Efficient Unidirectional Proxy Re-Encryption

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

[1] School of Information Systems,
Singapore Management University, Singapore
cryptjweng@gmail.com,robetdeng@smu.edu.sg
[2] Department of Computer Science
Courant Institute of Mathematical Sciences
New York University, NY 10012, USA
schow@cs.nyu.edu
[3] 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 message for Bob. The proxy only needs a re-encryption key given by Alice, and cannot learn anything about the message encrypted. This adds flexibility in various data security applications, such as confidential email, digital right management and distributed storage. In this paper, we study *unidirectional* PRE, where the re-encryption key only enables delegation in one direction but not the opposite. In PKC 2009, Shao and Cao [23] proposed a unidirectional PRE in the random oracle model. However, we show how to launch a chosen-ciphertext attack (CCA) on this recently proposed scheme and discuss the flaws in their proof. We then propose an efficient unidirectional PRE scheme (without resorting to pairings). We gain the high efficiency and CCA-security under the computational Diffie-Hellman assumption, in the random oracle model.

Key words: proxy re-encryption, unidirectional, 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 [20], the encrypted data often needs to be re-distributed in practice, i.e. the data encrypted under a public key pk_i should also be encrypted under another independently generated public key pk_j. This can be easily done if the holder of the secret key sk_i (corresponding to pk_i) is *online* – simply decrypts the ciphertext and re-encrypts the plaintext to 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 ciphertext.

To solve this key management problem which hinders the practical adoption of encryption, Blaze, Bleumer and Strauss [5] 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 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) [24], distributed file storage systems [1, 2], law enforcement [17], encrypted email forwarding [5], outsourced filtering of encrypted spam [1, 2], etc. 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 the problem that led to the compromise of Apple's iTunes DRM [24]. With a PRE scheme, the problem is solved since re-encryption can be performed without awarding the re-encryption 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, so 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 email messages to his colleague.

1.1 Related Work

Proxy *encryption* (no "re-") (e.g. [20, 18, 17]) also allows a delegator Alice to delegate her decryption power to a delegatee Bob with the help of a proxy. 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 [7, 19], PRE is a (strict) subset of proxy encryption.

Another notion with a similar name is universal re-encryption [14], in which the ciphertexts are re-randomized, instead of changing the underlying public key in PRE.

Blaze, Bleumer and Strauss's seminal work [5] proposed a bidirectional PRE scheme against chosen plaintext attack (CPA). However, as indicated by [1], 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.

There are a number of PRE schemes proposed afterwards. Their properties are summarized in Table 1. Within each category, the schemes are chronologically arranged. Unidirectional scheme is indicated by "$\rightarrow$" while bidirectional scheme is indicated by "$\leftrightarrow$". Generally speaking, a bidirectional scheme is easier to design than a unidirectional one (as one may inferred from the time of their appearances). "RO" denotes whether random oracle model is assumed for the security proof and "$\hat{e}(\cdot,\cdot)$" denotes whether the construction relies on bilinear pairings.

Schemes	Uni/Bi	Security	RO	$\hat{e}(\cdot,\cdot)$
Public-key-based				
Ateniese *et al.* [1]	$\rightarrow$	CPA	✓	✓
Hohenberger *et al.* [16]	$\rightarrow$	CPA	×	✓
Canetti-Hohenberger [7]	$\leftrightarrow$	CCA	×	✓
Libert-Vergnaud [19]	$\rightarrow$	RCCA[4]	×	✓
Deng *et al.* [11]	$\leftrightarrow$	CCA	✓	×
Shao-Cao [23]	$\rightarrow$	CCA?	✓	×
Ours	$\rightarrow$	CCA	✓	×
Identity-based				
Green-Ateniese [15]	$\rightarrow$	CCA	✓	✓
Matsuo [21][5]	$\leftrightarrow$	CPA	×	✓
Chu-Tzeng [9]	$\rightarrow$	CCA	×	✓

Table 1. Summary of Proxy Re-Encryption 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. Below we look into two schemes to see what "ingredients" are useful for constructing CCA-secure PRE.

Most existing PRE schemes [1, 16, 15, 7, 21, 9, 19], no matter ID-based or not, are realized by pairings. In the bidirectional scheme proposed by Canetti and Hohenberger [7], the transformation key is simply $\mathsf{rk}_{i \leftrightarrow j} = x_j / x_i$ for the pair of delegation partners[6] with public key $\mathsf{pk}_i = g^{x_i}$ and $\mathsf{pk}_j = g^{x_j}$. The ciphertext comes with the term pk_i^r for randomness $r \in \mathbb{Z}_p$ which can be transformed to pk_j^r easily by using $\mathsf{rk}_{i \leftrightarrow j}$. The ciphertext validity that was based on g and the original recipient's public key pk_i can still be checked after transformation with the help of the pairing function $\hat{e}(\cdot, \cdot)$ with respect to the generator g and the new public key pk_j. For the unidirectional PRE scheme proposed by Libert and Vergnaud [19] (hereinafter referred as $\mathcal{LV}08$), the transformation key is in the form $\mathsf{rk}_{i \leftrightarrow j} = g^{x_j / x_i}$. The ciphertext also comes with the term pk_i^r and the message is encrypted by $\hat{e}(g, g)^r$. As expected, a pairing will be applied to get $\hat{e}(g^{x_j/x_i}, \mathsf{pk}_i^r) = \hat{e}(g, g^r)^{x_j}$, so the message can be recovered by firstly cancelling x_j from this term. These techniques in performing unidirectional transformation and ciphertext validity checking intrinsically require the use of pairing function.

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 different techniques in constructing CCA-secure PRE. Practically, we want a PRE scheme with simple design, high computational efficiency and short ciphertext size. Removing pairing[7] from PRE constructions is an interesting problem, which is also one of the open problems left by Canetti and Hohenberger [7].

Very recently, Shao and Cao [23] proposed a unidirectional PRE scheme without pairing (referred as $\mathcal{SC}09$). Their proof for CCA-security is given in the random orale model, under the decisional Diffie-Hellman assumption over $\mathbb{Z}_{N^2}^*$, where N is a safe-prime modulus.

However, removing pairing is a mean, not the goal. $\mathcal{SC}09$ requires a several (4 to 5) exponentiations in $\mathbb{Z}_{N^2}^*$ for encryption, re-encryption and decryption[8], and incurs an overhead of a few (3 plus a proof-of-knowledge, to 5) $\mathbb{Z}_{N^2}^*$ elements for original ciphertext and transformed ciphertext. Note that the modulus being used is N^2, not N. Its performance over pairing-based scheme (e.g. $\mathcal{LV}08$), which instantiated on elliptic curves consist of much shorter group elements at the same security level, is questionable. Finally, their assumption is still of the Diffie-Hellman favor, and is decisional, which is stronger than its computational version.

[4] Replayable chosen-ciphertext attacks (RCCA) [8] is a weaker variant of chosen-ciphertext attack (CCA) in which a harmless mauling of the challenge ciphertext is tolerated.

[5] The first scheme proposed in [21] is a hybrid system which transforms ciphertexts encrypted under a traditional PKI-based public key into the ciphertexts that can be decrypted by an IBE secret key. The other one is purely ID-based; but it requires the key generation center (not the user) to give the re-encryption key.

[6] For bidirectional scheme, 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 computation resource limited settings.

[8] Speed-up by Chinese remainder theorem is not possible, except 2 of the exponentiations in decryption.

Most importantly, we identify flaws in their security proof which translate to a real-world chosen-ciphertext attack against $\mathcal{SC}09$. Possible fixes further degrade the performance in decryption time or ciphertext size. In view of this, we propose an efficient unidirectional PRE scheme without pairings, which is provably CCA-secure under the *standard* computational Diffie-Hellman assumption, in the random oracle model. Our design is based on ElGamal encryption [13] and Schnorr signature [22], which is (arguably) simple. Our re-encryption process is more natural – the decryption of transformed ciphertexts shares a similar process as that of original ciphertexts, in particular, it does not require the input of the delegator's public key (c.f. $\mathcal{SC}09$, the transformed ciphertext should have the delegator's public key included).

2 Preliminaries

2.1 Notations and Complexity Assumptions

For a prime q, let $\mathbb{Z}_q$ denote the set $\{0, 1, 2, \ldots, q-1\}$, and $\mathbb{Z}_q^*$ denote $\mathbb{Z}_q \backslash \{0\}$. For a finite set S, $x \stackrel{\$}{\leftarrow} S$ means choosing an element x from S with a uniform distribution.

Definition 1. *Let $\mathbb{G}$ be a cyclic multiplicative group with prime order q. The* Computational Diffie-Hellman *(CDH) problem in $\mathbb{G}$ is, given $(g, g^a, g^b) \in \mathbb{G}^3$ with $a, b \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$, to compute g^{ab}.*

Definition 2. *For an adversary $\mathcal{B}$, we define his advantage in solving the* CDH *problem 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 randomly choices of a, b and the random bits consumed by $\mathcal{B}$. We say that the (t, ϵ)-*CDH *assumption holds in $\mathbb{G}$ if no t-time adversary $\mathcal{B}$ has advantage at least ϵ in solving the* CDH *problem in $\mathbb{G}$.*

Bao *et al.* [4] introduced a variant of the CDH problem named divisible computation Diffie-Hellman (DCDH) problem, which is to compute g^{ab} given $(g, g^{\frac{1}{a}}, g^b) \in \mathbb{G}^3$ with unknown $a, b \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$. It is shown in [4] that the DCDH and CDH are equivalent in the same group.

2.2 Model of Unidirectional Proxy Re-Encryption Systems

Formally, a unidirectional PRE scheme consists of the following six algorithms [7]:

Setup(κ)**:** The setup algorithm takes as input a security parameter κ and outputs the global parameters *param*, which include a description of the message space $\mathcal{M}$. We assume that *param* is implicitly included in the input of the other algorithms for brevity.

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

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

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

ReEncrypt$(\mathsf{rk}_{i \to j}, \mathfrak{C}_i)$**:** The re-encryption algorithm takes as input a re-encryption key $\mathsf{rk}_{i \to j}$ and a ciphertext $\mathfrak{C}_i$ under public key pk_i. It outputs a ciphertext $\mathfrak{C}_j$ under public key pk_j.

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

Correctness requires that, for any parameters $param$, $m \in \mathcal{M}$, the following conditions hold:

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_i, C) = m \,\middle|\, (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}, C \leftarrow \mathsf{Encrypt}(\mathsf{pk}_i, m)\right] = 1,$$

$$\Pr\left[\mathsf{Decrypt}\left(\mathsf{sk}_j, \mathsf{rk}_{i,\rightarrow j}, \mathfrak{C}_j\right) = m \,\middle|\, \begin{array}{c} (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen},\ (\mathsf{sk}_j, \mathsf{pk}_j) \leftarrow \mathsf{KeyGen}, \\ \mathsf{rk}_{i,\rightarrow j} \leftarrow \mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j),\ \mathfrak{C}_i \leftarrow \mathsf{Encrypt}(\mathsf{pk}_i, m), \\ \mathfrak{C}_j \leftarrow \mathsf{ReEncrypt}(\mathsf{rk}_{i\rightarrow j}, \mathfrak{C}_i) \end{array}\right] = 1$$

Next, we review the game-based definition of chosen-ciphertext security for PRE systems derived from [7, 19]. We slightly modify it to allow the *adaptive* corruptions of users.

Setup. $\mathcal{C}$ takes a security parameter κ and runs algorithm Setup. It gives $\mathcal{A}$ the resulting global parameters $param$.

Phase 1. $\mathcal{A}$ adaptively issues queries $q_1, \ldots, q_m$ where each query q_i is one of the following:

- *Uncorrupted key generation query*: $\mathcal{C}$ first runs KeyGen to obtain a public/private key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$, and then sends pk_i to $\mathcal{A}$.
- *Corrupted key generation query*: $\mathcal{C}$ first runs KeyGen to obtain a public/private key pair $(\mathsf{pk}_j, \mathsf{sk}_j)$, and then gives $(\mathsf{pk}_j, \mathsf{sk}_j)$ to $\mathcal{A}$.
- *Re-encryption key generation query* $\langle \mathsf{pk}_i, \mathsf{pk}_j \rangle$: $\mathcal{C}$ runs $\mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$ and returns the generated re-encryption key $\mathsf{rk}_{i\rightarrow j}$ to $\mathcal{A}$. sk_i is the private key with respect to pk_i.
- *Re-encryption query* $\langle \mathsf{pk}_i, \mathsf{pk}_j, \mathfrak{C}_i \rangle$: $\mathcal{C}$ first runs $\mathsf{ReKeyGen}$to generate the re-encryption key $\mathsf{rk}_{i\rightarrow j}$. Then it returns the result of $\mathsf{ReEncrypt}(\mathsf{rk}_{i\rightarrow j}, \mathfrak{C}_i)$ to $\mathcal{A}$.
- *Decryption query* $\langle \mathsf{pk}, \mathfrak{C} \rangle$: Challenger $\mathcal{C}$ returns the result of $\mathsf{Decrypt}(\mathsf{sk}, \mathfrak{C})$ to $\mathcal{A}$, where sk is the private key with respect to pk.

For the last three kinds of queries, it is required that pk_i, pk_j, or pk were generated beforehand by a key generation query, either uncorrupted or corrupted.

Challenge. Once $\mathcal{A}$ decides that Phase 1 is over, it outputs two equal-length plaintexts $m_0, m_1 \in \mathcal{M}$ and a target public key pk_{i^*}, subjects to the following conditions:

1. $\mathsf{pk}_{i^*}$is generated by an *uncorrupted* key generation query,
2. $\mathcal{A}$ has never issued a re-encryption key generation query $\langle \mathsf{pk}_{i^*}, \mathsf{pk}_j \rangle$ if pk_j came from a corrupted key generation query.

Challenger $\mathcal{C}$ flips a random coin $\delta \in \{0, 1\}$, and sets the challenge ciphertext to be $\mathfrak{C}^* = \mathsf{Encrypt}(\mathsf{pk}_{i^*}, m_\delta)$, which is sent to $\mathcal{A}$.

Phase 2. $\mathcal{A}$ issues additional queries $q_{m+1}, \ldots, q_{max}$ of the following types:

- *Uncorrupted/Corrupted key generation query*: $\mathcal{C}$ responds as in Phase 1.
- *Re-encryption key generation query* $\langle \mathsf{pk}_i, \mathsf{pk}_j \rangle$: Challenger $\mathcal{C}$ responds as in Phase 1. Here it is required that, if $\mathcal{A}$ has obtained the private key sk_j with respect to pk_j, $\mathcal{A}$ is disallowed to issue the re-encryption key generation query $\langle \mathsf{pk}_{i^*}, \mathsf{pk}_j \rangle$.
- *Re-encryption query* $\langle \mathsf{pk}_i, \mathsf{pk}_j, \mathfrak{C}_i \rangle$: Challenger $\mathcal{C}$ responds as in Phase 1. Here it is required that, if $\mathcal{A}$ has obtained the private key sk_j with respect to pk_j, then $(\mathsf{pk}_i, \mathfrak{C}_i)$ cannot be a *derivative* of $(\mathsf{pk}_{i^*}, \mathfrak{C}^*)$ (to be defined later).
- *Decryption query* $\langle \mathsf{pk}, \mathfrak{C} \rangle$: Challenger $\mathcal{C}$ responds as in Phase 1. Here it is required that, $(\mathsf{pk}, \mathfrak{C})$ cannot be a *derivative* of $(\mathsf{pk}_{i^*}, \mathfrak{C}^*)$.

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

Derivative of $(\mathsf{pk}_{i^*}, \mathfrak{C}^*)$ is inductively defined in [7] as below:

1. $(\mathsf{pk}_{i^*}, \mathfrak{C}^*)$ is a derivative of itself (a trivial reflexivity condition).

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

We refer to adversary $\mathcal{A}$ as an IND-PRE-CCA adversary, and we define his advantage in attacking the PRE scheme as $\mathrm{Adv}_{\mathrm{PRE},\mathcal{A}}^{\text{IND-PRE-CCA}} = \left|\Pr[\delta' = \delta] - 1/2\right|$, where the probability is taken over the random coins consumed by the challenger and the adversary.

Definition 3. *A PRE scheme is said to be* $(t, q_u, q_c, q_{rk}, q_{re}, q_d, \epsilon)$-IND-PRE-CCA *secure, if for any t-time* IND-PRE-CCA *adversary $\mathcal{A}$ who makes at most q_u uncorrupted key generation queries, at most q_c corrupted key generation queries, 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* $\mathrm{Adv}_{PRE,\mathcal{A}}^{\text{IND-PRE-CCA}} \leq \epsilon$.

Delegator/Master Secret Security.[9] Delegator secret security is considered in Ateniese *et al* [1] which captures the intuition that, even if the dishonest proxy colludes with the delegatee, it is still impossible for them to derive the delegator's private key in full. The attack mode is quite simple so we defer its formalization to the Appendix.

3 Cryptanalysis of A Recent CCA-Secure Unidirectional PRE Scheme

3.1 Review of Shao-Cao's Scheme

Shao-Cao's scheme [23] is reviewed as below, up to minor notational differences. We use $\square$ to highlight the places which introduce the vulnerability.

Setup(κ): Given a security parameter κ, choose three hash functions H_1, H_2 and H_3 where $H_1 : \{0,1\} \to \{0,1\}^{\ell_2}$, $H_2 : \{0,1\} \to \{0,1\}^{\ell_0}$, and $H_3 : \{0,1\} \to \{0,1\}^{\ell_3}$. Here ℓ_0, ℓ_2 and ℓ_3 are security parameters determined by κ, and the message space $\mathcal{M}$ is $\{0,1\}^{\ell_0}$. The parameters are $param = (\kappa, H_1, H_2, H_3, \ell_0, \ell_2, \ell_3)$.

KeyGen: Given a security parameter κ, the key generation performs the following steps:
1. Choose two Sophie Germain primes p' and q', with $p' \neq q'$,and having bit-length of κ.
2. Compute $p = 2p' + 1$ and $q = 2q' + 1$, which are safe primes.
3. Choose three random numbers $\alpha \in \mathbb{Z}_{N^2}^*, a, b \in [1, pp'qq']$.
4. Choose a hash function $H : \{0,1\}^* \to \mathbb{Z}_{N^2}$.
5. Set $g_0 = \alpha^2 \bmod N^2$, $g_1 = g_0^a \bmod N^2$, and $g_2 = g_0^b \bmod N^2$.
6. The public key is $\mathsf{pk} = (H, N, g_0, g_1, g_2)$, the "weak" secret key is $\mathsf{wsk} = (a, b)$, and the long term secret key is $\mathsf{sk} = (p, q, p', q')$.

Either the long term secret key or the weak secret key can be used to decrypt (any) ciphertexts, but both the long term secret key and the weak secret key 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. $\mathsf{pk}_X = (H_X(\cdot), N_X, g_{X0}, g_{X1} = g_{X0}^{a_X}, g_{X2})$.

[9] This notion is named as *master secret security* in [1] since the delegator's public key is the master public key in their secure distributed storage application.

ReKeyGen$(\mathsf{sk}_X, \mathsf{pk}_Y)$**:** On input a long term secret key $\mathsf{sk}_X = (p_X, q_X, p'_X, q'_X)$, a "weak" secret key $\mathsf{wsk}_X = a_X$, and a public key $\mathsf{pk}_Y = (H_Y, N_Y, g_{Y0}, g_{Y1}, g_{Y2})$, 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. Randomly pick $\dot{\beta} \in \{0,1\}^{\ell_2}$.
2. Compute $rk^{(2)}_{X\to Y} = a_X - \dot{\beta} \bmod (p_X q_X p'_X q'_X)$.
3. Randomly pick $\dot{\sigma} \in \mathbb{Z}_{N_Y}$, compute $r_{X\to Y} = H_Y(\dot{\sigma}\|\dot{\beta})$.
4. Compute $\dot{C} = H_1(\dot{\sigma}) \oplus \dot{\beta}$.
5. Compute $\dot{A} = (g_{Y0})^{r_{X\to Y}} \bmod (N_Y)^2$ and $\dot{B} = (g_{Y2})^{r_{X\to Y}} \cdot (1 + \dot{\sigma}N_Y) \bmod (N_Y)^2$.

Encrypt(pk, m)**:** On input a public key $\mathsf{pk} = (H, N, g_0, g_1, g_2)$ and 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.[10]
5. Output the ciphertext $\mathfrak{C} = (A, B, C, D, c, s)$.

ReEncrypt$(\mathsf{rk}_{X\to Y}, \mathfrak{C}_X, \mathsf{pk}_X, \mathsf{pk}_Y)$**:** On input a re-encryption key $\mathsf{rk}_{X\to Y} = (\mathsf{rk}^{(1)}_{X\to Y}, \mathsf{rk}^{(2)}_{X\to Y})$ and a ciphertext $\mathfrak{C} = (A, B, C, D, c, s)$ under key $\mathsf{pk}_X = (H_X, N_X, g_{X0}, g_{X1}, g_{X2})$,

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 $A' = A^{\mathsf{rk}^{(2)}_{X\to Y}}$.
3. Output $\mathfrak{C}_Y = (A, \boxed{A'}, B, C, \mathsf{rk}^{(1)}_{X\to Y}) = (A, A', B, C, \dot{A}, \dot{B}, \dot{C})$.

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.

Decrypt$(\mathsf{sk}, \mathfrak{C})$**:** On input a private key and 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. Check if $c = H_3(A\|D\|g_0\|g_2\|(g_0)^s A^c\|(g_2)^s D^c\|(B\|C))$. If not, return $\perp$
     - if sk is in the form of (a, b), compute $\sigma = \frac{B/(A^a) - 1 \bmod N^2}{N}$.
     - if sk is in the form of (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 [6], and π is the inverse of $2p'q' \bmod N$.
  2. Compute $m = C \oplus H_2(\sigma)$.
  3. If $B = (g_1)^{H(\sigma\|m)} \cdot (1 + \sigma N) \bmod N^2$ holds, return m; else return $\perp$.
- If $\mathfrak{C}$ is in the form $\mathfrak{C} = (A, A', B, C, \dot{A}, \dot{B}, \dot{C})$ re-encrypted from pk_X to pk_Y:
  1. Note that the decryptor **is required to know** the delegator's public key. This deviates from our framework presented in Section 2.
     - if sk is in the form of (a, b), compute $\dot{\sigma} = \frac{\dot{B}/(\dot{A}^b) - 1 \bmod N_Y^2}{N_Y}$.
     - if sk is in the form of (p, q, p', q'), compute $\dot{\sigma} = \frac{(\dot{B}/g_{Y0}^{w_1})^{2p'q'} - 1 \bmod N_Y^2}{N_Y} \cdot \pi(\bmod N_Y)$, where w_1 is computed as that in [6], and π is the inverse of $2p'_Y q'_Y \bmod N_Y$.

[10] Let y_0, y_2, g_0, $g_2 \in \mathbb{G}$, where $\mathbb{G}$ is a cyclic group of quadratic residues modulo N^2 and N is a safe-prime modulus, and $H_3(\cdot) : \{0,1\}^* \to \{0,1\}^k$, where k is the security parameter. A pair (c, s) such that $c = H_3(y_0||y_2||g_0||g_2||g_0^s y_0^c||g_2^s y_2^c||m)$ is a signature of knowledge 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 $m \in \{0,1\}^*$, This pair can be computed by $\mathsf{SoK.Gen}(y_0, y_2, g_0, g_2, m)$ – 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.

2. Compute $\dot{\beta} = \dot{C} \oplus H_1(\dot{\sigma})$.
3. Check if $\dot{B} = (g_{Y1})^{H_Y(\dot{\sigma}\|\dot{\beta})} \cdot (1 + \dot{\sigma} N_Y) \bmod N_Y^2$. If not, return $\perp$.
   (Up to this point, only the decryptor's public key, $(H_Y, N_Y, g_{Y0}, g_{Y1}, g_{Y2})$, is used. Afterward, only the **delegator**'s public key, $(H_X, N_X, g_{X0}, g_{X1}, g_{X2})$, will be used.)
4. Compute $\sigma = (B/(\boxed{A' \cdot A^{\dot{\beta}}}) - 1 \bmod N_X^2)/N_X$, and $m = C \oplus H_2(\sigma)$.
5. If $B = (g_{X1})^{H_X(\sigma\|m)} \cdot (1 + \sigma N_X) \bmod N_X^2$ holds, return m; else return $\perp$.

3.2 Possible Vulnerabilities in the Re-Encryption Key

Before describing our attack, we first briefly explain how the re-encryption key is generated in *SC09*. Their ReKeyGen algorithm follows the "token-controlled encryption" paradigm, which is adopted by [15, 9] 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 [23].

Imagine that Bob (who holds public key pk_Y), who received delegation from Alice (who holds public key pk_X), now delegates his own decryption right to Carol. *If* the ReKeyGen algorithm requires Bob to use sk_Y (i.e. the whole private key) instead of just *some form* of the private key (e.g. a_Y in *SC09*), when his proxy colludes with Carol, sk_Y can be easily recovered. Furthermore, sk_Y can be used to recover $\dot{\beta}$ in the re-encryption key generated by Alice to Bob; the secret key of Alice, sk_X, can also be recovered exactly in the way how 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 [15, 9] fail, as pointed by [23].

3.3 Our Attack

Although some measures are taken in *SC09* to counter the above attack, we found that the token $\dot{\beta}$ is still not "securely" hidden. In particular, *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. These two weaknesses lead us to the following attack by an attacker $\mathcal{A}$. We suppose $\mathsf{pk}^*_X = (H_X(\cdot), N_X, g_{X0}, g_{X1}, g_{X2})$ is the challenge public key and $\mathfrak{C}^* = (A, B, C, D, c, s)$ is the challenge ciphertext.

1. Randomly pick $m \in M$ and $r \in \mathbb{Z}_{(N_X)^2}$, compute $\mathfrak{C} \leftarrow \mathsf{Encrypt}_{\mathsf{pk}^*_X}(m; r)$, i.e. using r as the randomness in the first step of Encrypt. *(Note that it is a* public key *encryption. Anyone can encrypt a message under* pk^*_X *using whatever randomness r he wants to use.)*
2. Issue a re-encryption oracle query to re-encrypt the ciphertext $\mathfrak{C}$ from pk^* to 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 SC09.)*
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 $(g_{X0})^{-r\dot{\beta}} \leftarrow (Z'/(g_{X1})^r)$. *(Recall that $\mathfrak{C}$ is prepared by $\mathcal{A}$ himself, so $\mathcal{A}$ knows r.)*

4. Issue a re-encryption oracle query to re-encrypt the ciphertext $\mathfrak{C}^*$ from pk^* to pk, $\mathcal{A}$ thus obtains $\mathfrak{C}_1 = (A, A', B, C, \dot{A}, \dot{B}, \dot{C})$. Now $(\mathsf{pk}, \mathfrak{C}_1)$ is also a *derivative* of the challenge. *(This is legitimate, since the secret key of* pk *is not compromised by* $\mathcal{A}$*.)*
5. Randomly pick $s \in \mathbb{Z}_{(N_X)^2}$, compute $\mathfrak{A}' \leftarrow A' \cdot (g_{X0}^{-r\dot{\beta}})^s$ and $\mathfrak{A} \leftarrow A \cdot (g_{X0})^{rs}$.
6. Issue a decryption oracle query under pk to decrypt $\mathfrak{C}' = (\mathfrak{A}, \mathfrak{A}', B, C, \dot{A}, \dot{B}, \dot{C})$.
7. Return the result of the decryption oracle as 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 $(\mathsf{pk}, \mathfrak{C}_1)$ of the challenge $(\mathsf{pk}^*, \mathfrak{C}^*)$, so the correct value of $\dot{\beta}$ can be recovered. (Note that $B, C, \dot{A}, \dot{B}, \dot{C}$ are the only values from the ciphertext being used for the first three steps of $\mathsf{Decrypt}$.) Moreover, in $\mathsf{Decrypt}$ (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 $\mathsf{Decrypt}$ 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) No such transitive relation exists; 3) $\mathcal{A}$ has made two re-encryption queries, $\mathfrak{C}$ has *nothing* to do with the challenge $\mathfrak{C}^*$, only the second one is on $\mathfrak{C}^*$ to obtain a ciphertext, thus only $(\mathsf{pk}, \mathfrak{C}_1)$ is considered as a derivative of the challenge, but $(\mathsf{pk}, \mathfrak{C}')$, where $\mathfrak{C}_1 \neq \mathfrak{C}'$, is *not* its derivative, and 4) $\mathcal{A}$ has not made any re-encryption key generation oracle query at all.

3.4 Flaws in the Proof and Possible Fixes

This attack originated from some flaws in their proof, 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. To encounter our attack, the first possible fix is to re-compute A in $\mathsf{Decrypt}$ and check whether it is correctly generated, which requires one more exponentiation in $\mathbb{Z}_{N^2}$. The other way is to include (c, s, D) in the transformed ciphertext for public ciphertext validity checking in both modes of decryption, but the ciphertext is even longer, and $\mathsf{Decrypt}$ requires at least three more exponentiations.

4 Our Proposed Unidirectional PRE Scheme

4.1 Construction

Our proposed unidirectional PRE scheme extends the *bidirectional* scheme proposed by Deng *et al.* [11], 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)**:** Given a security parameter κ, choose two primes p and q such that $q|p-1$ and the bit-length of q is 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, H_2, H_3 and H_4 where $H_1 : \{0,1\}^{\ell_0} \times \{0,1\}^{\ell_1} \to \mathbb{Z}_q^*$, $H_2 : \mathbb{G} \to \{0,1\}^{\ell_0+\ell_1}$, $H_3 : \{0,1\}^* \to \mathbb{Z}_q^*$ and $H_4 : \mathbb{G} \to \mathbb{Z}_q^*$. 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 $param = (q, \mathbb{G}, g, H_1, H_2, H_3, H_4, \ell_0, \ell_1)$.

KeyGen()**:** Randomly picks $\mathsf{sk}_i = (x_{i,1}, x_{i,2} \xleftarrow{\$} \mathbb{Z}_q^*)$, sets $\mathsf{pk}_i = (\mathsf{pk}_{i,1}, \mathsf{pk}_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$.

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

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

Encrypt(pk_i, m)**:** On input a public key $\mathsf{pk}_i = (\mathsf{pk}_{i,1}, \mathsf{pk}_{i,2})$ and a plaintext $m \in \mathcal{M}$:

1. Pick $u \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$ and compute $D = \left(\mathsf{pk}_{i,1}^{H_4(\mathsf{pk}_{i,2})}\mathsf{pk}_{i,2}\right)^u$.
2. Pick $\omega \stackrel{\$}{\leftarrow} \{0,1\}^{\ell_1}$, compute $r = H_1(m, \omega)$.
3. Compute $E = \left(\mathsf{pk}_{i,1}^{H_4(\mathsf{pk}_{i,2})}\mathsf{pk}_{i,2}\right)^r$ 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 $\mathfrak{C} = (D, E, F, s)$.

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

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

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

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

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

4.2 Security Analysis

We make three observations on the computation of a re-encryption key $\mathsf{rk}_{i\to j}$.

1. It just requires the input of $(\mathsf{sk}_i, \mathsf{pk}_j)$, but not sk_j, so our scheme is unidirectional.
2. Even though h can be recovered by anyone who owns sk_j, $\mathsf{rk}_{i\to j}^{\langle 1\rangle}$ only gives information about $x_{i,1}H_4(\mathsf{pk}_{i,2}) + x_{i,2}$ (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. The proof can be found in the appendix.

3. If the delegatee j is now a delegator to someone else (say k). Again, only $x_{j,1}H_4(\mathsf{pk}_{j,2})+x_{j,2}$ is known to a collusion of the delegatee k and a proxy, which is not useful in recovering the token h in $\mathsf{rk}_{i\to j}$, hence the chain collusion attack suffered by [15, 9] does not apply.

The chosen-ciphertext security of our scheme is asserted by the following theorem.

Theorem 1 *Our scheme is* IND-PRE-CCA *secure in the random oracle model, assuming the* CDH *assumption holds in group* $\mathbb{G}$ *and the Schnorr signature[22] is* EUF-CMA *secure. Concretely, if there exists an adversary* $\mathcal{A}$*, who asks at most* q_{H_i} *random oracle quires to* H_i *with* $i \in \{1,\dots,4\}$*, and breaks the* $(t, q_u, q_c, q_{rk}, q_{re}, q_d, \epsilon)$-IND-PRE-CCA *of our scheme, then, for any* $0 < \nu < \epsilon$*, there exists*

- *either an algorithm* $\mathcal{B}$ *which can break the* (t', ϵ')*-CDH assumption in* $\mathbb{G}$ *with*

$$t' \leq t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_u + q_c + q_{rk} + q_{re} + q_d)\mathcal{O}(1) \\ + (2q_u + 2q_c + 2q_{rk} + 5q_{re} + 2q_d + q_{H_1}q_{re} + (2q_{H_2} + 2q_{H_1})q_d)t_{\exp},$$

$$\epsilon' \geq \frac{1}{q_{H_2}}\Big(\frac{\epsilon-\nu}{e(1+q_{rk})} - \frac{q_{H_1}+q_{H_3}+(q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} - \frac{q_{re}+2q_d}{q}\Big),$$

  where $t_{\exp}$ *denotes the running time of an exponentiation in group* $\mathbb{G}$*.*
- *or an attacker who breaks the* EUF-CMA *security of the Schnorr signature with advantage* ν *within time* t'*.*

Proof. Without loss of generality, we assume that the Schnorr signature is (t', ν)-EUF-CMA secure for some probability $0 < \nu < \epsilon$. Since the CDH problem is equivalent to the DCDH problem, for convenience, here we show a reduction of DCDH problem. Specifically, suppose there exists a t-time adversary $\mathcal{A}$ who can break the IND-PRE-CCA security of our scheme with advantage $\epsilon - \nu$, then we show how to construct an algorithm $\mathcal{B}$ which can break the (t', ϵ')-DCDH assumption in $\mathbb{G}$, given as input a DCDH challenge tuple $(g, g^{\frac{1}{a}}, g^b)$.

To output g^{ab} eventually, algorithm $\mathcal{B}$ acts as the challenger and plays the IND-PRE-CCA game with adversary $\mathcal{A}$ in the following way.

Setup. Algorithm $\mathcal{B}$ gives $(q, \mathbb{G}, g, H_1, \dots, H_4, \ell_0, \ell_1)$ to $\mathcal{A}$. Here H_1, H_2, H_3 and H_4 are random oracles controlled by $\mathcal{B}$. $\mathcal{B}$ maintains four hash lists H_i^{list} with $i \in \{1,\dots,4\}$, 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 $H_3(D,E,F) = \gamma$.
- $H_4(\mathsf{pk})$: If this query has appeared on the H_4^{list} in a tuple (pk,τ), return the predefined value υ. Otherwise, choose $\upsilon \xleftarrow{\$} \mathbb{Z}_q^*$, add the tuple (pk,τ) to the list H_4^{list} and respond with $H_4(\mathsf{pk}) = \tau$.

Fig. 1. The Simulations for H_i for $i = 1,\dots,4$

Phase 1. Adversary $\mathcal{A}$ issues a series of queries as in the IND-PRE-CCA game. $\mathcal{B}$ maintains two lists K^{list} and R^{list} which are initially empty, and answers these queries for $\mathcal{A}$ as follows:

- *Uncorrupted key generation query.* $\mathcal{B}$ picks $x_{i,1}, x_{i,2} \xleftarrow{\$} \mathbb{Z}_q^*$. Next, using the Coron's technique [10], it flips a biased coin $c_i \in \{0,1\}$ that yields 1 with probability θ and 0 otherwise. If $c_i = 1$, it defines $\mathsf{pk}_i = (g^{x_{i,1}}, g^{x_{i,2}})$; else $\mathsf{pk}_i = (\mathsf{pk}_{i,1}, \mathsf{pk}_{i,2}) = \left(\left(g^{1/a}\right)^{x_{i,1}}, \left(g^{1/a}\right)^{x_{i,2}}\right)$. Next, it adds the tuple $(\mathsf{pk}_i, x_{i,1}, x_{i,2}, c_i)$ to K^{list} and returns pk_i.
- *Corrupted key generation query.* $\mathcal{B}$ picks $x_{j,1}, x_{j,2} \xleftarrow{\$} \mathbb{Z}_q^*$ and defines $\mathsf{pk}_j = (g^{x_{j,1}}, g^{x_{j,2}}), c_j =$ '$-$'. It then adds the tuple $(\mathsf{pk}_j, x_{j,1}, x_{j,2}, c_j)$ to K^{list} and returns $(\mathsf{pk}_j, (x_{j,1}, x_{j,2}))$.
- *Re-encryption key generation query* $\langle \mathsf{pk}_i, \mathsf{pk}_j \rangle$: If R^{list} has an entry for $(\mathsf{pk}_i, \mathsf{pk}_j)$, return the predefined re-encryption key to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ acts as follows:
  1. Recover tuples $(\mathsf{pk}_i, x_{i,1}, c_i)$ and $(\mathsf{pk}_j, x_{j,1}, c_j)$ from K^{list}.
  2. Pick $h \xleftarrow{\$} \{0,1\}^{\ell_0}, \pi \xleftarrow{\$} \{0,1\}^{\ell_1}$; compute $v = H_1(h, \pi), V = g^v, W = H_2(\mathsf{pk}_{j,2}^v) \oplus (h \| \pi)$.
  3. Construct the first component $\mathsf{rk}_{i \to j}^{\langle 1 \rangle}$ according to the following cases:
     - $c_i = 1$ or $c_i =$ '$-$': define $\mathsf{rk}_{i \to j}^{\langle 1 \rangle} = \frac{h}{x_{i,1} H_4(\mathsf{pk}_{i,2}) + x_{i,2}}$, and define $\tau = 1$.
     - $(c_i = 0 \ \wedge \ c_j = 1)$ or $(c_i = 0 \ \wedge \ c_j = 0)$: pick $\mathsf{rk}_{i \to j}^{\langle 1 \rangle} \xleftarrow{\$} \mathbb{Z}_q^*$, and define $\tau = 0$.
     - $(c_i = 0 \wedge c_j =$ '$-$'$)$: output "$\mathsf{failure}$" and **abort**.
  4. If $\mathcal{B}$ does not **abort**, add $(\mathsf{pk}_i, \mathsf{pk}_j, (\mathsf{rk}_{i \to j}^{\langle 1 \rangle}, V, W), h, \tau)$ into list R^{list}, return $(\mathsf{rk}_{i \to j}^{\langle 1 \rangle}, V, W)$.
- *Re-encryption query* $\langle \mathsf{pk}_i, \mathsf{pk}_j, \mathfrak{C}_i (= (D, E, F, s)) \rangle$: Parse pk_i as $\mathsf{pk}_i = (\mathsf{pk}_{i,1}, \mathsf{pk}_{i,2})$ and pk_j as $\mathsf{pk}_j = (\mathsf{pk}_{j,1}, \mathsf{pk}_{j,2})$. If $\left(\mathsf{pk}_{i,1}^{H_4(\mathsf{pk}_{i,2})} \mathsf{pk}_{i,2}\right)^s \neq D \cdot E^{H_3(D,E,F)}$, then return $\perp$. Otherwise:
  1. Recover tuples $(\mathsf{pk}_i, x_{i,1}, x_{i,2}, c_i)$ and $(\mathsf{pk}_j, x_{j,1}, x_{j,2}, c_j)$ from K^{list}.
  2. If $(c_i = 0 \ \wedge \ c_j =$ '$-$'$)$ does not hold, issue a re-encryption key generation query $\langle \mathsf{pk}_i, \mathsf{pk}_j \rangle$ to obtain $\mathsf{rk}_{i \to j}$, and then return $\mathsf{ReEncrypt}(\mathsf{rk}_{i \to j}, \mathfrak{C}_i, \mathsf{pk}_j)$ to $\mathcal{A}$.
  3. Else, search for the tuple $(m, \omega, r) \in H_1^{\text{list}}$ such that $(\mathsf{pk}_{i,1}^{H_4(\mathsf{pk}_{i,2})} \mathsf{pk}_{i,2})^r = E$. If there exists no such tuple, return $\perp$. Otherwise, choose $h \xleftarrow{\$} \{0,1\}^{\ell_0}, \pi \xleftarrow{\$} \{0,1\}^{\ell_1}$ and compute $v = H_1(h, \pi), V = g^v$ and $W = H_2(\mathsf{pk}_{j,2}^v) \oplus (h \| \pi)$. Finally, define $E' = g^{r \cdot h}$, and return (E', F, V, W) to $\mathcal{A}$. E' is correctly computed as long as r can be retrieved. (This corresponds to the event REErr to be explained).
- *Decryption query* $\langle \mathsf{pk}_i, \mathfrak{C}_i \rangle$: $\mathcal{B}$ first parse $\mathsf{pk}_i = (\mathsf{pk}_{i,1}, \mathsf{pk}_{i,2})$ and recovers tuple $(\mathsf{pk}_i, x_{i,1}, x_{i,2}, c)$ from K^{list}. If $c = 1$ or $c =$ '$-$', 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 $\left(\mathsf{pk}_{i,1}^{H_4(\mathsf{pk}_{i,2})} \mathsf{pk}_{i,2}\right)^s \neq D \cdot E^{H_3(D,E,F)}$, 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 exist $(m, \omega, r) \in H_1^{\text{list}}$ and $(R, \beta) \in H_2^{\text{list}}$ such that

    $$\left(\mathsf{pk}_{i,1}^{H_4(\mathsf{pk}_{i,2})} \mathsf{pk}_{i,2}\right)^r = E, \beta \oplus (m \| \omega) = F \ \text{ and } \ R = g^r.$$

    If yes, return m to $\mathcal{A}$. Otherwise, return $\perp$.
  - $\mathfrak{C}_i$ is a transformed ciphertext $\mathfrak{C}_i = (E', F, V, W)$ re-encrypted: Algorithm $\mathcal{B}$ recovers the tuple $(\mathsf{pk}_i, x_{i,1}, x_{i,2}, c)$ from K^{list}, then responds according to the following cases:
    * If there exist a tuple $(\mathsf{pk}_j, \mathsf{pk}_i, (\mathsf{rk}^{\langle 1 \rangle}, V, W), h, 0)$ in R^{list}: Compute $E = E'^{\frac{1}{\mathsf{rk}^{\langle 1 \rangle}}}$. Search lists H_1^{list} and H_2^{list} to see whether there exist $(m, \omega, r) \in H_1^{\text{list}}$ and $(R, \beta) \in H_2^{\text{list}}$ such that $\left(\mathsf{pk}_{j,1}^{H_4(\mathsf{pk}_{j,2})} \mathsf{pk}_{j,2}\right)^r = E, \beta \oplus (m \| \omega) = F$ and $R = g^r$. If yes, return m to $\mathcal{A}$, else return $\perp$. Note that all V, W values from R^{list} are correctly generated.

* Otherwise: Search lists H_1^{list} and H_2^{list} to see whether there exist $(m, \omega, r), (h, \pi, v) \in H_1^{\text{list}}$ and $(R, \beta), (R', \beta') \in H_2^{\text{list}}$ such that

$$g^v = V, \beta' \oplus (h \| \pi) = W, g^{r \cdot h} = E', \beta \oplus (m \| \omega) = F, R = g^r \text{ and } R' = \mathsf{pk}_{i,2}^v.$$

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 $\mathsf{pk}_{i^*} = (\mathsf{pk}_{i^*,1}, \mathsf{pk}_{i^*,2})$ and two equal-length messages $m_0, m_1 \in \{0,1\}^{\ell_0}$. Algorithm $\mathcal{B}$ responds as follows:

1. Recover tuple $(\mathsf{pk}_{i^*}, x_{i^*,1}, x_{i^*,2}, c^*)$ from K^{list}. Note that according to the constraints described in IND-PRE-CCA game, c^* must be equal to 1 or 0. If $c^* = 1$, $\mathcal{B}$ outputs "$\mathsf{failure}$" and **abort**. Otherwise, it means that $c^* = 0$, and $\mathcal{B}$ proceeds to execute the rest steps.
2. Pick $e^*, s^* \stackrel{\$}{\leftarrow} \mathbb{Z}_q^*$, and compute $D^* = \left(g^b\right)^{-(x_{i^*,1} H_4(\mathsf{pk}_{i^*,2}) + x_{i^*,2}) e^*} \left(g^{\frac{1}{a}}\right)^{(x_{i^*,1} H_4(\mathsf{pk}_{i^*,2}) + x_{i^*,2}) s^*}$ and $E^* = \left(g^b\right)^{x_{i^*,1} H_4(\mathsf{pk}_{i^*,2}) + x_{i^*,2}}$.
3. Pick $F^* \stackrel{\$}{\leftarrow} \{0,1\}^{\ell_0 + \ell_1}$ and define $H_3(D^*, E^*, F^*) = e^*$.
4. Pick $\delta \stackrel{\$}{\leftarrow} \{0,1\}, \omega^* \stackrel{\$}{\leftarrow} \{0,1\}^{\ell_1}$, and implicitly define $H_2(g^{ab}) = (m_\delta \| \omega^*) \oplus F^*$ and $H_1(m_\delta, \omega^*) = ab$ (Note that algorithm $\mathcal{B}$ knows neither ab nor g^{ab}).
5. Return $\mathfrak{C}^* = (D^*, E^*, F^*, s^*)$ as the challenged ciphertext to adversary $\mathcal{A}$.

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

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

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 $\mathcal{B}$. Algorithm $\mathcal{B}$ randomly picks a tuple (R, β) from the list H_2^{list} and outputs R as the solution to the given DCDH instance.

Analysis. The main idea of our analysis is borrowed from [3, 11]. 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 $\mathcal{A}$ 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 $\Pr[\mathsf{AskH}_3^*] = \frac{q_{H_3}}{2^{\ell_0+\ell_1}}$. 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 $\mathcal{A}$'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 $\theta_{\mathrm{opt}} = \frac{q_{rk}}{1+q_{rk}}$. 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 $\mathsf{rk}'_{i\to j}$ is randomly chosen. If event Abort does not happen, this is computationally indistinguishable from the real world according to the following facts. First, the secret key sk_j is unknown to $\mathcal{A}$ since $c_j \neq$ '$-$'. Second, $(g^v, H_2(\mathsf{pk}_{j,2}^v) \oplus (h\|\pi))$ with $v = H_1(h,\pi)$ is in fact an encryption of h under $\mathsf{pk}_{j,2}$ using the "hashed" ElGamal encryption scheme [13, 12, 3]. So, if $\mathcal{A}$ can distinguish $\mathsf{rk}'_{i\to j}$ from $\mathsf{rk}_{i\to j}$, it means that $\mathcal{A}$ can determine $(g^v, H_2(\mathsf{pk}_{j,2}^v) \oplus (h\|\pi))$ with $v = H_1(h,\pi)$ is an encryption of h or h'. Since $\mathcal{B}$ can implant the two given messages in the CCA2 game of the "hashed" ElGamal as the responses to the random oracle queries, this breaks the security of the "hashed" ElGamal, which is based on the CDH assumption. Therefore, if event Abort does not happen, the simulation of the re-encryption key queries is the same as the real one.

Next, we analyze the simulation of the re-encryption queries. This simulation is also perfect, unless $\mathcal{A}$ 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 $\Pr[\mathsf{REErr}] \leq \frac{q_{\mathrm{re}}}{q}$.

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

$$\begin{aligned}\Pr[\mathsf{Valid}|\neg\mathsf{AskH}_2] &= \Pr[\mathsf{Valid} \wedge \mathsf{AskH}_1|\neg\mathsf{AskH}_2] + \Pr[\mathsf{Valid} \wedge \neg\mathsf{AskH}_1|\neg\mathsf{AskH}_2] \\ &\leq \Pr[\mathsf{AskH}_1|\neg\mathsf{AskH}_2] + \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_1 \wedge \neg\mathsf{AskH}_2] \leq \frac{q_{H_1}}{2^{\ell_0+\ell_1}} + \frac{1}{q},\end{aligned}$$

and similarly we have $\Pr[\mathsf{Valid}|\neg\mathsf{AskH}_1] \leq \frac{q_{H_2}}{2^{\ell_0+\ell_1}} + \frac{1}{q}$. Thus we have

$$\Pr[\mathsf{Valid}|(\neg\mathsf{AskH}_1 \vee \neg\mathsf{AskH}_2)] \leq \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_1] + \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_2] \leq \frac{q_{H_1}+q_{H_2}}{2^{\ell_0+\ell_1}} + \frac{2}{q}.$$

Let DErr be the event that $\mathsf{Valid}|(\neg\mathsf{AskH}_1 \vee \neg\mathsf{AskH}_2)$ happens during the entire simulation. Then, since $\mathcal{A}$ 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 Good denote the event $\left(\mathsf{AskH}_2^* \vee (\mathsf{AskH}_1^*|\neg\mathsf{AskH}_2^*) \vee \mathsf{AskH}_3^* \vee \mathsf{REErr} \vee \mathsf{DErr}\right)\big|\neg\mathsf{Abort}$. If Good does not happen, due to the randomness of the output of the random oracle H_2, it is clear that adversary $\mathcal{A}$ cannot gain any advantage greater than $\frac{1}{2}$ in guessing δ. Namely, we have $\Pr[\delta = \delta'|\neg\mathsf{Good}] = \frac{1}{2}$. Hence, by splitting $\Pr[\delta' = \delta]$, we have

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

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

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

Since $\Pr[\mathsf{AskH}_1^*|\neg\mathsf{AskH}_2^*] \le \frac{q_{H_1}}{2^{\ell_0+\ell_1}}$, $\mathsf{AskH}_3^* \le \frac{q_{H_3}}{2^{\ell_0+\ell_1}}$, $\Pr[\mathsf{DErr}] \le \frac{(q_{H_1}+q_{H_2})q_d}{2^{\ell_0+\ell_1}} + \frac{2q_d}{q}$, $\Pr[\mathsf{REErr}] \le \frac{q_{\mathrm{re}}}{q}$ and $\Pr[\neg\mathsf{Abort}] \ge \frac{1}{e(1+q_{rk})}$, we obtain

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

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

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

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

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

□

4.3 Comparisons

In Table 2, we compare our scheme with Libert-Vergnaud's scheme [19] ($\mathcal{LV}08$) and Shao-Cao's scheme [23] ($\mathcal{SC}09$). We denote $t_{\hat{e}}, t_{\exp}, t_{\mathrm{sig}}$ and t_{ver} as the computational cost of a pairing, an exponentiation (over $\mathbb{G}_1$ of elliptic curve or G_T[11] in $\mathcal{LV}08$, over $\mathbb{Z}_{N^2}^*$ in $\mathcal{SC}09$, and over $\mathbb{G}$ in our scheme), signing and verifying a one-time signature, respectively. In our calculation, a multi-exponentiation (m-exp) (which we assume it multiplies only up to 3 exponentiations in one shot) is considered as 1.5 $t_{\exp}$. $\mathsf{Encrypt}$ of $\mathcal{LV}08$, $\mathsf{ReEncrypt}$ and $\mathsf{Decrypt}(\mathfrak{C})$ of $\mathcal{SC}09$ used 1, 2 and 2 m-exp respectively. In our scheme, we assume $\mathsf{pk}_{i,1}^{H_4(\mathsf{pk}_{i,2})}\mathsf{pk}_{i,2}$ 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.

We use $\mathfrak{C}$ to denote an original ciphertext and $\mathfrak{C}'$ to denote a transformed ciphertext, $|\mathfrak{C}|$ and $|\mathfrak{C}'|$ are their size. For $\mathcal{LV}08$, $\mathbb{G}_1$ and $\mathbb{G}_T$ are defined such that $\hat{e} : \mathbb{G}_1 \times G_1 \to G_T$, svk and σ denote the public key and the signature of the one-time signature respectively. For $\mathcal{SC}09$, N_X (N_Y) is the safe-prime modulus used by the delegator (delegatee).

The comparison results indicate that our scheme beats $\mathcal{SC}09$ in all aspects, and $\mathcal{LV}08$ except the reliance of random oracle. The CCA-security of our scheme is based on the standard and well-studied CDH assumption, while $\mathcal{LV}08$ is proved to be RCCA-secure under a stronger and less-studied 3-quotient decision bilinear Diffie-Hellman (3-QDBDH) assumption. Compared with $\mathcal{LV}08$ and $\mathcal{SC}09$, our re-encryption mechanism is more naturally designed.

[11] $\mathbb{G}_T$ is usually a subgroup of $\mathbb{Z}_{q^\alpha}$, which is vulnerable to sub-exponential discrete logarithm attacks, and needs very large representation. For example, for 128 bits security, $|\mathbb{G}_T| \ge 3072$ bits.

[12] In [19], one can test whether pk is the original delegator by checking if $\hat{e}(C_2', C_2'') = \hat{e}(\mathsf{pk}, g)$.

	LV08 [19]	*SC09* [23]	Our Scheme
Encrypt	$t_{\mathrm{sig}} + 2.5t_{\mathrm{exp}}$ (in $\mathbb{G}_1$) + t_{exp} (in $\mathbb{G}_T$)	$5t_{\mathrm{exp}}$ (in $\mathbb{Z}_{N^2}$)	$3t_{\mathrm{exp}}$ (in $\mathbb{G}$)
ReEncrypt	$2t_{\hat{e}} + t_{\mathrm{ver}} + 4t_{\mathrm{exp}}$ (in $\mathbb{G}_1$)	$4t_{\mathrm{exp}}$ (in $\mathbb{Z}_{N^2}$)	$2.5t_{\mathrm{exp}}$ (in $\mathbb{G}$)
Decrypt($\mathfrak{C}$)	$3t_{\hat{e}} + t_{\mathrm{ver}} + t_{\mathrm{exp}}$ (in $\mathbb{G}_1$) + t_{exp} (in $\mathbb{G}_T$)	$5t_{\mathrm{exp}}$ (in $\mathbb{Z}_{N^2}$)	$3.5t_{\mathrm{exp}}$ (in $\mathbb{G}$)
Decrypt($\mathfrak{C}'$)	$5t_{\hat{e}} + t_{\mathrm{ver}} + t_{\mathrm{exp}}$ (in $\mathbb{G}_1$) + t_{exp} (in $\mathbb{G}_T$)	$4t_{\mathrm{exp}}$ (in $\mathbb{Z}_{N^2}$)	$4t_{\mathrm{exp}}$ (in $\mathbb{G}$)
$\lvert\mathfrak{C}\rvert$	$\lvert svk\rvert + \lvert\sigma\rvert + 2\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$2k + 3\lvert(N_X)^2\rvert + \lvert m\rvert$	$3\lvert\mathbb{G}\rvert + \lvert\mathbb{Z}_q\rvert$
$\lvert\mathfrak{C}'\rvert$	$\lvert svk\rvert + \lvert\sigma\rvert + 4\lvert\mathbb{G}_1\rvert + \lvert\mathbb{G}_T\rvert$	$\ell_2 + 3\lvert(N_X)^2\rvert + 2\lvert(N_Y)^2\rvert + \lvert m\rvert$	$2\lvert\mathbb{G}\rvert + 2\lvert\mathbb{Z}_q\rvert$
Security	RCCA-Secure	CCA-Secure?	CCA-Secure
Assumption	3-QDBDH	DDH	CDH
RO-Free	✓	×	×
Nature of ReEncrypt	$\mathfrak{C}'$ contains information about the delegator[12]	Decryption of $\mathfrak{C}'$ requires pk_X of the delegator	No trace of the delegator is in $\mathfrak{C}'$

Table 2. Comparisons of Unidirectional Proxy Re-Encryption Schemes

5 Conclusions

Most existing unidirectional proxy re-encryption schemes rely on pairing except a recently proposed scheme by Shao and Cao [23]. 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 proxy re-encryption scheme without pairing that is very efficient. We remark that our schemes are single-hop and is proved only in the random oracle model. It would be interesting to construct a PRE scheme which is multi-hop, CCA-secure in the standard model, and yet without pairings.

Acknowledgement

We would like to thank Jun Shao for a helpful discussion of the attack.

References

1. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. In *NDSS*. The Internet Society, 2005.
2. Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.
3. Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Certificateless Public Key Encryption Without Pairing. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *ISC*, volume 3650 of *Lecture Notes in Computer Science*, pages 134–148. Springer, 2005.
4. Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of Diffie-Hellman Problem. In Sihan Qing, Dieter Gollmann, and Jianying Zhou, editors, *ICICS*, volume 2836 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2003.
5. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible Protocols and Atomic Proxy Cryptography. In *EUROCRYPT*, pages 127–144, 1998.
6. Emmanuel Bresson, Dario Catalano, and David Pointcheval. A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications. In Chi-Sung Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 37–54. Springer, 2003.
7. Ran Canetti and Susan Hohenberger. Chosen-Siphertext Cecure Proxy Re-Encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 185–194. ACM, 2007.
8. Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing Chosen-Ciphertext Security. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2003.

9. Cheng-Kang Chu and Wen-Guey Tzeng. Identity-Based Proxy Re-encryption Without Random Oracles. In Juan A. Garay, Arjen K. Lenstra, Masahiro Mambo, and René Peralta, editors, *ISC*, volume 4779 of *Lecture Notes in Computer Science*, pages 189–202. Springer, 2007.
10. Jean-Sébastien Coron. On the Exact Security of Full Domain Hash. In Mihir Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 2000.
11. Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen. Chosen-Ciphertext Secure Proxy Re-encryption without Pairings. In Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong, editors, *CANS*, volume 5339 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.
12. Eiichiro Fujisaki and Tatsuaki Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.
13. Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO*, pages 10–18, 1984.
14. Philippe Golle, Markus Jakobsson, Ari Juels, and Paul F. Syverson. Universal Re-encryption for Mixnets. In Tatsuaki Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2004.
15. Matthew Green and Giuseppe Ateniese. Identity-Based Proxy Re-encryption. In Jonathan Katz and Moti Yung, editors, *ACNS*, volume 4521 of *Lecture Notes in Computer Science*, pages 288–306. Springer, 2007.
16. Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely Obfuscating Re-encryption. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 233–252. Springer, 2007.
17. Anca-Andreea Ivan and Yevgeniy Dodis. Proxy Cryptography Revisited. In *NDSS*. The Internet Society, 2003.
18. Markus Jakobsson. On Quorum Controlled Asymmetric Proxy Re-encryption. In Hideki Imai and Yuliang Zheng, editors, *Public Key Cryptography*, volume 1560 of *Lecture Notes in Computer Science*, pages 112–121. Springer, 1999.
19. Benoît Libert and Damien Vergnaud. Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption. In Ronald Cramer, editor, *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2008.
20. 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.
21. Toshihiko Matsuo. Proxy Re-encryption Systems for Identity-Based Encryption. In Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, editors, *Pairing*, volume 4575 of *Lecture Notes in Computer Science*, pages 247–267. Springer, 2007.
22. Claus-Peter Schnorr. Efficient Identification and Signatures for Smart Cards. In Gilles Brassard, editor, *CRYPTO*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.
23. Jun Shao and Zhenfu Cao. CCA-Secure Proxy Re-encryption without Pairings. In Stanislaw Jarecki and Gene Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 357–376. Springer, 2009.
24. Tony Smith. DVD Jon: Buy DRM-less Tracks from Apple iTunes. Available online at http://www.theregister.co.uk/2005/03/18/itunes_pymusique, jan 2005.

A Delegator Secret Security

A.1 Definition

Delegator secret security is formally defined via the following game:

Setup. Challenger $\mathcal{C}$ runs $\mathsf{Setup}(1^\kappa)$ and gives the global parameters 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 KeyGen to obtain a public/private key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$, and then sends pk_i to $\mathcal{A}$.
- *Corrupted key generation query*: $\mathcal{C}$ first runs KeyGen to obtain a public/private key pair $(\mathsf{pk}_j, \mathsf{sk}_j)$, and then gives $(\mathsf{pk}_j, \mathsf{sk}_j)$ to $\mathcal{A}$.

– *Re-encryption key query* $\langle \mathsf{pk}_i, \mathsf{pk}_j \rangle$: $\mathcal{C}$ runs $\mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$ to generate a re-encryption key $\mathsf{rk}_{i\to j}$ and returns it to $\mathcal{A}$. Here sk_i is the private key with respect to pk_i. It is required that pk_i and pk_j were generated beforehand by algorithm KeyGen.

Output. Finally, $\mathcal{A}$ outputs a private key sk_{i^*} with respect to the public key pk_{i^*}. $\mathcal{A}$ wins the game if sk_{i^*} is indeed a valid private key and $\mathcal{A}$ has never issue the corrupted key generation query on $\langle i^* \rangle$ (i.e., $\mathcal{A}$ issue the uncorrupted key generation query on $\langle i^* \rangle$).

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 4. *We say that a PRE scheme is* $(t, q_u, q_c, q_{rk}, \epsilon)$-DSK *secure, if for any* t*-time DSK adversary* $\mathcal{A}$ *that makes at most* q_u *uncorrupted key generation queries, at most* q_c *corrupted key generation queries and at most* q_{rk} *re-encryption key queries,* $\mathrm{Adv}^{\mathrm{DSK}}_{PRE,\mathcal{A}} \leq \epsilon$.

A.2 Analysis

The delegator secret security of our scheme is based on the discrete logarithm problem (DLP).

Definition 5. *The DLP in* $\mathbb{G}$ *is, given a tuple* $(g, g^a) \in \mathbb{G}^2$ *with unknown* a*, to compute* a*.*

For a polynomial-time algorithm $\mathcal{B}$*, we define his advantage in solving the DLP in* $\mathbb{G}$ *as* $Pr[\mathcal{B}(g, g^a) = a]$*, 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 DLP in* $\mathbb{G}$*.*

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

Proof. Suppose $\mathcal{B}$ is given as input a DLP challenge tuple $(g, g^a) \in \mathbb{G}^2 \times \mathbb{G}_T$ 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 define $\mathsf{pk}_i = (\mathsf{pk}_{i,1}, \mathsf{pk}_{i,2}) = \Big((g^a)^{1/H_4(\mathsf{pk}_{i,2})} \cdot g^{x_{i,1}}, g^{x_{i,2}}/g^a\Big)$. Next, set $c_i = 0$ and add the tuple $(\mathsf{pk}_i, x_{i,1}, x_{i,2}, c_i)$ to the K^{list}. Finally, it returns pk_i to adversary $\mathcal{A}$. The private key with respect to pk_i is $\mathsf{sk}_i = (\frac{a}{H_4(\mathsf{pk}_{i,2})} + x_{i,1}, -a + x_{i,2})$, 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 $\mathsf{pk}_j = (g^{x_{j,1}}, g^{x_{j,2}})$ and $c_j = 1$. It then adds the tuple $(\mathsf{pk}_j, x_{j,1}, x_{j,2}, c_j)$ to the K^{list} and returns $(\mathsf{pk}_j, (x_{j,1}, x_{j,2}))$.

– *Re-encryption key query* $\langle \mathsf{pk}_i, \mathsf{pk}_j \rangle$: $\mathcal{B}$ parses pk_i as $\mathsf{pk}_i = (\mathsf{pk}_{i,1}, \mathsf{pk}_{i,2})$ and $\mathsf{pk}_j = (\mathsf{pk}_{j,1}, \mathsf{pk}_{j,2})$. Next, it recovers tuples $(\mathsf{pk}_i, x_{i,1}, x_{i,2}, c_i)$ and $(\mathsf{pk}_j, x_{j,1}, x_{j,2}, c_j)$ from the K^{list}. Then, it constructs the re-encryption key $\mathsf{rk}_{i\to j}$ for adversary $\mathcal{A}$ according to the following situations:

- If $c_i = 1$, $\mathcal{B}$ return the result of $\mathsf{ReKeyGen}(\mathsf{sk}_i, \mathsf{pk}_j)$ to $\mathcal{A}$ since $\mathsf{sk}_i = (x_{i,1}, x_{i,2})$ is known.
- If $c_i = 0$, it means that $\mathsf{sk}_i = (\frac{a}{H_4(\mathsf{pk}_{i,2})} + x_{i,1}, -a + x_{i,2})$. $\mathcal{B}$ picks $h \xleftarrow{\$} \{0,1\}^{\ell_0}, \pi \xleftarrow{\$} \{0,1\}^{\ell_1}$ and returns $\mathsf{rk}_{i \to j} = (\mathsf{rk}^{\langle 1 \rangle}_{i \to j} = \frac{h}{x_{i,1} H_4(\mathsf{pk}_{i,2}) + x_{i,2}}, V = g^{H_1(h,\pi)}, W = H_2(\mathsf{pk}^v_{j,2}) \oplus (h \| \pi))$, which is valid since $x_{i,1} H_4(\mathsf{pk}_{i,2}) + x_{i,2} = (\frac{a}{H_4(\mathsf{pk}_{i,2})} + x_{i,1}) H_4(\mathsf{pk}_{i,2}) + (-a + x_{i,2})$.

Output. Eventually, $\mathcal{A}$ outputs the private key $\mathsf{sk}_{i^*} = (\mathsf{sk}_{i^*,1}, \mathsf{sk}_{i^*,2})$ with respect to the public key pk_{i^*}. $\mathcal{B}$ recovers the tuple $(\mathsf{pk}_{i^*}, x_{i^*,1}, x_{i^*,2}, c_{i^*})$ from the K^{list} (Note that according to the restriction specified in the DSK game, we have $c_{i^*} = 0$), and then outputs $x_{i^*,2} - \mathsf{sk}_{i^*,2}$ as the solution to the DLP challenge. Note that, if $\mathsf{sk}_{i^*} = (\mathsf{sk}_{i^*,1}, \mathsf{sk}_{i^*,2})$ is a valid private key with respect to pk_{i^*}, we have $\mathsf{sk}_{i^*,1} = \frac{a}{H_4(\mathsf{pk}_{i^*,2})} + x_{i^*,1}$ and $\mathsf{sk}_{i^*,2} = -a + x_{i^*,2}$.

It can be verified that the responses for the key generation queries and the re-encryption key query are perfect. Thus, when adversary $\mathcal{A}$ outputs the valid private key sk_{i^*} with advantage ϵ, $\mathcal{B}$ can resolve the DLP with the same advantage. It can be easily seen that $\mathcal{B}$'s running time is bounded by $t' \leq t + \mathcal{O}(2q_u t_{\exp} + 2q_c t_{\exp} + 2q_{rk} t_{\exp})$. □