CCA Proxy Re-Encryption without Bilinear Maps in the Standard Model

Toshihide Matsuda[1], Ryo Nishimaki[2], and Keisuke Tanaka[1]

[1] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology,W8-55, 2-12-1 Ookayama Meguro-ku, Tokyo 152-8552, Japan
{matsuda5, keisuke}@is.titech.ac.jp
[2] NTT Laboratories, 3-9-11 Midori-cho, Musashino-shi, Tokyo, 180-8585, Japan
nishimaki.ryo@lab.ntt.co.jp

Abstract. Proxy re-encryption (PRE) is a cryptographic application proposed by Blaze, Bleumer, and Strauss. It is an encryption system with a special property in which the semi-honest third party, *the proxy*, can re-encrypt ciphertexts for Alice into other ciphertexts for Bob without using Alice's secret key. We can classify PRE into bidirectional and unidirectional schemes. Canetti and Hohenberger formalized the semantic security under chosen ciphertext attack for PRE, the PRE-CCA security. Several schemes satisfy the PRE-CCA security as a bidirectional or unidirectional scheme. However, some PRE schemes need a bilinear map in the standard model, and the other PRE schemes are PRE-CCA secure in the random oracle model before our work. In this paper, we construct a bidirectional PRE-CCA proxy re-encryption *without bilinear maps in the standard model*. We study lossy trapdoor functions (LTDFs) based on the decisional Diffie-Hellman (DDH) assumption proposed by Peikert and Waters. We define a new variant of LTDFs, *re-applicable LTDFs*, which are specialized LTDFs for PRE, and use them for our scheme.

1 Introduction

1.1 Background

Proxy re-encryption (PRE) is a cryptographic application proposed by Blaze, Bleumer, and Strauss [4]. It is an encryption system with a special property in which the semi-honest third party, *the proxy*, can re-encrypt ciphertexts for Alice into other ciphertexts for Bob without using Alice's secret key. In the other words, if the proxy has a re-encryption key $rk_{A\leftrightarrow B}$ from Alice to Bob, the proxy can translate a ciphertext C_A under Alice's public key pk_A into another ciphertext C_B under Bob's public key pk_B. This translation requires only $rk_{A\leftrightarrow B}$ and keeps the message m secret for the proxy. There are many PRE cryptographic applications, such as email-forwarding, secure file systems, DRM, and secure mailing lists [2, 4, 11, 21].

Ivan and Dodis classified PRE into two types, *bidirectional* and *unidirectional* [10]. The former means that, if the proxy has the re-encryption key $rk_{A\leftrightarrow B}$, the proxy cannot only re-encrypt ciphertexts from Alice to Bob, but also in the opposite direction. That is, we can assume that the re-encryption key $rk_{A\leftrightarrow B}$ from Alice to Bob is identical to

$rk_{B \leftrightarrow A}$ from Bob to Alice. On the other hand, the latter means that, the re-encryption key $rk_{A \to B}$ from Alice to Bob never helps re-encryption of the opposite direction. They also considered the notions of *single-hop* and *multi-hop*. The former means that a re-encrypted ciphertext cannot be further re-encrypted. In contrast, the later means that a ciphertext can be re-encrypted from Alice to Bob to Carol and so on. We discuss only bidirectional schemes in this paper.

The PRE-CCA Security. The security notion of indistinguishability against chosen cipher text attacks (IND-CCA) on encryption systems was proposed by Naor and Yung [15]. Many IND-CCA secure encryption systems have been proposed.

Canetti and Hohenberger applied the notion of IND-CCA to PRE for the bidirectional scheme [6]. They formalized the security notion as the (bidirectional) PRE-CCA security. They also investigated simulation-based security definitions that guarantee universally composable security. They constructed a bidirectional and multi-hop PRE-CCA scheme based on the decisional bilinear Diffie-Hellman (DBDH) assumption, which requires bilinear maps. Later, some research groups proposed bidirectional or unidirectional PRE-CCA schemes with bilinear maps or in the random oracle model [7, 12, 20, 22].

1.2 Our Contribution

We construct a bidirectional and multi-hop PRE-CCA scheme without *bilinear maps* in *the standard model*. All previous PRE-CCA secure schemes in the standard model use bilinear maps. Our scheme is constructed in three steps.

First, we define a new cryptographic primitive, *re-applicable lossy trapdoor functions* (re-applicable LTDFs), which are specialized lossy trapdoor functions for PRE. They consist of nine algorithms and a set of tags, (ParGen, LossyGen, LossyEval, LossyInv, ReIndex, ReEval, PrivReEval, Trans, FakeKey) and $\mathcal{T}$. Second, we construct a bidirectional PRE-CCA scheme by using re-applicable LTDFs. We modify the original Peikert and Waters encryption scheme on several points. Third, we construct re-applicable LTDFs based on the *decisional Diffie-Hellman* (DDH) assumption.

Our Techniques. As stated above, we modify the original Peikert and Waters encryption scheme on several points for PRE. Our PRE scheme uses an index of all-but-one functions as the public parameter. The original scheme uses this index as a part of a public key. Our scheme generates a signature of a part of a ciphertext (c_2, c_3) in the encryption scheme. The original scheme generates a signature of all the main parts of a ciphertext (c_1, c_2, c_3) in the encryption process. The most different point is that our scheme re-encrypts c_1 from pk_A to pk_B by using $rk_{A \leftrightarrow B}$.

We modify LTDFs proposed by Peikert and Waters on one point for construction of re-applicable LTDFs. It is that an injective index is not $\mathsf{Enc}_{pk}(\boldsymbol{I})$, but $\mathsf{Enc}_{pk}(\tau \boldsymbol{I})$, where τ is a tag and $\boldsymbol{I}$ is the identity matrix. This modification is a technical change that satisfies the definition of re-applicable LTDFs.

The Peikert and Waters Encryption. Peikert and Waters proposed LTDFs and constructed an IND-CCA public-key encryption by using LTDFs [17]. We briefly review their encryption scheme. Let f_s and $f_{s'}$ be functions with a domain $\{0,1\}^n$, where f_s is an injective function with a trapdoor td, and $f_{s'}$ is a lossy function of which the size of a range is 2^{n-k} at most. LTDFs have the following property: Given f, the distinguisher cannot distinguish whether f is f_s or $f_{s'}$. They constructed them as $f_s(\boldsymbol{x}) = \boldsymbol{x}\mathsf{Enc}_{pk}(\boldsymbol{I})$ and $f_{s'}(\boldsymbol{x}) = \boldsymbol{x}\mathsf{Enc}_{pk}(\boldsymbol{0})$ where $\boldsymbol{I}$ and $\boldsymbol{0}$ is the identity and the zero matrix, and Enc is a homomorphic matrix encryption. From the homomorphism of Enc, we obtain $f_s(\boldsymbol{x}) = \boldsymbol{x}\mathsf{Enc}_{pk}(\boldsymbol{I}) = \mathsf{Enc}_{pk}(\boldsymbol{x}\boldsymbol{I}) = \mathsf{Enc}_{pk}(\boldsymbol{x})$ and $f_{s'}(\boldsymbol{x}) = \boldsymbol{x}\mathsf{Enc}_{pk}(\boldsymbol{0}) = \mathsf{Enc}_{pk}(\boldsymbol{x}\boldsymbol{0}) = \mathsf{Enc}_{pk}(\boldsymbol{0})$. We can reconstruct $\boldsymbol{x}$ from $f_s(\boldsymbol{x})$ with a secret key sk. On the other hand, we never obtain $\boldsymbol{x}$ from $f_{s'}(\boldsymbol{x})$ *information-theoretically*. They also proposed *all-but-one trapdoor functions* which have similar property to LTDFs. Let g_{s',b^*} be a function with a domain $B\times\{0,1\}^n$, where B is a finite set and $b^* \in B$. For every $b \neq b^*$, $g_{s',b^*}(b,\cdot)$ is an invertible function with a trapdoor td. On the other hand, $g_{s',b^*}(b^*,\cdot)$ is a lossy function.

Peikert and Waters constructed their IND-CCA encryption scheme by using f_s and g_{s',b^*} as follows. The encryption algorithm randomly chooses $x \in \{0,1\}^n$ and selects a key pair of a one-time signature (vk, sk_σ). Then, it computes a ciphertext as

$$C = (vk, c_1, c_2, c_3, \sigma) = (vk, f_s(x), g_{s',b^*}(vk, x), h(x) \oplus m, \mathsf{SigSign}(sk_\sigma, c_1, c_2, c_3)),$$

where h is a pair-wise independent hash and $\mathsf{SigSign}$ is a signing algorithm in a signature scheme. They proved that this scheme satisfied the IND-CCA security.

Observation of the Peikert and Waters Encryption: Free Part for Signature. The above encryption algorithm must sign all (c_1, c_2, c_3) and make the signature σ. The signature σ and vk guarantees that (c_1, c_2, c_3) is signed with the signing key sk_σ. That is, (c_1, c_2, c_3) is fixed by σ and vk. For this fixed (c_1, c_2, c_3), the Peikert and Waters encryption achieves the IND-CCA security.

However, we find that it is not necessary to sign all (c_1, c_2, c_3) in order to achieve the IND-CCA security. Moreover, we find that it does not need to sign c_1. The reason is as follows. If (c_1, c_2, c_3) is fixed by σ and vk, randomness x and a message m also are fixed because of the injectivity of f_s and g_{s',b^*}. That is, we can consider σ as a signature of x and m as well as (c_1, c_2, c_3). In addition, if x and m are determined, (c_1, c_2, c_3) is determined. We understand that it is necessary to sign x and m, not (c_1, c_2, c_3). We replace a signature of (c_1, c_2, c_3) with a signature of (c_2, c_3). A pair of x and m is fixed similarly to the case of (c_1, c_2, c_3) because of the injectivity of g_{s',b^*}. That is, the signature of (c_2, c_3) performs tasks of a signature of (c_1, c_2, c_3). We do not need the signature of c_1 to achieve the IND-CCA security. This free c_1 is very important in constructing our proposed scheme, which satisfies the bidirectional PRE-CCA security.

Remarks on Our Scheme. We discuss two points of our scheme: 1. Comparison of efficiency and 2. Construction based on other assumptions.

Our scheme uses an index of re-applicable LTDFs as a public key. We represent this key as an $n \times n$ matrix, which has n^2 group elements. The public parameter contains $n \times n$ matrix as well as public keys since all-but-one trapdoor functions are based on the DDH assumption. One ciphertext and one re-encryption key have $O(n)$ group elements. However, in most of the previous schemes, they consist of a constant number of group elements. For example, one public key is one group element, and one ciphertext consists of five group elements, a verification key, and a signature in the Canetti and Hohenberger's bidirectional scheme, which satisfies the PRE-CCA security as well as ours [6]. Time complexity as well as space complexity are larger than others. Therefore, future work is to construct more efficient schemes.

LTDFs are constructed from various assumptions. Therefore, one may think that we can construct PRE-CCA secure schemes from other assumptions. However, we do not know how to construct PRE-CCA secure schemes from other assumptions now. Factoring, quadratic residue, RSA, or Paillier-based LTDFs do not clearly satisfy the definition of re-applicable lossy trapdoor functions. One might think that decision linear or lattice-based LTDFs work in our proposal, but this is not clear. In other words, they do not guarantee several properties of re-applicable LTDFs. However, we might be able to use LTDFs based on them with other techniques for PRE.

1.3 Related Work

We review previous work on PRE and LTDFs.

Proxy Re-Encryptions. Mambo and Okamoto first proposed the concept of proxy encryption, which delegates the ability of decryption through an interaction [13]. Based on their concept, Blaze, Bleumer, and Strauss proposed the notion of proxy cryptography. PRE is one concept in proxy cryptography [4]. Their construction is nearly similar to the ElGamal encryption and satisfies the CPA security. A re-encryption key is made from the division of secret keys in the scheme. Later, Ivan and Dodis point out that Blaze et.al.'s scheme is bidirectional and multi-hop. Ateniese, Fu, Green, and Hohenberger proposed the first unidirectional scheme, which was single-hop and satisfied the CPA security with a bilinear map [2]. Deng, Weng, Liu, and Chen constructed a bidirectional and single-hop PRE scheme without a bilinear map in the random oracle model [7]. Libert and Vergnaud discussed the unidirectional PRE-CCA security and constructed an unidirectional and single-hop PRE-RCCA[3] scheme with a bilinear map [12]. Shao and Cao proposed an unidirectional and single-hop PRE-CCA scheme in the random oracle model [20][4]. Weng, Chow, Yang, and Deng improved Shao and Cao's scheme, and their scheme also is in the random oracle model [22]. We put this previous work in chronological order in Figure 1. ROM stands for the random oracle model.

Hohenberger, Rothblum, shelat, and Vaikuntanathan argued the existence of obfuscators with PRE and practically constructed an obfuscator as a PRE scheme [9]. Their scheme is CPA-secure.

Recently, Ateniese, Benson, and Hohenberger introduced an additional property on PRE, which is key-privacy (or anonymous) [1]. It is desirable to have this property, if

[3] This security notion is truly weaker than the CCA security and stronger than the CPA security. An adversary can use a decryption oracle in a restricted way. If he sends messages (m_0, m_1) to the challenger and obtains a challenge c, the adversary cannot query ciphertexts of challenge messages m_0 and m_1 to the decryption oracle.

[4] Two research groups posed questions on the security model of this paper and published their discussions on ePrint Archive [22, 23]. However, they do not effect our results and we do not mention this in this paper.

Authors	Direction	Hop	Assumption	Security	Bilinear Map	ROM
Blaze et.al. [4]	$\leftrightarrow$	multi	DDH on $\mathbb{G}_p$	PRE-CPA	no	no
Ateniese et.al. [2]	$\rightarrow$	single	eDBDH	PRE-CPA	yes	no
Canetti and Hohenberger [6]	$\leftrightarrow$	multi	DBDH	PRE-CCA	yes	no
Libert and Vergnaud [12]	$\rightarrow$	single	3-wDBDHI	PRE-RCCA	yes	no
Deng et.al. [7]	$\leftrightarrow$	single	mCDH on $\mathbb{G}_p$	PRE-CCA	no	yes
Shao and Cao [20]	$\rightarrow$	single	DDH on $\mathbb{Z}_{N^2}$	PRE-CCA	no	yes
Weng et.al. [22]	$\rightarrow$	single	CDH on $\mathbb{G}_p$	PRE-CCA	no	yes
This paper	$\leftrightarrow$	multi	DDH on $\mathbb{G}_p$	PRE-CCA	*no*	*no*

Fig. 1. Comparison of our work with previous work.

the proxy can freely re-encrypt ciphertexts. Our scheme does not have the key-private property. They constructed a key-private scheme with bilinear maps in the standard model, which satisfies the PRE-CPA security, not CCA.

Lossy Trapdoor Functions. We review previous work related to LTDFs. Peikert and Waters proposed notions of LTDFs [17]. They showed cryptographic applications based on LTDFs, such as a (ordinary) trapdoor function, a collision-resistant hash function, an oblivious transfer, and an IND-CCA encryption scheme. They also showed the constructions of LTDFs based on the DDH assumption and the LWE assumption. They mentioned that the Paillier encryption realized LTDFs by the similar methodology to the construction based on the DDH assumption. Later, Rosen and Segev noted that the Damgård-Jurik encryption scheme simply satisfied the definition of LTDFs because of its number-theoretic property [18]. The Damgård-Jurik encryption scheme is considered as a generalized Paillier encryption. In addition to [18], Freedman, Goldreich, Kiltz, Rosen, and Segev proposed more constructions of LTDFs [8]. They are based on the d-Linear assumption and the QR assumption. Mol and Yilek showed that slightly LTDFs are sufficient for constructing a IND-CCA secure public-key encryption scheme [14]. Slightly LTDFs lost a $(1 - \omega(\log n))$ fraction of all its input bits.

Other applications of LTDFs have been proposed. Rosen and Segev proposed a new primitive, *a one-way function under correlated products* [19]. They showed the construction of a one-way function under correlated products from LTDFs and the IND-CCA secure encryption by using this primitive. Boldyreva, Fehr, and O'Neill applied LTDFs to the construction of *deterministic encryption* [5]. They constructed a CCA-secure deterministic encryption scheme in the standard model, where the CCA security meant the sense of the semantic security on a message, not indistinguishability of messages. Bellare, Hofheinz, and Yilek formalized a new security notion of encryption, *selective opening attack*, which meant that it kept secret even if an adversary selectively obtained *messages* and *randomness* of ciphertexts [3]. They used LTDFs for the above purpose. Nishimaki, Fujisaki, and Tanaka used all-but-one trapdoor functions for the universally composable commitment scheme [16]. They first proposed a non-interactive string-commitment scheme, which is universally composable.

Organization

In Section 2, we show preliminaries to describe our scheme. In Section 3, we define re-applicable LTDFs. In Section 4, we review the definition of bidirectional PRE schemes, propose our scheme, and describe a sketch of a proof of it. In Section 5, we construct re-applicable LTDFs based on the DDH assumption.

2 Preliminaries

In this section, we show preliminaries to describe our scheme.

2.1 Notation

Let S be a finite set. $s \in_R S$ denotes that the element s is chosen from S uniformly at random. For probabilistic algorithm A, $y \leftarrow A(x)$ denotes that A outputs y on input x with uniform randomness. If A runs in time polynomial in the security parameter, then A is a probabilistic polynomial-time (PPT) algorithm. We say that function $f : \mathbb{N} \to [0, 1]$ is negligible in $\lambda \in \mathbb{N}$ if for every constant $c \in \mathbb{N}$ there exists $k_c \in \mathbb{N}$ such that $f(\lambda) < \lambda^{-c}$ for any $\lambda > k_c$. We say that function $g : \mathbb{N} \to [0, 1]$ is overwhelming in $\lambda \in \mathbb{N}$ if function $f(\lambda) = 1 - g(\lambda)$ is negligible in $\lambda \in \mathbb{N}$. Let X_λ and Y_λ denote random variables over a finite set $Z_\lambda \subset \{0, 1\}^\lambda$, where $\lambda \in \mathbb{N}$ is the security parameter. We say that X_λ and Y_λ are (computationally) indistinguishable if, for every distinguisher D, $|\Pr[D(X_\lambda) = 1] - \Pr[D(Y_\lambda) = 1]|$ is negligible in $\lambda \in \mathbb{N}$. We say that X_λ and Y_λ are statistically indistinguishable if $\sum_{z \in Z_\lambda} |\Pr[X_\lambda = z] - \Pr[Y_\lambda = z]$ is negligible in $\lambda \in \mathbb{N}$.

2.2 DDH Assumption

We review the DDH assumption. Let $\mathcal{G}$ be an algorithm that takes as input a security parameter λ and outputs a tuple $(p, \mathbb{G}, g)$, where p is a prime with $2^{\lambda-1} \leq p < 2^\lambda$, $\mathbb{G}$ is a cyclic group of prime order p, and g is a generator of $\mathbb{G}$.

Assumption 1 (The Decisional Diffie-Hellman Assumption) For any PPT adversary A, the advantage $\mathsf{Adv}_A(k)$ is negligible in the security parameter k.

$$\mathsf{Adv}_A(k) = |\Pr[A((p, \mathbb{G}, g), g^a, g^b, g^{ab}) = 1] - \Pr[A((p, \mathbb{G}, g), g^a, g^b, g^c) = 1]|$$

The probability is over the random choices of $(p, \mathbb{G}, g) \leftarrow \mathcal{G}(\lambda)$, the random choices of $a, b, c \in \mathbb{Z}_p$ and the random coin of A.

2.3 All-But-One Trapdoor Functions

We review all-but-one trapdoor functions to describe our scheme. All-but-one trapdoor functions are made from the DDH assumption[17].

Definition 1 (All-but-one trapdoor functions) A collection of (n, k)-all-but-one trapdoor functions is a tuple of PPT algorithms $(\mathsf{G}_{\mathrm{abo}}, \mathsf{F}_{\mathrm{abo}}, \mathsf{F}^{-1}_{\mathrm{abo}})$ and sequence of branch sets $B = \{B_\lambda\}$ such that:

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). For every $b \in B_\lambda \backslash \{b^*\}$, 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 $\{0,1\}^n$, 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.

3 Re-Applicable Lossy Trapdoor Functions

In this section, we propose a new primitive, *re-applicable LTDFs*, which is an extension of LTDFs. Peikert and Waters proposed *LTDFs* and *all-but-one functions* in STOC'08 [17].

For our purpose, we transform LTDFs in several points. First, we add one algorithm, the *parameter-generation* algorithm ParGen. This algorithm generates public parameters which is common to every algorithm and applied in every evaluation. We introduce ParGen since PRE schemes are used in the multi-user setting, not single-user setting. In addition, the validity checks of ciphertexts require that each ciphertext of the ElGamal encryption in an index of LTDFs has common randomness for every user.

Second, we modify LTDFs so that the function-generation algorithm receives a *tag* in a set of tags $\mathcal{T}$, not injective or lossy commands. For every tag τ, except one special lossy tag τ_{los}, the function-generation algorithm outputs an index that represents an injective function. On the other hand, the function-generation algorithm given τ_{los} outputs an index that represents a lossy function.

Third, we define five new algorithms ReIndex, ReEval, PrivReEval, Trans, and FakeKey. ReIndex, ReEval, PrivReEval, and Trans are deterministic, and FakeKey is probabilistic. We apply ReIndex for generating re-encryption keys, ReEval for evaluation of re-encryption, and PrivReEval for a validity check of ciphertexts. The algorithms Trans and FakeKey are only used in the proof. The algorithm Trans guarantees the transitivity between re-encryption keys. In other word, we can make a re-encryption key $rk_{j \leftrightarrow k}$ from $rk_{i \leftrightarrow j}$ and $rk_{i \leftrightarrow k}$. The algorithm FakeKey generates a pair of public and re-encryption keys $(pk_j, rk_{i \leftrightarrow j})$ from another public key pk_i. Moreover, even if pk_i represents a lossy function, FakeKey always outputs pk_j, which represents an *injective* function. This property is necessary for the last modification in our proof. We introduce $\mathcal{T}$ to provide this property.

We call this new primitive, *a collection of re-applicable LTDFs*. They are specialized LTDFs for PRE. If we unify ParGen and LossyGen, ignore the other new algorithms, and define $\mathcal{T} = \{\tau_{\text{inj}}, \tau_{\text{los}}\}$, we can consider this new primitive as (ordinary) LTDFs proposed by Peikert and Waters.

Definition 2 (Re-applicable LTDFs with respect to function indices) Let (ParGen, LossyGen, LossyEval, LossyInv, ReIndex, ReEval, PrivReEval, Trans, FakeKey) be a tuple of PPT algorithms, and $\mathcal{T}$ be a set of tags that contains one lossy element τ_{los}. The algorithm ParGen(1^λ) outputs a public parameter par. The other algorithms apply the parameter par to their computations. Hereafter, we omit the input of the public parameter par for the algorithms.

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

Injectivity; For every public parameter $\mathsf{par} \leftarrow \mathsf{ParGen}(1^\lambda)$ and every tag $\tau \in \mathcal{T}\backslash\{\tau_{\mathrm{los}}\}$, $\mathsf{LossyGen}(\tau)$ 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 $|f_{s,\tau_{\mathrm{los}}}(\{0, 1\}^n)| \leq 2^{n-k}$.

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 $(\mathsf{par}, s_{\mathrm{los}}, \tau')$, 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 outputs from $\mathsf{LossyGen}(\tau)$ and $\mathsf{LossyGen}(\tau_{\mathrm{los}})$. 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 $\tau_i \neq \tau_{\mathrm{los}}$ and $\tau_j \neq \tau_{\mathrm{los}}$. 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 $x \in \{0, 1\}^n$, $x = \mathsf{LossyInv}(td_j, \tau_i, \mathsf{ReEval}(s_{i\leftrightarrow j}, \mathsf{LossyEval}(s_i, x)))$. We remark 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 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 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.)

4 Bidirectional and Multi-Hop PRE-CCA Scheme

In this section, we first review the definition of a bidirectional PRE scheme. Then, we describe our scheme and show a sketch of the proof.

A bidirectional PRE scheme consists of six algorithms Π = (Setup, KeyGen, Enc, Dec, $\mathsf{ReKeyGen}$, and ReEnc) as follows. $PP \leftarrow \mathsf{Setup}(1^\lambda)$: Given a security parameter 1^λ, the setup algorithm outputs a public parameter PP. This algorithm is executed by a trusted third party. $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$: Given a public parameter PP, the key generation algorithm outputs a public key pk and a secret key sk. $C \leftarrow \mathsf{Enc}(PP, pk, m)$: Given a public key pk and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext C, where $\mathcal{M}$ is a message space. $rk_{i\leftrightarrow j} \leftarrow \mathsf{ReKeyGen}(PP, sk_i, sk_j)$: Given a pair of secret keys sk_i, sk_j, where $i \neq j$, this algorithm outputs a re-encryption key $rk_{i\leftrightarrow j}$. We call $rk_{i\leftrightarrow j}$ the re-encryption key between i and j. $C_j \leftarrow \mathsf{ReEnc}(PP, rk_{i\leftrightarrow j}, C_i)$: Given a re-encryption key $rk_{i\leftrightarrow j}$ between i and j and a ciphertext C_i for i, this algorithm outputs another ciphertext C_j for j or the error symbol $\perp$. $m \leftarrow \mathsf{Dec}(PP, sk, C)$: Given a public key sk and a ciphertext C, the decryption algorithm outputs a message m or the error symbol $\perp$.

If the following two conditions holds, we say that the PRE scheme Π satisfies correctness. For every PP which is output from $\mathsf{Setup}(1^\lambda)$, every (pk, sk) which is output from $\mathsf{KeyGen}(PP)$ and every message $m \in \mathcal{M}$, the probability $\Pr[C \leftarrow \mathsf{Enc}(PP, pk, m) : \mathsf{Dec}(PP, sk, C) = m]$ is overwhelming. For every natural number $n \in \mathbb{N}$, every PP which is output from $\mathsf{Setup}(1^\lambda)$, every $(pk_1, sk_1) \ldots (pk_n, sk_n)$ which are outputs from $\mathsf{KeyGen}(PP)$, every message $m \in M$, and every $rk_{1\leftrightarrow 2} \ldots rk_{n-1\leftrightarrow n}$ which are outputs from $\mathsf{ReKeyGen}(rk_i, rk_{i+1})$ for each $i \in [1, n-1]$, the probability $\Pr[C_1 \leftarrow \mathsf{Enc}(PP, pk_1, m) : \mathsf{Dec}(PP, sk_n, \mathsf{ReEnc}(PP, rk_{n-1\leftrightarrow n}, \ldots \mathsf{ReEnc}(PP, rk_{1\leftrightarrow 2}, C_1) \ldots)) = m]$ is overwhelming.

4.1 Bidirectional and Multi-Hop PRE-CCA Security

We prove that our scheme satisfies the PRE-CCA security in the full version of this paper. This security notion was proposed by Canetti and Hohenberger [6].

Definition 3 (Bidirectional and Multi-Hop PRE-CCA Security) Let λ be the security parameter, A be an oracle TM, representing the adversary, and Γ_U and Γ_C be date structures. Date structures Γ_U and Γ_C are first initialized as empty in the game. The game consists of an execution of A with the following oracles, which can be invoked multiple times in any order, subject to the constraint below:

Setup Oracle: This oracle can be queried first in the game only once. This oracle makes a public parameter as $PP \leftarrow \mathsf{Setup}(1^\lambda)$. A is given PP.

Uncorrupted key generation: This oracle generates a new key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$ and adds pk in Γ_U, where PP is generated from the setup oracle. A is given pk.

Corrupted key generation: This oracle generates a new key pair $(pk, sk) \leftarrow \mathsf{KeyGen}(PP)$ and adds pk in Γ_C, where PP is generated from the setup oracle. A is given (pk, sk).

Challenge oracle: This oracle can be queried only once. On input (pk^*, m_0, m_1), the oracle chooses a bit $b \leftarrow \{0,1\}$ and returns $C^* = \mathsf{Enc}(PP, pk^*, m_b)$. We call pk^* the *challenge key* and C^* the *challenge ciphertext*. (We require the challenge key $pk^* \in \varGamma_U$ for A to win.)

Re-encryption key generation: On input (pk_i, pk_j) from the adversary, this oracle return the re-encryption key $rk_{i\leftrightarrow j} = \mathsf{ReKeyGen}(sk_i, sk_j)$, where sk_i and sk_j are the secret keys that correspond to pk_i and pk_j, respectively.
We require that pk_i and pk_j are in $\varGamma_C$, or alternatively both are in $\varGamma_U$. We do not allow for re-encryption key generation queries between a corrupted key and an uncorrupted key.

Re-encryption oracle: On input (pk_i, pk_j, C_i), if $pk_j \in \varGamma_C$ and (pk_i, C_i) is a *derivative* of (pk^*, C^*), then return a special symbol $\perp$, which is not in the domain of messages or ciphertext. Else, return the re-encrypted ciphertext $C_j = \mathsf{ReEnc}(\mathsf{ReKeyGen}(sk_i, sk_j), C_j)$. Derivatives of (pk^*, C^*) are defined inductively as follows.

- (pk^*, C^*) is a derivative of itself.
- If (pk, C) is a derivative of (pk^*, C^*), and (pk', C') is a derivative of (pk, C), then (pk', C') is a derivative of (pk^*, C^*).
- If A has queried the re-encryption oracle on input (pk, pk', C) and obtained response C', then (pk', C') is a derivative of (pk, C).
- If A has queried the re-encryption key generation oracle on input (pk, pk') or (pk', pk), and $C' = \mathsf{ReEnc}(\mathsf{ReKeyGen}(sk, sk'), C)$, then (pk', C') is a derivative of (pk, C), where sk and sk' are the secret keys that correspond 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^*, C^*), or pk is not in $\varGamma_U \cup \varGamma_C$, then return a special symbol $\perp$ which is not in the domain of messages. Else, return $\mathsf{Dec}(sk, C)$, where sk is the secret key that corresponds 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 $pk^* \in \varGamma_U$, then output 1. Else, output 0:

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

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

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

4.2 Description of Our Scheme

We next describe our scheme. Let λ be the security parameter, and let n, k, k', k'' and v be parameters depending on λ. Let $(\mathsf{SigGen}, \mathsf{SigSign}, \mathsf{SigVer})$ be a strongly unforgeable one-time signature scheme where verification keys are in $\{0,1\}^v$. Let $(\mathsf{ParGen}$, $\mathsf{LossyGen}$, $\mathsf{LossyEval}$, $\mathsf{LossyInv}$, $\mathsf{ReIndex}$, ReEval, $\mathsf{PrivReEval}$, Trans, $\mathsf{FakeKey})$ be a collection of re-applicable (n, k)-LTDFs and $\mathcal{T}$ be a set of tags. Let $(\mathsf{G}_{\text{abo}}, \mathsf{F}_{\text{abo}}, \mathsf{F}_{\text{abo}}^{-1})$ 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 $\{0,1\}^{k''}$. We require that the above parameters are $(k + k') - (k'' + n) \geq \delta = \delta_1 + \delta_2$ for some $\delta_1 = \omega(\log\lambda)$ and $\delta_2 = \omega(\log\lambda)$. Our cryptosystem has message space $\{0,1\}^{k''}$.

The algorithm Setup generates a public parameter $PP = (s_{\text{abo}}, \mathsf{par}, h)$, and the algorithm KeyGen makes a pair of keys $(pk, sk) = ((s_{\text{rltdf}}, \tau), (td_{\text{rltdf}}, s_{\text{rltdf}}, \tau))$. Except a tag τ, we can consider that both Setup and KeyGen are the same algorithm as the key generation algorithm in the Peikert and Waters encryption. The algorithm Enc is also the same algorithm as the encryption algorithm in the Peikert and Waters encryption, except that c_1 is not signed for the re-encryption. The algorithm ReKeyGen makes a re-encryption key, and the ReEnc re-encrypts a ciphertext into another ciphertext. These algorithms only use ReIndex and ReEval in re-applicable LTDFs. The algorithm Dec is the same algorithm as the decryption algorithm in the Peikert and Waters encryption, expect that, if a ciphertext is re-encrypted, it applies PrivReEval for the validity check of ciphertexts.

Setup(1^λ): Setup(1^λ) 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: $\mathsf{par} \leftarrow \mathsf{ParGen}(1^\lambda)$. Finally, it chooses a hash function $h \leftarrow \mathcal{H}$. It outputs a public parameter as $PP = (s_{\text{abo}}, \mathsf{par}, h)$.
(The algorithm Setup erases the trapdoor td_{abo} because the following algorithms do not use td_{abo}.)

KeyGen(PP): KeyGen takes $PP = (s_{\text{abo}}, \mathsf{par}, h)$ as input. It chooses a tag $\tau \in \mathcal{T} \backslash \{\tau_{\text{los}}\}$ and generates an injective index of re-applicable LTDFs: $(s_{\text{rltdf}}, td_{\text{rltdf}}) \leftarrow \mathsf{LossyGen}(\tau)$. A public key consists of the injective function index and the tag, and a secret key consisting of the trapdoor of s_{rltdf} and the tag: $pk = (s_{\text{rltdf}}, \tau)$, and $sk = (td_{\text{rltdf}}, s_{\text{rltdf}}, \tau)$.

Enc(PP, pk, m): Enc takes (PP, pk, m) as input, where $PP = (s_{\text{abo}}, \mathsf{par}, h)$ is a tuple of public parameters, $pk = (s_{\text{rltdf}}, \tau)$ is a public key, and $m \in \{0,1\}^\ell$ is a message. It chooses $x \in \{0,1\}^n$ uniformly at random. It generates a key-pair for the one-time signature scheme: $(vk, sk_\sigma) \leftarrow \mathsf{SigGen}(1^\lambda)$, then computes

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

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

ReKeyGen(PP, sk_i, sk_j) : ReKeyGen takes as input (PP, sk_i, sk_j), where $(sk_i, sk_j) = ((td_i, s_i, \tau_i), (td_j, s_j, \tau_j))$. It computes $s_{i \leftrightarrow j} \leftarrow \mathsf{ReIndex}(td_i, td_j)$, then outputs a re-encryption key as $rk_{i \leftrightarrow j} = s_{i \leftrightarrow j}$.

ReEnc($PP, rk_{i \leftrightarrow j}, C_i$) : ReEnc takes $(rk_{i \leftrightarrow j}, C_i)$ as input, where $rk_{i \leftrightarrow j} = s_{i \leftrightarrow j}$ is a re-encryption key and $C_i = (vk, c_{1,i}, c_2, c_3, \tau, \sigma)$ is a ciphertext. It 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 ciphertext for the user with sk_j.

Dec(PP, sk, C) : Dec takes (PP, sk, C) as input, where $PP = (s_{\text{abo}}, \mathsf{par}, h)$ is a tuple of public parameters, $sk = (td_{\text{rltdf}}, s_{\text{rltdf}}, \tau)$ is a secret key, and $C = (vk, c_1, c_2, c_3, \tau', \sigma)$ is a ciphertext. It first checks $\mathsf{SigVer}(vk, (c_2, c_3, \tau'), \sigma) = 1$; if not, it outputs $\perp$.

It then compute $x = \mathsf{LossyInv}(td_{\text{rltdf}}, \tau', c_1)$. If $\tau = \tau'$ then it checks $c_1 = \mathsf{LossyEval}(s_{\text{rltdf}}, x)$, otherwise, it checks $\mathsf{PrivReEval}(x, \tau', \tau, s_{\text{rltdf}}) = c_1$; if not, it outputs $\perp$. It also checks $c_2 = \mathsf{F}_{\text{abo}}(s_{\text{abo}}, vk, x)$; if not, it outputs $\perp$. Finally, it outputs $m = c_3 \oplus h(x)$. (We note that, if C was not re-encrypted, then $\tau = \tau'$. On the other hand, if C was re-encrypted, $\tau \neq \tau'$.)

4.3 Security of Our Scheme

In this section, we claim the following theorem and describe a sketch of the proof. We give the detail proof in the full version of this paper.

Theorem 1 The above proposed scheme satisfies the PRE-CCA security.

We now show modifications of games from Game_0 to Game_{10} to prove Theorem 1. Game_0 is identical to the PRE-CCA game. In Game_{10}, no adversary can win with meaningful probability. Every modification from Game_i to Game_{i+1} is perfect, statistically or computationally indistinguishable for each $i \in [0, n-1]$. Therefore, we conclude that no adversary also can win with meaningful probability in Game_0.

In every game, let $C^* = (vk^*, c_1^*, c_2^*, c_3^*, \tau^*, \sigma^*)$ and $pk^* = (s^*_{\text{rltdf}}, \tau^*)$ be the challenge ciphertext and the challenge public key, respectively.

Game_0**:** This game is identical to the PRE-CCA game.

Game_1**:** Let x^* denote a random input applied to making the challenge ciphertext. (i.e. $c_1^* = \mathsf{LossyEval}(s^*_{\text{rltdf}}, x^*)$, $c_2^* = \mathsf{F}_{\text{abo}}(s_{\text{abo}}, vk^*, x^*)$, $c_3^* = h(x^*) \oplus m_b$.)
Then, we modify the decryption oracle as follows: The decryption oracle is given a decryption query $(pk, C) = ((s, \tau), (vk, c_1, c_2, c_3, \tau', \sigma))$, where pk is limited to an output by the corrupted or the uncorrupted oracles. If (pk, C) is a derivative of (pk^*, C^*), then it outputs $\perp$. Else if the decryption query satisfies that $(vk, c_2, c_3, \tau', \sigma) = (vk^*, c_2^*, c_3^*, \tau^*, \sigma^*)$ and $\mathsf{PrivReEval}(x^*, \tau', \tau, s) = c_1$, then it outputs $m \leftarrow c_3 \oplus h(x^*)$. Otherwise, it outputs $\mathsf{Dec}(sk, C)$ in the ordinary decryption processes.

This modification does not affect any success probability of an adversary. From the injectivity of $\mathsf{PrivReEval}(\cdot, \tau', \tau, s)$, if $\mathsf{PrivReEval}(x^*, \tau', \tau, s) = c_1$, then we obtain $\mathsf{LossyInv}(td, \tau', c_1) = x^*$. In fact, the probability that the above queries satisfy $\mathsf{PrivReEval}(x^*, \tau', \tau, s) = c_1$ is negligible. We discuss this fact in the modification between Game_9 and Game_{10}. However, this check is necessary for the following modifications.

Game_2**:** We add the following check to the decryption oracle after checking a derivative: The decryption oracle is given a decryption query $(pk, C) = ((s, \tau), (vk, c_1, c_2, c_3, \tau', \sigma))$. The decryption oracle always outputs $\perp$, if $vk = vk^*$ and $(c_2, c_3, \tau', \sigma) \neq (c_2^*, c_3^*, \tau^*, \sigma^*)$.

This modification is negligible for the success probability of an adversary from the strongly existential unforgeability of the signature scheme.

Game_3**:** Let vk^* be the verification key used by the challenge oracle. We modify the setup oracle on a lossy branch as follows: We replace generates $(s_{\text{abo}}, td_{\text{abo}}) \leftarrow \mathsf{G}_{\text{abo}}(1^\lambda, 0^\nu)$ with $(s_{\text{abo}}, td_{\text{abo}}) \leftarrow \mathsf{G}_{\text{abo}}(1^\lambda, vk^*)$. The lossy branch is changed the verification key vk^* from a zero-padding 0^ν.

This modification is negligible for the success probability of an adversary from the computational indistinguishability of all-but-one trapdoor functions.

Game_4**:** Let $(s_{\text{abo}}, td_{\text{abo}})$ be an index of all-but-one trapdoor functions and its trapdoor. We modify the decryption oracle as follows: The decryption oracle uses the trapdoor td_{abo} to decrypt ciphertext, when it receives a ciphertext $C = (vk, c_1, c_2, c_3, \tau', \sigma)$. That is, in the case of $(vk, c_2, c_3, \tau', \sigma) \neq (vk^* c_2^*, c_3^*, \tau^*, \sigma^*)$ and $vk \neq vk^*$, we replace $x \leftarrow \mathsf{LossyInv}(td_{\text{rltdf}}, \tau', c_1)$ with $x \leftarrow \mathsf{F}_{\text{abo}}^{-1}(td_{\text{abo}}, vk, c_2)$, and proceed with the decryption processes (In the other cases, the decryption oracle executes the operations defined in Game_1 and Game_2.).

This modification does not affect any success probability of an adversary because of the injectivity of $\mathsf{LossyEval}(s, \cdot)$, $\mathsf{F}_{\text{abo}}(s_{\text{abo}}, vk, \cdot)$, and $\mathsf{PrivReEval}(\cdot, \tau', \tau, s)$. In this modification, $\mathsf{F}_{\text{abo}}(s_{\text{abo}}, vk, \cdot)$ is an injective function since vk is not the lossy branch vk^*.

In the following games, the challenger manages uncorrupted re-encryption keys to apply the table. That is, in the first of this game, the challenger makes an empty table, which memorizes uncorrupted re-encryption keys. We set that a pair of keys (pk_1, sk_1) is the first output by the uncorrupted oracle. At n-th query, the uncorrupted oracle outputs a pair of keys (pk_n, sk_n) and makes the re-encryption keys $rk_{1\leftrightarrow n}, \cdots, rk_{n-1\leftrightarrow n}$ and $rk_{n\leftrightarrow 1}, \cdots, rk_{n\leftrightarrow n-1}$. The challenger adds re-encryption keys to the table.

Game_5**:** We modify the re-encryption oracle as follows: The re-encryption oracle takes as query $(pk_a, pk_b, C_a) = ((s_a, \tau_a), (s_b, \tau_b), (vk, c_{1,a}, c_2, c_3, \tau'_a, \sigma))$ in the game. If pk_a is corrupted, then it evaluates $x \leftarrow \mathsf{LossyInv}(td_a, \tau'_a, c_{1,a})$. Then, it makes $c_{1,b} = \mathsf{PrivReEval}(x, \tau_a, \tau_b, s_b)$. If pk_b is uncorrupted, it searches $rk_{a\leftrightarrow b}$ in the re-encryption keys table and evaluates $c_{1,b} \leftarrow \mathsf{ReEnc}(rk_{a\leftrightarrow b}, c_{1,a})$. Then, it outputs $C_b = (vk, c_{1,b}, c_2, c_3, \tau'_a, \sigma)$ as a re-encrypted ciphertext for pk_b.

This modification does not affect any success probability of an adversary because of the equivalence between $\mathsf{PrivReEval}(\cdot, \tau_a, \tau_b, s_b)$ and $\mathsf{ReEnc}(rk_{a\leftrightarrow b}, \mathsf{LossyEval}(s_a., \cdot))$.

Game_6**:** We modify the re-encryption key generation oracle as follows: Given a pair (pk_a, pk_b), it searches the re-encryption keys $rk_{a\leftrightarrow b}$ from the table. Then, it outputs this re-encryption key $rk_{a\leftrightarrow b}$.

This modification does not affect any success probability of an adversary.

Game_7**:** We define the number $q_{A,\text{unc}}$ as the maximum number of times that an adversary A queries the uncorrupted oracle in the game. We modify the challenge oracle as follows.
First, the challenger chooses a random number $r \in \{1, \ldots, q_{A,\text{unc}}\}$. If the challenge oracle receives the challenge key $pk^* \neq pk_r$, the challenger outputs a random bit b and aborts this game, where pk_r is the r-th public key output by the uncorrupted oracle. Otherwise, it proceeds with this game.

This modification reduces the success probability of an adversary to $1/q_{A,\mathrm{unc}}$ fraction. However, this is not an important reduction since $q_{A,\mathrm{unc}}$ is polynomial of the security parameter λ.

Game_8**:** We modify the uncorrupted key generation oracle as follows.
First, it executes the following preprocessing. It choose a random number $r \in \{1, \ldots, q_{A,\mathrm{unc}}\}$ and generates a pair of keys $(pk_r, sk_r) \leftarrow \mathsf{KeyGen}(PP)$. We describe $pk_r = (s_r, \tau_r)$ and $sk_r = (td_r, s_r, \tau_r)$. Then, for every $i \in \{1, \ldots, q_{A,\mathrm{unc}}\}\backslash\{r\}$, it uses the fake key generation algorithm to generate a public key and the re-encryption key: it computes $(s_i, s_{r\leftrightarrow i}, \tau_i) \leftarrow \mathsf{FakeKey}(s_r, \tau_r)$ and sets $pk = (s_i, \tau_i)$, $rk_{r\leftrightarrow i}$. Then, it computes that $rk_{i\leftrightarrow j} = s_{i\leftrightarrow j} \leftarrow \mathsf{Trans}(s_{r\leftrightarrow i}, s_{r\leftrightarrow j})$ for every $i, j \in \{1, \ldots, q_{A,\mathrm{unc}}\}$ and $i \neq j$. It adds every above-mentioned re-encryption key $\{rk_{i\leftrightarrow j}\}_{i,j\in\{1,\ldots,q_{A,\mathrm{unc}}\},i\neq j}$ to the re-encryption keys table.
At $j \in \{1, \ldots, q_{A,unc}\}$ times query for the uncorrupted oracle, it outputs the public key $pk_j = s_j$ generated by the above preprocessing. The re-encryption key oracle and the re-encryption oracle execute their processes with the preprocessed table. The challenge oracle applies the above number r to the check $pk^* = pk_r$.

This modification is negligible for the success probability of an adversary from the statistical indistinguishability of the fake key algorithm.

Game_9**:** We modify the above preprocessing as follows: We replace the first key generation $pk_r = s_r \leftarrow \mathsf{LossyGen}(\tau_r)$ with $pk_r = s_r \leftarrow \mathsf{LossyGen}(\tau_{\mathrm{los}})$, where τ_{los} is a lossy tag.

This modification is negligible for the success probability of an adversary from the computational indistinguishability of LTDFs.

Game_{10}**:** We modify the decryption oracle as follows: The decryption oracle always outputs $\perp$, when it receives the query $C = (vk^*, c_1, c_2^*, c_3^*, \sigma^*)$ and $pk = s_{\mathrm{rltdf}}$.

This modification is negligible for the success probability of an adversary from the fact that the average-case min-entropy of x^* is high, and every public key $pk = s$, except a challenge key, represents an injective function $f_{s,\star}$. That is, adversary never compute c_1 such that $f_{s,\star}^{-1}(c_1) = x^*$ information-theoretically. This fact implies that Game_{10} is statistically close to Game_9. Due to this modification, we attach tags to re-applicable LTDFs.

From the above sketch, we transform the PRE-CCA game (i.e. Game_0) into the last game (i.e. Game_{10}). In the last game, Game_{10}, we conclude that any adversary does not detect which message is encrypted. The reason is, $h(x^*)$ is statistically close to U_ℓ since x^* has high average-case min-entropy and $h(\cdot)$ can extracts a random string from x^*.

5 Realization of Re-Applicable LTDFs Based on DDH Assumption

In this section, we describe the realization of re-applicable LTDFs from the DDH assumption. We modify the construction proposed by Peikert and Waters.

We now describe LTDFs based on the DDH assumption. We modify the construction as the proposed by Peikert and Waters on two points. One is a division of one function generation algorithm into two algorithms. The other is a change on an encrypting matrix in the injective function generator. Peikert and Waters proposed the function-generation algorithm which creates a function index as a ciphertext of a matrix. This ciphertext is encrypted with the ElGamal encryption on matrices. When generating an injective-function index, it encrypts the matrix $\boldsymbol{I} = (g^{\delta_{i,j}})_{i,j}$ on $\mathbb{G}^{n\times n}$, where $\delta_{i,j}$ is the Kronecker delta. When generating a lossy-function index, it encrypts the matrix $\boldsymbol{0} = (e)_{i,j}$ on $\mathbb{G}^{n\times n}$, where e is the identity in $\mathbb{G}$ (i.e. $e = g^0$). We also do the same for generating a lossy-function index, but we do another procedure for generating an injective-function index. We define a set of tags $\mathcal{T}$ as a group $\mathbb{G}$ and a special element τ_{los} as the identity $e \in \mathbb{G}$. We use a matrix $\boldsymbol{M} = (\tau^{\delta_{i,j}})_{i,j}$ on $\mathbb{G}^{n\times n}$, where τ is any element in $\mathbb{G}$. When generating a injective-function index, we set τ with $\tau \neq \tau_{\text{los}}$. When generating a lossy-function index, we set $\tau = \tau_{\text{los}}$.

– *Generation of a public parameter.* A parameter generator ParGen first executes $\mathcal{G}$, and $\mathcal{G}$ outputs a tuple $(p, \mathbb{G}, g)$. It next selects random numbers $r_1, \dots, r_n \in_R \mathbb{Z}_p$, then makes a public parameter $\boldsymbol{C}_1$ as

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

– *Generation of function indices.* A function generator LossyGen takes as input $\boldsymbol{C}_1$ and a tag τ, where $\boldsymbol{C}_1$ is the public parameter and τ is an element in $\mathbb{G}$. (We note that if $\tau = e$, it means execution of the lossy mode, otherwise, execution of the injective mode.) It first selects random elements $z_1, z_2, \dots, z_n \in_R \mathbb{Z}_p$, then computes a function index as

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

The function index consists of $(\boldsymbol{C}_1, \boldsymbol{C}_2)$. A trapdoor consists of the random elements $z = (z_1, \dots, z_n)$.

– *Evaluation algorithm.* An evaluation algorithm LossyEval takes as input $(\boldsymbol{C}_1, \boldsymbol{C}_2, \boldsymbol{x})$, where $(\boldsymbol{C}_1, \boldsymbol{C}_2)$ is a function index, and $\boldsymbol{x} = (x_1, \dots, x_n) \in \{0,1\}^n$ is an n-bit input interpreted as a vector.
It evaluates the linear product of $\boldsymbol{x}$ and $\boldsymbol{C}_1$: That is, $y_1 = \boldsymbol{x}\boldsymbol{C}_1 = \prod_{i=1}^n (c_i)^{x_i}$.Next, it evaluates the product of $\boldsymbol{x}$ and $\boldsymbol{C}_2$: That is, $\boldsymbol{y}_2 = \boldsymbol{x}\boldsymbol{C}_2 = (\prod_{i=1}^n c_{i,1}^{x_i}, \cdots, \prod_{i=1}^n c_{i,n}^{x_i}) = ((\prod_{i=1}^n c_i^{z_1 x_i})\tau^{x_1}, \cdots, (\prod_{i=1}^n c_i^{z_n x_i})\tau^{x_n})$. Finally, it outputs $(y_1, \boldsymbol{y}_2)$.

– *Inversion algorithm.* An inversion algorithm LossyInv takes as input $(td, \tau, (y_1, \boldsymbol{y}_2))$, where trapdoor information td consists of $z = (z_1, \dots, z_n)$, τ, which is an element in $\mathbb{G}\backslash\{e\}$, and $\boldsymbol{y}_2 = (y_{2,1}, \cdots, y_{2,n}) \in \mathbb{G}^{1\times n}$. It computes that $\boldsymbol{w} = (y_{2,1} \cdot y_1^{-z_1}, y_{2,2} \cdot y_1^{-z_2}, \cdots, y_{2,n} \cdot y_1^{-z_n})$. Then, if j-th element of $\boldsymbol{w}$ is the identity element of $\mathbb{G}$, then it sets $x_j = 0$, else if j-th element of $\boldsymbol{w}$ is τ then it sets $x_j = 1$; otherwise, it output $\perp$. Finally, it outputs $x = (x_1, x_2, \dots, x_n)$.

We can show that the above four algorithms satisfy injectivity, $(n, n - \log p)$-lossiness, and indistinguishability based on the DDH assumption. These proofs are nearly similar to the proof of Peikert and Waters [17]; therefore, we omit them in this paper.

Next, we show that the above algorithms satisfy the definition of re-applicable LTDFs.

Re-applying with respect to function indices: Let τ_i and τ_j be tags different from τ_{los} in $\mathcal{T}$. Let (s_i, td_i) and (s_j, td_j) be outputs from $\mathsf{LossyGen}(\tau_i)$ and $\mathsf{LossyGen}(\tau_j)$. We now define an algorithm $\mathsf{ReIndex}$, which takes td_i and td_j as inputs and outputs $s_{i\leftrightarrow j} = td_j - td_i = (z'_1 - z_1, z'_2 - z_2, \dots, z'_n - z_n) = (z_{1,i\leftrightarrow j}, \dots, z_{n,i\leftrightarrow j})$.
We next define an algorithm ReEval. The algorithm ReEval takes as input $(s_{i\leftrightarrow j}, (y_1, \boldsymbol{y}_2))$, where $(y_1, \boldsymbol{y}_2) = (y_1, (y_{2,1}, y_{2,2}, \cdots, y_{2,n}))$ is an output from $\mathsf{LossyEval}(s_i, \boldsymbol{x})$. It computes that $\boldsymbol{y}'_2 = (y'_{2,1}, y'_{2,2}, \dots, y'_{2,n}) = (y_{2,1} \cdot y_1^{z_{1,i\leftrightarrow j}}, y_{2,2} \cdot y_1^{z_{2,i\leftrightarrow j}}, \cdots, y_{2,n} \cdot y_1^{z_{n,i\leftrightarrow j}})$. Then, it outputs $(y_1, \boldsymbol{y}'_2)$.
For the above elements, we can describe that $(y_1, \boldsymbol{y}_2) = (g^{\sum_{i=1}^n x_i r_i}, (g^{z_1 \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_1}, \cdots, g^{z_n \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_n}))$ and $(y_1, \boldsymbol{y}'_2) = (g^{\sum_{i=1}^n x_i r_i}, (g^{z'_1 \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_1}, \cdots, g^{z'_n \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_n}))$. Therefore, we have $\boldsymbol{w}' = (\tau_i^{x_1}, \cdots, \tau_i^{x_n})$ in the algorithm $\mathsf{LossyInv}(td_j, \tau_i, (y_1, \boldsymbol{y}'_2))$. That is, we obtain that $\boldsymbol{x} = \mathsf{LossyInv}(td_j, \tau_i, (y_1, \boldsymbol{y}'_2))$.

Generating proper outputs: Let τ_i, τ_j, (s_i, td_i), (s_j, td_j), $s_{i\leftrightarrow j}$ $\boldsymbol{x}$, and $(y_1, \boldsymbol{y}'_2)$ be defined similarly to the above paragraph. We call $(y_1, \boldsymbol{y}'_2)$ a *proper* output for $\boldsymbol{x}$, τ_i, τ_j, and s_j, if they satisfy $(y_1, \boldsymbol{y}'_2) = \mathsf{ReEval}(s_{i\leftrightarrow j}, \mathsf{LossyEval}(s_i, \boldsymbol{x}))$ for some s_i made from a tag τ_i and $s_{i\leftarrow j}$. We can uniquely describe a proper $(y_1, \boldsymbol{y}'_2)$ as $(g^{\sum_{i=1}^n x_i r_i}, (g^{z'_1 \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_1}, \cdots, g^{z'_n \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_n}))$ from the above algorithms, where $td_j = (z'_1, \cdots, z'_n)$.
We define a new algorithm $\mathsf{PrivReEval}$, which takes $\boldsymbol{x}$, τ_i, τ_j, and s_j as input, where $\boldsymbol{x} = (x_1, \cdots, x_n)$ is n-bits input. It computes $(\hat{y}_1, \hat{\boldsymbol{y}}_2) \leftarrow \mathsf{LossyEval}(s_j, \boldsymbol{x})$. It makes $\hat{\boldsymbol{y}}'_2$ from $\hat{\boldsymbol{y}}_2$ in the following process: for each $i \in [1, n]$, if $x_i = 1$ then $\hat{y}'_{2,i} \leftarrow \hat{y}_{2,i} \tau_j^{-1} \tau_i$, else $\hat{y}_{2,i} \leftarrow \hat{y}_{2,i}$, where $\hat{y}_{2,i}$ and $\hat{y}'_{2,i}$ are the i-th elements of $\hat{\boldsymbol{y}}_2$ and $\hat{\boldsymbol{y}}'_2$. Finally it outputs $(\hat{y}_1, \hat{\boldsymbol{y}}'_2)$. The algorithm $\mathsf{PrivReEval}(\boldsymbol{x}, \tau_i, \tau_j, s_j)$ always computes a proper output for $\boldsymbol{x}$, τ_i, τ_j, and s_j. The reason is that, from an output $(\hat{y}_1, \hat{\boldsymbol{y}}_2) = (g^{\sum_{i=1}^n x_i r_i}, (g^{z'_1 \sum_{i=1}^n x_i r_i} \cdot \tau_j^{x_1}, \cdots, g^{z'_n \sum_{i=1}^n x_i r_i} \cdot \tau_j^{x_n}))$, we verify $(y_1, \hat{\boldsymbol{y}}'_2) = (g^{\sum_{i=1}^n x_i r_i}, (g^{z'_1 \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_1}, \cdots, g^{z'_n \sum_{i=1}^n x_i r_i} \cdot \tau_i^{x_n}))$ that τ_j is replaced with τ_i. This means that $\mathsf{PrivReEval}(\cdot, \tau_i, \tau_j, s_j)$ is equivalent to $\mathsf{ReEval}(s_{i\leftrightarrow j}, \mathsf{LossyEval}(s_i, \cdot))$ as a function.

Transitivity: We define an algorithm Trans, which takes $s_{i\leftrightarrow j}$, $s_{i\leftrightarrow k}$ and outputs $s_{i\leftrightarrow k} - s_{i\leftrightarrow j} = (td_k - td_i) - (td_j - td_i) = td_k - td_j = s_{j\leftrightarrow k}$. That is, $\mathsf{Trans}(s_{i\leftrightarrow j}, s_{i\leftrightarrow k}) \to s_{j\leftrightarrow k}$.

Statistical indistinguishability of the fake key: Now, we define an algorithm $\mathsf{FakeKey}$, which takes a function index s_i and a tag τ_i and makes a fake index s_j, a fake re-key $s_{i\leftrightarrow j}$, and a fake tag τ_j. The fake key generator $\mathsf{FakeKey}$ takes as input $s_i = (\boldsymbol{C}_1, \boldsymbol{C}_2)$ and $\tau_i \in \mathbb{G}$, where $(\boldsymbol{C}_1, \boldsymbol{C}_2)$ is a function index. It then selects a random element $t \in \mathbb{G}$. It next chooses a random number $s_{i\leftrightarrow j} = (z_{1,i\leftrightarrow j}, \dots, z_{n,i\leftrightarrow j}) \in_R \mathbb{Z}_p^n$, and makes a new matrix $\boldsymbol{C}'_2$ as follows.

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

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

From the abode description, outputs of $\mathsf{FakeKey}(s_i, \tau_i)$ and the proper index have the same distribution. The reason is that, when s_i and τ_i is made from the proper way, we can describe $\boldsymbol{C}'_2$ by

$$\boldsymbol{C}'_2 = \begin{pmatrix} g^{r_1(z_1+z_{1,i\leftrightarrow j})} \cdot \tau_i \cdot t \cdots & g^{r_1(z_n+z_{n,i\leftrightarrow j})} \\ \vdots \quad \ddots & \vdots \\ g^{r_n(z_1+z_{1,i\leftrightarrow j})} & \cdots g^{r_n(z_n+z_{n,i\leftrightarrow j})} \cdot \tau_i \cdot t \end{pmatrix} = \begin{cases} c'_{k,\ell} = g^{r_k(z_\ell+z_{\ell,i\leftrightarrow j})} \cdot \tau_i \cdot t & \text{if } k = \ell, \\ c'_{k,\ell} = g^{r_k(z_\ell+z_{\ell,i\leftrightarrow j})} & \text{otherwise,} \end{cases}$$

where $(z_1, \cdots, z_n)$ is the trapdoor of s_i. This means that, conditioned on $t \neq (\tau_i)^{-1}$, the distribution of $\{\tau_i, \tau_j, (s_i, td_j), (s_j, td_j), s_{i\leftrightarrow j}\}$ is identical to $\{\tau_i, \tau_j, \mathsf{LossyGen}(\tau_i), \mathsf{LossyGen}(\tau_j), \mathsf{ReIndex}(td_i, td_j)\}$. That is, distributions between them are statistically indistinguishable since the probability of $t = (\tau_i)^{-1}$ is $1/p$.

Generation of injective functions from lossy functions: Next, we consider $\mathsf{FakeKey}$ which $s_i = (\boldsymbol{C}_1, \boldsymbol{C}_2)$ is output from $\mathsf{LossyGen}(\tau_{\text{los}})$. In this case, we can describe $\boldsymbol{C}'_2$ as

$$\boldsymbol{C}'_2 = \begin{pmatrix} g^{r_1(z_1+z_{1,i\leftrightarrow j})} \cdot t \cdots & g^{r_1(z_n+z_{n,i\leftrightarrow j})} \\ \vdots \quad \ddots & \vdots \\ g^{r_n(z_1+z_{1,i\leftrightarrow j})} & \cdots g^{r_n(z_n+z_{n,i\leftrightarrow j})} \cdot \cdot t \end{pmatrix},$$

where $(z_1, \cdots, z_n)$ are logarithms between $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$. Then, assuming that $t \neq e$, $\mathsf{LossyEval}(s_j, \cdot)$ represents an injective function $f_{s_j,t}$. This probability is $1 - 1/p$ which is overwhelming in the security parameter λ.

References

1. G. Ateniese, K. Benson, and S. Hohenberger. Key-Private Proxy Re-encryption. In M. Fischlin, editor, *CT-RSA*, volume 5473 of *Lecture Notes in Computer Science*, pages 279–294, San Francisco, CA, USA, April 2009. Springer-Verlag.
2. G. Ateniese, K. Fu, M. Green, and S. Hohenberger. Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage. In *Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2005.
3. M. Bellare, D. Hofheinz, and S. Yilek. Possibility and Impossibility Results for Encryption and Commitment Secure under Selective Opening. In A. Joux, editor, *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 1–35, Cologne, Germany, April 2009. Springer-Verlag.
4. M. Blaze, G. Bleumer, and M. Strauss. Divertible Protocols and Atomic Proxy Cryptography. In K. Nyberg, editor, *EUROCRYPT*, volume 1403 of *Lecture Notes in Computer Science*, pages 127–144, Espoo, Finland, May 1998. Springer-Verlag.
5. A. Boldyreva, S. Fehr, and A. O'Neill. On Notions of Security for Deterministic Encryption, and Efficient Constructions without Random Oracles. In D. Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 335–359, Santa Barbara, California, USA, August 2008. Springer-Verlag.
6. R. Canetti and S. Hohenberger. Chosen-ciphertext secure proxy re-encryption. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 185–194, Alexandria, Virginia, USA, Octorber 2007. ACM.

7. R. H. Deng, J. Weng, S. Liu, and K. Chen. Chosen-Ciphertext Secure Proxy Re-encryption without Pairings. In M. K. Franklin, L. C. K. Hui, and D. S. Wong, editors, *CANS*, volume 5339 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2008.
8. D. Freeman, O. Goldreich, E. Kiltz, A. Rosen, and G. Segev. More Constructions of Lossy and Correlation-Secure Trapdoor Functions. In *PKC*, 2010.
9. S. Hohenberger, G. N. Rothblum, abhi shelat, and V. Vaikuntanathan. Securely Obfuscating Re-encryption. In S. P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 233–252, Amsterdam, The Netherlands, February 2007. Springer-Verlag.
10. A. Ivan and Y. Dodis. Proxy Cryptography Revisited. In *NDSS*. The Internet Society, 2003.
11. H. Khurana, J. Heo, and M. Pant. From Proxy Encryption Primitives to a Deployable Secure-Mailing-List Solution. In P. Ning, S. Qing, and N. Li, editors, *ICICS*, volume 4307 of *Lecture Notes in Computer Science*, pages 260–281. Springer, 2006.
12. B. Libert and D. Vergnaud. Unidirectional Chosen-Ciphertext Secure Proxy Re-encryption. In R. Cramer, editor, *Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 360–379, Barcelona, Spain, April 2008. Springer-Verlag.
13. M. Mambo and E. Okamoto. Proxy Cryptosystems: Delegation of the Power to Decrypt Ciphertexts. *IEICE transactions on fundamentals of electronics, Communications and computer sciences*, 80(1):54–63, 1997.
14. P. Mol and S. Yilek. Chosen-Ciphertext Security from Slightly Lossy Trapdoor Functions. In *PKC*, 2010.
15. M. Naor and M. Yung. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *STOC*, pages 427–437, New Orleans, Louisiana, USA, May 1990. ACM.
16. R. Nishimaki, E. Fujisaki, and K. Tanaka. Efficient Non-interactive Universally Composable String-Commitment Schemes. In J. Pieprzyk and F. Zhang, editors, *Provable Security*, volume 5848 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 2009.
17. C. Peikert and B. Waters. Lossy Trapdoor Functions and Their Applications. In R. E. Ladner and C. Dwork, editors, *STOC*, pages 187–196, Victoria, British Columbia, Canada, May 2008. ACM.
18. A. Rosen and G. Segev. Efficient Lossy Trapdoor Functions based on the Composite Residuosity Assumption. Cryptology ePrint Archive, Report 2008/134, 2008. http://eprint.iacr.org/.
19. A. Rosen and G. Segev. Chosen-Ciphertext Security via Correlated Products. In O. Reingold, editor, *TCC*, volume 5444 of *Lecture Notes in Computer Science*, pages 419–436, San Francisco, California, USA, March 2009. Springer-Verlag.
20. J. Shao and Z. Cao. CCA-Secure Proxy Re-encryption without Pairings. In S. Jarecki and G. Tsudik, editors, *Public Key Cryptography*, volume 5443 of *Lecture Notes in Computer Science*, pages 357–376, Irvine, California, USA, March 2009. Springer-Verlag.
21. G. Taban, A. A. Cárdenas, and V. D. Gligor. Towards a secure and interoperable DRM architecture. In M. Yung, K. Kurosawa, and R. Safavi-Naini, editors, *Digital Rights Management Workshop*, pages 69–78. ACM, 2006.
22. J. Weng, S. S. Chow, Y. Yang, and R. H. Deng. Efficient Unidirectional Proxy Re-Encryption. Cryptology ePrint Archive, Report 2009/189, 2009. http://eprint.iacr.org/.
23. X. Zhang, M.-R. Chen, and X. Li. Comments on Shao-Cao's Unidirectional Proxy Re-Encryption Scheme from PKC 2009. Cryptology ePrint Archive, Report 2009/344, 2009. http://eprint.iacr.org.