12-2010

Chosen-ciphertext secure bidirectional proxy re-encryption schemes without pairings

Jian Weng

Huijie, Robert DENG
Singapore Management University, robertdeng@smu.edu.sg

Shengli Liu

Kefei Chen

Chosen-Ciphertext Secure Proxy Re-Encryption Schemes without Pairings

Abstract

Proxy re-encryption realizes delegation of decryption rights, enabling a proxy holding a re-encryption key to convert a ciphertext originally intended for Alice into an encryption of the same message for Bob. Proxy re-encryption is a very useful primitive, having many applications in distributed file systems, outsourced filtering of encrypted spam, access control over network storage, and so on. Lately, Weng *et al.* proposed the first unidirectional proxy re-encryption scheme without using bilinear pairs. However, Weng et al.'s construction does not possess collusion resilience, in the sense that a coalition of the proxy and the delegatee can recover the delegator's private key. This is a serious weakness, since a user's private key should be strictly not revealed in any case. In this work, we present a scheme solving this problem, based on Weng et al.'s construction. We further extend our scheme to address several drawbacks inherent in virtually all existing proxy re-encryption schemes.

Keywords: Proxy re-encryption, bilinear pairing, chosen-ciphertext security.

1 Introduction

Proxy re-encryption allows a proxy possessing a re-encryption key to convert a ciphertext originally computed under Alice's public key into an one that can be opened by Bob using his own private key. The proxy performs the conversion without learning anything on the underlying plaintext and either of private keys of Alice and Bob. In this context, Alice is *delegator* and Bob is the *delegatee*. Proxy re-encryption is a very useful primitive, having many applications. Suppose for example, Alice, the manager of the company, is to be absent from office the whole morning for attending a meeting. She likes to entrust Bob, the vice manager, to temporarily take over in processing her (encrypted) official emails during her absence. By employing proxy re-encryption, Alice can configure the email server for her official account in such a way that the server is the proxy, and Alice gives it a re-encryption key; the server then converts Alice's incoming emails using the re-encryption key, and forwards the transformed ones to Bob. The use of proxy re-encryption is clearly more advantageous than the naive method that Alice directly gives her private key to Bob, who then in turn decrypts with Alice's private key.

1.1 Related Work

Proxy re-encryption evolves from the idea of delegating decryption rights, which was first proposed by Mambo and Okamoto [25] as a better-performance alternative to the trivial approach of decrypt-then-encrypt. Shortly after that, Blaze, Bleumer and Strauss [4] furthered this line of research by proposing the notion of "atomic proxy cryptography", which is eventually developed into proxy re-encryption. In their Elgamal-based scheme, where a

user's key pair is of the form ($pk = g^x \mod p, sk = x$), the re-encryption key $rk_{A\rightarrow B}$ for the proxy is $x_B/x_A \pmod q$; therefore, a ciphertext for Alice ($g^{x_A \cdot r} \mod p, m.g^r \mod p$) can be transformed into $((g^{x_A \cdot r})^{x_B/x_A} \mod p, m.g^r \mod p)$ for Bob. This scheme however has several drawbacks. First, it is inherently *bidirectional* in the sense that the delegation key $x_B/x_A \pmod q$ can also be used to convert ciphertext from Bob to Alice. Note that bidirectional proxy re-encryption is not always desirable, as mutual delegation is not necessarily needed in many applications. Second, delegation in their scheme is *transitive*: from $x_B/x_A \pmod q$ and $x_C/x_B \pmod q$, the proxy by itself can compute $x_C/x_A \pmod q$. Third, the scheme is not *collusion resilient*: if the proxy and Bob collude, they can compute Alice's private key x_A. Last, the generation of the re-encryption key $x_B/x_A \pmod q$ seems having to be *interactive* by involving Bob in the process.

Dodis and Ivan [12] were the first to realize *unidirectional* delegation of decryption. The basic idea for their ElGaml-based scheme is to partition the private key of Alice into two shares, and one is given to the proxy and the other to Bob. Their scheme is not "pure" proxy re-encryption, as decryption by Bob needs the share from Alice. Moreover, their scheme is not *key optimal*, due to the fact that the number of secrets held by Bob grows with the number of delegations he accepts. Ateniese *et al.* [1, 2] presented novel unidirectional proxy re-encryption schemes based on bilinear pairings. Their schemes are non-transitive, collusion resilient, non-interactive, and key optimal, among others. However, Ateniese *et al.*'s schemes are only secure against chosen-plaintext attack (CPA). Canetti and Hohenberger [10] gave a construction of CCA-secure bidirectional proxy re-encryption scheme. Subsequently, Libert and Vergnaud [21] presented a unidirectional PRE scheme with CCA security. Both of these constructions use bilinear pairings.

Proxy re-encryption has also been studied in the identity-based public key setting. Green and Ateniese [16] proposed the first identity-based proxy re-encryption schemes, with both CPA and CCA security in the random oracle model. They were succeeded by Chu and Tzeng [11], who gave constructions of CPA and CCA-secure identity-based proxy re-encryption schemes without random oracles.

All the above CCA-secure proxy re-encryption schemes are based on bilinear pairings. Lately, Weng *et al.* [13] were thus motivated to construct CCA-secure proxy re-encryption without using pairings. They proposed both unidirectional and bidirectional schemes. Considering the fact that pairing operation is still computationally costly, despite the recent advances in implementation techniques, Weng et al.'s constructions are of particular interest, especially in the resource limited setting.

1.2 Desirable features of Proxy Re-Encryption

Ateniese *et al.* [1, 2] summarized a number of desirable features that proxy re-encryption should possess. We reiterate them below, for a further understanding of proxy re-encryption.

- ♢ *Unidirectional*: Delegation from $A \rightarrow B$ does not allow re-encryption from $B \rightarrow A$. We notice that bidirectional delegation may be required in some applications, it is thus better to distinguish between unidirectional and bidirectional proxy re-encryption.
- ♢ *Noninteractive*: The generation of re-encryption key should be accomplished by Alice using Bob's public key; no interaction involving Bob is required.
- ♢ *Proxy invisibility*: It does not require the sender who sends message to Alice to be aware of the existence of the proxy. The same should hold for Bob, the delegatee.

- ♢ *Key optimality*: The number of secrets Bob needs to hold should remain constant, regardless of how many delegations he accepts.
- ♢ *Collusion resilience*: It is hard for the coalition of the proxy and Bob to compute Alice's private key.
- ♢ *Nontransitive*: The proxy, alone cannot redelegate decryption rights. Specifically, it should be hard for the proxy by itself to compute $rk_{A\to C}$ from $rk_{A\to B}$ and $rk_{B\to C}$.
- ♢ *Nontransferability*: It should be hard for the coalition of the proxy and the delegatee to redelegate decryption rights. It should note, however, that probably the best we can expect on nontransferability is that the proxy and the delegatee cannot generate $rk_{A\to C}$ from $rk_{A\to B}$ and $rk_{B\to C}$. The reason is that the following scenario is unavoidable in any proxy re-encryption scheme: suppose Alice delegates to Bob; then Bob decrypts the converted messages and gives the plaintext to Carol. In this case, Bob is required to be online in redelegating decryption rights. In contrast, Bob is offline if $rk_{A\to C}$ can be generated from $rk_{A\to B}$ and $rk_{B\to C}$. Anyway, in either case transferability implies Bob's willingness to share his privileges (i.e., access to Alice's messages) with Carol. Arguably, this does not often occur in practical applications.

1.3 Our Contributions

As mentioned earlier, Weng *et al.*'s schemes are the only CCA-secure proxy re-encryption in the literature that does not use bilinear pairings. However, their schemes do not satisfy collusion resilience; the proxy and the delegatee together can recover the delegator's private key. Collusion resilience is certainly an important feature, as exposure of private key in any way is fatal; security of the delegator against coalition of the delegatee and proxy should be one of the pursued goals in unidirectional proxy re-encryption [1, 2]. In this work, we propose an improved CCA-secure unidirectional proxy re-encryption scheme without using pairings, that satisfies all the above features except for nontransferability. We base our construction upon Weng et al.'s, and the two have similar efficiency. We also briefly discuss how to construct bidirectional proxy re-encryption.

We further observe the following weaknesses in the existing proxy re-encryption models (thus all of the existing schemes suffer from these weaknesses). First, the proxy feels free to share the re-encryption key with Bob, such that Bob by himself can convert and access Alice's messages. The traceable proxy re-encryption system proposed by Libert and Vergnaud [23] expects to deter the proxy from sharing re-encryption key, since otherwise the proxy can be identified by the delegator. Such a "trace-after-incident" strategy is not very effective, and a more satisfactory approach should be "prevention-before-incident". Second, it is hard to attain *change of delegation path*. For example, Alice originally delegates to Bob through proxy 1; but later she wants to give up that delegation, and instead delegate to Carol via proxy 2. In principle, the general "time limited delegation" approach can mitigate this problem to some extent, such that delegation is temporary and a re-encryption key is valid only for a short period of time [1, 2]. This approach forces all users to periodically reestablish re-encryption keys with their proxies, which is rather cumbersome. More importantly, it cannot entirely solve the problem, as change of delegation path is still impossible within a period.

In view of this, we extend our basic scheme to address the above weaknesses. Our extended scheme assumes a slightly different setting than standard proxy re-encryption: the proxy also has a key pair. This, however, is not a unreasonable assumption, since the proxy is anyhow an entity (like the delegator and the delegatee) within a PKI. In Figure 1,

we compare our schemes with Ateniese *et al.*'s scheme [1,2], Libert and Vergnaud's scheme [21], and Weng et al.'s scheme [13], which are typical of pairings based CPA-secure, pairings based CCA-secure proxy re-encryption, and CCA-secure proxy re-encryption without pairings, respectively.

	Ateniese *et al.* [1,2]	Libert *et al.* [21]	Weng *et al.* [13]	Ours
Nointeractive	Yes	Yes	Yes	Yes
Proxy invisibility	Yes	Yes	Yes	Yes
Key optimality	Yes	Yes	Yes	Yes
Collusion resilience	Yes	Yes	No	Yes
Nontransitive	Yes	Yes	Yes	Yes
Nontransferability	Yes*	Yes*	No	No
Re-enc. Key sharing	No	No	No	Solved
Change of deleg. path	No	No	No	Yes

* No scheme can attain full nontransferability. By "Yes", it achieves that $rk_{A->C}$ cannot be computed from $rk_{A->B}$ and $rk_{B->C}$

Figure 1: Comparison results

1.4 Outline

The rest of the paper is organized as follows. In Section 2, we review the model of unidirectional proxy re-encryption, as well as the complexity assumptions that will be used in our security proof. In Section 3, we present a unidirectional proxy re-encryption scheme without parings, and prove its security, and a bidirectional scheme is breifly introduced in Section 4. We give an extended scheme in Section 5, followed by the concluding remarks in Section 6.

2 Preliminaries

We review the formal model of unidirectional proxy re-encryption.

2.1 Model of Unidirectional Proxy Re-Encryption

Formally, a (single hop) unidirectional proxy re-encryption scheme consists of the following seven algorithms [23]:

GlobalSetup(κ): The global setup algorithm takes as input a security parameter κ, and outputs the global parameters $param$. For brevity, hereafter we assume that $param$ is implicitly included in the input of the following algorithms.

KeyGen(i): The key generation algorithm generates the public/secret key pair (pk_i, sk_i) for user i.

ReKeyGen(sk_i, pk_j): The re-encryption key generation algorithm takes as input the private key sk_i of the delegator and the public key pk_j of the delegatee. It outputs a re-encryption key $rk_{i \to j}$.

L1-Enc(pk, m): The first-level encryption algorithm takes as input a public key pk and a message $m \in \mathcal{M}$, and it outputs a first-level ciphertext that cannot be re-encrypted for another party. Here $\mathcal{M}$ denotes the message space. First-level encryption can be viewed as standard encryption, since it is not expected to be converted.

$\mathsf{L2\text{-}Enc}(pk, m)$: The second-level encryption algorithm takes as input a public key pk and a message m, and it outputs a second-level ciphertext that can be re-encrypted for another party using a suitable re-encryption key.

$\mathsf{ReEnc}(rk_{i\to j}, c_i)$: The re-encryption algorithm takes as input a re-encryption key $rk_{i\to j}$ and a second-level ciphertext c_i under public key pk_i. It outputs a first-level ciphertext c_j of the same message under public key pk_j.

$\mathsf{Dec}(sk, c)$: The decryption algorithm takes as input of a private key sk and a ciphertext c, and outputs a message $m \in \mathcal{M}$ or the error symbol $\perp$.

Correctness. Correctness requires that, for any $m \in \mathcal{M}$ and any couple of public/private key pair $(pk_i, sk_i), (pk_j, sk_j)$, the following conditions hold:

$$\mathsf{Dec}(sk_i, \mathsf{L1\text{-}Enc}(pk_i, m)) = m,$$
$$\mathsf{Dec}(sk_i, \mathsf{L2\text{-}Enc}(pk_i, m)) = m,$$
$$\mathsf{Dec}\left(sk_j, \mathsf{ReEnc}(\mathsf{ReKeyGen}(sk_i, sk_j), \mathsf{L2\text{-}Enc}(pk_i, m))\right) = m.$$

Security definition. The chosen-ciphertext security for unidirectional proxy re-encryption scheme Π is defined via the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:

Setup. $\mathcal{C}$ takes a security parameter κ and runs algorithm $\mathsf{GlobalSetup}$, which outputs to $\mathcal{A}$ the resulting global parameters $param$.

***Phase* 1.** $\mathcal{A}$ adaptively issues queries $q_1, \cdots, q_m$, with query q_i being one of the following:

- *Uncorrupted key generation query* $\langle i \rangle$: $\mathcal{C}$ first runs algorithm KeyGen to obtain a public/private key pair (pk_i, sk_i), and then sends pk_i to $\mathcal{A}$.
- *Corrupted key generation query* $\langle j \rangle$: $\mathcal{C}$ first runs algorithm KeyGen to obtain a public/private key pair (pk_j, sk_j), and then gives (pk_j, sk_j) to $\mathcal{A}$.
- *Re-encryption key generation query* $\langle pk_i, pk_j \rangle$: $\mathcal{C}$ first runs $\mathsf{ReKeyGen}(sk_i, pk_j)$ to generate a re-encryption key $rk_{i,j}$. Then $\mathcal{C}$ returns $rk_{i,j}$ to $\mathcal{A}$. Here sk_i and sk_j are private keys with respect to pk_i and pk_j, respectively. It is required that pk_i and pk_j were generated beforehand by algorithm KeyGen.
- *Re-encryption query* $\langle pk_i, pk_j, c_i \rangle$: $\mathcal{C}$ first runs algorithm $\mathsf{ReKeyGen}$ to generate the re-encryption key $rk_{i,j}$. Then it runs $\mathsf{ReEncrypt}(rk_{i,j}, c_i)$ to obtain the resulting ciphertext c_j, which is returned to $\mathcal{A}$. It is required that pk_i and pk_j were generated beforehand by KeyGen.
- *Decryption query* $\langle pk, c \rangle$: $\mathcal{C}$ returns the result of $\mathsf{Dec}(sk, c)$ to $\mathcal{A}$, where sk is the private key with respect to pk. It is required that pk was generated beforehand by KeyGen.

Challenge. Once $\mathcal{A}$ decides that Phase 1 is over, it outputs a target public key pk^* and two equal-length plaintexts $m_0, m_1 \in \mathcal{M}$. Here it is required that $\mathcal{A}$ did not previously corrupt the private key corresponding to pk^*. $\mathcal{C}$ flips a random coin $\delta \in \{0, 1\}$, and sets the challenge ciphertext to be $c^* = \mathsf{L2\text{-}Enc}(pk^*, m_\delta)$, which is sent to $\mathcal{A}$. Note that in the setting of proxy re-encryption, we certainly focus on the second level encryption.

***Phase* 2.** $\mathcal{A}$ issues additional queries $q_{m+1}, \cdots, q_{max}$, with each being one of the following:

- *Uncorrupted key generation query* $\langle i \rangle$: $\mathcal{C}$ responds as in Phase 1.

- *Corrupted key generation query* $\langle j \rangle$: $\mathcal{C}$ responds as in Phase 1, with the exception that $pk_j \neq pk^*$. Besides, if $\mathcal{A}$ has obtained a *derivative*[1] (pk', c') of (pk^*, c^*), it is required that $pk_j \neq pk'$.
- *Re-encryption key generation query* $\langle pk_i, pk_j \rangle$: $\mathcal{C}$ responds as in Phase 1, with the exception 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 $\langle pk^*, pk_j \rangle$.
- *Re-encryption query* $\langle pk_i, pk_j, c_i \rangle$: $\mathcal{C}$ responds as in Phase 1, except that if $\mathcal{A}$ has obtained the private key sk_j with respect to pk_j, then (pk_i, c_i) can not be a derivative of (pk^*, c^*).
- *Decryption query* $\langle pk, c \rangle$: $\mathcal{C}$ responds as in Phase 1, except that (pk, c) can not be a derivative of (pk^*, c^*).

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

We refer to adversary $\mathcal{A}$ as an IND-PRE-CCA adversary, and we define his advantage in attacking scheme Π as

$$\mathrm{Adv}_{\Pi,\mathcal{A}}^{\text{IND-PRE-CCA}} = \left|\Pr[\delta' = \delta] - \frac{1}{2}\right|,$$

where the probability is taken over the random coins consumed by the challenger and the adversary.

Definition 1 *A proxy re-encryption scheme* Π *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* $\mathrm{Adv}_{\Pi,\mathcal{A}}^{\text{IND-PRE-CCA}} \leq \epsilon$.

2.2 Complexity Assumptions

Throughout the paper, $x \xleftarrow{\$} S$ denotes that x is chosen randomly from a finite set S with a uniform distribution.

The security of our unidirectional proxy re-encryption scheme is based on the computational Diffie-Hellman (CDH) assumption.

Definition 2 *Let* $\mathbb{G}$ *be a cyclic multiplicative group with prime order* q*. The* CDH *problem in group* $\mathbb{G}$ *is, given a tuple* $(g, g^a, g^b) \in \mathbb{G}^3$ *with unknown* $a, b \xleftarrow{\$} \mathbb{Z}_q^*$*, to compute* g^{ab}.

Definition 3 *For a polynomial-time adversary* $\mathcal{B}$*, we define his advantage in solving the* CDH *problem in group* $\mathbb{G}$ *as*

$$\mathrm{Adv}_{\mathcal{B}}^{\text{CDH}} \triangleq \Pr\left[\mathcal{B}(g, g^a, g^b) = g^{ab}\right],$$

[1] Derivative of (pk^*, c^*) is inductively defined in [10] as below:

1. (pk^*, c^*) is a derivative of itself;
2. If (pk, c) is a derivative of (pk^*, c^*), and (pk', c') is a derivative of (pk, c), then (pk', c') is a derivative of (pk^*, c^*).
3. If $\mathcal{A}$ has issued a re-encryption query $\langle pk, pk', c \rangle$ and obtained the resulting re-encryption ciphertext c', then (pk', c') is a derivative of (pk, c).
4. If $\mathcal{A}$ has issued a re-encryption key generation query $\langle pk, pk' \rangle$, and $\mathsf{Dec}(sk', c') \in \{m_0, m_1\}$ (here sk' is the private key with respect to pk'), then (pk', c') is a derivative of (pk, c).

*where the probability is taken over the randomly choices of a, b and the random coins consumed by $\mathcal{B}$. We say that the (t, ϵ)-*CDH *assumption holds in group $\mathbb{G}$ if no t-time adversary $\mathcal{B}$ has advantage at least ϵ in solving the* CDH *problem in group $\mathbb{G}$.*

Bao *et al.* [5] introduced a variant of the CDH problem named divisible computation Diifie-Hellman (DCDH) problem. The DCDH problem in group $\mathbb{G}$ is, given $(g, g^{\frac{1}{a}}, g^b) \in \mathbb{G}^3$ with unknown $a, b \xleftarrow{\$} \mathbb{Z}_q^*$, to compute g^{ab}. In [5], Bao *et al.* showed the relation between CDH problem and DCDH problem in the following lemma:

Lemma 1 *The* DCDH *problem in group $\mathbb{G}$ is equivalent to the* CDH *problem in the same group.*

We also give a construction of bidirectional proxy re-encryption, whose security is based on the modified computational Diffie-Hellman (mCDH) assumption, a combination of the CDH problem and the DCDH problem. The mCDH assumption has been recently used to construct multi-use unidirectional proxy re-signatures [22].

Definition 4 *Let $\mathbb{G}$ be a cyclic multiplicative group with prime order q. The* mCDH *problem in group $\mathbb{G}$ is, given a tuple $(g, g^{\frac{1}{a}}, g^a, g^b) \in \mathbb{G}^4$ with unknown $a, b \xleftarrow{\$} \mathbb{Z}_q^*$, to compute g^{ab}.*

Definition 5 *For a polynomial-time adversary $\mathcal{B}$, we define his advantage in solving the* mCDH *problem in group $\mathbb{G}$ as*

$$\mathrm{Adv}_{\mathcal{B}}^{\mathrm{mCDH}} \triangleq \Pr\left[\mathcal{B}(g, g^{\frac{1}{a}}, g^a, g^b) = g^{ab}\right],$$

*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)-*mCDH *assumption holds in group $\mathbb{G}$ if no t-time adversary $\mathcal{B}$ has advantage at least ϵ in solving the* mCDH *problem in group $\mathbb{G}$.*

3 Unidirectional Proxy Re-Encryption Scheme

In this section, based on Weng et al.'s construction, we propose a CCA-secure unidirectional proxy re-encryption scheme without parings, denoted Π_{Uni}.

3.1 Overview

To facilitate understanding of our scheme, we first outline the basic idea underlying our construction.

CCA-secure ElGamal encryption Figure 2 shows the CCA-secure "hashed" ElGamal encryption scheme [8, 14, 15]. However, note that, in the ciphertext component $F = H_2(pk^r) \oplus (m \| \omega)$, the recipient's public key pk is embedded in the hash function H_2. This makes it impossible for the proxy to convert the ciphertext, and hence this original scheme cannot be directly used for proxy re-encryption. Fortunately, the modified scheme shown in Figure 3 (see the bolded parts) has the potential. The reason is that the ciphertext component F does not involve the recipient's public key, and the part $E = pk^r = g^{xr}$ involving the public key is not hashed; so it can be re-encrypted into another ciphertext component $E' = E^{\frac{\triangle}{x}} = g^{\triangle \cdot r}$($\triangle$ involves another public key $pk' = g^y$ in a certain manner), given that the re-encryption key is of the form $rk_{pk \to pk'} = \frac{\triangle}{x}$.

Setup(κ):	Encrypt(pk, m):	Decrypt($(E, F), sk$):
$x \xleftarrow{\$} \mathbb{Z}_q^*; pk = g^x; sk = x$ Return (pk, sk)	$\omega \xleftarrow{\$} \{0,1\}^{l_1}; r = H_1(m, \omega)$ $E = g^r; F = H_2(pk^r) \oplus (m \| \omega)$ Return c $= (E, F)$	$m \| \omega = F \oplus H_2(E^{sk})$ If $E = g^{H_1(m,\omega)}$ return m Else return $\perp$

Note: H_1 and H_2 are hash functions such that $H_1 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \to \mathbb{Z}_q^*, H_2 : \mathbb{G} \to \{0,1\}^{l_0+l_1}$. The massage space is $\mathcal{M} = \{0,1\}^{l_0}$.

Figure 2: CCA-secure "hashed" ElGamal encryption scheme

Setup(κ):	Encrypt(pk, m):	Decrypt($(E, F), sk$):
$x \xleftarrow{\$} \mathbb{Z}_q^*; pk = g^x; sk = x$ Return (pk, sk)	$\omega \xleftarrow{\$} \{0,1\}^{l_1}; r = H_1(m, \omega)$ $E = \boldsymbol{pk^r}; F = H_2(\boldsymbol{g^r}) \oplus (m \| \omega)$ Return c $= (E, F)$	$m \| \omega = F \oplus H_2(\boldsymbol{E^{\frac{1}{sk}}})$ If $\boldsymbol{E = pk^{H_1(m,\omega)}}$ return m Else return $\perp$

Figure 3: Modified CCA-secure "hashed" ElGamal encryption scheme

Indeed, the modified scheme can achieve the chosen-ciphertext security as a traditional public key encryption. However, it does not satisfy the chosen-ciphertext security for proxy re-encryptions. To see this, let's take the following attack as an example: suppose $\mathcal{A}$ is given a challenged ciphertext under a target public key $pk^* = g^x$, say $c^* = (E^*, F^*) = \big(g^{xr^*}, H_2(g^{r^*}) \oplus (m_\delta \| \omega^*)\big)$. Then adversary $\mathcal{A}$ can win the IND-PRE-CCA game as follows: He first picks $z \xleftarrow{\$} \{0,1\}^{l_0+l_1}$, and modifies the challenged ciphertext to get a new, although invalid, ciphertext $c' = (E', F') = (E^*, F^* \oplus z) = (g^{xr^*}, H_2(g^{r^*}) \oplus (m_\delta \| \omega^*) \oplus z)$. Next, he issues a corrupted key generation query to obtain a public/secret key pair $(\overline{pk}, \overline{sk}) = (g^y, y)$, and then issues a re-encryption query to obtain a re-encrypt ciphertext, say $c'' = (E'', F'') = (g^{\triangle \cdot r^*}, H_2(g^{r^*}) \oplus (m_\delta \| \omega^*) \oplus z)$, under the public key $\overline{pk} = g^y$. Finally, using the secret key $\overline{sk} = y$, $\mathcal{A}$ can recover $(m_\delta \| \omega^*)$ as $(m_\delta \| \omega^*) = F'' \oplus H_2((E'')^{\frac{1}{\triangle}}) \oplus z$, and eventually obtain the bit δ. Note that according to the constraints described in the IND-PRE-CCA game, it is legal for $\mathcal{A}$ to issue the above queries.

Combining with Schnorr signature The above attack succeeds due to the fact that, the validity of second-level ciphertexts can only be checked by the recipient, not any other parties including the proxy, i.e., the ciphertexts are not publicly verifiable. So, to achieve the IND-PRE-CCA security for a proxy re-encryption scheme, the proxy must be able to check the validity of the ciphertexts *without* seeing the plaintexts. It is not an easy task to achieve this without using linear pairings (e.g., all existing CCA-secure proxy re-encryption schemes are pairing based).

We get over this obstacle by resorting to the Schnorr signature scheme [26], which is given in Figure 4. In particular, given the ciphertext components $(E, F) = (pk^r, H_2(g^r) \oplus (m \| \omega))$, we generate the Schnorr signature as follows: viewing F as the message to be signed, and $(E, r) = (pk^r, r)$ as the verification/signing key pair (here the base pk in pk^r is similarly viewed as the base g in g^x), we pick $u \xleftarrow{\$} \mathbb{Z}_q^*$ and output the signature as $(D, s) = (pk^u, u + rH_3(D, E, F))$. The final ciphertext is (D, E, F, s).

Setup(κ):	Sign(sk, m):	Verify($pk, (D, s), m$):
$x \xleftarrow{\$} \mathbb{Z}_q^*; pk = g^x; sk = x$ Return (pk, sk)	$u \xleftarrow{\$} \mathbb{Z}_q^*; D = g^u$ $e = H(m, D); s = (u + sk \cdot e) \mod q$ Return $\sigma = (D, s)$	If $g^s = D \cdot pk^{H(m,D)}$ return 1 Else return 0

Figure 4: Schnorr signature scheme

Generation of Re-encryption key It remains to decide the format of re-encryption key. Recall that in the above, we assume that the re-encryption key takes the form of $rk_{pk\to pk'} = \frac{\triangle}{x}$. To ensure that only the delegatee can decrypt the converted ciphertext, $\triangle$ should be a quantity that can be generated by any one using public key $pk' = g^{x'}$, but once generated, can only be re-generated using x'. To this end, we choose $\triangle$ to be of the form pk'^v (please see the scheme below for the actual format of $\triangle$), where v is a random number; consequently, only the delegatee can compute $\triangle$, given g^v. However, it does not suffice for the re-encryption key to be in the current form, as it is not collusion resilient. It is easy to see that the proxy and the delegatee together can recover x. This is one the main issues we want to address. Our solution is to mask x using another value x', such that $rk_{pk\to pk'} = \frac{\triangle}{x+x'}$. As such, the coalition of the proxy and the delegatee can only get $x + x'$.

3.2 Proposed Scheme Π_{Uni}

Details of the proposed scheme Π_{Bi} are the following:

$\mathsf{GlobalSetup}(\kappa)$: Given a security parameter κ, choose two big 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. Besides, choose four hash functions H_1, H_2, H_3, H_4, and H_5 such that $H_1 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \to \mathbb{Z}_q^*, H_2 : \mathbb{G} \to \{0,1\}^{l_0+l_1}, H_3 : \{0,1\}^* \to \mathbb{Z}_q^*, H_4 : \mathbb{Z}_q^* \times \mathbb{G} \to \mathbb{Z}_q^*$, and $H_5 : \mathbb{G}^3 \to \mathbb{Z}_q^*$. l_0 and l_1 in the above are security parameters, and the message space is $\{0,1\}^{l_0}$. The global parameters are

$$param = (q, \mathbb{G}, g, H_1, H_2, H_3, H_4, H_5, l_0, l_1).$$

$\mathsf{KeyGen}(i)$: To generate the public/private key pair for user i, this key generation algorithm picks randomly $x_i, x_i' \xleftarrow{\$} \mathbb{Z}_q^*$, and then sets $pk_i = (pk_i^{\langle 1\rangle}, pk_i^{\langle 2\rangle}) = (g^{x_i}, g^{x_i'})$ and $sk_i = (x_i, x_i')$.

$\mathsf{ReKeyGen}(sk_i, pk_j)$: On input user i's secret key sk_i and user j's public key pk_j, this algorithm generates the re-encryption key $rk_{i\to j}$ from user i to j as below:

1. Pick $v' \xleftarrow{\$} \mathbb{Z}_q^*$. Compute $v = H_4(v', pk_j^{\langle 1\rangle})$.
2. Compute $V = g^v$, and $h_j = H_5(V, pk_j^{\langle 1\rangle}, (pk_j^{\langle 1\rangle})^v)$.
3. Define $rk_{i\to j}^{\langle 1\rangle} = \frac{h_j}{x_i + x_i'}$. Return $rk_{i\to j} = (rk_{i\to j}^{\langle 1\rangle}, V)$.

$\mathsf{L1\text{-}Enc}(pk, m)$:] On input a public key pk and a plaintext $m \in \{0,1\}^{l_0}$, first-level encryption works as below:

1. Pick $v' \xleftarrow{\$} \mathbb{Z}_q^*$. Compute $v = H_4(v', pk^{\langle 1\rangle})$.
2. Compute $V = g^v$, and $h = H_5(V, pk^{\langle 1\rangle}, (pk^{\langle 1\rangle})^v)$.
3. Pick $\omega \xleftarrow{\$} \{0,1\}^{l_1}$, and compute $r = H_1(m, \omega)$.
4. Compute $E = (g^h)^r, F = H_2(g^r) \oplus (m\|\omega)$.
5. Output the ciphertext $c = (E, F, V)$.

$\mathsf{L2\text{-}Enc}(pk, m)$: On input a public key pk and a plaintext $m \in \{0,1\}^{l_0}$, second-level encryption works as below:

1. Pick $u \xleftarrow{\$} \mathbb{Z}_q^*, \omega \xleftarrow{\$} \{0,1\}^{l_1}$, and compute $r = H_1(m, \omega)$.

2. Compute $D = (pk^{\langle 1\rangle})^u, E = (pk^{\langle 1\rangle}.pk^{\langle 2\rangle})^r, F = H_2(g^r) \oplus (m\|\omega), s = u + r \cdot H_3(D, E, F) \mod q$.
3. Output the ciphertext $c = (D, E, F, s)$.

$\mathsf{ReEnc}(rk_{i\to j}, c_i)$: On input a re-encryption key $rk_{i\to j}$, a second-level ciphertext c_i under public key pk_i, this algorithm re-encrypts the ciphertext as follows:

1. Parse c_i as $c_i = (D, E, F, s)$ and $rk_{i\to j}$ as $rk_{i\to j} = (rk_{i\to j}^{\langle 1\rangle}, V)$.
2. Check whether $(pk_i^{\langle 1\rangle})^s = D \cdot E^{H_3(D,E,F)}$ holds. If not, output $\perp$.
3. Otherwise, compute $E' = E^{rk_{i\to j}^{\langle 1\rangle}} = g^{(r\cdot(x_i+x_i'))\cdot\frac{h_j}{x_i+x_i'}} = g^{r\cdot h_j}$, and output ciphertext $c_j' = (E', F, V)$.

$\mathsf{Dec}(c, sk)$: On input a secret key $sk = (x, x')$ and ciphertext c, this algorithm works according to two cases:

- c is a first-level ciphertext $c = (E, F, V)$: Compute $h = H_5(V, pk^{\langle 1\rangle}, V^x)$ and $(m\|\omega) = F \oplus H_2(E^{\frac{1}{h}})$, and check whether $E = g^{H_1(m,\omega)\cdot h}$ holds. If yes, return m; otherwise, return $\perp$.
- c is a second-level ciphertext $c = (D, E, F, s)$: If $\left(g^{(x+x')}\right)^s = D \cdot E^{H_3(D,E,F)}$ does not hold, output $\perp$. Otherwise, compute $(m\|\omega) = F \oplus H_2(E^{\frac{1}{x+x'}})$, and return m if $E = \left(pk^{\langle 1\rangle}\right)^{H_1(m,\omega)}$ holds; else return $\perp$.

It can be verified that, given the re-encryption key $rk_{i\to j} = (\frac{H_5(V,pk_j,pk_j^v)}{x_i+x_i'}, V)$, the proxy is unable to generate the re-encryption key $rk_{j\to i}$ for the opposite direction, and hence it is impossible for him to convert a ciphertext intended for user j into a ciphertext for user i. Our scheme is thus unidirectional.

Remark 1 (Certification of public key). In our scheme, a user's key material includes two components (g^x, x) and $(g^{x'}, x')$, both are standard ElGamal key pairs. In practice, the certificate for the public key does not necessarily cover both components. More specifically, we can treat (g^x, x) as the *main key*, with usage beyond the setting of proxy re-encryption, e.g., they are a key pair in the regular sense, used for encryption and digital signature. Indeed, the first-level encryption defined as such in the above is solely to be in consistent with the converted ciphertext; otherwise, it can be simply the standard ElGamal encryption in Figure 2 or 3. In contrast, we see $(g^{x'}, x')$ as an *ancillary key*, specific to the proxy re-encryption setting. Consequently, it suffices to only certify (g^x, x) under the PKI, while $(g^{x'}, x')$ is certified by (g^x, x).

Remark 2 (Time limited delegation). Ateniese *et al.* [1,2] suggested time limited delegation, such that a delegation is temporary, only valid for a short period of time. Temporary delegation facilitates revocation of delegation. They proposed an efficient mechanism to renew delegation by assuming a trusted universal server that periodically broadcasts a new system parameter, using which all users re-establish re-encryption keys with their respective proxies. A drawback of this method is that all users's delegations are forced to expire after a period. A more satisfactory solution should be that only the delegation of the user who wants to revoke expires, while others' remain. In general, users renewing their keys can trivially achieve this objective, but it is clearly not satisfactory either. Our scheme can well implement the general approach by renewing the ancillary key, while keeping the main key intact; after renewal, the main key certifies the new ancillary key.

3.3 Performance Comparison

Next, we compare our scheme Π_{Uni} with existing CCA-secure unidirectional PRE schemes. Till now, there exist two such schemes: one is in public key scenarios [21] and the other is in identity-based settings [1, 2]. To conduct a fair comparison, we compare our scheme Π_{Uni} with Libert-Vergnaud's scheme [21] (denoted by LV Scheme), since both are in public key scenarios. The comparison results indicate that our scheme Π_{Uni} is much more efficient than LV Scheme. It is worth noting that the computational cost and the ciphertext length in our scheme *decrease* with re-encryption, while those in LV Scheme *increase* with re-encryption. The security of our scheme is related to the standard and well-studied CDH assumption, while LV Scheme is proved under a stronger and less-studied assumption, named 3-quotient decision bilinear Diffie-Hellman (3-QDBDH) assumption. A limitation of our scheme is that it is proved in the random oracle model, while LV Scheme is proved in the standard model.

Schemes			Libert-Vergnaud's Scheme	Our Π_{Bi}
Comput. Cost	Encrypt		$2t_e + 1t_{me} + 1t_{\mathsf{s}}$	$3t_e$
	Re-Encrypt		$2t_p + 4t_e + 1t_{\mathsf{v}}$	$3t_e$
	Decrypt	2nd-level CiphTxt	$3t_p + 2t_e + 1t_{\mathsf{v}}$	$4t_e$
		1st-level CiphTxt	$5t_p + 2t_e + 1t_{\mathsf{v}}$	$3t_e$
CiphTxt Length	2nd-level CiphTxt		$1\|pk_{\mathsf{s}}\|+2\|\mathbb{G}_e\|+1\|\mathbb{G}_T\|+1\|\sigma_{\mathsf{s}}\|$	$3\|\mathbb{G}\|+1\|\mathbb{Z}_q\|$
	1st-level CiphTxt		$1\|pk_{\mathsf{s}}\|+4\|\mathbb{G}_e\|+1\|\mathbb{G}_T\|+1\|\sigma_{\mathsf{s}}\|$	$3\|\mathbb{G}\|$
Without Random Oracles?			✓	×
Underlying Assumptions			3-QDBDH	CDH

Table 1: Efficiency Comparison between Scheme Π_{Uni} and Libert-Vergnaud's Scheme

3.4 Security Analysis

In this subsection, we prove the chosen-ciphertext security for scheme Π_{Uni} under the CDH assumption.

Theorem 1 *Our scheme* Π_{Uni} *is* IND-PRE-CCA *secure in the random oracle model, assuming the* CDH *assumption holds in group* $\mathbb{G}$ *and the Schnorr signature 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* $i \in \{1, \cdots, 5\}$*, and breaks the* $(t, q_u, q_c, q_{rk}, q_{re}, q_d, \epsilon)$-IND-PRE-CCA *of our scheme* Π_{Uni}*, then, for any* $0 < \nu < \epsilon$*, there exists*

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

$$\begin{aligned} t' &\leq t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_u + q_c + q_{rk} + q_{re} + q_d)\mathcal{O}(1) \\ &\quad + (q_u + q_c + 2q_{rk} + 5q_{re} + q_d + q_{H_1}q_{re} + (q_{H_4} + 2q_{H_1})q_d)t_e, \\ \epsilon' &\geq \frac{1}{q_{H_2}}\left(\frac{2(\epsilon - \nu)}{e(1 + q_{rk})} - \frac{q_{H_1} + (q_{H_1} + q_{H_2} + q_{H_4})q_d}{2^{l_0 + l_1}} - \frac{q_{re} + 3q_d}{q}\right), \end{aligned}$$

  where t_e *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, we here prove this theorem under the DCDH problem. Suppose there exists a t-time adversary $\mathcal{A}$ who can break the IND-PRE-CCA security of scheme Π_{Uni} with advantage $\epsilon - \nu$. Then we show how to construct an algorithm $\mathcal{B}$ which can solve the (t', ϵ')-DCDH problem in group $\mathbb{G}$.

Suppose $\mathcal{B}$ is given as input a DCDH challenge tuple $(g, g^{\frac{1}{a}}, g^b)$ with unknown $a, b \xleftarrow{\$} \mathbb{Z}_q^*$. Algorithm $\mathcal{B}$'s goal is to output g^{ab}. 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 $(q, \mathbb{G}, g, H_1, H_2, H_3, H_4, H_5, l_0, l_1)$ to $\mathcal{A}$. Here H_1, H_2, H_3, H_4 and H_5 are random oracles controlled by $\mathcal{B}$.

Hash Oracle Queries. At any time adversary $\mathcal{A}$ can issue the random oracle queries H_i with $i \in \{1 \cdots, 5\}$. Algorithm $\mathcal{B}$ maintains five hash lists H_i^{list} with $i \in \{1 \cdots, 5\}$, which are initially empty. $\mathcal{B}$ responds H_2 and H_3 queries in the same way as in Theorem 2, and responds the other hash queries as below:

- H_1 *queries*: On receipt of an H_1 queries on (m, ω), if this query has appeared on the H_1^{list} in a tuple (m, ω, r), return the predefined value r as the result of the query. 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_4 *queries*: On receipt of an H_4 query $(v', pk) \in \mathbb{Z}_q^* \times \mathbb{G}$, if this query has appeared on the H_4^{list} in a tuple (v', pk, v), return the predefined value v as the result of the query. Otherwise, choose $v \xleftarrow{\$} \mathbb{Z}_q^*$, add the tuple (v', pk, v) to the list H_4^{list} and respond with $H_4(v', pk) = v$.
- H_5 *queries*: On receipt of an H_5 query $(V, pk, S) \in \mathbb{G}^3$, if this query has appeared on the H_5^{list} in a tuple (V, pk, S, μ), return the predefined value μ as the result of the query. Otherwise, choose $\mu \xleftarrow{\$} \mathbb{Z}_q^*$, add the tuple (V, pk, S, μ) to the list H_5^{list} and respond with $H_5(V, pk, S) = \mu$.

Phase 1. In this phase, adversary $\mathcal{A}$ issues a series of queries as in the definition of 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* $\langle i \rangle$. Algorithm $\mathcal{B}$ first picks $x_i \xleftarrow{\$} \mathbb{Z}_q^*$ and flips a biased coin $c_i \in \{0, 1\}$ that yields 0 with probability θ and 1 with probability $1 - \theta$. If $c_i = 0$, it defines $pk_i = \left(g^{1/a}\right)^{x_i}$; else defines $pk_i = g^{x_i}$. Next, it adds the tuple (pk_i, x_i, c_i) to K^{list} and returns pk_i to adversary $\mathcal{A}$.
- *Corrupted key generation query* $\langle j \rangle$. Algorithm $\mathcal{B}$ first picks $x_j \xleftarrow{\$} \mathbb{Z}_q^*$ and defines $pk_j = g^{x_j}, c_j =$ '$-$'. Next, it adds the tuple (pk_j, x_j, c_j) to K^{list} and returns (pk_j, x_j) to adversary $\mathcal{A}$.
- *Re-encryption key generation query* $\langle pk_i, pk_j \rangle$: If R^{list} has contains a tuple for this entry (pk_i, pk_j), return the predefined re-encryption key to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ acts as follows:
  1. Recover tuples (pk_i, x_i, c_i) and (pk_j, x_j, c_j) from K^{list}.
  2. Pick $v' \xleftarrow{\$} \mathbb{Z}_q^*$. Compute $v = H_4(v', pk_j)$, $V = g^v$ and $h_j = H_5(V, pk_j, pk_j^v)$.
  3. Construct the first component $rk_{i,j}^{\langle 1 \rangle}$ according to the following cases:

- $c_i = 1$ or $c_i =$ '$-$': define $rk_{i,j}^{\langle 1\rangle} = \frac{h_j}{x_i}$, and add $(pk_i, pk_j, (rk_{i,j}^{\langle 1\rangle}, V), h_j, 1)$ into list R^{list}.
- $(c_i = 0 \wedge c_j = 1)$ or $(c_i = 0 \wedge c_j = 0)$: pick $rk_{i,j}^{\langle 1\rangle} \xleftarrow{\$} \mathbb{Z}_q^*$, and add $(pk_i, pk_j, (rk_{i,j}^{\langle 1\rangle}, V), h_j, 0)$ into list R^{list}.
- $(c_i = 0 \wedge c_j =$ '$-$'$)$: output "failure" and **abort**.

4. Finally, return $rk_{i,j} = (rk_{i,j}^{\langle 1\rangle}, V)$ to $\mathcal{A}$.

- *Re-encryption query* $\langle pk_i, pk_j, \text{CT}_i (= (D, E, F, s))\rangle$: If $pk_i^s \neq D \cdot E^{H_3(D,E,F)}$, then output $\perp$. Otherwise, algorithm $\mathcal{B}$ responds to this query as follows:

1. Recover tuples (pk_i, x_i, c_i) and (pk_j, x_j, c_j) from K^{list}.
2. If $(c_i = 0 \wedge c_j =$ '$-$'$)$ does not hold, issue a re-encryption key generation query $\langle pk_i, pk_j\rangle$ to obtain $rk_{i,j}$, and then return $\mathsf{ReEncrypt}(rk_{i,j}, \text{CT}_i, pk_j)$ to $\mathcal{A}$.
3. Else, search whether there exists a tuple $(m, \omega, pk_i, r) \in H_1^{\text{list}}$ such that $pk_i^r = E$. If there exists no such tuple, return $\perp$. Otherwise, first choose $v' \xleftarrow{\$} \mathbb{Z}_q^*$. Next, compute $v = H_4(v', pk_j), V = g^v$ and $h_j = H_5(V, pk_j, pk_j^v)$. Finally, define $E' = g^{h_j r}$, and return (E', F, V) to $\mathcal{A}$.

- *Decryption query* $\langle pk, \text{CT}\rangle$: $\mathcal{B}$ first recovers tuple (pk, x, c) from K^{list}. If $c = 1$ or $c =$ '$-$', algorithm $\mathcal{B}$ runs $\mathsf{Decrypt}(\text{CT}, x)$ and returns the result to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ works according to the following two cases:
  - CT is a second-level ciphertext $\text{CT} = (D, E, F, s)$: If $pk^s \neq D \cdot E^{H_3(D,E,F)}$, return $\perp$ to $\mathcal{A}$ indicating that CT is an invalid ciphertext. Otherwise, search lists H_1^{list} and H_2^{list} to see whether there exist $(m, \omega, pk, r) \in H_1^{\text{list}}$ and $(R, \beta) \in H_2^{\text{list}}$ such that
  $$pk^r = E, \beta \oplus (m\|\omega) = F \quad \text{and} \quad R = g^r.$$
  If yes, return m to $\mathcal{A}$. Otherwise, return $\perp$.
  - CT is a first-level ciphertext $\text{CT} = (E', F, V)$ re-encrypted from pk': Algorithm $\mathcal{B}$ first recovers tuples (pk, x, c) and (pk', x', c') from K^{list}, and then responds according to the following three cases:
    - If there exist a tuple $(pk', pk, (rk^{\langle 1\rangle}, V), h, 1)$ in R^{list}: Compute $(m\|\omega) = F \oplus H_2(E'^{\frac{1}{h}})$. If $E' = g^{H_1(m,\omega)\cdot h}$ holds, return m, else return $\perp$.
    - If there exist a tuple $(pk', pk, (rk^{\langle 1\rangle}, V), h, 0)$ in R^{list}: First, compute $E = E'^{\frac{1}{rk^{\langle 1\rangle}}}$. Next, 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
    $$pk'^r = E, \beta \oplus (m\|\omega) = F \quad \text{and} \quad R = g^r.$$
    If yes, return m to $\mathcal{A}$, else return $\perp$.
    - Otherwise: First search list H_4^{list} to see whether there exist a tuple $(v', pk, v) \in H_4^{\text{list}}$ such that $g^v = V$. If no such tuple exists, return $\perp$. Otherwise, compute $h = H_5(V, pk, pk^v)$, and then 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
    $$pk^h = E, \beta \oplus (m\|\omega) = F \quad \text{and} \quad R = g^r.$$
    If yes, return m to $\mathcal{A}$, else return $\perp$.

Challenge. When $\mathcal{A}$ decides that Phase 1 is over, it outputs a target public key pk^* and two equal-length messages $m_0, m_1 \in \{0,1\}^{l_0}$. Algorithm $\mathcal{B}$ responds as follows:

1. Recover tuple (pk^*, x^*, 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 "failure" and **abort**. Otherwise, it means that $c^* = 0$, and $\mathcal{B}$ proceeds to execute the rest steps.
2. Pick $e^*, s^* \xleftarrow{\$} \mathbb{Z}_q^*$, and compute $D^* = \left(g^b\right)^{-e^* x^*} \left(g^{\frac{1}{a}}\right)^{x^* s^*}$ and $E^* = \left(g^b\right)^{x^*}$.
3. Pick $F^* \xleftarrow{\$} \{0,1\}^{l_0+l_1}$ and define $H_3(D^*, E^*, F^*) = e^*$.
4. Pick $\delta \xleftarrow{\$} \{0,1\}, \omega^* \xleftarrow{\$} \{0,1\}^{l_1}$, 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 $\text{CT}^* = (D^*, E^*, F^*, s^*)$ as the challenged ciphertext to adversary $\mathcal{A}$.

Again, let $u^* \triangleq s^* - abe^*$ and $r^* \triangleq ab$, it can be easily verified that the challenged ciphertext CT^* has the same distribution as the real one.

Phase 2. Adversary $\mathcal{A}$ continues to issue the rest of 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: Similar to the analysis in Theorem 2, we can have that algorithm $\mathcal{B}$'s advantage against the DCDH challenge is at least

$$\epsilon' \leq \frac{1}{q_{H_2}} \left(\frac{2(\epsilon - \nu)}{e(1 + q_{rk})} - \frac{q_{H_1} + (q_{H_1} + q_{H_2} + q_{H_4}) q_d}{2^{l_0+l_1}} - \frac{q_{re} + 3q_d}{q} \right),$$

and its time complexity is bounded by

$$\begin{aligned} t' \leq\ & t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_{H_5} + q_u + q_c + q_{rk} + q_{re} + q_d)\mathcal{O}(1) \\ & + (q_u + q_c + 2q_{rk} + 5q_{re} + q_d + q_{H_1} q_{re} + (q_{H_4} + 2q_{H_1}) q_d) t_e. \end{aligned}$$

This completes the proof of Theorem 1. □

4 Bidirectional Proxy Re-Encryption

While unidirectional proxy re-encryption is often desirable, in some cases bidirectional delegation may also be useful. We thus next give a bidirectional proxy re-encryption scheme, based on the idea of the earlier unidirectional scheme.

Proposed Scheme Π_{Bi} :

$\mathsf{GlobalSetup}(\kappa)$: Let p, q, g, l_0, l_1 be the same as in Π_{Uni}. Define $H_1 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \times \mathbb{G} \to \mathbb{Z}_q^*$, $H_2 : \mathbb{G} \to \{0,1\}^{l_0+l_1}$, $H_3 : \{0,1\}^* \to \mathbb{Z}_q^*$ and $H_4 : \mathbb{G} \times \mathbb{G} \to \{0,1\}^{l_0+l_1}$. The global parameters are

$$param = (q, \mathbb{G}, g, H_1, H_2, H_3, H_4, l_0, l_1).$$

$\mathsf{KeyGen}(i)$: Generate the public/privatet key pair for user i as $pk_i = (g^{x_i}, g^{x_i'})$ and $sk_i = (x_i, x_i')$, where $x_i, x_i' \xleftarrow{\$} \mathbb{Z}_q^*$.

ReKeyGen(sk_i, sk_j): On input two secret keys $sk_i = x_i$ and $sk_j = x_j$, the bidirectional re-encryption key is defined to be $rk_{i\to j} = \frac{x_j + x'_j}{x_i + x'_i} \mod q$.

L1-Enc(pk, m): The first level encryption is the following:

1. Pick $\omega \xleftarrow{\$} \{0,1\}^{l_1}$, and compute $r = H_1(m, \omega, g^x)$.
2. Compute $E = (g^x)^r, F = H_2(g^r) \oplus (m \| \omega)$.
3. Output the ciphertext $c = (E, F)$.

L2-Enc(pk, m): The second level encryption works as follows:

1. Pick $u \xleftarrow{\$} \mathbb{Z}_q^*, \omega \xleftarrow{\$} \{0,1\}^{l_1}$, and compute $r = H_1(m, \omega, pk)$.
2. Compute $D = (g^{x+x'})^u, E = (g^{x+x'})^r, F = H_2(g^r) \oplus (m \| \omega), s = u + r \cdot H_3(D, E, F) \mod q$.
3. Output the ciphertext $c = (D, E, F, s)$.

ReEnc$(rk_{i\to j}, c_i, pk_j)$: On input a re-encryption key $rk_{i\to j} = \frac{x_j + x'_j}{x_i + x'_i}$, a second-level ciphertext c_i under public key pk_i, this algorithm re-encrypts the ciphertext under public key pk_j as follows:

1. Parse c_i as $c_i = (D, E, F, s)$.
2. Check whether $(g^{x_i}.g^{x'_i})^s = D \cdot E^{H_3(D,E,F)}$ holds. If not, output $\perp$.
3. Otherwise, compute $E' = E^{rk_{i,j}} = g^{(r\cdot(x_i+x'_i)\cdot x_j + x'_j/x_i + x'_i} = g^{r\cdot(x_j+x'_j)}, F' = F \oplus H_4(E', g^{rk_{i\to j}})$, and output $c'_j = (pk_i, E', F')$.

Dec(c, sk): On input a secret key $sk = (x, x')$ and ciphertext c, decryption works according to three cases:

- c is the first-level ciphertext $c = (E, F)$: $(m \| \omega) = F \oplus H_2(E^{1/x})$; if $E = (g^x)^{H_1(m,\omega,g^x)}$ return m, otherwise return $\perp$.
- c is a second-level ciphertext $c = (D, E, F, s)$: If $(g^x.g^{x'})^s = D \cdot E^{H_3(D,E,F)}$ does not hold, output $\perp$, else compute $m \| \omega = F \oplus H_2(E^{\frac{1}{x+x'}})$, and return m if $E = (g^x.g^{x'})^{H_1(m,\omega,pk)}$ holds and $\perp$ otherwise.
- c is a re-encrypted ciphertext $c = (pk_i = (g^{x_i}, g^{x'_i}), E', F')$: First compute $m \| \omega = F \oplus H_2(E'^{\frac{1}{x+x'}}) \oplus H_4(E', (g^{x_i}.g^{x'_i})^{1/x+x'})$. If $E' = (g^x)^{H_1(m,\omega,pk_i)}$ holds return m; otherwise return $\perp$.

Security For the security of the above construction, we have the following theorem under Weng et al.'s model [13], assuming random oracles. The security proof can be found in Appendix.

Theorem 2 *Our PRE scheme* Π_{Bi} *is* IND-PRE-CCA *secure in the random oracle model, assuming the* mCDH *assumption holds in group* $\mathbb{G}$ *and the Schnorr signature is existential unforgeable against chosen message attack* (EUF-CMA). *Concretely, if there exists an adversary* $\mathcal{A}$*, who asks at most* q_{H_i} *random oracle quires to* H_i *with* $i \in \{1, \cdots, 4\}$*, and breaks the* $(t, q_u, q_c, q_{rk}, q_{re}, q_d, \epsilon)$-IND-PRE-CCA *security of our scheme* Π_{Bi}*, then, for any* $0 < \nu < \epsilon$*, there exists*

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

$$\begin{aligned} t' &\leq t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_u + q_c + q_{rk} + q_{re} + q_d)\mathcal{O}(1) \\ &\quad + (q_u + q_c + 4q_{re} + 3q_d + (2q_d + q_{re})q_{H_1})t_e, \\ \epsilon' &\geq \frac{1}{q_{H_2}}\left(2(\epsilon - \nu) - \frac{q_{H_1} + (q_{H_1} + q_{H_2})q_d}{2^{l_0+l_1}} - \frac{q_{re} + 2q_d}{q}\right), \end{aligned}$$

  where t_e denotes the running time of an exponentiation in $\mathbb{G}$;

- *or an attacher who breaks the EUF-CMA security of the Schnorr signature with advantage ν within time t'.*

5 Extension

In this section, we extend our unidirectional scheme to address the problems of re-encryption key sharing and change of delegation path, which have not been well solved thus far in all existing proxy re-encryption schemes (See Section 1). We assume that the proxy has a key pair (pk_p, sk_p) for standard encryption. We stress again that this assumption is reasonable, considering that the proxy, like the delegator and the delegatee, is anyhow an entity within the PKI. We also like to point out that we do not see other proxy re-encryption schemes can straightforwardly solve the problems even they have the same assumption.

The extended scheme simply slightly modifies L2-Enc and ReEnc in the basic scheme Π_{Uni}, and all other steps and algorithms remain unchanged. In particular, recall that in our basic scheme, the second-level encryption L2-Enc generates $E = (pk^{\langle 1 \rangle}.pk^{\langle 2 \rangle})^r$. In the extension, we separate E into $E_1 = (pk^{\langle 1 \rangle})^r$ and $E_2' = (pk^{\langle 2 \rangle})^r$; then encrypts E_2' using the proxy's public key pk_p, which yields $E_2 = Enc_{pk_p}(E_2')$. As such, the resulting second-level ciphertext is $(D, E = (E_1, E_2), F, s)$. In the ReEnc algorithm, after verification of $(pk_i^{\langle 1 \rangle})^s = D \cdot E^{H_3(D,E,F)}$, the proxy then decrypts E_2 using its private key sk_p to get $E_2' = (pk^{\langle 2 \rangle})^r$, and computes $E_1.E_2'$, which is exactly $(pk^{\langle 1 \rangle}.pk^{\langle 2 \rangle})^r$.

We next see how the extended scheme solves the re-encryption key sharing problem and the change of delegation path problem. For the former, the proxy can still share the re-encryption key with the delegatee, but such a sharing does not help the delegatee, since E_2 is an encryption under the proxy's public key. Unless the proxy shares its private key with the delegatee, the delegatee still cannot convert the ciphertext. Here, the assumption is that the proxy does not afford disclosing its private key, which is a general assumption in PKI. For the second problem, to enable change of delegation path, the sender who sends messages to the delegator simply encrypts E_2' using the new proxy's public key.

Remark 3 (Proxy invisibility). It is evident that in the extended scheme, the sender needs to be aware of the delegator's proxy, and needs to check the validity of the proxy's public key. We can mitigate this problem by treating the proxy's public key in a way similar to handling the delegator's ancillary key (see Remark 1). In particular, the delegator certifies her proxy's public key using her main private key, and includes it into her public key as an application specific component (besides the ancillary key). In this way, the sender does not need to verify the proxy's public key as a regular key in the PKI, and only needs to trust the delegator's public key. In changing delegation path, the delegator removes the old proxy's public key, and certifies and includes the new one's.

Remark 4 (Direct encryption to delegatee?) One may argue that as the (second-level) encryption involves encrypting to another party (i.e., the proxy), besides the delegator,

why not simply encrypt to the delegator as well as the delegatee. We have two reasons in favor of our strategy. First, in practical applications it seems that proxies are more stable than the delegatees. So the relatively less-frequent-changes of proxy save the sender's effort in checking public key validity. Second, in the case of two or more delegatees under a proxy, our strategy clearly has both better computation and communication performances.

6 Conclusions

Recently, Weng *et al.* proposed the first unidirectional proxy re-encryption scheme without using costly bilinear pairings. A drawback of their construction, however, is that a coalition of the proxy and the delegatee can recover the delegator's private key. In this work, we improved over their scheme by offering collusion resilience. Furthermore, we extend our scheme to address the re-encryption key sharing problem and the change of delegation path problem, which exist in all previous proxy re-encryption schemes.

References

[1] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. In Proc. of NDSS 2005, pp. 29-43, 2005.

[2] G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage. ACM Transactions on Information and System Security (TISSEC), 9(1):1-30, February 2006.

[3] D. Boneh, and X. Boyen. Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. In advances in Cryptology-Eurocrypt'04, LNCS 3027, pp. 223-238, Springer-Verlag, 2004.

[4] M. Blaze, G. Bleumer, and M. Strauss. Divertible Protocols and Atomic Proxy Cryptography. In advances in Cryptology-Eurocrypt'98, LNCS 1403, pp. 127-144, Springer-Verlag, 1998.

[5] F. Bao, R. H. Deng, H. Zhu. Variations of Diffie-Hellman Problem. In Proc. of ICICS'03, LNCS 2836, pp. 301-312, Springer-Verlag, 2003.

[6] D. Boneh and M. Franklin. Identity based encryption from the Weil pairing. In Advanecs in Cryptology-Crypto'01, LNCS 2139, pp. 213-229. Springer-Verlag, 2001.

[7] D. Boneh, E.-J. Goh, and T. Matsuo. Proposal for P1363.3 Proxy Re-encryption. http://grouper.ieee.org/groups/1363/IBC/submissions/NTTDataProposal-for-P1363.3-2006-09-01.pdf.

[8] J. Baek, R. Safavi-Naini, and W. Susilo. Certificateless Public Key Encryption without Pairing. In Proc. of ISC'05. LNCS 3650, pp. 134-148, Springer-Verlag, 2005.

[9] R. Canetti, S. Goldwasser. An Efficient Threshold Public Key Cryptosystem Secure against Adaptive Chosen Ciphertext Attack. In advances in Cryptology-Eurocrypt'99, LNCS 1592, pp.90-106. Springer-Verlag, 1999.

[10] R. Caneti and S. Hohenberger. Chosen-Ciphertext Secure Proxy Re-Encryption. In Proceeding of ACM CCS 2007.

[11] C. Chu and W. Tzeng. Identity-Based Proxy Re-Encryption without Random Oracles. In Proc. of ISC'07, LNCS 4779, pp. 189-202, Springer-Verlag, 2007.

[12] Y. Dodis, and A.-A. Ivan. Proxy Cryptography Revisited. In Proc. of NDSS'03, 2003.

[13] R. H. Deng, J. Weng, S. Liu, K. Chen. Chosen-Ciphertext Secure Proxy Re-encryption without Pairings. In proc. of International Conference on Cryptology and Network Security, CANS'08, pp. 1-17, 2008.

[14] T. ElGamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In Advances in Cryptology-Crypto'84, LNCS 196, pp.10-18, Springer-Verlag, 1984.

[15] E. Fujisaki and T. Okamoto. Secure Integration of Asymmetric and Symmetric Encryption Schemes, In Advances in Cryptology-Crypto'99, LNCS 1666, pp. 537-554, Springer-Verlag, 1999.

[16] M. Green and G. Ateniese. Identity-Based Proxy Re-Encryption. In Proc. of ACNS'07, LNCS 4521, pp. 288-306, Springer-Verlag, 2007.

[17] P. Golle, M. Jakobsson, A. Juels, and P. F. Syverson. Universal Re-Encryption for Mixnets. In Proc. of CT-RSA'04, LNCS 2964, pp. 163-178, Springer-Verlag, 2004.

[18] M. Jakobsson. On Quorum Controlled Asummetric Proxy Re-Encryption. In Proc. of PKC'99, LNCS 1560, pp. 112-121, Springer-Verlag, 1999.

[19] E. Kiltz and D. Galindo. Direct Chosen-Ciphertext Secure Identity-Based Key Encapsulation without Random Oracles. Cryptology ePrint Archive, Report 2006/034, 2006. http://eprint.iacr.org/.

[20] Eike Kiltz. Chosen-Ciphertext Secure Identity-Based Encryption in the Standard Model with Short Ciphertexts. Cryptology ePrint Archive, Report 2006/122, 2006. http://eprint.iacr.org/.

[21] B. Libert and D. Vergnaud. Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption. In Proc. of PKC'08, LNCS 4929, pp. 360-379, Springer-Verlag, 2008.

[22] B. Libert and D. Vergnaud. Multi-Use Unidirectional Proxy Re-Signatures. In P. Syverson and S. Jha, editor(s), 15th ACM Conference on Computer and Communications Security (ACM CCS 2008), ACM Press, October 2008, To appear.

[23] B. Libert and D. Vergnaud. Tracing Malicious Proxies in Proxy Re-Encryption. In Proc. of Pairing'2008, LNCS 5209, pp. 332-353. Springer-Verlag, 2008.

[24] T. Matsuo. Proxy Re-Encryption Systems for Identity-Based Encryption. In Proc. of Paring'07, LNCS 4575, pp. 247-267, Springer-Verlag, 2007.

[25] Masahiro Mambo and Eiji Okamoto. Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts. IEICE Trans. Fund. Electronics Communications and Computer Science, E80-A/1:54-63, 1997.

[26] C. P. Schnorr. Efficient Identifications and Signatures for Smart Cards. In advances in Cryptology-Crypto'89, LNCS 435, pp. 239-251, Springer-Verlag, 1990.

Proof for Theorem 2

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 scheme Π_{Bi} with advantage $\epsilon - \nu$. Then we show how to construct an algorithm $\mathcal{B}$ which can solve the (t', ϵ')-mCDH problem in group $\mathbb{G}$.

Suppose $\mathcal{B}$ is given as input an mCDH challenge tuple $(g, g^{\frac{1}{a}}, g^a, g^b) \in \mathbb{G}^4$ with unknown $a, b \xleftarrow{\$} \mathbb{Z}_q^*$. Algorithm $\mathcal{B}$'s goal is to output g^{ab}. 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 $(q, \mathbb{G}, g, H_1, H_2, H_3, H_4, l_0, l_1)$ to $\mathcal{A}$. Here H_1, H_2, H_3 and H_4 are random oracles controlled by $\mathcal{B}$.

Hash Oracle Queries. At any time adversary $\mathcal{A}$ can issue the random oracle queries H_1, H_2, H_3 and H_4. Algorithm $\mathcal{B}$ maintains four hash lists $H_1^{\text{list}}, H_2^{\text{list}}$, H_3^{list} and H_4^{list} which are initially empty, and responds as below:

- H_1 *queries*: On receipt of an H_1 queries on (m, ω, pk), if this query has appeared on the H_1^{list} in a tuple (m, ω, pk, r), return the predefined value r as the result of the query. Otherwise, choose $r \xleftarrow{\$} \mathbb{Z}_q^*$, add the tuple (m, ω, pk, r) to the list H_1^{list} and respond with $H_1(m, \omega, pk) = r$.
- H_2 *queries*: On receipt of an H_2 query $R \in \mathbb{G}$, if this query has appeared on the H_2^{list} in a tuple (R, β), return the predefined value β as the result of the query. Otherwise, choose $\beta \xleftarrow{\$} \{0,1\}^{l_0+l_1}$, add the tuple (R, β) to the list H_2^{list} and respond with $H_2(R) = \beta$.
- H_3 *queries*: On receipt of an H_3 query (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 as the result of the query. 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 *queries*: On receipt of an H_4 query (E', U), if this query has appeared on the H_4^{list} in a tuple (E', U, λ), return the predefined value λ as the result of the query. Otherwise, choose $\lambda \xleftarrow{\$} \{0,1\}^{l_0+l_1}$, add the tuple (E', U, λ) to the list H_4^{list} and respond with $H_4(E', U) = \lambda$.

Phase 1. In this phase, adversary $\mathcal{A}$ issues a series of queries as in the definition of the IND-PRE-CCA 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* $\langle i \rangle$. Algorithm $\mathcal{B}$ first picks $x_i \xleftarrow{\$} \mathbb{Z}_q^*$ and defines $pk_i = \left(g^{1/a}\right)^{x_i}, c_i = 0$. Next, it adds the tuple (pk_i, x_i, c_i) to K^{list} and returns pk_i to adversary $\mathcal{A}$. Here the bit c_i is used to denote whether the secret key with respect to pk_i is corrupted, i.e., $c_i = 0$ indicates uncorrupted and $c_i = 1$ means corrupted.
- *Corrupted key generation query* $\langle j \rangle$. Algorithm $\mathcal{B}$ first picks $x_j \xleftarrow{\$} \mathbb{Z}_q^*$ and defines $pk_j = g^{x_j}, c_j = 1$. Next, it adds the tuple (pk_j, x_j, c_j) to K^{list} and returns (pk_j, x_j) to adversary $\mathcal{A}$.

- *Re-encryption key generation query* $\langle pk_i, pk_j \rangle$: Recall that according to the definition of IND-PRE-CCA game, it is required that pk_i and pk_j were generated beforehand, and either both of them are corrupted or alternately both are uncorrupted. Algorithm $\mathcal{B}$ first recovers tuples (pk_i, x_i, c_i) and (pk_j, x_j, c_j) from K^{list}, and then returns the re-encryption key x_j/x_i to $\mathcal{A}$.
- *Re-encryption query* $\langle pk_i, pk_j, \text{CT}_i (= (D, E, F, s)) \rangle$: If $pk_i^s \neq D \cdot E^{H_3(D,E,F)}$, then output $\perp$. Otherwise, algorithm $\mathcal{B}$ responds to this query as follows:
  1. Recover tuples (pk_i, x_i, c_i) and (pk_j, x_j, c_j) from K^{list}.
  2. If $c_i = c_j$, compute $E' = E^{x_j/x_i}, F' = F \oplus H_4(E', g^{x_i/x_j})$ and return (E', F') as the first-level ciphertext to $\mathcal{A}$.
  3. Else, search whether there exists a tuple $(m, \omega, pk_i, r) \in H_1^{\text{list}}$ such that $pk_i^r = E$. If there exists no such tuple, return $\perp$. Otherwise, first compute $E' = pk_j^r$. Next, if $c_i = 1 \wedge c_j = 0$, define $F' = F \oplus H_4(E', g^{\frac{x_i a}{x_j}})$; else if $c_i = 0 \wedge c_j = 1$, define $F' = F \oplus H_4(E', g^{\frac{x_i}{a x_j}})$. Finally, return (E', F') as the first-level ciphertext to $\mathcal{A}$.
- *Decryption query* $\langle pk, \text{CT} \rangle$: Algorithm $\mathcal{B}$ first recovers tuple (pk, x, c) from list K^{list}. If $c = 1$, algorithm $\mathcal{B}$ runs $\mathsf{Decrypt}(\text{CT}, x)$ and returns the result to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ works according to the following two cases:
  - CT is a second-level ciphertext $\text{CT} = (D, E, F, s)$: If $pk^s \neq D \cdot E^{H_3(D,E,F)}$, return $\perp$ to $\mathcal{A}$. Otherwise, search lists H_1^{list} and H_2^{list} to see whether there exist $(m, \omega, pk, r) \in H_1^{\text{list}}$ and $(R, \beta) \in H_2^{\text{list}}$ such that
    $$pk^r = E, \beta \oplus (m \| \omega) = F \quad \text{and} \quad R = g^r.$$
    If yes, return m to $\mathcal{A}$. Otherwise, return $\perp$.
  - CT is a first-level ciphertext $\text{CT} = (pk'', E', F')$: Algorithm $\mathcal{B}$ acts as follows:
    1. Recover tuples (pk, x, c) and (pk'', x'', c'') from K^{list}.
    2. Define U according to the following three cases:
       - If $c = c''$: Define $U = g^{\frac{x''}{x}}$;
       - If $c = 0 \wedge c'' = 1$: Define $U = g^{\frac{x'' a}{x}}$;
       - If $c = 1 \wedge c'' = 0$: Define $U = g^{\frac{x}{a x}}$.
    3. search lists H_1^{list} and H_2^{list} to see whether there exist $(m, \omega, pk, r) \in H_1^{\text{list}}$ and $(R, \beta) \in H_2^{\text{list}}$ such that
       $$pk^r = E', \beta \oplus (m \| \omega) \oplus H_4(E', U) = F' \quad \text{and} \quad R = g^r.$$
       If yes, return m to $\mathcal{A}$. Otherwise, return $\perp$.

Challenge. When $\mathcal{A}$ decides that Phase 1 is over, it outputs a target public key pk^* and two equal-length messages $m_0, m_1 \in \{0,1\}^{l_0}$. Algorithm $\mathcal{B}$ responds as follows:

1. Recover tuple (pk^*, x^*, c^*) from K^{list}. Recall that according to the constraints described in IND-PRE-CCA game, K^{list} should contain this tuple, and c^* is equal to 0 (indicating that $pk^* = g^{\frac{x^*}{a}}$).
2. Pick $e^*, s^* \xleftarrow{\$} \mathbb{Z}_q^*$, and compute $D^* = \left(g^b\right)^{-e^* x^*} \left(g^{\frac{1}{a}}\right)^{x^* s^*}$ and $E^* = \left(g^b\right)^{x^*}$.
3. Pick $F^* \xleftarrow{\$} \{0,1\}^{l_0 + l_1}$ and define $H_3(D^*, E^*, F^*) = e^*$.
4. Pick $\delta \xleftarrow{\$} \{0,1\}, \omega^* \xleftarrow{\$} \{0,1\}^{l_1}$, and implicitly define $H_2(g^{ab}) = (m_\delta \| \omega^*) \oplus F^*$ and $H_1(m_\delta, \omega^*, pk^*) = ab$ (Note that algorithm $\mathcal{B}$ knows neither ab nor g^{ab}).

5. Return $\mathrm{CT}^* = (D^*, E^*, F^*, s^*)$ as the challenged ciphertext to adversary $\mathcal{A}$.

Note that by the construction given above, by letting $u^* \triangleq s^* - abe^*$ and $r^* \triangleq ab$, we can see that the challenged ciphertext CT^* has the same distribution as the real one, since H_2 acts as a random oracle, and

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

Phase 2. Adversary $\mathcal{A}$ continues to issue the rest of 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 mCDH instance.

Analysis: Now let's analyze the simulation. The main idea of the analysis is borrowed from [8]. We first evaluate the simulations of the random oracles. From the constructions of H_3 and H_4, it is clear that the simulations of H_3 and H_4 are perfect. As long as adversary $\mathcal{A}$ does not query $(m_\delta, \omega^*, pk^*)$ to H_1 nor g^{ab} to H_2, where δ and ω^* are chosen by $\mathcal{B}$ in the Challenge phase, the simulations of H_1 and H_2 are perfect. By AskH_1^* we denote the event that (m_δ, ω^*) has been queried to H_1. Also, by AskH_2^* we denote the event that g^{ab} has been queried to H_2.

As argued before, the challenged ciphertext provided for $\mathcal{A}$ is identically distributed as the real one from the construction. From the description of the simulation, it can be seen that the responses to $\mathcal{A}$'s re-encryption key queries are also perfect.

Next, we analyze the simulation of the re-encryption oracle. The responses to adversary $\mathcal{A}$'s re-encryption queries are perfect, unless $\mathcal{A}$ can submit valid second-level ciphertexts without querying hash function H_1(denote this event by $\mathsf{ReEncErr}$). However, since H_1 acts as a random oracle and adversary $\mathcal{A}$ issues at most q_{re} re-encryption queries, we have

$$\Pr[\mathsf{ReEncErr}] \leq \frac{q_{re}}{q}.$$

Now, we evaluate the simulation of the decryption oracle. The simulation of the decryption oracle is perfect, with the exception that simulation errors may occur in rejecting some valid ciphertexts. Fortunately, these errors are not significant as shown below: Suppose that (pk, CT), where $\mathrm{CT} = (D, E, F, s)$ or $\mathrm{CT} = (E, F)$, has been issued as a *valid* ciphertext. Even CT is valid, there is a possibility that CT can be produced without querying g^r to H_2, where $r = H_1(m, \omega, pk)$. Let Valid be an event that CT is valid, and let AskH_2 and AskH_1 respectively be events that g^r has been queried to H_2 and (m, ω, pk) has been queried to H_1 with respect to $(E, F) = (pk^r, H_2(g^r) \oplus (m \| \omega))$, where

$r = H_1(m, \omega, pk)$. We then have

$$\begin{aligned}\Pr[\mathsf{Valid}|\neg\mathsf{AskH}_2] &= \Pr[\mathsf{Valid} \wedge \mathsf{AskH}_1|\neg\mathsf{AskH}_2] + \Pr[\mathsf{Valid} \wedge \neg\mathsf{AskH}_1|\neg\mathsf{AskH}_2] \\ &\leq \Pr[\mathsf{AskH}_1|\neg\mathsf{AskH}_2] + \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_1 \wedge \neg\mathsf{AskH}_2] \\ &\leq \frac{q_{H_1}}{2^{l_0+l_1}} + \frac{1}{q},\end{aligned}$$

and similarly $\Pr[\mathsf{Valid}|\neg\mathsf{AskH}_1] \leq \frac{q_{H_2}}{2^{l_0+l_1}} + \frac{1}{q}$. Thus we have

$$\Pr[\mathsf{Valid}|(\neg\mathsf{AskH}_1 \vee \neg\mathsf{AskH}_2)] \leq \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_1] + \Pr[\mathsf{Valid}|\neg\mathsf{AskH}_2] \leq \frac{q_{H_1}+q_{H_2}}{2^{l_0+l_1}} + \frac{2}{q}.$$

Let DecErr be the event that $\mathsf{Valid}|(\neg\mathsf{AskH}_1 \vee \neg\mathsf{AskH}_2)$ happens during the entire simulation. Then, since q_d decryption oracles are issued, we have

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

Now let Good denote the event $\mathsf{AskH}_2^* \vee (\mathsf{AskH}_1^*|\neg\mathsf{AskH}_2^*) \vee \mathsf{ReEncErr} \vee \mathsf{DecErr}$. If event Good does not happen, it is clear that adversary $\mathcal{A}$ can not gain any advantage in guessing δ due to the randomness of the output of the random oracle H_2. Namely, we have $\Pr[\delta = \delta'|\neg\mathsf{Good}] = \frac{1}{2}$. Hence, by splitting $\Pr[\delta' = \delta]$, we have

$$\begin{aligned}\Pr[\delta' = \delta] &= \Pr[\delta' = \delta|\neg\mathsf{Good}]\Pr[\neg\mathsf{Good}] + \Pr[\delta' = \delta|\mathsf{Good}]\Pr[\mathsf{Good}] \\ &\leq \frac{1}{2}\Pr[\neg\mathsf{Good}] + \Pr[\mathsf{Good}] \\ &= \frac{1}{2}(1 - \Pr[\mathsf{Good}]) + \Pr[\mathsf{Good}] \\ &= \frac{1}{2} + \frac{1}{2}\Pr[\mathsf{Good}]\end{aligned}$$

and

$$\Pr[\delta' = \delta] \geq \Pr[\delta' = \delta|\neg\mathsf{Good}]\Pr[\neg\mathsf{Good}] = \frac{1}{2}(1 - \Pr[\mathsf{Good}]) = \frac{1}{2} - \frac{1}{2}\Pr[\mathsf{Good}].$$

Then we have

$$\left|\Pr[\delta' = \delta] - \frac{1}{2}\right| \leq \frac{1}{2}\Pr[\mathsf{Good}].$$

By definition of the advantage $(\epsilon - \nu)$ for the IND-PRE-CCA adversary, we then have

$$\begin{aligned}\epsilon - \nu &= \left|\Pr[\delta' = \delta] - \frac{1}{2}\right| \\ &\leq \frac{1}{2}\Pr[\mathsf{Good}] = \frac{1}{2}\left(\Pr[\mathsf{AskH}_2^* \vee (\mathsf{AskH}_1^*|\neg\mathsf{AskH}_2^*) \vee \mathsf{ReEncErr} \vee \mathsf{DecErr}]\right) \\ &\leq \frac{1}{2}\left(\Pr[\mathsf{AskH}_2^*] + \Pr[\mathsf{AskH}_1^*|\neg\mathsf{AskH}_2^*] + \Pr[\mathsf{ReEncErr}] + \Pr[\mathsf{DecErr}]\right).\end{aligned}$$

Since $\Pr[\mathsf{ReEncErr}] \leq \frac{q_{re}}{q}$, $\Pr[\mathsf{DecErr}] \leq \frac{(q_{H_1}+q_{H_2})q_d}{2^{l_0+l_1}} + \frac{2q_d}{q}$ and $\Pr[\mathsf{AskH}_1^*|\neg\mathsf{AskH}_2^*] \leq \frac{q_{H_1}}{2^{l_0+l_1}}$, we obtain

$$\begin{aligned}\Pr[\mathsf{AskH}_2^*] &\geq 2(\epsilon - \nu) - \Pr[\mathsf{AskH}_1^*|\neg\mathsf{AskH}_2^*] - \Pr[\mathsf{DecErr}] - \Pr[\mathsf{ReEncErr}] \\ &\geq 2(\epsilon - \nu) - \frac{q_{H_1}}{2^{l_0+l_1}} - \frac{(q_{H_1}+q_{H_2})q_d}{2^{l_0+l_1}} - \frac{2q_d}{q} - \frac{q_{re}}{q} \\ &= 2(\epsilon - \nu) - \frac{q_{H_1} + (q_{H_1}+q_{H_2})q_d}{2^{l_0+l_1}} - \frac{q_{re}+2q_d}{q}.\end{aligned}$$

Meanwhile, if event AskH_2^* happens, algorithm $\mathcal{B}$ will be able to solve the mCDH instance, and consequently, we obtain

$$\epsilon' \geq \frac{1}{q_{H_2}} \left(2(\epsilon - \nu) - \frac{q_{H_1} + (q_{H_1} + q_{H_2})q_d}{2^{l_0+l_1}} - \frac{q_{re} + 2q_d}{q} \right).$$

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

$$\begin{aligned} t' \quad \leq \quad & t + (q_{H_1} + q_{H_2} + q_{H_3} + q_{H_4} + q_u + q_c + q_{rk} + q_{re} + q_d)\mathcal{O}(1) \\ & + (q_u + q_c + 4q_{re} + 3q_d + (2q_d + q_{re})q_{H_1})t_e. \end{aligned}$$

This completes the proof of Theorem 2.