK_{T,2})} PK_{T,2}\Big)^{u_{Fake}}$.
  - Pick $r_{Rand} \in_R \mathbb{Z}_q^*$. Here it should be noted that r_{Rand} does not follow the actual algorithm, instead it is picked at random from $\mathbb{Z}_q^*$.
  - Pick $\omega_{Fake} \in_R \{0,1\}^{l_1}$ and compute $r_{Fake} = H_1(m, \omega_{Fake})$. Note that in the **Encrypt** algorithm, r_{Fake} is the output of H_1 oracle on giving a message and a random string (ω_{Fake}) of size $\{0,1\}^{l_1}$ as input.
  - Compute $E_{Fake} = \Big((PK_{T,1})^{H_4(PK_{T,2})} PK_{T,2}\Big)^{r_{Rand}}$.
  - Choose $F_{Rand} \in_R \{0,1\}^{l_0+l_1}$. In the **Encrypt** algorithm, F is the encryption of the message to be encrypted along with a random string ω_{Fake} of length $\{0,1\}^{l_1}$. But in the construction of σ_{Fake}, we note that F_{Rand} is chosen at random.
  - Compute $s_{Fake} = u_{Fake} + r_{Rand} H_3(D_{Fake}, E_{Fake}, F_{Rand}) \mod q$.
  - Output the ciphertext $\sigma_{Fake} = (D_{Fake}, E_{Fake}, F_{Rand}, s_{Fake})$. It should be noted that the components F_{Rand} and r_{Rand} violate the definitions given in the **Encrypt** algorithm. However, σ_{Fake} passes the ciphertext validation test of equation (1). In fact:

$$\begin{aligned} RHS &= D_{Fake} \cdot (E_{Fake})^{H_3(D_{Fake}, E_{Fake}, F_{Rand})} \\ &= \Big((PK_{T,1})^{H_4(PK_{T,2})} PK_{T,2}\Big)^{u_{Fake}} \cdot \Big((PK_{T,1})^{H_4(PK_{T,2})} PK_{T,2}\Big)^{r_{Rand} H_3(D_{Fake}, E_{Fake}, F_{Rand})} \\ &= \Big((PK_{T,1})^{H_4(PK_{T,2})} PK_{T,2}\Big)^{s_{Fake}} \\ &= LHS. \end{aligned}$$

The important properties possessed by σ_{Real} and σ_{Fake} are:

1. Output of $\mathbf{Decrypt}(\sigma_{Real}, PK_T, SK_T, PARAMS)$ is m and the output of $\mathcal{O}_{Decrypt}(PK_T, \sigma_{Real})$ is m. This is because σ_{Real} is a legitimate ciphertext of m produced by *Encrypt* algorithm.
2. Output of $\mathbf{Decrypt}(\sigma_{Fake}, PK_T, SK_T, PARAMS)$ is $\perp$ and the output of the $\mathcal{O}_{Decrypt}(PK_T, \sigma_{Fake})$ is $\perp$. This is because:
   – In **Decrypt** algorithm: Since $F_{Rand} \in_R \{0,1\}^{l_0+l_1}$, according to equation (2), we obtain $(m_k||\omega_k) \leftarrow F \oplus H_2(E^{\frac{1}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}})$ where $(m_k||\omega_k)$ is a junk message which does not satisfy the validity check: $E \neq (PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2})^{H_1(m_k, \omega_k)}$. Consequently, $\perp$ is returned.
   – In $\mathcal{O}_{Decrypt}$ Simulation: Note that r_{Rand} is used for the construction of σ_{Fake} and there exists no tuples $\langle m, \omega_{Fake}, r\rangle \in H_1^{list}$ and $\langle R, \beta\rangle \in H_2^{list}$ such that condition (3) holds. Hence, $\perp$ is returned.
3. σ_{Real} is a valid ciphertext and σ_{Fake} is an invalid ciphertext with respect to both **Decrypt** algorithm and $\mathcal{O}_{Decrypt}$ oracle. Therefore, the simulation of the decryption algorithm is perfect.
4. Both σ_{Real} and σ_{Fake} are valid ciphertexts corresponding to the **ReEncrypt** algorithm. This is because σ_{Real} is a legitimate ciphertext of m produced by the *Encrypt* algorithm. Again, σ_{Fake} passes the ciphertext verification test of equation (1) and the algorithm computes the re-encrypted ciphertext $\hat{\sigma}_{Fake} = (\hat{E}, F, V, W)$ as per the protocol where $\hat{E}_{Fake} = g^{r_{Fake}h}$, where $r_{Fake} = H_1(m, \omega_{Fake})$ has already been computed.

Next, we re-encrypt both σ_{Real} and σ_{Fake} under the public key PK_j of a corrupt user. Let us consider the following notations. We use $\hat{\sigma}_{Real}^{(Scheme)}$ to denote the result of re-encryption of σ_{Real} obtained from the re-encryption algorithm

ReEncrypt i.e., $\hat{\sigma}^{(Scheme)}_{Real} \leftarrow$ **ReEncrypt**$(\sigma_{Real}, PK_T, PK_j, RK_{T\rightarrow j}, PARAMS)$ and $\hat{\sigma}^{(Oracle)}_{Real}$ to denote the result of re-encryption of σ_{Real} obtained from the re-encryption oracle $\mathcal{O}_{ReEncrypt}$ i.e., $\hat{\sigma}^{(Oracle)}_{Real} \leftarrow \mathcal{O}_{ReEncrypt}(PK_T, PK_j, \sigma_{Real})$. Also, we use $\hat{\sigma}^{(Scheme)}_{Fake}$ to denote the result of re-encryption of σ_{Fake} obtained from the re-encryption algorithm **ReEncrypt** i.e., $\hat{\sigma}^{(Scheme)}_{Fake} \leftarrow$**ReEncrypt**$(\sigma_{Fake}, PK_T, PK_j, RK_{T\rightarrow j}, PARAMS)$ and $\hat{\sigma}^{(Oracle)}_{Fake}$ to denote the result of re-encryption of σ_{Fake} obtained from the re-encryption oracle $\mathcal{O}_{ReEncrypt}$ i.e., $\hat{\sigma}^{(Oracle)}_{Fake} \leftarrow \mathcal{O}_{ReEncrypt}(PK_T, PK_j, \sigma_{Fake})$.

Observations on $\hat{\sigma}^{(Scheme)}_{Real}$ and $\hat{\sigma}^{(Oracle)}_{Real}$:

1. $\hat{\sigma}^{(Scheme)}_{Real} = \hat{\sigma}^{(Oracle)}_{Real}$.
2. $\hat{\sigma}^{(Scheme)}_{Fake} \neq \hat{\sigma}^{(Oracle)}_{Fake}$.

The reason for observation 1 follows directly from the fact that σ_{Real} is a valid ciphertext. The reason for the violation in observation 2 is that the **ReEncrypt** algorithm is only a function of the re-encryption key but $\mathcal{O}_{ReEncrypt}$ oracle makes use of the knowledge of r_{Fake} to generate $\hat{\sigma}^{(Oracle)}_{Fake}$. However, in the construction of σ_{Fake}, r_{Rand} is used in the generation of $\hat{\sigma}^{(Oracle)}_{Fake}$. The question here is, how will the adversary find this difference, that is $\hat{\sigma}^{(Scheme)}_{Fake} \neq \hat{\sigma}^{(Oracle)}_{Fake}$. Let us now demonstrate how the adversary captures this difference shown by the $\mathcal{O}_{ReEncrypt}$ oracle simulation and the **ReEncrypt** algorithm. We first analyse the ciphertexts $\hat{\sigma}^{(Scheme)}_{Fake}$ and $\hat{\sigma}^{(Oracle)}_{Fake}$.

Closer Look at $\hat{\sigma}^{(Scheme)}_{Fake}$:

– $\hat{\sigma}^{(Scheme)}_{Fake} = \left(\hat{E}^{(Scheme)}_{Fake}, F^{(Scheme)}_{Rand}, V^{(Scheme)}_{Fake}, W^{(Scheme)}_{Fake}\right) \leftarrow$ **ReEncrypt**$(\sigma_{Fake}, PK_T, PK_j, RK_{T\rightarrow j}, PARAMS)$.
– $\hat{E}^{Scheme}_{Fake} = E^{RK_{i\rightarrow j}}_{Fake} = \left(PK^{H_4(PK_{T,2})}_{T,1} PK_{T,2}\right)^{r_{Rand}\left(\frac{h}{x_{T,1}H_4(PK_{T,2})+x_{T,2}}\right)} = (g^{r_{Rand}})^h$.
– $v = H_1(h, \pi)$ where $h \in_R \{0,1\}^{l_0}$ and $\pi \in_R \{0,1\}^{l_1}$.
– $V_{Fake} = PK^v_{j,2}$ and $W_{Fake} = H_2(g^v) \oplus (h||\pi)$.
– So, we have $\hat{\sigma}^{(Scheme)}_{Fake} = \left(g^{r_{Rand}h}, F_{Rand}, PK^v_{j,2}, H_2(g^v) \oplus (h||\pi)\right)$.

Closer look at $\hat{\sigma}^{(Oracle)}_{Fake}$:

– $\perp \leftarrow \mathcal{O}_{ReEncrypt}(PK_T, PK_j, \sigma_{Fake})$.
Since there exists no tuples $\langle m, \omega, r\rangle$ in H^{list}_1 such that $(PK^{H_4(PK_{T,2})}_{T,1} PK_{T,2})^r \stackrel{?}{=} E$, the oracle returns $\perp$. This is because, although there exists a tuple $\langle m, \omega_{Fake}, r_{Fake}\rangle$ in H^{list}_1, r_{Rand} is used in the construction of E_{Fake}.

Distinguishing The Oracle From The Real Algorithm:

Using the above observations, the adversary can easily distinguish between the real algorithm and the simulated environment provided by the challenger $\mathcal{C}$. The adversary $\mathcal{A}$ can perform the following simple test to distinguish the simulated $\mathcal{O}_{ReEncrypt}$ and the real algorithm **ReEncrypt**:

Distinguisher

1. $\mathcal{C}$ provides the system parameters $PARAMS$ to $\mathcal{A}$.
2. After getting training in **Phase-I**, $\mathcal{A}$ provides two messages m_0 and m_1 of equal length and a target public key PK_T to $\mathcal{C}$.
3. $\mathcal{C}$ generates the challenge ciphertext σ_T and gives as challenge to $\mathcal{A}$.
4. $\mathcal{A}$ now does the following:
   (a) Generate $\sigma_{Fake} =$**Encrypt**$_{Fake}(m_0, PK_T) = (D_{Fake}, E_{Fake}, F_{Rand}, s_{Fake})$ where:
      – $D_{Fake} = \left((PK_T)^{H_4(PK_T)} PK_T\right)^{u_{Fake}}$.
      – $E_{Fake} = \left((PK_T)^{H_4(PK_T)} PK_T\right)^{r_{Rand}}$ where $r_{Rand} \in_R \mathbb{Z}^*_q$.
      – $F_{Rand} \in_R \{0,1\}^{l_0+l_1}$.
      – $s_{Fake} = u_{Fake} + r_{Rand}H_3(D_{Fake}, E_{Fake}, F_{Rand}) \mod q$.
   Here $\mathcal{A}$ knows r_{Rand} and u_{Fake}.
   (b) $\mathcal{A}$ queries $\mathcal{O}_{ReEncrypt}(\sigma_{Fake}, PK_T, PK_j, RK_{T\rightarrow j}, PARAMS)$. It should noted that v, V, h, π, W are fixed for $T \rightarrow j$ delegation.
   (c) **Test**: If $\perp \leftarrow \mathcal{O}_{ReEncrypt}((\sigma_{Fake}, PK_T, PK_j, RK_{T\rightarrow j}, PARAMS))$, then $ReEncrypt \neq \mathcal{O}_{ReEncrypt}$ and $\mathcal{A}$ knows that it is not the real system and will abort. Else, $\mathcal{A}$ learns no clue about the simulation.

4.3 Fixing the flaw

Note that modifying the re-encryption algorithm to fix the flaw is not possible since re-encryption of a valid ciphertext σ_T will always require the knowledge of $r = H_1(m, \omega)$ as no other trapdoor exists to obtain a re-encrypted ciphertext $\hat{\sigma}_j$. Again, the knowledge of the private key of the delegator is necessary to generate the re-encryption keys and re-encrypted ciphertexts. Consequently, we cannot provide a trivial fix to the scheme in order to address the problem. As a solution, we propose a new collusion-resistant unidirectional proxy re-encryption scheme without any pairing operation. We have incorporated additional information to the existing *Encrypt* algorithm along with ciphertext validity checks in both the *Re-Encrypt* and the *Decrypt* algorithm. Our technique prevents any cooked up ciphertexts violating the definitions of *Encrypt* algorithm to clear the ciphertext validity checks of both the *Re-Encrypt* and *Decrypt* algorithms.

5 A Unidirectional Proxy Re-Encryption Scheme

- **Setup**(λ): Choose two primes p and q such that $q|p-1$ and the bit-length of q is the security parameter λ. Let $\mathbb{G}$ be a subgroup of $\mathbb{Z}_q^*$ with order q. g is a generator of the group $\mathbb{G}$. Choose five hash functions:

$$
\begin{aligned}
&H_1 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \to \mathbb{Z}_q^*, \\
&H_2 : \mathbb{G} \to \{0,1\}^{l_0+l_1}, \\
&H_3 : \{0,1\}^* \to \mathbb{Z}_q^* \\
&H_4 : \mathbb{G} \to \mathbb{Z}_q^*, \\
&H_5 : \mathbb{G}^4 \times \{0,1\}^{l_0+l_1} \to \mathbb{G}.
\end{aligned}
$$

  The hash functions are modelled as random oracles in the security proof reduction. Here l_0 and l_1 are security parameters determined by λ, and the message space $\mathcal{M}$ is $\{0,1\}^{l_0}$.
  Return the public parameters $PARAMS = (q, \mathbb{G}, g, H_1, H_2, H_3, H_4, H_5, l_0, l_1)$.

- **KeyGen**($U_i, PARAMS$): To generate the private key (SK_i) and the corresponding public key (PK_i) of user U_i:
  - Pick $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and set $SK_i = (x_{i,1}, x_{i,2})$.
  - Compute $PK_i = (PK_{i,1}, PK_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$.

- **ReKeyGen**($SK_i, PK_i, PK_j, PARAMS$): On input of a user i's private key $SK_i = (x_{i,1}, x_{i,2})$ and public key $PK_i = (PK_{i,1}, PK_{i,2})$ and user j's public key $PK_j = (PK_{j,1}, PK_{j,2})$, generate the re-encryption key $RK_{i\to j}$ as shown:
  - Pick $h \in_R \{0,1\}^{l_0}$, $\pi \in_R \{0,1\}^{l_1}$.
  - Compute $v = H_1(h, \pi)$.
  - Compute $V = PK_{j,2}^{v}$ and $W = H_2(g^v) \oplus (h||\pi)$.
  - Define $RK_{i\to j}^{\langle 1\rangle} = \frac{h}{x_{i,1}H_4(PK_{i,2}) + x_{i,2}}$.
  - Return $RK_{i\to j} = (RK_{i\to j}^{\langle 1\rangle}, V, W)$.

- **Encrypt**($m, PK_i, PARAMS$): To encrypt a message $m \in \mathcal{M}$:
  - Pick $u \in_R \mathbb{Z}_q^*$.
  - Compute $D = \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^u$.
  - Compute $\bar{D} = H_5(PK_{i,1}, PK_{i,2}, D, E, F)^u$.
  - Pick $\omega \in_R \{0,1\}^{l_1}$.
  - Compute $r = H_1(m, \omega)$.
  - Compute $E = \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^r$.
  - Compute $\bar{E} = H_5(PK_{i,1}, PK_{i,2}, D, E, F)^r$.
  - Compute $F = H_2(g^r) \oplus (m||\omega)$.
  - Compute $s = u + r \cdot H_3(D, \bar{E}, F) \mod q$.
  - Output the ciphertext $\sigma_i = (D, \bar{E}, F, s)$.

- **ReEncrypt**($\sigma_i, PK_i, PK_j, RK_{i\to j}, PARAMS$): On input of an original ciphertext $\sigma_i = (D, \bar{E}, F, s)$ encrypted under public key $PK_i = (PK_{i,1}, PK_{i,2})$, the public keys PK_i and PK_j, a re-encryption key $RK_{i\to j} = (RK_{i\to j}^{\langle 1\rangle}, V, W)$, re-encrypt σ_i into a ciphertext $\hat{\sigma_j}$ under the public key $PK_j = (PK_{j,1}, PK_{j,2})$ as follows:

- Compute E and $\bar{D}$ as follows:

$$\begin{aligned}
E &= \left(\left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{s} \cdot D^{-1}\right)^{H_3(D,\bar{E},F)^{-1}} \\
&= \left(\left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{(u+r\cdot H_3(E,\bar{E},F))} \cdot \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{-u}\right)^{H_3(D,\bar{E},F)^{-1}} \\
&= \left(\left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{rH_3(D,\bar{E},F)}\right)^{H_3(D,\bar{E},F)^{-1}} \\
&= \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{r}.
\end{aligned}$$

$$\begin{aligned}
\bar{D} &= H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{s} \cdot (\bar{E}^{H_3(D,\bar{E},F)})^{-1} \\
&= H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{(u+r\cdot H_3(D,\bar{E},F))} \cdot \left(H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{rH_3(D,\bar{E},F)}\right)^{-1} \\
&= H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{u}.
\end{aligned}$$

- Check if the following verification for the well-formedness of the ciphertext holds:

$$\left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{s} \stackrel{?}{=} D \cdot E^{H_3(D,\bar{E},F)} \tag{4}$$

$$\left(H_5(PK_{i,1}, PK_{i,2}, D, E, F)\right)^{s} \stackrel{?}{=} \bar{D} \cdot \bar{E}^{H_3(D,\bar{E},F)} \tag{5}$$

  If the above checks do not hold, return $\perp$.
- Else, compute $\bar{E} = E^{RK_{i\to j}^{\langle 1\rangle}} = g^{rh}$.
- Output $\hat{\sigma_j} = (\bar{E}, F, V, W) = (g^{r\cdot h}, H_2(g^r) \oplus (m||\omega),\ PK_{j,2}^{v}, H_2(g^v) \oplus (h||\pi))$.

– **Encrypt**$_1(m, PK_i, PARAMS)$: To generate a non-transformable ciphertext under public key PK_i of a message $m \in \mathcal{M}$:
- Pick $h \in_R \{0,1\}^{l_0}$ and $\pi \in_R \{0,1\}^{l_1}$.
- Compute $v = H_1(h, \pi)$.
- Compute $V = PK_{j,2}^{v}$ and $W = H_2(g^v) \oplus (h||\pi)$.
- Pick $\omega \in_R \{0,1\}^{l_1}$ and compute $r = H_1(m, \omega)$.
- Compute $\hat{E} = (g^r)^h$ and $F = H_2(g^r) \oplus (m||\omega)$.
- Output the non-transformable ciphertext $\hat{\sigma_j} = (\hat{E}, F, V, W)$.

– **Decrypt**$(\sigma_i, PK_i, SK_i, PARAMS)$: On input a ciphertext σ_i, public key PK_i and private key $SK_i = (x_{i,1}, x_{i,2})$, decrypt according to two cases:
- Original ciphertext of the form $\sigma_i = (D, \bar{E}, F, s)$:
  - Check if the ciphertext is well-formed by computing the values of E and $\bar{D}$ and checking if equations (4) and (5) holds. If they do not hold, return $\perp$.
  - Otherwise, extract the message as:

$$(m||\omega) = F \oplus H_2(E^{\frac{1}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}}) \tag{6}$$

  - Return m if the following checks hold, else return $\perp$.

$$\begin{aligned}
D &\stackrel{?}{=} \left(PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2}\right)^{s} \cdot (E)^{H_5(D,\bar{E},F)^{-1}} \\
\bar{E} &\stackrel{?}{=} H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{H_1(m,\omega)}
\end{aligned}$$

- Transformed ciphertext or a non-transformable ciphertext of the form $\sigma_i = (\hat{E}, F, V, W)$:
  - Compute $(h||\pi) = W \oplus H_2(V^{1/SK_{i,2}})$ and extract the message as:

$$(m||\omega) = F \oplus H_2(\hat{E}^{1/h}) \tag{7}$$

  - Return m if $V \stackrel{?}{=} PK_{i,2}^{H_1(h,\pi)}$ and $\hat{E} \stackrel{?}{=} g^{H_1(m,\omega)\cdot h}$ holds; else return $\perp$.

5.1 Correctness

– Correctness of ciphertext verification from equation (4):

$$\begin{aligned}
RHS &= D \cdot E^{H_3(D,\bar{E},F)} \\
&= (PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2})^{u} \cdot (PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2})^{rH_3(D,\bar{E},F)} \\
&= (PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2})^{u+rH_3(D,\bar{E},F)} \\
&= (PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2})^{s} \\
&= LHS.
\end{aligned}$$

– Correctness of ciphertext verification from equation (5):

$$\begin{aligned}
RHS &= \bar{D} \cdot \bar{E}^{H_3(D,\bar{E},F)} \\
&= H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{u} \cdot H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{rH_3(D,\bar{E},F)} \\
&= H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{u+rH_3(D,\bar{E},F)} \\
&= H_5(PK_{i,1}, PK_{i,2}, D, E, F)^{s} \\
&= LHS.
\end{aligned}$$

– Consistency of *Encryption* and *Decryption* of Original Ciphertext from equation (6):

$$\begin{aligned}
RHS &= F \oplus H_2(E^{\frac{1}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}}) \\
&= F \oplus H_2((PK_{i,1}^{H_4(PK_{i,2})} PK_{i,2})^{r\times \frac{1}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}}) \\
&= H_2(g^r) \oplus (m||\omega) \oplus H_2(g^r) \\
&= (m||\omega) \\
&= LHS.
\end{aligned}$$

– Consistency of *Encryption* and *Decryption* of Transformed/Non-transformable ciphertext from equation (7):

$$\begin{aligned}
RHS &= F \oplus H_2(\hat{E}^{\frac{1}{h}}) \\
&= H_2(g^r) \oplus (m||\omega) \oplus H_2(g^{rh\times\frac{1}{h}}) \\
&= (m||\omega).
\end{aligned}$$

5.2 Security Proof

Original Ciphertext Security:

Theorem 1. *The proposed scheme is CCA-secure for the original ciphertext under the DCDH assumption and the $EUF-CMA$ security of Schnorr signature scheme [19]. If a (t,ϵ)IND-PRE-CCA $\mathcal{A}$ with an advantage ϵ breaks the IND-PRE-CCA security of the given scheme in time t, $\mathcal{C}$ can solve the DCDH problem with advantage ϵ' within time t' where:*

$$\begin{aligned}
\epsilon' &\geq \frac{1}{q_{H_2}}\Big(\frac{\epsilon}{e(q_{RK}+1)} - \frac{q_{H_1}}{2^{l_1}} - \frac{q_{H_3}+q_{H_5}}{2^{l_0+l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big) - \epsilon_1 - \epsilon_2\Big), \\
t' &\leq t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + n_h + n_c + q_{RK} + q_{RE} + q_d)O(1) \\
&\quad + (2n_h + 2n_c + 2q_{RK} + 5q_{RE} + 2q_d + q_{H_1}q_{RE} + (2q_{H_2} + 2q_{H_1})q_d)t_e,
\end{aligned}$$

We note that e is the base of natural logarithm, ϵ_1 denotes the advantage in breaking the CCA security of the hashed Elgamal encryption scheme and ϵ_2 denotes the advantage in breaking the EUF-CMA security of the Schnorr Signature scheme and t_e denotes the time taken for exponentiation in group $\mathbb{G}$.

Proof. Given an adversary $\mathcal{A}$ that breaks the (t,ϵ)*IND-PRE-CCA* security of the scheme, we show how to construct a polynomial time algorithm $\mathcal{C}$ which breaks the *DCDH* assumption in $\mathcal{G}$ or the existential unforgeability against chosen message attack (*EUF-CMA*) of the Schnorr Signature with a non-negligible advantage. $\mathcal{C}$ maintains two lists L_{key} and L_{RK} to store the lists of the public/private key pairs and the re-encryption keys of the users respectively. Both the lists are initially empty consisting of tuples of the form:

- L_{key} : $\langle PK_i, x_{i,1}, x_{i,2}, c_i \rangle$.
- L_{RK} : $\langle PK_i, PK_j, RK^{\langle 1 \rangle}_{i \to j}, V, W, h, \tau \rangle$.

– **Key Generation:** $\mathcal{C}$ maintains the list L_{key} that contains information of all the user keys. $\mathcal{C}$ generates the keys of the users in the following ways:
  - Uncorrupted User Keys: $\mathcal{C}$ uses Coron's coin tossing technique [9] to generate the uncorrupted user keys by flipping a coin $c_i \in \{0, 1\}$ that takes the value 1 with probability ρ, which we shall determine later. $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes PK_i according to the following cases:
    - ∗ If $c_i = 1$, compute $PK_i = (PK_{i,1}, PK_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$. Add the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 1 \rangle$ to L_{key}.
    - ∗ If $c_i = 0$, compute $PK_i = (PK_{i,1}, PK_{i,2}) = ((g^a)^{x_{i,1}}, (g^a)^{x_{i,2}})$. Add the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 0 \rangle$ to L_{key}.
  - Corrupted User Keys: $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes $PK_i = (PK_{i,1}, PK_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$. It adds the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = - \rangle$ to L_{key}.

– **Phase 1:** $\mathcal{C}$ answers the queries issues by $\mathcal{A}$ as follows:
  - Oracle Queries:

    $H_1(m, \omega)$: $\mathcal{C}$ maintains a list L_{H_1} with tuples $\langle m, \omega, r \rangle$. If a tuple $\langle m, \omega, r \rangle$ already appears in L_{H_1} list, respond with $H_1(m, \omega) = r$. Else pick $r \in_R \mathbb{Z}_q^*$, add the tuple $\langle m, \omega, r \rangle$ to L_{H_1} and return r.

    $H_2(R)$: $\mathcal{C}$ maintains a list L_{H_2} with tuples $\langle R, \theta \rangle$. If a tuple $\langle R, \theta \rangle$ already appears in L_{H_2} list, respond with $H_2(R) = \theta$. Else pick $\theta \in_R \{0, 1\}^{l_0 + l_1}$, add the tuple $\langle R, \theta \rangle$ to L_{H_2} and return θ.

    $H_3(D, \bar{E}, F)$: $\mathcal{C}$ maintains a list L_{H_3} with tuples $\langle D, \bar{E}, F, \psi \rangle$. If a tuple $\langle D, \bar{E}, F, \psi \rangle$ already appears in L_{H_3} list, respond with $H_3(D, \bar{E}, F) = \psi$. Else pick $\psi \in_R \mathbb{Z}_q^*$, add the tuple $\langle D, \bar{E}, F, \psi \rangle$ to L_{H_3} and return ψ.

    $H_4(PK_{i,2})$: $\mathcal{C}$ maintains a list L_{H_4} with tuples $\langle PK_{i,2}, \nu \rangle$. If a tuple $\langle PK_{i,2}, \nu \rangle$ already appears in L_{H_4} list, respond with $H_4(PK_{i,2}) = \nu$. Else pick $\nu \in_R \mathbb{Z}_q^*$, add the tuple $\langle PK_{i,2}, \nu \rangle$ to L_{H_4} and return ν.

    $H_5(PK_{i,1}, PK_{i,2}, D, E, F)$: $\mathcal{C}$ maintains a list L_{H_5} with tuples $\langle PK_{i,1}, PK_{i,2}, D, E, F, \beta, \gamma \rangle$. If a tuple $\langle PK_{i,1}, PK_{i,2}, D, E, F, \beta, \gamma \rangle$ already appears in L_{H_5} list, respond with $H_5(PK_{i,1}, PK_{i,2}, D, E, F) = \gamma$. Else, pick $\beta \in_R \mathbb{Z}_q^*$. Compute $\gamma = g^\beta$ and set $H_5(PK_{i,1}, PK_{i,2}, D, E, F) = \gamma$.

  - $\mathcal{O}_{ReKeyGen}(PK_i, PK_j)$: $\mathcal{C}$ maintains the list L_{RK} that contains information of the re-encryption keys of the users. If the re-encryption keys from PK_i to PK_j already exists in L_{RK}, retrieve and return $RK_{i \to j} = (RK^{\langle 1 \rangle}_{i \to j}, V, W)$. Else, $\mathcal{C}$ computes the re-encryption keys as shown:
    - ∗ Select $h \in_R \{0, 1\}^{l_0}$, $\pi \in_R \{0, 1\}^{l_1}$.
    - ∗ Compute $v = H_1(h, \pi)$, $V = (PK_{j,2})^v$, $W = H_2(g^v) \oplus (h || \pi)$.
    - ∗ Compute the value of $RK^{\langle 1 \rangle)}_{i \to j}$ according to the following cases:
      - · If $c_i = 1 \vee c_i = -$, compute $RK^{\langle 1 \rangle}_{i \to j} = \frac{h}{x_{i,1} H_4(PK_{i,2}) + x_{i,2}}$. Set $\tau = 1$ and update the list L_{RK}.
      - · If $c_i = 0 \wedge (c_j \in \{0, 1\})$, pick $RK^{\langle 1 \rangle}_{i \to j} \in_R \mathbb{Z}_q^*$ and set $\tau = 0$. Since a random value of $RK_{i \to j}$ would not match the value h associated with (V, W), we rely on the security of hashed Elgamal encryption scheme [3,11,12] as shown in the security proof of original paper [7].
      - · If $c_i = 0 \wedge c_j = -$, abort and report failure.
    - ∗ Return the re-encryption key $RK_{i \to j} = (RK^{\langle 1 \rangle}_{i \to j}, V, W)$ to $\mathcal{A}$.

  - $\mathcal{O}_{ReEncrypt}(\sigma_i, PK_i, PK_j)$: Check for the well-formedness of σ_i by computing E and $\bar{D}$ and verifying if equations (4) and (5) hold *(both the checks ensure that only legitimate ciphertexts obeying the definition of the Encrypt algorithm can clear the verification condition)*. If they do not hold, return $\perp$. Else, compute the re-encrypted ciphertext $\hat{\sigma_j}$ according to the following cases:
    - ∗ If $(c_i = 0 \wedge c_j = -)$ does not hold, check for the existence of a tuple $\langle PK_i, PK_j, RK^{\langle 1 \rangle}_{i \to j}, V, W, h, \tau \rangle$ in L_{RK}. If not found, compute the re-encryption keys by issuing a re-key generation query $\mathcal{O}_{ReKeyGen}(PK_i, PK_j)$ and obtain $\hat{\sigma_j} = ReEncrypt(\sigma_i, PK_i, PK_j, RK_{i \to}, PARAMS)$. Return $\hat{\sigma_j}$ to $\mathcal{A}$.
    - ∗ Otherwise if $(c_i = 0 \wedge c_j = -)$ holds, retrieve tuple $\langle PK_{i,1}, PK_{i,2}, D, E, F, \beta, \gamma \rangle$ from L_{H_5}. Compute $R = (\bar{E})^{\frac{1}{\beta}}$ and extract $(m || \omega) = F \oplus H_2(R)$. Search list L_{RK} for a tuple $\langle PK_i, PK_j, \perp, V, W, h, - \rangle$. If found, compute $\hat{E} = g^{r \cdot h}$ and return $\hat{\sigma_j} = (\hat{E}, F, V, W)$ to $\mathcal{A}$. Else, select $h \in_R \{0, 1\}^{l_0}$, $\pi \in_R \{0, 1\}^{l_1}$.

Compute $v = H_1(h, \pi)$, $V = (PK_{j,2})^v$, $W = H_2(g^v) \oplus (h||\pi)$ and $\hat{E} = g^{r \cdot h}$. Update the list L_{RK} with the tuple $\langle PK_i, PK_j, \perp, V, W, h, - \rangle$. Return $\hat{\sigma_j} = (\hat{E}, F, V, W)$ to $\mathcal{A}$.

- $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$: The challenger decrypts the original ciphertexts and transformed ciphertexts respectively as follows:
  - ∗ σ_i is an Original ciphertext: Check for the well-formedness of σ_i by computing E and $\bar{D}$ and verifying if equations (4) and (5) hold. If they do not hold, return $\perp$. Else, if $c_i = -$ or $c_i = 1$, run *Decrypt* algorithm to extract and return m. Else, if $c_i = 0$, retrieve tuple $\langle PK_{i,1}, PK_{i,2}, D, E, F, \beta, \gamma \rangle$ from L_{H_5}. Compute $R = (\bar{E})^{\frac{1}{\beta}}$ and extract $(m||\omega) = F \oplus H_2(R)$. Return m to $\mathcal{A}$.
  - ∗ σ_i is a transformed ciphertext: $\mathcal{C}$ decrypts according to the following two scenarios:
    - If there exists a tuple $\langle PK_j, PK_i, (RK^{\langle 1 \rangle}, V, W), h, 0 \rangle$ in L_{RK}, compute $E = \hat{E}^{\frac{1}{RK^{\langle 1 \rangle}}}$. Search for the existence of a tuple $\langle m, \omega, r \rangle$ in L_{H_1} and $\langle R, \theta \rangle$ in L_{H_2} such that the following conditions hold:

$$\begin{aligned}
&\left(PK_{j,1}^{H_4(PK_{j,2})} PK_{j,2}\right)^r \stackrel{?}{=} E, \\
&\theta \oplus (m||\omega) = F, \\
&R = g^r.
\end{aligned}$$

      If they hold, return m, else return $\perp$.
    - Else, search L_{H_1} for two tuples $\langle m, \omega, r \rangle$ and $\langle h, \pi, v \rangle$ and L_{H_2} for two tuples $\langle R, \theta \rangle$ and $\langle R', \theta' \rangle$ such that the following conditions hold:

$$\begin{aligned}
&g^{r \cdot h} = \hat{E}, \\
&\theta \oplus (m||\omega) = F, \\
&R = g^r, \\
&PK_{i,2}^v = V, \\
&\theta' \oplus (h||\pi) = W, \\
&R' = g^v.
\end{aligned}$$

  If all the checks are satisfied, return m. Otherwise, return $\perp$.

– **Challenge:** $\mathcal{A}$ outputs two messages $m_0, m_1 \in_R \{0,1\}^{l_0}$ and the target public key PK_i^*. $\mathcal{C}$ recovers tuple $\langle PK_i^*, x_{i,1}^*, x_{i,2}^*, c_i^* \rangle$ from list L_{key} and check if $c_i^* = 1$. If so, $\mathcal{C}$ aborts and reports failure. Else, if $c_i^* = 0$, $\mathcal{C}$ picks $\delta \in_R \{0,1\}$ and computes the challenge ciphertext σ_i^* in the following steps:
  1. Pick $\omega^* \in_R \{0,1\}^{l_1}$ and implicitly define $H_1(m_\delta, \omega^*) = b/a$.
  2. Pick $\bar{e}^*, f^* \in_R \mathbb{Z}_q^*$. Set $u \triangleq \bar{e}^* - \frac{b}{a} f^*$.
  3. Compute $D^* = (g^a)^{(x_{i,1}^* H_4(PK_{i,2}^*) + x_{i,2}^*)\bar{e}^*} (g^b)^{-(x_{i,1}^* H_4(PK_{i,2}^*) + x_{i,2}^*) f^*}$
     $= \left(g^{a(x_{i,1}^* H_4(PK_{i,2}^*) + x_{i,2}^*)}\right)^{\bar{e}^* - \frac{b}{a} f^*}$
     $= \left(g^{a(x_{i,1}^* H_4(PK_{i,2}^*) + x_{i,2}^*)}\right)^{u}$
     $= \left(PK_{i,1}^{H_4(PK_{i^*,2})} PK_{i^*,2}\right)^u$.
  4. Compute $E^* = (g^b)^{x_{i,1}^* H_4(PK_{i,2}^*) + x_{i,2}^*} = (g^{a x_{i,1}^* H_4(PK_{i,2}^*) + a x_{i,2}^*})^{\frac{b}{a}} = \left(PK_{i^*,1}^{H_4(PK_{i^*,2})} PK_{i^*,2}\right)^r$.
  5. Set $H_5(PK_{i^*,1}, PK_{i^*,2}, D^*, E^*, F^*) = \gamma = (g^a)^\beta$, where $\beta \in_R \mathbb{Z}_q^*$. Update list L_{H_5}.
  6. Compute $\bar{E}^* = (g^b)^\beta = (g^{a\beta})^{\frac{b}{a}} = H_5(PK_{i^*,1}, PK_{i^*,2}, D^*, E^*, F^*)^r$.
  7. Choose $F^* \in_R \{0,1\}^{l_0 + l_1}$ and define $H_3(E^*, \bar{E}^*, F^*) = f^*$.
  8. Implicitly define $H_2(g^{b/a}) = (m_\delta||\omega^*) \oplus F^*$.
  9. Set $s^* = \bar{e}$.
  10. Output the challenge ciphertext $\sigma_i^* = (E^*, \bar{E}^*, F^*, s^*)$ to $\mathcal{A}$.

– **Phase 2:** $\mathcal{A}$ issues queries to $\mathcal{C}$ as shown in Phase 1, with the restrictions described for IND-PRE-CCA game.

– **Guess:** $\mathcal{A}$ returns its guess $\delta' \in_R \{0,1\}$ to $\mathcal{C}$. $\mathcal{C}$ randomly picks a tuple $\langle R, \theta \rangle$ from L_{H_2} list and outputs R as the solution to the DCDH instance.

– **Probability Analysis:** We first analyse the simulation of the random oracles. The simulations of the hash function H_4 is perfect. Also, the simulations of H_1, H_2, H_3 and H_5 are perfect unless the following event occurs:

- $E_{H_1^*}$: (m_δ, ω^*) has been queried to H_1.
- $E_{H_2^*}$: $(g^{b/a})$ has been queried to H_2.
- $E_{H_3^*}$: $(D^*, \bar{E}^*, F^*)$ has not been queried to H_3 before the *Challenge* phase.
- $E_{H_5^*}$: $(PK_{i^*,1}, PK_{i^*,2}, D^*, E^*, F^*)$ has not been queried to H_5 before the *Challenge* phase.

Note that, in the *Challenge* phase, F^* is chosen uniformly at random from $\{0,1\}^{l_0+l_1}$ by the Challenger $\mathcal{C}$, and hence $Pr[E_{H_3}] \leq \frac{q_{H_3}}{2^{l_0+l_1}}$. Also, $Pr[E_{H_5}] \leq \frac{q_{H_5}}{2^{l_0+l_1}}$.
Next we analyse the probability with which $\mathcal{C}$ aborts during the simulation. Let *Abort* denote the probability that $\mathcal{C}$ aborts during the $re-encryption\ key\ generation$ or *Challenge* phase. In both these phases, C does not abort in the following events:
- E_1: $c_i = 1$ in the *re-encryption key generation* query.
- E_2: $c_i^* = 0$ in the *Challenge* phase.

We have $Pr[\neg Abort] \geq \rho^{q_{RK}}(1-\rho)$, which has a maximum value at $\rho_{OPT} = \frac{q_{RK}}{1+q_{RK}}$. Using ρ_{OPT}, we obtain:

$$Pr[\neg Abort] \geq \frac{1}{e(1+q_{RK})}.$$

The simulation of the decryption oracle is perfect unless valid ciphertexts are rejected by the oracle. This error occurs since a valid ciphertext can be produced without querying H_1 and H_2 i.e., without querying g^r to H_2 where $r = H_1(m,\omega)$. Let E_{val} denote the event that the ciphertext is a valid ciphertext. We use E_{H_1} and E_{H_2} to denote the events that (m,ω) is queried to H_1 and g^r is queried to H_2. We analyse $Pr[E_{val}|\neg E_{H_1} \vee \neg E_{H_2}] \leq Pr[E_{val}|\neg E_{H_1}] + Pr[E_{val}|\neg E_{H_2}]$:

$$\begin{aligned} Pr[E_{val}|\neg E_{H_1}] &= Pr[E_{val} \wedge E_{H_2}|\neg E_{H_1}] + Pr[E_{val} \wedge \neg E_{H_2}|\neg E_{H_1}] \\ &\leq Pr[E_{val} \wedge E_{H_2}|\neg E_{H_1}] + Pr[E_{val}|\neg E_{H_2} \wedge \neg E_{H_1}] \\ &= \frac{E_{val} \wedge E_{H_2} \wedge \neg E_{H_1}}{\neg E_{H_1}} + \frac{1}{q} \\ &\leq \frac{q_{H_2}}{2^{l_0+l_1}} + \frac{1}{q} \end{aligned}$$

Similarly, we obtain $Pr[E_{val}|\neg E_{H_1}] \leq \frac{q_{H_1}}{2^{l_0+l_1}} + \frac{1}{q}$, and $Pr[E_{val}|(\neg E_{H_1} + \neg E_{H_2})] \leq \frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}$

Let E_{der} denote the event that $E_{val}|(\neg E_{H_1} + \neg E_{H_2})$ takes place during the entire simulation. We obtain:

$$Pr[E_{der}] \leq q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big).$$

Let E_{err} denote the event $(E_{H_1^*} \vee E_{H_2^*} \vee E_{H_3^*} \vee E_{H_5^*} \vee E_{der})|\neg Abort$. If E_{err} does not occur, the adversary $\mathcal{A}$ does not gains no advantage greater than $\frac{1}{2}$ in guessing δ due to the randomness in the output of H_2, i.e., $Pr[\delta' = \delta|\neg E_{err}] = \frac{1}{2}$. The probability that the adversary correctly guesses δ is:

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

Again,

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

From the definition of the advantage of IND-PRE-CCA adversary, we know:

$$\begin{aligned} \epsilon &= |2Pr[\delta' = \delta] - 1| \\ &\leq Pr[E_{err}] = Pr[(E_{H_1^*} \vee E_{H_2^*} \vee E_{H_3^*} \vee E_{der})|\neg Abort] \\ &\leq \frac{Pr[E_{H_1^*}] + Pr[E_{H_2^*}] + Pr[E_{H_3^*}] + Pr[E_{H_5^*}] + Pr[E_{der}]}{Pr[\neg Abort].} \end{aligned}$$

Note that $Pr[E_{H_1^*}] \leq \frac{q_{H_1}}{2^{l_1}}$ as $\mathcal{C}$ picks $\omega \in_R \{0,1\}^{l_1}$, and we obtain the following bound on $Pr[E_{H_2^*}]$:

$$\begin{aligned} Pr[E_{H_2^*}] &\geq Pr[\neg Abort] \cdot \epsilon - Pr[E_{H_1^*}] - Pr[E_{H_3^*}] - Pr[E_{H_5^*}] - Pr[E_{der}] \\ &\geq \frac{\epsilon}{e(q_{RK}+1)} - \frac{q_{H_1}}{2^{l_1}} - \frac{q_{H_3}}{2^{l_0+l_1}} - \frac{q_{H_5}}{2^{l_0+l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big) \end{aligned}$$

We observe that, if event $E_{H_2^*}$ occurs, then the challenger $\mathcal{C}$ solves the $DCDH$ instance with advantage:

$$\epsilon' \geq \frac{1}{q_{H_2}} Pr[E_{H_2^*}] \geq \frac{1}{q_{H_2}}\Big(\frac{\epsilon}{e(q_{RK}+1)} - \frac{q_{H_1}}{2^{l_1}} - \frac{q_{H_3}+q_{H_5}}{2^{l_0+l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big)\Big)$$

We also bound the running time of $\mathcal{C}$ to solve DCDH instance by:

$$\begin{aligned} t' &\leq t + (q_{H_1}+q_{H_2}+q_{H_3}+q_{H_4}+q_{H_5}+n_h+n_c+q_{RK}+q_{RE}+q_d)O(1) \\ &\quad + (2n_h+2n_c+2q_{RK}+5q_{RE}+2q_d+q_{H_1}q_{RE}+(2q_{H_2}+2q_{H_1})q_d)t_e. \end{aligned}$$

This completes the proof of the theorem. □

Transformed Ciphertext Security:

Theorem 2. *The proposed scheme is CCA-secure for the transformed ciphertext under the DCDH assumption and the $EUF-CMA$ security of Schnorr signature scheme [19]. If a (t,ϵ)IND-PRE-CCA $\mathcal{A}$ with an advantage ϵ breaks the IND-CPRE-CCA security of the given scheme, $\mathcal{C}$ can solve the DCDH problem with advantage ϵ' within time t' where:*

$$\begin{aligned} \epsilon' &\geq \frac{1}{q_{H_2}}\Big(\frac{2\epsilon}{e(2+q_{RK})^2} - \frac{q_{H_1}}{2^{l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big) - \epsilon_2\Big), \\ t' &\leq t + (q_{H_1}+q_{H_2}+q_{H_3}+q_{H_4}+q_{H_5}+n_h+n_c+q_{RK}+q_{RE}+q_d)O(1) \\ &\quad + (2n_h+2n_c+2q_{RK}+3q_{RE}+2q_d+(2q_{H_2}+2q_{H_1})q_d)t_e, \end{aligned}$$

We note that e is the base of natural logarithm, ϵ_2 denotes the advantage in breaking the EUF-CMA security of the Schnorr Signature scheme and t_e denotes the time taken for exponentiation in group $\mathbb{G}$.

Proof. Given an adversary $\mathcal{A}$ that breaks the (t,ϵ)*IND-PRE-CCA* security of the scheme, we show how to construct a polynomial time algorithm $\mathcal{C}$ which breaks the $DCDH$ assumption in $\mathcal{G}$ or the existential unforgeability against chosen message attack (*EUF-CMA*) of the Schnorr Signature with a non-negligible advantage. As described in the *Original Ciphertext Security* game, $\mathcal{C}$ maintains a list L_{key} to store the lists of the public/private key pairs of the users. $\mathcal{C}$ also maintains list L_{RK} with tuples of the form $\langle PK_i, PK_j, RK^{\langle 1\rangle}_{i\to j}, V, W, h, z\rangle$ to store the lists of re-encryption keys of the users. Both the lists are initially empty.

- **Key Generation:** $\mathcal{C}$ maintains a list L_{key} that contains tuples of the form $\langle PK_i, x_{i,1}, x_{i,2}, c_i\rangle$ which includes information of all the user keys. $\mathcal{C}$ generates the keys of the users in the following ways:
  - Uncorrupted User Keys: $\mathcal{C}$ uses Coron's coin tossing technique [9] to generate the uncorrupted user keys by flipping a biased coin $c_i \in \{0,1\}$ that yields value 1 with probability ρ, which we shall determine later. $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes PK_i according to the following cases:
    - If $c_i = 1$, compute $PK_i = (PK_{i,1}, PK_{i,2}) = ((g^a)^{1/H_4(PK_{i,2})} \cdot g^{x_{i,1}}, g^{x_{i,2}}/g^a)$. Add the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 1\rangle$ to L_{key}.
    - If $c_i = 0$, compute $PK_i = (PK_{i,1}, PK_{i,2}) = ((g^a)^{x_{i,1}}, (g^a)^{x_{i,2}})$. Add the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 0\rangle$ to L_{key}.
  - Corrupted User Keys: $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes $PK_i = (PK_{i,1}, PK_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$. It adds the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = -\rangle$ to L_{key}.

- **Phase 1:** $\mathcal{C}$ answers the queries issues by $\mathcal{A}$ as follows:
  - Oracle Queries: $\mathcal{C}$ responds to the hash function queries of $\mathcal{A}$ in the same way as it responds in the *Original Ciphertext Security.*

  - $\mathcal{O}_{ReKeyGen}(PK_i, PK_j)$: $\mathcal{C}$ maintains a list L_{RK} with entries of the form $\langle PK_i, PK_j, RK^{\langle 1\rangle}_{i\to j}, V, W, h, \tau\rangle$. If the re-encryption keys from PK_i to PK_j already exists in L_{RK}, retrieve and return $RK_{i\to j} = (RK^{\langle 1\rangle}_{i\to j}, V, W)$. Else, $\mathcal{C}$ computes the re-encryption keys as shown:
    - If $c_i = 0 \wedge c_j \in \{1, '-'\}$: abort and report failure.
    - If $c_i = 1 \vee c_i = '-'$:
      - Define $RK^{\langle 1\rangle}_{i\to j} = \frac{h}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}$.

- $\cdot$ Select $h \in_R \{0,1\}^{l_0},\ \pi \in_R \{0,1\}^{l_1}$.
- $\cdot$ Compute $v = H_1(h, \pi)$, $V = (PK_{j,2})^v$, $W = H_2(g^v) \oplus (h||\pi)$.
- $\cdot$ Set $\tau = 1,\ z = 0$.

We note that the computation of $RK^{\langle 1\rangle}_{i\to j}$ is correct for both the cases, as we have:
- $\cdot$ For $c_i =' -'$, the correctness is trivial from the definition of the private key $SK_i = (x_{i,1}, x_{i,2})$.
- $\cdot$ For $c_i = 1$, $RK^{\langle 1\rangle}_{i\to j} = \frac{h}{\left(\frac{a}{H_4(PK_{i,2})}+x_{i,1}\right)H_4(PK_{i,2})-a+x_{i,2}} = \frac{h}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}$.

$*$ If $c_i = 0 \wedge c_j = 0$:
- $\cdot$ Pick $RK^{\langle 1\rangle}_{i\to j} \in_R \mathbb{Z}_q^*$ and set $\tau = 0$. Since a random value of $RK_{i\to j}$ would not match the value h associated with (V, W), we rely on the security of hashed Elgamal encryption scheme [3,11,12] as shown in the security proof of original paper [7].
- $\cdot$ Select $z \in_R \mathbb{Z}_q^*$ and set $V = \left(g^b\right)^z$. Observe that $bz = (ax_{j,2})v$ i.e., $g^v = \left(g^{b/a}\right)^{ax_{j,2}}$ follows from the definition in Scheme description.
- $\cdot$ Select $h \in_R \{0,1\}^{l_0},\ \pi \in_R \{0,1\}^{l_1}$.
- $\cdot$ Pick $W \in_R \{0,1\}^{l_0+l_1}$ and implicitly define $H_2(\left(g^{b/a}\right)^{ax_{j,2}}) = (h||\pi) \oplus W$.
- $\cdot$ Store tuple $\langle PK_i, PK_j, RK^{\langle 1\rangle}_{i\to j}, V, W, h, 0, z\rangle$ in the list L_{RK}.

$*$ Return the re-encryption key $RK_{i\to j} = (RK^{\langle 1\rangle}_{i\to j}, V, W)$ to $\mathcal{A}$.

- $\mathcal{O}_{ReEncrypt}(\sigma_i, PK_i, PK_j)$: The re-encryption oracle remains the same as demonstrated in the *Original Ciphertext Security* game.

- $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$: The decryption oracle is simulated in the exact manner as demonstrated in the *Original Ciphertext Security* game.

– **Challenge:** $\mathcal{A}$ outputs two messages $m_0, m_1 \in_R \{0,1\}^{l_0}$, the delegator's public key $PK_{i'}$ and the target(delegatee) public key PK_i^*. $\mathcal{C}$ recovers tuples $\langle PK_{i'}, x_{i',1}, x_{i',2}, c_i'\rangle$ and $\langle PK_i^*, x_{i,1}^*, x_{i,2}^*, c_i^*\rangle$ from list L_{key}. If $c_i' = 1$ or $c_i^* = 1$, $\mathcal{C}$ aborts and reports failure. Else, $\mathcal{C}$ picks $\delta \in_R \{0,1\}$ and computes the challenge ciphertext σ_i^* as shown below. Note that, $c_i' = - \ \wedge \ c_i^* = 0$ is a special case of the simulation below as $\mathcal{A}$ cannot query for the re-key $RK_{i'\to i^*}$.
1. Pick $t \in_R \mathbb{Z}_q^*$ and $\omega^* \in \{0,1\}^{l_1}$. Define $\hat{E}^* = (g^b)^t$, which implies $r^*h^* = bt$, i.e., $r^* = H_1(m_\delta, \omega^*) = bt/h^*$.
2. Observe that $RK_{i'\to i^*} = \frac{h^*}{a(x_{i',1}H_4(PK_{i',2})+x_{i',2})}$. Therefore, $h^* = RK_{i'\to i^*}\left(a(x_{i',1}H_4(PK_{i',2}) + x_{i',2})\right)$, which implicitly defines $r^* = (b/a)(t/(RK_{i'\to i^*}(x_{i',1}H_4(PK_{i',2}) + x_{i',2}))$.
3. Pick $F^* \in_R \{0,1\}^{l_0+l_1}$ and implicitly define $H_2(g^{(b/a)^{(t/(RK_{i'\to i^*}(x_{i',1}H_4(PK_{i',2})+x_{i',2}))}}) = F^* \oplus (m_\delta \oplus \omega^*)$.
4. Retrieve the tuple $\langle PK_{i'}, PK_{i^*}, RK^{\langle 1\rangle}_{i'\to i^*}, V^*, W^*, \perp, 0, z^*\rangle$ from L_{RK}. If such a tuple does not exist, define V^*, W^*, h^*, z^* in the following way:
   - Select $z^* \in_R \mathbb{Z}_q^*$ and set $V^* = \left(g^b\right)^{z^*}$. Observe that $bz^* = (ax_{i^*,2})v$ i.e., $g^v = \left(g^{b/a}\right)^{ax_{i^*,2}}$.
   - Select $h^* \in_R \{0,1\}^{l_0},\ \pi^* \in_R \{0,1\}^{l_1}$.
   - Pick $W^* \in_R \{0,1\}^{l_0+l_1}$ and implicitly define $H_2(\left(g^{b/a}\right)^{ax_{i^*,2}}) = (h^*||\pi^*) \oplus W^*$.
   - Store tuple $\langle PK_{i'}, PK_{i^*}, RK^{\langle 1\rangle}_{i'\to i^*}, V^*, W^*, h^*, 0, z^*\rangle$ in the list L_{RK}.
5. Output the challenge ciphertext $\hat{\sigma_i^*} = (\hat{E}^*, F^*, V^*, W^*)$ to $\mathcal{A}$.

– **Phase 2:** $\mathcal{A}$ issues queries to $\mathcal{C}$ as shown in Phase 1, with the restrictions described for IND-PRE-CCA game.

– **Guess:** $\mathcal{A}$ returns its guess $\delta' \in_R \{0,1\}$ to $\mathcal{C}$, who first retrieves tuple $\langle m_{\delta'}, \omega, r\rangle$ from L_{H_1} and checks if $(g^a)^{r\cdot RK_{i'\to i^*}(x_{i',i^*}H_4(PK_{i',2})+x_{i',2})/t} = g^b$. If no such entry exists, $\mathcal{C}$ randomly picks a tuple $\langle R, \theta\rangle$ from L_{H_2} list and outputs R as the solution to the DCDH instance.

– **Probability Analysis:** We first analyse the simulation of the random oracles. The simulations of the hash functions H_3, H_4 and H_5 are perfect. Also, the simulations of H_1 and H_2 are perfect unless the following event occurs:
  - $E_{H_1^*}$: (h^*, π^*) has been queried to H_1.
  - $E_{H_2^*}$: $(g^{b/a})$ or $(g^{b/a})^{t/RK_{i'\to i^*}(x_{i',1}H_4(PK_{i',2}+x_{i',2}))} \oplus (m_\delta||\omega^*)$ has been queried to H_2.

  Next we analyse the probability with which $\mathcal{C}$ aborts during the simulation. Let *Abort* denote the probability that $\mathcal{C}$ aborts during the *re − encryption key generation* or *Challenge* phase. In both these phases, C does not abort in the following events:
  - E_1: $c_i = 1$ in the *re − encryption key generation* query.

– E_2: $c_i^* = 0 \wedge c_i' \neq 1$ in the *Challenge* phase.

We have $Pr[\neg Abort] \geq \rho^{q_{RK}}(1-\rho)^2$, which has a maximum value at $\rho_{OPT} = \frac{q_{RK}}{2+q_{RK}}$. Using ρ_{OPT}, we obtain:

$$Pr[\neg Abort] \geq \frac{2}{e(2+q_{RK})^2}.$$

The analysis of the simulation of the decryption oracle is remains the same as shown for the original ciphertext security. The probability of the decryption oracle rejecting valid ciphertexts throughout the entire simulation denoted by Let E_{der} is:

$$Pr[E_{der}] \leq q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big).$$

Again, let E_{err} denote the event $(E_{H_1^*} \vee E_{H_2^*} \vee E_{der})|\neg Abort$. Following a similar analysis as shown in Section 5.2 and from the definition of the advantage of IND-PRE-CCA adversary, we obtain the following bound on $Pr[E_{H_2^*}]$:

$$\begin{aligned} Pr[E_{H_2^*}] &\geq Pr[\neg Abort] \cdot \epsilon - Pr[E_{H_1^*}] - Pr[E_{der}] \\ &\geq \frac{2\epsilon}{e(2+q_{RK})^2} - \frac{q_{H_1}}{2^{l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big) \end{aligned}$$

We observe that, if event $E_{H_2^*}$ occurs, then the challenger $\mathcal{C}$ solves the $DCDH$ instance with advantage:

$$\epsilon' \geq \frac{1}{q_{H_2}} Pr[E_{H_2^*}] \geq \frac{1}{q_{H_2}}\bigg(\frac{2\epsilon}{e(2+q_{RK})^2} - \frac{q_{H_1}}{2^{l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big)\bigg)$$

We also bound the running time of $\mathcal{C}$ to solve DCDH instance by:

$$\begin{aligned} t' &\leq t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + n_h + n_c + q_{RK} + q_{RE} + q_d)O(1) \\ &\quad + (2n_h + 2n_c + 2q_{RK} + 3q_{RE} + 2q_d + (2q_{H_2} + 2q_{H_1})q_d)t_e. \end{aligned}$$

This completes the proof of the theorem. □

Non-transformable Ciphertext Security:

Theorem 3. *The proposed scheme is CCA-secure for the non-transformable ciphertext under the CDH assumption. If a $(t, \epsilon - \epsilon_2)$IND-PRE-CCA $\mathcal{A}$ with an advantage $\epsilon - \epsilon_2$ breaks the IND-PRE-CCA security of the given scheme, $\mathcal{C}$ can solve the CDH problem with advantage ϵ' within time t' where:*

$$\begin{aligned} \epsilon' &\geq \frac{1}{q_{H_2}}\bigg(\epsilon - \epsilon_2 - \frac{q_{H_1}}{2^{l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big)\bigg), \\ t' &\leq t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + n_h + n_c + q_{RK} + q_d)O(1) \\ &\quad + (2n_h + 2n_c + 2q_{RK} + 2q_d + (2q_{H_2} + 2q_{H_1})q_d)t_e, \end{aligned}$$

We note that ϵ_2 is the advantage of an attacker against $EUF-CMA$ security game of the Schnorr signature scheme. e is the base of natural logarithm and t_e denotes the time taken for exponentiation in group $\mathbb{G}$.

Proof. Let us assume that the Schnorr Signature is $(t', \epsilon_2) - EUF - CMA$ where the probability $\epsilon_2 < \epsilon$. Given an adversary $\mathcal{A}$ that breaks the $(t, (\epsilon - \epsilon_2))$*IND-PRE-CCA* of the scheme, we show how to construct a polynomial time algorithm $\mathcal{C}$ which breaks the CDH assumption in $\mathcal{G}$ with a non-negligible advantage. $\mathcal{C}$ maintains two lists L_{key} in the same manner as described in *Original Ciphertext Security* game and L_{RK} with tuples of the form $\langle PK_i, PK_j, RK_{i\to j}^{\langle 1\rangle}, V, W, h\rangle$ to store the lists of the public/private key pairs and the re-encryption keys of the users respectively. Both the lists are initially empty.

– **Key Generation:** $\mathcal{C}$ maintains a list L_{key} that contains tuples of the form $\langle PK_i, x_{i,1}, x_{i,2}, c_i\rangle$ which includes information of all the user keys. $\mathcal{C}$ generates the keys of the users in the following ways:
  • Uncorrupted User Keys: $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes $PK_i = (PK_{i,1}, PK_{i,2}) = ((g^a)^{1/H_4(PK_{i,2})} \cdot g^{x_{i,1}}, g^{x_{i,2}}/g^a)$. Add the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 0\rangle$ to L_{key}.
  • Corrupted User Keys: $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes $PK_i = (PK_{i,1}, PK_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$. It adds the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 1\rangle$ to L_{key}.

– **Phase 1:** $\mathcal{C}$ answers the queries issues by $\mathcal{A}$ as follows:
  - Oracle Queries: $\mathcal{C}$ responds to the hash function queries of $\mathcal{A}$ in the same way as it responds in the *Original Ciphertext Security.*

  - $\mathcal{O}_{ReKeyGen}(PK_i, PK_j)$: $\mathcal{C}$ maintains a list L_{RK} with entries of the form $\langle PK_i, PK_j, RK^{\langle 1\rangle}_{i\to j}, V, W, h\rangle$. If the re-encryption keys from PK_i to PK_j already exists in L_{RK}, retrieve and return $RK_{i\to j} = (RK^{\langle 1\rangle}_{i\to j}, V, W)$. Else, $\mathcal{C}$ computes the re-encryption keys as shown:
    * $c_i = 1$: Return the re-encryption key obtained by calling the re-encryption algorithm *ReKeyGen* $(SK_i, PK_i, PK_j, PARAMS)$ to $\mathcal{A}$.
    * $c_i = 0$: Compute the re-encryption key as shown below:
      · Pick $h \in_R \{0,1\}^{l_0}$, $\pi \in_R \{0,1\}^{l_1}$.
      · Compute $RK^{\langle 1\rangle}_{i\to j} = \frac{h}{(\frac{a}{H_4(PK_{j,2})}+x_{i,1})H_4(PK_{i,2})-a+x_{i,2}} = \frac{h}{x_{i,1}H_4(PK_{i,2})+x_{i,2}}$. Note that the knowledge of a is not needed to compute $RK^{\langle 1\rangle}_{i\to j}$.
      · Compute $v = H_1(h,\pi)$, $V = g^v$.
      · Compute $W = H_2(PK^v_{j,2}) \oplus (h||\pi)$.
      · Return $RK_{i\to j} = (RK^{\langle 1\rangle}_{i\to j}, V, W)$.
  - $\mathcal{O}_{Decrypt}(\sigma_i, PK_i)$: The decryption oracle is simulated in the exact manner as demonstrated in the *Original Ciphertext Security* game.

– **Challenge:** $\mathcal{A}$ outputs two messages $m_0, m_1 \in_R \{0,1\}^{l_0}$ and the target public key PK^*_i. $\mathcal{C}$ recovers tuple $\langle PK^*_i, x^*_{i,1}, x^*_{i,2}, c^*_i\rangle$ from list L_{key}, picks $\delta \in_R \{0,1\}$ and computes the challenge ciphertext $\hat{\sigma}^*_i$ in the following steps:
  1. Pick $\omega^* \in_R \{0,1\}^{l_1}$ and issues a H_1 query $H_1(m_\sigma, \omega^*)$ to obtain r^*.
  2. Pick $h^* \in_R \{0,1\}^{l_0}$, $\pi \in_R \{0,1\}^{l_1}$. Implicitly define $H_1(h^*, \pi^*) = b$ (b is not known to $\mathcal{C}$).
  3. Choose $W^* \in_R \{0,1\}^{l_0+l_1}$ and implicitly define $H_2(g^{-ab}g^{bx_{i^*,2}}) = (h^*||\pi^*)\oplus W^*$. Note that $W^* = H_2(g^{-ab}g^{bx_{i^*,2}})\oplus (h^*||\pi^*) = H_2(g^v) \oplus (h^*||\pi^*)$.
  4. Compute $V^* = g^b = PK^v_{i^*,2}$.
  5. Compute $\hat{E}^* = g^{r^*h^*}$, $F^* = H_2(g^{r^*}) \oplus (m_\sigma||\omega^*)$.
  6. Return the ciphertext $\hat{\sigma_{i^*}} = (\hat{E}^*, F^*, V^*, W^*)$.

– **Phase 2:** $\mathcal{A}$ issues queries to $\mathcal{C}$ as shown in Phase 1, with the restrictions described for IND-PRE-CCA game.

– **Guess:** $\mathcal{A}$ returns its guess $\delta' \in_R \{0,1\}$ to $\mathcal{C}$. $\mathcal{C}$ randomly picks a tuple $\langle R, \theta\rangle$ from L_{H_2} list and outputs $\Big(\frac{R}{g^{bx_{i^*,2}}}\Big)^{-1}$ as the solution to the CDH instance.

– **Probability Analysis:** We first analyse the simulation of the random oracles. The simulations of the hash functions H_3, H_4 and H_5 are perfect. Also, the simulations of H_1 and H_2 are perfect unless the following event occurs:
  – $E_{H^*_1}$: (h^*, π^*) has been queried to H_1.
  – $E_{H^*_2}$: $(g^{b/a})$ or $(g^{b/a})^{t/RK_{i'\to i^*}(x_{i',1}H_4(PK_{i',2}+x_{i',2}))} \oplus (m_\delta||\omega^*)$ has been queried to H_2.

  The analysis of the simulation of the decryption oracle is remains the same as shown for the original ciphertext security. The probability of the decryption oracle rejecting valid ciphertexts throughout the entire simulation denoted by Let E_{der} is:

$$Pr[E_{der}] \le q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big).$$

  Again, let E_{err} denote the event that $E_{H^*_1}$, $E_{H^*_2}$ or E_{der} occurs i.e., $E_{err} = (E_{H^*_1} \vee E_{H^*_2} \vee E_{der})$. From the analysis as shown in the Original Ciphertext security game and from the definition of the advantage of IND-PRE-CCA adversary, we obtain:

$$\begin{aligned}\epsilon - \epsilon_2 &= |2Pr[\delta' = \delta] - 1| \\ &\le Pr[E_{err}] = Pr[(E_{H^*_1} \vee E_{H^*_2} \vee E_{der})] \\ &\le Pr[E_{H^*_1}] + Pr[E_{H^*_2}] + Pr[E_{der}]\end{aligned}$$

  We obtain the following bound on $Pr[E_{H^*_2}]$:

$$\begin{aligned}Pr[E_{H^*_2}] &\ge \epsilon - \epsilon_2 - Pr[E_{H^*_1}] - Pr[E_{der}] \\ &\ge \epsilon - \epsilon_2 - \frac{q_{H_1}}{2^{l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big)\end{aligned}$$

We observe that, if event $E_{H_2^*}$ occurs, then the challenger $\mathcal{C}$ solves the $DCDH$ instance with advantage:

$$\begin{aligned}\epsilon' &\geq \frac{1}{q_{H_2}} Pr[E_{H_2^*}] \\ &\geq \frac{1}{q_{H_2}}\left(\epsilon - \epsilon_2 - \frac{q_{H_1}}{2^{l_1}} - q_d\Big(\frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}\Big)\right)\end{aligned}$$

We also bound the running time of $\mathcal{C}$ to solve DCDH instance by:

$$\begin{aligned}t' \leq\ & t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + n_h + n_c + q_{RK} + q_{RE} + q_d)O(1) \\ &+ (2n_h + 2n_c + 2q_{RK} + 2q_d + (2q_{H_2} + 2q_{H_1})q_d)t_e.\end{aligned}$$

This completes the proof of the theorem. □

Delegator Secret Security:

Theorem 4. *The proposed scheme is DSK-secure under the DL assumption. If a $(t,\epsilon)DSK$ $\mathcal{A}$ with an advantage ϵ breaks the DSK security of the given scheme in time t, $\mathcal{C}$ can solve the DL problem with advantage ϵ within time t' where:*

$$t' \leq t + O(2q_{RK} + 2n_h + 2n_c)t_e,$$

We note that t_e denotes the time taken for exponentiation in group $\mathbb{G}$.

Proof. Given an adversary $\mathcal{A}$ that breaks the $(t,\epsilon)DSK$ security of the scheme, we show how to construct a polynomial time algorithm $\mathcal{C}$ which breaks the DL assumption in $\mathcal{G}$ with a non-negligible advantage. Note that we do not model the hash functions $H_1,\ H_2,\ H_3,\ H_4$ and H_5 as random oracles in this proof. $\mathcal{C}$ maintains a list L_{key} with tuples of the form $\langle PK_i, x_{i,1}, x_{i,2}, c_i\rangle$ to store the lists of the public/private key pairs of the users, with the list being initially empty. $\mathcal{C}$ plays the DSK game with $\mathcal{A}$ in the following way:

– **Setup**: $\mathcal{C}$ runs $Setup(\lambda)$ and gives the resulting system parameters $PARAMS = (q, \mathbb{G}, g, H_1, H_2, H_3, H_4, H_5, l_0, l_1)$ to $\mathcal{A}$.

– **Queries**: $\mathcal{C}$ responds to the queries of $\mathcal{A}$ in the following way:
  - Uncorrupted-Key Generation (U_i): $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes $PK_i = (PK_{i,1}, PK_{i,2}) = ((g^a)^{1/H_4(PK_{i,2})} \cdot g^{x_{i,1}}, g^{x_{i,2}}/g^a)$. Add the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 0\rangle$ to L_{key}. $\mathcal{C}$ returns PK_i to $\mathcal{A}$.
  - Corrupted-Key Generation (U_i): $\mathcal{C}$ chooses $x_{i,1}, x_{i,2} \in_R \mathbb{Z}_q^*$ and computes $PK_i = (PK_{i,1}, PK_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$. It adds the tuple $\langle PK_i, x_{i,1}, x_{i,2}, c_i = 1\rangle$ to L_{key}.
  - Re-encryption Key Generation (PK_i, PK_j): $\mathcal{C}$ computes the re-encryption keys as shown:
    * $c_i = 1$: Return the re-encryption key obtained by calling the re-encryption algorithm $ReKeyGen$ $(SK_i, PK_i, PK_j, PARAMS)$ to $\mathcal{A}$.
    * $c_i = 0$: Compute the re-encryption key as shown below:
      · Pick $h \in_R \{0,1\}^{l_0},\ \pi \in_R \{0,1\}^{l_1}$.
      · Compute $RK^{\langle 1\rangle}_{i\to j} = \frac{h}{(\frac{a}{H_4(PK_{j,2})} + x_{i,1})H_4(PK_{i,2}) - a + x_{i,2}} = \frac{h}{x_{i,1}H_4(PK_{i,2}) + x_{i,2}}$. Note that the knowledge of a is not needed to compute $RK^{\langle 1\rangle}_{i\to j}$.
      · Compute $v = H_1(h, \pi),\ V = g^v$.
      · Compute $W = H_2(PK^v_{j,2}) \oplus (h||\pi)$.
      · Return $RK_{i\to j} = (RK^{\langle 1\rangle}_{i\to j}, V, W)$.

– **Output**: Eventually, $\mathcal{A}$ returns SK_i^* as the private key corresponding to the public key PK_i^*, where PK_i^* is uncorrupt $(c_i^* = 0)$. $\mathcal{C}$ recovers the tuple $\langle PK_i^*, x_{i^*,1}, x_{i^*,2}, c_i^* = 0\rangle$ from list L_{key} and returns $x_{i^*,2} - SK_{i^*,2}$ as a solution to the DL problem. Note that for a valid private key $SK_i^* = (SK_{i^*,1}, SK_{i^*,2})$ corresponding to the public key PK_i^*, $SK_{i^*,1} = \frac{a}{H_4(PK_{i^*,2}) + x_{i^*,1}}$ and $SK_{i^*,2} = -a + x_{i^*,2}$.

– **Probability Analysis**: We note that, if a DSK adversary returns a valid private key SK_i^* with advantage ϵ and breaks the $(t,\epsilon)DSK$ security of the scheme, $\mathcal{C}$ breaks the DL assumption with the same advantage ϵ. This is because $\mathcal{C}$ provides valid responses to all the queries issues by $\mathcal{A}$ and the simulation is perfect. We bound the running time of $\mathcal{C}$ as:

$$t^* \leq t + O(2q_{RK} + 2n_h + 2n_c)t_e$$

This completes the proof of the theorem. □

6 Conclusion

Although pairing is an expensive operation, only a few pairing-free unidirectional PRE schemes have been proposed in the literature, of which only one scheme due to Chow *et al.* [8] reported the collusion-resistance property. However, in this paper, we point out that the security proof in the scheme is flawed. We have shown that the adversary will be able to determine that the simulation provided by the challenger is not consistent with the real system. This makes the proof incomplete and the scheme is not provable secure. Additionally, we remark that the flaw cannot be corrected. Also, we present the first construction of a unidirectional proxy re-encryption scheme without bilinear pairing that provides collusion-resistance. Our scheme is proven CCA-secure under a variant of the computational Diffie-Hellman assumption in the random oracle model.

References

1. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2005, San Diego, California, USA*, 2005.
2. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.
3. Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Certificateless public key encryption without pairing. In *Information Security, 8th International Conference, ISC 2005, Singapore, September 20-23, 2005, Proceedings*, pages 134–148, 2005.
4. Paulo S. L. M. Barreto, Hae Yong Kim, Ben Lynn, and Michael Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 354–368, 2002.
5. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology - EUROCRYPT '98, International Conference on the Theory and Application of Cryptographic Techniques, Espoo, Finland, May 31 - June 4, 1998, Proceeding*, pages 127–144, 1998.
6. Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 185–194, 2007.
7. Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient unidirectional proxy re-encryption. *IACR Cryptology ePrint Archive*, 2009:189, 2009.
8. 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.
9. Jean-Sébastien Coron. On the exact security of full domain hash. In *Annual International Cryptology Conference*, pages 229–235. Springer, 2000.
10. Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen. Chosen-ciphertext secure proxy re-encryption without pairings. In *Cryptology and Network Security, 7th International Conference, CANS 2008, Hong-Kong, China, December 2-4, 2008. Proceedings*, pages 1–17, 2008.
11. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 537–554, 1999.
12. Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in Cryptology, Proceedings of CRYPTO '84, Santa Barbara, California, USA, August 19-22, 1984, Proceedings*, pages 10–18, 1984.
13. Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In *Applied Cryptography and Network Security, 5th International Conference, ACNS 2007, Zhuhai, China, June 5-8, 2007, Proceedings*, pages 288–306, 2007.
14. Thomas S. Heydt-Benjamin, Hee-Jin Chae, Benessa Defend, and Kevin Fu. Privacy for public transportation. In *Privacy Enhancing Technologies, 6th International Workshop, PET 2006, Cambridge, UK, June 28-30, 2006, Revised Selected Papers*, pages 1–19, 2006.
15. Anca-Andreea Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2003, San Diego, California, USA*, 2003.
16. Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *International Workshop on Public Key Cryptography*, pages 360–379. Springer, 2008.
17. Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. *IEEE Trans. Information Theory*, 57(3):1786–1802, 2011.
18. Masahiro Mambo and Eiji Okamoto. Proxy cryptosystems: Delegation of the power to decrypt ciphertexts. *IEICE transactions on fundamentals of electronics, Communications and computer sciences*, 80(1):54–63, 1997.
19. Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.
20. Jun Shao and Zhenfu Cao. Cca-secure proxy re-encryption without pairings. In *Public Key Cryptography - PKC 2009, 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings*, pages 357–376, 2009.

21. Anat Talmy and Oren Dobzinski. Abuse freedom in access control schemes. In *20th International Conference on Advanced Information Networking and Applications (AINA 2006), 18-20 April 2006, Vienna, Austria*, pages 77–86, 2006.
22. Smith. Tony. Dvd jon: buy drm-less tracks from apple itunes. http://www.theregister.co.uk/2005/03/18/itunes pymusique, 2005.
23. Jian Weng, Robert H. Deng, Shengli Liu, and Kefei Chen. Chosen-ciphertext secure bidirectional proxy re-encryption schemes without pairings. *Inf. Sci.*, 180(24):5077–5089, 2010.