On the Security of a Bidirectional Proxy Re-Encryption Scheme from PKC 2010

Jian Weng[1], Yunlei Zhao[2]

[1] Department of Computer Science, Jinan University, Guangzhou 510632, P.R. China
[2]Software School, Fudan University, Shanghai 201203, P. R. China

Abstract

In PKC 2010, Matsuda, Nishimaki and Tanaka proposed a bidirectional proxy re-encryption (PRE) scheme without bilinear maps, and claimed that their scheme is chosen-ciphertext secure in the standard model. However, by giving a concrete attack, in this paper we indicate that their PRE scheme fails to achieve the chosen-ciphertext security. The purpose of this paper is to clarify the fact that, it is still an open problem to come up with a chosen-ciphertext secure PRE scheme without bilinear maps in the standard model.

Keywords: bilinear map, chosen-ciphertext security, standard model.

1 Introduction

Proxy re-encryption (PRE), introduced by Blaze, Bleumer and Strauss [3] in Eurocrypt'98, allows a semi-trust proxy to translate a ciphertext intended for Alice into another ciphertext intended for Bob. The proxy, however, can not learn anything about the underlying messages. According to the direction of transformation, PRE can be categorized into *bidirectional* PRE, in which the proxy can transform from Alice to Bob and vice versa, and *unidirectional* PRE, in which the proxy cannot transform ciphertexts in the opposite direction. PRE can also be categorized into *multi-hop* PRE, in which the ciphertexts can be transformed from Alice to Bob and then to Charlie and so on, and *single-hop* PRE, in which the ciphertexts can only be transformed once.

In their seminal paper, Blaze *et al.* [3] proposed the first bidirectional PRE scheme. Ateniese *et al.* [1, 2] presented unidirectional PRE schemes from bilinear maps. All of these schemes are only secure against chosen-plaintext attack (CPA). However, applications often require security against chosen-ciphertext attacks (CCA).

To fill this gap, Canetti and Hohenberger [6] presented the first CCA-secure bidirectional multi-hop PRE scheme in the standard model. Libert and Vergnaud [10, 11] proposed a unidirectional single-hop PRE scheme, which is replayable CCA-secure [7] in the standard model. Recently, Weng et al. [14] presented a unidirectional single-hop PRE scheme, which is CCA-secure against adaptive corruption of users in the standard model. The above schemes rely on bilinear maps. In spite of the recent advances in implementation technique, compared with standard operations such as modular exponentiation in finite fields, the bilinear map computation is still considered as a rather expensive operation. It would be desirable for cryptosystems to be constructed without relying on pairings, especially in computation resource limited settings. Thus, how to construct a CCA-secure PRE scheme without bilinear maps is left as an open problem in [6].

Deng et al. [9] presented a bidirectional single-hop PRE scheme without bilinear maps, and proved its CCA-security in the random oracle model. Shao and Cao presented a unidirectional single-hop PRE scheme without bilinear maps, and claimed that their scheme is CCA-secure in the random oracle model. However, Sherman, Weng and Yang et al. [8] present a concrete attack, and indicated that Shao and Cao's PRE scheme is not CCA-secure. They further presented a CCA-secure unidirectional single-hop PRE scheme without bilinear maps, again in the random oracle model. It is well-known [4, 5] that a proof in the random oracle model can only serve as an argument, which does not imply the security for real implementations. Thus, it is more desirable to come up with a CCA-secure PRE scheme without bilinear maps in the standard model.

In PKC 2010, Matsuda, Nishimaki and Tanaka proposed a bidirectional multi-hop proxy re-encryption (PRE) scheme without bilinear maps, and claimed that their scheme is CCA-secure in the standard model. However, in this paper, we present a concrete attack, and indicate that their PRE scheme fails to achieve the CCA-security. Thus it is still an open problem to come up with a CCA-secure PRE scheme without bilinear maps in the standard model.

2 Preliminaries

Matsuda-Nishimaki-Tanaka PRE scheme involves the primitives of all-but-one trapdoor function and re-applicable (n, k) lossy trapdoor functions (LTDFs). Thus in this section, we shall review the definitions of these two primitives (for more details, the reader is referred to [12] and [13]). We shall also review the definition and security notion for bidirectional multi-hop PRE.

2.1 All-But-One Trapdoor Function

Let $B = \{B_\lambda\}_{\lambda\in\mathbb{N}}$ be a collection of sets whose elements represents the branches. A collection of (n, k)-all-but-one trapdoor functions is a tuple of probabilistic polynomial time (PPT) algorithms $(\mathsf{G}_{\text{abo}}, \mathsf{F}_{\text{abo}}, \mathsf{F}^{-1}_{\text{abo}})$ having the following properties:

- **All-but-one property**: Given a lossy branch $b^* \in B_\lambda$, the algorithm $\mathsf{G}_{\text{abo}}(1^\lambda, b^*)$ outputs a pair (s, td), where s is a function index and td is its trapdoor. For any $b \in B_\lambda\backslash\{b^*\}$, the algorithm $\mathsf{F}_{\text{abo}}(s, b, \cdot)$ computes an injective function $f_{s,b}(\cdot)$ over $\{0, 1\}^n$, and $\mathsf{F}^{-1}_{\text{abo}}(td, b, \cdot)$ computes $f^{-1}_{s,b}(\cdot)$. For the lossy branch b^*, $\mathsf{F}_{\text{abo}}(s, b^*, \cdot)$ computes a lossy function $f_{s,b^*}(\cdot)$ over the domain $\{0, 1\}^n$, where $|f_{s,b^*}(\{0, 1\}^n)| \leq 2^{n-k}$.
- **Indistinguishability:** For every b^*_1 and $b^*_2 \in B_\lambda$, the first output s_0 of $\mathsf{G}_{\text{abo}}(1^\lambda, b^*_0)$ and the first output s_1 of $\mathsf{G}_{\text{abo}}(1^\lambda, b^*_1)$ are computationally indistinguishable.

2.2 Re-Applicable (n, k)-Lossy Trapdoor Functions

A collection of re-applicable (n, k)-lossy trapdoor functions (LTDFs) with respect to function indices is a tuple of PPT algorithms (ParGen, $\mathsf{LossyGen}$, $\mathsf{LossyEval}$, $\mathsf{LossyInv}$, $\mathsf{ReIndex}$, ReEval, $\mathsf{PrivReEval}$, Trans, $\mathsf{FakeKey}$) such that:

Injectivity: For every public parameter $\mathsf{par} \leftarrow \mathsf{ParGen}(1^\lambda)$ and every tag $\tau \in \mathcal{T}\backslash\{\tau_{\text{los}}\}$, $\mathsf{LossyGen}(\tau)$ outputs a pair of a function index and its trapdoor (s, td), $\mathsf{LossyEval}(s, \cdot)$ computes an injective function $f_{s,\tau}(\cdot)$ over $\{0, 1\}^n$, and $\mathsf{LossyInv}(td, \tau, \cdot)$ computes $f^{-1}_{s,\tau}(\cdot)$. (We represent the function $f_{s,\tau}$, not f_s, in order to clarify a tag τ. If we do not need to clarify a tag, we represent a function as $f_{s,\star}$.)

Lossiness: For every public parameter $\mathsf{par} \leftarrow \mathsf{ParGen}(1^\lambda)$, the algorithm $\mathsf{LossyGen}(\tau_{\mathrm{los}})$ outputs $(s, \perp)$ and $\mathsf{LossyEval}(s, \cdot)$ computes a function $f_{s,\tau_{\mathrm{los}}}(\cdot)$ over $\{0,1\}^n$, where $|f_{s,\tau_{\mathrm{los}}}(\{0,1\}^n)| \le 2^{n-k}$.

Indistinguishability between injective and lossy indices: Let X_λ denote the distribution of $(\mathsf{par}, s_{\mathrm{inj}}, \tau)$, and let Y_λ denote the distribution of $(par, s_{\mathrm{los}}, \tau')$, where par is a public parameter from $\mathsf{ParGen}(1^\lambda)$, τ and τ' are random elements in $\mathcal{T}$, and the function indices s_{inj} and s_{los} are the first element output from $\mathsf{LossyGen}(\tau)$ and $\mathsf{LossyGen}(\tau_{\mathrm{los}})$ respectively. Then, $\{X_\lambda\}$ and $\{Y_\lambda\}$ are computationally indistinguishable.

Re-applying with respect to function indices: Let τ_i and τ_j be any tags with $\tau_i \ne \tau_{\mathrm{los}}$ and $\tau_j \ne \tau_{\mathrm{los}}$. The algorithm $\mathsf{ReIndex}(td_i, td_j)$ outputs $s_{i\leftrightarrow j}$, where td_i and td_j are the second elements of $\mathsf{LossyGen}(\tau_i)$ and $\mathsf{LossyGen}(\tau_j)$. Then, for every $x \in \{0,1\}^n$, $x = \mathsf{LossyInv}(td_j, \tau_i, \mathsf{ReEval}(s_{i\leftrightarrow j}, \mathsf{LossyEval}(s_i, x)))$. Note that $\mathsf{LossyInv}$ takes τ_i as one of the inputs, not τ_j.

Generating proper outputs: Let c be an output from $\mathsf{ReEval}(s_{i\leftrightarrow j}, \mathsf{LossyEval}(s_i, x))$, where $s_{i\leftrightarrow j}$ and s_i have the same meaning as that in the above paragraph. Then, $\mathsf{PrivReEval}(x, \tau_i, \tau_j, s_j)$ outputs the same c, where x, τ_i, τ_j, and s_j have the same meaning as that in the above paragraph. That is, $\mathsf{ReEval}(s_{i\leftrightarrow j}, \mathsf{LossyEval}(s_i, \cdot))$ and $\mathsf{PrivReEval}(\cdot, \tau_i, \tau_j, s_j)$ are equivalent as a function (i.e. any output of $\mathsf{ReEval}(s_{i\leftrightarrow j}, \mathsf{LossyEval}(s_i, \cdot))$ is independent of s_i).

Transitivity: Let (s_i, td_i), (s_j, td_j) and (s_k, td_k) be outputs from $\mathsf{LossyGen}(\tau_i)$, $\mathsf{LossyGen}(\tau_j)$, and $\mathsf{LossyGen}(\tau_k)$, and let $s_{i\leftrightarrow j}$ and $s_{i\leftrightarrow k}$ be the outputs from $\mathsf{ReIndex}(td_i, td_j)$ and $\mathsf{ReIndex}(td_i, td_k)$, respectively. Then, $\mathsf{Trans}(s_{i\leftrightarrow j}, s_{i\leftrightarrow k})$ outputs $s_{j\leftrightarrow k}$ which is the same output from $\mathsf{ReIndex}(td_j, td_k)$.

Statistical indistinguishability of the fake key: The algorithm $\mathsf{FakeKey}(s_i, \tau_i)$ outputs $(s'_j, s'_{i\leftrightarrow j}, \tau'_j)$, where s_i is the first element of an output from $\mathsf{LossyGen}(\tau_i)$. Let X_λ denote the distribution of $(\mathsf{par}, s_i, s_j, s_{i\leftrightarrow j}, \tau_i, \tau_j)$, and let Y_λ denote the distribution of $(\mathsf{par}, s_i, s'_j, s'_{i\leftrightarrow j}, \tau_i, \tau'_j)$, where each $\mathsf{par}, s_j, s_{i\leftrightarrow j}$, and τ_j has the same meaning as that in the above paragraph. Then, $\{X_\lambda\}$ and $\{Y_\lambda\}$ are statistically indistinguishable.

Generation of injective functions from lossy functions: Let s be the first element of an output from $\mathsf{FakeKey}(s_{\mathrm{los}}, \tau)$, where τ is a tag and s_{los} is the first element of an output from $\mathsf{LossyGen}(\tau_{\mathrm{los}})$. Then, for every τ, $\mathsf{LossyEval}(s, \cdot)$ represents an injective function $f_{s,\star}$ with overwhelming probability, where a random variable is the randomness of $\mathsf{FakeKey}(s_{\mathrm{los}}, \tau)$. (We do not require other properties of index s if $f_{s,\star}$ is injective. The function $f_{s,\star}$ cannot have any trapdoor information.)

2.3 Realization of Re-applicable LTDFs

Based on Peikert and Waters' LTDFs [13], Matsuda, Nishimaki and Tanaka [12] gave a realization of re-applicable LTDFs, which is specified as below (for more details, the reader is referred to [12]):

ParGen: This algorithm first generates a cyclic group $\mathbb{G}$ with prime order p, and then chooses a random generator $g \in \mathbb{G}$. Next, it selects random numbers $r_1, \cdots, r_n \in_R \mathbb{Z}_p$, and outputs the public parameters $\mathbf{C}_1$ as

$$\mathbf{C}_1 = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} g^{r_1} \\ \vdots \\ g^{r_n} \end{pmatrix}.$$

$\mathsf{LossyGen}$: Taking as input the public parameter $\mathbf{C}_1$ and a tag $\tau \in \mathbb{G}$ (note that if τ is the identity element e of $\mathbb{G}$, it means execution of the lossy mode; otherwise, execution of the injective mode), this algorithm first selects random elements $z_1, z_2, \cdots, z_n \in_R \mathbb{Z}_p$, and then computes a function index as

$$\mathbf{C}_2 = \begin{pmatrix} c_{1,1} & \cdots & c_{1,n} \\ \vdots & \ddots & \vdots \\ c_{n,1} & \cdots & c_{n,n} \end{pmatrix} = \begin{pmatrix} c_1^{z_1} \cdot \tau & \cdots & c_1^{z_n} \\ \vdots & \ddots & \vdots \\ c_n^{z_1} & \cdots & c_n^{z_n} \cdot \tau \end{pmatrix} = \begin{cases} c_{i,j} = c_i^{z_j} \cdot \tau, & \text{if } i = j; \\ c_{i,j} = c_i^{z_j}, & \text{otherwise.} \end{cases}$$

Finally, it outputs the function index $s = (\mathbf{C}_1, \mathbf{C}_2)$ and the trapdoor $td = z = (z_1, \cdots, z_n)$.

LossyEval: Taking as input a function index $s = (C_1, C_2)$ and an n-bit input $\mathbf{x} = (x_1, \cdots, x_n) \in \{0,1\}^n$, this algorithm outputs $(y_1, \mathbf{y}_2)$ such that

$$y_1 = \mathbf{x}\mathbf{C}_1 = \prod_{i=1}^{n} c_i^{x_i},$$

$$\mathbf{y}_2 = \mathbf{x}\mathbf{C}_2 = \left(\prod_{i=1}^{n} c_{i,1}^{x_i}, \cdots, \prod_{i=1}^{n} c_{i,n}^{x_i} \right) = \left((\prod_{i=1}^{n} c_i^{z_1 x_i})\tau^{x_1}, \cdots, (\prod_{i=1}^{n} c_i^{z_n x_i})\tau^{x_n} \right).$$

LossyInv: Taking as input $(td, \tau, (y_1, \mathbf{y}_2))$, where the trapdoor information td consists of $z = (z_1, \cdots, z_n)$, the tag τ is an element in $\mathbb{G}\backslash\{e\}$, and $\mathbf{y}_2 = (y_{2,1}, \cdots, y_{2,n}) \in \mathbb{G}^{1\times n}$, this algorithm computes $\mathbf{w} = (y_{2,1} \cdot y_1^{-z_1}, y_{2,2} \cdot y_1^{-z_2}, \cdots, y_{2,n} \cdot y_1^{-z_n})$. Then, if j-th element of $\mathbf{w}$ is the identity element of $\mathbb{G}$, then it sets $x_j = 0$; else if the j-th element of $\mathbf{w}$ is τ then it sets $x_j = 1$; otherwise, it outputs $\perp$. Finally, it outputs $x = (x_1, \cdots, x_n)$.

ReIndex: Taking as input trapdoors $td_i = (z_1, \cdots, z_n)$ and $td_j = (z'_1, \cdots, z'_n)$, this algorithm outputs $s_{i\leftrightarrow j} = td_j - td_i = (z'_1 - z_1, \cdots, z'_n - z_n) = (z_{1,i\leftrightarrow j}, \cdots, z_{n,i\leftrightarrow j})$.

ReEval: On input $(s_{i\leftrightarrow j}, (y_1, \mathbf{y}_2))$, where $s_{i\leftrightarrow j} = (z_{1,i\leftrightarrow j}, z_{2,i\leftrightarrow j}, \cdots, z_{n,i\leftrightarrow j})$ and $(y_1, \mathbf{y}_2) = (y_1, (y_{2,1}, y_{2,2}, \cdots, y_{2,n}))$, this algorithm computes $\mathbf{y}'_2 = (y'_{2,1}, y'_{2,2}, \cdots, y'_{2,n}) = (y_{2,1} \cdot y_1^{z_{1,i\leftrightarrow j}}, y_{2,2} \cdot y_1^{z_{2,i\leftrightarrow j}}, \cdots, y_{2,n} \cdot y_1^{z_{n,i\leftrightarrow j}})$. Then it outputs $(y_1, \mathbf{y}'_2)$.

PrivReEval: Taking as input $\mathbf{x}, \tau_i, \tau_j$ and s_j, where $\mathbf{x} = (x_1, \cdots, x_n)$ is n-bit input, this algorithm first computes $(\hat{y}_1, \hat{\mathbf{y}}_2) \leftarrow \mathsf{LossyEval}(s_j, \mathbf{x})$. Next, it makes $\hat{\mathbf{y}}'_2$ from $\hat{\mathbf{y}}_2$ in the following process: for each $i \in [1, n]$, if $x_i = 1$ then $\hat{y}'_{2,i} = \hat{y}_{2,i}\tau_j^{-1}\tau_i$; else $\hat{y}'_{2,i} = \hat{y}_{2,i}$, where $\hat{y}_{2,i}$ and $\hat{y}'_{2,i}$ are the i-th elements of $\hat{\mathbf{y}}_2$ and $\hat{\mathbf{y}}'_2$ respectively. Finally, it outputs $(\hat{y}_1, \hat{\mathbf{y}}'_2)$.

Trans: Taking as input $s_{i\leftrightarrow j}$ and $s_{i\leftrightarrow k}$, this algorithm outputs $s_{i\leftrightarrow k} - s_{i\leftrightarrow j} = (td_k - td_i) - (td_j - td_i) = td_k - td_j = s_{i\leftrightarrow k}$.

FakeKey: Taking as input a function index $s_i = (\mathbf{C}_1, \mathbf{C}_2)$ and a tag $\tau_i \in \mathbb{G}$, this algorithm first chooses a random element $t \in \mathbb{G}$. Next, it chooses random numbers $s_{i\leftrightarrow j} = (z_{1,i\leftrightarrow j}, \cdots, z_{n,i\leftrightarrow j}) \in_R \mathbb{Z}_p^n$. Then it makes a new matrix $\mathbf{C}'_2$ as follows:

$$\mathbf{C}'_2 = \begin{pmatrix} c_{1,1} \cdot c_1^{z_{1,i\leftrightarrow j}} \cdot t & \cdots & c_{1,n} \cdot c_1^{z_{n,i\leftrightarrow j}} \\ \vdots & \ddots & \vdots \\ c_{n,1} \cdot c_n^{z_{1,i\leftrightarrow j}} & \cdots & c_{n,n} \cdot c_n^{z_{n,i\leftrightarrow j}} \cdot t \end{pmatrix} = \begin{cases} c'_{k,\ell} = c_{k,\ell} \cdot c_k^{z_{\ell,i\leftrightarrow j}} \cdot t, & \text{if } k = \ell; \\ c'_{k,\ell} = c_{k,\ell} \cdot c_k^{z_{\ell,i\leftrightarrow j}}, & \text{otherwise,} \end{cases}$$

where c_k is the k entry of $\mathbf{C}_1$, and $c_{k,\ell}$ is the (k,ℓ) entry of $\mathbf{C}_2$. Finally, it outputs $s_j = (\mathbf{C}_1, \mathbf{C}'_2), s_{i\leftrightarrow j} = (z_{1,i\leftrightarrow j}, \cdots, z_{n,i\leftrightarrow j})$ and $\tau_j = \tau_i \cdot t$.

3 Bidirectional Multi-Hop PRE

A bidirectional PRE scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{ReKeyGen}, \mathsf{ReEnc}, \mathsf{Dec})$ consists of the following six algorithms:

- $\mathsf{Setup}(1^\lambda)$: Given a security parameter 1^λ, this setup algorithm outputs a public parameter PP. Denote this by $PP \leftarrow \mathsf{Setup}(1^\lambda)$.
- $\mathsf{KeyGen}(PP)$: Given the public parameter PP, this key generation algorithm outputs a public key pk and a secret key sk. Denote this by $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$.
- $\mathsf{Enc}(PP, pk, m)$: Given the public parameter PP, a public key pk and a message m in the message space $\mathcal{M}$, this encryption algorithm outputs a ciphertext C. Denote this by $C \leftarrow \mathsf{Enc}(PP, pk, m)$.
- $\mathsf{ReKeyGen}(PP, sk_i, sk_j)$: Given the public parameter PP, a pair of secret keys sk_i and sk_j where $i \neq j$, this re-encryption key generation algorithm outputs a re-encryption key $rk_{i\leftrightarrow j}$. Denote this by $rk_{i\leftrightarrow j} \leftarrow \mathsf{ReKeyGen}(PP, sk_i, sk_j)$.
- $\mathsf{ReEnc}(PP, rk_{i\leftrightarrow j}, C_i)$: Given the public parameter PP, a re-encryption key $rk_{i\leftrightarrow j}$ and a ciphertext C_i intended for user i, this re-encryption algorithm outputs another ciphertext C_j for user j or the error symbol $\perp$. Denote this by $C_j \leftarrow \mathsf{ReEnc}(PP, rk_{i\leftrightarrow j}, C_i)$.
- $\mathsf{Dec}(PP, sk, C)$: Given the public parameter PP, a public key sk and a ciphertext C, this decryption algorithm outputs a message m or the error symbol $\perp$.

Next, we review the definition of chosen-ciphertext security for bidirectional multi-hop PRE scheme as defined in [6, 12]. Let λ be the security parameter, $\mathcal{A}$ be an oracle TM, representing the adversary, and Γ_U and Γ_C be two lists which are initially empty. The game consists of an execution of $\mathcal{A}$ with the following oracles, which can be invoked multiple times in any order, subject to the constraints specified as below:

Setup Oracle: This oracle can be queried first in the game only once. This oracle generates the public parameters $PP \leftarrow \mathsf{Setup}(1^\lambda)$, and gives PP to $\mathcal{A}$.

Uncorrupted key generation: This oracle first generates a new key pair by running $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$. Next, it adds pk in Γ_U, and gives pk to $\mathcal{A}$.

Corrupted key generation: This oracle generates a new key pair by running $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$. Next, it adds pk in Γ_C, and gives (pk, sk) to $\mathcal{A}$.

Challenge oracle: This oracle can be queried only once. On input (pk_{i^*}, m_0, m_1), this oracle randomly chooses a bit $b \in \{0, 1\}$ and gives $C_{i^*} = \mathsf{Enc}(PP, pk_{i^*}, m_b)$ to $\mathcal{A}$. Here it is required that $pk_{i^*} \in \Gamma_U$. We call pk_{i^*} the challenge key and C_{i^*} the challenge ciphertext.

Re-encryption key generation: On input (pk_i, pk_j) from the adversary, this oracle gives the re-encryption key $rk_{i\leftrightarrow j} = \mathsf{ReKeyGen}(PP, sk_i, sk_j)$ to $\mathcal{A}$, where sk_i and sk_j are the secret keys corresponding to pk_i and pk_j, respectively. Here it is required that pk_i and pk_j are both in Γ_C, or alternatively are both in Γ_U.

Re-encryption oracle: On input (pk_i, pk_j, C_i), if $pk_j \in \Gamma_C$ and (pk_i, C_i) is a *derivative* of (pk_{i^*}, C_{i^*}), this oracle give $\mathcal{A}$ a special symbol $\perp$, which is not in the domain of messages or ciphertext. Otherwise, it gives the re-encrypted ciphertext $C_j = \mathsf{ReEnc}(PP, \mathsf{ReKeyGen}(PP, sk_i, sk_j), C_i)$ to $\mathcal{A}$. Derivatives of (pk_{i^*}, C_{i^*}) are defined inductively as follows:

- (pk_{i^*}, C_{i^*}) is a derivative of itself.
- If (pk, C) is a derivative of (pk_{i^*}, C_{i^*}), and (pk', C') is a derivative of (pk, C), then (pk', C') is a derivative of (pk_{i^*}, C_{i^*}).
- If $\mathcal{A}$ has queried the re-encryption oracle on input (pk, pk', C) and obtained the response C', then (pk', C') is a derivative of (pk, C).
- If $\mathcal{A}$ has queried the re-encryption key generation oracle on input (pk, pk') or (pk', pk), and $C' = \mathsf{ReEnc}(PP, \mathsf{ReKeyGen}(PP, sk, sk'), C)$, then (pk', C') is a deriva-

tive of (pk, C), where sk and sk' are the secret keys corresponding to pk and pk', respectively.

Decryption oracle: On input (pk, C), if the pair (pk, C) is a derivative of the challenge key and ciphertext (pk_{i^*}, C_{i^*}), or pk is not in $\Gamma_U \cup \Gamma_C$, this oracle returns the special symbol $\perp$ to $\mathcal{A}$. Otherwise, it returns the result of $\mathsf{Dec}(PP, sk, C)$ to $\mathcal{A}$, where sk is the secret key with respect to pk.

Decision oracle: This oracle can be queried at the end of the game. On input b', if $b' = b$ and the challenge key $pk_{i^*} \in \Gamma_U$, this algorithm output 1; else output 0.

We describe the output of the decision oracle in the above game as $\mathsf{Expt}^{\text{bid-PRE-CCA}}_{\Pi,\mathcal{A}}(\lambda) = b$ for an adversary A and a scheme Π. We define the advantage of adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\text{bid-PRE-CCA}}_{\Pi,\mathcal{A}}(\lambda) \stackrel{\text{def}}{=} \left| \Pr[\mathsf{Expt}^{\text{bid-PRE-CCA}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right|,$$

where the probability is over the random choices of $\mathcal{A}$ and oracles. We say that the scheme Π is secure under the bidirectional PRE-CCA attack, if for any PPT adversary A, his advantage $\mathsf{Adv}^{\text{bid-PRE-CCA}}_{\Pi,\mathcal{A}}(\lambda)$ is negligible in the security parameter λ (for sufficiently large λ).

4 Review of Matsuda-Nishimaki-Tanaka PRE Scheme

In this section, we shall review Matsuda-Nishimaki-Tanaka bidirectional multi-hop PRE Scheme.

Let λ be the security parameter, and let n, k, k', k'' and v be parameters depended on λ. Let $(\mathsf{SigGen}, \mathsf{SigSign}, \mathsf{SigVer})$ be a strongly unforgeable one-time signature scheme where the verification keys are in $\{0,1\}^v$. Let $(\mathsf{ParGen}, \mathsf{LossyGen}, \mathsf{LossyEval}, \mathsf{LossyInv}, \mathsf{ReEval}, \mathsf{PrivReEval}, \mathsf{Trans}, \mathsf{FakeKey})$ be a collection of re-applicable (n, k)-LTDFs and $\mathcal{T}$ be a set of tags. Let $(\mathsf{G}_{\text{abo}}, \mathsf{F}_{\text{abo}}, \mathsf{F}^{-1}_{\text{abo}})$ be a collection of (n, k')-ABO trapdoor functions with branches $B_\lambda = \{0,1\}^v$, which contains the set of signature verification keys. Let $\mathcal{H}$ be a family of pairwise independent hash functions from $\{0,1\}^n$ to $\{0,1\}^{k''}$. It is required that the above parameters satisfy $(k + k') - (k'' + n) \geq \delta = \delta_1 + \delta_2$ for some $\delta_1 = \omega(\log \lambda)$ and $\delta_2 = \omega(\log \lambda)$. The message space of the system is $\{0,1\}^{k''}$. The Matsuda-Nishimaki-Tanaka PRE scheme [12] is specified by the following algorithms:

$\mathsf{Setup}(1^\lambda)$: This algrithm first generates an index of all-but-one trapdoor functions with lossy branch 0^v: $(s_{\text{abo}}, td_{\text{abo}}) \leftarrow \mathsf{G}_{\text{abo}}(1^\lambda, 0^v)$. Then, it generates a public parameter of re-applicable LTDFs: $\mathsf{par} \leftarrow \mathsf{ParGen}(1^\lambda)$. Next, it chooses a hash function h from $\mathcal{H}$. Finally, it outputs a public parameter as $PP = (s_{\text{abo}}, \mathsf{par}, h)$.
Note that the algorithm Setup erases the trapdoor td_{abo} because the following algorithms do not use td_{abo}.

$\mathsf{KeyGen}(PP)$: Taking as input the pubic parameters $PP = (s_{\text{abo}}, \mathsf{par}, h)$, this algorithm first chooses a tag $\tau \in \mathcal{T} \backslash \{\tau_{\text{los}}\}$ and generates an injective index of re-applicable LTDFs: $(s_{\text{rltdf}}, td_{\text{rltdf}}) \leftarrow \mathsf{LossyGen}(\tau)$. Finally, it outputs the public key $pk = (s_{\text{rltdf}}, \tau)$ and the secret key $sk = (td_{\text{rltdf}}, s_{\text{rltdf}}, \tau)$.

$\mathsf{Enc}(PP, pk, m)$: Taking as input the public parameters $PP = (s_{\text{abo}}, \mathsf{par}, h)$, public key $pk = (s_{\text{rltdf}}, \tau)$ and a message $m \in \{0,1\}^{k''}$, this encryption algorithm first chooses $x \in \{0,1\}^n$ uniformly at random. Next it generates a key-pair for the one-time signature scheme: $(vk, sk_\sigma) \leftarrow \mathsf{SigGen}(1^\lambda)$, and computes

$$c_1 = \mathsf{LossyEval}(s_{\text{rltdf}}, x), c_2 = \mathsf{F}_{\text{abo}}(s_{\text{abo}}, vk, x), c_3 = h(x) \oplus m.$$

Then it signs a tuple (c_2, c_3, τ) as $\sigma \leftarrow \mathsf{SigSign}(sk_\sigma, (c2, c3, \tau))$. Finally, it outputs the ciphertext $C = (vk, c_1, c_2, c_3, \tau, \sigma)$.

ReKeyGen(PP, sk_i, sk_j)**:** Taking as input the public parameter $PP = (s_{\text{abo}}, \mathsf{par}, h)$, the secret keys $sk_i = (td_i, s_i, \tau_i)$ and $sk_j = (td_j, s_j, \tau_j)$, this algorithm computes $s_{i\leftrightarrow j} \leftarrow \mathsf{ReIndex}(td_i, td_j)$, and then outputs a re-encryption key $rk_{i\leftrightarrow j} = s_{i\leftrightarrow j}$.

ReEnc$(PP, rk_{i\leftrightarrow j}, C_i)$**:** Taking as input the public parameter $PP = (s_{\text{abo}}, \mathsf{par}, h)$, the re-encryption key $rk_{i\leftrightarrow j} = s_{i\leftrightarrow j}$ and a ciphertext $C_i = (vk, c_{1,i}, c_2, c_3, \tau, \sigma)$, this algorithm computes $c_{1,j} \leftarrow \mathsf{ReEval}(s_{i\leftrightarrow j}, c_{1,i})$. It then outputs $C_j = (vk, c_{1,j}, c_2, c_3, \tau, \sigma)$ as a new ciphertex for the user with sk_j.

Dec(PP, sk, C)**:** Taking as input the public parameter $PP = (s_{\text{abo}}, \mathsf{par}, h)$, a secret key $sk = (td_{\text{rltdf}}, s_{\text{rltdf}}, \tau)$ and a ciphertext $C = (vk, c_1, c_2, c_3, \tau', \sigma)$, this decryption algorithm acts as follows:

1. Check whether $\mathsf{SigVer}(vk, (c_2, c_3, \tau'), \sigma) = 1$ holds. If not, output $\perp$.
2. Compute $x = \mathsf{LossyInv}(td_{\text{rltdf}}, \tau', c_1)$. If $\tau = \tau'$, it checks $\mathsf{LossyEval}(s_{\text{rltdf}}, x) = c_1$; else it checks $\mathsf{PrivReEval}(x, \tau', \tau, s_{\text{rltdf}}) = c_1$. If not, it outputs $\perp$. It also checks $\mathsf{F}_{\text{abo}}(s_{\text{abo}}, vk, x) = c_2$. If not, it outputs $\perp$.
3. Finally, output $m = c_3 \oplus h(x)$.

5 Attack

In this section, we shall present a concrete CCA-attack against Matsuda-Nishimaki-Tanaka PRE scheme. Before presenting the attack, we would like mention a fundamental principle for designing CCA-secure PRE schemes, i.e., the validity of *all* the ciphertext components in the original ciphertext should be able to be verified by the *proxy*. Unfortunately, Matsuda-Nishimaki-Tanaka PRE scheme violates this principle. Indeed, for a ciphertext $C_i = (vk, c_{1,i}, c_2, c_3, \tau, \sigma)$, the validity of vk, c_2, c_3, τ and σ can be ensured by checking whether $\mathsf{SigVer}(vk, (c_2, c_3, \tau'), \sigma) = 1$ holds[1]. However, it is impossible for the proxy to verify the validity of component $c_{1,i}$: observe that in the encryption algorithm, component $c_{1,i}$ is not included in the generation of the one-time signature, and it will be transformed into $c_{1,j}$ in the re-encryption algorithm. Thus, Matsuda-Nishimaki-Tanaka PRE scheme inevitably suffers from a chosen-ciphertext attack.

Roughly speaking, an adversary can break the CCA-security of Matsuda-Nishimaki-Tanaka PRE scheme as follows: Given the challenge ciphertext $C_{i^*} = (vk, c_{1,i^*}, c_2, c_3, \tau, \sigma)$, the adversary can first modify the ciphertext component c_{1,i^*} to obtain a new (ill-formed) ciphertext C'_{i^*} and then ask the re-encryption oracle to re-encrypt C'_{i^*} into another ciphertext C'_j for a *corrupted* user j (note that according to the security model, it is legal for the adversary to issue such a query); next, the adversary can modify C'_j to obtain the *right* re-encrypted ciphertext C_j of the challenge ciphertext, and thus he can obtain the underlying plaintext by decrypting C_j with user j's secret key.

Below we give the attack details. For an easy explanation of how the adversary can modify C'_j to obtain the right transformed ciphertext C_j, when describing the underlying re-applicable LTDFs we shall take Matsuda-Nishimaki-Tanaka's concrete realization (recalled in Section 2.3) as the example. Concretely, the adversary works as follows:

1. The adversary first obtains the public parameters PP from the setup oracle.
2. The adversary obtains a public key pk_{i^*} from the uncorrupted key generation oracle. Note that pk_{i^*} will be added in Γ_U by the oracle.
3. The adversary obtains a public/secret key pair (pk_j, sk_j) from the corrupted key generation oracle. Note that pk_j will be added in Γ_C by the oracle.

[1] In the ReEnc algorithm of Matsuda-Nishimaki-Tanaka PRE scheme, it even overlooks this checking.

4. The adversary submits (pk_{i^*}, m_0, m_1) to the challenge oracle, and then is given the challenge ciphertext $C_{i^*} = (vk^*, c_{1,i^*}, c_2^*, c_3^*, \tau^*, \sigma^*)$, where c_{1,i^*} is the output of function $\mathsf{LossyEval}$. Here we use Matsuda-Nishimaki-Tanaka's concrete realization of $\mathsf{LossyEval}$ as an example. Wlog, suppose $c_{1,i^*} = (y_1, \mathbf{y}_2) = (y_1, (y_{2,1}, \cdots, y_{2,n}))$.
5. The adversary first randomly picks $\tilde{y}_{2,1}, \cdots, \tilde{y}_{2,n}$ from $\mathbb{G}$, and modifies the challenge ciphertext to obtain a new (ill-formed) ciphertext $C'_{i^*} = (vk^*, c'_{1,i^*}, c_2^*, c_3^*, \tau^*, \sigma^*)$, where $c'_{1,i^*} = (y_1, (\tilde{y}_{2,1}, \cdots, \tilde{y}_{2,n}))$. Then, the aversary submits $(pk_{i^*}, pk_j, C'_{i^*})$ to the re-encryption oracle. Note that, although $pk_j \in \Gamma_C$, it is legal for the adversary to issue this query, since (pk_{i^*}, C'_{i^*}) is *not* a derivate of (pk_{i^*}, C_{i^*}). Note that, the re-encryption algorithm ReEnc cannot check the validity of the ciphertext component c'_{1,i^*}. So, the re-encryption oracle will return the re-encrypted ciphertext $C'_j = \mathsf{ReEnc}(PP, \mathsf{ReKeyGen}(PP, sk_{i^*}, sk_j), C'_{i^*})$ to the adversary.
   According to the re-encryption algorithm, we have $C'_j = (vk^*, c'_{1,j}, c_2^*, c_3^*, \tau^*, \sigma^*)$, where $c'_{1,j} = \mathsf{ReEval}(s_{i^* \leftrightarrow j}, c'_{1,i^*})$. According to Matsuda-Nishimaki-Tanaka's concrete realization of ReEval, we have $c'_{1,j} = (y_1, (\tilde{y}'_{2,1}, \cdots, \tilde{y}'_{2,n})) = \left(y_1, \left(\tilde{y}_{2,1} \cdot y_1^{z_{1,i^* \leftrightarrow j}}, \cdots, \tilde{y}_{2,n} \cdot y_1^{z_{n,i^* \leftrightarrow j}}\right)\right)$.
   Now, from $c'_{1,j} = (y_1, (\tilde{y}'_{2,1}, \cdots, \tilde{y}'_{2,n}))$, the adversary can compute the following
$$\begin{aligned} c_{1,j} &= \left(y_1, (\frac{\tilde{y}'_{2,1} y_{2,1}}{\tilde{y}_{2,1}}, \cdots, \frac{\tilde{y}'_{2,n} y_{2,n}}{\tilde{y}_{2,n}})\right) \\ &= \left(y_1, (\frac{\tilde{y}_{2,1} \cdot y_1^{z_{1,i^* \leftrightarrow j}} y_{2,1}}{\tilde{y}_{2,1}}, \cdots, \frac{\tilde{y}_{2,n} \cdot y_1^{z_{n,i^* \leftrightarrow j}} y_{2,n}}{\tilde{y}_{2,n}})\right) \\ &= \left(y_1, (y_{2,1} \cdot y_1^{z_{1,i^* \leftrightarrow j}}, \cdots, y_{2,n} \cdot y_1^{z_{n,i^* \leftrightarrow j}})\right). \end{aligned}$$
   Observe that $c_{1,j}$ is indeed equivalent to the result of $\mathsf{ReEval}(s_{i^* \leftrightarrow j}, c_{1,i^*})$. Thus, $C_j = (vk^*, c_{1,j}, c_2^*, c_3^*, \tau^*, \sigma^*)$ is indeed the result of $\mathsf{ReEnc}(PP, \mathsf{ReKeyGen}(PP, sk_{i^*}, sk_j), C_{i^*})$, which is an encryption of m_b. Now, the adversary can obtain the underlying plaintext m_b by decrypting the re-encrypted ciphertext C_j using the secret key sk_j, and obviously can break the CCA-security of Matsuda-Nishimaki-Tanaka PRE scheme.

The above attack can also be simply extended to the case that the user j is uncorrupted. In this case, the adversary $\mathcal{A}$ directly request (pk_j, C_j) to the decryption oracle, which will return the plaintext m_b to $\mathcal{A}$.

6 Discussions and Conclusion

The authors constructed 10 games, Game-0 to Game-10, to prove the CCA security of the PRE scheme developed in [12] , where Game 0 is just the CCA definitional game of PRE (recalled in Section 3). Our above concrete attack shows that the CCA security proof in [12] must be flawed. A careful investigation shows that the proof has already been flawed for the arguments between Game-0 and Gmae-1.

Specifically, the Game-1 in [12] is identical to Game-0, except for some modifications on the actions of decryption oracles. Particularly, in Game-1, when queried with $(pk_j, C_j) = ((s_{\text{rltdf}}, \tau), (vk, c_{1,j}, c_2, c_3, \tau', \sigma))$ such that $(vk, c_2, c_3, \tau', \sigma) = (vk^*, c_2^*, c_3^*, \tau^*, \sigma^*)$, where $C_{i^*} = (vk^*, c_{1,i^*}, c_2^*, c_3^*, \tau^*, \sigma^*)$ and $pk_{i^*} = (s^*_{\text{rltdf}}, \tau^*)$ are the challenge ciphertext and the challenge public-key (w.r.t. the uncorrupted player i^*) respectively, the decryption oracle performs as follows: it first checks whether $\mathsf{PrivReEval}(x^*, \tau', \tau, s_{\text{rltdf}}) = c_1$ where x^* denotes the random value used to form the challenge ciphertext (i.e., $c^*_{1,i^*} = \mathsf{LossyEval}(s^*_{\text{rltdf}}, x^*)$, $c_2^* = \mathsf{F}_{\text{abo}}(s^*_{\text{abo}}, vk^*, x^*)$ and $c_3^* = h(x^*) \oplus m_b$), and then outputs $m = c_3 \oplus h(x^*)$ in case the check is successful. It is claimed in [12] that this modification does not affect any success probability of an adversary, and in fact, the probability that the queries satisfying $\mathsf{PrivReEval}(x^*, \tau', \tau, s_{\text{rltdf}}) = c_1$ is *negligible.*

But, our concrete attack shows that the above security arguments are wrong. In particular, the ciphertext C_j generated by the concrete attack satisfies both $(vk, c_2, c_3, \tau', \sigma) = (vk^*, c_2^*, c_3^*, \tau^*, \sigma^*)$ and $\mathsf{PrivReEval}(x^*, \tau', \tau, s) = c_1$ (which occurs with probability 1), and the decryption oracle will return back the challenge plaintext m_b in this case.

The PRE scheme developed in [12] is based upon the CCA-secure public-key encryption (PKE) scheme of Peikert and Waters (that is in turn based upon LTDFs) [13], which can be viewed as an extension of the Peikert-Waters PKE scheme into the proxy re-encryption setting. One key difference between the Peikert-Waters PKE construction and the PRE construction of [12] is that: all components in the ciphertext of the Peikert-Waters PKE are signed by the one-time signature (under the verification key vk), but the key component c_1 is not signed in the ciphertext of the PRE of [12]. Of course, signing c_1 can prevent our concrete attack, but the resultant scheme is not a PRE scheme any longer (particularly, the proxy cannot translate ciphertexts among players, as the underlying signing key w.r.t. vk is unknown to the proxy). From our view, developing CCA proxy re-encryption without pairing in the standard model may need significantly new ideas and techniques. As a consequence, it is still an open problem to come up with a (bidirectional or unidirectional) proxy re-encryption scheme without pairings in the standard model.

Acknowledgement

We sincerely thank Dr. Goichiro Hanaoka for helpful discussions.

References

[1] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. In *NDSS*. The Internet Society, 2005.

[2] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. *ACM Trans. Inf. Syst. Secur.*, 9(1):1–30, 2006.

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

[4] Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited (Preliminary Version). In *STOC*, pages 209–218, 1998.

[5] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[6] Ran Canetti and Susan Hohenberger. Chosen-Ciphertext Secure Proxy Re-Encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 185–194. ACM, 2007.

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

[8] Sherman S. M. Chow, Jian Weng, Yanjiang Yang, and Robert H. Deng. Efficient Unidirectional Proxy Re-Encryption. In Daniel J. Bernstein and Tanja Lange, editors, *AFRICACRYPT*, volume 6055 of *Lecture Notes in Computer Science*, pages 316–332. Springer, 2010.

[9] Robert H. Deng, Jian Weng, Shengli Liu, and Kefei Chen. Chosen-Ciphertext Secure Proxy Re-encryption without Pairings. In Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong, editors, *CANS*, volume 5339 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.

[10] Benoît Libert and Damien Vergnaud. Unidirectional Chosen-Ciphertext Secure Proxy Re-Encryption. `http://hal.inria.fr/inria-00339530/en/`. This is the extended version of [11].

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

[12] Toshihide Matsuda, Ryo Nishimaki, and Keisuke Tanaka. CCA Proxy Re-Encryption without Bilinear Maps in the Standard Model. In P.Q. Nguyen and D. Pointcheval, editors, *PKC*, volume 6056 of *Lecture Notes in Computer Science*, page 261C278. Springer, 2010.

[13] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *STOC*, pages 187–196. ACM, 2008.

[14] Jian Weng, Min-Rong Chen, Yanjiang Yang, Robert H. Deng, Kefei Chen, and Feng Bao. CCA-Secure Unidirectional Proxy Re-Encryption in the Adaptive Corruption Model without Random Oracles. *Science China: Information Science*, 53(3):593–606, 2010.