Proxy Re-Encryption and Re-Signatures from Lattices

Xiong Fan * Feng-Hao Liu †

Abstract

Proxy re-encryption (PRE) and Proxy re-signature (PRS) were introduced by Blaze, Bleumer and Strauss [Eurocrypt '98]. Basically, PRE allows a semi-trusted proxy to transform a ciphertext encrypted under one key into an encryption of the same plaintext under another key, without revealing the underlying plaintext. Since then, many interesting applications have been explored, and many constructions in various settings have been proposed, while PRS allows a semi-trusted proxy to transform Alice's signature on a message into Bob's signature on the same message, but the proxy cannot produce new valid signature on new messages for either Alice or Bob.

Recently, for PRE related progress, Cannetti and Honhenberger [CCS '07] defined a stronger notion – CCA-security and construct a bi-directional PRE scheme. Later on, several work considered CCA-secure PRE based on bilinear group assumptions. Very recently, Kirshanova [PKC '14] proposed the first single-hop CCA1-secure PRE scheme based on learning with errors (LWE) assumption. For PRS related progress, Ateniese and Hohenberger [CCS'05] formalized this primitive and provided efficient constructions in the random oracle model. At CCS 2008, Libert and Vergnaud presented the first multi-hop uni-directional proxy re-signature scheme in the standard model, using assumptions in bilinear groups.

In this work, we first point out a subtle but serious mistake in the security proof of the work by Kirshanova. This reopens the direction of lattice-based CCA1-secure constructions, even in the single-hop setting. Then we construct a single-hop PRE scheme that is proven secure in our new tag-based CCA-PRE model. Next, we construct the *first* multi-hop PRE construction. Lastly, we also construct the *first* PRS scheme from lattices that is proved secure in our proposed unified security model.

1 Introduction

Proxy re-encryption (PRE) allows a (semi-trusted) proxy to transform an encryption of m under Alice's public key into another encryption of the same message under Bob's public key. The proxy, however, cannot learn the underlying message m, and thus both parties' privacy can be maintained. This primitive (and its variants) have various applications ranging from encrypted email forwarding [BBS98], securing distributed file systems [AFGH05], to digital rights management (DRM) systems [Smi05]. In addition application-driven purposes, various works have shown connections between re-encryption (and its variants) with other cryptographic primitives, such as program obfuscation [HRsV07, CCV12, CCL$^+$14] and fully-homomorphic encryption [CLTV15, ABF$^+$13]. Thus studies along this line are both important and interesting for theory and practice.

Another primitive, called proxy re-signature (PRS), allows a semi-trusted proxy to transform Alice's signature σ_A on a message μ into Bob's signature σ_B on the same message μ, but the proxy

*Cornell University, Email: xfan@cs.cornell.edu.This material is based upon work supported by IBM under Agreement 4915013672. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsors.

†Florida Atlantic University, Email: fenghao.liu@fau.edu.

cannot produce new valid signature on new messages for either Alice or Bob. PRS is employed in various applications, such as providing a proof that a certain path in a graph is taken.

Both concepts of PRE and PRS were introduced by Blaze, Bleumer, and Strauss [BBS98], who also gave the first construction of a CPA (i.e. chosen-plaintext attacks) secure *bi-directional multi-hop* PRE scheme under the Decisional Diffie-Hellman assumption, and a restricted PRS construction. Later on, Ateniese and Hohenberger [AH05] formalized security notions for PRS, and gave two PRS constructions (one is bi-directional, and the other one is uni-directional) based on bilinear maps in the random oracle model. Ateniese, Fu, Green and Hohenberger [AFGH05] constructed the first CPA secure *uni-directional* scheme based on bilinear maps, yet their construction can only support a *single-hop* re-encryption. Hohenberger et al. [HRsV07] and Chase et al. [CCV12] used an obfuscation-based approach and constructed CPA secure uni-directional single-hop PRE scheme (and its variants). Recently, Chase et al. [CCL$^+$14], using the obfuscation-based approach, constructed the first CPA secure uni-directional multi-hop PRE scheme based on lattices assumptions.

For the PRE part, as argued that CPA security can be insufficient for some useful scenarios, Canetti and Hohenberger [CH07] considered a natural stronger security notion — chosen-ciphertext attacks (CCA) security where the adversary has access to a decryption oracle. Intuitively, this security notion guarantees that the underlying message of the challenge ciphertext remains hidden even if the adversary can somehow obtain decryptions of other ciphertexts. They give a meaningful security formulation of CCA secure PRE, and then constructed the first CCA-secure bidirectional multi-hop PRE scheme. Later, Shao et al. [SCL10] constructed a CCA-secure uni-directional single-hop PRE, and Chow et al. [CWYD10] proposed another CCA-secure uni-directional scheme in random oracle model. Libert and Vergnaud [LV08b] improved the result by constructing a CCA uni-directional single-hop PRE without random oracles, and this remains the state of the art of the current construction (for the setting of uni-directional CCA-PRE under the definition of [CH07]). We note that it is unclear how to extend security of the previous obfuscation-approach [HRsV07, CCV12, CCL$^+$14] (that are only CPA-secure) to the CCA setting. One particular technical challenge is that the re-encryption key output by the simulator might be distinguishable given the CCA decryption oracle, and thus the previous security analyses cannot go through. For CPA security, our understanding is quite well — we know how to construct PRE schemes that are uni-directional and multi-hop in the standard model. However, for CCA security, our understanding in the standard model is much limited in the following sense. First, there is no known scheme that achieves both uni-directional and multi-hop at the same time. Moreover, all currently known constructions [BBS98, AFGH05, CH07, LV08b, SCL10, CWYD10, ABPW13] are based on Diffie-Hellman-style assumptions. Very recently, Kirshanova [Kir14] proposed a single-hop construction based on lattices, and argued that it is CCA1 secure[1]. However, after a careful examination of her security proof, we found a subtle mistake in the security proof, and thus how to construct a lattice-based PRE that achieves CCA1-security, (even for the single-hop case) remains open.

For the PRS part, Ateniese and Hohenberger [AH05] left some open problems such as how to construct uni-directional PRS where the proxy can only translate signatures in one direction. Can we avoid the random oracle analysis? Libert and Vergnaud [LV08a] answered these questions positively by constructing the first *multi-use* unidirectional PRS in standard model relying on a new computational assumption in bilinear group.

In this paper, we study lattice-based PRE and PRS constructions. In particular, we make

[1]CCA1 security is weaker in the sense that the attacker does not have the decryption oracle after receiving the challenge ciphertext.

contributions in the following four folds:

- First, we point out a subtle mistake in the security proof of the work [Kir14] (the CCA1 construction), and argue that this is not easy to fix.
- Second, we propose a new model called tag-based CCA that lies in between the CCA1 and CCA2 model. We construct a lattice-based PRE scheme that achieves a slightly relaxed version functionality (and thus slightly relaxed security). We then describe a generic transformation from the relaxed functionality to the full-fledged one using know techniques (i.e., zero-knowledge proofs).
- Third, we define a selective notion of tag-based CCA security for multi-hop PRE where the attacker needs to commit to a tree structure for the challenging ciphertext at the beginning. Then we prove that our basic single-hop construction, with a slight modification, can be extended to the multi-hop setting and achieve such a security notion. This is, to our knowledge, the *first* construction of multi-hop PRE that achieves a relaxed yet meaningful notion of CCA security.
- Lastly, we propose a simpler and unified security model for PRS which captures more dynamic settings. We show that the idea of our multi-hop PRE model and the construction can be extended to construct PRS that achieves the security notion. This is the *first* (to our knowledge) multi-hop unidirectional PRS from lattices.

1.1 Technique Highlights

In the following, we highlight our technical ideas for the four contributions as described above.

Part I: the subtle mistake in the work [Kir14]. The subtle mistake comes in the security proof where the work [Kir14] constructs two adjacent hybrids that are distinguishable. For clarification of exposition, we first briefly present the main idea of the construction [Kir14]. Then we will point out where the subtlety is and explain why the problem cannot be easily fixed.

Basically, the PRE construction can be regarded as an extension of CCA-secure public key encryption scheme in [MP12]. For concreteness, we consider two users: User 1 has public key $\mathsf{pk}_1 = (\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \mathbf{H})$, and User 2 has public key $\mathsf{pk}_2 = (\mathbf{A}'_0, \mathbf{A}'_1, \mathbf{A}'_2, \mathbf{H}')$, where each public key consists of four matrices. The secret key of User 1 consists of low-norm matrices $\mathbf{R}_1, \mathbf{R}_2$ satisfying $\mathbf{A}_1 = -\mathbf{A}_0\mathbf{R}_1, \mathbf{A}_2 = -\mathbf{A}_0\mathbf{R}_2$, and it is similar for the case of User 2. We note that the readers here do not need to worry about the dimensions. To encrypt under pk_1, we consider an encryption matrix $\mathbf{A}_u = [\mathbf{A}_0|\mathbf{A}_1 + \mathbf{H}\mathbf{G}|\mathbf{A}_2 + \mathbf{H}_u\mathbf{G}]$, where $\mathbf{H}_u$ is a random invertible matrix (as a tag to the ciphertext), then encrypt messages using the dual-Regev style encryption [GPV08], i.e. $\mathsf{ct} = \boldsymbol{s}^\mathsf{T}\mathbf{A}_u + \boldsymbol{e} + \mathsf{encode}(m)$. Similarly, we can encrypt under pk_2 with the same structure.

To generate a re-encryption key from User 1 to User 2, the work [Kir14] considers a short matrix $\mathbf{X}$ satisfying the following relation:

$$[\mathbf{A}_0|\mathbf{A}_1 + \mathbf{H}\mathbf{G}|\mathbf{A}_2 + \mathbf{H}_u\mathbf{G}] \begin{bmatrix} \mathbf{X}_{00} & \mathbf{X}_{01} & \mathbf{X}_{02} \\ \mathbf{X}_{10} & \mathbf{X}_{11} & \mathbf{X}_{12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}'_0|\mathbf{A}'_1 + \mathbf{H}'\mathbf{G}|\mathbf{A}'_2 + \mathbf{H}_u\mathbf{G}].$$

In particular, for the last column of the re-encryption key matrix, it holds that

$$[\mathbf{A}_0|\mathbf{A}_1 + \mathbf{H}\mathbf{G}] \begin{bmatrix} \mathbf{X}_{02} \\ \mathbf{X}_{12} \end{bmatrix} = \mathbf{A}'_2 - \mathbf{A}_2. \tag{1}$$

It is not hard to see that $\mathsf{ct} \cdot \mathbf{X} = \boldsymbol{s}^{\mathsf{T}} \cdot \mathbf{A}'_u + \tilde{\boldsymbol{e}} + \mathsf{encode}(m)$, a ciphertext of m under pk_2, so the correctness property is guaranteed.

To prove security, the work [Kir14] uses a standard reduction argument based on the LWE assumption: suppose there exists an adversary that can break the PRE scheme, then there exists a reduction, with oracle access to the adversary, who can break the underlying LWE assumption. For this type of proofs, typically the reduction needs to embed the hard instance (LWE instance for this case), then simulates a scheme (PRE) to the adversary, and finally the reduction can use the adversary to break the underlying hardness assumption. It is **crucially important** that the simulated scheme cannot be distinguished by the adversary; otherwise, the adversary can always output $\perp$ if he detects the scheme is different from the real scheme, and such adversary is useless to the reduction. The security proof in the work [Kir14] missed this point. At a high level, her reduction simulated a PRE scheme that *can* be distinguishable by the adversary easily, so the whole argument breaks down. Below we further elaborate on the details.

For simplicity we consider a simple case where there are only two honest users, Users 1 and 2 and the adversary only gets one re-encryption key from User 1 to User 2. The challenge ciphertext comes from an encryption of User 1, i.e. pk_1. For such case, the reduction of the work [Kir14] pre-selects a tag matrix $\mathbf{H}_{u^*}$ (for the challenge ciphertext), matrices $\mathbf{R}^*_1, \mathbf{R}^*_2$, and then embeds an LWE instance $\mathbf{A}^*$ in the encryption matrix: $\mathbf{A}^*_u = [\mathbf{A}^* | -\mathbf{A}^*\mathbf{R}^*_1 | -\mathbf{A}^*\mathbf{R}^*_2 + (\mathbf{H}_u - \mathbf{H}_{u^*})\mathbf{G}]$. In this case, the reduction sets $\mathsf{pk}_1 = (\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_2, \mathbf{H})$ to be $(\mathbf{A}^*, -\mathbf{A}^*\mathbf{R}^*_1 - \mathbf{H}^*\mathbf{G}, -\mathbf{A}^*\mathbf{R}^*_2 - \mathbf{H}_{u^*}\mathbf{G}, \mathbf{H}^*)$ for some random invertible $\mathbf{H}^*$.

To generate re-encryption key from the challenge user 1 to User 2, the reduction first pre-samples small matrices $\mathbf{X}_{00}, \mathbf{X}_{01}, \mathbf{R}'_1, \mathbf{R}'_2$, and a random invertible matrix $\mathbf{H}'$. Then it computes:

$$\mathbf{A}'_0 = [\mathbf{A}^* | -\mathbf{A}^*\mathbf{R}^*_1] \begin{bmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{bmatrix}, \quad \mathbf{A}'_i = [\mathbf{A}^* | -\mathbf{A}^*\mathbf{R}^*_1] \begin{bmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{bmatrix} \cdot \mathbf{R}'_i, \forall i = 1, 2$$

The reduction sets

$$\mathsf{pk}_2 = (\mathbf{A}'_0, \mathbf{A}'_1, \mathbf{A}'_2, \mathbf{H}'), \quad \mathsf{rk}_{1\to 2} = \begin{bmatrix} \begin{pmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{pmatrix} & \begin{pmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{pmatrix} \mathbf{R}'_1 & \begin{pmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{pmatrix} \mathbf{R}'_2 \\ 0 & 0 & \mathbf{I} \end{bmatrix}$$

generated as above. Then obviously the matrices $\mathbf{A}'_1, \mathbf{A}'_2$ can be expressed as $\mathbf{A}'_1 = \mathbf{A}'_0\mathbf{R}'_1, \mathbf{A}'_2 = \mathbf{A}'_0\mathbf{R}'_2$, where $\mathbf{R}'_1, \mathbf{R}'_2$ are small matrices and still act as secret key for User 2. Therefore, the reduction can still use the same algorithm in the real scheme to answer decryption queries for User 2.

However, if $\mathbf{A}'_2$ is generated in this way, then it is easy to check and compare with Equation (1):

$$[\mathbf{A}_0 | \mathbf{A}_1 + \mathbf{H}\mathbf{G}] \begin{bmatrix} \mathbf{X}_{02} \\ \mathbf{X}_{12} \end{bmatrix} = [\mathbf{A}^* | -\mathbf{A}^*\mathbf{R}^*_1] \begin{bmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{bmatrix} \cdot \mathbf{R}'_2 \neq \mathbf{A}'_2 - \mathbf{A}_2. \tag{2}$$

This means adversary, given the simulated $\mathsf{pk}_1, \mathsf{pk}_2, \mathsf{rk}_{1\to 2}$, adversary can easily tell whether they are from the real scheme or the simulated scheme. Thus, the security proof in this way [Kir14] is not correct.

A straightforward fix would be to set $\mathbf{A}'_2 = [\mathbf{A}^* | -\mathbf{A}^*\mathbf{R}^*_1] \begin{bmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{bmatrix} \cdot \mathbf{R}'_2 + \mathbf{A}_2 = \mathbf{A}'_0 \cdot \mathbf{R}'_2 + \mathbf{A}_2$ so that Equations (1) and (2) match. But in this way it is not clear how to express $\mathbf{A}_2$ as $\mathbf{A}'_0\mathbf{R}$ for some small matrix $\mathbf{R}$, because it is not clear how to express $\mathbf{A}_2$ as $\mathbf{A}'_0\tilde{\mathbf{R}}$ for some small $\tilde{\mathbf{R}}$. Note that $\mathbf{R}$ serves as the secret key of pk_2 to simulate decryption queries. Consequently, it is not clear how the reduction can answer decryption queries as the previous approach. It seems that

this construction/proof is facing a dilemma: either the reduction can answer the decryption queries but the re-encryption key can be distinguished, or the reduction can generate an indistinguishable re-encryption key but cannot answer the decryption queries.

Part II: our new construction for single-hop PRE. To overcome the dilemma, we consider a new matrix structure: the setup algorithm outputs a public matrix $\mathbf{A}$, and each user extends the previous matrix structure to be $\mathbf{A}_u = [\mathbf{A}|\mathbf{A}_1 + \mathbf{H}\mathbf{G}|\mathbf{A}_2 + \mathbf{H}_u\mathbf{G}]$, where $\mathbf{A}_1 = -\mathbf{A}\mathbf{R}_1, \mathbf{A}_2 = -\mathbf{A}\mathbf{R}_2$ and the matrices $\mathbf{R}_1, \mathbf{R}_2$ are the corresponding secret key. The shared matrix $\mathbf{A}$ offers a significant advantage for the simulation: the reduction can embed the LWE instance $\mathbf{A}^*$ as the public shared matrix, and then sets

$$\mathbf{A}'_2 = [\mathbf{A}^*| - \mathbf{A}^*\mathbf{R}^*_1] \begin{bmatrix} \mathbf{X}_{00} \\ \mathbf{X}_{10} \end{bmatrix} \cdot \mathbf{R}'_2 - \mathbf{A}^*\mathbf{R}^*_2.$$

This allows the reduction to express $\mathbf{A}'_2$ as $\mathbf{A}^*\mathbf{R}$ for some small and known matrix $\mathbf{R}$. Then the reduction can use this to simulate the decryption queries, while the Equation (1) will match for the real scheme and the simulated scheme. Our modified construction achieves a relaxed re-encryption functionality in comparison to the construction proposed in [LV08b], i.e. the re-encryption key can only transform well-formed ciphertexts into indistinguishable re-encrypted ciphertexts, but transformation of maliciously chosen cihpertexts can be distinguished if the adversary has the secret key of the target user. In Section 3, we present more detailed discussions and a simple transformation from the relaxed functionality to the "full-fledged" functionality using zero-knowledge proofs[2].

Part III: extension to multi-hop PRE. We further observe that the matrix structure in our construction can be extended to the multi-hop case with a slight modification. Interestingly, our scheme itself can support general network structures (for functionalities), yet our security proof (for CCA security), however, requires the structure of tree-structured networks (i.e. the adversary can only query re-encryption keys that form a tree among the users). If the adversary's queries form a general graph, then security of our scheme becomes unclear: we are not able to prove security under the current techniques, but there is no known attack, either. We leave it as an interesting open problem to determine whether our construction is secure under general network structures.

A technical reason for this phenomenon comes from the order of sampling for the simulation. We give a simple example for illustration: let there be three parties in the network, Users one, two, and three. It is easy for the reduction to simulate in the following order pk_1, $\mathsf{rk}_{1\to2}$, pk_2, $\mathsf{rk}_{2\to3}$, and then pk_3 *without* knowing a trapdoor of the LWE instance $\mathbf{A}^*$. The reduction, however, would get stuck if he needs to further generate $\mathsf{rk}_{1\to3}$, which should be consistent with the already sampled pk_1 and pk_3. We recall that the reduction is able to check whether $\mathsf{rk}_{1\to3}$ is consistent with pk_1 and pk_3 in both the real scheme and the simulated scheme (as Equation (1)). Thus, the reduction must simulate such consistency as the real scheme. Even though there are techniques from the Ring-LWE [LPR10, GGH13] that allows sampling in the *reverse* order of pk_3, $\mathsf{rk}_{2\to3}$, pk_2, $\mathsf{rk}_{1\to2}$, pk_1, it does not help to solve the problem because the reduction still does not know how to generate $\mathsf{rk}_{1\to3}$ after pk_1 and pk_3 are sampled, without a trapdoor of $\mathbf{A}^*$.

Part IV: unified model and construction for multi-hop PRS. An interesting observation from our multi-hop CCA-PRE construction is that it is also compatible with the lattice signature structure in the work of Boyen [Boy10]. In particular, in that work, the signature scheme has the following structure: $[\mathbf{A}|\mathbf{B}_\mu]$, where is an encoded matrix for message μ. This message-dependent

[2]Under current techniques, zero knowledge proof systems based on pure lattices assumptions either require interactions or random oracles.

matrix can be extended to a similar structure similar to that in multi-hop PRE construction. Recall that prior PRS work [AH05, LV08a] consider four scenarios for the security requirement. In each scenario, the adversary has access to a subset of oracles (signing, re-signing, re-key generation), and security requires that the adversary cannot forge a signature on behalf of honest users (whose secret keys are not at the adversary's hand). Our unified security model is based on the approach of multi-hop PRE model with necessary modifications to fit into the signature framework.

1.2 Related Works

Proxy re-encryption. As mentioned above, in recent years, there has been multiple PRE constructions achieving different security notions from different assumptions. In addition to the bi-directional PRE-CPA constructions [BBS98, CH07], there is also some work [AFGH05, HRsV07] about building uni-directional PRE-CPA from various assumptions. For CCA-PRE construction, we only know how to construct single-hop scheme from bilinear group assumption as shown in work [LV08b], and single-hop scheme from LWE assumption in the random oracle model as shown in [ABPW13]. Besides the above mentioned work, recently Nuñez et al. [NAL15] proposed a nice framework capturing more fine-grained CCA-security of PRE, corresponding to the adversary's ability in the security experiment. Our multi-hop tag-based CCA-secure PRE construction described in Section 5 can be categorized as $\mathsf{CCA}_{1,2}$ model in their paper regarding a special structure (trees).

Proxy re-signature. Bi-directional PRS was considered in the literature [AH05, CP08]. The generation of re-key algorithm needs to take inputs both users' secret key. The more fine-grained notion, uni-directional PRS scheme was proposed in [LV08a]. Shao et al. [SFZ+10] cooked up a bilinear group based scheme (in random orcale model) that is insecure but proven secure in prior PRS model [AH05, LV08a], but their result cannot be extended to the lattice setting.

2 Preliminaries

Notations. Let PPT denote probabilistic polynomial time. We use bold uppercase letters to denote matrices, and bold lowercase letters for vectors. We let λ be the security parameter and $[n]$ denote the set $\{1, ..., n\}$. We use $[\cdot|\cdot]$ to denote the concatenation of vectors or matrices, and use ℓ_∞ norm for the norms of all vectors and matrices used in our paper. We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we $\mathsf{negl}(n)$ to denote a negligible function of n. Let X and Y be two random variables taking values in Ω. Define the statistical distance, denoted as $\Delta(X, Y)$ as

$$\Delta(X, Y) := \frac{1}{2} \sum_{s \in \Omega} |\mathbf{Pr}[X = s] - \mathbf{Pr}[Y = s]|$$

Let $X(\lambda)$ and $Y(\lambda)$ be ensembles of random variables. We say that X and Y are statistically close if $d(\lambda) := \Delta(X(\lambda), Y(\lambda))$ is a negligible function of λ. We say two ensembles $X(\lambda)$ and $Y(\lambda)$ are computationally indistinguishable (denoted as $X(\lambda) \approx Y(\lambda)$) if for every PPT distinguisher D, it holds that

$$|\mathbf{Pr}[D(X(\lambda)) = 1] - \mathbf{Pr}[D(Y(\lambda)) = 1]| = \mathsf{negl}(\lambda)$$

2.1 Lattice Background

A full-rank m-dimensional integer lattice $\Lambda \subset \mathbb{Z}^m$ is a discrete additive subgroup whose linear span is $\mathbb{R}^m$. Let Λ be a discrete subset of $\mathbb{Z}^m$. For any vector $\boldsymbol{c} \in \mathbb{R}^m$, and any positive parameter $\sigma \in \mathbb{R}$,

let $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \exp(-\pi||\boldsymbol{x}-\boldsymbol{c}||^2/\sigma^2)$ be the Gaussian function on $\mathbb{R}^m$ with center $\boldsymbol{c}$ and parameter σ. Next, we set $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{x}\in\Lambda}\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{x}}$ over Λ and $\mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) := \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$. Let S^m denote the set of vectors in $\mathbb{R}^m$ whose length is 1. Then the norm of a matrix $\mathbf{R}\in\mathbb{R}^{m\times m}$ is defined to be $\mathsf{sup}_{\boldsymbol{x}\in S^m}||\mathbf{R}\boldsymbol{x}||$. We have the following lemma, which bounds the norm for some specified distributions.

Lemma 2.1 ([ABB10]). *Regarding the norm defined above, we have the following bounds:*

- *Let* $\mathbf{R}\in\{-1,1\}^{m\times m}$ *be chosen at random, then* $\mathbf{Pr}[||\mathbf{R}|| > 12\sqrt{2m}] < e^{-2m}$.
- *Let* $\mathbf{R}$ *be sampled from* $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$, *then we have* $\mathbf{Pr}[||\mathbf{R}|| > \sigma\sqrt{m}] < e^{-2m}$.

Randomness extraction. We will use the following lemma to argue the indistinghishability of two different distributions, which is a generalization of the leftover hash lemma proposed by Dodis et al. [DRS04].

Lemma 2.2 ([ABB10]). *Suppose that* $m > (n+1)\log q + w(\log n)$. *Let* $\mathbf{R}\in\{-1,1\}^{m\times k}$ *be chosen uniformly at random for some polynomial* $k = k(n)$. *Let* $\mathbf{A},\mathbf{B}$ *be matrix chosen randomly from* $\mathbb{Z}_q^{n\times m},\mathbb{Z}_q^{n\times k}$ *respectively. Then, for all vectors* $\boldsymbol{w}\in\mathbb{Z}^m$, *the two following distributions are statistically close:*

$$(\mathbf{A},\mathbf{AR},\mathbf{R}^\mathsf{T}\boldsymbol{w}) \approx (\mathbf{A},\mathbf{B},\mathbf{R}^\mathsf{T}\boldsymbol{w})$$

Learning With Errors. The LWE problem was introduced by Regev [Reg05], who showed that solving it *on the average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*, when the error distribution is instantiated as discrete Gaussian distribution with proper parameters.

Definition 2.3 (LWE). *For an integer* $q = q(n) \geq 2$, *and an error distribution* $\chi = \chi(n)$ *over* $\mathbb{Z}_q$, *the learning with errors problem* $\mathsf{LWE}_{n,m,q,\chi}$ *is to distinguish between the following pairs of distributions:*

$$\{\mathbf{A},\mathbf{A}^\mathsf{T}\boldsymbol{s}+\boldsymbol{x}\} \text{ and } \{\mathbf{A},\boldsymbol{u}\}$$

where $\mathbf{A}\xleftarrow{\$}\mathbb{Z}_q^{n\times m}$, $\boldsymbol{s}\xleftarrow{\$}\mathbb{Z}_q^n$, $\boldsymbol{u}\xleftarrow{\$}\mathbb{Z}_q^m$, *and* $\boldsymbol{x}\leftarrow\chi^n$.

Small Integer Solution. The SIS problem was first suggested to be hard on average by Ajtai [Ajt99] and then formalized by Micciancio and Regev [MR04]. It is known to be as hard as certain worst-case problems (e.g., SIVP) in standard lattices[Ajt99, MR04, GPV08, MP13].

Definition 2.4 (SIS). *For any* $n\in\mathbb{Z}$, *and any functions* $m = m(n), q = q(n), \beta = \beta(n)$, *the average-case Small Integer Solution problem* ($\mathsf{SIS}_{q,n,m,\beta}$) *is: Given an integer* q, *a matrix* $\mathbf{A}\in\mathbb{Z}_q^{n\times m}$ *chosen uniformly at random and a real* $\beta\in\mathbb{R}$, *find a non-zero integer vector* $\boldsymbol{z}\in\mathbb{Z}^m - \{\mathbf{0}\}$, *such that* $\mathbf{A}\boldsymbol{z} = 0 \mod q$ *and* $||\boldsymbol{z}|| \leq \beta$.

G-trapdoors and sampling algorithms. We briefly describe the main results in [MP12]: the definition of $\mathbf{G}$-trapdoor and the algorithms $\mathsf{Invert}^{\mathcal{O}}$ and $\mathsf{Sample}^{\mathcal{O}}$. Roughly speaking, a $\mathbf{G}$-trapdoor is a transformation, represented by a matrix $\mathbf{R}$ from a public matrix $\mathbf{A}$ to a special matrix $\mathbf{G}$. The formal definition is as follows:

Definition 2.5 ([MP12]). *Let* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ *and* $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$ *be matrices with* $m \geq w \geq n$. *A* $\mathbf{G}$*-trapdoor for* $\mathbf{A}$ *is a matrix* $\mathbf{R} \in \mathbb{Z}^{m-w} \times w$ *such that* $\mathbf{A}\left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{HG}$ *for some invertible matrix* $\mathbf{H} \in \mathbb{Z}_q^{n\times n}$. *We refer to* $\mathbf{H}$ *as the tag or label of the trapdoor. The quality of the trapdoor is measured by its largest singular value* $s_1(\mathbf{R})$.

In order to embed matrix $\mathbf{G}$ into a uniformly looking matrix $\mathbf{A}$ together with a transformation $\mathbf{R}$, we should start with a uniform matrix $\mathbf{A}_0$ and a matrix $\mathbf{R}$, and construct $\mathbf{A} = [\mathbf{A}_0|-\mathbf{A}_0\mathbf{R}+\mathbf{HG}]$. For an appropriate chosen dimensions $(\mathbf{A}, \mathbf{AR})$ is negligible from uniformly random distribution by the Lattice-based Leftover Hash Lemma.

Following the work of Micciancio and Peikert [MP12], our scheme uses a special collection of elements defined over ring $\mathcal{R} = \mathbb{Z}_q[x]/(f(x))$, where $f(x) = x^n + f_{n-1}x^{n-1} + \cdots + f_0$ is a irreducible modulo every p dividing q. Since $\mathcal{R}$ is a free $\mathbb{Z}_q$-module of rank n, thus elements of $\mathcal{R}$ can be represented as vectors in $\mathbb{Z}_q$ relative to standard basis of monomials $1, x, ..., x^{n-1}$. Multiplication by any fixed element of $\mathcal{R}$ then acts as a linear transformation on $\mathbb{Z}_q^n$ according to the rule

$$x \cdot (a_0, ..., a_{n-1})^\mathsf{T} = (0, a_0, ..., a_{n-2})^\mathsf{T} - a_{n-1}(f_0, f_1, ..., f_{n-1})^\mathsf{T}$$

and so can be represented by an matrix in $\mathbb{Z}_q^{n\times n}$ relative to the standard basis. In other words, there is an injective ring homomorphism $h : \mathcal{R} \to \mathbb{Z}_q^{n\times n}$ that maps any $a \in \mathcal{R}$ to matrix $\mathbf{H} = h(a)$ representing multiplication by a. As introduced in [MP12], we need a very large set $\mathcal{U} = \{u_1, ..., u_l\}$ with the "unit differences" property: for any $i \neq j$, the difference $u_i - u_j \in \mathcal{R}^*$, and hence $h(u_i - u_j) = h(u_i) - h(u_j) \in \mathbb{Z}_q^{n\times n}$ is invertible.

Lemma 2.6 ([MP12]). *There is an efficient algorithm* $\mathsf{Sample}^{\mathcal{O}}(\mathbf{R}, \mathbf{A}', \mathbf{H}, \boldsymbol{u}, s)$, *where* $\mathbf{R}$ *is a* $\mathbf{G}$*-trapdoor for matrix* $\mathbf{A}$ *with invertible tag* $\mathbf{H}$, *a vector* $\boldsymbol{u} \in \mathbb{Z}^n$ *and an oracle* $\mathcal{O}$ *for Gaussian sampling over a desired coset* $\Lambda_q^{\boldsymbol{v}}(\mathbf{G})$. *It will output a vector drawn from a distribution within negligible statistical distance of* $\mathcal{D}_{\Lambda^{\boldsymbol{u}}(\mathbf{A}),s}$, *where* $\mathbf{A} = [\mathbf{A}'| - \mathbf{A}'\mathbf{R} + \mathbf{HG}]$.

In the following, we provide two extensions of the LWE inversion algorithms proposed by Micciancio and Peikert [MP12], which would be used in the security proof and scheme respectively.

- $\mathsf{Invert}^{\mathcal{O}}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{A}, \boldsymbol{b})$: On input a vector $\boldsymbol{b} = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T}$, a matrix $\mathbf{A} = [\mathbf{A}_0| - \mathbf{A}_0\mathbf{R}_1 + \mathbf{H}_1\mathbf{G}| - \mathbf{A}_0\mathbf{R}_2 + \mathbf{H}_2\mathbf{G}]$ and its corresponding $\mathbf{G}$-trapdoor $\mathbf{R}_1, \mathbf{R}_2$ with invertible tag $\mathbf{H}_1, \mathbf{H}_2$, the algorithm first computes $\boldsymbol{b}' = \boldsymbol{b}^\mathsf{T}\left[\begin{smallmatrix}\mathbf{R}_1+\mathbf{R}_2\\ \mathbf{I}\\ \mathbf{I}\end{smallmatrix}\right]$, and then run the oracle $\mathcal{O}(\boldsymbol{b}')$ to get $(\boldsymbol{s}', \boldsymbol{e}')$. The algorithm outputs $\boldsymbol{s} = (\mathbf{H}_1 + \mathbf{H}_2)^{-1}\boldsymbol{s}'$ and $\boldsymbol{e} = \boldsymbol{b} - \boldsymbol{s}^\mathsf{T}\mathbf{A}$.
- $\mathsf{Invert}'^{\mathcal{O}}(\mathbf{R}_1, \mathbf{R}_2, \mathbf{A}, \boldsymbol{b})$: On input a vector $\boldsymbol{b} = \boldsymbol{s}^\mathsf{T}\mathbf{A} + \boldsymbol{e}^\mathsf{T}$, a matrix $\mathbf{A} = [\mathbf{A}_0| - \mathbf{A}_0\mathbf{R}_1| - \mathbf{A}_0\mathbf{R}_2 + \mathbf{H}_2\mathbf{G}]$ and its corresponding $\mathbf{G}$-trapdoor $\mathbf{R}_1, \mathbf{R}_2$ with invertible tag $\mathbf{H}_1, \mathbf{H}_2$, the algorithm first computes $\boldsymbol{b}' = \boldsymbol{b}^\mathsf{T}\left[\begin{smallmatrix}\mathbf{R}_1+\mathbf{R}_2\\ \mathbf{I}\\ \mathbf{I}\end{smallmatrix}\right]$, and then run the oracle $\mathcal{O}(\boldsymbol{b}')$ to get $(\boldsymbol{s}', \boldsymbol{e}')$. The algorithm outputs $\boldsymbol{s} = \mathbf{H}_2^{-1}\boldsymbol{s}'$ and $\boldsymbol{e} = \boldsymbol{b} - \boldsymbol{s}^\mathsf{T}\mathbf{A}$.

3 Proxy Re-Encryption: Syntax and Security Definitions

In this section, we first recall the syntax of single-hop PRE [LV08b], and then we define a new variant of CCA-PRE security, i.e. tag-based CCA-PRE security that captures constructions associated with tags. However, for lattice-based constructions, our current technique cannot achieve the full-fledged PRE construction in that the re-encryption algorithm does not provide the full-fledged functionality in that it does not fully implement the regular re-encryption oracle which decrypts first, outputs $\perp$

if the decrypted value is invalid, and outputs a fresh ciphertext of the same message, otherwise. Our re-encryption algorithm guarantees the functionality when the input ciphertexts are well-formed, but if the input ciphertexts are not well-formed, the re-encryption algorithm is not able output $\perp$, yet it can only output re-encrypted ciphertexts that will be decrypted to $\perp$. The difference seems subtle, but it plays an important role in the security we can achieve. We will elaborate on this later.

We remark that our relaxed functionality is not far from the full-fledged functionality if the input-ciphretext provider is required to prove the validity of the ciphertexts. We note that there exists an efficient lattice-based Σ protocol [AJL+12] with interaction, and we can further use the Fiat-Shamir transform [FS87] to achieve a NIZK proof system if a random oracle is assumed. We also note that the full-fledged re-encryption oracle requires the ability to distinguish valid ciphertexts from invalid ciphertexts, which implies a verifiable encryption. As we are not aware of any pure lattice-based verifiable encryption schemes in the plain model, i.e. without a random oracle nor interaction, it remains unclear how to achieve the full functionality of the re-encryption. We leave this as an interesting open problem.

The relaxed the re-encryption algorithm provided meaningful security guarantees and allows a modular design that achieves the full-fledged functionality, e.g., the proxy additionally requests a proof of well-formness of the input ciphertexts. The Σ-proofs in the work [AJL+12] are quite practical, so in practice, our scheme can be combined with either the Σ-proof or the Fiat-Shamir transform to achieve the full-fledged functionality.

3.1 Single-Hop PRE Syntax

We recall the syntax of uni-directional PRE, which can be regarded as a natural extension of bi-directional case defined in [CH07] and later studied in uni-directional scenario by Libert and Vergnaud [LV08b]. The PRE scheme consists a tuple of PPT algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKeyGen}, \mathsf{ReEnc}$), which can be defined as follows:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ generates the public parameters pp.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ generates $(\mathsf{pk}, \mathsf{sk})$ for each user.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}, \mu, i)$ encrypts a message μ at level $i \in \{1, 2\}$. The re-encryption can only operate on ciphertexts that are at level 1.
- $\mu' = \mathsf{Dec}(\mathsf{sk}, (\mathsf{ct}, i))$ decrypts a ciphertext ct.
- $\mathsf{rk}_{i \to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j)$ computes the re-encryption key $\mathsf{rk}_{i \to j}$.
- $(\mathsf{ct}', 2) \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{i \to j}, (\mathsf{ct}, 1))$ computes the re-encrypted ciphertext ct'. If the well-formedness of ciphertext ct is publicly verifiable, the algorithm should output "invalid" when ct is ill-formed.

Correctness. For correctness, we consider two cases for the PRE scheme: one for "fresh" ciphertexts generated by encryption algorithm, and the other for re-encryption ciphertexts generated by the re-encryption algorithm. We say that a single-hop PRE scheme is correct if the following holds.

- For any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, any $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, any message μ and level $i \in \{1, 2\}$, it holds that
$$\mathbf{Pr}[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, \mu, i)) = \mu] = 1 - \mathsf{negl}(\lambda)$$

- For any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, any $(\mathsf{pk}_i, \mathsf{sk}_i), (\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$, any message μ, it holds that

$$\mathbf{Pr}[\mathsf{Dec}(\mathsf{sk}_j, \mathsf{ReEnc}(\mathsf{rk}_{i\to j}, \mathsf{ct})) = \mu] = 1 - \mathsf{negl}(\lambda)$$

  where $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \mu, 1), \mathsf{rk}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j)$.

3.2 Single-Hop PRE Security Definitions

In the security part, we first present the CCA-PRE definition proposed in [LV08b] with minor modifications – in particular the definition of derivative in security model. Next, we describe a weaker security model considered in [Kir14], whose restriction is: the re-encryption queries submitted by the adversary are only allowed between honest users. Then we propose an intermediate model, where the capability of re-encryption oracle is slightly weaker than its counterpart in [LV08b]. Intuitively, we say a ciphertext is well-formed if it is an encryption of a message under the claimed public key. In the re-encryption oracle in [LV08b], the well-formedness of ciphertext is public verifiable, i.e the verification only needs public keys. However, in our intermediate model, the verification needs the assistance of secret keys. Let $\mathcal{A}$ denote any PPT adversary, and Π be a PRE scheme. We define the notion of CCA-secure PRE in the uni-directional setting using the following experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{CCA\text{-}PRE}}(1^\lambda)$, which describes the interaction between several oracles and an adversary $\mathcal{A}$. As we discussed before, we include public parameters pp in each user's public key pk and secret key sk, so we will omit pp in the description for simplicity. The experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{single}}(1^\lambda)$ consists of an execution of $\mathcal{A}$ with the following oracles with detail as follows:

- The challenger runs setup algorithm $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$ and initializes two empty sets $\mathcal{H} = \emptyset, \mathcal{C} = \emptyset$. Then he sends pp to adversary $\mathcal{A}$.
- Proceeding adaptively, adversary $\mathcal{A}$ has access to the following oracles:

  Uncorrupted key generation oracle: Obtain a new key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Send pk_i back to adversary $\mathcal{A}$, set the honest user set $\mathcal{H} = \mathcal{H} \cup \{i\}$ and pass the the tuple $(i, \mathsf{pk}_i, \mathsf{sk}_i)$ to re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ and decryption oracle $\mathcal{O}_{\mathsf{Dec}}$.

  Corrupted key generation oracle: Obtain a new key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$. Send the key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ back to adversary $\mathcal{A}$, set the corrupted user set $\mathcal{C} = \mathcal{C} \cup \{i\}$ and pass the tuple $(i, \mathsf{pk}_i, \mathsf{sk}_i)$ to re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ and decryption oracle $\mathcal{O}_{\mathsf{Dec}}$.

  Re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$: On input an index pair (i, j) from the adversary, if the query (i, j) is made after accessing the challenge oracle, then output $\perp$ if $i = i^*$. Otherwise, do the following:

  - If the pair (i, j) is queried for the first time, the oracle returns a re-encryption key $\mathsf{rk}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j)$;
  - else (the pair (i, j) has been queried before), the oracle returns the re-encryption key $\mathsf{rk}_{i\to j}$.

  Re-encryption oracle $\mathcal{O}_{\mathsf{ReEnc}}$: On input $(i, j, (\mathsf{ct}, k))$, the oracle returns a special symbol $\perp$ if (ct, k) is not a well-formed first level ciphertext, or $j \in \mathcal{C}$ and $(i, \mathsf{ct}) = (i^*, \mathsf{ct}^*)$. Otherwise, it computes re-encrypted ciphertext $\mathsf{ct}' \leftarrow \mathsf{ReEnc}(\mathsf{rk}_{i\to j}, \mathsf{ct})$ and sends back $(\mathsf{ct}', 2)$.

  Decryption oracle $\mathcal{O}_{\mathsf{Dec}}$: On input (i, ct), if $i \notin \mathcal{C} \cup \mathcal{H}$ or ct is not a valid ciphertext, then return a special symbol $\perp$. It also outputs a special symbol $\perp$ if (i, ct) is a *Derivative* (c.f. Definition 3.2) of the challenge pair (i^*, ct^*). Otherwise, it returns $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$ to adversary $\mathcal{A}$.

Challenge oracle: This oracle can be queried only once. On input (i^*, μ_0, μ_1), where $i^* \in \mathcal{H}$ and no re-encryption key from i^* to corrupted users $\mathcal{C}$ has been queried by adversary, the oracle chooses a bit $b \in \{0, 1\}$ and returns $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{i^*}, \mu_b, 1)$ as the challenge ciphertext, and passes i^* to re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$, and (i^*, ct^*) to re-encryption oracle $\mathcal{O}_{\mathsf{ReEnc}}$.

Decision oracle: This oracle can be queried only once. On input b' from adversary $\mathcal{A}$, the oracle outputs 1 if $b' = b$, and 0 otherwise.

The advantage of an adversary in the above experiment $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{single}}(1^\lambda)$ is defined as $|\mathbf{Pr}[b' = b] - \frac{1}{2}|$.

Definition 3.1 (CCA-PRE model)**.** *A uni-directional PRE scheme is CCA-PRE secure if all* PPT *adversaries have at most a negligible advantage in experiment* $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{single}}(1^\lambda)$.

In our PRE construction, every ciphertext is associated with a tag u chosen randomly in the encryption algorithm, thus we call our security model *tag-based* CCA security. In [LV08b], a pair (i, ct) is called *derivative* of the challenge ciphertext pair (i^*, ct^*) if $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}_i) \in \{\mu_0, \mu_1\}$, where $\{\mu_0, \mu_1\}$ are the challenge message pair. We achieve a slightly stronger notion of derivative as defined in the following

Definition 3.2 (Derivative)**.** *A pair* $(i, (\mathsf{ct}, u))$ *is called derivative of the challenge ciphertext pair* $(i^*, (\mathsf{ct}^*, u^*))$ *if* $u = u^*$.

Remark 3.3. It is obvious to see that tag-based CCA security is stronger than CCA1 security (where the adversary cannot access the decryption oracle after the challenge ciphertext), and is slightly weaker than CCA2 security. This relaxation is meaningful and can be nearly the best we can achieve if we further require the property of *unlinkability* for re-encrypted ciphertexts. That is, if we want the re-encrypted algorithm to produce statistically indistinguishable ciphertexts, i.e. the re-encrypted ciphertexts are almost identically distributed as fresh ones, then arguably it is not possible to achieve CCA2 security, because the decryption oracle cannot distinguish a re-encryption of challenge ciphertext from a fresh ciphertext, so an adversary can easily break the security game by querying the decryption oracle with a re-encrypted ciphertext of the challenge ciphertext. For tag-based schemes, where the tag remains the same for re-encrypted ciphertexts, we can ensure that the challenge ciphertext will not be decrypted by the decryption oracle due to derivative definition (c.f. Definition 3.2). The tag-based CCA security guarantees the challenge ciphertext remains hidden, even if the adversary can obtain decryptions of ciphertexts with other tags.

The above security model only captures the CCA security of ciphertexts on the first level. We also present the CCA security of ciphertexts on the second level. Since the challenge ciphertext is on the second level, which means it cannot be further re-encrypted to ciphertext under other public keys, so there is no need to restrict the re-encryption queries regarding the challenge ciphertext. We highlight the difference comparing to security model of first level ciphertexts in the following definition.

Definition 3.4 (Second-level security)**.** *The difference of experiment between second-level security and the security definition in Definition 3.1 are below:*

- *In challenge oracle: the oracle returns* $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_{i^*}, \mu_b, 2)$ *as the challenge ciphertext.*
- *The re-encryption oracle* $\mathcal{O}_{\mathsf{ReEnc}}$ *does not need to check whether the queried tuple is the same as challenge ciphertext.*

Definition 3.5 (PRE with Relaxed Functionality)**.** *A PRE scheme with a relaxed functionality if the re-encryption algorithm outputs statistically close to the distribution of fresh ciphertexts of the second level when the input ciphertexts are well-formed. That is, if* $(\mathsf{ct}, 1)$ *is a well-formed ciphertext, then* $\mathsf{ReEnc}(\mathsf{rk}_{i\to j}, (\mathsf{ct}, 1))$ *is statistically close to* $(\mathsf{ct}', 2) \leftarrow \mathsf{Enc}(\mathsf{pk}_j, \mathsf{Dec}(\mathsf{ct}, 1), 2)$*. If the input ciphertexts are not well-formed, then only* $\mathsf{Dec}(\mathsf{sk}_j, (\mathsf{ct}', 2)) = \perp$ *is guaranteed.*

Remark 3.6. *As we argued above, the relaxed functionality does not completely implement the re-encryption oracle* $\mathcal{O}_{\mathsf{ReEnc}}$ *as in the above definition. The difference can be bridged by a crypto proof system, (either interactively or non-interactively) assuming the input ciphertext is associated with a proof. We present the formal description of this idea in the full version of this paper.*

In our construction, we do not allow querying the relaxed functionality directly with arbitrary input ciphertexts re-encrypted to a corrupted party, e.g., invalid input ciphertexts chosen by the adversary to some corrupted Party j*. As the transformation can leak the re-encryption key, if the adversary corrupts Party* j *and can obtain a re-encryption key* $\mathsf{rk}_{i\to j}$*, then he can easily break the security of* pk_i*.*

We note that the the CCA model of [Kir14] is weaker than the model considered in this paper. In particular, the model [Kir14] has the following restrictions: the re-encryption key queries (or re-encryption queries) submit by adversary $\mathcal{A}$ are restricted among honest users (we ignore the re-encryption queries within corrupted users, since adversary can generate by himself).

3.3 Multi-Hop Proxy Re-Encryption

As mentioned in the introduction, the security model of multi-hop is defined with respect to a tree committed by the adversary before obtaining the public parameters. Let $L = L(\lambda)$ denotes the maximum level the system supports. A multi-hop PRE scheme consists of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKeyGen}, \mathsf{ReEnc})$. In comparison with single-hop PRE, we only emphasize the difference in algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{ReKeyGen})$.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$ generates public parameters pp.
- $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{pk}_i, \mu, \kappa)$ computes a ciphertext ct for message μ at level κ. The re-encryption can only operate on ciphertexts at level $\kappa \in [L-1]$.
- $\mathsf{rk}^\kappa_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j, \kappa)$ computes a re-encryption key from level κ of public key pk_i to level $\kappa + 1$ of public key pk_j as $\mathsf{rk}^\kappa_{i\to j}$.

Correctness. The correctness of multi-hop PRE can be defined similarly as single-hop PRE, i.e. by requiring successful decryption for "fresh" ciphertexts and re-encrypted ciphertext.

Security definitions. Next, we present the security definitions of multi-hop PRE. We first define an experiment $\mathbf{Expt}^{\mathsf{multi}}_{\mathcal{A}}(1^\lambda)$ between a challenger associated with several oracles and an adversary $\mathcal{A}$:

- In the beginning of the experiment, adversary $\mathcal{A}$ commits to challenger a tree structure $T = (V, E)$ and target level κ^*, where the root r of the tree serves as the target user. The challenger first runs a Deep-First Search to associated each edge with a level index $\ell(i)$, by increasing from root user's target level κ^*, and updates the edge set by augmenting each pair (i, j) with its level index $(i, j, \ell(i))$. The challenger then runs $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$; for every user $i \in V$, the challenger computes $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ and for every $(i, j, \kappa) \in E$, the challenger computes

re-encryption keys $\{\mathsf{rk}^{\kappa}_{i\to j}\}$. He then sets $\mathcal{H} = V$, $K = \{\mathsf{rk}^{\kappa}_{i\to j}\}_{(i,j,\kappa)\in E}$. Additionally, the challenger maintains a graph $G = (V', E')$ initially set to $V' = V$, $E' = E$. The challenger then sends $(\mathsf{pp}, \{\mathsf{pk}_i\}_{i\in V})$ to adversary $\mathcal{A}$.

- Adversary $\mathcal{A}$ has access to the following oracles adaptively:

  Honest key generation oracle: Obtain a new key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for a new index i. The oracle updates the honest users set $\mathcal{H} = \mathcal{H} \cup \{i\}$ and $V' = V' \cup \{i\}$, and then sends pk_i to adversary $\mathcal{A}$.

  Corrupt key generation oracle: Obtain a new key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for a new index i. The oracle updates the corrupted users set $\mathcal{C} = \mathcal{C} \cup \{i\}$ and $V' = V' \cup \{i\}$, and sends $(\mathsf{pk}_i, \mathsf{sk}_i)$ to adversary $\mathcal{A}$.

  Decryption oracle $\mathcal{O}_{\mathsf{Dec}}$: On input $(i, (\mathsf{ct}, \kappa))$, if $i \notin V'$ or $\kappa \leq L$ or ct is not a valid ciphertext, then return a special symbol $\perp$. It also outputs a special symbol $\perp$ if (i, ct) is a Derivative of the challenge pair (i^*, ct^*), which means $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}_i) \in \{\mu_0, \mu_1\}$. Otherwise, it returns $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$ to adversary $\mathcal{A}$.

  Re-encryption oracle $\mathcal{O}_{\mathsf{ReEnc}}$: On input $(i, j, (\mathsf{ct}, \kappa))$, the oracle outputs $\perp$ if the ciphertext ct is not well formed. Otherwise,

  - If the edge $(i, j, \kappa) \in E'$, then find re-encryption key $\mathsf{rk}^{\kappa}_{i\to j}$, and computes $\mathsf{ct}' \leftarrow \mathsf{ReEnc}(\mathsf{rk}^{\kappa}_{i\to j}, \mathsf{ct})$. Then send back $(\mathsf{ct}', \kappa + 1)$ to adversary $\mathcal{A}$.
  - If edge $(i, j, \kappa) \notin E', j \in V$ and $\kappa = \ell(j) - 1$, then output $\perp$.
  - Otherwise, call re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ with input (i, j, κ) to obtain re-encryption key $\mathsf{rk}^{\kappa}_{i\to j}$, and send back $\sigma' \leftarrow \mathsf{ReEnc}(\mathsf{rk}^{\kappa}_{i\to j}, \sigma)$ to adversary.

  Re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$: On input user index i, j and level index κ, the oracle outputs $\perp$ if $j \in \mathcal{C}$ and the edge (i, j, κ) is not an *admissible edge* (c.f. Definition 3.7) with respect to edge set E and target user and level (r, κ^*). Otherwise,

  - If $(i, j, \kappa) \in E'$, then search for the re-encryption key $\mathsf{rk}^{\kappa}_{i\to j}$ in set K, and send back $\mathsf{rk}^{\kappa}_{i\to j}$ to adversary $\mathcal{A}$.
  - Otherwise, compute and send back $\mathsf{rk}^{\kappa}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j, \kappa)$ to adversary $\mathcal{A}$, then updates set $E' = E' \cup (i, j, \kappa), K = K \cup \{\mathsf{rk}^{\kappa}_{i\to j})\}$.

  Challenge oracle: This oracle can be queried only once. On input (μ_0, μ_1), the oracle chooses a bit $b \in \{0, 1\}$ and returns $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}_r, \mu_b, \kappa^*)$ as the challenge ciphertext.

  Decision oracle: This oracle can be queried only once. On input b' from adversary $\mathcal{A}$, the oracle outputs 1 if $b' = b$, and 0 otherwise.

The advantage of an adversary in the above experiment $\mathbf{Expt}^{\mathsf{multi}}_{\mathcal{A}}(1^\lambda)$ is defined as $|\mathbf{Pr}[b' = b] - \frac{1}{2}|$.

Definition 3.7 (Admissible edge)**.** *We call edge (i, j, κ) a legal edge regarding edge set E and vertex/level pair (t, κ^*), if there dose not exist a path $\{(t, s, \kappa^*), (s, r, \kappa^* + 1), ..., (i, j, \kappa)\} \in E \cup \{(i, j, \kappa)\}$.*

Definition 3.8 (Multi-hop uni-directional CCA-PRE)**.** *A L-hop uni-directional PRE scheme is CCA-PRE secure if all* PPT *adversaries have at most a negligible advantage in experiment $\mathbf{Expt}^{\mathsf{multi}}_{\mathcal{A}}(1^\lambda)$.*

Remark 3.9. For our multi-hop model and construction, we only consider the intermediate model, i.e. the verification of well formedness of queried ciphertext to re-encryption needs the assistance of secret keys. The generic transformation from intermediate security model to the model in [LV08b] dose not work for multi-hop scenario. Therefore, we leave it as an interesting open problem.

The following Figure 1 provides the idea of admissible edges: suppose adversary $\mathcal{A}$ first commits the tree $T = (V, E)$ as in Figure 1 and target level κ^*, where $V = \{r, i_1, i_2, i_3, i_4, i_5\}$ and edge set E as in the figure. The challenger then runs a deep first traversal to associate each edge with its starting level, e.g. $(i_1, r, \ell(i_1) = \kappa^* + 1)$, and updates the edge set E correspondingly. Adversary can later request a corrupt user, named user j, and ask the re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ to generate re-encryption keys for $(j, i_2, \kappa^* + 2)$ (the red dashed line in Figure 1) and (k, i_3, κ) (the green dashed line in Figure 1), where $\kappa \neq \kappa^* + 3$. For edge $(j, i_2, \kappa^* + 2)$, since there is a path, i.e. $\{(r, i_2, \kappa^* + 1), (i_2, j, \kappa^* + 2)\}$ from the target level of root user (r, κ^*), thus the edge $(j, i_2, \kappa^* + 2)$ is not admissible, which means $\mathcal{O}_{\mathsf{ReKeyGen}}$ would output $\perp$. However, since edge (k, i_3, κ) is admissible, oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ would output re-encryption key $\mathsf{rk}^{\kappa}_{k \to i_3}$ to adversary $\mathcal{A}$.

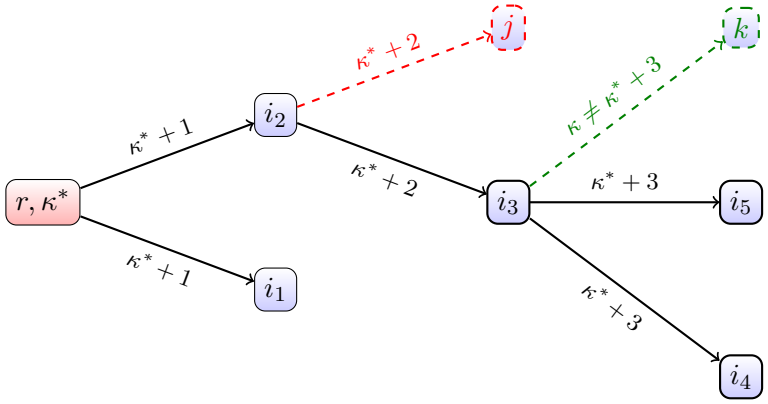

Figure 1: Illustration of multi-hop PRE security model

4 Single-hop Tag-based CCA-Secure PRE Construction

In this section, we present our construction of single-hop PRE. The PRE system has message space $\{0,1\}^{nk}$, which we map bijectively to the cosets of $\Lambda/2\Lambda$ for $\Lambda = \Lambda(\mathbf{G}^t)$ via some encoding function encode that is efficient to evaluate and invert. In particular, letting $\mathbf{S} \in \mathbb{Z}^{nk \times nk}$ be any basis of Λ, we can map $\boldsymbol{\mu} \in \{0,1\}^{nk}$ to $\mathsf{encode}(\boldsymbol{\mu}) = \mathbf{S}\boldsymbol{\mu} \in \mathbb{Z}^{nk}$. The PRE scheme $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{ReKeyGen}, \mathsf{ReEnc})$ can be described as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: The global setup algorithm set the lattice parameter (n, k, q, s). Then it randomly selects a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times nk}$, and outputs the public parameter $\mathsf{pp} = (\mathbf{A}, n, m, q, s)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: The key generation algorithm for i-th user chooses random matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2} \leftarrow \mathcal{D}_{\mathbb{Z}^{nk \times nk}, s}$, letting $\mathbf{A}_{i1} = \mathbf{A}\mathbf{R}_{i1} \bmod q$ and $\mathbf{A}_{i2} = \mathbf{A}\mathbf{R}_{i2} \bmod q$. The public key is $\mathsf{pk}_i = \mathbf{A}_i = [\mathbf{A}| - \mathbf{A}_{i1}| - \mathbf{A}_{i2}]$, and the secret key is $\mathsf{sk}_i = [\mathbf{R}_{i1}|\mathbf{R}_{i2}]$.
- $\mathsf{Enc}(\mathsf{pk}_i, \boldsymbol{\mu}, \ell)$: The encryption algorithm does
  - If $\ell = 1$, choose non-zero $u \leftarrow \mathcal{U}$ and let the message/level-dependent matrix
  $$\mathbf{A}_{i,u,l} = [\mathbf{A}| - \mathbf{A}_{i1} + h(\ell)\mathbf{G}| - \mathbf{A}_{i2} + h(u)\mathbf{G}]$$
  Choose $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n, \boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \mathcal{D}_{\mathbb{Z},s}^{nk}$. Let
  $$\boldsymbol{b}^\mathsf{T} = (\boldsymbol{b}_0, \boldsymbol{b}_1, \boldsymbol{b}_2) = 2(\boldsymbol{s}^\mathsf{T}\mathbf{A}_{i,u,\ell} \bmod q) + \boldsymbol{e}^\mathsf{T} + (0, 0, \mathsf{encode}(\boldsymbol{\mu})^\mathsf{T}) \bmod 2q$$
  where $\boldsymbol{e} = (\boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2)$. Output the ciphertext $\mathsf{ct} = (u, \boldsymbol{b}, 1)$.

– If $\ell = 2$, the algorithm uses the same procedure to encrypt the message, except it chooses error $\boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2 \leftarrow \mathcal{D}^{nk}_{\mathbb{Z},s'}$, and outputs $\mathsf{ct} = (u, \boldsymbol{b}, 2)$.

- $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$: The decryption algorithm

  1. If ct does not parse or $u = 0$, output $\perp$. Otherwise, reconstruct the message/level-dependent matrix $\mathbf{A}_{i,u,\ell}$
  $$\mathbf{A}_{i,u,l} = [\mathbf{A}| - \mathbf{A}_{i1} + h(\ell)\mathbf{G}| - \mathbf{A}_{i2} + h(u)\mathbf{G}]$$
  Call $\mathsf{Invert}^{\mathcal{O}}([\mathbf{R}_{i1}|\mathbf{R}_{i2}], \mathbf{A}_u, \boldsymbol{b} \bmod q)$ to get values $\boldsymbol{z} \in \mathbb{Z}_q^n$ and $\boldsymbol{e} = (\boldsymbol{e}_0, \boldsymbol{e}_1, \boldsymbol{e}_2)$ for which $\boldsymbol{b} = \boldsymbol{z} + \boldsymbol{e} \bmod q$. If the algorithm Invert fail for any reason, output $\perp$.
  2. Check the length of the obtained error vectors.
  3. Let $\boldsymbol{v} = \boldsymbol{b} - \boldsymbol{e}$, and parse $\boldsymbol{v} = (\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2)$. If $\boldsymbol{v}_0 \notin 2\Lambda(\mathbf{A}^{\mathsf{T}})$, output $\perp$. Finally, output
  $$\mathsf{encode}^{-1}(\boldsymbol{v}^{\mathsf{T}} \begin{bmatrix} \mathbf{R}_{i1} & \mathbf{R}_{i2} \\ \mathbf{I} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \bmod 2q) \in \{0,1\}^{nk}$$
  if it exists, otherwise output $\perp$.

- $\mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j)$: The re-encryption key generation algorithm does:

  1. Use $\mathsf{sk}_i = [\mathbf{R}_{i1}|\mathbf{R}_{i2}]$ to run extended sampling algorithm $\mathsf{Sample}^{\mathcal{O}}$ to sample $\mathbf{X}_{01}, \mathbf{X}_{02}, \mathbf{X}_{11}, \mathbf{X}_{12} \in \mathbb{Z}^{nk \times nk}$ such that
  $$[\mathbf{A}| - \mathbf{A}_{i1} + h(1)\mathbf{G}| - \mathbf{A}_{i2} + \mathbf{B}] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{01} & \mathbf{X}_{02} \\ 0 & \mathbf{X}_{11} & \mathbf{X}_{12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}| - \mathbf{A}_{j1} + h(2)\mathbf{G}| - \mathbf{A}_{j2} + \mathbf{B}]$$
  for any matrix $\mathbf{B} \in \mathbb{Z}^{n \times nk}$.
  2. Output the re-encryption key
  $$\mathsf{rk}_{i \to j} = \{\mathbf{X}_{01}, \mathbf{X}_{02}, \mathbf{X}_{11}, \mathbf{X}_{12}\}$$

- $\mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct})$: The re-encryption algorithm output a special symbol $\perp$ if $\ell = 2$. Otherwise, it computes
$$\boldsymbol{b}^{\mathsf{T}} \cdot \mathsf{rk}_{i \to j} = \boldsymbol{s}^{\mathsf{T}}[\mathbf{A}| - \mathbf{A}_{j1} + h(1)\mathbf{G}| - \mathbf{A}_{j2} + h(u)\mathbf{G}] + \boldsymbol{e}'^{\mathsf{T}} + \widetilde{\boldsymbol{e}^{\mathsf{T}}} + (0, 0, \mathsf{encode}(\mu)^{\mathsf{T}})$$
where $\boldsymbol{e}' = (\boldsymbol{e}'_0, \boldsymbol{e}'_1, \boldsymbol{e}'_2), \widetilde{\boldsymbol{e}} \leftarrow \mathcal{D}_{\mathbb{Z}^{3nk}, s'}$, and
$$\boldsymbol{e}'_0 = \boldsymbol{e}_0, \qquad \boldsymbol{e}'_1 = \boldsymbol{e}_0\mathbf{X}_{01} + \boldsymbol{e}_1\mathbf{X}_{11}, \qquad \boldsymbol{e}'_2 = \boldsymbol{e}_0\mathbf{X}_{02} + \boldsymbol{e}_1\mathbf{X}_{12} + \boldsymbol{e}_2$$
Then, it outputs $\mathsf{ct}' = (u, \boldsymbol{b}', 2)$.

Correctness. We show that our construction, with appropriate parameter setting specified in Section 6.5, satisfies the correctness condition defined above.

Lemma 4.1 (Correctness)**.** *Let* $(\mathbf{A}_i, (\mathbf{R}_{i1}, \mathbf{R}_{i2}))$ *and* $(\mathbf{A}_j, (\mathbf{R}_{j1}, \mathbf{R}_{j2}))$ *be the public/secret key pair for* i, j*-th user respectively in the PRE system. Let* $\mathsf{ct} = (u, \boldsymbol{b}, 1)$ *be the ciphertext of plaintext* $\boldsymbol{\mu}$ *for* i*-th user, and* $\mathsf{ct}' = \mathsf{ReEnc}(\mathsf{rk}_{i \to j}, \mathsf{ct})$ *be the re-encrypted ciphertext for* j*-th user. Then as we require above, it holds* $\mu \leftarrow \mathsf{Dec}(\mathsf{ct}', \mathsf{sk}_j)$.

Proof. Parse ciphertext ct as $\mathsf{ct} = (u, \boldsymbol{b}, 1)$. Per correctness, we have

$$\boldsymbol{b}^\mathsf{T} = 2(\boldsymbol{s}^\mathsf{T}\mathbf{A}_{i,u,1} \bmod q) + \boldsymbol{e}^\mathsf{T} + (0, 0, \mathsf{encode}(\boldsymbol{\mu})^\mathsf{T}) \bmod 2q$$

where $\mathbf{A}_{i,u,1} = [\mathbf{A}| - \mathbf{A}_{i1} + h(1)\mathbf{G}| - \mathbf{A}_{i2} + h(u)\mathbf{G}]$. The re-encryption process can be re-phrased as follows:

$$\begin{aligned}
&\boldsymbol{s}^\mathsf{T}[\mathbf{A}| - \mathbf{A}_{i1} + h(1)\mathbf{G}| - \mathbf{A}_{i2} + h(u)\mathbf{G}] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{01} & \mathbf{X}_{02} \\ 0 & \mathbf{X}_{11} & \mathbf{X}_{12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} + \ \mathsf{noise} \ + (0, 0, \mathsf{encode}(\boldsymbol{\mu})^\mathsf{T}) \\
&= \boldsymbol{s}^\mathsf{T}[\mathbf{A}| - \mathbf{A}_{j1} + h(2)\mathbf{G}| - \mathbf{A}_{j2} + h(u)\mathbf{G}] + \ \mathsf{noise} \ + (0, 0, \mathsf{encode}(\boldsymbol{\mu})^\mathsf{T}) \\
&= \boldsymbol{s}^\mathsf{T}\mathbf{A}_{j,u,2} + \ \mathsf{noise} \ + (0, 0, \mathsf{encode}(\boldsymbol{\mu})^\mathsf{T})
\end{aligned}$$

where the noise terms is defined in algorithm ReEnc. It is obvious that the re-encrypted ciphertext can be decrypted using j-th secret key, thus we omit the detail for decryption here. □

4.1 Security Proof

We follow the intuition explained in the introduction part to prove security.

Theorem 4.2. *Assuming the hardness of* $\mathsf{LWE}_{q,\alpha'}$ *($\alpha' = \alpha/3 \geq 2\sqrt{n}/q$), the proxy re-encryption scheme is CCA-PRE secure as defined in Definition 3.1.*

Proof. First, using the same technique in [MP12], we can transform the samples from LWE distribution to what we will need below. Given access to an LWE distribution $A_{\boldsymbol{s},\alpha'}$ over $\mathbb{Z}_q \times \mathbb{T}$ (where $\mathbb{T} = \mathbb{R}/\mathbb{Z}$), we can transform its samples $(\boldsymbol{a}, b = \langle \boldsymbol{s}, \boldsymbol{a} \rangle/q + e \bmod 1)$ to have the form $(\boldsymbol{a}, 2(\langle \boldsymbol{s}, \boldsymbol{\alpha} \rangle \bmod q) + e' \bmod 2q)$ for $e' \leftarrow D_{\mathbb{Z},\alpha q}$, by mapping b to $2qb + D_{\mathbb{Z}-2qb,s} \bmod 2q$ where $s^2 = (\alpha q)^2 - (2\alpha' q)^2 \geq 4n \geq \eta_\epsilon(\mathbb{Z})^2$. The transformation maps the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{T}$ to the uniform distribution over $\mathbb{Z}_q^n \times \mathbb{Z}_{2q}$. Once the LWE samples are of the desired form, we construct column-wise matrix $\mathbf{A}^*$ from these samples $\boldsymbol{a}$ and a vector $\boldsymbol{b}^*$ from the corresponding b. Without loss of generality, we randomly choose one uncorrputed user as the challenge user, which will result in a polynomial loss in the security proof. We proceed via a sequence of hybrid games:

Hybrid H_0 : The game H_0 is exactly the CCA attack with the real system described above.

Hybrid H_1 : In game H_1, we change the way to generate uncorrupted pk, challenge ciphertext ct^* and re-encryption keys rk, that are hard to distinguish from the counterparts in game H_0. We set the public parameter matrix $\mathbf{A} = \mathbf{A}^*$, where $\mathbf{A}^*$ is from LWE instance $(\mathbf{A}^*, \boldsymbol{b}^*)$, and select a random element $u^* \in \mathcal{U}$. We initialize two empty sets $\mathcal{H}$ and $\mathcal{C}$.

- **Uncorrupted key generation oracle**: To obtain a public key for uncorrupted user of the challenge user $i^* \in \mathcal{H}$, the oracle chooses random matrices $\mathbf{R}_{i^*1}, \mathbf{R}_{i^*2}$ from $\{-1, 1\}^{m \times m}$, then output the public key to be

  $$\mathsf{pk}_{i^*} = [\mathbf{A}^*| - \mathbf{A}^*\mathbf{R}_{i^*1} - h(1)\mathbf{G}| - \mathbf{A}^*\mathbf{R}_{i^*2} - h(u^*)\mathbf{G}]$$

  For uncorrupted query $i \in \mathcal{H}$ other than challenge user i^*, the oracle firstly chooses and stores matrices $\mathbf{X}_{i,01}, \mathbf{X}_{i,11}, \mathbf{X}_{i,02}, \mathbf{X}_{i,12}$ from $\mathcal{D}_{\mathbb{Z},s}^{nk \times nk}$, and set

  $$\mathbf{A}_{i1} = [\mathbf{A}| - \mathbf{A}_{i^*1}] \begin{bmatrix} \mathbf{X}_{i,01} \\ \mathbf{X}_{i,11} \end{bmatrix} \qquad \mathbf{A}_{i2} = [\mathbf{A}| - \mathbf{A}_{i^*1}] \begin{bmatrix} \mathbf{X}_{i,02} \\ \mathbf{X}_{i,12} \end{bmatrix}$$

where $\mathbf{A}_{i^*1} = \mathbf{AR}_{i^*1}, \mathbf{A}_{i^*2} = \mathbf{AR}_{i^*2}$. Then the oracle outputs the public key for i-th query as

$$\mathsf{pk}_i = [\mathbf{A}| - \mathbf{A}_{i1} - h(2)\mathbf{G}| - \mathbf{A}_{i2} + \mathbf{A}_{i^*2}]$$

Also since the oracle does not reveal any secret key of uncorrupted users, the output of $\{\mathsf{pk}_i\}_{i\in\mathcal{C}}$ other than the challenge ciphertext does not reveal the choice of u^* as well.

- **Corrupted key generation oracle**: To obtain the pair of secret and public keys for corrupted user i, the oracle chooses random matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2} \leftarrow \mathcal{D}$, letting $\mathbf{A}_{i1} = \mathbf{AR}_{i1} \bmod q$ and $\mathbf{A}_{i2} = \mathbf{AR}_{i2} \bmod q$, sends back $\mathsf{sk}_i = [\mathbf{R}_{i1}|\mathbf{R}_{i2}]$ and sets $\mathcal{C} = \mathcal{C} \bigcup \{i\}$
- **Re-encryption key generation oracle**: On input (i, j), re-encryption oracle outputs $\perp$ if $i = i^*$ and $j \in \mathcal{C}$. Otherwise, since there exist some differences in the key generation process between the first query and the rest queries, we divide the process into two cases as follows. For the cases where $i \in \mathcal{C}$, the adversary can compute the re-encryption key from corrupted users to any user by himself because of the fact that he has the secret key, thus we omit the cases here.
  1. The generation of re-encryption key from challenge user i^* to other user $i \in \mathcal{H}$ from level 1 to level 2: The oracle can use the pre-sampled matrices $\{\mathbf{X}\}$ in the generation of pk_i to re-construct the re-encryption key $\mathsf{rk}_{i^*\to i}$ as:

$$\mathsf{rk}_{i^*\to i} = \begin{bmatrix} \mathbf{I} & \mathbf{X}_{i,01} & \mathbf{X}_{i,02} \\ 0 & \mathbf{X}_{i,11} & \mathbf{X}_{i,12} \\ 0 & 0 & \mathbf{I} \end{bmatrix}$$

     where matrices $\mathbf{X}_{i,01}, \mathbf{X}_{i,11}, \mathbf{X}_{i,02}, \mathbf{X}_{i,12}$ are pre-sampled previously in the key generation oracle. Each entry of the resulting re-encryption key is an inner product of a discrete Gaussian vector and a vector consisting of $\{0, 1\}$, so the simulated re-encryption keys from challenge user to other uncorrupted users have the same distribution as a re-encryption key in the scheme.
  2. The generation of re-encryption key from user $i \in \mathcal{H}$ $(i \neq i^*)$ to user j $(j \in \mathcal{H} \bigcup \mathcal{C})$ from level 1 to level 2: We first reconstruct the level-dependent matrix $\mathbf{A}_i$ for i-th users as:

$$\mathbf{A}_i = [\mathbf{A}| - \mathbf{A}_{i1} + (h(1) - h(2))\mathbf{G}] = [\mathbf{A}^*| - \mathbf{A}^*\mathbf{R}^*_{i1} + (h(1) - h(2))\mathbf{G}]$$

     where $\mathbf{R}^*_{i1} = \mathbf{X}_{i,01} - \mathbf{R}_{i^*1}\mathbf{X}_{i,11}$. We can still compute the re-encryption key matrix using Sample algorithm in the same way as in H_0, since matrix $(h(1) - h(2))\mathbf{G}$ is non-zero in encryption matrix $\mathbf{A}_i$. The generation of re-encryption keys in this case still uses algorithm $\mathsf{Sample}^{\mathcal{O}}$, thus the distribution of simulated re-encryption keys is statistically close to that in the real scheme.
- **Re-encryption oracle**: On input query (i, j, ct) from adversary $\mathcal{A}$, the oracle first parses ciphertext $\mathsf{ct} = (u, \boldsymbol{b}, \ell)$, and outputs $\perp$ if $\ell = 2$ or $u = u^*, j \in \mathcal{C}$ when the re-encryption query is made after the challenge query. The oracle answers the queries in following cases:
  - $i \neq i^*$: In this case, the oracle uses the re-encryption key generated before to compute $\boldsymbol{b}' = \mathsf{ReEnc}(\mathsf{rk}_{i\to j}, b)$ and output $(u, \boldsymbol{b}', 2)$.
  - $i = i^*$: In this case, the oracle first decrypts ciphertext by calling decryption oracle described below on input (i^*, ct) to obtain μ. Then it computes $\boldsymbol{b}' \leftarrow \mathsf{Enc}(\mathsf{pk}_{i^*}, \mu, 2; u)$ and outputs $(u, \boldsymbol{b}', 2)$.
- **Decryption oracle**: On decryption query (i, ct) from adversary $\mathcal{A}$, the oracle first parses ciphertext $\mathsf{ct} = (u, \boldsymbol{b}, \ell)$, and outputs $\perp$ if $u = 0$. If the decryption queries are made after the challenge oracle query, then oracle outputs $\perp$ if $u = u^*$. Then oracle divides the decryption process into following four cases:

1. If $i = i^*$ and $\ell = 1$, oracle reconstructs the message/level-dependent matrix $\mathbf{A}_{i^*,u,1}$ as

$$\mathbf{A}_{i^*,u,1} = [\mathbf{A}^* | - \mathbf{A}^*\mathbf{R}_{i^*1}| - \mathbf{A}^*\mathbf{R}_{i^*2} + (h(u) - h(u^*))\mathbf{G}]$$

   Call $\mathsf{Invert}'^{\mathcal{O}}([\mathbf{R}_{i^*1}|\mathbf{R}_{i^*2}], \mathbf{A}_{i^*,u}, \boldsymbol{b} \bmod q)$ to get some $\boldsymbol{z} \in \mathbb{Z}^n$ and $\boldsymbol{e}$. Then oracle performs step 3 exactly as in Dec, except using $[\mathbf{R}_{i^*1}|\mathbf{R}_{i^*2}]$ to decode message.
2. If $i = i^*$ and $\ell = 2$, oracle reconstructs the message/level-dependent matrix $\mathbf{A}_{i^*,u,2}$ as

$$\mathbf{A}_{i^*,u,1} = [\mathbf{A}^* | - \mathbf{A}^*\mathbf{R}_{i^*1} + (h(2) - h(1))\mathbf{G}| - \mathbf{A}^*\mathbf{R}_{i^*2} + (h(u) - h(u^*))\mathbf{G}]$$

   Call $\mathsf{Invert}^{\mathcal{O}}([\mathbf{R}_{i^*1}|\mathbf{R}_{i^*2}], \mathbf{A}_{i^*,u}, \boldsymbol{b} \bmod q)$ to get some $\boldsymbol{z} \in \mathbb{Z}^n$ and $\boldsymbol{e}$. Then oracle performs step 3 exactly as in Dec, except using $[\mathbf{R}_{i^*1}|\mathbf{R}_{i^*2}]$ to decode message.
3. If $i \neq i^*$ and $\ell = 1$, oracle reconstructs the message/level-dependent matrix $\mathbf{A}_{i,u,1}$ as

$$\begin{aligned}\mathbf{A}_{i,u,1} &= [\mathbf{A}^* | - \mathbf{A}^*\mathbf{R}^*_{i1} + (h(1) - h(2))\mathbf{G}| - \mathbf{A}^*\mathbf{R}^*_{i2} - \mathbf{A}_{i^*3} + (h(u) - h(u^*))\mathbf{G}] \\ &= [\mathbf{A}^* | - \mathbf{A}^*\mathbf{R}^*_{i1} + (h(1) - h(2))\mathbf{G}| - \mathbf{A}^*(\mathbf{R}^*_{i2} + \mathbf{R}_{i^*2}) + (h(u) - h(u^*))\mathbf{G}]\end{aligned}$$

   where $\mathbf{R}^*_{i2} = (\mathbf{X}_{i,02} - \mathbf{R}_{i^*1}\mathbf{X}_{i,12})$. Call algorithm $\mathsf{Invert}^{\mathcal{O}}([\mathbf{R}^*_{i1}|\mathbf{R}^*_{i2} + \mathbf{R}_{i^*2}], \mathbf{A}_{i,u},$ $\boldsymbol{b} \bmod q)$ to get some $\boldsymbol{z} \in \mathbb{Z}^n$ and $\boldsymbol{e}$. Then oracle performs step 3 exactly as in Dec, except using $[\mathbf{R}^*_{i1}|\mathbf{R}^*_{i2} + \mathbf{R}_{i^*2}]$ to decode message.
4. If $i \neq i^*$ and $\ell = 2$, oracle reconstructs the message/level-dependent matrix $\mathbf{A}_{i,u,2}$ as

$$\begin{aligned}\mathbf{A}_{i,u,2} &= [\mathbf{A}^* | - \mathbf{A}^*\mathbf{R}^*_{i1}| - \mathbf{A}^*\mathbf{R}^*_{i2} - \mathbf{A}_{i^*3} + (h(u) - h(u^*))\mathbf{G}] \\ &= [\mathbf{A}^* | - \mathbf{A}^*\mathbf{R}^*_{i1}| - \mathbf{A}^*(\mathbf{R}^*_{i2} + \mathbf{R}_{i^*2}) + h(u - u^*)\mathbf{G}]\end{aligned}$$

   where $\mathbf{R}^*_{i2} = (\mathbf{X}_{i,02} - \mathbf{R}_{i^*1}\mathbf{X}_{i,12})$. Call algorithm $\mathsf{Invert}'^{\mathcal{O}}([\mathbf{R}^*_{i1}|\mathbf{R}^*_{i2} + \mathbf{R}_{i^*2}], \mathbf{A}_{i,u},$ $\boldsymbol{b} \bmod q)$ to get some $\boldsymbol{z} \in \mathbb{Z}^n$ and $\boldsymbol{e}$. Then oracle performs step 3 exactly as in Dec, except using $[\mathbf{R}^*_{i1}|\mathbf{R}^*_{i2} + \mathbf{R}_{i^*2}]$ to decode message.

   In summarize, the decryption oracle can answer any decryption queries for uncorrupted users as long as $u \neq u^*$, which is ensure with overwhelming probability because u^* is statistically hidden, and by the "unit difference" property on set $\mathcal{U}$ we have $h(u) - h(u^*) = h(u - u^*)$ is invertible, as require by calling $\mathsf{Invert}^{\mathcal{O}}$ algorithm.

- **Challenge oracle**: The oracle produces challenge ciphertext $(u, \boldsymbol{b}, 1)$ on a message $\boldsymbol{\mu}^* \in \{0,1\}^{nk}$ as follows. Let $u = u^*$, then the message/level-dependent matrix is $\mathbf{A}_{i^*,u^*,1} = [\mathbf{A}^* | - \mathbf{A}^*\mathbf{R}_{i^*1}| - \mathbf{A}^*\mathbf{R}_{i^*2}]$. Then oracle sets the first nk coordinates of challenge ciphertext $\boldsymbol{b}$ to be $\boldsymbol{b}_0 = \boldsymbol{b}^*$, where $(\mathbf{A}^*, \boldsymbol{b}^*)$ is the LWE instance. The last $2nk$ coordinates can be set as

$$\boldsymbol{b}_1 = \boldsymbol{b}_0^\intercal \mathbf{R}_{i^*1} + \boldsymbol{e}_1 \bmod 2q, \qquad \boldsymbol{b}_2 = \boldsymbol{b}_0^\intercal \mathbf{R}_{i^*2} + \boldsymbol{e}_2 + \mathsf{encode}(\boldsymbol{\mu}^*) \bmod 2q$$

  where $\boldsymbol{e}_1, \boldsymbol{e}_2 \in \mathcal{D}^{nk}_{\mathbb{Z},s}$. Then oracle output the challenge ciphertext $\mathsf{ct} = (\mu^*, \boldsymbol{b})$.

Hybrid H_2 : In game H_2, the vector in LWE instance $\widetilde{\boldsymbol{b}^*} \in \mathbb{Z}^{nk}_{2q}$ is sampled uniformly random. We change how the first nk coordinate of challenge ciphertext is created, by letting it be $\widetilde{\boldsymbol{b}^*}$. We construct the pubic key for each user, answer the queries, and construct the last $3nk$ coordinates of challenge ciphertext in the exactly way as in H_1.

Claim 4.3. *Hybrids* H_0 *and* H_1 *are statistically close.*

Proof. The adversary obtains all the public keys and re-encryption keys as he wants in hybrid H_0 and H_1, where in H_1, challenge public key is

$$\mathsf{pk}_{i^*} = [\mathbf{A}^* | -\mathbf{A}^*\mathbf{R}_{i^*1} - h(1)\mathbf{G} | -\mathbf{A}^*\mathbf{R}_{i^*2} - h(u^*)\mathbf{G}]$$

Note the challenge public key pk_{i^*} is still $\mathsf{negl}(\lambda)$-uniform for any choice of u^*, thus is identically distributed as in H_0. Therefore also conditioned on any fixed choice of pk_{i^*}, the value of u^* is statistically hidden from the adversary. For uncurrupted users other than the challenge user, we have

$$\mathbf{A}_{i1} = [\mathbf{A} | -\mathbf{A}_{i^*1}] \begin{bmatrix} \mathbf{X}_{i,01} \\ \mathbf{X}_{i,11} \end{bmatrix}, \qquad \mathbf{A}_{i2} = [\mathbf{A} | -\mathbf{A}_{i^*1}] \begin{bmatrix} \mathbf{X}_{i,02} \\ \mathbf{X}_{i,12} \end{bmatrix}$$

Since matrices $\mathbf{X}_{i,01}, \mathbf{X}_{i,11}, \mathbf{X}_{i,02}, \mathbf{X}_{i,12}$ are sampled from discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z},s}^{nk \times nk}$, the public keys are identically distributed as in H_0. For re-encryption keys, because they are all sampled from discrete Gaussian distribution in hybrid $\mathsf{H}_0, \mathsf{H}_1$, they are identically distributed as well. Therefore, hybrid H_0 and H_1 are identically distributed.

We now show that the distribution of $(u^*, \boldsymbol{b})$ is within $\mathsf{negl}(\lambda)$ statistical distance of that in game H_0 from the adversary's view. Clearly, u^* and $\boldsymbol{b}_0$ have the same distribution as in H_0, because u^* is $\mathsf{negl}(\lambda)$-uniform given the public keys of uncorrupted users, and $\boldsymbol{b}_0 = 2(\boldsymbol{s}^\mathsf{T}\mathbf{A}^* \bmod q) + \widetilde{\boldsymbol{e}_0^\mathsf{T}}$ is from the LWE instance. Since the noise item in $\boldsymbol{b}_1$ is $\widetilde{\boldsymbol{e}_0^\mathsf{T}}\mathbf{R}_{i^*1} + \boldsymbol{e}_1$, by Corollary 3.10 in [Reg05], vector $\boldsymbol{b}_1$ are within $\mathsf{negl}(\lambda)$-statistical distance from discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^n k,s}$. The same argument also applies to $\boldsymbol{b}_2$. □

Claim 4.4. *Assuming the hardness of* LWE *assumption, then hybrid* H_1 *and* H_2 *are computationally indistinguishable.*

Proof. It can be verified that when the distribution provided to simulation is indeed $A_{\boldsymbol{s},\alpha'}$, the simulation emulates hybrid H_1. The main observations are that decryption oracle works when $u \neq u^*$, the distribution of re-encryption keys computed by $\mathcal{O}_{\mathsf{ReKeyGen}}$ are statistically close to the that in the real scheme. On the other hand, when the distribution provided to simulation is uniform, the simulation emulates hybrid H_2, since $\widetilde{\boldsymbol{b}^*}$ is uniform and independent of all other variables. □

Claim 4.5. *In hybrid* H_2*, the probability of adversary wining the game is zero.*

Proof. This claim can be proved by Leftover Hash Lemma 2.2,

$$(\mathbf{A}^*, \widetilde{\boldsymbol{b}^*}, -\mathbf{A}^*\mathbf{R}_{i^*1}, -\mathbf{A}^*\mathbf{R}_{i^*2}, \widetilde{\boldsymbol{b}^{*\mathsf{T}}}\mathbf{R}_{i^*1}, \widetilde{\boldsymbol{b}^{*\mathsf{T}}}\mathbf{R}_{i^*2})$$

is $\mathsf{negl}(\lambda)$-uniform when matrices $\mathbf{R}_{i^*1}, \mathbf{R}_{i^*2}, \mathbf{R}_{i^*3}$ are chosen as in H_2. Also, observe the vector $\widetilde{\boldsymbol{b}^*}$ is uniform and independent of the key. Therefore, the challenge ciphertext has the same distribution (up to $\mathsf{negl}(\lambda)$ statistical distance) for any encrypted message, and the distinguishing advantage of any adversary is zero. □

Combining hybrids $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$ and the above claims, we complete the proof. □

4.2 Parameter Selection

In this section, we set the lattice parameters used in our construction. $\mathbf{G} \in \mathbb{Z}_q^{n \times nk}$ is a gadget matrix for $q = \mathsf{poly}(n), n = \mathsf{poly}(\lambda)$ and $k = O(\log q) = O(\log n)$. For matrix $\mathbf{A} \in \mathbf{Z}_q^{n \times m}$ in the public parameters and secret keys $\mathbf{R} \leftarrow \mathcal{D}$, we set $m = O(nk)$ and $\mathcal{D} = \mathcal{D}_{\mathbf{Z}, w(\sqrt{\log n})}^{m \times nk}$ respectively. We set the deviation s for discrete Gaussian distribution used in security proof to be $s = w(\sqrt{\log n})$. For the error rate α in the LWE assumption, we set sufficiently large $1/\alpha = O(nk) \cdot w(\sqrt{\log n})$.

5 Multi-hop Tag-based CCA-Secure PRE Construction

In this section, we present our construction for multi-hop tag-based CCA secure PRE scheme as defined in Definition 3.8. The construction can be viewed as a natural variant of the single-hop scheme presented in Section 4. The maximum times of re-encryption applied to the ciphertext depends on the types of LWE assumptions.

We construct the large set $\mathcal{U}$ and encode messages $\mu \in \{0,1\}^{nk}$ using the same technique as the single-hop case as Section 4. Here instead of encoding the level ℓ in the encryption matrix (recall that in the previous scheme $\ell \in \{1,2\}$), in the multi-hop scheme, each user chooses a random public element $v_i \in \mathcal{U}$. The encryption algorithm will choose a random element $u \in \mathcal{U}$ (as a tag) for each encryption, and embeds the matrices $h(v_i), h(u)$ into the encryption matrix as: $\mathbf{A}_{i,u} = [\mathbf{A}| - \mathbf{A}_{i1} + h(v_i)\mathbf{G}| - \mathbf{A}_{i2} + h(u)\mathbf{G}]$. We describe the scheme in detail as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: The setup algorithm sets the lattice parameter $n = n(\lambda, \ell), k = k(\lambda, \ell), q = q(\lambda, \ell)$ and Gaussian parameter $s = s(\lambda, \ell)$. Then it randomly selects a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times nk}$, and outputs the public parameter $\mathsf{pp} = (\mathbf{A}, n, k, q, s)$.

- $\mathsf{KeyGen}(\mathsf{pp})$: The key generation algorithm for i-th user chooses random matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2} \leftarrow \mathcal{D}_{\mathbb{Z}_q^{nk \times nk}, s}$, letting $\mathbf{A}_{i1} = \mathbf{A}\mathbf{R}_{i1} \bmod q$ and $\mathbf{A}_{i2} = \mathbf{A}\mathbf{R}_{i2} \bmod q$. The public key is $\mathsf{pk}_i = [\mathbf{A}| - \mathbf{A}_{i1}| - \mathbf{A}_{i2}]$, and the secret key is $\mathsf{sk}_i = [\mathbf{R}_{i1}|\mathbf{R}_{i2}]$.

- $\mathsf{Enc}(\mathsf{pk}_i, \boldsymbol{\mu}, \kappa)$: The encryption algorithm chooses non-zero $u \leftarrow \mathcal{U}$ and let the message/user-dependent matrix
$$\mathbf{A}_{i,u} = [\mathbf{A}| - \mathbf{A}_{i1} + h(\kappa)\mathbf{G}| - \mathbf{A}_{i2} + h(u)\mathbf{G}]$$
The rest of encryption algorithm is the same as encryption algorithm in single-hop CCA-PRE scheme described in Section 4.

- $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$:The decryption algorithm does:

  1. If ct does not parse or $u = 0$, output $\perp$. Otherwise, reconstruct the message/user-dependent matrix $\mathbf{A}_{i,u}$
  $$\mathbf{A}_{i,u} = [\mathbf{A}| - \mathbf{A}_{i1} + h(\kappa)\mathbf{G}| - \mathbf{A}_{i2} + h(u)\mathbf{G}]$$
  The rest of decryption algorithm is the same as decryption algorithm in single-hop CCA-PRE scheme described in Section 4.

- $\mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{sk}_i, \mathsf{pk}_j, \kappa)$: The re-encryption key generation algorithm does:

  1. First, it parses i-th public/secret key pair and j-th public key as follows:
  $$\mathsf{pk}_i = [\mathbf{A}| - \mathbf{A}_{i1}| - \mathbf{A}_{i2}], \quad \mathsf{sk}_i = [\mathbf{R}_{i1}|\mathbf{R}_{i2}], \quad \mathsf{pk}_j = [\mathbf{A}| - \mathbf{A}_{j1}| - \mathbf{A}_{j2}]$$

2. Use extended sampling algorithm $\mathsf{Sample}^{\mathcal{O}}$ to sample $\mathbf{X}_{01}, \mathbf{X}_{02}, \mathbf{X}_{11}, \mathbf{X}_{12} \in \mathbb{Z}^{nk \times nk}$ such that

$$[\mathbf{A}| - \mathbf{A}_{i1} + h(\kappa)\mathbf{G}| - \mathbf{A}_{i2} + \mathbf{B}] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{01} & \mathbf{X}_{02} \\ 0 & \mathbf{X}_{11} & \mathbf{X}_{12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}| - \mathbf{A}_{j1} + h(\kappa+1)\mathbf{G}| - \mathbf{A}_{j2} + \mathbf{B}]$$

for any matrix $\mathbf{B} \in \mathbb{Z}^{n \times nk}$.

3. Output the re-encryption key

$$\mathsf{rk}^{\kappa}_{i \to j} = \{\mathbf{X}_{01}, \mathbf{X}_{02}, \mathbf{X}_{11}, \mathbf{X}_{12}\}$$

- $\mathsf{ReEnc}(\mathsf{rk}^{\kappa}_{i \to j}, \mathsf{ct})$: The re-encryption algorithm for tree-based PRE is the same as the counterpart in single-hop CCA-PRE secure scheme in Section 4.

It is obviously to see that with appropriate parameter setting as specified in Section 5.2, the correctness of the construction can be implied by the correctness of previous scheme as proof for Lemma 4.1.

5.1 Security Proof

In this section, we prove our tag–based multi-hop PRE construction is CCA-secure as Definition 3.8. As described in experiment $\mathbf{Expt}^{\mathsf{multi}}_{\mathcal{A}}(1^{\lambda})$, the adversary needs to commit a tree structure T before obtaining public parameters and public keys for nodes in the tree. Given the tree T beforehand, the reduction can simulate the public keys and re-encryption keys in a specified order. On the other hand, if the structure is unknown, then it is unclear how the reduction can simulate all the rk's in an arbitrary order of queries. This limitation comes from current techniques in security proof, yet there are no known attacks to the construction, to our knowledge. It is an interesting open problem to determine whether the construction achieves the full security notion as defined in Definition 3.8.

Theorem 5.1. *Assuming the hardness of* $\mathsf{LWE}_{q,\alpha'}$ *where* $\alpha' = \alpha/3 \geq 2\sqrt{n}/q$*, the proxy re-encryption scheme is CCA-secure as defined in Definition 3.8.*

Intuition. At a high level, the reduction simulates the oracles described in experiment $\mathbf{Expt}^{\mathsf{multi}}_{\mathcal{A}}(1^{\lambda})$ using LWE instance. In particular, the public key of the root user is generated by embedding the committed information of challenge level κ^* and a randomly chosen target tag t^* for challenge ciphertext, and for generating public keys of the vertex on the tree, the reduction first sample re-encryption key for edges (associated with levels as described in Definition 3.8), then the public key of vertex is a multiplication of re-encryption key and its parent's public key. Generating public key for vertex i in this manner ensures the trapdoor using for computing re-encryption keys dose not vanish except for its tree level $\ell(i)$. Therefore, for re-encryption key queries with respect to admissible edges, the reduction can still obtain re-encryption key from the distribution. The reduction generates the public key for other users that are not on the tree, using the KeyGen algorithm, thus the queries for re-encryption keys between these users or re-encryption keys towards these users are not restricted. For the decryption query, the reduction can decrypt ciphertext as long as the tag is not the target one t^*, since the trapdoor for decryption dose not vanish. At the end of experiment, the adversary has to guess the random bit b' selected in challenge oracle, where the challenge ciphertext is computed from the LWE instance.

Proof. First, we obtain the desired LWE form $(\mathbf{A}^*, \boldsymbol{b}^*)$ by using the same technique described in proof of Theorem 4.2. We proceed via a sequence of games.

Hybrid H_0 : The game H_0 is exactly the same as the CCA attack with the real system described above.

Hybrid H_1 : In game H_1, we change how the public keys and re-encryption key for users in the tree $\mathsf{T} = (V, E)$ and challenge ciphertext $\mathsf{ct}^* = (u^*, \boldsymbol{b}^*)$ are constructed, and the way that decryption queries are answered, but in a way that the probability of distinguishing hybrid H_0 and H_1 is $\mathsf{negl}(\lambda)$. Suppose adversary $\mathcal{A}$ sends a tree structure $T = (V, E)$, a target level $\kappa^* \leq L$, where the depth of tree T is less than $L - \kappa^*$ and the root r is treated as the target user.

Setup. Reduction $\mathcal{B}$ uses the LWE instance $(\mathbf{A}^*, \boldsymbol{b}^*)$ to setup the PRS system as follows:

1. Set the lattice parameters q, n, m according to the LWE problem, and Gaussian parameter $s = s(n)$. Initialize set $\mathcal{H} = V$ for honest users, $\mathcal{C} = \emptyset$ for corrupt users, $K = \emptyset$ for generated re-encryption keys. Set graph $G = (E', V')$, where edge set $E' = E$ and vertex set $V' = V$.
2. Let the matrix in public parameter pp to be $\mathbf{A} = \mathbf{A}^*$ and choose a random element $u^* \in \mathcal{U}$.
3. Run a Deep-First traversal to associated each edge with a level index $\ell(i)$, by increasing from root user's target level κ^*, and update the edge set by augmenting each edge $(i, j) \in E$ with its level index $(i, j, \ell(i))$. Update the edge E' correspondingly.
4. To generate a public key for the root user $r \in V$, the oracle chooses random matrices $\mathbf{R}_{r1}, \mathbf{R}_{r2}$ from $\{-1, 1\}^{m \times m}$, then output the public key to be

$$\mathbf{A}_{r1} = \mathbf{A}\mathbf{R}_{r1} + h(\kappa^*)\mathbf{G}, \qquad \mathbf{A}_{r2} = \mathbf{A}\mathbf{R}_{r2} + h(u^*)\mathbf{G}]$$

5. For node i which is the child node of the root node r, sample matrices $\mathbf{X}_{i,01}, \mathbf{X}_{i,02}$, $\mathbf{X}_{i,11}, \mathbf{X}_{i,12} \leftarrow \mathcal{D}_{\mathbb{Z}_q^{m \times m}, s}$, such that

$$[\mathbf{A}|\mathbf{A}_r + h(\kappa^*)\mathbf{G}|\mathbf{A}'_r] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{i,01} & \mathbf{X}_{i,02} \\ 0 & \mathbf{X}_{i,11} & \mathbf{X}_{i,12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}|\mathbf{A}_i + h(\ell(i))\mathbf{G}|\mathbf{A}'_i]$$

   where matrices $(\mathbf{A}_i, \mathbf{A}'_i)$ are set as

$$\mathbf{A}_i = \mathbf{A}\mathbf{R}_{i1} - h(\ell(i))\mathbf{G} = \mathbf{A}(\mathbf{X}_{i,01} + \mathbf{R}_{r1}\mathbf{X}_{i,11}) - h(\ell(i))\mathbf{G}$$
$$\mathbf{A}'_i = \mathbf{A}\mathbf{R}_{r2} - t\mathbf{G} = \mathbf{A}(\mathbf{X}_{i,02} + \mathbf{R}_{r1}\mathbf{X}_{j,12} + \mathbf{R}_{r2}) - h(u^*)\mathbf{G}$$

   and the re-encryption key $\mathsf{rk}_{i \to r}^{\ell(i)} = (\mathbf{X}_{i,01}, \mathbf{X}_{i,02}, \mathbf{X}_{i,11}, \mathbf{X}_{i,12})$. For child j of node i, we can generate the public key $(\mathbf{A}_j, \mathbf{A}'_j)$ and re-encryption key $\mathsf{rk}_{j \to i}^{\ell(j)}$ using the same procedure. Put all the generated re-encryption keys into set K.
6. Send the public parameter pp and public keys $\{\mathsf{pk}_i\}_{i \in V}$ back to adversary $\mathcal{A}$.

Next, we present the description of oracles used in the security proof:

- **Uncorrupted key generation oracle**: To obtain the public key for uncorropted user i, the oracle chooses random matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2} \leftarrow \mathcal{D}$, letting $\mathbf{A}_{i1} = \mathbf{A}\mathbf{R}_{i1} \bmod q$ and $\mathbf{A}_{i2} = \mathbf{A}\mathbf{R}_{i2} \bmod q$, and sends back $\mathsf{pk}_i = (\mathbf{A}_{i1}, \mathbf{A}_{i2})$. Then it sets $\mathcal{H} = \mathcal{H} \cup \{i\}, V' = V' \cup \{i\}$.
- **Corrupted key generation oracle**: To obtain the pair of secret and public keys for corrupted user i, the oracle chooses random matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2} \leftarrow \mathcal{D}$, letting $\mathbf{A}_{i1} = \mathbf{A}\mathbf{R}_{i1} \bmod q$ and $\mathbf{A}_{i2} = \mathbf{A}\mathbf{R}_{i2} \bmod q$, and sends back $\mathsf{pk}_i = (\mathbf{A}_{i1}, \mathbf{A}_{i2}), \mathsf{sk}_i = (\mathbf{R}_{i1}, \mathbf{R}_{i2})$. Then it sets $\mathcal{C} = \mathcal{C} \cup \{i\}, V' = V' \cup \{i\}$.

- **Re-encryption key generation oracle**: On input user index i, j and level index κ, oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ outputs $\perp$ if the edge (i, j, κ) is not an admissible edge with respect to edge set E and target user/level (r, κ^*). Otherwise,
  - If $(i, j, \kappa) \in E'$, then look for the re-encryption key $\mathsf{rk}_{i\to j}^{\kappa}$ already generated in set K and send $\mathsf{rk}_{i\to j}^{\kappa}$ back to adversary $\mathcal{A}$.
  - Otherwise, we divide into two cases: (1) $j \in V, \kappa \neq \ell(j)$; (2) $j \notin V$. For these two cases, the trapdoor used for sampling re-encryption key dose not vanish, thus sample small matrices $\mathsf{rk}_{i\to j}^{\kappa} = (\mathbf{X}_{i,01}, \mathbf{X}_{i,11}, \mathbf{X}_{i,02}, \mathbf{X}_{i,12})$, using

$$(\mathbf{X}_{i,01}, \mathbf{X}_{i,11}) \leftarrow \mathsf{SampleRight}(\mathbf{A}, h(\kappa+1)\mathbf{G}, \mathbf{R}_{j1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}_i + h(\kappa)\mathbf{G}, s),$$

$$(\mathbf{X}_{i,02}, \mathbf{X}_{i,12}) \leftarrow \mathsf{SampleRight}(\mathbf{A}, h(\kappa+1)\mathbf{G}, \mathbf{R}_{j1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}'_i - \mathbf{A}'_j, s)$$

    Therefore it holds that

$$[\mathbf{A}|\mathbf{A}_j + h(\kappa+1)\mathbf{G}|\mathbf{A}'_j] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{i,01} & \mathbf{X}_{i,02} \\ 0 & \mathbf{X}_{i,11} & \mathbf{X}_{i,12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}|\mathbf{A}_i + h(\kappa)\mathbf{G}|\mathbf{A}'_i]$$

    Then update $E' = E' \cup \{(i, j, \kappa)\}, K = K \cup \{\mathsf{rk}_{i\to j}^{\kappa}\}$ and send $\mathsf{rk}_{i\to j}^{\kappa}$ to adversary $\mathcal{A}$.
- **Re-encryption key generation oracle**: On query (i, j, κ), the oracle answers queries in the following cases:
  1. If $i \in V, j \notin V$ and $\kappa = \ell(i)$, then oracle outputs $\perp$.
  2. If $i, j \in V$ but $(i, j) \notin E$, then oracle outputs $\perp$.
  3. If $(i, j) \in E$ and $\kappa = \ell(i)$, then output the re-encryption key $\mathsf{rk}_{i\to j}^{\ell(i)}$ computed in the key generation of tree T.
  4. Otherwise, since the trapdoor for pk_i^{κ} dose not vanish, we can divide the rest into the following three cases:
     - If $(i, j) \in E$ and $\kappa \neq \ell(i)$, then first reconstruct the public key for user i, j at level κ and $\kappa + 1$ respectively as

$$\mathsf{pk}_i^{\kappa} = [\mathbf{A}|-\mathbf{A}_{i1}+(h(\kappa)-h(\ell(i)))\mathbf{G}|-\mathbf{A}_{i2}+\mathbf{B}], \quad \mathsf{pk}_j^{\kappa+1} = [\mathbf{A}|-\mathbf{A}_{j1}+(h(\kappa+1)-h(\ell(j)))\mathbf{G}|-\mathbf{A}_{j2}+\mathbf{B}]$$

       Since the trapdoor of pk_i^{κ} dose not vanish, we can still use the same method as in the scheme to generate the re-encryption key $\mathsf{rk}_{i\to j}^{\kappa}$.
     - If $i \notin V$ and $j \in V$, since the trapdoor of pk_i^{κ} dose not vanish, we can still use the same method as in the scheme to generate the re-encryption key $\mathsf{rk}_{i\to j}^{\kappa}$.
     - If $i \in V, j \notin V$ and $\kappa \neq \ell(i)$, we can first reconstruct the public key for user i at level κ as

$$\mathsf{pk}_i^{\kappa} = [\mathbf{A}| - \mathbf{A}_{i1} + (h(\kappa) - h(\ell(i)))\mathbf{G}| - \mathbf{A}_{i2} + \mathbf{B}]$$

       The trapdoor in pk_i^{κ} dose not vanish, so we can compute the re-encryption key $\mathsf{rk}_{i\to j}^{\kappa}$ using the same method as in the scheme.
- **Re-encryption oracle**: On input $(i, j, \kappa, \mathsf{ct})$, the oracle first parses the ciphertext as $\mathsf{ct} = (u, \boldsymbol{b}, \kappa)$, and output $\perp$ if $u = 0, \kappa \geq L - 1$ or the ciphertext ct is not well formed.
  - If the edge $(i, j, \kappa) \in E'$, then find re-encryption key $\mathsf{rk}_{i\to j}^{\kappa}$ in set K. Output re-signature $(\mathsf{rk}_{i\to j}^{\kappa} \cdot \boldsymbol{b}, \kappa + 1, u)$ to adversary.

– Otherwise if the edge $(i, j, \kappa) \notin E', j \in V$ and $\kappa = \ell(j) - 1$, then output $\perp$. Otherwise, call re-encryption key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ described above, and output re-signature $(\mathsf{rk}_{i\to j}^{\kappa} \cdot \boldsymbol{b}, \kappa + 1, u)$ to adversary.

- **Re-encryption oracle**: On input $(i, j, \kappa, \mathsf{ct})$, the oracle first parses the ciphertext as $\mathsf{ct} = (u, \boldsymbol{b})$, and output $\perp$ if $u = 0$. Otherwise, the oracle uses secret key sk_i to check the validness of ciphertext ct by running $\mu \leftarrow \mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$ to check if $\boldsymbol{\mu}$ is in $\{0,1\}^{nk}$. The oracle outputs $\perp$ if ct is an invalid ciphertext. Then it computes the re-encrypted ciphertext as
  1. If $i \in V$ and $\kappa = \ell(i)$, then oracle outputs the multiplication $\boldsymbol{b} \cdot \mathsf{rk}_{i\to j}^{\kappa}$, where $\mathsf{rk}_{i\to j}^{\kappa}$ is pre-sampled in the generation of tree T.
  2. If $i \in V, \kappa = \ell$ and $j \in \mathcal{H} \bigcup \mathcal{C}$, the oracle first reconstructs the encryption matrix as
  $$\mathbf{A}_{i,u}^{\kappa} = [\mathbf{A}| - \mathbf{A}\mathbf{R}_{i1}|\mathbf{A}\mathbf{R}_{i1} + h(u - u^*)\mathbf{G}]$$
  Since $u \neq u^*$, oracle can decrypt the ciphertext to obtain the message $\boldsymbol{\mu}$. Then oracle outputs $\mathsf{Enc}(\mathsf{pk}_j, \boldsymbol{\mu}, \kappa + 1)$.
  3. Otherwise, the oracle first computes the re-encryption key $\mathsf{rk}_{i\to j}^{\kappa}$ as described in re-encryption key generation oracle as above, then outputs outputs the multiplication $\boldsymbol{b} \cdot \mathsf{rk}_{i\to j}^{\kappa}$.
- **Decryption oracle**: On decryption query (i, ct, κ) from adversary $\mathcal{A}$, the oracle first parses ciphertext as $\mathsf{ct} = (u, \boldsymbol{b})$, and output $\perp$ if $u = 0$. If the decryption queries are made after the challenge oracle query, then oracle outputs $\perp$ if $u = u^*$. Then oracle divides the decryption process into following two cases:
  1. If $i \in V$, the oracle first re-constructs the encryption matrix as
  $$\begin{aligned}\mathbf{A}_{i,u}^{\kappa} &= [\mathbf{A}| - \mathbf{A}_{i1} - (h(\kappa) - h(\ell(i)))\mathbf{G}|\mathbf{A}_{r2} - h(u^*)\mathbf{G} + h(u)\mathbf{G}] \\ &= [\mathbf{A}| - \mathbf{A}\mathbf{R}_{i1} - (h(\kappa) - h(\ell(i)))\mathbf{G}|\mathbf{A}\mathbf{R}_{i1} + h(u - u^*)\mathbf{G}]\end{aligned}$$
  where matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2}$ can be computed from the generation of public keys for tree T. Call algorithm $\mathsf{Invert}'^{\mathcal{O}}([\mathbf{R}_{i1}|\mathbf{R}_{i2}], \mathbf{A}_{i,u}^{\kappa}, \boldsymbol{b} \bmod q)$ to get some $\boldsymbol{z} \in \mathbb{Z}^n$ and $\boldsymbol{e}$. Then oracle performs step 3 exactly as in Dec, except using $[\mathbf{R}_{i1}|\mathbf{R}_{i2}]$ to decode message.
  2. If $i \notin V$, then we generated user i's public key and secret key using the same method as the scheme. Thus, we can decrypt the ciphertext ct using algorithm $\mathsf{Dec}(\mathsf{sk}_i, \mathsf{ct})$.
- **Challenge oracle**: On input a challenge tuple $(i^*, \kappa, \boldsymbol{\mu}_0, \boldsymbol{\mu}_1)$, the oracle outputs $\perp$ if $i^* \notin V$ or $\kappa \neq \ell(i)$. Otherwise, the oracle produces challenge ciphertext $(u^*, \boldsymbol{b}^*)$ on a challenge query $(i^*, \boldsymbol{\mu}_b)$ as follows. Let $u = u^*$, then the message/user-dependent matrix is $\mathbf{A}_{i^*,u^*}^{\kappa} = [\mathbf{A}^*| - \mathbf{A}^*\mathbf{T}_{i^*1}| - \mathbf{A}^*\mathbf{T}_{i^*2}]$. Then oracle sets the first nk coordinates of challenge ciphertext $\boldsymbol{b}$ to be $\boldsymbol{b}_0 = \boldsymbol{b}^*$, where $(\mathbf{A}^*, \boldsymbol{b}^*)$ is the LWE instance. The last $2nk$ coordinates can be set as
  $$\boldsymbol{b}_1^* = \boldsymbol{b}_0^t\mathbf{T}_{i^*1} + \boldsymbol{e}_1 \bmod 2q, \qquad \boldsymbol{b}_2^* = \boldsymbol{b}_0^t\mathbf{T}_{i^*2} + \boldsymbol{e}_2 + \mathsf{encode}(\boldsymbol{\mu}_b) \bmod 2q$$
  where $\boldsymbol{e}_1, \boldsymbol{e}_2 \in \mathcal{D}_{\mathbb{Z},s}^{nk}$. Then oracle output the challenge ciphertext $\mathsf{ct} = (\mu^*, \boldsymbol{b}^*)$. We now show that the distribution of $(u^*, \boldsymbol{b}^*)$ is within $\mathsf{negl}(\lambda)$ statistical distance of that in game H_0 from the adversary's view. Clearly, u^* and $\boldsymbol{b}_0$ have the same distribution as in H_0, because u^* is computationally close to uniform given the public keys of uncorrupted users, and $\boldsymbol{b}_0^* = 2(\boldsymbol{s}^t\mathbf{A} \bmod q) + \tilde{\boldsymbol{e}_0^t}$ is from the LWE instance. Since the noise item in $\boldsymbol{b}_1^*$ is $\tilde{\boldsymbol{e}_0^t}\mathbf{R}_{i^*1} + \boldsymbol{e}_1$, by Corollary 3.10 in [Reg05], vector $\boldsymbol{b}_1^*$ are within $\mathsf{negl}(\lambda)$-statistical distance from discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^n k,s}$. The same argument also applies to $\boldsymbol{b}_2^*$.

Hybrid H_2 : The same as the hybrid H_2 in the proof of Theorem 4.2.

Claim 5.2. *Hybrids* H_0 *and* H_1 *are statistically close.*

Claim 5.3. *Assuming the hardness of LWE assumption, then hybrid* H_1 *and* H_2 *are computationally indistinguishable.*

Claim 5.4. *In hybrid* H_2*, the probability of adversary wining the game is zero.*

The proofs of the above claims is analogous to that in Theorem 4.2, thus we omit the detail here. Combining hybrids $\mathsf{H}_0, \mathsf{H}_1, \mathsf{H}_2$ and the above claims, we complete the proof. □

5.2 Parameter Selection

$\mathbf{G} \in \mathbb{Z}_q^{n \times nk}$ is a gadget matrix for $k = O(\log q) = O(\log n)$. For matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ in the public parameters and secret keys $\mathbf{R} \leftarrow \mathcal{D}$, we set $m = O(nk)$ and $\mathcal{D} = \mathcal{D}^{m \times nk}_{\mathbb{Z}, w(\sqrt{\log n})}$ respectively. We set the deviation s for discrete Gaussian distribution used in security proof to be $s = w(\sqrt{\log n})$. After each re-encryption the noise will grow a $\sqrt{ms}$ factor. For the error rate α in the LWE assumption, we set sufficiently large $1/\alpha = O(nk) \cdot w(\sqrt{\log n})$. We use H to denote the maximum depth of the tree structure used in our construction, and we have $H = O(n/\log n)$. In order to achieve fixed poly-log depth H, so q has to set to be greater then $m^{H/2}$. Thus we have to rely sub-exponential LWE to set $q = 2^{n^\epsilon}, 0 < \epsilon < 1$, then $n = 2^{\lambda^\epsilon}, 0 < \epsilon < 1$. We notice that in [LL15], Laine and Lauter proposed an attack for LWE assumption when q is exponentially large (i.e. 2^{cn}), but in our setting, $q = 2^{n^\epsilon}$ is sub-exponentially in terms of the security parameter, so our assumption still holds. If we want to rely on standard LWE assumption, then we can achieve constant depth by setting $q = \mathsf{poly}(n), n = \mathsf{poly}(\lambda)$.

6 Proxy Re-Signature with Selectively Chosen Tag

In this section, we present the syntax and security definition of PRS, and then describe our construction.

6.1 Syntax and Correctness Definition

We first recall the syntax and security definition of PRS in [AH05, LV08a], then propose a simpler and unified security model that captures the security requirements. Our model adapts the same spirit of the prior security model of proxy re-encryption [CH07, LV08b], with necessary modifications to fit into the signature framework. We also compare our new notion with the previous security model in [AH05, LV08a].

Let $L = L(\lambda)$ denotes the maximum level the PRS system supports. The scheme $\Sigma = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{ReKeyGen}, \mathsf{ReSign})$ is described as follows:

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$ generates the public parameter pp for the whole system.
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$ generates $(\mathsf{pk}_i, \mathsf{sk}_i)$ for user i.
- $\sigma \leftarrow \mathsf{Sign}(\mathsf{sk}_i, \mu, \kappa)$ computes a signature σ for μ at level κ.
- $\mathsf{Verify}(\mathsf{pk}_i, \sigma, \mu, \kappa)$ outputs 1 (accept) or 0 (reject).
- $\mathsf{rk}^\kappa_{i \to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{pk}_j, \mathsf{sk}_j, \kappa)$ computes a re-signing key from the i-th user at level κ to the j-th user at level $\kappa + 1$.

- $\mathsf{ReSign}(\mathsf{rk}^{\kappa}_{i\to j}, \mu, \sigma, \kappa)$ computes a re-signature σ' under pk_j if $\mathsf{Verify}(\mathsf{pk}_i, \sigma, \mu, \kappa) = 1$, or $\perp$ otherwise.

Correctness. For all security parameter λ, any $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$, all couples of secret/public key pairs $(\mathsf{sk}_i, \mathsf{pk}_i), (\mathsf{sk}_j, \mathsf{pk}_j)$ generated by $\mathsf{KeyGen}(\mathsf{pp})$, for any message μ and $\kappa \in [L]$, it holds that

$$\mathsf{Verify}(\mathsf{pk}_i, \mu, \kappa, \mathsf{Sign}(\mathsf{sk}_i, \mu, \kappa)) = 1$$

$$\mathsf{Verify}(\mathsf{pk}_j, \kappa + 1, \mu, \sigma) = 1$$

where $\sigma = \mathsf{ReSign}(\mathsf{rk}^{\kappa}_{i\to j}, \mu, \kappa, \mathsf{Sign}(\mathsf{sk}_i, \mu, \kappa))$ and $\mathsf{rk}^{\kappa}_{i\to j} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_i, \mathsf{pk}_j, \mathsf{sk}_j, \kappa)$.

6.2 Security Definition

Security of PRS is first defined by Ateniese and Hohenberger [AH05], and then later Libert and Vergnaud [LV08a] augmented the notion by associating re-keys with levels. The previous security notion basically considers four scenarios, namely, *external security, limited proxy security, delegatee security*, and *delegator security*, capturing signature unforgeability when the adversary obtains signing/re-signing oracles and re-keys with respect to different scenarios. In this section, we define a simpler and unified model that captures all these four scenarios. Our new definition is similar to the CCA-PRE definition of multi-hop PRE (c.f. Definition 3.8).

Intuitively, the adversary needs to commit to the tree structure and the target level before seeing the public parameters *pp* and the public keys of the tree nodes. The root node along with the committed level serve as the *target pair* for the adversary. Then adversary can register honest and corrupt users of his choice in the experiment, and also ask signing oracle to sign an arbitrary message using honest users' secret key. Upon receiving valid signatures, the re-signing oracle compute the re-signature by first looking up (or calculating) the appropriate re-signing key and then doing the re-signing procedure. We also define the notion *admissible edge* for a (edge, level) pair in the PRS scenario, where there does not exist a path from the newly added (edge, level) pair to the target pair. The re-signing key generation oracle can only outputs re-signing keys for admissible edge. We say the adversary wins the experiment if he can output a valid signature for a message of the target pair, where the adversary never queries signing and re-signing oracle about the message.

The definition of PRS scheme can be easily extended to a tag-based scheme where each signature is associated with a tag. Similarly, we can consider the *selective* model where the adversary needs to commit to the challenge tag at the beginning, or the adaptive model without this restriction. In Section 6.1, we focus on constructing tag-based multi-hop PRS in selective model. We note that the lattice-mixing and vanishing techniques in [Boy10] can help achieving adaptively chose tags.

In comparison with prior PRS security model. We briefly recall the four scenarios defined in [AH05, LV08a]. *External security* captures the scenario that the adversary is outside PRS system, where he only has access to signing and re-signing oracle. *Limited proxy security* means the adversary acts as the proxy, where he can sign message on behalf of the delegatee or create signatures for the delegator, unless the messages were first signed by one of the latter's delegatees. *Delegatee security* captures the scenario that the honest delegatee should be immune to a collusion between the delegators and proxy. *Delegator security* captures the scenario that collusions between the delegatees and proxy should be of no harm to the honest delegator.

Then we argue that our unified security model also captures the four desired scenarios above. For *external security*, since the adversary in our model is provided the oracle access to signing and re-signing oracle, the adversary can request the same information as that in the prior models. For the case of *limited proxy security*, our security model allows the adversary to receive re-signing keys and have access to the signing/re-signing oracles as the prior model, yet our selective security poses some constraints that the adversary needs to commit to all paths to the target pair. Our security model can emulate *delegator security* by restricting the committed tree from adversary to contain a single vertex r and its target level to be maximum level L. Then, any added edge (i, r, κ) except for $\kappa = L - 1$ is considered to be *admissible* for (r, L). Similarly, for *delegator security*, we can also emulate this by setting the target level to be 1.

6.3 Our PRS Construction

Now we present our PRS construction and its security proof sketch. For simplicity, we first present the scheme with security regarding a selective chosen tag, where in the security experiment, the adversary needs to commit to the challenge tag before obtaining public parameters and public keys. In the full version, we also describe how to modify our construction, slightly, to achieve security for adaptively chosen tags. Let the message space be $\mathcal{M} = \mathbb{Z}_q$, and the tag space be $\mathcal{T} = \mathbb{Z}_q$. The description is the following:

- $\mathsf{Setup}(1^\lambda, 1^L)$: The setup algorithm sets the lattice parameters (n, q, m, s), then randomly chooses a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and vectors $\boldsymbol{b}, \boldsymbol{v} \in \mathbb{Z}_q^n$. Output the public parameter $\mathsf{pp} = (\mathbf{A}, \boldsymbol{b}, \boldsymbol{v}, q, n, m)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: The key generation algorithm computes $(\mathsf{pk}_i, \mathsf{sk}_i)$ as follows:
  1. Sample two small matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2}$ from discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^{m \times m}, s}$.
  2. Compute $\mathbf{A}_i = \mathbf{A} \cdot \mathbf{R}_{i1} \bmod q$ and $\mathbf{A}'_i = \mathbf{A} \cdot \mathbf{R}_{i2} \bmod q$.
  3. The public key pk_i and secret key sk_i for i-th user is
  $$\mathsf{pk}_i = (\mathbf{A}_i, \mathbf{A}'_i), \qquad \mathsf{sk}_i = (\mathbf{R}_{i1}, \mathbf{R}_{i2})$$
- $\mathsf{Sign}(\mathsf{pp}, \mathsf{sk}_i, \mu, \kappa)$: The signing algorithm does:
  1. Randomly select a non-zero tag $t \in \mathbb{Z}_q^*$, and define the signing matrix to be
  $$\mathbf{F}_{t,i,\kappa} = [\mathbf{A} | \mathbf{A}_i + h(\kappa)\mathbf{G} | \mathbf{A}'_i + t\mathbf{G}]$$
  2. Sample a vector $\boldsymbol{r}_1 \leftarrow \mathcal{D}_{\mathbb{Z}^m, s}$, then sample vector $(\boldsymbol{r}_0, \boldsymbol{r}_2) \in \mathbb{Z}^{2m}$, using
  $$(\boldsymbol{r}_0, \boldsymbol{r}_2) \leftarrow \mathsf{Sample}^{\mathcal{O}}(\mathbf{A}, t\mathbf{G}, \mathbf{R}_{i2}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{b} + \mu\boldsymbol{v} - (\mathbf{A}'_i + h(\kappa)\mathbf{G})\boldsymbol{r}_1, s)$$
  Therefore, it holds that $\mathbf{F}_{t,i,\kappa} \cdot \sigma = \boldsymbol{b} + \mu\boldsymbol{v} \bmod q$, where $\sigma = (\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2)$.
  3. Output the signature (σ, t, i, κ).
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_i, \mu, (\sigma, t, i, \kappa))$: The verification algorithm dose:
  1. Parse the signature tuple as $\sigma = (\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2)$, tag t, user index i and level index κ, then first check the norm of $|\sigma| = |(\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2)|$. Output 0 if $|\sigma| \geq B$.
  2. Reconstruct the signing matrix
  $$\mathbf{F}_{t,i,\kappa} = [\mathbf{A} | \mathbf{A}_i + h(\kappa)\mathbf{G} | \mathbf{A}'_i + t\mathbf{G}]$$
  and output 1 if $\mathbf{F}_{t,i,\kappa} \cdot \sigma = \boldsymbol{b} + \mu\boldsymbol{v}$, otherwise output 0.

- $\mathsf{ReKeyGen}(\mathsf{pk}_i, (\mathsf{sk}_j, \mathsf{pk}_j), \kappa)$: The re-signing key generation:

  1. Sample small matrices $(\mathbf{X}_{01}, \mathbf{X}_{11}, \mathbf{X}_{02}, \mathbf{X}_{12})$, using

  $$(\mathbf{X}_{01}, \mathbf{X}_{11}) \leftarrow \mathsf{Sample}^{\mathcal{O}}(\mathbf{A}, h(\kappa+1)\mathbf{G}, \mathbf{R}_{j1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}_i + h(\kappa)\mathbf{G}, s),$$

  $$(\mathbf{X}_{02}, \mathbf{X}_{12}) \leftarrow \mathsf{Sample}^{\mathcal{O}}(\mathbf{A}, h(\kappa+1)\mathbf{G}, \mathbf{R}_{j1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}'_i - \mathbf{A}'_j, s)$$

  Therefore it holds that

  $$[\mathbf{A}|\mathbf{A}_j + h(\kappa+1)\mathbf{G}|\mathbf{A}'_j + t\mathbf{G}] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{01} & \mathbf{X}_{02} \\ 0 & \mathbf{X}_{11} & \mathbf{X}_{12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}|\mathbf{A}_i + h(\kappa)\mathbf{G}|\mathbf{A}'_i + t\mathbf{G}]$$

  2. Output the re-signing key $\mathsf{rk}^{\kappa}_{i\to j} = (\mathbf{X}_{01}, \mathbf{X}_{02}, \mathbf{X}_{11}, \mathbf{X}_{12})$.

- $\mathsf{ReSign}(\mathsf{rk}^{\kappa}_{i\to j}, (\sigma, t, i, \kappa), \mu, pk_i)$: The re-signing algorithm does:

  1. First parse $\sigma = (\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2)$. Output $\perp$ if $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_i, \mu, (\sigma, t, i, \kappa)) = 0$.
  2. Otherwise, output the re-signature tuple $(\sigma', t, j, \kappa+1)$, where $\sigma' = \mathsf{rk}^{\kappa}_{j\to j} \cdot \sigma$.

6.4 Security Proof

In this part, we show that our construction above is secure as in Definition 3.1 assuming the hardness of SIS assumption.

Theorem 6.1. *Assuming the hardness of $\mathsf{SIS}_{q,n,m,\beta}$, the PRS construction described above is existential unforgeable as in Definition 3.1.*

Intuition. At a high level, the reduction simulates the oracles described in Definition 3.1 using the SIS instance, where the answers output by these simulated oracles are statistically close to the answers in real execution. After querying these oracles, adversary would output a forgery tuple $(r, \mu^*, \sigma^*, \kappa^*)$, where the reduction derives the solution to SIS instance from the forgery tuple. Therefore, the probability of forging a signature for the PRS system is approximately the same as breaking the SIS assumption. In particular, the public key of the root user is generated by embedding the committed information of challenge level κ^* and tag t^*, and for generating public keys of the vertex on the tree, the reduction first sample re-signing key for edges (associated with levels as described in Definition 3.1), then the public key of vertex is a multiplication of re-signing key and its parent's public key. Generating public key for vertex i in this manner ensures the trapdoor using for computing re-signing keys dose not vanish except for its tree level $\ell(i)$. Therefore, for re-signing key queries with respect to admissible edges, the reduction can still obtain re-signing key from the distribution. The reduction generates the public key for other users that are not on the tree, using the KeyGen algorithm, thus the queries for re-signing keys between these users or re-signing keys towards these users are not restricted. For the signing query, the reduction can generate signatures from the same distribution as long as the tag is not the target one t^*, since the trapdoor for computing signatures dose not vanish. At the end of experiment, adversary $\mathcal{A}$ outputs a forgery tuple $(r, \mu^*, \sigma^*, t^*, \kappa^*)$, which vanishes the trapdoors in target user. Then, reduction can derive an SIS solution from the forgery tuple.

Proof. We show that, suppose there exists an adversary $\mathcal{A}$ that breaks the security as defined in Definition 3.1 with probability ϵ, then there exists a reduction $\mathcal{B}$ that breaks the underlying $\mathsf{SIS}_{q,n,m,\beta}$ assumption with probability $\epsilon + \mathsf{negl}(\lambda)$, which suffices to prove the theorem.

Invocation. Reduction $\mathcal{B}$ is invoked on an instance of the SIS assumption, i.e. $\mathbf{A}^* \in \mathbb{Z}_q^{n\times m}$, and is asked to return a solution $\boldsymbol{e} \in \mathbb{Z}_q^m$, such that $\mathbf{A}\cdot\boldsymbol{e} = 0 \bmod q$, and $0 \neq ||\boldsymbol{e}|| \leq \beta$. Adversary $\mathcal{A}$ sends a tree structure $T = (V, E)$, a target level $\kappa^* \leq L$ and the challenge tag $t^* \in \mathbb{Z}_q$, where the depth of the tree is less than L, and the root r is treated as the target user.

Setup. Reduction $\mathcal{B}$ uses the SIS instance $\mathbf{A}^*$ to setup the PRS system as follows:

1. Set the lattice parameters q, n, m according to the $\mathsf{SIS}_{q,n,m,\beta}$ problem, and Gaussian parameter $s = s(n)$. Initialize set $Q = \emptyset$ for signing queries, $\mathcal{H} = V$ for honest users, $\mathcal{C} = \emptyset$ for corrupt users, $K = \emptyset$ for generated re-signing keys. Set graph $G = (E', V')$, where edge set $E' = E$ and vertex set $V' = V$.
2. Randomly choose small vectors $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \mathcal{D}_{\mathbb{Z}^m,s}$, and set $(\mathbf{A}, \boldsymbol{b}, \boldsymbol{v})$ in public parameters pp as
$$\mathbf{A} = \mathbf{A}^*, \qquad \boldsymbol{b} = \mathbf{A}^*\boldsymbol{s}_1, \qquad \boldsymbol{v} = \mathbf{A}^*\boldsymbol{s}_2$$
3. Run a Deep-First traversal to associated each edge with a level index $\ell(i)$, by decreasing from root user's target level κ^*, and update the edge set by augmenting each edge $(i, j) \in E$ with its level index $(i, j, \ell(i))$. Update the edge E' correspondingly.
4. Sample small matrices $\mathbf{R}_{r1}, \mathbf{R}_{r2} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},s}$, and set matrices $\mathbf{A}_r, \mathbf{A}'_r$ as
$$\mathbf{A}_r = \mathbf{A}\mathbf{R}_{r1} - h(\kappa^*)\mathbf{G}, \quad \mathbf{A}'_r = \mathbf{A}\mathbf{R}_{r2} - t^*\mathbf{G}$$
5. For node i which is the child node of the root node r, sample matrices $\mathbf{X}_{i,01}, \mathbf{X}_{i,02}, \mathbf{X}_{i,11}, \mathbf{X}_{i,12} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},s}$, such that
$$[\mathbf{A}|\mathbf{A}_r + h(\kappa^*)\mathbf{G}|\mathbf{A}'_r] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{i,01} & \mathbf{X}_{i,02} \\ 0 & \mathbf{X}_{i,11} & \mathbf{X}_{i,12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}|\mathbf{A}_i + h(\ell(i))\mathbf{G}|\mathbf{A}'_i]$$
where matrices $(\mathbf{A}_i, \mathbf{A}'_i)$ are set as
$$\mathbf{A}_i = \mathbf{A}\mathbf{R}_{i1} - h(\ell(i))\mathbf{G} = \mathbf{A}(\mathbf{X}_{i,01} + \mathbf{R}_{r1}\mathbf{X}_{i,11}) - h(\ell(i))\mathbf{G}$$
$$\mathbf{A}'_i = \mathbf{A}\mathbf{R}_{r2} - t\mathbf{G} = \mathbf{A}(\mathbf{X}_{i,02} + \mathbf{R}_{r1}\mathbf{X}_{j,12} + \mathbf{R}_{r2}) - t^*\mathbf{G}$$
and the re-signing key $\mathsf{rk}_{i\to r}^{\ell(i)} = (\mathbf{X}_{i,01}, \mathbf{X}_{i,02}, \mathbf{X}_{i,11}, \mathbf{X}_{i,12})$. For child j of node i, we can generate the public key $(\mathbf{A}_j, \mathbf{A}'_j)$ and re-signing key $\mathsf{rk}_{j\to i}^{\ell(j)}$ using the same procedure. Put all the generated re-signing keys into set K.
6. Send the public parameter pp and public keys $\{\mathsf{pk}_i\}_{i\in V}$ back to adversary $\mathcal{A}$.

Oracle simulation. Reduction $\mathcal{B}$ simulates the uncorrupt key generation, corrupt key generation, signing, re-signing and re-signing key generation oracles as follows:

- **Uncorrupt key generation oracle**: Sample small matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},s}$, and set $\mathbf{A}_i = \mathbf{A}\cdot\mathbf{R}_{i1} \bmod q$ and $\mathbf{A}'_i = \mathbf{A}\cdot\mathbf{R}_{i2} \bmod q$. Then send $\mathsf{pk}_i = (\mathbf{A}_i, \mathbf{A}'_i)$ to adversary $\mathcal{A}$, and update sets $\mathcal{H} = \mathcal{H} \cup \{i\}, V' = V' \cup \{i\}$.
- **Corrupt key generation oracle**: Sample small matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},s}$, and set $\mathbf{A}_i = \mathbf{A}\cdot\mathbf{R}_{i1} \bmod q$ and $\mathbf{A}'_i = \mathbf{A}\cdot\mathbf{R}_{i2} \bmod q$. Then send $\mathsf{pk}_i = (\mathbf{A}_i, \mathbf{A}'_i), \mathsf{sk}_i = (\mathbf{R}_{i1}, \mathbf{R}_{i2})$ to adversary $\mathcal{A}$, and update sets $\mathcal{C} = \mathcal{C} \cup \{i\}, V' = V' \cup \{i\}$.

- **Signing oracle**: On input a user index i, a message μ and a level index κ, oracle $\mathcal{O}_{\mathsf{Sign}}$ first selects a non-zero tag $t \in \mathbb{Z}_q^*$, such that $t \neq t^*$, and defines the signing matrix to be

$$\mathbf{F}_{t,i,\kappa} = [\mathbf{A}|\mathbf{A}_i + h(\kappa)\mathbf{G}|\mathbf{A}'_i + t\mathbf{G}]$$

Then sample a vector $\boldsymbol{r}_1 \leftarrow \mathcal{D}_{\mathbb{Z}^m,s}$, then sample vector $(\boldsymbol{r}_0, \boldsymbol{r}_2) \in \mathbb{Z}_q^{2m}$, using

$$(\boldsymbol{r}_0, \boldsymbol{r}_2) \leftarrow \mathsf{SampleRight}(\mathbf{A}, t\mathbf{G}, \mathbf{R}_{i2}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{b} + \mu\boldsymbol{v} - (\mathbf{A}'_i + h(\kappa)\mathbf{G})\boldsymbol{r}_1, s)$$

Therefore, it holds that $\mathbf{F}_{t,i,\kappa} \cdot \sigma = \boldsymbol{b} + \mu\boldsymbol{v} \bmod q$, where $\sigma = (\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2)$. Oracle $\mathcal{O}_{\mathsf{Sign}}$ outputs $(\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2, t)$ and updates $Q = Q \cup \{(i, \mu, \kappa)\}$ if $i \in \mathcal{H}$.

- **Re-signing oracle**: On input user index i, j, a message/signature pair (μ, σ) and level index κ, oracle $\mathcal{O}_{\mathsf{ReSign}}$ outputs $\perp$ if $\mathsf{Verify}(\mathsf{pk}_i, \mu, \sigma, \kappa) = 0$. Otherwise,
  - If the edge $(i, j, \kappa) \in E'$, then find re-signing key $\mathsf{rk}^{\kappa}_{i\to j}$ in set K. Output re-signature $(\mathsf{rk}^{\kappa}_{i\to j} \cdot \sigma, \kappa + 1)$ to adversary. Then update the set $Q = Q \cup \{(j, \mu, \kappa + 1)\}$.
  - Otherwise if the edge $(i, j, \kappa) \notin E', j \in V$ and $\kappa = \ell(j) - 1$, then output $\perp$. Otherwise, call re-signing key generation oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ described below, and output re-signature $(\mathsf{rk}^{\kappa}_{i\to j} \cdot \sigma, \kappa + 1)$ to adversary. Update $Q = Q \cup \{(j, \mu, \kappa + 1)\}$.
- **Re-signing key generation oracle**: On input user index i, j and level index κ, oracle $\mathcal{O}_{\mathsf{ReKeyGen}}$ outputs $\perp$ if the edge (i, j, κ) is not an admissible edge with respect to edge set E and target user/level (r, κ^*). Otherwise,
  - If $(i, j, \kappa) \in E'$, then look for the re-signing key $\mathsf{rk}^{\kappa}_{i\to j}$ already generated in set K and send $\mathsf{rk}^{\kappa}_{i\to j}$ back to adversary $\mathcal{A}$.
  - Otherwise, we divide into two cases: (1) $j \in V, \kappa \neq \ell(j) - 1$; (2) $j \notin V$. For these two cases, the trapdoor used for sampling re-signing key dose not vanish, thus sample small matrices $\mathsf{rk}^{\kappa}_{i\to j} = (\mathbf{X}_{i,01}, \mathbf{X}_{i,11}, \mathbf{X}_{i,02}, \mathbf{X}_{i,12})$, using

$$(\mathbf{X}_{i,01}, \mathbf{X}_{i,11}) \leftarrow \mathsf{SampleRight}(\mathbf{A}, h(\kappa+1)\mathbf{G}, \mathbf{R}_{j1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}_i + h(\kappa)\mathbf{G}, s),$$

$$(\mathbf{X}_{i,02}, \mathbf{X}_{i,12}) \leftarrow \mathsf{SampleRight}(\mathbf{A}, h(\kappa+1)\mathbf{G}, \mathbf{R}_{j1}, \mathbf{T}_{\mathbf{G}}, \mathbf{A}'_i - \mathbf{A}'_j, s)$$

Therefore it holds that

$$[\mathbf{A}|\mathbf{A}_j + h(\kappa+1)\mathbf{G}|\mathbf{A}'_j] \begin{bmatrix} \mathbf{I} & \mathbf{X}_{i,01} & \mathbf{X}_{i,02} \\ 0 & \mathbf{X}_{i,11} & \mathbf{X}_{i,12} \\ 0 & 0 & \mathbf{I} \end{bmatrix} = [\mathbf{A}|\mathbf{A}_i + h(\kappa)\mathbf{G}|\mathbf{A}'_i]$$

Then update $E' = E' \cup \{(i, j, \kappa)\}, K = K \cup \{\mathsf{rk}^{\kappa}_{i\to j}\}$ and send $\mathsf{rk}^{\kappa}_{i\to j}$ to adversary $\mathcal{A}$.

Forgery. Reduction $\mathcal{B}$ receives from adversary $\mathcal{A}$ a forgery tuple $(i^*, \mu^*, \sigma^*, \kappa^*)$, and construct an solution to SIS instance $\mathbf{A}^*$ as follows:

1. Derive the inadmissible set Q' from set Q with respect to edge set E' and target user/level (r, κ^*). Output $\perp$ if $(i^*, \mu^*, \kappa^*) \in Q'$ or $i^* \neq r$.
2. Otherwise parse the signature $\sigma^* = (\boldsymbol{r}_0^*, \boldsymbol{r}_1^*, \boldsymbol{r}_3^*)$ construct the signing matrix $\mathbf{F}_{t,r,\kappa^*}$ as

$$\mathbf{F}_{t,r,\kappa^*} = [\mathbf{A}|\mathbf{A}_r + h(\kappa^*)\mathbf{G}|\mathbf{A}'_r + t^*\mathbf{G}] = [\mathbf{A}^*|\mathbf{A}^*\mathbf{R}_{r1}|\mathbf{A}^*\mathbf{R}_{r2}]$$

Thus, per correctness, it holds that

$$\mathbf{A}^*\boldsymbol{r}_0^* + \mathbf{A}^*\mathbf{R}_{r1}\boldsymbol{r}_1^* + \mathbf{A}^*\mathbf{R}_{r2}\boldsymbol{r}_2^* = \mathbf{A}^*\boldsymbol{s}_1 + \mu^*\mathbf{A}^*\boldsymbol{s}_2$$

Re-arranging, we have

$$\mathbf{A}^*(\boldsymbol{r}_0^* + \mathbf{R}_{r1}\boldsymbol{r}_1^* + \mathbf{R}_{r2}\boldsymbol{r}_2^* - \boldsymbol{s}_1 - \mu^*\boldsymbol{s}_2) = 0$$

Since matrices $\mathbf{R}_{r1}, \mathbf{R}_{r2}$ and vectors $\boldsymbol{s}_1, \boldsymbol{s}_2$ are independent of each other, and hidden from adversary A's view, then with overwhelming probability $(\boldsymbol{r}_0^*+\mathbf{R}_{r1}\boldsymbol{r}_1^*+\mathbf{R}_{r2}\boldsymbol{r}_2^*-\boldsymbol{s}_1-\mu^*\boldsymbol{s}_2) \neq 0$.

3. Output $(\boldsymbol{r}_0^* + \mathbf{R}_{r1}\boldsymbol{r}_1^* + \mathbf{R}_{r2}\boldsymbol{r}_2^* - \boldsymbol{s}_1 - \mu^*\boldsymbol{s}_2)$ as the solution for SIS instance $\mathbf{A}^*$.

Next, we show that our reduction works by the following claim:

Claim 6.2. *The probability of successfully outputting a valis solution to* $\mathsf{SIS}_{q,n,m,\beta}$ *by reduction* $\mathcal{B}$ *is*

$$\mathbf{Pr}[\mathcal{B} \textit{ breaks } \mathsf{SIS}] \geq \frac{c}{N}\mathbf{Pr}[\mathcal{A} \textit{ breaks PRS}] + \mathsf{negl}(\lambda)$$

where N is the number of users in the PRS system, and c is a constant number.

Proof. We first analysis the indistinguishability of the real execution and the simulated experiment. Let pp be the public parameters, $\{\sigma_i\}_i$ be the signatures replied to adversary $\mathcal{A}$'s signing queries, $\{\mathsf{pk}_i\}_{i\in\mathcal{H}}$ be the public key for honest users, $\{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in\mathcal{C}}$ be the public/secret key pairs for corrupt users, and $\{\mathsf{rk}_{i\to j}^{\kappa}\}$ be the re-signing keys sent to adversary $\mathcal{A}$. We show the distribution

$$\{\mathsf{pp}, \{\sigma_i\}_i, \{\mathsf{pk}_i\}_{i\in\mathcal{H}}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in\mathcal{C}}, \{\mathsf{rk}_{i\to j}^{\kappa}\}, (i, \kappa, \mu, \sigma)_{(i,\kappa,\mu)\in Q}\}$$

in real execution and simulated experiment are statistically indistinguishable. The difference between real execution and simulated experiment are summarized as follows:

1. In real Setup, matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and vector $\boldsymbol{b}, \boldsymbol{v} \in \mathbb{Z}_q^n$ are chosen at random, while in the simulated setup, matrix $\mathbf{A}$ is chosen uniformly random (by the average-case SIS generator), and vector $\boldsymbol{b}, \boldsymbol{v}$ are set respectively as $\boldsymbol{b} = \mathbf{A}^*\boldsymbol{s}_1, \boldsymbol{v} = \mathbf{A}^*\boldsymbol{s}_2$, where $\boldsymbol{s}_1, \boldsymbol{s}_2 \leftarrow \mathcal{D}_{\mathbb{Z}^m,s}$.

2. In real KeyGen, for each user i, the public key $\mathsf{pk}_i = (\mathbf{A}_i, \mathbf{A}_i')$ is generated by first sampling matrices $\mathbf{R}_{i1}, \mathbf{R}_{i2}$ from distribution $\mathcal{D}_{\mathbb{Z}^{m\times m},s}$, and setting $\mathbf{A}_{i1} = \mathbf{A}\cdot\mathbf{R}_{i1}, \mathbf{A}_{i2} = \mathbf{A}\cdot\mathbf{R}_{i2}$. In simulated experiment, as we described above, the generation of public key for user i is divided into the following three cases:

   - If $i = r$, then the public key $\mathsf{pk}_r = (\mathbf{A}_r, \mathbf{A}_r')$ is set to be $\mathbf{A}_r = \mathbf{A}\mathbf{R}_{r1} - h(\kappa^*)\mathbf{G}$, $\mathbf{A}_r' = \mathbf{A}\mathbf{R}_{r2} - t^*\mathbf{G}$, where $\mathbf{R}_{r1}, \mathbf{R}_{r2} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m},s}$.
   - If $i \in V - \{r\}$, we first run a deep-first traversal starting from root level with target level κ^*, and then sample the re-signing key $\mathsf{rk}_{i\to j}^{\kappa} = (\mathbf{X}_{i,01}, \mathbf{X}_{i,11}, \mathbf{X}_{i,02}, \mathbf{X}_{i,12})$, where j is vertex i's parent node, and $\ell(i)$ is i's associated level. The public key $\mathsf{pk}_i = (\mathbf{A}_i, \mathbf{A}_i')$ is set to be

   $$\mathbf{A}_i = \mathbf{A}\mathbf{X}_{i,01} + \mathbf{A}_j\mathbf{X}_{i,11} - h(\ell(i))\mathbf{G}, \quad \mathbf{A}_i' = \mathbf{A}\mathbf{X}_{i,02} + \mathbf{A}_j\mathbf{X}_{i,12} + \mathbf{A}_j' - t^*\mathbf{G}$$

   - Otherwise, the public/secret keys are generated in the same manner as the real scheme.

3. For edge $(i, j, \kappa) \in E'$, in real $\mathsf{ReKeyGen}$, re-signing key $\mathsf{rk}_{i\to j}^{\kappa} = (\mathbf{X}_{i,01}, \mathbf{X}_{i,02}, \mathbf{X}_{i,11}, \mathbf{X}_{i,12})$ is generated using algorithm $\mathsf{SampleRight}$ along with gadget matrix $\mathbf{G}$'s trapdoor. Whereas, in the reduction, the generation of re-signing key $\mathsf{rk}_{i\to j}^{\kappa}$ is divided into the following two cases:

   - If $(i, j, \kappa) \in E$, then the re-signing key $\mathsf{rk}_{i\to j}^{\kappa}$ is sampled from Gaussian distribution $\mathcal{D}_{\mathbb{Z}^{m\times m},s}$ in the generation of public key pk_i as we described above.

- If $(i, j, \kappa) \in E' - E$, then the re-signing key $\mathsf{rk}_{i\to j}^{\kappa}$ is generated in the same manner as the real execution, i.e. using algorithm $\mathsf{SampleRight}$, as the trapdoor dose not vanish in the public key.

4. For signing query (i, μ, κ), in real Sign, signature of the node i at level κ is generated by first randomly selecting a tag t and a short vector $\boldsymbol{r}_1$, and then computing the other two vectors using algorithm $\mathsf{SampleRight}$ along with gadget matrix $\mathbf{G}$'s trapdoor. While in the simulation, the oracle first randomly samples a tag $t \neq t^*$, and then uses the same procedure to generate the signature, since the gadget matrix $(t - t^*)\mathbf{G}$'s trapdoor dose not vanish.

5. The re-signing process in real execution and reduction are the same, i.e. multiplying signatures with the corresponding re-signing key.

We now argue the following distribution in reduction is statistically indistinguishable from that in simulated execution:

$$\{\mathsf{pp}, \{\sigma_i\}_i, \{\mathsf{pk}_i\}_{i\in\mathcal{H}}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in\mathcal{C}}, \{\mathsf{rk}_{i\to j}^{\kappa}\}, (i, \kappa, \mu, \sigma)_{(i,\kappa,\mu)\in Q}\}$$

. By Leftover Hash Lemma 2.2, we have the following

$$(\mathbf{A}, \boldsymbol{b}, \boldsymbol{v}, \mathbf{A}\mathbf{R}_{i1}, \mathbf{R}_{i2}) \approx (\mathbf{A}^*, \mathbf{A}^*\boldsymbol{s}_1, \mathbf{A}^*\boldsymbol{s}_2, \mathbf{A}\mathbf{R}_{i1} - h(\kappa^*)\mathbf{G}, \mathbf{A}\mathbf{R}_{i2} - t^*\mathbf{G})$$

where $(\boldsymbol{s}_1, \boldsymbol{s}_2) \leftarrow \mathcal{D}_{\mathbb{Z}^m, s}, (\mathbf{R}_{i1}, \mathbf{R}_{i2}) \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m}, s}$. Thus the public parameters pp and public key pk_r in both executions are statistically close. For user $i \in V' - \{r\}$, the public key $\mathsf{pk}_i = (\mathbf{A}_i, \mathbf{A}'_i)$ is set to be

$$\mathbf{A}_i = \mathbf{A}\mathbf{X}_{i,01} + \mathbf{A}_j\mathbf{X}_{i,11} - h(\ell(i))\mathbf{G}, \quad \mathbf{A}'_i = \mathbf{A}\mathbf{X}_{i,02} + \mathbf{A}_j\mathbf{X}_{i,12} + \mathbf{A}'_j - t^*\mathbf{G}$$

where $\mathbf{X}_{i,01}, \mathbf{X}_{i,02}, \mathbf{X}_{i,11}, \mathbf{X}_{i,12} \leftarrow \mathcal{D}_{\mathbb{Z}^{m\times m}, s}$. Therefore, the public keys $\{\mathsf{pk}_i\}_{i\in V-\{r\}}$ in the reduction are statistically indistinguishable from those in real execution by appropriate parameter setting as in Section 6.5. For other users $i \in V' - V$, the public key are generated using the same procedure.

For edge $(i, j, \kappa) \in E$, the re-signing key $\mathsf{rk}_{i\to j}^{\kappa}$ are sample directly from Gaussian distribution in reduction, thus it is statistically close to that in real execution by the property of algorithm $\mathsf{SampleRight}$ in Lemma 2.6. For edge $(i, j, \kappa) \in E' - E$, the re-signing key $\mathsf{rk}_{i\to j}^{\kappa}$ in reduction is generated in the same manner as in the real execution. Therefore, the distribution $\{\mathsf{pp}, \{\sigma_i\}_i, \{\mathsf{pk}_i\}_{i\in\mathcal{H}}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in\mathcal{C}}, \{\mathsf{rk}_{i\to j}^{\kappa}\}\}$ in reduction is statistically close to that in real execution.

Next, for signature query (i, κ, μ) and re-signing query $(i, j, \kappa, \mu, \sigma)$, as we stated above, the computing process in reduction and real execution are the same. Therefore, we prove that adversary $\mathcal{A}$'s view in reduction and real execution are statistically close. □

Combining the reduction and Claim 6.2, we conclude the security proof. □

6.5 Parameter Setting

Let λ be the security parameter. For $L = \mathsf{polylog}(\lambda)$ maximum allowed re-signing, we set the parameters of our scheme based on standard SIS assumption as

$$q = n^{O(L)}, \quad n = \mathsf{poly}(\lambda), \quad L = \mathsf{polylog}(\lambda), \quad m = O(n \log q)$$

To ensure the SIS instance has a worst-case lattice reduction as shown in [MR04], i.e. $q \geq \beta\omega(\sqrt{n \log n})$, we set $\beta = \mathsf{polylog}(n)$. In order to achieve indistinguishability between real execution and reduction, the Gaussian parameter is set to be $s = \omega(\sqrt{\log n})$. As a signature produced by algorithm Sign has the size of $O(s\sqrt{m})$, and after each re-signing, the size grows at the rate of $O(sm)$, so we set parameter used in verification to be $B = \omega(2^L)$. Our PRS construction can support $L = \mathsf{poly}(\lambda)$-hop using subexponential SIS assumption.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Gilbert [Gil10], pages 553–572.

[ABF+13] Joël Alwen, Manuel Barbosa, Pooya Farshim, Rosario Gennaro, S Dov Gordon, Stefano Tessaro, and David A Wilson. On the relationship between functional encryption, obfuscation, and fully homomorphic encryption. In *Cryptography and Coding*, pages 65–84. Springer, 2013.

[ABPW13] Yoshinori Aono, Xavier Boyen, Le Trieu Phong, and Lihua Wang. Key-private proxy re-encryption under LWE. In Goutam Paul and Serge Vaudenay, editors, *INDOCRYPT 2013*, volume 8250 of *LNCS*, pages 1–18. Springer, Heidelberg, December 2013.

[AFGH05] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *NDSS 2005*. The Internet Society, February 2005.

[AH05] Giuseppe Ateniese and Susan Hohenberger. Proxy re-signatures: New definitions, algorithms, and applications. In Vijayalakshmi Atluri, Catherine Meadows, and Ari Juels, editors, *ACM CCS 05*, pages 310–319. ACM Press, November 2005.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In Pointcheval and Johansson [PJ12], pages 483–501.

[Ajt99] Miklós Ajtai. Determinism versus non-determinism for linear time RAMs (extended abstract). In *31st ACM STOC*, pages 632–641. ACM Press, May 1999.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 127–144. Springer, Heidelberg, May / June 1998.

[Boy10] Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 499–517. Springer, Heidelberg, May 2010.

[CCL+14] Nishanth Chandran, Melissa Chase, Feng-Hao Liu, Ryo Nishimaki, and Keita Xagawa. Re-encryption, functional re-encryption, and multi-hop re-encryption: A framework for achieving obfuscation-based security and instantiations from lattices. In Krawczyk [Kra14], pages 95–112.

[CCV12] Nishanth Chandran, Melissa Chase, and Vinod Vaikuntanathan. Functional re-encryption and collusion-resistant obfuscation. In Ronald Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 404–421. Springer, Heidelberg, March 2012.

[CH07] 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 CCS 07*, pages 185–194. ACM Press, October 2007.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

[CP08] Sherman S. M. Chow and Raphael C.-W. Phan. Proxy re-signatures in the standard model. In Tzong-Chen Wu, Chin-Laung Lei, Vincent Rijmen, and Der-Tsai Lee, editors, *ISC 2008*, volume 5222 of *LNCS*, pages 260–276. Springer, Heidelberg, September 2008.

[CWYD10] Sherman SM Chow, Jian Weng, Yanjiang Yang, and Robert H Deng. Efficient unidirectional proxy re-encryption. In *Progress in Cryptology–AFRICACRYPT 2010*, pages 316–332. Springer, 2010.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194. Springer, Heidelberg, August 1987.

[GGH13] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 2013.

[Gil10] Henri Gilbert, editor. *EUROCRYPT 2010*, volume 6110 of *LNCS*. Springer, Heidelberg, May 2010.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[HRsV07] Susan Hohenberger, Guy N. Rothblum, abhi shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 233–252. Springer, Heidelberg, February 2007.

[Kir14] Elena Kirshanova. Proxy re-encryption from lattices. In Krawczyk [Kra14], pages 77–94.

[Kra14] Hugo Krawczyk, editor. *PKC 2014*, volume 8383 of *LNCS*. Springer, Heidelberg, March 2014.

[LL15] Kim Laine and Kristin Lauter. Key recovery for LWE in polynomial time. Cryptology ePrint Archive, Report 2015/176, 2015. `http://eprint.iacr.org/2015/176`.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Gilbert [Gil10], pages 1–23.

[LV08a] Benoît Libert and Damien Vergnaud. Multi-use unidirectional proxy re-signatures. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08*, pages 511–520. ACM Press, October 2008.

[LV08b] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In Ronald Cramer, editor, *PKC 2008*, volume 4939 of *LNCS*, pages 360–379. Springer, Heidelberg, March 2008.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In Pointcheval and Johansson [PJ12], pages 700–718.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of SIS and LWE with small parameters. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 21–39. Springer, Heidelberg, August 2013.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004.

[NAL15] David Nunez, Isaac Agudo, and Javier Lopez. A parametric family of attack models for proxy re-encryption. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th*, pages 290–301. IEEE, 2015.

[PJ12] David Pointcheval and Thomas Johansson, editors. *EUROCRYPT 2012*, volume 7237 of *LNCS*. Springer, Heidelberg, April 2012.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[SCL10] Jun Shao, Zhenfu Cao, and Peng Liu. CCA-Secure PRE scheme without random oracles. Cryptology ePrint Archive, Report 2010/112, 2010. `http://eprint.iacr.org/2010/112`.

[SFZ+10] Jun Shao, Min Feng, Bin Zhu, Zhenfu Cao, and Peng Liu. The security model of unidirectional proxy re-signature with private re-signature key. In *Australasian Conference on Information Security and Privacy*, pages 216–232. Springer, 2010.

[Smi05] Tony Smith. *DVD Jon: Buy DRM-less Tracks from Apple iTunes*, 2005. `http://www.theregister.co.uk/2005/03/18/itunes_pymusique/`.