Efficient Unidirectional Proxy Re-Encryption*

Sherman S.M. Chow[1], Jian Weng[2,3,4],
Yanjiang Yang[5], and Robert H. Deng[3]

[1] Department of Computer Science
Courant Institute of Mathematical Sciences
New York University, NY, USA
schow@cs.nyu.edu
[2] Department of Computer Science, Jinan University, Guangzhou, China
[3] School of Information Systems, Singapore Management University, Singapore
cryptjweng@gmail.com, robertdeng@smu.edu.sg
[4] State Key Laboratory of Information Security
Institute of Software, Chinese Academy of Sciences, Beijing, China
[5] Institute for Infocomm Research, Singapore
yyang@i2r.a-star.edu.sg

Abstract. Proxy re-encryption (PRE) allows a semi-trusted proxy to convert a ciphertext originally intended for Alice into one encrypting the same plaintext for Bob. The proxy only needs a re-encryption key given by Alice, and cannot learn anything about the plaintext encrypted. This adds flexibility in various applications, such as confidential email, digital right management and distributed storage. In this paper, we study *unidirectional* PRE, which the re-encryption key only enables delegation in one direction but not the opposite. In PKC 2009, Shao and Cao proposed a unidirectional PRE assuming the random oracle. However, we show that it is vulnerable to chosen-ciphertext attack (CCA). We then propose an efficient unidirectional PRE scheme (without resorting to pairings). We gain high efficiency and CCA-security using the "token-controlled encryption" technique, under the computational Diffie-Hellman assumption, in the random oracle model and a relaxed but reasonable definition.

Keywords: proxy re-encryption, unidirection, chosen-ciphertext attack

1 Introduction

Every application which requires some sort of confidentiality uses encryption as a building block. As pointed out by Mambo and Okamoto [MO97], the encrypted data often needs to be re-distributed in practice, i.e., the data encrypted under a public key $\mathtt{pk}_i$ should also be encrypted under another independently generated public key $\mathtt{pk}_j$. This can be easily done if the holder of the secret key $\mathtt{sk}_i$ (corresponding to $\mathtt{pk}_i$) is *online* – simply decrypts the ciphertext and re-encrypts the plaintext to $\mathtt{pk}_j$. However, this is not always practical. It is also undesirable to just disclose the secret key to some untrusted server to do the transformation of ciphertexts.

To solve this key management problem which hinders the practical adoption of encryption, Blaze, Bleumer and Strauss [BBS98] introduced the concept of proxy re-encryption (PRE). PRE schemes allow a secret key holder to create a re-encryption key. A semi-trusted proxy can use this key to translate a message m encrypted under the delegator's public key into an encryption of the same message under a delegatee's public key, as specified by the delegator.

* This work is partially supported by the Office of Research, Singapore Management University. It is also partially supported by the National Science Foundation of China under Grant No. 60903178. We thank Jun Shao for a discussion of the attack. This is a preliminary full version of our Africacrypt 2010 paper.

This can be done without allowing the proxy any ability to perform tasks outside of these proxy delegations. In particular, the proxy can neither recover the delegator's secret key nor decrypt the delegator's ciphertext.

Proxy re-encryption schemes have applications in digital rights management (DRM) [Smi05], distributed file storage systems [AFGH06], law enforcement [ID03], encrypted email forwarding [BBS98], and outsourced filtering of encrypted spam [AFGH06]. In all these cases, the gist is that the process of re-encryption, i.e., decrypting under one key for encryption under another key, should not allow the re-encryptor module to compromise the secrecy of encrypted messages. This was related to the compromise of Apple's iTunes DRM [Smi05]. With a PRE scheme, the problem is solved since re-encryption can be performed without awarding the proxy any information about the encrypted message. Besides DRM, distributed file storage systems also benefit in the sense that the storage server (proxy) can re-encrypt the files for different servers without knowing the underlying file content, and hence it is less attractive for hacker attacks since compromising the server does not compromise the files. Similarly, email servers can re-encrypt emails for different users with the same effect, say when a user is on vacation and wants to forward his encrypted emails to his colleague.

1.1 The Use of Pairings in Proxy Re-Encryption

Blaze, Bleumer and Strauss's seminal work [BBS98] proposed a bidirectional PRE scheme against chosen plaintext attack (CPA). However, as indicated by [AFGH06], their scheme has a few shortcomings – 1) the delegation in their scheme is transitive, which means that the proxy alone can create delegation rights between two entities that have never agreed on this, 2) the delegator's secret key can be recovered in full if the proxy and the delegate collude. Afterwards, a number of PRE schemes have been proposed. Their properties are summarized in Table 1. The schemes are chronologically arranged.

Schemes	Uni/Bi Directional	Security	RO -Free	Pairing -Free	Collusion -Resistant
Public-key-based					
Ateniese *et al.* [AFGH06]	$\rightarrow$	CPA	$\times$	$\times$	✓
Hohenberger *et al.* [HRSV07]	$\rightarrow$	CPA	✓	$\times$	✓
Canetti-Hohenberger [CH07]	$\leftrightarrow$	CCA	✓	$\times$	$\times$
Libert-Vergnaud [LV08c]	$\rightarrow$	RCCA	✓	$\times$	✓
Libert-Vergnaud-Trace [LV08b]	$\rightarrow$	CPA	✓	$\times$	✓
Deng *et al.* [DWLC08]	$\leftrightarrow$	CCA	$\times$	✓	$\times$
Shao-Cao [SC09]	$\rightarrow$	CCA?	$\times$	✓	$\times$
Ateniese *et al.* [ABH09]	$\rightarrow$	CPA	✓	$\times$	✓
Ours	$\rightarrow$	CCA	$\times$	✓	✓
Identity-based					
Green-Ateniese [GA07]	$\rightarrow$	CCA	$\times$	$\times$	$\times$
Chu-Tzeng [CT07]	$\rightarrow$	RCCA	✓	$\times$	$\times$

Table 1. Summary of PRE Schemes.

In this paper, we study unidirectional public-key-based PRE schemes which are secure against adaptive chosen-ciphertext attack (CCA). Informally, CCA models an adversary who can choose many ciphertexts and obtain their decryption under an unknown key, after seeing the challenge ciphertext (the one encrypting the message of interest) and previous decryption results. CCA-secure schemes often require ciphertext validity checking. As shown in Table 1, most existing PRE schemes most existing PRE schemes no matter ID-based or not, are realized by pairings. Below we look into two schemes to see why pairing is a useful "ingredient". In the bidirectional scheme proposed by Canetti and Hohenberger [CH07], the transformation key is simply $\mathtt{rk}_{i\leftrightarrow j} = x_j/x_i \in \mathbb{Z}_p$ for the pair of delegation partners[6] $\mathtt{pk}_i = g^{x_i}$ and $\mathtt{pk}_j = g^{x_j}$. The ciphertext comes with the term $\mathtt{pk}_i^r$ for randomness $r \in \mathbb{Z}_p$ which can be transformed to $\mathtt{pk}_j^r$ easily by using $\mathtt{rk}_{i\leftrightarrow j}$. The ciphertext validity can be checked with the help of the pairing function $\hat{e}(\cdot,\cdot)$ with respect to the generator g and the public key $\mathtt{pk}_i$ or $\mathtt{pk}_j$. For the unidirectional PRE scheme proposed by Libert and Vergnaud [LV08c] (hereinafter referred as $\mathcal{LV}08$), the transformation key is in the form $\mathtt{rk}_{i\leftrightarrow j} = g^{x_j/x_i}$. The ciphertext also comes with the term $\mathtt{pk}_i^r$ and the message is encrypted by $\hat{e}(g,g)^r$. To recover the message, a pairing will be applied to get $\hat{e}(g^{x_j/x_i}, \mathtt{pk}_i^r) = \hat{e}(g, g^r)^{x_j}$, $\hat{e}(g,g)^r$ can then be covered with x_j. These techniques for unidirectional transformation and ciphertext validity checking intrinsically require the pairings. Moreover, the security guarantee provided by $\mathcal{LV}08$ is only against replayable chosen-ciphertext attacks (RCCA) [CKN03], a weaker variant of CCA tolerating a "harmless mauling" of the challenge ciphertext.

1.2 Our Contributions

From a theoretical perspective, we would like to have PRE scheme realized under a broader class of complexity assumptions, and see techniques other than using pairing in constructing CCA-secure PRE. Practically, we want a PRE scheme with simple design, short ciphertext size and high computational efficiency[7]. Removing pairing from PRE constructions is one of the open problems left by [CH07].

Recently, Shao and Cao [SC09] proposed a unidirectional PRE scheme without pairings (referred as $\mathcal{SC}09$). Let N be a safe-prime modulus. $\mathcal{SC}09$ requires 4 to 5 exponentiations in $\mathbb{Z}_{N^2}^*$ for encryption, re-encryption and decryption[8], and incurs an ciphertext overhead of 3 (plus proof-of-knowledge) to 5 $\mathbb{Z}_{N^2}^*$ elements. The modulus being used is N^2. Its performance over pairing-based scheme (e.g., $\mathcal{LV}08$), which is instantiated on elliptic curves consist of much shorter group elements at the same security level, is questionable. Their security proof relies on the random oracle and the decisional (not computational) Diffie-Hellman assumption over $\mathbb{Z}_{N^2}^*$.

Most importantly, we identify flaws in their security proof which translate to a real-world chosen-ciphertext attack against $\mathcal{SC}09$. A possible fix further degrades the performance in decryption time. In view of this, we propose an efficient unidirectional CCA-secure PRE scheme without pairings, under the *standard* computational Diffie-Hellman assumption, in the random oracle model. Our design is based on ElGamal encryption [Gam84] and Schnorr signature [Sch91], which is (arguably) simple. Our decryption process is more natural and does not require the input of the delegator's public key, which is required in $\mathcal{SC}09$.

---

[6] For the bidirectional schemes, once a delegation is made, a delegator becomes a delegatee and a delegate becomes a delegator simultaneously.

[7] In spite of the recent advances in implementation technique, compared with modular exponentiation, pairing is still considered as a rather expensive operation, especially in computational resource limited settings.

[8] Speed-up by Chinese remainder theorem is not possible except 2 exponentiations in decryption, due to the lack of the factoring of the delegator's modulus.

In this paper, collusion attack refers to any collusion of a proxy and a delegatee which aimed to comproise the security of the delegator in *any* meaningful way.[9] Finally, to the best of our knowledge, there was no (R)CCA-secure unidirectional scheme which is collusion-resistant.

1.3 Related Notions

Proxy *encryption* (no "re-") (e.g., [MO97,Jak99,ID03]) also allows a delegator Alice to delegate her decryption power to a delegatee Bob with the help of a proxy. ciphertext for Bob. Different from PRE, these schemes require Alice to split her secret key between Bob and the proxy. In other words, Bob needs to obtain and store an additional secret for *each* decryption delegation. This may introduce other key management issues. In PRE, Bob just needs to use his own secret to decrypt ciphertext originally addressed to him or ciphertext transformed for him. Theoretically, he can be totally unaware of the delegation until he received the first transformed ciphertext from the proxy. As argued in [CH07,LV08c], PRE is a (strict) subset of proxy encryption.

Another notion with a similar name is universal re-encryption [GJJS04], in which the ciphertexts are re-randomized, but the underlying public keys are not changed as in PRE.

2 Our Definitions of Unidirectional Proxy Re-Encryption

2.1 Framework of Unidirectional Proxy Re-Encryption

A unidirectional PRE scheme consists of the following six algorithms [CH07]:

$\mathsf{Setup}(\kappa)$**:** The setup algorithm takes as input a security parameter κ and outputs the global parameters $\mathtt{param}$, which include a description of the message space $\mathcal{M}$.

$\mathsf{KeyGen}()$**:** The key generation algorithm generates a public/private key pair $(\mathtt{pk}_i, \mathtt{sk}_i)$.

$\mathsf{ReKeyGen}(\mathtt{sk}_i, \mathtt{pk}_j)$**:** The re-encryption key generation algorithm takes as input a private key $\mathtt{sk}_i$ and another public key $\mathtt{pk}_j$. It outputs a re-encryption key $\mathtt{rk}_{i\to j}$.

$\mathsf{Encrypt}(\mathtt{pk}, m)$**:** The encryption algorithm takes as input a public key $\mathtt{pk}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext $\mathfrak{C}$ under $\mathtt{pk}$.

$\mathsf{ReEncrypt}(\mathtt{rk}_{i\to j}, \mathfrak{C}_i)$**:** The re-encryption algorithm takes as input a re-encryption key $\mathtt{rk}_{i\to j}$ and a ciphertext $\mathfrak{C}_i$ under public key $\mathtt{pk}_i$. It outputs a ciphertext $\mathfrak{C}_j$ under public key $\mathtt{pk}_j$. This can be either deterministic or probabilistic.

$\mathsf{Decrypt}(\mathtt{sk}, \mathfrak{C})$**:** The decryption algorithm takes as input a private key $\mathtt{sk}$ and a ciphertext $\mathfrak{C}$. It outputs a message $m \in \mathcal{M}$ or the error symbol $\perp$ if the ciphertext is invalid.

To lighten notations, we omit the public parameters $\mathtt{param}$ as the input of the algorithms. Correctness requires that, for any parameters $\mathtt{param}$, $m \in \mathcal{M}$, the following probabilities are

[9] For example, the collusion-resistance claimed in [SC09] can be more accurately described as delegator-secret-key security (also see Section 2.2), and we listed it as *not* collusion-resistant due to the following attack. A collusion of a delegatee of X and his proxy can recover a weak secret key ($\mathtt{wsk}_X$) of X. Any re-enryption of ciphertext of X to *other* delegatee contains most part of the original one, in particular, it is decryptable by applying $\mathtt{wsk}_X$ on the original components (also see Section 3.)

equal to 1:

$$\Pr\left[\mathsf{Decrypt}(\mathtt{sk}_i, \mathfrak{C}) = m \,\middle|\, (\mathtt{sk}_i, \mathtt{pk}_i) \leftarrow \mathsf{KeyGen}(), \mathfrak{C} \leftarrow \mathsf{Encrypt}(\mathtt{pk}_i, m)\right],$$

$$\Pr\left[\mathsf{Decrypt}(\mathtt{sk}_j, \mathfrak{C}_j) = m \,\middle|\, \begin{array}{c} (\mathtt{sk}_i, \mathtt{pk}_i) \leftarrow \mathsf{KeyGen}(), \\ (\mathtt{sk}_j, \mathtt{pk}_j) \leftarrow \mathsf{KeyGen}(), \\ \mathtt{rk}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathtt{sk}_i, \mathtt{pk}_j), \\ \mathfrak{C}_i \leftarrow \mathsf{Encrypt}(\mathtt{pk}_i, m), \\ \mathfrak{C}_j \leftarrow \mathsf{ReEncrypt}(\mathtt{rk}_{i\to j}, \mathfrak{C}_i) \end{array}\right]$$

2.2 Security Models for "Token-Controlled" Re-Encryption

Our game-based definitions for single-hop unidirectional PRE systems are adaptions of the definitions of the original (second level) ciphertext security and the transformed (first level) ciphertext security in [LV08c]. As in [CH07,LV08c] our static corruption model makes the knowledge of secret key (KOSK) assumption, the adversary only gets uncorrupted public key or corrupted public/private key pair from the challenger, and is not allowed to adaptively determine which parties will be compromised. Compared with [CH07,LV08c], our definition considers the standard CCA security instead of RCCA security. However, this is at the expense of a relaxation requiring additional constraint on the re-encryption key that can be compromised.

Definition 1 (Game Template of Chosen-Ciphertext Security).

Setup. *The challenger $\mathcal{C}$ takes a security parameter κ and executes the setup algorithm to get the system parameters* $\mathtt{param}$. *$\mathcal{C}$ executes the key generation algorithm n_u times resulting a list of public/private keys $\mathcal{PK}_{good}, \mathcal{SK}_{good}$, and executes the key generation algorithm for n_c times to get a list of corrupted public/private keys $\mathcal{PK}_{corr}, \mathcal{SK}_{corr}$. $\mathcal{A}$ gets* $\mathtt{param}$, *$\mathcal{SK}_{corr}$, and $\mathcal{PK} = (\mathcal{PK}_{good} \cup \mathcal{PK}_{corr}) = \{\mathtt{pk}_i\}_{i\in[1,n_u+n_c]}$.*

Phase 1. *$\mathcal{A}$ adaptively queries to oracles* OReK, OReE *and* ODec.

- OReK *oracle takes $\langle \mathtt{pk}_i, \mathtt{pk}_j \rangle$ and returns a re-encryption key $\mathtt{rk}_{i\to j}$.*
- OReE *oracle takes public keys $\langle \mathtt{pk}_i, \mathtt{pk}_j \rangle$ and a ciphertext $\mathfrak{C}$ and returns a re-encryption of $\mathfrak{C}$ from $\mathtt{pk}_i$ to $\mathtt{pk}_j$.*
- ODec *oracle takes a public key* $\mathtt{pk}$ *and a ciphertext $\mathfrak{C}$ and returns the decryption of $\mathfrak{C}$ using the private key with respect to* $\mathtt{pk}$.

Challenge. *When $\mathcal{A}$ decides that Phase 1 is over, it also decides whether it wants to be challenged with a original ciphertext or a transformed ciphertext. It outputs two equal-length plaintexts $m_0, m_1 \in \mathcal{M}$, and a target public key $\mathtt{pk}_{i^*}$. Challenger $\mathcal{C}$ flips a random coin $\delta \in \{0,1\}$, and sends to $\mathcal{A}$ a challenge ciphertext $\mathfrak{C}^*$ depending on $\mathtt{pk}_{i^*}$ and m_δ*

Phase 2. *$\mathcal{A}$ issues queries as in Phase 1.*

Guess. *Finally, $\mathcal{A}$ outputs a guess $\delta' \in \{0,1\}$.*

The public keys supplied by $\mathcal{A}$ subject to the following constraints:

1. *The public keys involved in all queries must come from $\mathcal{PK}$.*
2. *The target public key $\mathtt{pk}_{i^*}$ is from $\mathcal{PK}_{good}$, i.e., uncorrupted.*

The actual construction of $\mathfrak{C}^*$ and the constraints on the queries made by $\mathcal{A}$ are to be defined according to different security notions.

Definition 2 (Original Ciphertext Security). *For original ciphertext security, the adversary $\mathcal{A}$ plays the CCA game with the challenger $\mathcal{C}$ as in Definition 1, where the challenge ciphertext is formed by $\mathfrak{C}^* = \mathsf{Encrypt}(\mathtt{pk}_{i^*}, m_\delta)$, and $\mathcal{A}$ has the following additional constraints:*

1. $\mathsf{OReK}(\mathtt{pk}_{i^*}, \mathtt{pk}_j)$ *is only allowed if* $\mathtt{pk}_j$ *came from* $\mathcal{PK}_{good}$.
2. *If* $\mathcal{A}$ *issued* $\mathsf{OReE}(\mathtt{pk}_i, \mathtt{pk}_j, \mathfrak{C}_i)$ *where* $\mathtt{pk}_j$ *came from* $\mathcal{PK}_{corr}$, $(\mathtt{pk}_i, \mathfrak{C}_i)$ *cannot be a* derivative *of* $(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$ *(to be defined later).*
3. $\mathsf{ODec}(\mathtt{pk}, \mathfrak{C})$ *is only allowed if* $(\mathtt{pk}, \mathfrak{C})$ *is not a* derivative *of* $(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$.

Definition 3 (Derivative for Chosen-Ciphertext Security). *Derivative of* $(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$ *in the CCA setting is inductively defined in [SC09] as below, which is adopted from the RCCA-based definition in [CH07]*[10]*:*

1. *Reflexivity:* $(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$ *is a derivative of itself.*
2. *Derivation by re-encryption: If* $\mathcal{A}$ *has issued a re-encryption query* $\langle \mathtt{pk}, \mathtt{pk}', \mathfrak{C} \rangle$ *and obtained the resulting re-encryption ciphertext* $\mathfrak{C}'$, *then* $(\mathtt{pk}', \mathfrak{C}')$ *is a derivative of* $(\mathtt{pk}, \mathfrak{C})$.
3. *Derivation by re-encryption key: If* $\mathcal{A}$ *has issued a re-encryption key generation query* $\langle \mathtt{pk}, \mathtt{pk}' \rangle$ *to obtain the re-encryption key* $\mathtt{rk}$, *and* $\mathfrak{C}' = \mathsf{ReEncrypt}(\mathtt{rk}, \mathfrak{C})$, *then* $(\mathtt{pk}', \mathfrak{C}')$ *is a derivative of* $(\mathtt{pk}, \mathfrak{C})$.

Definition 4 (Transformed Ciphertext Security). *For transformed ciphertext, the adversary* $\mathcal{A}$ *plays the CCA game with the challenger* $\mathcal{C}$ *as in Definition 1, where* $\mathcal{A}$ *can also specify the delegator* $\mathtt{pk}_{i'}$. *The challenge ciphertext is then created by the re-encryption process, specifically,* $\mathfrak{C}^* = \mathsf{ReEncrypt}(\mathtt{rk}_{i' \to i^*}, \mathsf{Encrypt}(\mathtt{pk}_{i'}, m_\delta))$. *The only constraints of* $\mathcal{A}$ *are:*

1. $\mathsf{ODec}(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$ *is not allowed.*
2. *If* $\mathtt{pk}_{i'}$ *came from* $\mathcal{PK}_{corr}$, $\mathcal{C}$ *would not return* $\mathtt{rk}_{i' \to i^*}$ *to* $\mathcal{A}$ *in phase 2.*
3. *If* $\mathcal{A}$ *obtained* $\mathtt{rk}_{i' \to i^*}$, $\mathcal{A}$ *cannot choose* $\mathtt{pk}_{i'}$ *as the delegator in the challenge phase.*

This can be considered as a weaker notion when compared with [LV08c].

Definition 5 (CCA Security of a PRE). *We define* $\mathcal{A}$*'s advantage in attacking the PRE scheme as* $Adv_{PRE,\mathcal{A}}^{IND\text{-}PRE\text{-}CCA} = \left| \Pr[\delta' = \delta] - 1/2 \right|$, *where the probability is taken over the random coins consumed by the challenger and the adversary. A single-hop unidirectional PRE scheme is defined to be* $(t, n_u, n_c, q_{rk}, q_{re}, q_d, \epsilon)$*-IND-PRE-CCA secure, if for any* t*-time IND-PRE-CCA adversary* $\mathcal{A}$ *who makes at most* q_{rk} *re-encryption key generation queries, at most* q_{re} *re-encryption queries and at most* q_d *decryption queries, we have* $\mathrm{Adv}_{\mathrm{PRE},\mathcal{A}}^{\mathrm{IND-PRE-CCA}} \leq \epsilon$.

Derivative and Two Different Kinds of Security. Intuitively speaking, original ciphertext security models the an adversary $\mathcal{A}$ challenged with an untransformed ciphertext encrypted for a target user i^*. In a PRE scheme, however, $\mathcal{A}$ can ask for the re-encryption of many ciphertexts or even a set of re-encryption keys. These queries are allowed as long as they would not allow $\mathcal{A}$ to decrypt trivially. For examples, $\mathcal{A}$ *should not* get the re-encryption key from user i^* to user j if the secret key of user j has been compromised; on the other hand, $\mathcal{A}$ can certainly get a re-encryption of the challenge ciphertext from user i^* to user j as long as j is an honest user and the decryption oracle of user j has not been queried with the

[10] These original definitions also consider transitivity – If $(\mathtt{pk}, \mathfrak{C})$ is a derivative of $(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$ and $(\mathtt{pk}', \mathfrak{C}')$ is a derivative of $(\mathtt{pk}, \mathfrak{C})$, then $(\mathtt{pk}', \mathfrak{C}')$ is a derivative of $(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$. However, this is irrelevant for single-hop scheme like ours and [SC09].

resulting transformed ciphertext. This explains the intuition behind the notion of derivative and the associated restrictions.

Since $\mathcal{A}$ can derive a transformed ciphertext with a certain related re-encryption key, one may wonder why there is another notion about transformed ciphertext security. This latter notion makes sense when the PRE system is *single-hop*, i.e., a transformed ciphertext cannot be re-encrypted further to someone else. If a proxy colludes with a delegatee, by the correct functionalities of a PRE, this collusion group can certainly decrypt any *original* ciphertext of the target user. However, for a single-hop scheme, there is no reason that this collusion group can decrypt any *transformed* ciphertext since it cannot be re-encrypted further. To conclude, the adversary is allowed to transform an original ciphertext in the former notion, but there are some re-encryption keys which it is not allowed to get (recall the constraints related to derivatives); while in the latter, the adversary only sees the transformed ciphertext but not the original one, and the adversary can get more re-encryption keys.

Our Definition of Transformed Ciphertext Security. The second constraint deserves more discussion. The compromise of $\mathtt{rk}_{i' \to i^*}$ corresponds to the fact that the proxy, which is designated by the delegator $\mathtt{pk}_{i'}$ for the delegation to the delegatee $\mathtt{pk}_{i^*}$, is compromised. Ideally, it seems that whether the delegator $\mathtt{pk}_{i'}$ is compromised or not in this situation does not affect the security of the transformed ciphertext for $\mathtt{pk}_{i^*}$. This is also what has been modelled by the definition in [LV08c]. However, if the adversary $\mathcal{A}$ compromised the delegator $\mathtt{pk}_{i'}$ and also the proxy, $\mathcal{A}$ can simply ask the proxy to surrender the original ciphertext $\mathsf{Encrypt}(\mathtt{pk}_{i'}, m_\delta)$ before any actual transformation, and use $\mathtt{sk}_{i'}$ to decrypt trivially. It is true that if the proxy was initially honest and erased the original ciphertexts after their transformation, the same attack does not apply; however, ciphertext is by definition public in nature and the adversary may have captured the ciphertext already and decrypt it when $\mathtt{sk}_{i'}$ is obtained. We believe that the relaxed notion still have significance in the real world.

Nontransformable (First-Level) Ciphertext. To view the above relaxation from another angle, one may feel that we lost a possible benefit of a single-hop scheme – some ciphertexts are not further transformable so very sensitive information can be encrypted in this form ("first level" ciphertext that cannot be re-encrypted). Actually, our definition does not rule out this possibility. Our definition given above only considers *transformed* ciphertext, that is, the challenge ciphertext which is generated from the re-encryption algorithm. It does not rule out the possibility of having another encryption algorithm $\mathsf{Encrypt}_1$ which directly produces non-transformable ciphertext, when $\mathsf{ReEncrypt}(\mathtt{rk}_{i' \to i^*}, \mathsf{Encrypt}(\mathtt{pk}_{i'}, m_\delta))$ and $\mathsf{Encrypt}_1(\mathtt{pk}_{i^*}, m_\delta)$ are actually distinguishable.

We view this as one way to get CCA security instead of RCCA security. Using $\mathcal{LV}08$, it is possible to directly encrypt ciphertexts that cannot be re-encrypted which is indistinguishable from re-encryption, and the reason is that re-randomization can be done in the re-encryption process. Recall that the security guarantee of $\mathcal{LV}08$ actually allows the adversary to compromise all proxies of the system; indeed, the re-randomizaation in $\mathcal{LV}08$ can be done by any one without any secret knowledge – this explains why $\mathcal{LV}08$ is at most RCCA secure.

Of course, it is required to augment the PRE systems with yet another encryption algorithm. However, it is often the case that the original decryption algorithm sufficies to decrypt ciphertext produced in this way. The interface of $\mathsf{Encrypt}_1$ and its correctness requirement are exactly the same as those of $\mathsf{Encrypt}$. The security definition is also simple.

Definition 6 (Nontransformable Ciphertext Security). *For nontransformable ciphertext, the adversary $\mathcal{A}$ plays the CCA game with the challenger $\mathcal{C}$ as in Definition 1, where the challenge ciphertext is given by $\mathfrak{C}^* = \mathsf{Encrypt}_1(\mathtt{pk}_{i^*}, m_\delta)$, and $\mathcal{A}$ is disallowed from making $\mathsf{ODec}(\mathtt{pk}_{i^*}, \mathfrak{C}^*)$ query only. In particular, $\mathcal{A}$ can get* all *the re-encryption keys.*

Delegator/Master Secret Security. Delegator secret security[11] is considered in Ateniese *et al.* [AFGH06] which captures the intuition that, even if a dishonest proxy colludes with the delegatee, they still cannot derive the delegator's private key in full. The attack mode is quite simple and can be covered by the nontransformable / first-level ciphertext security [LV08c]. The reason behind is easy to see – there is no restriction in the re-encryption key generation queries, and decryption is easy when the adversary can derive the delegator's private key in full.

3 Analysis of a CCA-Secure Unidirectional PRE Scheme

3.1 Review of Shao-Cao's Scheme

$\mathcal{SC}09$ [SC09] is reviewed as below, up to minor notational differences. We use $\boxed{}$ to highlight the places which introduce the vulnerability.

$\mathsf{Setup}(\kappa)$**:** Given a security parameter κ, choose three hash functions $H_1 : \{0,1\} \to \{0,1\}^{\ell_1}$, $H_2 : \{0,1\} \to \{0,1\}^{\ell_2}$, and $H_3 : \{0,1\} \to \{0,1\}^{\ell_3}$, where ℓ_1, ℓ_2 and ℓ_3 are determined by κ, and the message space $\mathcal{M}$ is $\{0,1\}^{\ell_2}$. The parameters are $\mathtt{param} = (\kappa, H_1, H_2, H_3, \ell_1, \ell_2, \ell_3)$.

$\mathsf{KeyGen}()$**:** Given a security parameter κ, perform the following steps:
1. Choose two distinct Sophie Germain primes p' and q' of κ-bit.
2. Compute safe primes $p = 2p' + 1$ and $q = 2q' + 1$ (their primalities are guaranteed since p' and q' are Sophie Germain primes).
3. Compute a safe-prime modulus $N = pq$.
4. Store $\mathtt{sk} = (p, q, p', q')$ as the long term secret key.
5. Choose a hash function $H : \{0,1\}^* \to \mathbb{Z}_{N^2}$.
6. Pick $a, b \xleftarrow{\$} [1, pp'qq']$, store $\mathtt{wsk} = (a, b)$ as the "weak" secret key.
7. Randomly pick $\alpha \in \mathbb{Z}^*_{N^2}$, set $g_0 = \alpha^2 \bmod N^2$, $g_1 = g_0^a \bmod N^2$, and $g_2 = g_0^b \bmod N^2$; the public key is $\mathtt{pk} = (H(), N, g_0, g_1, g_2)$.

Either secret key can be used to decrypt (any) ciphertexts, but both of them are required to produce a re-encryption key. Note that in the following description, the elements from the key of user X contain an additional subscript of X, e.g., $\mathtt{pk}_X = (H_X(\cdot), N_X, g_{X0}, g_{X1} = g_{X0}^{a_X}, g_{X2})$.

$\mathsf{ReKeyGen}(\mathtt{sk}_X, \mathtt{pk}_Y)$**:** On input a long term secret key (p_X, q_X, p'_X, q'_X), a weak secret (a_X, b_X), and a public key $\mathtt{pk}_Y = (H_Y, N_Y, g_{Y0}, g_{Y1}, g_{Y2})$, it outputs the re-encryption key $rk_{X\to Y} = (rk^{(1)}_{X\to Y}, rk^{(2)}_{X\to Y})$, where $rk^{(1)}_{X\to Y} = (\dot{A}, \dot{B}, \dot{C})$, as follows:
1. Pick $\dot{\beta} \xleftarrow{\$} \{0,1\}^{\ell_1}$, compute $rk^{(2)}_{X\to Y} = a_X - \dot{\beta} \bmod (p_X q_X p'_X q'_X)$.

[11] This notion is named as *master secret security* in [AFGH06] since the delegator's public key is the master public key in their secure distributed storage application. It is also called "collusion-resistance" in some literatures.

2. Pick $\dot{\sigma} \xleftarrow{\$} \mathbb{Z}_{N_Y}$, compute $r_{X\to Y} = H_Y(\dot{\sigma}\|\dot{\beta})$.
3. Compute $\dot{C} = H_1(\dot{\sigma}) \oplus \dot{\beta}$.
4. Compute $\dot{A} = (g_{Y0})^{r_{X\to Y}} \bmod (N_Y)^2$.
5. Compute $\dot{B} = (g_{Y2})^{r_{X\to Y}} \cdot (1 + \dot{\sigma}N_Y) \bmod (N_Y)^2$.

$\mathsf{Encrypt}(\mathtt{pk} = (H(), N, g_0, g_1, g_2), m)$**:** To encrypt a message $m \in \mathcal{M}$:
1. Randomly pick $\sigma \in \mathbb{Z}_N$, compute $r = H(\sigma\|m)$.
2. Compute $C = H_2(\sigma) \oplus m$.
3. Compute $A = (g_0)^r \bmod N^2$, $B = (g_1)^r \cdot (1 + \sigma N) \bmod N^2$ and $D = (g_2)^r \bmod N^2$.
4. Run $(c, s) \leftarrow \mathsf{SoK.Gen}(A, D, g_0, g_2, (B, C))$, where the underlying hash function is H_3.[12]
5. Output the ciphertext $\mathfrak{C} = (A, B, C, D, c, s)$.

$\mathsf{ReEncrypt}(\mathtt{rk}_{X\to Y}, \mathfrak{C}_X, \mathtt{pk}_X, \mathtt{pk}_Y)$**:** On input a re-encryption key $\mathtt{rk}_{X\to Y} = (\mathtt{rk}^{(1)}_{X\to Y}, \mathtt{rk}^{(2)}_{X\to Y})$ and a ciphertext $\mathfrak{C} = (A, B, C, D, c, s)$ under key $\mathtt{pk}_X = (H_X, N_X, g_{X0}, g_{X1}, g_{X2})$,
1. Check if $c = H_3(A\|D\|g_{X0}\|g_{X2}\|(g_{X0})^s A^c\|(g_{X2})^s D^c\ |(B\|C))$. If not, return $\perp$.
2. Otherwise, compute $A' = A^{\mathtt{rk}^{(2)}_{X\to Y}}$.
3. Output $\mathfrak{C}_Y = (A, \boxed{A'}, B, C, \mathtt{rk}^{(1)}_{X\to Y}) = (A, A', B, C, \dot{A}, \dot{B}, \dot{C})$.

The only "new" thing in $\mathfrak{C}_Y$ is $A' = (g_{X0})^{r(a_X - \dot{\beta})} \bmod (N_X)^2 = (g_{X1})^r \boxed{(g_{X0})^{-r\dot{\beta}}} \bmod (N_X)^2$. The second equality holds since $g_{X1} = g_{X0}^{a_X}$, by the public key construction in KeyGen.

$\mathsf{Decrypt}(\mathtt{sk}, \mathfrak{C})$**:** On input a private key and a ciphertext $\mathfrak{C}$, parse $\mathfrak{C}$,

- If $\mathfrak{C}$ is an original ciphertext in the form $\mathfrak{C} = (A, B, C, D, c, s)$:
  1. Return $\perp$ if $c \neq H_3(A\|D\|g_0\|g_2\|(g_0)^s A^c\|(g_2)^s D^c\|(B\|C))$.
  2. If $\mathtt{sk}$ is in the form of (a, b), compute $\sigma = \frac{B/(A^a) - 1 \bmod N^2}{N}$.
  3. If $\mathtt{sk} = (p, q, p', q')$, compute $\sigma = \frac{(B/g_0^{w_1})^{2p'q'} - 1 \bmod N^2}{N} \cdot \pi \bmod N$, where w_1 is computed as that in [BCP03], and π is the inverse of $2p'q' \bmod N$.
  4. Compute $m = C \oplus H_2(\sigma)$.
  5. If $B = (g_1)^{H(\sigma\|m)} \cdot (1 + \sigma N) \bmod N^2$, return m; else return $\perp$.
- If $\mathfrak{C} = (A, A', B, C, \dot{A}, \dot{B}, \dot{C})$ re-encrypted from $\mathtt{pk}_X$ to $\mathtt{pk}_Y$:
  1. If $\mathtt{sk}$ is in the form of (a, b), compute $\dot{\sigma} = \frac{\dot{B}/(\dot{A}^b) - 1 \bmod N_Y^2}{N_Y}$.
  2. If $\mathtt{sk} = (p, q, p', q')$, similar to decrypting an original ciphertext, compute $\dot{\sigma} = \frac{(\dot{B}/g_{Y0}^{w_1})^{2p'q'} - 1 \bmod N_Y^2}{N_Y} \cdot \pi \bmod N_Y$, .
  3. Compute $\dot{\beta} = \dot{C} \oplus H_1(\dot{\sigma})$.
  4. If $\dot{B} \neq (g_{Y1})^{H_Y(\dot{\sigma}\|\dot{\beta})} \cdot (1 + \dot{\sigma}N_Y) \bmod N_Y^2$, return $\perp$.
  5. Compute $\sigma = (B/(\boxed{A' \cdot A^{\dot{\beta}}}) - 1 \bmod N_X^2)/N_X$.
  6. Compute $m = C \oplus H_2(\sigma)$.
  7. Return m if $B = (g_{X1})^{H_X(\sigma\|m)} \cdot (1 + \sigma N_X) \bmod N_X^2$; else $\perp$.

  The **delegator**'s public key $(H_X, N_X, g_{X0}, g_{X1}, g_{X2})$ is required in the last few steps. This deviates from our framework in Section 2.

[12] A signature of knowledge (c, s) of the discrete logarithm of both $y_0 = g_0^x$ w.r.t. base g_0 and $y_2 = g_2^x$ w.r.t. base g_2, on a message $(B, C) \in \{0, 1\}^*$ can be computed by first picking $t \in \{0, \ldots, 2^{|N^2|+k} - 1\}$, then computing $c = H_3(y_0||y_2||g_0||g_2||g_0^t||h_0^t||m)$ and $s = t - cx$. This requires 2 exponentiations.

3.2 Possible Vulnerabilities in the Re-Encryption Key

Before describing our attack, we briefly explain how the re-encryption key is generated in $\mathcal{SC}09$. Their ReKeyGen algorithm follows the "token-controlled encryption" paradigm, which is adopted by [GA07,CT07] and our scheme to be presented. Specifically, ReKeyGen first selects a random token $\dot{\beta}$ to "hide" (some form of) the delegator's secret key a_X (i.e., $rk^{(2)}_{X\to Y} = a_X - \dot{\beta}$), and then encrypts this token $\dot{\beta}$ under the delegatee's public key, (i.e., $rk^{(1)}_{X\to Y} = (\dot{A}, \dot{B}, \dot{C})$).

Note that when the proxy and the delegatee collude, it is possible to recover a_X. So the encryption of the token should use a mechanism that is *different* from the usual encryption on the plaintext (i.e., $\dot{B}$ is computed using g_2 while B component in Encrypt is computed using g_1). Otherwise, it will subject to the following "chain collusion attack" mentioned in [SC09].

Imagine that Bob (who holds public key $\mathtt{pk}_Y$), who received delegation from Alice (who holds public key $\mathtt{pk}_X$), now delegates his own decryption right to Carol. *If* the ReKeyGen algorithm requires Bob to use $\mathtt{sk}_Y$ (i.e., the whole private key) instead of just *some form* of the private key (e.g., a_Y in $\mathcal{SC}09$), when his proxy colludes with Carol, $\mathtt{sk}_Y$ can be easily recovered. Furthermore, $\mathtt{sk}_Y$ can be used to recover $\dot{\beta}$ in the re-encryption key generated by Alice to Bob; the secret key of Alice, $\mathtt{sk}_X$, can also be recovered exactly in the way how $\mathtt{sk}_Y$ is recovered. This clearly compromises the security of Alice out of her expectation, since her only delegatee Bob has done nothing wrong (perhaps except using an insecure scheme). This is where the schemes [GA07,CT07] fail, as pointed by [SC09].

3.3 Our Attack

Shao and Cao [SC09] claimed that their PRE scheme is CCA-secure. However, in this section, we demonstrate that it is not the case.

Before describing our attack, we briefly explain how the re-encryption key is generated in $\mathcal{SC}09$. Their ReKeyGen algorithm follows the "token-controlled encryption" paradigm, which is adopted by [GA07,CT07] and our scheme to be presented. Specifically, ReKeyGen first selects a random token $\dot{\beta}$ to "hide" (some form of) the delegator's secret key a_X (i.e., $rk^{(2)}_{X\to Y} = a_X - \dot{\beta}$), and then encrypts this token $\dot{\beta}$ under the delegatee's public key (i.e., $rk^{(1)}_{X\to Y} = (\dot{A}, \dot{B}, \dot{C})$). First, we found that *any* re-encryption query (not necessary of the challenge ciphertext) reveals partial information about $\dot{\beta}$. Moreover, there is no validity check on the A' component of the transformed ciphertext. The combined effect leads us to the following efficient attacker $\mathcal{A}$, which aims to decrypt challenge ciphertext $\mathfrak{C}^* = (A, B, C, D, c, s)$ encrypted for $\mathtt{pk}^*_X = (H_X(\cdot), N_X, g_{X0}, g_{X1}, g_{X2})$.

1. Randomly pick $m \in \mathcal{M}$ and $r \in \mathbb{Z}_{(N_X)^2}$, compute $\mathfrak{C} \leftarrow \mathsf{Encrypt}_{\mathtt{pk}^*_X}(m; r)$, i.e., using r as the randomness in the first step of Encrypt.
   (Being a public key encryption, anyone can perform the encryption.)
2. Issue a re-encryption oracle query to re-encrypt the ciphertext $\mathfrak{C}$ from $\mathtt{pk}^*$ to $\mathtt{pk}$, in particular, $\mathcal{A}$ obtains $Z' = g_{X0}^{r(a_X - \dot{\beta})}$ as the second component of the resulting transformed ciphertext $\mathfrak{C}_0$. *(Z' here corresponds to $\boxed{A'}$ in the above description of $\mathcal{SC}09$.)*
3. Since Z' is in the form of $(g_{X1})^r \boxed{(g_{X0})^{-r\dot{\beta}}} \bmod (N_X)^2$, $\mathcal{A}$ can compute $(g_{X0})^{-r\dot{\beta}} \leftarrow (Z'/(g_{X1})^r)$. *($\mathfrak{C}$ is prepared by $\mathcal{A}$, so $\mathcal{A}$ knows r.)*
4. Issue a re-encryption oracle query to re-encrypt the ciphertext $\mathfrak{C}^*$ from $\mathtt{pk}^*$ to $\mathtt{pk}$, and obtain $\mathfrak{C}_1 = (A, A', B, C, \dot{A}, \dot{B}, \dot{C})$ as a result.
   (The secret key of $\mathtt{pk}$ *is not compromised by $\mathcal{A}$, so this is legitimate.)*

5. Pick $s \xleftarrow{\$} \mathbb{Z}_{(N_X)^2}$, compute $\mathfrak{A}' \leftarrow A' \cdot (g_{X0}^{-r\dot{\beta}})^s$ and $\mathfrak{A} \leftarrow A \cdot (g_{X0})^{rs}$.
6. Prepare $\mathfrak{C}' = (\mathfrak{A}, \mathfrak{A}', B, C, \dot{A}, \dot{B}, \dot{C})$ issue a decryption oracle query under $\mathtt{pk}$ to decrypt $\mathfrak{C}'$, and the result is the message encrypted in $\mathfrak{C}^*$.

To see the correctness of the attack, first note that $B, C, \dot{A}, \dot{B}, \dot{C}$ just come from the derivative $(\mathtt{pk}, \mathfrak{C}_1)$ of the challenge $(\mathtt{pk}^*, \mathfrak{C}^*)$, and they are the only values from the ciphertext being used for the first three steps of Decrypt, so the correct value of $\dot{\beta}$ can be recovered. Moreover, in Decrypt (refer to $\boxed{A' \cdot A^{\dot{\beta}}}$), $\mathfrak{A}'\mathfrak{A}^{\dot{\beta}} = A'(g_{X0}{}^{-r\dot{\beta}})^s(A \cdot g_{X0}{}^{rs})^{\dot{\beta}} = A' \cdot g_{X0}^{-r\dot{\beta}s} \cdot A^{\dot{\beta}} \cdot g_{X0}^{r\dot{\beta}s} = A'A^{\dot{\beta}}$, which is exactly what Decrypt will compute for the challenge.

Finally, $\mathfrak{C}'$ is *not* a derivative of $\mathfrak{C}^*$. To check against the definition of derivative: 1) $\mathfrak{C}^* \neq \mathfrak{C}'$; 2) $\mathcal{A}$ has made two re-encryption queries, $\mathfrak{C}$ has *nothing* to do with the challenge $\mathfrak{C}^*$, only $(\mathtt{pk}, \mathfrak{C}_1)$ is considered as a derivative of the challenge, but $(\mathtt{pk}, \mathfrak{C}')$, where $\mathfrak{C}_1 \neq \mathfrak{C}'$, is *not* its derivative; and 3) $\mathcal{A}$ has not made any re-encryption key generation oracle query at all.

3.4 Flaws in the Proof and A Possible Fix

This attack originated from some flaws in their proof [SC09], specifically, two rejection rules regarding A in the decryption oracle simulation. There is no checking of A when decrypting a transformed ciphertext in the real scheme, which makes a noticeable difference to the adversary. The crux of our attack is the formulation of a new A component. One possible fix is to re-compute A in Decrypt and check whether it is correctly generated, which requires one more exponentiation in $\mathbb{Z}_{N^2}$.

4 Our Proposed Unidirectional PRE Scheme

4.1 Construction

Our proposed unidirectional PRE scheme extends the *bidirectional* scheme proposed by Deng *et al.* [DWLC08], again by the "token-controlled encryption" technique. As previously discussed in Section 3, however, this should be carefully done to avoid possible attacks.

Setup(κ): Choose two primes p and q such that $q|p-1$ and the bit-length of q is the security parameter κ. Let g be a generator of group $\mathbb{G}$, which is a subgroup of $\mathbb{Z}_q^*$ with order q. Choose four hash functions $H_1 : \{0,1\}^{\ell_0} \times \{0,1\}^{\ell_1} \to \mathbb{Z}_q^*, H_2 : \mathbb{G} \to \{0,1\}^{\ell_0+\ell_1}, H_3 : \{0,1\}^* \to \mathbb{Z}_q^*$ and $H_4 : \mathbb{G} \to \mathbb{Z}_q^*$. The former three will be modeled as random oracles in our security proof. Here ℓ_0 and ℓ_1 are security parameters determined by κ, and the message space $\mathcal{M}$ is $\{0,1\}^{\ell_0}$. The parameters are $\mathtt{param} = (q, \mathbb{G}, g, H_1, H_2, H_3, H_4, \ell_0, \ell_1)$.

KeyGen(): Pick $\mathtt{sk}_i = (x_{i,1} \xleftarrow{\$} \mathbb{Z}_q^*, x_{i,2} \xleftarrow{\$} \mathbb{Z}_q^*)$ and set $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$.

ReKeyGen($\mathtt{sk}_i, \mathtt{pk}_j$): On input user i's private key $\mathtt{sk}_i = (x_{i,1}, x_{i,2})$ and user j's public key $\mathtt{pk}_j = (\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2})$, this algorithm generates the re-encryption key $\mathtt{rk}_{i\to j}$ as below:

1. Pick $h \xleftarrow{\$} \{0,1\}^{\ell_0}$ and $\pi \xleftarrow{\$} \{0,1\}^{\ell_1}$, compute $v = H_1(h, \pi)$.
2. Compute $V = \mathtt{pk}_{j,2}^v$ and $W = H_2(g^v) \oplus (h\|\pi)$.
3. Define $\mathtt{rk}_{i\to j}^{\langle 1\rangle} = \frac{h}{x_{i,1}H_4(\mathtt{pk}_{i,2})+x_{i,2}}$. Return $\mathtt{rk}_{i\to j} = (\mathtt{rk}_{i\to j}^{\langle 1\rangle}, V, W)$.

$\mathsf{Encrypt}(\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}), m)$**:** To encrypt a plaintext $m \in \mathcal{M}$:

1. Pick $u \xleftarrow{\$} \mathbb{Z}_q^*$ and compute $D = \left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})}\mathtt{pk}_{i,2}\right)^u$.
2. Pick $\omega \xleftarrow{\$} \{0,1\}^{\ell_1}$, compute $r = H_1(m, \omega)$.
3. Compute $E = \left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})}\mathtt{pk}_{i,2}\right)^r$ and $F = H_2(g^r) \oplus (m\|\omega)$.
4. Compute $s = u + r \cdot H_3(D, E, F) \mod q$.
5. Output the ciphertext $\mathfrak{C} = (D, E, F, s)$.

$\mathsf{ReEncrypt}(\mathtt{rk}_{i\to j}, \mathfrak{C}_i, \mathtt{pk}_i, \mathtt{pk}_j)$**:** On input a re-encryption (user i to user j) key $\mathtt{rk}_{i\to j} = (\mathtt{rk}^{\langle 1\rangle}_{i\to j}, V, W)$, an original ciphertext $\mathfrak{C}_i = (D, E, F, s)$ under public key $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2})$, this algorithm re-encrypts $\mathfrak{C}_i$ into another one under public key $\mathtt{pk}_j = (\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2})$ as follows:

1. If $\left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})}\mathtt{pk}_{i,2}\right)^s = D \cdot E^{H_3(D,E,F)}$ does not hold, return $\perp$.
2. Otherwise, compute $E' = E^{\mathtt{rk}^{\langle 1\rangle}_{i\to j}}$, and output (E', F, V, W).

Let $r = H_1(m, \omega)$, $v = H_1(h, \pi)$, the transformed ciphertext is of the following forms:

$$\mathfrak{C}_j = (E', F, V, W) = \left(g^{r\cdot h}, H_2(g^r) \oplus (m\|\omega), \mathtt{pk}_{j,2}^v, H_2(g^v) \oplus (h\|\pi)\right).$$

$\mathsf{Encrypt}_1(\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}), m)$**:** To create a nontransformable ciphertext under public key $\mathtt{pk}_i$ of a message $m \in \mathcal{M}$:

1. Pick $h \xleftarrow{\$} \{0,1\}^{\ell_0}$ and $\pi \xleftarrow{\$} \{0,1\}^{\ell_1}$, compute $v = H_1(h, \pi)$.
2. Compute $V = \mathtt{pk}_{j,2}^v$ and $W = H_2(g^v) \oplus (h\|\pi)$.
3. Pick $\omega \xleftarrow{\$} \{0,1\}^{\ell_1}$, compute $r = H_1(m, \omega)$.
4. Output the ciphertext $\mathfrak{C} = (E', F, V, W)$.

$\mathsf{Decrypt}(\mathtt{sk}_i, \mathfrak{C}_i)$**:** On input a private key $\mathtt{sk}_i = (x_{i,1}, x_{i,2})$ and ciphertext $\mathfrak{C}_i$, parse $\mathfrak{C}_i$, then work according to two cases:

- $\mathfrak{C}$ is an original ciphertext in the form $\mathfrak{C} = (D, E, F, s)$:
  1. If $\left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})}\mathtt{pk}_{i,2}\right)^s = D \cdot E^{H_3(D,E,F)}$ does not hold, return $\perp$.
  2. Otherwise, compute $(m\|\omega) = F \oplus H_2(E^{\frac{1}{x_{i,1}H_4(\mathtt{pk}_{i,2})+x_{i,2}}})$.
  3. Return m if $E = \left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})}\mathtt{pk}_{i,2}\right)^{H_1(m,\omega)}$ holds; else return $\perp$.
- $\mathfrak{C}$ is a transformed ciphertext in the form $\mathfrak{C} = (E', F, V, W)$:
  1. Compute $(h\|\pi) = W \oplus H_2(V^{1/\mathtt{sk}_{i,2}})$ and $(m\|\omega) = F \oplus H_2(E'^{1/h})$.
  2. Return m if $V = \mathtt{pk}_{i,2}^{H_1(h,\pi)}$ and $E' = g^{H_1(m,\omega)\cdot h}$ hold; else $\perp$.

4.2 Security Analysis

The intuition of CCA security can be seen from the below properties.

1. The validity of the original ciphertexts can be *publicly verifiable* by everyone including the proxy; otherwise, it will suffer from an attack as illustrated in [DWLC08]. For our scheme, the ciphertext component (D, s) in the original ciphertext (D, E, F, s) can be viewed as a signature signing the "message" (E, F), that is how we get pubic verifiability.

2. The original ciphertexts should be CCA-secure. The original ciphertext produced by our scheme is indeed a "hashed" CCA-secure ElGamal encryption tightly integrated with a Schnorr signature.
3. The transformed ciphertexts should be CCA-secure In our scheme, a transformed ciphertext can be viewed as two seamlessly integrated "hashed" CCA-secure ElGamal encryptions.

We make four observations on the re-encryption key computation.

1. It takes the input of $\mathtt{sk}_i$, but not $\mathtt{sk}_j$, so our scheme is unidirectional.
2. Even though h can be recovered by anyone who owns $\mathtt{sk}_j$, $\mathtt{rk}^{\langle 1 \rangle}_{i \to j}$ only gives information about $x_{i,1} H_4(\mathtt{pk}_{i,2}) + x_{i,2}$ (no matter whom the delegatee j is), but not the concrete value of $x_{i,1}$ or $x_{i,2}$. This gives an intuition why our scheme achieves delegator secret security.
3. A collusion of the delegatee and the proxy cannot recover $x_{i,1}$, which is needed to decrypt original ciphertexts.
4. If the delegatee j is now a delegator to someone else (say k). Again, only $x_{j,1} H_4(\mathtt{pk}_{j,2}) + x_{j,2}$ is known to a collusion of the delegatee k and a proxy, which is not useful in recovering the token h in $\mathtt{rk}_{i \to j}$, hence the chain collusion attack suffered by [GA07,CT07] does not apply.

Theorem 1. *Our scheme is IND-PRE-CCA secure in the random oracle model, if the CDH assumption holds in group $\mathbb{G}$ and the Schnorr signature [Sch91] is existentially unforgeable against chosen message attack.*

The detailed proof can be found in the appendix. The proof first uses Coron's technique [Cor00] to implant our hard problem to many uncorrupted public keys. At the same time, for those uncorrupted public keys which is generated as usual (without the problem embedded), re-encryption key can still be generated with non-negligible probability.

To prove the original ciphertext security is relatively simple. For transformed ciphertext, an implicitly defined random h value which is unknown to the simulator may be used in the re-encryption key returned as the response to the oracles query. To answer decryption oracle queries, the simulator can extract the random h value used from the random oracle and unwrap the given ciphertext. For the challenge ciphertext generation, our definition of security rules out the case that both the delegator and the proxy are compromised, so any partial information regarding the value of h used in the re-encryption key would not affect the (different) h value associated with the challenge ciphertext.

For nontransformable ciphertext security, the situation is much simpler. The h value used in the challenge ciphertext is essentially a one-time pad, and the reduction boils down to the underlying hashed ElGamal encryption, so the simulator can compute all the re-encryption keys.

4.3 Efficiency Comparisons

In Table 2, we compare our scheme with $\mathcal{SC09}$ [SC09] with our suggested fix. We use $t_{\exp}$ to denote the computational cost of an exponentiation. In our calculation, a multi-exponentiation (m-exp) (which we assume it multiplies only up to 3 exponentiations in one shot) is considered as $1.5t_{\exp}$. $\mathsf{Encrypt}$ of $\mathcal{LV08}$, $\mathsf{ReEncrypt}$ and $\mathsf{Decrypt}(\mathfrak{C})$ of $\mathcal{SC09}$ used 1, 2 and 2 m-exp respectively. In our scheme, we assume $\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})} \mathtt{pk}_{i,2}$ is pre-computed. Even not, it only adds at most $1t_{\exp}$ in $\mathsf{Encrypt}$, $\mathsf{ReEncrypt}$ and $\mathsf{Decrypt}(\mathfrak{C})$ using m-exp, since there are other exponentiations to be done. The comparison indicates that our scheme beats $\mathcal{SC09}$ in all aspects.

Schemes	$\mathcal{SC}09$ [SC09]	Our Scheme
Encrypt	$5t_{\exp}$ (in $\mathbb{Z}_{N^2}$)	$3t_{\exp}$ (in $\mathbb{G}$)
ReEncrypt	$4t_{\exp}$ (in $\mathbb{Z}_{N^2}$)	$2.5t_{\exp}$ (in $\mathbb{G}$)
Decrypt($\mathfrak{C}$)	$5t_{\exp}$ (in $\mathbb{Z}_{N^2}$)	$3.5t_{\exp}$ (in $\mathbb{G}$)
Decrypt($\mathfrak{C}'$)	$5t_{\exp}$ (in $\mathbb{Z}_{N^2}$)	$4t_{\exp}$ (in $\mathbb{G}$)
$\lvert\mathfrak{C}\rvert$	$2k + 3\lvert(N_X)^2\rvert + \lvert m\rvert$	$3\lvert\mathbb{G}\rvert+\lvert\mathbb{Z}_q\rvert$
$\lvert\mathfrak{C}'\rvert$	$\ell_1 + 3\lvert(N_X)^2\rvert + 2\lvert(N_Y)^2\rvert + \lvert m\rvert$	$2\lvert\mathbb{G}\rvert + 2\lvert\mathbb{Z}_q\rvert$
Security	Not Collusion-Resistant	CCA-Secure
Assumption	DDH over $\mathbb{Z}_{N^2}$	CDH over $\mathbb{G}$
RO-Free	$\times$	$\times$
Nature of Decrypt	Decryption of $\mathfrak{C}'$ requires $\mathtt{pk}_X$ of the delegator	No delegator public key is required

Table 2. Comparisons of Unidirectional Proxy Re-Encryption Schemes. $\mathfrak{C}$ denotes an original ciphertext and $\mathfrak{C}'$ denotes a transformed ciphertext, $|\mathfrak{C}|$ and $|\mathfrak{C}'|$ are their size. N_X (N_Y) is the safe-prime modulus used by the delegator (delegatee).

5 Conclusions

Most existing unidirectional proxy re-encryption (PRE) schemes rely on pairing except a recently proposed scheme by Shao and Cao [SC09]. However, we showed that their CCA-security proof in the random oracle model is flawed, and presented a concrete attack. Possible fixes of their scheme further degrades either the decryption efficiency or the transformed ciphertext length. We then presented a natural construction of CCA-secure unidirectional PRE scheme without pairings that is very efficient.

Our scheme is single-hop and relies on the random oracle. It would be interesting to construct a multi-hop scheme in the standard model. It seems to be possible to use the token-controlled encryption approach to build a multi-hop scheme; however, the design may be inelegant and the efficiency may not be ideal. We remark that our scheme is proven under a relaxed security definition. We left it as an open problem to devise a pairing-free CCA-secure scheme without this relaxation. Another interesting problem, which possibly requires a different set of techniques, is to construct other schemes in proxy re-cryptography, such as conditional PRE schemes [CWC+09] and proxy re-signatures [CP08,LV08a], without pairings.

References

[ABH09] Giuseppe Ateniese, Karyn Benson, and Susan Hohenberger. Key-Private Proxy Re-encryption. In *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 279–294. Springer, 2009.

[AFGH06] 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.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible Protocols and Atomic Proxy Cryptography. In *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 1998.

[BCP03] Emmanuel Bresson, Dario Catalano, and David Pointcheval. A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications. In *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2003.

[BDZ03] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of Diffie-Hellman Problem. In *ICICS*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2003.

[BSNS05] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Certificateless Public Key Encryption Without Pairing. In *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2005.

[CH07] Ran Canetti and Susan Hohenberger. Chosen-Ciphertext Secure Proxy Re-Encryption. In *ACM Conference on Computer and Communications Security*, pages 185–194. ACM, 2007.

[CKN03] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing Chosen-Ciphertext Security. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2003.

[Cor00] Jean-Sébastien Coron. On the Exact Security of Full Domain Hash. In *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 2000.

[CP08] Sherman S.M. Chow and Raphael C.-W. Phan. Proxy Re-signatures in the Standard Model. In *ISC*, volume 5222 of *Lecture Notes in Computer Science*, pages 260–276. Springer, 2008.

[CT07] Cheng-Kang Chu and Wen-Guey Tzeng. Identity-Based Proxy Re-encryption Without Random Oracles. In *ISC*, volume 4779 of *Lecture Notes in Computer Science*, pages 189–202. Springer, 2007.

[CWC+09] Cheng-Kang Chu, Jian Weng, Sherman S.M. Chow, Jianying Zhou, and Robert H. Deng. Conditional Proxy Broadcast Re-Encryption. In *ACISP*, volume 5594 of *Lecture Notes in Computer Science*, pages 327–342. Springer, 2009.

[DWLC08] Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen. Chosen-Ciphertext Secure Proxy Re-encryption without Pairings. In *CANS*, volume 5339 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[GA07] Matthew Green and Giuseppe Ateniese. Identity-Based Proxy Re-encryption. In *ACNS*, volume 4521 of *Lecture Notes in Computer Science*, pages 288–306. Springer, 2007.

[Gam84] Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO*, pages 10–18, 1984.

[GJJS04] Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. Universal Re-encryption for Mixnets. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2004.

[HRSV07] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely Obfuscating Re-encryption. In *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 233–252. Springer, 2007.

[ID03] Anca-Andreea Ivan and Yevgeniy Dodis. Proxy Cryptography Revisited. In *NDSS*. The Internet Society, 2003.

[Jak99] Markus Jakobsson. On Quorum Controlled Asymmetric Proxy Re-encryption. In *Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 1999.

[LV08a] Benoît Libert and Damien Vergnaud. Multi-use Unidirectional Proxy Re-Signatures. In *ACM Conference on Computer and Communications Security*, pages 511–520. ACM, 2008.

[LV08b] Benoît Libert and Damien Vergnaud. Tracing Malicious Proxies in Proxy Re-encryption. In *Pairing*, volume 5209 of *Lecture Notes in Computer Science*, pages 332–353. Springer, 2008.

[LV08c] Benoît Libert and Damien Vergnaud. Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption. In *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2008.

[MO97] Masahiro Mambo and Eiji Okamoto. Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E80-A(1):54–63, 1997.

[SC09] Jun Shao and Zhenfu Cao. CCA-Secure Proxy Re-encryption without Pairings. In *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 357–376. Springer, 2009.

[Sch91] Claus-Peter Schnorr. Efficient Signature Generation by Smart Cards. *J. Cryptology*, 4(3):161–174, 1991.

[Smi05] Tony Smith. DVD Jon: Buy DRM-less Tracks from Apple iTunes. Available online at http://www.theregister.co.uk/2005/03/18/itunes_pymusique, January 2005.

A Proof of Chosen-Ciphertext Security

A.1 Complexity Assumptions

Definition 7 (Computational Diffie-Hellman (CDH) Problem). *Let $\mathbb{G}$ be a cyclic multiplicative group with prime order q. The CDH problem in $\mathbb{G}$ is, given $(g,\, g^a,\, g^b) \in \mathbb{G}^3$ with $a, b \xleftarrow{\$} \mathbb{Z}_q^*$, to compute g^{ab}.*

Definition 8 (CDH Assumption). *For an algorithm $\mathcal{B}$, its advantage in solving the CDH problem is defined as* $\mathrm{Adv}_{\mathcal{B}}^{\mathrm{CDH}} \triangleq \Pr\left[\mathcal{B}(g, g^a, g^b) = g^{ab}\right]$, *where the probability is taken over the random choices of a, b and those made by $\mathcal{B}$. We say that the (t, ϵ)-CDH assumption holds in $\mathbb{G}$ if no t-time algorithm $\mathcal{B}$ has advantage at least ϵ in solving the CDH problem in $\mathbb{G}$.*

We show our reduction to an equivalent problem for higher readability, which is the divisible computation Diffie-Hellman (DCDH) problem introduced by Bao *et al.* [BDZ03]. The task is to compute $g^{b/a}$ given $(g, g^a, g^b) \in \mathbb{G}^3$ with unknown $a, b \xleftarrow{\$} \mathbb{Z}_q^*$. It is shown in [BDZ03] that the DCDH and CDH are equivalent in the same group.

A.2 Preliminaries for the Proofs

Given an adversary $\mathcal{A}$, who asks at most q_{H_i} random oracle quires to H_i with $i \in \{1, 2, 3\}$, and breaks the $(t, n_u, n_c, q_{rk}, q_{re}, q_d, \epsilon)$-IND-PRE-CCA security of our scheme, we will show how to construct a polynomial time algorithm $\mathcal{B}$ which can break the CDH assumption in $\mathbb{G}$ or the existential unforgeability against chosen message attack (EUF-CMA) of the Schnorr signature with non-negligible advantage. For a cleaner proof, we assume that Schnorr signature is EUF-CMA secure.

Adversary $\mathcal{A}$ can choose to either attack the original ciphertext security (denoted by $\mathcal{A}_{\mathrm{orig}}$), the transformed ciphertext security (denoted by $\mathcal{A}_{\mathrm{tran}}$) or the nontransformable ciphertext security (denoted by $\mathcal{A}_{\mathrm{notr}}$). The proofs for security against $\mathcal{A}_{\mathrm{orig}}$ and $\mathcal{A}_{\mathrm{tran}}$ share many similarities, and the former may be a bit simpler. The proof for security against $\mathcal{A}_{\mathrm{notr}}$ is the simplest among all three. The corresponding reduction algorithms are $\mathcal{B}_{\mathrm{orig}}$, $\mathcal{B}_{\mathrm{tran}}$, and $\mathcal{B}_{\mathrm{notr}}$. For brevity, we do not repeat parts of the simulations which are the same, but for these parts we will refer the reduction algorithm by $\mathcal{B}$ ($\in \{\mathcal{B}_{\mathrm{orig}}, \mathcal{B}_{\mathrm{tran}}, \mathcal{B}_{\mathrm{notr}}\}$) to avoid confusion.

Our proofs are given in the random oracle model, so we first describe how $\mathcal{B}$ simulates the random oracles. Algorithm $\mathcal{B}$ gives $(q, \mathbb{G}, g, H_1, \ldots, H_4, \ell_0, \ell_1)$ to $\mathcal{A}$. Here H_1, H_2 and H_3 are random oracles controlled by $\mathcal{B}$. $\mathcal{B}$ maintains four hash lists H_i^{list} with $i \in \{1, 2, 3\}$, which are initially empty, and responds the random oracles queries for $\mathcal{A}$ as shown in Figure 1.

- $H_1(m, \omega)$: If this query has appeared on the H_1^{list} in a tuple (m, ω, r), return the predefined value r. Otherwise, choose $r \xleftarrow{\$} \mathbb{Z}_q^*$, add the tuple (m, ω, r) to the list H_1^{list} and respond with $H_1(m, \omega) = r$.
- $H_2(R)$: If this query has appeared on the H_2^{list} in a tuple (R, β), return the predefined value β. Otherwise, choose $\beta \xleftarrow{\$} \{0, 1\}^{\ell_0 + \ell_1}$, add the tuple (R, β) to the list H_2^{list} and respond with $H_2(R) = \beta$.
- $H_3(D, E, F)$: If this query has appeared on the H_3^{list} in a tuple (D, E, F, γ), return the predefined value γ. Otherwise, choose $\gamma \xleftarrow{\$} \mathbb{Z}_q^*$, add the tuple (D, E, F, γ) to the list H_3^{list} and respond with $H_3(D, E, F) = \gamma$.

Fig. 1. Simulations for H_i for $i = 1, 2, 3$

$\mathcal{B}$ maintains two lists K^{list} and R^{list} which are initially empty, which stores the list of public/private key pairs and re-encryption key generated respectively.

A.3 Original Ciphertext Security

Key generations. $\mathcal{B}_{\mathrm{orig}}$ generates the uncorrupted-keys and corrupted-keys as follows.

– *Uncorrupted-key generation.* $\mathcal{B}_{\text{orig}}$ picks $x_{i,1} \xleftarrow{\$} \mathbb{Z}_q^*, x_{i,2} \xleftarrow{\$} \mathbb{Z}_q^*$. and uses Coron's technique [Cor00] – flips a biased coin $c_i \in \{0,1\}$ that yields 1 with probability θ and 0 otherwise.
  - If $c_i = 1$, it defines $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$;
  - If $c_i = 0$, it defines $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}) = ((g^a)^{x_{i,1}}, (g^a)^{x_{i,2}})$.

  $\mathcal{B}_{\text{orig}}$ adds the tuple $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ to K^{list} and returns $\mathtt{pk}_i$ to $\mathcal{A}$.

– *Corrupted-key generation.* $\mathcal{B}$ picks $x_{j,1}, x_{j,2} \xleftarrow{\$} \mathbb{Z}_q^*$, and defines $\mathtt{pk}_j = (g^{x_{j,1}}, g^{x_{j,2}})$, $c_j =$ '$-$'. It then adds the tuple $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ to K^{list} and returns $(\mathtt{pk}_j, (x_{j,1}, x_{j,2}))$ to $\mathcal{A}$.

Phase 1. Adversary $\mathcal{A}$ issues a series of queries which $\mathcal{B}$ answers $\mathcal{A}$ as follows:

– $\mathsf{OReK}(\mathtt{pk}_i, \mathtt{pk}_j)$: If R^{list} has an entry for $(\mathtt{pk}_i, \mathtt{pk}_j)$, return the predefined re-encryption key to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ acts as follows:
  1. Recover tuples $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ and $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ from K^{list}.
  2. Pick $h \xleftarrow{\$} \{0,1\}^{\ell_0}$, $\pi \xleftarrow{\$} \{0,1\}^{\ell_1}$. Compute $v = H_1(h, \pi)$
  3. Compute $V = \mathtt{pk}_{j,2}^v$, $W = H_2(g^v) \oplus (h \| \pi)$.
     (The above two steps are exactly the same as those in $\mathsf{ReKeyGen}$ algorithm.)
  4. Construct the first component $\mathtt{rk}_{i \to j}^{\langle 1 \rangle}$ according to the following cases:
     - $c_i = 1$ or $c_i =$ '$-$': define $\mathtt{rk}_{i \to j}^{\langle 1 \rangle} = \frac{h}{x_{i,1} H_4(\mathtt{pk}_{i,2}) + x_{i,2}}$, and define $\tau = 1$.
     - $(c_i = 0 \ \wedge \ c_j = 1)$ or $(c_i = 0 \ \wedge \ c_j = 0)$: pick $\mathtt{rk}_{i \to j}^{\langle 1 \rangle} \xleftarrow{\$} \mathbb{Z}_q^*$, and define $\tau = 0$.
     - $(c_i = 0 \wedge c_j =$ '$-$'$)$: output "$\mathsf{failure}$" and **aborts**.

     For $c_i = 1$ or $c_i =$ '$-$', $\mathtt{rk}_{i \to j}$ is obviously correct due to $\mathtt{sk}_i = (x_{i,1}, x_{i,2})$.
     For the case $(c_i = 0 \wedge c_j =$ '$-$'$)$, we defer the probability analysis to later part.
     For the cases where $(c_i = 0 \wedge c_j \neq$ '$-$'$)$, using a random $\mathtt{rk}_{i \to j}$ would not match with the value of h associated with (V, W). For this, we will rely on the security of "hashed" ElGamal encryption scheme [Gam84,FO99,BSNS05].
  5. If $\mathcal{B}$ does not **abort**, add $(\mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{rk}_{i \to j}^{\langle 1 \rangle}, V, W), h, \tau)$ into list R^{list}.
  6. Return $\mathtt{rk}_{i \to j} = (\mathtt{rk}_{i \to j}^{\langle 1 \rangle}, V, W)$ to $\mathcal{A}$.

– $\mathsf{OReE}(\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}), \mathtt{pk}_j = (\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2}), \mathfrak{C}_i = (D, E, F, s))$:
  1. If $\left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})} \mathtt{pk}_{i,2}\right)^s \neq D \cdot E^{H_3(D,E,F)}$, return $\perp$ since $\mathfrak{C}_i$ is invalid.
  2. Recover tuples $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ and $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ from K^{list}.
  3. If $(c_i = 0 \ \wedge \ c_j =$ '$-$'$)$ does not hold, issue a re-encryption key generation query $\langle \mathtt{pk}_i, \mathtt{pk}_j \rangle$ to obtain $\mathtt{rk}_{i \to j}$, and then return $\mathsf{ReEncrypt}(\mathtt{rk}_{i \to j}, \mathfrak{C}_i, \mathtt{pk}_i, \mathtt{pk}_j)$ to $\mathcal{A}$.
  4. Else, search for the tuple $(m, \omega, r) \in H_1^{\text{list}}$ such that $(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})} \mathtt{pk}_{i,2})^r = E$. If there exists no such tuple, return $\perp$. (This corresponds to the event REErr to be explained).
  5. Retrieve $(\mathtt{pk}_i, \mathtt{pk}_j, (*, V, W), h,$ '$-$'$)$ from list R^{list}, define $E' = g^{r \cdot h}$.
  6. If it is not found, we prepare a "partial" re-encryption key as follows.
  7. Pick $h \xleftarrow{\$} \{0,1\}^{\ell_0}, \pi \xleftarrow{\$} \{0,1\}^{\ell_1}$. Compute $v = H_1(h, \pi)$.
  8. Compute $V = \mathtt{pk}_{j,2}^v$, $W = H_2(g^v) \oplus (h \| \pi)$.
  9. Store $(\mathtt{pk}_i, \mathtt{pk}_j, (\perp, V, W), h,$ '$-$'$)$ into list R^{list}, define $E' = g^{r \cdot h}$.
  10. E' is consistently computed as long as r can be retrieved, return (E', F, V, W) to $\mathcal{A}$

– $\mathsf{ODec}(\mathtt{pk}_i, \mathfrak{C}_i)$: $\mathcal{B}$ first parses $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2})$ and recovers tuple $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ from K^{list}. If $c_i = 1$ or $c_i =$ '$-$', algorithm $\mathcal{B}$ runs $\mathsf{Decrypt}((x_{i,1}, x_{i,2}), \mathfrak{C}_i)$ and returns the result to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ works according to the following two cases:

- $\mathfrak{C}_i$ is an original ciphertext $\mathfrak{C}_i = (D, E, F, s)$: If $\left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})}\mathtt{pk}_{i,2}\right)^s \neq D \cdot E^{H_3(D,E,F)}$, return $\perp$ to $\mathcal{A}$ indicating that $\mathfrak{C}_i$ is an invalid ciphertext. Otherwise, search lists H_1^{list} and H_2^{list} to see whether there exists $(m, \omega, r) \in H_1^{\mathrm{list}}$ and $(R, \beta) \in H_2^{\mathrm{list}}$ such that
$$\left(\mathtt{pk}_{i,1}^{H_4(\mathtt{pk}_{i,2})}\mathtt{pk}_{i,2}\right)^r = E, \beta \oplus (m\|\omega) = F \quad \text{and} \quad R = g^r.$$
If yes, return m to $\mathcal{A}$. Otherwise, return $\perp$.
- $\mathfrak{C}_i$ is a transformed ciphertext $\mathfrak{C}_i = (E', F, V, W)$: $\mathcal{B}$ decrypts according to two cases:
  * If there exists a tuple $(\mathtt{pk}_j, \mathtt{pk}_i, (\mathtt{rk}^{\langle 1\rangle}, V, W), h, 0)$ in R^{list}: Compute $E = E'^{\frac{1}{\mathtt{rk}^{\langle 1\rangle}}}$. Search to see whether there exists $(m, \omega, r) \in H_1^{\mathrm{list}}$ and $(R, \beta) \in H_2^{\mathrm{list}}$ such that
$$\left(\mathtt{pk}_{j,1}^{H_4(\mathtt{pk}_{j,2})}\mathtt{pk}_{j,2}\right)^r = E, \quad \beta \oplus (m\|\omega) = F, \quad R = g^r.$$
If yes, return m to $\mathcal{A}$, else return $\perp$.
Note that all V, W values from R^{list} are correctly generated.
  * Else, search for $(m, \omega, r), (h, \pi, v) \in H_1^{\mathrm{list}}$ and $(R, \beta), (R', \beta') \in H_2^{\mathrm{list}}$ such that
$$\mathtt{pk}_{i,2}^v = V, \beta' \oplus (h\|\pi) = W, g^{r\cdot h} = E', \beta \oplus (m\|\omega) = F, R = g^r \text{ and } R' = g^v.$$
If yes, return m to $\mathcal{A}$, else return $\perp$.

Challenge. When $\mathcal{A}$ decides that Phase 1 is over, it outputs a public key $\mathtt{pk}_{i^*} = (\mathtt{pk}_{i^*,1}, \mathtt{pk}_{i^*,2})$ and two equal-length messages $m_0, m_1 \in \{0,1\}^{\ell_0}$. Algorithm $\mathcal{B}$ recovers tuple $(\mathtt{pk}_{i^*}, x_{i^*,1}, x_{i^*,2}, c^*)$ from K^{list}. According to the constraints described in IND-PRE-CCA game, c^* must be equal to 1 or 0. $\mathcal{B}_{\mathrm{orig}}$ picks $\delta \xleftarrow{\$} \{0,1\}$ and simulates the challenge ciphertext as follows.

1. If $c^* = 1$, $\mathcal{B}_{\mathrm{orig}}$ outputs "failure" and **aborts**.
2. Compute $E^* = \left(g^b\right)^{x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2}}$.
3. Pick $e^*, s^* \xleftarrow{\$} \mathbb{Z}_q^*$, and compute $D^* = \left(g^b\right)^{-(x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2})e^*} \left(g^{\frac{1}{a}}\right)^{(x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2})s^*}$.
4. Pick $F^* \xleftarrow{\$} \{0,1\}^{\ell_0+\ell_1}$ and define $H_3(D^*, E^*, F^*) = e^*$.
5. Pick $\omega^* \xleftarrow{\$} \{0,1\}^{\ell_1}$, and *implicitly* define $H_1(m_\delta, \omega^*) = ab$ and $H_2(g^{ab}) = (m_\delta\|\omega^*) \oplus F^*$.
6. Return $\mathfrak{C}^* = (D^*, E^*, F^*, s^*)$ as the challenge original ciphertext to adversary $\mathcal{A}_{\mathrm{orig}}$.

Observe that the challenge ciphertext $\mathfrak{C}^*$ is identically distributed as the real one from the construction. To see this, letting $u^* \triangleq s^* - abe^*$ and $r^* \triangleq ab$, we have

$$\begin{aligned}
D^* &= \left(g^b\right)^{-(x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2})e^*} \left(g^{\frac{1}{a}}\right)^{(x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2})s^*} \\
&= \left(\left(g^{\frac{1}{a}}\right)^{x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2}}\right)^{s^* - abe^*} = \left(g^{\frac{1}{a}\cdot x_{i^*,1}H_4(\mathtt{pk}_{i^*,2})} g^{\frac{1}{a}\cdot x_{i^*,2}}\right)^{s^* - abe^*} \\
&= \left(\mathtt{pk}_{i^*,1}^{H_4(\mathtt{pk}_{i^*,2})}\mathtt{pk}_{i^*,2}\right)^{u^*}, \\
E^* &= \left(g^b\right)^{x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2}} = \left(\left(g^{\frac{1}{a}}\right)^{x_{i^*,1}H_4(\mathtt{pk}_{i^*,2}) + x_{i^*,2}}\right)^{ab} = \left(\mathtt{pk}_{i^*,1}^{H_4(\mathtt{pk}_{i^*,2})}\mathtt{pk}_{i^*,2}\right)^{r^*}, \\
F^* &= H_2(g^{ab}) \oplus (m_\delta\|\omega^*) = H_2(g^{r^*}) \oplus (m_\delta\|\omega^*), \\
s^* &= (s^* - abe^*) + abe^* = u^* + ab \cdot H_3(D^*, E^*, F^*) = u^* + r^* \cdot H_3(D^*, E^*, F^*).
\end{aligned}$$

Phase 2. Adversary $\mathcal{A}$ continues to issue queries as in Phase 1, with the restrictions described in the IND-PRE-CCA game. Algorithm $\mathcal{B}$ responds to these queries for $\mathcal{A}$ as in Phase 1.

Guess. Eventually, adversary $\mathcal{A}$ returns a guess $\delta' \in \{0,1\}$ to $\mathcal{B}_{\text{orig}}$. Algorithm $\mathcal{B}_{\text{orig}}$ randomly picks a tuple (R, β) from the list H_2^{list}. and outputs R as the solution to the given DCDH instance.

This completes the description of the simulation. It remains to related the probability for success and the execution time, which will be shown in Lemma 1 and Lemma 2. □

A.4 Transformed Ciphertext Security

Key generations. $\mathcal{B}_{\text{tran}}$ generates the uncorrupted-keys and corrupted-keys as follows.

– *Uncorrupted-key generation.* $\mathcal{B}_{\text{tran}}$ firstly picks $x_{i,1}, x_{i,2} \xleftarrow{\$} \mathbb{Z}_q^*$.
  $\mathcal{B}_{\text{tran}}$ flips a biased coin $c_i \in \{0,1\}$ that yields 1 with probability θ and 0 otherwise.
  If $c_i = 1$, defines $\mathtt{pk}_{i,2} = g^{x_{i,2}}/g^a$, $\mathtt{pk}_{i,1} = (g^a)^{1/H_4(\mathtt{pk}_{i,2})} \cdot g^{x_{i,1}}$ (different from $\mathcal{B}_{\text{orig}}$);
  If $c_i = 0$, defines $\mathtt{pk}_{i,2} = (g^a)^{x_{i,2}}$, $\mathtt{pk}_{i,1} = (g^a)^{x_{i,1}}$.
  $\mathcal{B}_{\text{tran}}$ adds the tuple $((\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}), x_{i,1}, x_{i,2}, c_i)$ to K^{list} and returns $(\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2})$ to $\mathcal{A}$.
– *Corrupted-key generation.* Same as $\mathcal{B}_{\text{orig}}$.

Phase 1. Adversary $\mathcal{A}$ issues a series of queries which $\mathcal{B}$ answers $\mathcal{A}$ as follows:

– $\mathsf{OReK}(\mathtt{pk}_i, \mathtt{pk}_j)$: If R^{list} has an entry for $(\mathtt{pk}_i, \mathtt{pk}_j)$, return the predefined re-encryption key to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ acts as follows:
  1. Recover tuples $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ and $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ from K^{list}.
  2. Construct the first component $\mathtt{rk}_{i\to j}^{\langle 1\rangle}$ according to the following cases:
     - $c_i = 1$ or $c_i =$ '−':
       (a) Define $\mathtt{rk}_{i\to j}^{\langle 1\rangle} = \frac{h}{x_{i,1}H_4(\mathtt{pk}_{i,2}) + x_{i,2}}$, and define $\tau = 1$.
       (b) Pick $h \xleftarrow{\$} \{0,1\}^{\ell_0}$, $\pi \xleftarrow{\$} \{0,1\}^{\ell_1}$. Compute $v = H_1(h, \pi)$
       (c) Compute $V = \mathtt{pk}_{j,2}^v$, $W = H_2(g^v) \oplus (h\|\pi)$.
       (The above two steps are exactly the same as those in $\mathsf{ReKeyGen}$ algorithm.)
       For $c_i =$ '−', $\mathtt{rk}_{i\to j}$ is obviously correct due to $\mathtt{sk}_i = (x_{i,1}, x_{i,2})$.
       For $c_i = 1$ $(\frac{a}{H_4(\mathtt{pk}_{i,2})} + x_{i,1})H_4(\mathtt{pk}_{i,2}) + (-a + x_{i,2}) = x_{i,1}H_4(\mathtt{pk}_{i,2}) + x_{i,2}$. Looking ahead, $c_i = 0$ is where we plug the hard problem instance and pray that the adversary will choose it as the target in challenge phase
     - $c_i = 0 \ \wedge \ c_j = 0$:
       (a) Pick $\mathtt{rk}_{i\to j}^{\langle 1\rangle} \xleftarrow{\$} \mathbb{Z}_q^*$, and define $\tau = 0$.
       (b) Pick $z \xleftarrow{\$} \mathbb{Z}_q^*$, set $V = (g^b)^z$, which defines $bz = ax_{j,2}v$, i.e., $g^v = g^{bz/(ax_{j,2})}$.
       (c) Pick $W \xleftarrow{\$} \{0,1\}^{\ell_0+\ell_1}$, *implicitly* define $H_2((g^{b/a})^{z/x_{j,2}}) = (h\|\pi) \oplus W$, h, π.
       (d) Store $(\mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{rk}_{i\to j}, V, W), \perp, 0, z)$ into list R^{list}.
       Using a random $\mathtt{rk}_{i\to j}$ would not match with the value of h associated with (V, W). For this, we will rely on the security of "hashed" ElGamal encryption scheme [Gam84,FO99,BSNS05].
     - $(c_i = 0 \ \wedge \ c_j = 1)$ or $(c_i = 0 \wedge c_j =$ '−'$)$: output "$\mathsf{failure}$" and **aborts**. We defer the probability analysis to later part.

3. If $\mathcal{B}$ does not **abort**, add $(\mathtt{pk}_i, \mathtt{pk}_j, (\mathtt{rk}^{\langle 1\rangle}_{i\to j}, V, W), h, \tau)$ into list R^{list}.
4. Return $\mathtt{rk}_{i\to j} = (\mathtt{rk}^{\langle 1\rangle}_{i\to j}, V, W)$ to $\mathcal{A}$.

– OReE, ODec: Same as $\mathcal{B}_{\text{orig}}$.

Challenge. When $\mathcal{A}$ decides that Phase 1 is over, it outputs a delegator's public key $\mathtt{pk}_{i'} = (\mathtt{pk}_{i',1}, \mathtt{pk}_{i',2})$, a delegatee's public key $\mathtt{pk}_{i^*} = (\mathtt{pk}_{i^*,1}, \mathtt{pk}_{i^*,2})$ and two equal-length messages $m_0, m_1 \in \{0,1\}^{\ell_0}$. Algorithm $\mathcal{B}$ recovers tuples $(\mathtt{pk}_{i^*}, x_{i^*,1}, x_{i^*,2}, c^*)$ and tuple $(\mathtt{pk}_{i'}, x_{i',1}, x_{i',2}, c')$ from K^{list}. According to the constraints described in IND-PRE-CCA game, c^* must be equal to 1 or 0. $\mathcal{B}$ simulates the challenge ciphertext as follows.

1. If $c' = 1$ or $c^* = 1$, $\mathcal{B}_{\text{tran}}$ outputs "failure" and **aborts**.
2. If $c' =$ '$-$', in our security model $\mathcal{A}_{\text{tran}}$ would not get $\mathtt{rk}_{i'\to i^*}$ which makes the simulation of ciphertext a special case of the simulation below.
3. Pick $t \xleftarrow{\$} \mathbb{Z}_q^*$, define $E'^* = (g^b)^t$, which implicitly defines $r^* h^* = bt$, i.e., $r^* = bt/h^*$.
4. Pick $F^* \xleftarrow{\$} \{0,1\}^{\ell_0+\ell_1}$, *implicitly* define $F^* = H_2((g^{b/a})^{t/\mathtt{rk}_{i'\to i^*}(x_{i',1}H_4(\mathtt{pk}_{i',2})+x_{i',2})}) \oplus (m_\delta \| \omega^*)$. Recall that $h^* = \mathtt{rk}_{i'\to i^*} a(x_{i',1}H_4(\mathtt{pk}_{i',2}) + x_{i',2})$ for $c' = 0$, which implicitly defines $H_1(m_\delta, \omega^*) = r^* = (b/a)(t/\mathtt{rk}_{i'\to i^*}(x_{i',1}H_4(\mathtt{pk}_{i',2}) + x_{i',2}))$. This explains F^*.
5. Retrieve $(\mathtt{pk}_{i'}, \mathtt{pk}_{i^*}, (\mathtt{rk}_{i'\to i^*}, V^*, W^*), \perp, 0, z^*)$ from list R^{list}. If not found, do the following to define V^*, W^* and z^* (and store them into list R^{list} afterwards).
6. Pick $z^* \xleftarrow{\$} \mathbb{Z}_q^*$, set $V^* = (g^b)^{z^*}$, which defines $bz = ax_{i^*,2}v$ for $c^* = 0$, i.e., $g^v = g^{bz/(ax_{i^*,2})}$.
7. Pick $W^* \xleftarrow{\$} \{0,1\}^{\ell_0+\ell_1}$, *implicitly* define $H_2((g^{b/a})^{z^*/x_{i^*,2}}) = (h^* \| \pi^*) \oplus W^*$, h^*, π^*.
8. Return $\mathfrak{C}^* = (E'^*, F^*, V^*, W^*)$ as the challenge ciphertext to adversary $\mathcal{A}_{\text{tran}}$.

Phase 2. Adversary $\mathcal{A}$ continues to issue queries as in Phase 1, with the restrictions described in the IND-PRE-CCA game. Algorithm $\mathcal{B}$ responds to these queries for $\mathcal{A}_{\text{tran}}$ as in Phase 1.

Guess. Eventually, adversary $\mathcal{A}_{\text{tran}}$ returns a guess $\delta' \in \{0,1\}$ to $\mathcal{B}$. Algorithm $\mathcal{B}_{\text{tran}}$ first retrieves $(m_{\delta'}, \omega, r)$ from the list H_1^{list} and test if $(g^a)^{r \cdot \mathtt{rk}_{i'\to i^*}(x_{i',1}H_4(\mathtt{pk}_{i',2})+x_{i',2})/t} = g^b$. If no such entry is found, $\mathcal{B}_{\text{tran}}$ randomly picks a tuple (R, β) from the list H_2^{list} and outputs $R^{x_{i^*,2}/z^*}$ as the solution to the given DCDH instance.

This completes the description of the simulation. It remains to related the probabilities for success and the execution times of the simulation and the adversary, which will be shown in Lemma 1 and Lemma 2. □

A.5 Nontransformable Ciphertext Security

Without loss of generality, we assume that the Schnorr signature is (t', ν)-EUF-CMA secure for some probability $0 < \nu < \epsilon$. suppose there exists a t-time adversary $\mathcal{A}$ who can break the IND-PRE-CCA security of our scheme for nontransformable ciphertext with advantage $\epsilon - \nu$, then we show how to construct an algorithm $\mathcal{B}$ which can break the (t', ϵ')-CDH assumption in $\mathbb{G}$, given as input a CDH challenge tuple (g, g^a, g^b). To output g^{ab} eventually, algorithm $\mathcal{B}_{\text{notr}}$ acts as the challenger and plays the IND-PRE-CCA game with adversary $\mathcal{A}_{\text{notr}}$ in the following way.

– *Uncorrupted key generation*: Algorithm $\mathcal{B}_{\text{notr}}$ first picks $x_{i,1}, x_{i,2} \xleftarrow{\$} \mathbb{Z}_q^*$, and define $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}) = \left((g^a)^{1/H_4(\mathtt{pk}_{i,2})} \cdot g^{x_{i,1}}, g^{x_{i,2}}/g^a\right)$. Next, set $c_i = 0$ and add the tuple $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$

to the K^{list}. Finally, it returns $\mathtt{pk}_i$ to adversary $\mathcal{A}$. The private key with respect to $\mathtt{pk}_i$ is $\mathtt{sk}_i = (\frac{a}{H_4(\mathtt{pk}_{i,2})} + x_{i,1}, -a + x_{i,2})$, is unknown to both $\mathcal{B}_{\text{notr}}$ and $\mathcal{A}_{\text{notr}}$.

– *Corrupted key generation*: $\mathcal{B}_{\text{notr}}$ picks $x_{j,1}, x_{j,2} \xleftarrow{\$} \mathbb{Z}_q^*$ and defines $\mathtt{pk}_j = (g^{x_{j,1}}, g^{x_{j,2}})$ and $c_j = 1$. It then adds the tuple $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ to the K^{list} and returns $(\mathtt{pk}_j, (x_{j,1}, x_{j,2}))$.
– *Re-encryption key generation*: For the re-encryption key from user i to user j, $\mathcal{B}_{\text{notr}}$ parses $\mathtt{pk}_i$ as $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2})$ and $\mathtt{pk}_j = (\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2})$. Next, it recovers tuples $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ and $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ from the K^{list}. Then, it constructs the re-encryption key $\mathtt{rk}_{i\to j}$ for adversary $\mathcal{A}$ according to the following situations:
  - If $c_i = 1$, $\mathcal{B}_{\text{notr}}$ return the result of $\mathsf{ReKeyGen}(\mathtt{sk}_i, \mathtt{pk}_j)$ to $\mathcal{A}$ since $\mathtt{sk}_i = (x_{i,1}, x_{i,2})$ is known.
  - If $c_i = 0$, it means that $\mathtt{sk}_i = (\frac{a}{H_4(\mathtt{pk}_{i,2})} + x_{i,1}, -a + x_{i,2})$. $\mathcal{B}_{\text{notr}}$ picks $h \xleftarrow{\$} \{0,1\}^{\ell_0}, \pi \xleftarrow{\$} \{0,1\}^{\ell_1}$ and returns $\mathtt{rk}_{i\to j} = (\mathtt{rk}_{i\to j}^{\langle 1\rangle} = \frac{h}{x_{i,1}H_4(\mathtt{pk}_{i,2}) + x_{i,2}}, V = g^{H_1(h,\pi)}, W = H_2(\mathtt{pk}_{j,2}^v) \oplus (h\|\pi))$, which is valid since $x_{i,1}H_4(\mathtt{pk}_{i,2}) + x_{i,2} = (\frac{a}{H_4(\mathtt{pk}_{i,2})} + x_{i,1})H_4(\mathtt{pk}_{i,2}) + (-a + x_{i,2})$.

Phase 1. ODec: Same as $\mathcal{B}_{\text{orig}}$.
Challenge. When $\mathcal{A}_{\text{notr}}$ decides that Phase 1 is over, it outputs a public key $\mathtt{pk}_{i^*} = (\mathtt{pk}_{i^*,1}, \mathtt{pk}_{i^*,2})$ and two equal-length messages $m_0, m_1 \in \{0,1\}^{\ell_0}$. Algorithm $\mathcal{B}_{\text{notr}}$ responds as follows:

1. Recover tuple $(\mathtt{pk}_{i^*}, x_{i^*,1}, x_{i^*,2}, c^*)$ from K^{list}.
2. Pick $\delta \xleftarrow{\$} \{0,1\}, \omega^* \xleftarrow{\$} \{0,1\}^{\ell_1}$, and issue an H_1 query on (m_δ, ω^*) to obtain the response r^*.
3. Pick $h^* \xleftarrow{\$} \{0,1\}^{\ell_0}, \pi^* \xleftarrow{\$} \{0,1\}^{\ell_1}$ and $W^* \xleftarrow{\$} \{0,1\}^{\ell_0+\ell_1}$. Then *implicitly* define $H_1(h^*, \pi^*) = b$ and $H_2(g^{-ab}g^{b\cdot x_{i^*,2}}) = (h^*\|\pi^*) \oplus W^*$ (note that $\mathcal{B}_{\text{notr}}$ knows neither b nor $g^{-ab}g^{b\cdot x_{i^*,2}}$).
4. Define $E'^* = g^{r^*h^*}$, $F^* = H_2(g^{r^*}) \oplus (m_\delta\|\omega^*), V^* = g^b$.
5. Return $\mathfrak{C}^* = (E'^*, E^*, F^*, W^*)$ as the challenge ciphertext to adversary $\mathcal{A}_{\text{notr}}$.

Observe that the challenge ciphertext $\mathfrak{C}^*$ is identically distributed as the real one from the construction. To see this, letting $r^* = b$, we have

$$V^* = g^b = g^{v^*},$$
$$W^* = H_2(g^{-ab}g^{b\cdot x_{i^*,2}}) \oplus (h^*\|\pi^*) = H_2((g^{-a+x_{i^*,2}})^b) \oplus (h^*\|\pi^*) = H_2(pk_{i^*,2}^{r^*}) \oplus (h^*\|\pi^*).$$

Guess. Same as Phase 1.

Eventually, adversary $\mathcal{A}$ returns a guess $\delta' \in \{0,1\}$ to $\mathcal{B}_{\text{notr}}$. Algorithm $\mathcal{B}_{\text{notr}}$ randomly picks a tuple (R, β) from the list H_2^{list} and outputs $\left(\frac{R}{g^{b\cdot x_{i^*,2}}}\right)^{-1}$ as the solution to the given CDH instance.

Analysis. It is clear that the public keys and the re-encryption key are distributed correctly. The simulation of the decryption oracle is perfect, with the exception that simulation errors may occur in rejecting some valid ciphertexts (denote this event by DErr). A similar analysis as in Appendix A.6 can yield $\Pr[\mathsf{DErr}] \le \frac{(q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} + \frac{2q_d}{q}$.

Next, we evaluate the simulations of the random oracles. It is clear that the simulations of H_3 and H_4 are perfect. Let AskH_1^* be the event that (h^*, π^*) has been queried to H_1, and AskH_2^* be the event that $g^{-ab}g^{b\cdot x_{i^*,2}}$ has been queried to H_2. The simulations of H_1 and H_2 are also perfect, as long as AskH_1^* and AskH_2^* did not occur, where h^* and π^* are chosen by $\mathcal{B}_{\text{notr}}$ in the Challenge phase.

Now, let Good denote the event $(\mathsf{AskH}_2^* \vee (\mathsf{AskH}_1^* | \neg \mathsf{AskH}_2^*) \vee \mathsf{DErr})$. A similar analysis as in Appendix A.6 can yields

$$\epsilon - \nu \leq \Pr[\mathsf{Good}] \leq (\Pr[\mathsf{AskH}_2^*] + \Pr[\mathsf{AskH}_1^* | \neg \mathsf{AskH}_2^*] + \Pr[\mathsf{DErr}]),$$

and then

$$\Pr[\mathsf{AskH}_2^*] \geq \epsilon - \nu - \Pr[\mathsf{AskH}_1^* | \neg \mathsf{AskH}_2^*] - \Pr[\mathsf{DErr}] \geq \epsilon - \nu - \frac{q_{H_1} + (q_{H_1} + q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{2q_d}{q}.$$

If AskH_2^* happens, algorithm $\mathcal{B}$ will be able to solve DCDH instance. Therefore, we obtain

$$\epsilon' \geq \frac{1}{q_{H_2}} \Pr[\mathsf{AskH}_2^*] \geq \frac{1}{q_{H_2}} \left(\epsilon - \nu - \frac{q_{H_1} + (q_{H_1} + q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{2q_d}{q} \right).$$

From the description of the simulation, $\mathcal{B}$'s running time can be bounded by

$$\begin{aligned} t' \leq\ & t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_u + q_c + q_{rk} + q_d)O(1) \\ & + (2q_u + 2q_c + 2q_{rk} + 2q_d + (2q_{H_2} + 2q_{H_1})q_d)t_{\text{exp}}. \end{aligned}$$

A.6 Lemmata for Probability Analysis of the Simulations

Lemma 1. *With $\mathcal{A}_{\text{orig}}$, $\mathcal{B}$ can solve the DCDH problem with advantage ϵ' within time t' where*

$$\begin{aligned} t' \leq\ & t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + n_u + n_c + q_{rk} + q_{re} + q_d)O(1) \\ & + (2n_u + 2n_c + 2q_{rk} + 5q_{re} + 2q_d + q_{H_1}q_{re} + (2q_{H_2} + 2q_{H_1})q_d)t_{\text{exp}}, \\ \epsilon' \geq\ & \frac{1}{q_{H_2}} \Big(\frac{\epsilon}{e(1+q_{rk})} - \frac{q_{H_1}2^{\ell_0} + q_{H_3} + (q_{H_1} + q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{q_{re} + 2q_d}{q} - \epsilon_1 - \epsilon_2 \Big), \end{aligned}$$

t_{exp} denotes the time to exponentiate in group $\mathbb{G}$, e is the base of the natural logarithm, ϵ_1 denotes the advantage in breaking the CCA security of the underlying "hashed" ElGamal encryption and ϵ_2 denotes the advantage in breaking the existential unforgeability of the underlying Schnorr signature.

Proof (Lemma 1). The main idea of the proof is borrowed from [BSNS05]. We first evaluate the simulations of the random oracles. It is clear that the simulation of H_4 is perfect. Let AskH_3^* be the event that $\mathcal{A}_{\text{orig}}$ queried (D^*, E^*, F^*) to H_3 before Challenge phase. The simulation of H_3 is also perfect, as long as AskH_3^* did not occur. Since F^* is randomly chosen from $\{0,1\}^{\ell_0+\ell_1}$ by the challenger in Challenge phase, we have $\Pr[\mathsf{AskH}_3^*] \leq \frac{q_{H_3}}{2^{\ell_0+\ell_1}}$. Let AskH_1^* be the event that (m_δ, ω^*) has been queried to H_1, and AskH_2^* be the event that g^{ab} has been queried to H_2. The simulations of H_1 and H_2 are also perfect, as long as AskH_1^* and AskH_2^* did not occur, where δ and ω^* are chosen by $\mathcal{B}$ in the Challenge phase.

It is clear that the responses to $\mathcal{A}_{\text{orig}}$'s uncorrupted/corrupted-key generation queries are perfect. Let Abort denote the event of $\mathcal{B}$'s aborting during the simulation of the re-encryption key queries or in the Challenge phase. We have $\Pr[\neg\mathsf{Abort}] \geq \theta^{q_{rk}}(1-\theta)$, which is maximized at $\theta_{\text{opt}} = \frac{q_{rk}}{1+q_{rk}}$. Using θ_{opt}, the probability $\Pr[\neg\mathsf{Abort}]$ is at least $\frac{1}{e(1+q_{rk})}$.

The simulation of the re-encryption key queries is the same as the real one, except for the case $(c_i = 0 \ \wedge \ c_j = 1)$ or $(c_i = 0 \ \wedge \ c_j = 0)$, in which the component $\mathtt{rk}_{i \to j}^{(1)}$ is randomly chosen. If event Abort does not happen, this is computationally indistinguishable from the real world according to the following facts.

1. The secret key $\mathtt{sk}_j$ is unknown to $\mathcal{A}$ since $c_j \neq$ '$-$'.
2. $(\mathtt{pk}_{j,2}^v, H_2(g^v) \oplus (h\|\pi))$ with $v = H_1(h, \pi)$ is in fact an encryption of h under $\mathtt{pk}_{j,2}$ using the "hashed" ElGamal encryption scheme [Gam84,FO99,BSNS05], which is based on the CDH assumption.

To reduce the indistinguishability to that of the underlying encryption scheme, we need the two following facts.

1. The value v is generated at random and is unrelated to any other values – v is determined by $H_1(h, \pi)$, see the point below.
2. The values h and π are not used elsewhere in the proof – this is ensured since $\mathtt{rk}'_{i\to j}$ is randomly chosen (this is the only other place where h may appear) and the decryption oracle only returns the message instead of any intermediate values like h or π.

One may also refer to the proof for the transformed ciphertext security to see how the DCDH problem instance is embedded to (V, W). We remark that even though the value of h is a function of the unknown secret key of the delegator, the simulator can execute in a different "mode" such that the value of the secret key of the delegator is known, as we are relying on the security of the underlying encryption with respect to the delegatee here.

Next, we analyze the simulation of the re-encryption queries. This simulation is also perfect, unless $\mathcal{A}_{\text{orig}}$ can submit valid original ciphertexts without querying hash function H_1 (denote this event by REErr). However, since H_1 acts as a random oracle, we have $\Pr[\mathsf{REErr}]\, leq \frac{q_{\text{re}}}{q}$.

The simulation of the decryption oracle is perfect, with the exception that simulation errors may occur in rejecting some valid ciphertexts. However, these errors are not significant as shown below: Suppose a ciphertext $\mathfrak{C}$ has been queried to the decryption oracle. Even if $\mathfrak{C}$ is a *valid* ciphertext, there is a possibility that $\mathfrak{C}$ can be produced without querying g^r to H_2, where $r = H_1(m, \omega)$. Let Valid be an event that $\mathfrak{C}$ is valid. Let AskH_2 and AskH_1 respectively be the events that g^r has been queried to H_2 and (m, ω) has been queried to H_1. We have $\Pr[\mathsf{Valid}|(\neg\mathsf{AskH}_1 \vee \neg\mathsf{AskH}_2)] \leq \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_1] + \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_2)] \leq \frac{2}{q}$. To see, the probability that $\mathcal{A}$ can come up with a "valid" E with respect to the public key and the H_1's output without querying H_1 at that point is $\frac{1}{q}$. Similarly, $\mathcal{A}$ can come up with a "valid" F with respect to the H_2's output without querying H_2 at the concerned point is again $\frac{1}{q}$.

Let DErr be the event that $\mathsf{Valid}|(\neg\mathsf{AskH}_1 \vee \neg\mathsf{AskH}_2)$ happens during the entire simulation. Then, since $\mathcal{A}_{\text{orig}}$ issues at most q_d decryption oracles, we have $\Pr[\mathsf{DErr}] \leq \frac{(q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} + \frac{2q_d}{q}$.

Now, let Err denote the event $(\mathsf{AskH}_2^* \vee \mathsf{AskH}_1^* \vee \mathsf{AskH}_3^* \vee \mathsf{REErr} \vee \mathsf{DErr})\big|\neg\mathsf{Abort}$. If Err does not happen, due to the randomness of the output of the random oracle H_2, it is clear that adversary $\mathcal{A}_{\text{orig}}$ cannot gain any advantage greater than $\frac{1}{2}$ in guessing δ. Namely, we have $\Pr[\delta = \delta'|\neg\mathsf{Err}] = \frac{1}{2}$. Hence, by splitting $\Pr[\delta' = \delta]$, we have

$$\begin{aligned}
\Pr[\delta' = \delta] &= \Pr[\delta' = \delta|\neg\mathsf{Err}]\Pr[\neg\mathsf{Err}] + \Pr[\delta' = \delta|\mathsf{Err}]\Pr[\mathsf{Err}] \\
&\leq \frac{1}{2}\Pr[\neg\mathsf{Err}] + \Pr[\mathsf{Err}] = \frac{1}{2} + \frac{1}{2}\Pr[\mathsf{Err}] \\
\text{and} \qquad \Pr[\delta' = \delta] &\geq \Pr[\delta' = \delta|\neg\mathsf{Err}]\Pr[\neg\mathsf{Err}] = \frac{1}{2} - \frac{1}{2}\Pr[\mathsf{Err}].
\end{aligned}$$

By definition of the advantage for the IND-PRE-CCA adversary, we then have

$$\begin{aligned}
\epsilon &= \big|2 \times \Pr[\delta' = \delta] - 1\big| \\
&\leq \Pr[\mathsf{Err}] = \Pr[(\mathsf{AskH}_2^* \vee \mathsf{AskH}_1^* \vee \mathsf{AskH}_3^* \vee \mathsf{REErr} \vee \mathsf{DErr})\big|\neg\mathsf{Abort}] \\
&\leq (\Pr[\mathsf{AskH}_2^*] + \Pr[\mathsf{AskH}_1^*] + \Pr[\mathsf{AskH}_3^*] + \Pr[\mathsf{REErr} + \Pr[\mathsf{DErr}]) \,/\, \Pr[\neg\mathsf{Abort}].
\end{aligned}$$

Since $\mathcal{B}$ picks $\omega \xleftarrow{\$} \{0,1\}^{\ell_1}$ which is hidden by the "one-time pad" given by H_2, $\Pr[\mathsf{AskH}_1^*] \leq \frac{q_{H_1}}{2^{\ell_1}}$, we obtain the following bound:

$$\begin{aligned}\Pr[\mathsf{AskH}_2^*] &\geq \Pr[\neg\mathsf{Abort}] \cdot \epsilon - \Pr[\mathsf{AskH}_1^*] - \Pr[\mathsf{AskH}_3^*] - \Pr[\mathsf{DErr}] - \Pr[\mathsf{REErr}] \\ &\geq \frac{\epsilon}{e(1+q_{rk})} - \frac{q_{H_1}}{2^{\ell_1}} - \frac{q_{H_3}}{2^{\ell_0+\ell_1}} - \frac{(q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{2q_d}{q} - \frac{q_{\mathrm{re}}}{q} \\ &= \frac{\epsilon}{e(1+q_{rk})} - \frac{q_{H_1}2^{\ell_0} + q_{H_3} + (q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{q_{\mathrm{re}}+2q_d}{q}.\end{aligned}$$

If AskH_2^* happens, algorithm $\mathcal{B}$ will be able to solve DCDH instance. Therefore, we obtain

$$\epsilon' \geq \frac{1}{q_{H_2}} \Pr[\mathsf{AskH}_2^*] \geq \frac{1}{q_{H_2}}\Big(\frac{\epsilon}{e(1+q_{rk})} - \frac{q_{H_1}2^{\ell_0} + q_{H_3} + (q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{q_{\mathrm{re}}+2q_d}{q}\Big).$$

From the description of the simulation, $\mathcal{B}$'s running time can be bounded by

$$\begin{aligned}t' \leq\ & t + (q_{H_1}+q_{H_2}+q_{H_3}+q_{H_4}+n_u+n_c+q_{rk}+q_{re}+q_d)O(1) \\ &+ (2n_u+2n_c+2q_{rk}+5q_{re}+2q_d+q_{H_1}q_{re}+(2q_{H_2}+2q_{H_1})q_d)t_{\exp}.\end{aligned}$$

This completes the proof of Lemma 1. □

Lemma 2. *With $\mathcal{A}_{\mathrm{tran}}$, $\mathcal{B}$ can solve the CDH problem with advantage ϵ' within time t' where*

$$\begin{aligned}t' \leq\ & t + (q_{H_1}+q_{H_2}+q_{H_3}+q_{H_4}+n_u+n_c+q_{rk}+q_{re}+q_d)O(1) \\ &+ (2n_u+2n_c+2q_{rk}+3q_{re}+2q_d+(2q_{H_2}+2q_{H_1})q_d)t_{\exp}, \\ \epsilon' \geq\ & \frac{1}{q_{H_2}}\Big(\epsilon - \frac{q_{H_1}2^{\ell_0} + (q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{2q_d}{q} - \epsilon_2\Big),\end{aligned}$$

$t_{\exp}$ denotes the running time of an exponentiation in group $\mathbb{G}$ and ϵ_2 denotes the advantage in breaking the existential unforgeability of the underlying Schnorr signature.

Proof (Lemma 2). It is clear that the responses to $\mathcal{A}_{\mathrm{tran}}$'s uncorrupted/corrupted queries, re-encryption key generation queries and re-encryption queries are all perfect. The simulation of the decryption queries is perfect, with the exception that simulation errors may occur in rejecting some valid ciphertexts (denote this event by DErr). A similar analysis as in Lemma 1 can yields $\Pr[\mathsf{DErr}] \leq \frac{(q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} + \frac{2q_d}{q}$.

Next, we evaluate the simulations of the random oracles. It is clear that the simulations of H_3 and H_4 are perfect. Let AskH_1^* be the event that (h^*, π^*) has been queried to H_1, and AskH_2^* be the event that $g^{b/a}$ or $(g^{b/a})^{t/\mathtt{rk}_{i'\to i^*}(x_{i',1}H_4(\mathtt{pk}_{i',2})+x_{i',2})} \oplus (m_\delta \| \omega^*)$ has been queried to H_2. The simulations of H_1 and H_2 are also perfect, as long as AskH_1^* and AskH_2^* did not occur, where h^* and π^* are chosen by $\mathcal{B}$ in the Challenge phase.

Let Err denote $(\mathsf{AskH}_2^* \vee \mathsf{AskH}_1^* \vee \mathsf{DErr})$. A similar analysis as in Lemma 1 can yield

$$\Pr[\mathsf{AskH}_2^*] \geq \epsilon - \Pr[\mathsf{AskH}_1^*] - \Pr[\mathsf{DErr}] \geq \epsilon - \frac{q_{H_1}2^{\ell_0} + (q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{2q_d}{q}.$$

If AskH_2^* happens, algorithm $\mathcal{B}$ will be able to solve DCDH instance. Therefore, we obtain

$$\epsilon' \geq \frac{1}{q_{H_2}} \Pr[\mathsf{AskH}_2^*] \geq \frac{1}{q_{H_2}}\left(\epsilon - \frac{q_{H_1}2^{\ell_0} + (q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{2q_d}{q} - \epsilon_2\right).$$

From the description of the simulation, $\mathcal{B}$'s running time can be bounded by

$$t' \leq t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + n_u + n_c + q_{rk} + q_{re} + q_d)O(1) \\ + (2n_u + 2n_c + 2q_{rk} + 3q_{re} + 2q_d + (2q_{H_2} + 2q_{H_1})q_d)t_{\text{exp}}.$$

This completes the proof of Lemma 2. □

B Delegator Secret Security

Delegator secret security is formally defined via the following game:

Setup. Challenger $\mathcal{C}$ runs $\mathsf{Setup}(1^\kappa)$ and gives the global parameters $\mathtt{param}$ to $\mathcal{A}$.

Queries. $\mathcal{A}$ adaptively issues queries $q_1, \ldots, q_m$ where query q_i is one of the following:

- *Uncorrupted-key generation query*: $\mathcal{C}$ first runs $\mathsf{KeyGen}()$ to obtain a public/private key pair $(\mathtt{pk}_i, \mathtt{sk}_i)$, and then sends $\mathtt{pk}_i$ to $\mathcal{A}$.
- *Corrupted-key generation query*: $\mathcal{C}$ first runs $\mathsf{KeyGen}()$ to obtain a public/private key pair $(\mathtt{pk}_j, \mathtt{sk}_j)$, and then gives $(\mathtt{pk}_j, \mathtt{sk}_j)$ to $\mathcal{A}$.
- *Re-encryption key query* $\langle \mathtt{pk}_i, \mathtt{pk}_j \rangle$: $\mathcal{C}$ runs $\mathsf{ReKeyGen}(\mathtt{sk}_i, \mathtt{pk}_j)$ to generate a re-encryption key $\mathtt{rk}_{i \to j}$ and returns it to $\mathcal{A}$. Here $\mathtt{sk}_i$ is the private key with respect to $\mathtt{pk}_i$. It is required that $\mathtt{pk}_i$ and $\mathtt{pk}_j$ were generated beforehand a key generation query, either corrupted or uncorrupted.

Output. Finally, $\mathcal{A}$ outputs a private key $\mathtt{sk}_{i^*}$ with respect to the public key $\mathtt{pk}_{i^*}$. $\mathcal{A}$ wins the game if $\mathtt{sk}_{i^*}$ is indeed a valid private key came from a *uncorrupted*-key generation query.

We refer to the above adversary $\mathcal{A}$ as a DSK adversary, and define his advantage in attacking the PRE scheme's delegator secret security as $\mathrm{Adv}^{\mathrm{DSK}}_{\mathrm{PRE},\mathcal{A}} = \Pr[\mathcal{A} \text{ wins}]$, where the probability is taken over the random coins consumed by the challenger and the adversary.

Definition 9. *We say that a PRE scheme is $(t, n_u, n_c, q_{rk}, \epsilon)$-DSK secure, if for any t-time DSK adversary $\mathcal{A}$ that makes at most q_{rk} re-encryption key queries,* $\mathrm{Adv}^{\mathrm{DSK}}_{\mathrm{PRE},\mathcal{A}} \leq \epsilon$.

Definition 10. *The discrete logarithm (DL) problem in $\mathbb{G}$ is, given a tuple $(g, g^a) \in \mathbb{G}^2$ with unknown a, to compute a.*

Definition 11. *For a polynomial-time algorithm $\mathcal{B}$, we define his advantage in solving the DL problem in $\mathbb{G}$ as*

$\mathrm{Adv}^{\mathrm{DL}}_{\mathcal{B}} \triangleq \Pr[\mathcal{B}(g, g^a) = a]$, *where the probability is taken over the random choices of a in $\mathbb{Z}_q$, the random choice of g in $\mathbb{G}$, and the random bits consumed by $\mathcal{B}$. We say that the (t, ϵ)-DL assumption holds in group $\mathbb{G}$, if no t- time adversary $\mathcal{B}$ has advantage at least ϵ in solving the DL problem in $\mathbb{G}$.*

The delegator secret security of our scheme can be ensured by the following Theorem 2. We remark that it does *not* rely on the random oracle model.

Theorem 2. *Our scheme has delegator secret security if the DL assumption holds in $\mathbb{G}$. Concretely, if there exists an DSK adversary $\mathcal{A}$, who breaks the $(t, q_u, q_c, q_{rk}, \epsilon)$-DSK security of our scheme, then there exists an algorithm $\mathcal{B}$ which can break the (t', ϵ)-DL assumption in $\mathbb{G}$ with $t' \leq t + O(2n_u t_{\text{exp}} + 2n_c t_{\text{exp}} + 2q_{rk} t_{\text{exp}})$.*

Proof. Suppose $\mathcal{B}$ is given as input a DL challenge tuple $(g, g^a) \in \mathbb{G}^2$ with unknown $a \xleftarrow{\$} \mathbb{Z}_q^*$. Algorithm $\mathcal{B}$'s goal is to output a. Algorithm $\mathcal{B}$ acts as a challenger and plays the DSK game with adversary $\mathcal{A}$ in the following way:

Setup. Algorithm $\mathcal{B}$ gives $(q, \mathbb{G}, g, H_1, \ldots, H_4, \ell_0, \ell_1)$ to $\mathcal{A}$. Here H_1, H_2, H_3 and H_4 are just cryptographic hash functions which are *not* modelled as random oracles.

Queries. Adversary $\mathcal{A}$ issues a series of queries as defined in the DSK game. $\mathcal{B}$ maintains a list K^{list}, which is initially empty, and answers these queries for $\mathcal{A}$ as follows:

- *Uncorrupted-key generation query*: Algorithm $\mathcal{B}$ first picks $x_{i,1}, x_{i,2} \xleftarrow{\$} \mathbb{Z}_q^*$, and defines $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2}) = \Big((g^a)^{1/H_4(\mathtt{pk}_{i,2})} \cdot g^{x_{i,1}}, g^{x_{i,2}}/g^a\Big)$. Next, set $c_i = 0$ and add the tuple $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ to the K^{list}. Finally, it returns $\mathtt{pk}_i$ to adversary $\mathcal{A}$. Note that the private key with respect to $\mathtt{pk}_i$ is $\mathtt{sk}_i = (\frac{a}{H_4(\mathtt{pk}_{i,2})} + x_{i,1}, -a + x_{i,2})$, which is unknown to both $\mathcal{B}$ and $\mathcal{A}$.
- *Corrupted-key generation query*: $\mathcal{B}$ picks $x_{j,1}, x_{j,2} \xleftarrow{\$} \mathbb{Z}_q^*$ and defines $\mathtt{pk}_j = (g^{x_{j,1}}, g^{x_{j,2}})$ and $c_j = 1$. It then adds the tuple $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ to the K^{list} and returns $(\mathtt{pk}_j, (x_{j,1}, x_{j,2}))$ to $\mathcal{A}$.
- *Re-encryption key query* $\langle \mathtt{pk}_i, \mathtt{pk}_j \rangle$: $\mathcal{B}$ parses $\mathtt{pk}_i$ as $\mathtt{pk}_i = (\mathtt{pk}_{i,1}, \mathtt{pk}_{i,2})$ and $\mathtt{pk}_j = (\mathtt{pk}_{j,1}, \mathtt{pk}_{j,2})$. Next, it recovers tuples $(\mathtt{pk}_i, x_{i,1}, x_{i,2}, c_i)$ and $(\mathtt{pk}_j, x_{j,1}, x_{j,2}, c_j)$ from the K^{list}. Then, it constructs the re-encryption key $\mathtt{rk}_{i\to j}$ for adversary $\mathcal{A}$ according to the following situations:
  - If $c_i = 1$, $\mathcal{B}$ can return the result of $\mathsf{ReKeyGen}(\mathtt{sk}_i, \mathtt{pk}_j)$ to $\mathcal{A}$, since $\mathtt{sk}_i = (x_{i,1}, x_{i,2})$ is known.
  - If $c_i = 0$, it means that $\mathtt{sk}_i = (\frac{a}{H_4(\mathtt{pk}_{i,2})} + x_{i,1}, -a + x_{i,2})$. $\mathcal{B}$ picks $h \xleftarrow{\$} \{0,1\}^{\ell_0}, \pi \xleftarrow{\$} \{0,1\}^{\ell_1}$ and returns $\mathtt{rk}_{i\to j} = \Big(\mathtt{rk}^{\langle 1\rangle}_{i\to j} = \frac{h}{x_{i,1}H_4(\mathtt{pk}_{i,2}) + x_{i,2}}, V = g^{H_1(h,\pi)}, W = H_2(\mathtt{pk}^v_{j,2}) \oplus (h\|\pi)\Big)$, which is valid since $x_{i,1}H_4(\mathtt{pk}_{i,2}) + x_{i,2} = (\frac{a}{H_4(\mathtt{pk}_{i,2})} + x_{i,1})H_4(\mathtt{pk}_{i,2}) + (-a + x_{i,2})$.

Output. Eventually, $\mathcal{A}$ outputs the private key $\mathtt{sk}_{i^*} = (\mathtt{sk}_{i^*,1}, \mathtt{sk}_{i^*,2})$ with respect to the public key $\mathtt{pk}_{i^*}$, which is came from an uncorrupted-key generation query. $\mathcal{B}$ recovers the tuple $(\mathtt{pk}_{i^*}, x_{i^*,1}, x_{i^*,2}, c_{i^*})$ from the K^{list} (Note that it must be $c_{i^*} = 0$), and then outputs $x_{i^*,2} - \mathtt{sk}_{i^*,2}$ as the solution to the DL challenge. Note that, if $\mathtt{sk}_{i^*} = (\mathtt{sk}_{i^*,1}, \mathtt{sk}_{i^*,2})$ is a valid private key with respect to $\mathtt{pk}_{i^*}$, we have $\mathtt{sk}_{i^*,1} = \frac{a}{H_4(\mathtt{pk}_{i^*,2})} + x_{i^*,1}$ and $\mathtt{sk}_{i^*,2} = -a + x_{i^*,2}$.

It can be verified that the responses for the uncorrupted/corrupted-key generation queries and the re-encryption key query are perfect. Thus, when adversary $\mathcal{A}$ outputs the valid private key $\mathtt{sk}_{i^*}$ with advantage ϵ, $\mathcal{B}$ can resolve the DL instance with the same advantage. It can be easily seen that $\mathcal{B}$'s running time is bounded by $t' \leq t + O(2n_u t_{\exp} + 2n_c t_{\exp} + 2q_{rk} t_{\exp})$. $\square$