Improving the Security of an Efficient Unidirectional Proxy Re-Encryption Scheme

Sébastien Canard*
Orange Labs - Applied Crypto Group
Caen, France
sebastien.canard@orange-ftgroup.com
Julien Devigne*
Orange Labs - Applied Crypto Group/GREYC - Université de Caen Basse-Normandie
Caen, France
julien.devigne@orange-ftgroup.com
Fabien Laguillaumie*
GREYC - Université de Caen Basse-Normandie
Caen, France
fabien.laguillaumie@unicaen.fr

Abstract

A proxy re-encryption (PRE) scheme allows a designated *proxy*, that has beforehand received a so-called re-encryption key, to translate a ciphertext intended to one user to a ciphertext intended to another one. Traditionally, the re-encryption key is generated at the initiative of the initial receiver and ideally, no secret keys should be known to the proxy. Such scheme is said unidirectional if the transformation from one user to another does not necessarily imply the possibility to make the inverse transformation. Regarding the literature on unidirectional proxy re-encryption, it seems hard to prove the strongest security level (namely indistinguishability under chosen ciphertext attacks - IND-CCA) of such schemes. Most of the time, PRE either reaches a chosen-plaintext security or a *replayable* CCA security. At Africacrypt 2010, Chow, Weng, Yang and Deng proposed a scheme that satisfies CCA security in the random oracle model. However, their model can actually be strengthen. Indeed, we show in this paper how to modify this scheme so that its improved security achieves a full CCA security. In particular, we now allow the adversary of the CCA security for re-encryption to corrupt the user i' who is the initial receiver of the challenged ciphertext and at the same time to obtain the re-encryption key from i' to the targeted users. The resulting scheme is therefore a fully secure PRE which does *not* rely on pairings, and secure in the random oracle model. It can be implemented efficiently with any traditional modular arithmetic.
Keywords: Proxy re-encryption, unidirectional, CCA security.

1 Introduction

Public-key encryption, as the core of all security systems, has been deeply studied and the most secure schemes can be very efficiently implemented. This is however not always the case of encryption schemes "with special features" that can be found in the literature. In this paper, we focus on the case of *proxy re-encryption schemes* (PRE) that have been introduced in [4]. Such a scheme allows a semi-trusted entity, called the proxy, to convert a "level 2" ciphertext originally intended to a receiver, say Alice, into a "level 1" ciphertext intended to another receiver, called Bob, by using some re-encryption

*This work has been supported by the French Agence Nationale de la Recherche under the PACE 07 TCOM Project, and by the European Commission through the ICT Program under Contract ICT-2007-216676 ECRYPT II.

key. Obviously, this should be possible only if the initial receiver agrees and if the proxy does not learn either the plaintext, or the secret key of both Alice and Bob.

There currently exists different flavors of proxy re-encryption schemes. Some of them are *unidirectional* [2, 14, 17, 1, 7] and allow the proxy to only translate a ciphertext from Alice to Bob. The other case is called *bidirectional* [4, 5, 9, 15] and permits the proxy, with only one re-encryption key, to translate from Alice to Bob but also from Bob to Alice. Some schemes are *multi-hop* if a ciphertext can be forwarded several times between users, and *single-hop* if a ciphertext can be transformed just once. We will now focus on single-hop unidirectional re-encryption schemes.

Concerning the security aspects of a unidirectional PRE scheme, the strongest notion, introduced in [5] (for bidirectional schemes but easily adaptable for the unidirectional case, as explained in [14]) is related to the traditional *indistinguishability under chosen ciphertext attacks* (IND-CCA). In this model, ciphertexts should remain indistinguishable even if the adversary has access to a re-encryption oracle (translating ciphertexts chosen by the adversary), to a re-encryption key generation oracle and to traditional decryption oracles. If we consider the security of existing unidirectional schemes [2, 14, 17, 1, 7], the Ateniese *et al.*'s schemes [2, 1] only reach a CPA (chosen plaintext attack) security, the Libert-Vergnaud scheme [14] is only *replayable* CCA secure [6, 5] and the Shao *et al.* [17] has been shown in [7] to only reach a CPA security.

As we see, it seems very difficult to prove the CCA security of proxy re-encryption schemes, since current constructions are either broken or only achieve a more restrictive security definition. To the best of our knowledge, it only remains the paper from Chow *et al.* [7] to be CCA secure (in the random oracle model), under the CDH assumption. But as we will show, the model presented in [7] can be strengthen.

Indeed, the security of the Chow *et al.* scheme [7] can further be improved. First, for one (level 2) ciphertext (meaning that this ciphertext can be transformed into a ciphertext for another recipient) and one re-encryption key, the result of the re-encryption procedure given in [7] is deterministic. Making this algorithm probabilistic permits to improve the CCA security at level 1 (called *transformed ciphertext security* in [7]). More precisely, in the related security experiment, the adversary cannot corrupt the user i' who is the original recipient of the challenged ciphertexts and cannot obtain in the same time the re-encryption key from i' to the targeted users. Second, in this same level 1 CCA security proof, the authors claim that (using Coron's trick [8] on user i' or $i^\star$) it is possible to give the adversary access to some keys whereas this is not true. Finally, for the level 2 CCA security (called *original ciphertext security* in [7]), Chow *et al.* sometimes abort the experiment while this is possible to go on through the proof.

In this paper, we first modify Chow *et al.*'s scheme proposed in [7] so that the re-encryption procedure becomes probabilistic. This permits to reach a stronger notion of CCA security at level 2, as noticed in [7], since the restrictions given in Definition 4 (on transformed ciphertext security) are not relevant any more. Secondly, we make some corrections to the security proof proposed in [7]. Eventually, instead of aborting when some special cases occur, we succeed to go on the proof with some extraction techniques in non-interactive zero-knowledge proofs. In particular, we use Fischlin's construction of a non-interactive proof of knowledge [12]. Our resulting security proof is consequently improved compared to the original, while keeping the sole CDH assumption, in the random oracle model.

The paper is now organized as follows. In Section 2, we define the security model for unidirectional PRE schemes and next give some useful tools to design the modified scheme. Section 3 is dedicated to our corrections on the Chow *et al.* scheme and Section 4 gives our new security proof. We finally conclude in Section **??**.

2 Unidirectional Proxy Re-Encryption

2.1 Syntactic Definition

Definition 1 (PRE)**.** *Let κ be an integer. A* single-hop unidirectional proxy re-encryption scheme *consists of the eight algorithms defined as follows.*

- $\mathsf{Setup}(\kappa) \rightarrow \mathscr{P}$*: this setup algorithm takes a security parameter κ as input and produces a set of public parameters $\mathscr{P}$ shared by all parties.*
- $\mathsf{KeyGen}(\mathscr{P}) \rightarrow (sk, pk)$*: this key generation algorithm, whose inputs are the public parameters, outputs a pair of secret and public keys (sk, pk), and is executed by users.*
- $\mathsf{ReKeygen}(\mathscr{P}, sk_i, pk_i, pk_j) \rightarrow R_{i\rightarrow j}$*: given the public parameters, the secret and public keys of the user i, the public key of the user j, this algorithm produces a re-encryption key $R_{i\rightarrow j}$ which allows to transform second level ciphertexts intended to i into first level ciphertexts for j.*
- $\mathsf{Encrypt}_1(\mathscr{P}, pk, m) \rightarrow C$*: this first level encryption algorithm takes as inputs $\mathscr{P}$, a public key and a message. It outputs a first level ciphertext C that cannot be re-encrypted.*
- $\mathsf{Encrypt}_2(\mathscr{P}, pk, m) \rightarrow C$*: this second level encryption algorithm takes $\mathscr{P}$, a public key and a message as inputs, and produces a second level ciphertext C that can be re-encrypted.*
- $\mathsf{ReEncrypt}(\mathscr{P}, R_{i\rightarrow j}, C) \rightarrow C' / \perp$*: this algorithm takes as input the public parameters, a re-encryption key $R_{i\rightarrow j}$ and a second level ciphertext intended to user i. The output is a first level ciphertext*[1] *C' re-encrypted for user j or an invalid message $\perp$.*
- $\mathsf{Decrypt}_1(\mathscr{P}, sk, C) \rightarrow m / \perp$*: this first level decryption algorithm takes as input $\mathscr{P}$, a secret key and a first level ciphertext and outputs a plaintext m or an invalid message $\perp$.*
- $\mathsf{Decrypt}_2(\mathscr{P}, sk, C) \rightarrow m / \perp$*: this second level decryption algorithm takes as input the public parameters, a secret key and a second level ciphertext and outputs a plaintext m or $\perp$.*

For correctness *conditions, these algorithms must satisfy the following properties: for all public parameter $\mathscr{P}$ generated by the* Setup *algorithm, for any message m, and any couple of valid secret/public key pair (sk_i, pk_i), (sk_j, pk_j)*

$$\mathsf{Decrypt}_1(\mathscr{P}, sk_i, \mathsf{Encrypt}_1(\mathscr{P}, pk_i, m)) \rightarrow m, \;\; \mathsf{Decrypt}_2(\mathscr{P}, sk_i, \mathsf{Encrypt}_2(\mathscr{P}, pk_i, m)) \rightarrow m$$
$$\mathsf{Decrypt}_1(\mathscr{P}, sk_j, \mathsf{ReEncrypt}(\mathscr{P}, \mathsf{ReKeygen}(\mathscr{P}, sk_i, pk_i, pk_j), \mathsf{Encrypt}_2(\mathscr{P}, pk_i, m))) \rightarrow m.$$

The wellformedness of a ciphertext cannot be publicly checked during a re-encryption, so that one can re-encrypt incorrect ciphertexts, which will be revealed as invalid by the decryption algorithms.

2.2 Security

The security of our scheme is conducted in an adaptive corruption model, where the challenger generates public keys for all users and allows the adversary to get secret keys of some of these users (the *corrupted* ones). Our model implicitly makes the *knowledge of secret key (KOSK)* assumption , meaning that all users know the secret key corresponding to their published public key. In other words, the

[1] In a single-hop scheme, C' cannot be re-encrypted.

KOSK means that when a user wants its public key to be certified by a certification authority, he has to provide a proof of knowledge of his secret key. See [14] for a discussion of the stronger scenario of chosen-key model.

Different (essentially equivalent) variants of the CCA indistinguishability model exist for PRE. The one we present is inspired by the bidirectional case [5] (extending ideas from [6]). Our model differs a bit in the presentation from those in [14, 2], and is precisely described in the following. In particular, each algorithm that produces ciphertexts has to satisfy indistinguishability. This leads to the three experiments depicted in Fig. 1 corresponding to the $\mathsf{PRE.Encrypt}_1$, $\mathsf{PRE.Encrypt}_2$ and PRE.ReEncrypt algorithms.

$\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_1,\mathscr{A}}(\kappa,n)$
$\mathscr{P} \leftarrow \mathsf{Setup}(\kappa)$
for $i = 1$ to n do $(sk_i, pk_i) \leftarrow \mathsf{KeyGen}(\mathscr{P})$
$\mathscr{PK} \leftarrow \{pk_i\}_{i=1..n}$
$\mathscr{O} = \{\mathsf{ODec}_1, \mathsf{OReKG}, \mathsf{OSecKey}\}$
$(m_0, m_1, pk_{i^\star}, st) \leftarrow \mathscr{A}_f^{\mathscr{O}}(\mathscr{P}, \mathscr{PK})$
$\delta \xleftarrow{\$} \{0,1\}$
$C^\star \leftarrow \mathsf{Encrypt}_1(\mathscr{P}, pk_{i^\star}, m_\delta)$
$\delta' \leftarrow \mathscr{A}_g^{\mathscr{O}}(st, C^\star)$
Return $(\delta' = \delta)$

$\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_2,\mathscr{A}}(\kappa,n)$
$\mathscr{P} \leftarrow \mathsf{Setup}(\kappa)$
for $i = 1$ to n do $(sk_i, pk_i) \leftarrow \mathsf{KeyGen}(\mathscr{P})$
$\mathscr{PK} \leftarrow \{pk_i\}_{i=1..n}$
$\mathscr{O} = \{\mathsf{ODec}_1, \mathsf{ODec}_2, \mathsf{OReKG}, \mathsf{OReEnc}, \mathsf{OSecKey}\}$
$(m_0, m_1, pk_{i^\star}, st) \leftarrow \mathscr{A}_f^{\mathscr{O}}(\mathscr{P}, \mathscr{PK})$
$\delta \xleftarrow{\$} \{0,1\}$
$C^\star \leftarrow \mathsf{Encrypt}_2(\mathscr{P}, pk_{i^\star}, m_\delta)$
$\delta' \leftarrow \mathscr{A}_g^{\mathscr{O}}(st, C^\star)$
Return $(\delta' = \delta)$

$\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.ReEncrypt},\mathscr{A}}(\kappa,n)$
$\mathscr{P} \leftarrow \mathsf{Setup}(\kappa)$
for $i = 1$ to n do $(sk_i, pk_i) \leftarrow \mathsf{KeyGen}(\mathscr{P})$
$\mathscr{PK} \leftarrow \{pk_i\}_{i=1..n}$
$\mathscr{O} = \{\mathsf{ODec}_1, \mathsf{OReKG}, \mathsf{OSecKey}\}$
$(C_0, C_1, pk_{i^\star}, pk_{i'}, st) \leftarrow \mathscr{A}_f^{\mathscr{O}}(\mathscr{P}, \mathscr{PK})$ with C_0, C_1 two "good messages" which can be re-encrypted from $pk_{i'}$ to $pk_{i^\star}$.
$\delta \xleftarrow{\$} \{0,1\}$
$R_{i' \to i^\star} \leftarrow \mathsf{ReKeyGen}(\mathscr{P}, sk_{i'}, pk_{i'}, pk_{i^\star})$
$C^\star \leftarrow \mathsf{ReEncrypt}(\mathscr{P}, R_{i' \to i^\star}, c_\delta)$
$\delta' \leftarrow \mathscr{A}_g^{\mathscr{O}}(st, C^\star)$
Return $(\delta' = \delta)$

Figure 1: Security experiments for indistinguishability

IND-CCA SECURITY OF $\mathsf{PRE.Encrypt}_1$. We define the CCA indistinguishability of the $\mathsf{Encrypt}_1$ algorithm of a single-hop unidirectional PRE scheme by describing a two-stage adversary $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$ having access to the following oracles. The key $pk_{i^\star}$ must be a key of an uncorrupted user.

- ODec_1: when queried on (pk, C), a first level ciphertext C intended to $pk \in \mathscr{PK}$, this oracle answers with $\mathsf{Decrypt}_1(\mathscr{P}, sk, C)$. The second stage adversary $\mathscr{A}_g$ is not allowed to query $(pk_{i^\star}, C^\star)$ to ODec_1.
- OReKG: when queried on (pk_i, pk_j) for a re-encryption key from user i to user j, OReKG answers with $R_{i \to j} \leftarrow \mathsf{ReKeygen}(\mathscr{P}, sk_i, pk_i, pk_j)$. There is no restriction on this oracle, therefore,

there is no need for a decryption oracle for level 2 ciphertexts, nor for a re-encryption oracle.

- OSecKey: when queried on pk_i, OSecKey answers with sk_i the secret key associated to pk_i.

Definition 2 (IND-CCA security of $\mathsf{PRE.Encrypt}_1$)**.** *Let κ and n be integers. Let $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$ be an adversary against the* CCA *indistinguishability of* $\mathsf{PRE.Encrypt}_1$*. Let* $\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_1,\mathscr{A}}(\kappa,n) := 2 \cdot \Pr\left[\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_1,\mathscr{A}}(\kappa,n) \rightarrow \mathit{true}\right] - 1$*, with* $\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_1,\mathscr{A}}$ *as defined in Fig. 1 We say that* PRE *has* IND-CCA *security of* $\mathsf{PRE.Encrypt}_1$ *if for every p.p.t. adversary $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$, the advantage* $\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_1 i,\mathscr{A}}(\kappa,n)$ *is negligible.*

IND-CCA SECURITY OF $\mathsf{PRE.Encrypt}_2$**.** We define the indistinguishability under a chosen ciphertext attack of the $\mathsf{Encrypt}_2$ algorithm of a single-hop unidirectional proxy re-encryption scheme by describing a two-stage attacker $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$ having access to the following oracles. $\mathscr{A}$ is not allowed to attack a key $pk_{i^\star}$ if it is those of a corrupted user, or if he has queried for a re-encryption key from $pk_{i^\star}$ to a corrupted user.

- ODec_1: when queried on (pk, C), a first level ciphertext C intended to $pk \in \mathscr{P}\mathscr{K}$, this oracle answers with $\mathsf{Decrypt}_1(\mathscr{P}, sk, C)$. If the second stage adversary $\mathscr{A}_g$ asks for a *derivative* of $(pk_{i^\star}, C^\star)$, this oracle responds with $\perp$. Such a derivative is any pair (pk, C) such that $(pk, C) = (pk_{i^\star}, C^\star)$ or $C = \mathsf{ReEncrypt}(\mathscr{P}, \mathsf{ReKeygen}(\mathscr{P}, sk_{i^\star}, pk_{i^\star}, pk), C^\star)$.
- ODec_2: when queried on (pk, C), a second level ciphertext C intended to $pk \in \mathscr{P}\mathscr{K}$, this oracle answers with $\mathsf{Decrypt}_2(\mathscr{P}, sk, C)$. The second stage adversary $\mathscr{A}_g$ is not allowed to query $(pk_{i^\star}, C^\star)$ to ODec_2.
- OReEnc: on input (pk_i, pk_j, C) where C is a second level ciphertext, this oracle answers $\perp$ if $(pk_i, C) = (pk_{i^\star}, C^\star)$ and pk_j is those of a corrupted user, otherwise it computes the re-encryption key $R_{i\rightarrow j} = \mathsf{ReKeygen}(\mathscr{P}, sk_i, pk_i, pk_j)$ and outputs $C' \leftarrow \mathsf{ReEncrypt}(\mathscr{P}, R_{i\rightarrow j}, C)$.
- OReKG: when queried on (pk_i, pk_j) for a re-encryption key from user i to user j, OReKG answers with $R_{i\rightarrow j} \leftarrow \mathsf{ReKeygen}(\mathscr{P}, sk_i, pk_i, pk_j)$. The attacker is not allowed to query $(pk_{i^\star}, pk)$ with pk associated to a corrupted user.
- OSecKey: when queried on pk_i, OSecKey answers with sk_i the secret key associated to pk_i. The user associated to pk_i becomes corrupted. If the second stage adversary $\mathscr{A}_g$ asks for a secret key associated to pk_i which has already been used on an input (pk_{i^*}, pk_i, C^*) to OReEnc, this oracle answers with $\perp$.

Definition 3 (IND-CCA security of $\mathsf{PRE.Encrypt}_2$)**.** *Let κ and n be two integers. Let $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$ be an adversary against the* CCA *indistinguishability of* $\mathsf{PRE.Encrypt}_2$*. Let* $\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_2,\mathscr{A}}(\kappa,n) := 2 \cdot \Pr\left[\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_2,\mathscr{A}}(\kappa,n) \rightarrow \mathit{true}\right] - 1$*, with* $\mathbf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_2,\mathscr{A}}$ *as defined in Fig. 1 We say that* PRE *has* IND-CCA *security of* $\mathsf{PRE.Encrypt}_2$ *if for every p.p.t. adversary $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$,* $\mathbf{Adv}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_2,\mathscr{A}}(\kappa,n)$ *is negligible.*

IND-CCA SECURITY OF PRE.ReEncrypt**.** In the security game for the indistinguishability under CCA of PRE.ReEncrypt, the two-stage attacker $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$ can specify the delegator i'. In this case, he is not allowed to attack a key $pk_{i^\star}$ if it is those of a corrupted user. The sole restriction in this case, and contrary to Chow *et al.,* is that the key $pk_{i^\star}$ must be a key of an uncorrupted user. There is

no restriction on the key of the delegator $pk_{i'}$, which can be one of a corrupted user. Therefore, the oracles and their restrictions are the same as in the case of the IND-CCA security of PRE.Encrypt1.

Definition 4 (IND-CCA security of PRE.ReEncrypt)**.** *Let κ and n be integers. Let $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$ be an adversary against the* CCA *indistinguishability of* PRE.ReEncrypt*. Let* $\mathsf{Adv}^{\text{ind-cca}}_{\mathsf{PRE.ReEncrypt},\mathscr{A}}(\kappa,n) := 2 \cdot \Pr\left[\mathsf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.Encrypt}_1,\mathscr{A}}(\kappa,n) \rightarrow \mathsf{true}\right] - 1$*, with* $\mathsf{Exp}^{\text{ind-cca}}_{\mathsf{PRE.ReEncrypt},\mathscr{A}}$ *as defined in Fig. 1 We say that* PRE *has* IND-CCA *security of* PRE.ReEncrypt *if for every p.p.t. adversary $\mathscr{A} = (\mathscr{A}_f, \mathscr{A}_g)$,* $\mathsf{Adv}^{\text{ind-cca}}_{\mathsf{PRE.ReEncrypt},\mathscr{A}}(\kappa,n)$ *is negligible.*

Lemma 1. *The* IND-CCA *security of* PRE.ReEncrypt *implies the Transformed Ciphertext Security of [7]*

Proof. We construct an algorithm $\mathscr{B}$ which will attack the IND-CCA security of PRE.ReEncrypt out of an adversary $\mathscr{A}$ against the Transformed Ciphertext Security of IND-CCA security of [7]. If $\mathscr{A}$ asks $\mathscr{B}$'s oracles for a request, then $\mathscr{B}$ asks oracles for the same request and returns the result to $\mathscr{A}$. For the challenge phase, $\mathscr{A}$ outputs m_0, m_1, $pk_{i'}$, pk_{i^*}. $\mathscr{B}$ computes $C_0 = Encrypt_2(\mathscr{P}, sk_{i'}, pk_{i'}, m_0)$ and $C_1 = Encrypt_2(\mathscr{P}, sk_{i'}, pk_{i'}, m_1)$. Those two messages are "good messages" which can be re-encrypted from $pk_{i'}$ to pk_{i^*}. $\mathscr{B}$ outputs C_0, C_1, $pk_{i'}$, pk_{i^*} in its challenge phase. It receives $C'^* = ReEncrypt(\mathscr{P}, R_{i'\rightarrow i^*}, C_\delta)$ and returns it to $\mathscr{A}$. $\mathscr{A}$ outputs δ' as answer to his challenge and $\mathscr{B}$ simply forwards it to its own challenger. We easily see that this algorithm allows to attack the IND-CCA security of PRE.ReEncrypt. □

Lemma 2. *A deterministic re-encryption cannot achieve* IND-CCA *security of* PRE.ReEncrypt.

Proof. The proof is trivial and left to the reader. □

MASTER SECRET KEY SECURITY. Another important security notion was suggested in [2] which refers to the impossibility of a coalition of dishonest delegatees to pool together their re-encryption keys to recover the secret key of their delegator. This notion is actually implied by the IND-CCA security of $\mathsf{PRE.Encrypt}_1$, where all re-encryption keys are given to the attacker (see *e.g.* [14]). Essentially, the difficulty of the discrete logarithm problem is sufficient to prove this notion in our case.

2.3 Toolbox

In the following, $[\![a;b]\!]$ denotes the set of all integers between a and b.

2.3.1 Complexity Assumptions

Our modified scheme rely on the computational Diffie-Hellman assumption. The Computational Diffie-Hellman assumption (CDH) posits the hardness of computing g^{ab} given (g, g^a, g^b).

The Divisible Computation Diffie-Hellman assumption (DCDH), introduced by Bao *et al.* [3], posits the hardness of computing $g^{b/a}$ given (g, g^a, g^b). The DCDH assumption is shown in [3] to be equivalent to the CDH assumption in the same group.

2.3.2 Non-interactive zero-knowledge proofs with online extractors.

Roughly speaking, a zero knowledge proof of knowledge [10] describes the way an entity proves to a verifier that he knows secret values $\alpha_1, \ldots, \alpha_q$ verifying a given relation $\mathscr{R}$ (that is $\mathscr{R}(\alpha_1, \ldots, \alpha_q) = 1$) without revealing anything about the secrets. In the following, we need a non-interactive proof of

knowledge (NIZK) with an *online extractor* to extract the secret during the security proof. In [7], Chow *et al.* use a non-interactive version of the Schnorr protocol [16], using the Fiat-Shamir heuristic [11] in the random oracle model. In our context, this makes the reduction in the security proof not polynomial time because of the rewinds to get some secret keys during the proof. For our purpose, in order to get a proof of the IND-CCA security, we use Fischlin's constructions from [12]. More precisely Fischlin shows how to turn out an interactive proof of knowledge into a NIZK with online extractors (NIZKOE). This construction is proven to be secure in the random oracle model and can be used for any discrete logarithm relation, which includes the Schnorr one. The NIZKOE we are interested in is denoted NIZKOE$(r : E = T^r)(m)$ where r is the secret key known by the prover, E and T are public values and m is a message to be signed.

Let κ denotes the security parameter. Roughly speaking, this signature is produced by first choosing at random values $\mathtt{com}_1, .., \mathtt{com}_l \in \mathbb{Z}_q^*$ with $l = O(\log(\kappa))$ and computing $T_k = T^{\mathtt{com}_k}$, $\forall k \in [\![1; l]\!]$. Let $\mathscr{H}_3$ be a hash function modelled as a random oracle mapping to $\{0,1\}^u$ with $u = O(\log(\kappa))$ and $u \leq t$. Then for each $k \in [\![1; l]\!]$, for each $ch_k \in [\![1; 2^t]\!]$ with $t = O(\log(\kappa))$ and $2^t \leq q$, the signer computes $\mathtt{resp}_k = \mathtt{com}_k + r.\mathtt{ch}_k$, until one verifies the relation $\mathscr{H}_3(E, T_1, .., T_l, k, \mathtt{ch}_k, \mathtt{resp}_k, m) = 0^u$. If no such tuple is found, then one picks the first one for which the hash value is minimal among all 2^t hash values. The signature consists of $(T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1,..,l}$. The verification procedure consists in checking that if $T^{\mathtt{resp}_k} = T_k E^{\mathtt{ch}_k}$, for each $k \in [\![1; l]\!]$ and if $\sum_{k=1}^{l} \mathscr{H}_3(E, T_1, ..T_l, k, \mathtt{ch}_k, \mathtt{resp}_k, m) \leq S$ with $S = O(l)$.

As explained in [12], we can extract the secret r in the random oracle model, except with negligible probability. Let $(T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1,..,l}$ be a valid signature for public values E and T on the message m. By browsing the list of queries to the random oracle $\mathscr{H}_3$, we can find one such query $(E, T_1, ..T_l, k, \mathtt{ch}'_k, \mathtt{resp}'_k, m)$ with $\mathtt{ch}'_k \neq \mathtt{ch}_k$ but such that $T^{\mathtt{resp}'_k} = T_k E^{\mathtt{ch}'_k}$ for one $k \in [\![1; l]\!]$, which is possible with an overwhelming probability as a valid signature depends over the choice of $\mathscr{H}_3$. With such a query, as we have two Schnorr signatures for the same commit $\mathtt{com}_k$, it is easy to recover r by computing it as $r = \frac{\mathtt{resp}'_k - \mathtt{resp}_k}{\mathtt{ch}'_k - \mathtt{ch}_k}$.

2.3.3 Hash Elgamal encryption.

The Elgamal encryption scheme (naturally IND-CPA under the DDH assumption) can be made IND-CCA secure, in the random oracle model, thanks to Fujisaki-Okamoto's technique [13], and is called in this case the Hash Elgamal encryption scheme. For our purpose, the encryption process consists in computing $T_0 = (m \| r) \oplus \mathscr{H}_2(g^{\mathscr{H}_1(m,r)})$ and $T_1 = y^{\mathscr{H}_1(m,r)}$ where m is the message, g is a generator of the used group, $y = g^x$ is the public key, related to the secret key x, and $\mathscr{H}_1$ and $\mathscr{H}_2$ are hash functions modeled as random oracles. The decryption of (T_0, T_1) is done by computing $m \| r$ as $T_0 \oplus \mathscr{H}_2(T_1^{1/x})$ and checking that $T_1 = y^{\mathscr{H}_1(m,r)}$ to recover m.

3 The modified scheme

In this section, we give the general framework of the Chow *et al.* scheme [7] and next proposed our corrections to improve its security.

3.1 Review of Chow *et al.* scheme

We first review the important points of the design of Chow *et al.*'s unidirectional PRE scheme [7] (keeping their notations in a discrete logarithm setting) by giving few words about the main steps of their scheme (in particular, we avoid the Setup and the Enc_2 phases, which are not crucial for the understanding).

- $\mathsf{Keygen}()$. User i has a secret key $(x_{i,1}, x_{i,2})$ and a public one $(g_{i,1}, g_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$.
- $\mathsf{ReKeyGen}(sk_i, pk_j)$. The generation of re-encryption keys is somehow original since it does not generate, as usual, a re-encryption key from the key of both users i and j of the form g^{x_i/x_j}. In fact, it generates this key using the key of user i and a fresh randomly picked key, denoted h. The re-encryption key is of the form $R_{i\to j} = \frac{h}{x_{i,1}\mathscr{H}_4(g_{i,2})+x_{i,2}}$ and a Hash Elgamal encryption of h, w.r.t. the public key $g_{j,2}$ of user j, is added.
- $\mathsf{Enc}(pk_i, m)$. The encryption of the message m is done with a Hash Elgamal encryption. The message is $(m\|w)$, where w is a random, the public key is $g_{i,1}^{\mathscr{H}_4(g_{i,2})} g_{i,2}$ (which is related to the secret key $x_{i,1}\mathscr{H}_4(g_{i,2}) + x_{i,2}$) and the random number is $r = \mathscr{H}_1(m, w)$. To reach the IND-CCA property (and more precisely to allow the access to a decryption oracle), the proposed scheme makes use of a Schnorr signature with the secret key r.
- $\mathsf{ReEnc}(R_{i\to j}, C, pk_i, pk_j)$. If the ciphertext C is correct[2] (*i.e.,* if the Schnorr signature is correct), the re-encryption procedure uses the re-encryption key $R_{i\to j}$ to transform the Hash Elgamal encryption of m and the public key $g_{i,1}^{\mathscr{H}_4(g_{i,2})} g_{i,2}$ into the encryption of the same message m but with the public key g^h. The Hash Elgamal encryption of h with the public key $g_{j,2}$ is also returned.
- $\mathsf{Dec}_1(sk_j, C')$. The decryption of a level 1 ciphertext is done by using $x_{j,2}$ to retrieve the secret value h, which is then used to decrypt the couple $(m\|w)$.
- $\mathsf{Dec}_2(sk_i, C)$. The decryption of a level 2 ciphertext is a Hash Elgamal decryption with secret key $x_{i,1}\mathscr{H}_4(g_{i,2}) + x_{i,2}$ and the verification of the Schnorr signature and of the computation of r as $\mathscr{H}_1(m, w)$.

3.2 Discussion

As mentioned in the introduction, this scheme has been proven in [7] to be CCA secure. However, looking at the proposed security model and the corresponding security proof, we argue that it is possible to add some improvements to the scheme, the model and the proofs, following the three points below.

(1) First, we note that the ReEnc procedure always outputs the same re-encryption when applied on the same ciphertext and users: this procedure is deterministic.

This can be avoided by generating different re-encryption keys (as they are random from j's point of view) but the result is not very practical. Another solution is to add randomness to the resulting ciphertext. When applied to the Hash Elgamal encryption $(E = (g_{i,1}^{\mathscr{H}_4(g_{i,2})} g_{i,2})^r, F = \mathscr{H}_2(g^r) \oplus (m\|w))$ of the message $(m\|w)$, the re-encryption procedure already modifies E but not the value F. In our modified scheme, we use for that purpose a one-time-pad on the value F, using a fresh key $\mathscr{H}_2(g^z)$. The value z is next encrypted, using Hash Elgamal, with the key $x_{j,2}$.

[2] which does not mean it is well-formed

Thanks to this, it is now possible to prove the CCA security of the re-encryption while giving the adversary the possibility to both corrupt the user i' who is the original recipient of the challenged ciphertext, and obtain the re-encryption key from i' to the targeted users. Note that this restriction was said by the authors of [7] to be weaker than the definition given in [14], that only obtained a RCCA security. Consequently, we obtain a scheme with the advantages and none of the drawbacks of [7, 14].

(2) In the same CCA security at level 1, the authors conclude by saying that some re-encryption keys can be given to the adversary while in the security proof given in [7], this is not allowed. More precisely, according to the simulation of the re-keygen oracle, the re-encryption keys from i' or $i^\star$ to a corrupted user cannot be given to the adversary. In our proof, given in Section 4, we fix it.

(3) Finally, regarding CCA security at level 2, the proof given in [7] sometimes aborts the experiment while it will actually be possible to conclude as we will see. More precisely, in the simulation of the ReEnc procedure, if the used value r does not come from the random oracle, then the experiment is aborted. In our modified scheme, we use a NIZKOE, to extract without rewind, from the signature, the secret r, using the random oracle related to this signature.

3.3 Our Modified Scheme

Using the above remarks and high-level description, we now give the complete updated scheme.

$\mathsf{Setup}(\kappa)$: Choose p and q two primes such that $q|p-1$ and the bit-length of q is the security parameter κ. Let g be a generator of $\mathbb{G}$, which is a subgroup of $\mathbb{Z}_p^\star$ of order q. Choose three hash functions $\mathscr{H}_1 : \{0,1\}^{l_0} \times \{0,1\}^{l_1} \to \mathbb{Z}_q^\star$, $\mathscr{H}_2 : \mathbb{G} \to \{0,1\}^{l_0+l_1}$ and $\mathscr{H}_4 : \mathbb{G} \to \mathbb{Z}_q^\star$. Here l_0 and l_1 are security parameters polynomial in κ and the message space is $\{0,1\}^{l_0}$. The global parameters are $\mathscr{P} = (\kappa, q, \mathbb{G}, g, l_0, l_1, \mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \mathscr{H}_4)$ ($\mathscr{H}_3$ comes from Fischlin's construction, see Section 2.3).

$\mathsf{Keygen}(\mathscr{P})$: On input the public parameters, this algorithm picks $sk = (x_1, x_2) \xleftarrow{\$} \mathbb{Z}_q^\star \times \mathbb{Z}_q^\star$ and sets $pk = (pk_1, pk_2) = (g^{x_1}, g^{x_2})$.

$\mathsf{ReKeygen}(\mathscr{P}, sk_i, pk_i, pk_j)$: On input the public parameters, user i's private key $sk_i = (x_{i,1}, x_{i,2})$ and public key $pk_i = (pk_{i,1}, pk_{i,2})$ and user j's public key $pk_j = (pk_{j,1}, pk_{j,2})$, this algorithm generates the re-encryption key $R_{i\to j}$ as follows.

1. Pick $h \xleftarrow{\$} \{0,1\}^{l_0}$ and $\varpi \xleftarrow{\$} \{0,1\}^{l_1}$, compute $v = \mathscr{H}_1(h, \varpi)$.
2. Compute $V = pk_{j,2}^v$ and $W = \mathscr{H}_2(g^v) \oplus (h \| \varpi)$.
3. Define $\tilde{R}_{i\to j} = \frac{h}{x_{i,1}\mathscr{H}_4(g^{x_{i,2}}) + x_{i,2}}$ and return $R_{i\to j} = (pk_i, pk_{j,2}, \tilde{R}_{i\to j}, V, W)$.

$\mathsf{Encrypt}_1(\mathscr{P}, pk_i, m)$: On input the public parameters, user i's public key $pk_i = (pk_{i,1}, pk_{i,2})$ and a plaintext $m \in \{0,1\}^{l_0}$, this algorithm creates a first level ciphertext of m intended to user i as follows.

1. Pick $h \xleftarrow{\$} \{0,1\}^{l_0}$, $\varpi \xleftarrow{\$} \{0,1\}^{l_1}$ and compute $v = \mathscr{H}_1(h, \varpi)$.
2. Compute $V = pk_{i,2}^v$ and $W = \mathscr{H}_2(g^v) \oplus (h \| \varpi)$.
3. Pick $\omega \xleftarrow{\$} \{0,1\}^{l_1}$ and compute $r = \mathscr{H}_1(m, \omega)$, $E' = g^{rh}$.
4. Pick $z \xleftarrow{\$} \in \{0,1\}^{l_0}$, $\varpi_2 \xleftarrow{\$} \in \{0,1\}^{l_1}$ and compute $x = \mathscr{H}_1(z, \varpi_2)$, $X = pk_{j,2}^x$, $Y = \mathscr{H}_2(g^x) \oplus (z \| \varpi_2)$ and $F' = \mathscr{H}_2(g^z) \oplus \mathscr{H}_2(g^r) \oplus (m \| \omega)$.

5. Output the first level ciphertext $C' = (E', F', V, W, X, Y)$.

$\mathsf{Encrypt}_2(\mathscr{P}, pk_i, m)$: On input the public parameters, user i's public key $pk_i = (pk_{i,1}, pk_{i,2})$ and a plaintext $m \in \{0,1\}^{l_0}$, this encryption algorithm works as below.

1. Pick $\omega \xleftarrow{\$}$ and compute $r = \mathscr{H}_1(m, \omega)$.
2. Compute $E = (pk_{i,1}^{\mathscr{H}_4(pk_{i,2})} pk_{i,2})^r$ and $F = \mathscr{H}_2(g^r) \oplus (m \| \omega)$.
3. Define $(T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1..l} = \mathrm{NIZKOE}(r : E = (pk_{i,1}^{\mathscr{H}_4(pk_{i,2})} pk_{i,2})^r)(F)$ and output the second level ciphertext $C = (E, F, (T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1..l})$.

$\mathsf{ReEncrypt}(\mathscr{P}, R_{i\to j}, C)$: On input the public parameters, a re-encryption key $R_{i\to j} = (pk_i,\ pk_{j,2},\ \tilde{R}_{i\to j},\ V, W)$ and a second level ciphertext $C = (E, F, (T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1..l})$ intended to user i, this algorithm re-encrypts this ciphertext into another one intended to user j as follows.

1. If $(T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1..l}$ is not valid for F under E and pk_i, return $\perp$.
2. Otherwise, compute $E' = E^{\tilde{R}_{i\to j}}$.
3. Pick $z \xleftarrow{\$} \in \{0,1\}^{l_0}$, $\varpi_2 \xleftarrow{\$} \in \{0,1\}^{l_1}$ and compute $x = \mathscr{H}_1(z, \varpi_2)$, $X = pk_{j,2}^x$, $Y = \mathscr{H}_2(g^x) \oplus (z \| \varpi_2)$ and $F' = \mathscr{H}_2(g^z) \oplus F$.
3. Output the first level ciphertext $C' = (E', F', V, W, X, Y)$.

$\mathsf{Decrypt}_1(\mathscr{P}, sk_i, C')$: On input user i's private key $sk_i = (x_{i,1}, x_{i,2})$ and a first level ciphertext $C' = (E', F', V, W, X, Y)$, this algorithm works as follows.

1. Compute $(z \| \varpi_2) = Y \oplus \mathscr{H}_2(X^{1/x_{i,2}})$, $(h \| \varpi) = W \oplus \mathscr{H}_2(V^{1/x_{i,2}})$ and $(m \| \omega) = F' \oplus \mathscr{H}_2(E'^{1/h}) \oplus \mathscr{H}_2(g^z)$.
2. If $X = pk_{i,2}^{\mathscr{H}_1(z, \varpi_2)}$, $V = pk_{i,2}^{\mathscr{H}_1(h, \varpi)}$ and $E' = g^{\mathscr{H}_1(m,\omega)h}$ then return m, else return $\perp$.

$\mathsf{Decrypt}_2(\mathscr{P}, sk_i, C)$: On input user i's private key $sk_i = (x_{i,1}, x_{i,2})$ and a second level ciphertext $C = (E, F, (T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1..l})$, this algorithm works as follows.

1. If $(T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1..l}$ is not valid for F under E and pk_i, return $\perp$.
2. Otherwise, compute $(m \| \omega) = F \oplus \mathscr{H}_2(E^{\frac{1}{x_{i,1}\mathscr{H}_4(pk_{i,2}) + x_{i,2}}})$.
3. If $E = (pk_{i,1}^{\mathscr{H}_4(pk_{i,2})} pk_{i,2})^r$ then return m, else return $\perp$.

The correctness of the scheme is easy to verify and is left to the reader.

4 Security

This section includes the security results (in the sense of Def. 2, 3, 4) concerning our modification of Chow *et al.*'s scheme. The algorithmic assumptions underlying our security proofs are described in Appendix 2.3.1.

In the sequel, $q_{\mathscr{H}_1}, q_{\mathscr{H}_2}, q_{D_1}, q_{D_2}, q_{rk}, q_{re}$ denote respectively the number of queries to the random oracle $\mathscr{H}_1$, $\mathscr{H}_2$, to the decryption oracle for first level ciphertexts, to the decryption oracle for second level ciphertexts, to the re-encryption key generation oracle and to the re-encryption oracle. Let τ denote the negligible probability of success for an adversary against the CCA security of our Hash

Elgamal encryption given in Section 2.3.3. Let ϕ, denote the negligible probability of non-success to extract the secret from a NIZKOE and let ψ, denote the negligible probability of non-success to simulate a valid proof of knowledge associated to a NIZKOE (see [12]).

4.1 IND − CCA security of PRE.Encrypt$_2$

Theorem 1. *The scheme has* IND-CCA *security of* PRE.Encrypt$_2$ *under the* CDH *assumption in the random oracle model.*

Proof. We construct an algorithm $\mathscr{B}$ which breaks the DCDH assumption out of an adversary $\mathscr{A}$ against the IND-CCA security of PRE.Encrypt$_2$ in the random oracle model with probability greater than

$$\frac{\mathsf{Adv}^{\text{ind-cca}}_{\text{PRE.Encrypt}_2,\mathscr{A}}(\kappa)}{q_{\mathscr{H}_2}e(1+n_c+q_{rk}+q_{re})} - \frac{q_{\mathscr{H}_1}}{q_{\mathscr{H}_2}2^{l_1}} - \frac{\psi}{q_{\mathscr{H}_2}} - \frac{(q_{rk}+q_{D_2})\phi}{q_{\mathscr{H}_2}} - \max\left(\frac{q_{D_1}\tau}{q_{\mathscr{H}_2}}, \frac{3q_{D_1}}{qq_{\mathscr{H}_2}}\right).$$

Let (g, g^a, g^b) be a DCDH instance, the aim of $\mathscr{B}$ is to compute $g^{b/a}$. In a preparation phase, $\mathscr{B}$ provides $\mathscr{A}$ with public parameters $(\kappa, q, \mathbb{G}, g, l_0, l_1, \mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \mathscr{H}_4)$. Random oracles $\mathscr{H}_1$, $\mathscr{H}_2$ and $\mathscr{H}_3$ (for the Fischlin construction) are controlled by $\mathscr{B}$, who maintains three lists: $\mathscr{H}_1^{\mathsf{list}}$, $\mathscr{H}_2^{\mathsf{list}}$ and $\mathscr{H}_3^{\mathsf{list}}$, which are initialized as empty. $\mathscr{B}$ answers to the queries to random oracles as follows:

- $(E, T_1, .., T_l, k, \mathtt{ch}, \mathtt{resp}, m)$ is asked to $\mathscr{H}_3$: if there is a tuple $(E, T_1, .., T_l, k, \mathtt{ch}, \mathtt{resp}, m, c)$ in $\mathscr{H}_3^{\mathsf{list}}$ then return c, otherwise choose $c \xleftarrow{\$} \{0,1\}^u$, add the tuple $(E, T_1, .., T_l, k, \mathtt{ch}, \mathtt{resp}, m, c)$ to $\mathscr{H}_3^{\mathsf{list}}$ and return c.
- (m, ω) is asked to $\mathscr{H}_1$: if there is a tuple (m, ω, r) in $\mathscr{H}_1^{\mathsf{list}}$ then return r, otherwise choose $r \xleftarrow{\$} \mathbb{Z}_q\star$, add the tuple (m, ω, r) to $\mathscr{H}_1^{\mathsf{list}}$ and return r.
- R is asked to $\mathscr{H}_2$: if there is a tuple (R, β) in $\mathscr{H}_2^{\mathsf{list}}$ then return β, otherwise choose $\beta \xleftarrow{\$} \{0,1\}^{l_0+l_1}$, add the tuple (R, β) to $\mathscr{H}_2^{\mathsf{list}}$ and return β.

$\mathscr{B}$ maintains three lists: $\mathscr{K}^{\mathsf{list}}$, $\mathscr{R}^{\mathsf{list}}$ and $\mathscr{C}^{\mathsf{list}}$, which are initially set as empty and will store public/private keys, re-encryption keys and some ciphertext with its re-encryption in some special cases.

Key generation: $\mathscr{B}$ generates public-keys as follows: $\mathscr{B}$ picks $(x_{i,1}, x_{i,2}) \in \mathbb{Z}_q^\star \times \mathbb{Z}_q^\star$ and uses Coron's technique [8]: c_i is a bit set at 1 with probability θ and at 0 otherwise (θ will be chosen later).

- if $c_i = 1$, $\mathscr{B}$ defines $pk_i = (pk_{i,1}, pk_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$.
- if $c_i = 0$, $\mathscr{B}$ defines $pk_i = (pk_{i,1}, pk_{i,2}) = ((g^a)^{x_{i,1}}, (g^a)^{x_{i,2}})$.

Then $\mathscr{B}$ adds the tuple $(pk_i, x_{i,1}, x_{i,2}, c_i)$ to $\mathscr{K}^{\mathsf{list}}$ and returns pk_i to $\mathscr{A}$.

Phase 1. $\mathscr{B}$ answers different queries of $\mathscr{A}$ as follows:

OSecKey($\mathsf{pk_i}$) : $\mathscr{B}$ begins by recovering the entry corresponding to pk_i from $\mathscr{K}^{\mathsf{list}}$. If $c_i = 0$, B outputs "failure" and aborts, otherwise it returns $sk_i = (x_{i,1}, x_{i,2})$ to $\mathscr{A}$.

$\mathsf{OReKG}(\mathsf{pk_i}, \mathsf{pk_j})$: $\mathscr{B}$ begins by recovering the entries corresponding to pk_i and pk_j from $\mathscr{K}^{\mathsf{list}}$ and next generates re-encryption keys as follows:

- $c_i = 1$: $\mathscr{B}$ runs fairly the $\mathsf{ReKeygen}$ algorithm.
- $c_i = c_j = 0$: $\mathscr{B}$ generates a random (fake) re-encryption key. Indeed, it picks $\tilde{R}_{i\to j} \xleftarrow{\$} \mathbb{Z}_q^\star$, $h \xleftarrow{\$} \{0,1\}^{l_0}$, $\varpi \xleftarrow{\$} \{0,1\}^{l_1}$ and computes $v = \mathscr{H}_1(h, \varpi)$, $V = pk_{j,2}^v$, $W = \mathscr{H}_2(g^v) \oplus (h \| \varpi)$.
- $c_i = 0$ and $c_j = 1$: $\mathscr{B}$ aborts and outputs "failure".

If $\mathscr{B}$ does not abort, then it adds the tuple $(pk_i, pk_j, \tilde{R}_{i\to j}, V, W, h)$ to $\mathscr{R}^{\mathsf{list}}$ and returns $R_{i\to j} = (pk_i, pk_{j,2}, \tilde{R}_{i\to j}, V, W)$ to $\mathscr{A}$.

$\mathsf{OReEnc}(\mathsf{pk_i}, \mathsf{pk_j}, \mathsf{C})$: the ciphertext C is parsed as $(E, F, (T_k, ch_k, resp_k)_{k=1..l})$. In the case where $(T_k, ch_k, resp_k)_{k=1..l}$ is not valid for F under E and pk_i, return $\perp$. Otherwise, there are several cases.

- $c_i = 1$ or $c_i = c_j = 0$: recover the re-encryption key $R_{i\to j}$ (or ask $\mathsf{OReKG}(pk_i, pk_j)$ as described above), then re-encrypt to obtain C' and add (i, j, C, C') to $\mathscr{C}^{\mathsf{list}}$. If $c_i = c_j = 0$, the plaintext related to C' can not correspond to the one related to C as (see above) the re-encryption key is fake. As $\mathscr{A}$ may have this fake key in possession, $\mathscr{B}$ should re-encrypt "correctly", even in this case. Note that we need to consider this case in the simulation of the decryption (see below).
- $c_i = 0$ and $c_j = 1$: $\mathscr{B}$ does not know the re-encryption key and cannot work similarly. In this case, Chow *et al.* [7] use the random oracle $\mathscr{H}_1$ to retrieve r. But, as the adversary may have created this level 2 ciphertext without asking the random oracle, they sometimes need to abort. On contrary, we use in this case the Fischlin NIZKOE (see [12] and Section 2.3) which permits us to always extract the secret r. This way, $\mathscr{B}$ will be able to re-encrypt correctly as follows.
  - If $\mathscr{R}^{\mathsf{list}}$ has an entry $(pk_i, pk_j, \perp, V, W, h)$, recover it. Otherwise pick $h \xleftarrow{\$} \{0,1\}^{l_0}$, $\varpi \xleftarrow{\$} \{0,1\}^{l_1}$, compute $v = \mathscr{H}_1(h, \varpi)$, $V = pk_{j,2}^v$ and $W = \mathscr{H}_2(g^v) \oplus (h \| \varpi)$. Add $(pk_i, pk_j, \perp, V, W, h)$ to $\mathscr{R}^{\mathsf{list}}$.
  - Pick $z \xleftarrow{\$}\in \{0,1\}^{l_0}$, $\varpi_2 \xleftarrow{\$}\in \{0,1\}^{l_1}$ and compute $E' = g^{rh}$, $x = \mathscr{H}_1(z, \varpi_2)$, $X = pk_{j,2}^x$, $Y = \mathscr{H}_2(g^x) \oplus (z \| \varpi_2)$ and $F' = \mathscr{H}_2(g^z) \oplus F$.
  - Output the first level ciphertext $C' = (E', F', V, W, X, Y)$.

$\mathsf{ODec_2}(pk_i, C)$: $\mathscr{B}$ decrypts a second level ciphertext as follows.

- $c_i = 1$: run $\mathsf{Decrypt_2}(\mathscr{P}, sk_i, C)$.
- $c_i = 0$: in this case, $\mathscr{B}$ does not have the decryption key. As for the OReEnc oracle above, we use the Fischlin NIZKOE to extract r (which, again, was not always possible in [7]). This way, $\mathscr{B}$ will be able to decrypt correctly. For this purpose, it computes $(m \| \omega) = F \oplus \mathscr{H}_2(g^r)$ and return m (and $\perp$ otherwise) if $E = (pk_{i,1}^{\mathscr{H}_4(pk_{i,2})} pk_{i,2})^{\mathscr{H}_1(m,\omega)}$.

$\mathsf{ODec_1}(pk_j, C')$: the ciphertext C' is parsed as (E', F', V, W, X, Y). $\mathscr{B}$ decrypts this ciphertext as follows. If $c_j = 1$, it runs $\mathsf{Decrypt_1}(\mathscr{P}, sk_j, C')$. Otherwise we distinguish the two following cases.

- If there is an entry $(pk_i, pk_j, R_{i\to j}, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$, then there are two cases.
  - $c_i = 0$: if there is an entry (i, j, C, C') in $\mathscr{C}^{\mathsf{list}}$, call $\mathsf{ODec}_2(pk_i, C)$ and return the result to $\mathscr{A}$. Otherwise, if possible, use the random oracles $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover z from (X,Y), then compute $F = F' \oplus \mathscr{H}_2(g^z)$ and compute $E = E'^{\tilde{R}_{i\to j}}$ and if possible use the random oracle $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover m from (E,F) and return m if it succeeds. Else, return $\perp$.
  - $c_i = 1$: $\mathscr{B}$ first obtains h by recovering the entry $(pk_i, pk_j, R_{i\to j}, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$. Again, $\mathscr{B}$ can use $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover z from (X,Y) (otherwise, return $\perp$). Then $\mathscr{B}$ computes $(m \| \omega) = F' \oplus \mathscr{H}_2(g^z) \oplus \mathscr{H}_2(E'^{1/h})$ and returns either m if $E' = g^{\mathscr{H}_1(m,\omega)h}$ and $\perp$ otherwise.
- If there is no entry $(pk_i, pk_j, R_{i\to j}, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$, it uses the random oracles $\mathscr{H}_1$ and $\mathscr{H}_2$ to recover z, h and m. If it succeeds, it returns m, and it returns $\perp$ otherwise.

Challenge. $\mathscr{A}$ outputs an uncorrupted public key $pk_{i^\star} = (pk_{i^\star,1}, pk_{i^\star,2})$ and two messages m_0 and m_1.
If $c_i^\star = 1$, then $\mathscr{B}$ aborts and outputs "failure". Else:

- $\mathscr{B}$ picks $\delta \xleftarrow{\$} \{0,1\}$, $\omega^\star \xleftarrow{\$} \in \{0,1\}^{l_1}$. It computes $E^\star = (g^b)^{x_{i^\star,1}\mathscr{H}_4(pk_{i^\star,2}) + x_{i^\star,2}}$, implicitly defines $r = \mathscr{H}_1(m_\delta, \omega^\star) = b/a$.
- It uses the random oracle $\mathscr{H}_3$ to create a valid NIZKOE on a random $F^\star$ under $E^\star$ and $pk_i^\star$. For each k, let τ_k be a random application over $[\![1; 2^t]\!]$ mapping to $\{0,1\}^u$. Let $ch_k^\star$ be the smallest one obtaining the minimum over all these 2^t values. It then chooses at random $\mathtt{resp}_k^\star \in \mathbb{Z}_q^\star$ and computes $T_k^\star = (g^a)^{\mathtt{resp}_k^\star (x_{i^\star,1}\mathscr{H}_4(pk_{i^\star,2}) + x_{i^\star,2})} (g^b)^{-\mathtt{ch}_k^\star (x_{i^\star,1}\mathscr{H}_4(pk_{i^\star,2}) + x_{i^\star,2})}$. It implicitly defines $\mathtt{com}_k^\star = \mathtt{resp}_k^\star - \frac{b}{a}\mathtt{ch}_k^\star$. It picks $F^\star \xleftarrow{\$} \{0,1\}^{l_0+l_1}$ and defines the value $H_3(E^\star, T_1^\star, .., T_l^\star, k, ch_k^\star, resp_k^\star, F^\star) = \tau_k(ch_k^\star)$.
- $\mathscr{B}$ gives $(E^\star, F^\star, (T_k^\star, \mathtt{ch}_k^\star, \mathtt{resp}_k^\star)_{k=1..l})$ to $\mathscr{A}$.

Phase 2. $\mathscr{B}$ answers $\mathscr{A}$'s queries as in phase 1 for $\mathsf{OSecKey}$, OReKG, $\mathsf{ODec2}$ with restrictions of the $\mathsf{IND\text{-}CCA}$ security $\mathsf{PRE.Encrypt}_2$. It differs a bit for OReEnc and $\mathsf{ODec1}$, as $\mathscr{B}$ has to check if the input of a query to these oracles include the challenge ciphertext or a re-encryption of it. $\mathscr{B}$ answers different queries of $\mathscr{A}$ to both oracles as follows:

$\mathsf{OReEnc}(\mathsf{pk_i}, \mathsf{pk_j}, \mathsf{C})$: the ciphertext C is parsed as $(E, F, (T_k, ch_k, resp_k)_{k=1..l})$. In the case where $(T_k, ch_k, resp_k)_{k=1..l}$ is not valid for F under E and pk_i, return $\perp$. Otherwise, there are several cases.

- $c_i = 1$ or $c_i = c_j = 0$: recover the re-encryption key $R_{i\to j}$ (or ask $\mathsf{OReKG}(pk_i, pk_j)$ as described above), then re-encrypt to obtain C' and add (i, j, C, C') to $\mathscr{C}^{\mathsf{list}}$. If $c_i = c_j = 0$, the plaintext related to C' can not correspond to the one related to C as (see above) the re-encryption key is fake. As $\mathscr{A}$ may have this fake key in possession, $\mathscr{B}$ should re-encrypt "correctly", even in this case. Note that we need to consider this case in the simulation of the decryption (see below).
- $c_i = 0$ and $c_j = 1$: if $i = i^*$ and $C = C^*$, then $\mathscr{B}$ aborts and outputs "failure". Otherwise, $\mathscr{B}$ does not know the re-encryption key and cannot work similarly. In this case, Chow *et al.* [7] use the random oracle $\mathscr{H}_1$ to retrieve r. But, as the adversary may have created

this level 2 ciphertext without asking the random oracle, they sometimes need to abort. On contrary, we use in this case the Fischlin NIZKOE (see [12] and Section 2.3) which permits us to always extract the secret r. This way, $\mathscr{B}$ will be able to re-encrypt correctly as follows.

- If $\mathscr{R}^{\mathsf{list}}$ has an entry $(pk_i, pk_j, \perp, V, W, h)$, recover it. Otherwise pick $h \xleftarrow{\$} \{0,1\}^{l_0}$, $\varpi \xleftarrow{\$} \{0,1\}^{l_1}$, compute $v = \mathscr{H}_1(h, \varpi)$, $V = pk_{j,2}^v$ and $W = \mathscr{H}_2(g^v) \oplus (h \| \varpi)$. Add $(pk_i, pk_j, \perp, V, W, h)$ to $\mathscr{R}^{\mathsf{list}}$.
- Pick $z \xleftarrow{\$} \in \{0,1\}^{l_0}$, $\varpi_2 \xleftarrow{\$} \in \{0,1\}^{l_1}$ and compute $E' = g^{rh}$, $x = \mathscr{H}_1(z, \varpi_2)$, $X = pk_{j,2}^x$, $Y = \mathscr{H}_2(g^x) \oplus (z \| \varpi_2)$ and $F' = \mathscr{H}_2(g^z) \oplus F$.
- Output the first level ciphertext $C' = (E', F', V, W, X, Y)$.

$\mathsf{ODec}_1(pk_j, C')$: the ciphertext C' is parsed as (E', F', V, W, X, Y). $\mathscr{B}$ decrypts this ciphertext as follows. If $c_j = 1$, it runs $\mathsf{Decrypt}_1(\mathscr{P}, sk_j, C')$. Otherwise we distinguish the two following cases.

- If there is an entry $(pk_i, pk_j, R_{i\to j}, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$, then there are three cases.
  - $c_i = 1$: $\mathscr{B}$ first obtains h by recovering the entry $(pk_i, pk_j, R_{i\to j}, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$. Again, $\mathscr{B}$ can use $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover z from (X, Y) (otherwise, return $\perp$). Then $\mathscr{B}$ computes $(m \| \omega) = F' \oplus \mathscr{H}_2(g^z) \oplus \mathscr{H}_2(E'^{1/h})$ and returns either m if $E' = g^{\mathscr{H}_1(m,\omega)h}$ and $\perp$ otherwise.
  - $i = i^*$: if there is an entry (i^*, j, C^*, C') in $\mathscr{C}^{\mathsf{list}}$, then $\mathscr{B}$ outputs $\perp$ as C' is a re-encryption of the challenge ciphertext. Else it uses the random oracles $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover z from (X, Y), otherwise it returns $\perp$. Then $\mathscr{B}$ computes $E = (E')^{1/\tilde{R}_{i^*\to j}}$ and $F = F' \oplus H_2(g^z)$. If $E = E^*$ and $F = F^*$, then it outputs $\perp$, as C' is a re-encryption of C^*. Otherwise if possible, it uses the random oracles $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover m from (E, F) and return m if it succeeds. Else, return $\perp$.
  - $c_i = 0$ and $i \neq i^*$: if there is an entry (i, j, C, C') in $\mathscr{C}^{\mathsf{list}}$, $\mathscr{B}$ calls $\mathsf{ODec}_2(pk_i, C)$ and returns the result to $\mathscr{A}$. Otherwise, if possible, it uses the random oracles $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover z from (X, Y), then computes $F = F' \oplus \mathscr{H}_2(g^z)$, $E = E'^{1/\tilde{R}_{i\to j}}$ and if possible it uses the random oracles $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover m from (E, F) and return m if it succeeds. Else, return $\perp$.
- If there is no entry $(pk_i, pk_j, R_{i\to j}, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$, it uses the random oracles $\mathscr{H}_1$ and $\mathscr{H}_2$ to recover z, h and m. If it succeeds, it returns m, and it returns $\perp$ otherwise.

Guess. $\mathscr{A}$ returns δ' and $\mathscr{B}$ picks a tuple $(R, \beta) \in \mathscr{H}_2^{\mathsf{list}}$ and returns R as solution to the DCDH instance.

ANALYSIS OF THE SIMULATION. We now want to compute the success probability of our algorithm $\mathscr{B}$. For this purpose, we study each step to detect where the simulation is not perfect.

1) The simulation of oracle $\mathscr{H}_4$ is perfect, as well as these of $\mathscr{H}_1$, $\mathscr{H}_2$, $\mathscr{H}_3$, except when $\frac{b}{a}$ is asked to $\mathscr{H}_1$, $g^{b/a}$ is asked to $\mathscr{H}_2$ and $(E^\star, T_1^\star, .., T_l^\star, k, ch_k^\star, resp_k^\star, F^\star)$ is asked to H_3 before the challenge phase for some $k \in [\![1; l]\!]$.

2) The simulation of the **key generaiton** is perfect.

3) If $\mathscr{B}$ does not abort, the simulation of $\mathsf{OSecKey}$ is perfect.

4) The simulation of OReKG is perfect, except when $c_i = c_j = 0$ while $\mathscr{B}$ does not abort. In this case, since $\mathscr{A}$ has neither h nor $x_{i,1}\mathscr{H}_4(pk_{i,2}) + x_{i,2}$, a real re-encryption key ($\tilde{R}_{i\to j} = \frac{h}{x_{i,1}\mathscr{H}_4(g^{x_{i,2}})+x_{i,2}}$) is computationally indistinguishable from a random value given by $\mathscr{B}$ since (i) the value $(x_{i,1}, x_{i,2})$ is unknown from $\mathscr{A}$ and (ii) under the indistinguishability of the hash Elgamal encryption scheme, and thus the CDH assumption [13]. Then if $\mathscr{B}$ does not abort, the simulation of OReKG is also perfect.

5) If $\mathscr{B}$ does not abort, the simulation of OReEnc in the second phase is the same as the one in the first phase (with some restrictions of the $\mathsf{IND\text{-}CCA}$ security $\mathsf{PRE.Encrypt}_2$), so we analyse the simulation of this oracle independtly of both phases. The simulation of OReEnc re-encrypts correctly with the help of the online extractor, which, according to [12], allows to recover the value r with a high probability $1-\phi$ (see [12]). The sole exception is the case $c_i = c_j = 0$. Here, $\mathscr{B}$ executes a true re-encryption but with a fake re-encryption key. As $\mathscr{A}$ does not have the corresponding decryption key, $\mathscr{A}$ cannot detect it and thus, the simulation of OReEnc is also perfect.

6) The simulation of ODec_2 is perfect too due to the online extractors with a high probability $1-\phi$ (see [12]).

7) The simulation of ODec_1: we see easily that the simulation $\mathsf{ODec1}$ in the second phase finally gives the same result as the one in the first phase (with restrictions of the $\mathsf{IND\text{-}CCA}$ security $\mathsf{PRE.Encrypt}_2$), so we analyse the simulation of this oracle independently of both phases. If $c_j = 1$, the simulation is perfect. Otherwise, we distinguish the two following cases.

   - There is no entry $(pk_i, pk_j, \tilde{R}_{i\to j}, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$. The simulation is perfect, except when the simulation gives $\perp$. This case implies that the oracle lists $\mathscr{H}_1^{\mathsf{list}}$ and $\mathscr{H}_2^{\mathsf{list}}$ cannot be used to recover z, h, m. In fact, the probability that the adversary comes up with a valid triple (z, ϖ_2, x) (resp. (h, ϖ, v) and (m, ω, r)) *w.r.t.* $\mathscr{H}_1$ but without requesting the random oracle is $\frac{1}{q}$.
   - There is an entry $(pk_i, pk_j, *, V, W, h)$ in $\mathscr{R}^{\mathsf{list}}$, then there are two cases. Except with probability $\leq \frac{1}{q}$, there is only one entry in $\mathscr{R}^{\mathsf{list}}$ with V, W for j so $\mathscr{B}$ will use the right entry with probability $1-\frac{1}{q}$. There are next two cases.
     * $c_i = 1$: $\mathscr{B}$ recovers the value h in the entry $(pk_i, pk_j, \tilde{R}_{i\to j}, V, W, h)$ of $\mathscr{R}^{\mathsf{list}}$. As this value necessarily corresponds to the correct value h, $\mathscr{B}$ next uses the random oracles $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover z and m. The simulation is also perfect, except when the simulation gives $\perp$, which occurs with the probability $\frac{1}{q}$, as seen above.
     * $c_i = 0$: if there is an entry (i, j, C, C'), the simulation runs the simulation of ODec_2 on (i, C) which is perfectly simulated as we saw it before. Otherwise, there are two cases.
       · If the ciphertext C' corresponds to the re-encryption of a ciphertext C under the fake key $R_{i\to j}$, then we are not able to use C as we have no way to obtain it. We thus use the random oracle $\mathscr{H}_1$ and/or $\mathscr{H}_2$ to recover z or return $\perp$ (which occurs with probability $\frac{1}{q}$). Using z, $\mathscr{B}$ can next compute $E = E'^{\tilde{R}_{i\to j}}$ and $F = F' \oplus \mathscr{H}_2(g^z)$ (as for a true decryption) and finally uses again the random oracle

$\mathscr{H}_1$ to recover m. The simulation is next perfect, except when the random oracle cannot be used, which occurs with probability $\frac{1}{q}$.

- Otherwise, the ciphertext C' is a direct level 1 ciphertext. First, $\mathscr{A}$ could not know the value h since it only knows (V,W) and does not have the corresponding decryption key. Thus, if the ciphertext C' is well-formed with h hidden in (V,W), the only way for $\mathscr{A}$ to obtain h is to break the IND-CCA security of hash ElGamal, which occurs with negligible probability τ. As a consequence, similarly as above, we can use the random oracles to output the correct message m, except with probability $\leq \frac{2}{q}$ when the random oracle are not useful.

8) If $\mathscr{B}$ does not abort, the simulation of the challenge is perfect if $(E^\star, T_1^\star, .., T_l^\star, k, ch_k^\star, resp_k^\star, F^\star)$ has not been asked to $\mathscr{H}_3$ before the challenge phase for any $k \in [\![1;l]\!]$.

We now consider the following events to compute our final success probability.

- Let $\mathscr{H}_1^\star$ be the event that $\mathscr{A}$ queries $(m_\delta, \omega^\star)$ to $\mathscr{H}_1$ during the game. The answer of this request is $\frac{b}{a}$, which is unknown.
- Let $\mathscr{H}_2^\star$ be the event that $\mathscr{A}$ queries $g^{\frac{b}{a}}$ to $\mathscr{H}_2$ during the game. It is the value we are looking for.
- Let $\mathscr{H}_3^\star$ be the event that $\mathscr{A}$ queries $(E^\star, T_1^\star, .., T_l^\star, k, ch_k, resp_k^\star, F^\star)$ is asked to H_3 before the challenge phase for some $k \in [\![1;l]\!]$.
- Let Abort be the event that $\mathscr{B}$ aborts and outputs "failure" in the game.
- Let OEErr be the event that there is a value r that $\mathscr{B}$ cannot recover with the help of the online extractor in OReEnc and ODec2.
- Let $\mathsf{D_1Err}$ be the event that the simulation of Decrypt1 gives something different from what it should.
- Let $\mathsf{Err} = (\mathscr{H}_1^\star \vee \mathscr{H}_2^\star \vee \mathscr{H}_3^\star \vee \mathsf{OEErr} \vee \mathsf{D_1Err}) | \neg \mathsf{Abort}$

Due to the randomness of the output of the random oracle $\mathscr{H}_2$, it is easily to see if Err does not occur, A will have any advantage greater than $\frac{1}{2}$ to win its challenge. So $\Pr(\delta = \delta' | \neg \mathsf{Err}) = \frac{1}{2}$. As explain in [7], we have $e_{\text{adv}} \leq \Pr(\mathsf{Err})$, where e_{adv} denote the advantage of $\mathscr{A}$ to win its challenge. We also obtain $\Pr(\mathscr{H}_2^\star) \geq \Pr(\neg\mathsf{Abort}) e_{\text{adv}} - \Pr(\mathscr{H}_1^\star) - \Pr(\mathsf{OEErr}) - \Pr(\mathsf{D_1Err})$. And as $\mathscr{B}$ picks a tuple $(R,\beta) \in \mathscr{H}_2^{\mathsf{list}}$ and returns R as solution to the challenge, if the event $\mathscr{H}_2^\star$ occurs, there is a probability of $\frac{1}{q_{\mathscr{H}_2}}$ to $\mathscr{B}$ to return the good one. The success probability of $\mathscr{B}$ to win the DCDH challenge is finally lower bounded by $\frac{\mathscr{H}_2^\star}{q_{\mathscr{H}_2}} \geq \frac{\Pr(\neg\mathsf{Abort})\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PRE.Encrypt}_2,\mathscr{A}}(\kappa)}{q_{\mathscr{H}_2}} - \frac{\Pr(\mathscr{H}_1^\star)}{q_{\mathscr{H}_2}} - \frac{\Pr(\mathscr{H}_3^\star)}{q_{\mathscr{H}_2}} - \frac{\Pr(\mathsf{OEErr})}{q_{\mathscr{H}_2}} - \frac{\Pr(\mathsf{D_1Err})}{q_{\mathscr{H}_2}}$. We now detail each remaining probabilities.

- $\Pr(\mathscr{H}_1^\star)$: at most, $q_{\mathscr{H}1}$ chances over 2^{l_1} to find $\omega^\star$, $\Pr(\mathscr{H}_1^\star) \leq \frac{q_{\mathscr{H}_1}}{2^{l_1}}$.
- $\Pr(\mathscr{H}_3^\star) \leq \psi$: see [12] for more details on this value.
- $\Pr(\mathsf{OEErr})$: the probability that $\mathscr{B}$ cannot recover the value r for one NIZKOE is ϕ (see [12] for more details on this value), so for q_{rk} requests to OReEnc and q_{D_2} requests to $\mathsf{ODeC_2}$, oracles which can use the extractor, $\Pr(\mathsf{OEErr}) \leq 1 - (1-\phi^{q_{rk}+q_{D_2}}) \leq (q_{rk}+q_{D_2})\phi$ as ϕ is negligible.

- $\Pr(\mathsf{D_1Err})$: using our remarks above, we clearly have $\Pr(\mathsf{D_1Err}) \leq \max(q_{D_1}\tau, \frac{2q_{D_1}}{q})$.
- $\Pr(\neg\mathsf{Abort})$: $\mathscr{B}$ aborts if (i) $\mathsf{OSecKey}$ is asked for pk_i with $c_i = 0$, (ii) OReKG is asked for (pk_i, pk_j) with $c_i = 0$ and $c_j = 1$, (iii) $c_i^\star = 1$ in the challenge or (iv) $i = i^*$, $C = C^*$, $c_j = 1$, with pk_j not yet associated to a corrupted user, in OReEnc of the second phase (if pk_j is associated to a corrupted user, then $\mathscr{A}$ is not allowed to ask for it). The fact that $c_i^\star \neq 1$ in the challenge phase is controlled with the value θ. For queries to the secret key oracle or to the re-encryption oracle, we cannot control the case where $i = i^\star$ because it is not a random entity among uncorrupted entities. But for the $\mathsf{IND\text{-}CCA}$ security $\mathsf{PRE.Encrypt_2}$, it is not a problem as $\mathscr{A}$ does not have access to the secret key of i^* or to the re-encryption key from $i^\star$ to j corrupted. So we have $\neg\mathsf{Abort}$: $c_i^\star = 0 \wedge (c_i = 1 \text{ in } \mathsf{OSecKey}) \wedge (c_i \neq 0 \vee c_j \neq 1 \text{ in } \mathsf{OReenc}) \wedge (i \neq i^* \vee C \neq C^* \vee c_j \neq 0 \vee pk_j$ is already associated to a corrupted user). We see easily that $\Pr(\neg\mathsf{Abort}) \geq \Pr(c_i^\star = 0 \wedge c_i = 1 \text{ in } \mathsf{OSecKey}, \mathsf{OReKG} \text{ and } \mathsf{OReEnc}) \geq (1-\theta)\theta^{q_{n_c}+q_{rk}+q_{re}}$ which is maximized at $\theta_{\text{opt}} = \frac{n_c+q_{rk}+q_{re}}{1+n_c+q_{rk}+q_{re}}$ by $\left(\frac{1}{1+n_c+q_{rk}+q_{re}}\right)\left(\frac{n_c+q_{rk}+q_{re}}{1+n_c+q_{rk}+q_{re}}\right)^{n_c+q_{rk}+q_{re}} \geq \frac{1}{e(1+n_c+q_{rk}+q_{re})}$, which proves the lower bound announced in the theorem.

□

4.2 IND-CCA security of $\mathsf{PRE.Encrypt_1}$ and PRE.ReEncrypt

Theorem 2. *The scheme has* $\mathsf{IND\text{-}CCA}$ *security of* $\mathsf{PRE.Encrypt_1}$ *and* PRE.ReEncrypt *under the* CDH *assumption in the random oracle model.*

Proof. We construct an algorithm $\mathscr{B}$ which breaks the DCDH assumption out of an adversary $\mathscr{A}$ against the $\mathsf{IND\text{-}CCA}$ security of $\mathsf{PRE.Encrypt_1}$ and PRE.ReEncrypt in the random oracle model with success probability greater than $\frac{\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PRE.Algo},\mathscr{A}}(\kappa)}{q_{\mathscr{H}_2}e(1+n_c)} - \frac{q_{\mathscr{H}_1}}{q_{\mathscr{H}_2}2^{l_1}} - \frac{q_{D_1}}{qq_{\mathscr{H}_2}}$ with $\mathsf{Algo} \in \{\mathsf{ReEncrypt}, \mathsf{Encrypt_1}\}$.

Let (g, g^a, g^b) be a DCDH instance, the aim of $\mathscr{B}$ is to compute $g^{b/a}$. In a preparation phase, $\mathscr{B}$ provides $\mathscr{A}$ with public parameters $(\kappa, q, \mathbb{G}, g, l_0, l_1, \mathscr{H}_1, \mathscr{H}_2, \mathscr{H}_3, \mathscr{H}_4)$. Random oracles $\mathscr{H}_1$, $\mathscr{H}_2$ and $\mathscr{H}_3$ (for the Schnorr signature) are controlled by $\mathscr{B}$, who maintains three lists: $\mathscr{H}_1^{\mathsf{list}}$, $\mathscr{H}_2^{\mathsf{list}}$ and $\mathscr{H}_3^{\mathsf{list}}$, which are initialized as empty and $\mathscr{B}$ answers to the queries to random oracles as show for the proof of Theorem 4.1. $\mathscr{B}$ maintains two lists $\mathscr{K}^{\mathsf{list}}$ and $\mathscr{R}^{\mathsf{list}}$, which are initially set as empty and will store public/private keys, re-encryption keys.

Key generation. $\mathscr{B}$ picks $(x_{i,1}, x_{i,2}) \in \mathbb{Z}_q^\star \times \mathbb{Z}_q^\star$ and uses Coron's technique [8]: c_i is a bit set at 1 with probability θ and at 0 otherwise (θ will be chosen later). If $c_i = 1$, $\mathscr{B}$ defines $pk_i = (pk_{i,1}, pk_{i,2}) = (g^{x_{i,1}}, g^{x_{i,2}})$. If $c_i = 0$, $\mathscr{B}$ defines $pk_i = (pk_{i,1}, pk_{i,2})$ with $pk_{i,2} = (g^a)^{x_{i,2}}$ and $pk_{i,1} = \left(\left(\frac{g}{g^a}\right)^{x_{i,2}/\mathscr{H}_4(pk_{i,2})}\right) g^{x_{i,1}}$, implicitly define $sk_{i,1} = \frac{(1-a)x_{i,2}}{\mathscr{H}_4(pk_{i,2})} + x_{i,1}$ and $sk_{i,2} = ax_{i,2}$. We remark that $sk_{i,1}\mathscr{H}_4(pk_{i,2}) + sk_{i,2} = (1-a)x_{i,2} + x_{i,1}\mathscr{H}_4(pk_{i,2}) + ax_{i,2} = x_{i,1}\mathscr{H}_4(pk_{i,2}) + x_{i,2}$, value that $\mathscr{B}$ knows. Then $\mathscr{B}$ adds the tuple $(pk_i, x_{i,1}, x_{i,2}, c_i)$ to $\mathscr{K}^{\mathsf{list}}$ and returns pk_i to $\mathscr{A}$.

Phase 1. $\mathscr{B}$ answers different queries of $\mathscr{A}$ as follows:

$\mathsf{OSecKey}(pk_i)$: $\mathscr{B}$ begins by recovering the entry corresponding to pk_i from $\mathscr{K}^{\mathsf{list}}$. If $c_i = 0$, $\mathscr{B}$ outputs "failure" and aborts, otherwise it returns $sk_i = (x_{i,1}, x_{i,2})$ to $\mathscr{A}$.

$\mathsf{OReKG}(pk_i, pk_j)$: $\mathscr{B}$ begins by recovering the entries corresponding to pk_i and pk_j from $\mathscr{K}^{\mathsf{list}}$. In all cases, $\mathscr{B}$ knows $sk_{i,1}\mathscr{H}_4(pk_{i,2}) + sk_{i,2}$, so it does as in ReEncrypt. Then it adds the tuple $(pk_i, pk_j, \tilde{R}_{i\to j}, V, W, h)$ to $\mathscr{R}^{\mathsf{list}}$ and returns $R_{i\to j} = (pk_i, pk_{j,2}, \tilde{R}_{i\to j}, V, W)$ to $\mathscr{A}$.

$\mathsf{ODec}_1(pk_j, C')$: If $c_j = 1$, $\mathscr{B}$ has the secret key $sk_{j,2}$, so it does as in $\mathsf{Decrypt1}$. If $c_j = 0$, $\mathscr{B}$ needs to recover z, h and m with the random oracles and returns m to $\mathscr{A}$ or $\perp$ if it cannot succeed.

Challenge for the IND-CCA security of $\mathsf{PRE.Encrypt_1}$. $\mathscr{A}$ outputs two messages m_0, m_1 and an uncorrupted key $pk_{i^\star} = (pk_{i^\star,1}, pk_{i^\star,2})$. If $c_i^* = 1$, then $\mathscr{B}$ aborts and outputs "failure". Else, $\mathscr{B}$ picks $h^\star \xleftarrow{\$} \{0,1\}^{l_0}$, $\varpi \xleftarrow{\$} \{0,1\}^{l_1}$ and computes $v^\star = \mathscr{H}_2(h^\star, \varpi^\star)$, $V^\star = pk_{i^\star,2}^{v^\star}$ and $W^\star = \mathscr{H}_2(g^{v^\star}) \oplus (h^\star \| \varpi^\star)$. It next picks $\delta \xleftarrow{\$} \{0,1\}$, $\omega^\star \xleftarrow{\$} \{0,1\}^{l_1}$ and computes $r^\star = \mathscr{H}_1(m_\delta, \omega^\star)$, $E'^\star = g^{r^\star h^\star}$, $F^\star = \mathscr{H}_2(g^{r^\star}) \oplus (m_\delta \| \omega^\star)$. Then $\mathscr{B}$ picks $z^\star \xleftarrow{\$} \{0,1\}^{l_0}$, $\varpi_2 \xleftarrow{\$} \{0,1\}^{l_1}$, $Y^\star \xleftarrow{\$} \{0,1\}^{l_0+l_1}$, $t^\star \xleftarrow{\$} \mathbb{Z}_q^\star$ and computes $F' = \mathscr{H}_2(g^{z^\star}) \oplus F^\star$, $X^\star = (g^b)^{t^\star}$, implicitly define $x^\star = \mathscr{H}_1(z^\star, \varpi_2^\star) = \frac{b}{a}\frac{t^\star}{x_{i^\star,2}}$ and $\mathscr{H}_2((g^{\frac{b}{a}})^{\frac{t^\star}{x_{i^\star,2}}}) = Y^\star \oplus (z\star \| \varpi_2^\star)$. It finally gives $(E'^\star, F'^\star, V^\star, W^\star, X^\star, Y^\star)$ to $\mathscr{A}$.

Challenge for the IND-CCA security of PRE.ReEncrypt. $\mathscr{A}$ outputs an uncorrupted key $pk_{i^*} = (pk_{i^*,1}, pk_{i^*,2})$, a (corrupted or not) key $pk_{i'} = (pk_{i',1}, pk_{i',2})$ and two "good messages" C_0, C_1 which can be re-encrypted from $pk_{i'}$ to pk_{i^*}. If $c_i^* = 1$, then $\mathscr{B}$ aborts and outputs "failure". Else, $\mathscr{B}$ recovers $(pk_{i'}, pk_{i^\star}, R_{i'\to i^\star}, V^\star, W^\star, h)$ from $\mathscr{R}^{\mathsf{list}}$ or runs $\mathsf{OReEnc}(i', i^\star)$. It next picks $\delta \xleftarrow{\$} \{0,1\}$, parses C_δ as $(E, F, (T_k, \mathtt{ch}_k, \mathtt{resp}_k)_{k=1..l}))$ and computes $E'^\star = E^{R_{i'\to i^*}}$. Then it picks $z^\star \xleftarrow{\$} \{0,1\}^{l_0}$, $\varpi_2^\star \xleftarrow{\$} \{0,1\}^{l_1}$, $Y^\star \xleftarrow{\$} \{0,1\}^{l_0+l_1}$, $t^\star \xleftarrow{\$} \mathbb{Z}_q^\star$ and computes $F' = \mathscr{H}_2(g^{z^\star}) \oplus F$, $X^\star = (g^b)^{t^\star}$, implicitly define $x^\star = \mathscr{H}_1(z^\star \| \varpi_2^\star) = \frac{b}{a}\frac{t^\star}{x_{i^\star,2}}$ and $\mathscr{H}_2((g^{\frac{b}{a}})^{\frac{t^\star}{x_{i^\star,2}}}) = Y^\star \oplus (z^\star \| \varpi_2^\star)$. Finally, $\mathscr{B}$ gives $(E'^\star, F'^\star, V^\star, W^\star, X^\star, Y^\star)$ to $\mathscr{A}$.

Phase 2. $\mathscr{B}$ answers $\mathscr{A}$'s queries as in phase 1, with restrictions of the IND-CCA considered security.

Guess. $\mathscr{A}$ returns δ' and $\mathscr{B}$ picks a tuple $(R, \beta) \in \mathscr{H}_2^{\mathsf{list}}$ and return $R^{\frac{x_{i^\star,2}}{t^\star}}$ as solution to the DCDH instance.

ANALYSIS OF THE SIMULATION.

1) The simulation of oracles $\mathscr{H}_3$ and $\mathscr{H}_4$ are perfect, as well as these of $\mathscr{H}_1$, $\mathscr{H}_2$, except when $\frac{b}{a}\frac{t^\star}{x_{i^\star,2}}$ is asked to $\mathscr{H}_1$ and $(g^{\frac{b}{a}})^{\frac{t^\star}{x_{i^\star,2}}}$ is asked to $\mathscr{H}_2$.
2) The simulation of $\mathsf{OPubKey}$ and of OReKG are perfect.
3) If $\mathscr{B}$ does not abort, the simulation of the challenge and of $\mathsf{OSecKey}$ is perfect too.
4) The simulation of ODec_1: Applying the idea $(*)$, we can conclude that except for some special cases with probability $\leq \frac{1}{q}$, the simulation will return the same result as $\mathsf{Decrypt}_1$.

We consider different events:

- Let $\mathscr{H}_1^\star$ be the event that $\mathscr{A}$ queries $(z^\star, \varpi_2^\star)$ to $\mathscr{H}_1$ (since $\mathscr{B}$ does not know $\frac{b}{a}\frac{t^\star}{x_{i^\star,2}}$).
- Let $\mathscr{H}_2^\star$ be the event that $\mathscr{A}$ queries $(g^{\frac{b}{a}})^{\frac{t^\star}{x_{i^\star,2}}}$ to $\mathscr{H}_2$. It is the value we are looking for.

- Let Abort be the event that $\mathscr{B}$ aborts and outputs "failure" in the game.
- Let $\mathsf{D_1Err}$ be the event that the simulation of Decrypt1 gives something different from what it should.
- Let $\mathsf{Err} = (\mathscr{H}_1^\star \vee \mathscr{H}_2^\star \vee \mathsf{D_1Err}) | \neg \mathsf{Abort}$

Due to the randomness of the output of $\mathscr{H}_2$, if Err does not occur then the advantage of $\mathscr{A}$ is greater than $\frac{1}{2}$. So $\Pr(\delta = \delta' | \neg \mathsf{Err}) = \frac{1}{2}$. As explain in [7], we have $\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PRE.Algo},\mathscr{A}}(\kappa) \leq \Pr(\mathsf{Err})$, where $\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PRE.Algo},\mathscr{A}}(\kappa)$ denotes the advantage of $\mathscr{A}$ to win its challenge for $\mathsf{Algo} \in \{\mathsf{ReEncrypt}, \mathsf{Encrypt}_1\}$. We also obtain $\Pr(\mathscr{H}_2^\star) \geq \Pr(\neg Abort).\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PRE.Algo},\mathscr{A}}(\kappa) - \Pr(\mathscr{H}_1^\star) - \Pr(\mathsf{D_1Err})$. And as $\mathscr{B}$ picks a tuple $(R, \beta) \in \mathscr{H}_2^{\mathsf{list}}$ and returns $R^{\frac{x_{i^\star,2}}{i^\star}}$ as solution to the challenge, if the event $\mathscr{H}_2^\star$ occurs, there is a probability of $\frac{1}{q_{\mathscr{H}_2}}$ to $\mathscr{B}$ to return the good one. The success probability of $\mathscr{B}$ is finally bounded by: $\frac{\mathscr{H}_2^\star}{q_{\mathscr{H}_2}} \geq \frac{\Pr(\neg \mathsf{Abort}).\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PRE.Algo},\mathscr{A}}(\kappa)}{q_{\mathscr{H}_2}} - \frac{\Pr(\mathscr{H}_1^\star)}{q_{\mathscr{H}_2}} - \frac{\Pr(\mathsf{D_1Err})}{q_{\mathscr{H}_2}}$. We now detail each remaining probabilities.

- $\Pr(\mathscr{H}_1^\star)$: at most, $q_{\mathscr{H}_1}$ chance over 2^{l_1} to find $\varpi_2^\star$, $\Pr(\mathscr{H}_1^\star) \leq \frac{q_{\mathscr{H}_1}}{2^{l_1}}$.
- $\Pr(\mathsf{D_1Err})$: we already have seen $\Pr(\mathsf{D_1Err}) \leq \frac{q_{D_1}}{q}$.
- $\Pr(\neg \mathsf{Abort})$: $\mathscr{B}$ aborts if (i) OSecKey is asked for pk_i with $pk_i = 0$ or (ii) $c_i^* = 1$ in the challenge. As we saw it before, we have $\Pr(\neg Abort) \geq (1-\theta)\theta^{n_c}$ which is maximized at $\theta_{\text{opt}} = \frac{n_c}{1+n_c}$ by $\left(\frac{1}{1+n_c}\right)\left(\frac{n_c}{1+n_c}\right)^{n_c} \geq \frac{1}{e(1+n_c)}$, which proves the lower bound announced in the theorem.

□

Acknowledgements

We are grateful to Damien Vergnaud, from the École Normale Supérieure (Paris, France), for his suggestions of improvement, and to anonymous referees for their valuable comments.

References

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

[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. 2006*, 9(1):1–30, 2006.

[3] Feng Bao, Robert H. Deng, and Huafei Zhu. Variations of diffie-hellman problem. In *ICICS*, pages 301–312, 2003.

[4] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144. Springer, 1998.

[5] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *ACM Conference on Computer and Communications Security 2007*, pages 185–194. ACM, 2007.

[6] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2003.

[7] Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient unidirectional proxy re-encryption. In *AFRICACRYPT 2010*, volume 6055 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2010.

[8] Jean-Sébastien Coron. On the exact security of full domain hash. In *CRYPTO'00*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, 2000.

[9] Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen. Chosen-ciphertext secure proxy re-encryption without pairings. In *CANS 2008*, volume 5339 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.

[10] U. Feige, A. Fiat, and A. Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988.

[11] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer, 1986.

[12] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, pages 152–168, 2005.

[13] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO'99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[14] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *Public Key Cryptography 2008*, volume 4939 of *Lecture Notes in Computer Science*, pages 360–379. Springer, 2008.

[15] Toshihide Matsuda, Ryo Nishimaki, and Keisuke Tanaka. Cca proxy re-encryption without bilinear maps in the standard model. In *Public Key Cryptography 2010*, volume 6056 of *Lecture Notes in Computer Science*, pages 261–278. Springer, 2010.

[16] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 1989.

[17] Jun Shao and Zhenfu Cao. Cca-secure proxy re-encryption without pairings. In *Public Key Cryptography 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 357–376. Springer, 2009.