Lattice-based Revocable (Hierarchical) IBE with Decryption Key Exposure Resistance

Shuichi Katsumata[1,2], Takahiro Matsuda[2], and Atsushi Takayasu[1,2]

[1] The University of Tokyo, Tokyo, Japan,
[2] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan
takayasu@mist.i.u-tokyo.ac.jp

Abstract. *Revocable* identity-based encryption (RIBE) is an extension of IBE that supports a key revocation mechanism, which is an indispensable feature for practical cryptographic schemes. Due to this extra feature, RIBE is often required to satisfy a strong security notion unique to the revocation setting called *decryption key exposure resistance* (DKER). Additionally, *hierarchal* IBE (HIBE) is another orthogonal extension of IBE that supports key delegation functionalities allowing for scalable deployments of cryptographic schemes. So far, R(H)IBE constructions with DKER are only known from bilinear maps, where all constructions rely heavily on the so-called *key re-randomization* property to achieve the DKER and/or hierarchal feature. Since lattice-based schemes seem to be inherently ill-fit with the key re-randomization property, no construction of lattice-based R(H)IBE schemes with DKER are known.
In this paper, we propose the first lattice-based RHIBE scheme with DKER *without* relying on the key re-randomization property, departing from all the previously known methods. We start our work by providing a generic construction of RIBE schemes with DKER, which uses as building blocks any two-level standard HIBE scheme and (weak) RIBE scheme *without* DKER. Based on previous lattice-based RIBE constructions *without* DKER, our result implies the first lattice-based RIBE scheme *with* DKER. Then, building on top of our generic construction, we construct the first lattice-based RHIBE scheme with DKER, by further exploiting the algebraic structure of lattices. To this end, we prepare a new tool called the *level conversion keys*, which enables us to achieve the hierarchal feature without relying on the key re-randomization property.

1 Introduction

Identity-based encryption (IBE) is an advanced form of public key encryption, where an arbitrary string can be used as user's public keys. One extension of IBE is *hierarchical* IBE (HIBE), which further supports a key delegation functionality; an attractive feature for scalable deployments of IBE. However, as opposed to ordinary public key encryption, (H)IBE does not support a key/user revocation mechanism due to the absence of the public key infrastructures and there are no trivial ways to drive malicious users out from an ordinary (H)IBE

system. Therefore, adding a key revocation mechanism to (H)IBE is considered to be one of the important research themes when considering practical deployments of (H)IBE. For instance, Boneh and Franklin [7] proposed a method for adding a simple revocation mechanism to any IBE system. However, the bottleneck of their proposal was its efficiency. The number of keys generated for every time period was proportional to the number of all users in the IBE system and the scheme did not scale if the number of users became too large. Since then, constructing an (H)IBE scheme with a scalable revocation mechanism has been a sought-after goal. Below, we refer to (H)IBE that allows for such a scalable revocation mechanism as *revocable (H)IBE*.

The first revocable IBE (RIBE) scheme was proposed by Boldyreva et al. [6]. RIBE requires three types of keys: a *secret key*, a *key update*, and a *decryption key*. As in IBE, each user is issued a secret key that is associated with his identity. However, in order to achieve the key revocation mechanism, each user's secret key itself does not allow them to decrypt ciphertexts. To allow the users to decrypt, the key generation center (KGC) broadcasts *key updates* for every time period through a public channel. Roughly, the key update incorporates public information of the users that are currently allowed in the system. Specifically, although the key update is meaningless information to revoked users, it allows non-revoked users to combine with their secret keys to derive a *decryption key*, which effectively enables them to properly decrypt ciphertexts. To achieve a scalable revocation mechanism, Boldyreva et al. utilized a subset cover framework called the complete subtree (CS) method [25], so that the size of the key update sent by the KGC in each time period will be logarithmic in the number of system users. The work of Boldyreva et al. [6] attracted numerous followup works [15,18,20,33,37] and their RIBE construction was also extended to revocable *hierarchical* IBE (RHIBE) which simultaneously support scalable key revocation and key delegation functionalities [13,17,19,32,34,35].

Considering that RIBE and RHIBE were introduced by envisioning the real-world use of (H)IBE systems, their security definitions should take into account as many realistic threats and attack scenarios as possible. For example, leakage of decryption keys due to social/cyber attacks or unexpected human errors are common incidents in practice. Motivated by this, Seo and Emura [33,35] introduced a security notion unique to R(H)IBE called *decryption key exposure resistance* (DKER). Roughly speaking, this security notion guarantees that an exposure of a user's decryption key at some time period will not compromise the confidentiality of ciphertexts that are encrypted for different time periods — a clearly desirable security guarantee in practice. After the introduction of the new security notion DKER, it has quickly become one of the default security requirements for R(H)IBE and attracted many followup works concerning R(H)IBE schemes with DKER [13,15,17,18,19,23,28,29,32,35,37]. So far constructions of R(H)IBE schemes with DKER are all based on bilinear or multilinear maps.

State of affairs of Lattice-based R(H)IBE. Lattice-based cryptography has been paid much attention in the last decade, however, construction of R(H)IBE schemes with DKER has been rather elusive. In 2012, Chen et al. [10] proposed

the first lattice-based RIBE scheme *without* DKER; a work before the now default security notion of DKER was formalized by Seo and Emura [33], building on top of the standard IBE constructions of [1,8]. The only followup work was done recently by Takayasu and Watanabe [36] who partially solved the problem of achieving RIBE with DKER by proposing a variant of [10]. Unfortunately, their scheme only satisfies *bounded* DKER, a strictly weaker notion than DKER, which only allows a bounded number of decryption keys to be leaked. Therefore, constructing an RIBE scheme with (unbounded) DKER based on lattices still remains an unsolved problem. This is in sharp contrast with the bilinear map setting where many constructions are known [13,15,17,18,19,32,33,35,37]. Moreover, extending the RIBE scheme of Chen et al. [10] to the hierarchal setting seems to be highly non-trivial since no construction of lattice-based RHIBEs are known regardless of the scheme being DKER or not.

One of the main reasons why constructing R(H)IBE schemes with DKER in the lattice-setting has been difficult is because the algebraic structure of lattices seems to be ill-fit with the so-called *key re-randomization* property. So far, all RIBE schemes [15,18,23,29,33,37] and RHIBE schemes with DKER [13,17,19,28,32,35] are based on number theoretical assumptions, e.g., bilinear maps and multilinear maps, which all rely heavily on this key re-randomization property. At a high level, this is the property with which each user can re-randomize their key so that the re-randomized key is distributed identically to (or at least statistically close to) a key generated using a fresh randomness. In essence, this is the central property that enables DKER. Furthermore, this property is also heavily utilized when generating the children's secret keys for fixed randomness without using any secret information, hence, achieving the hierarchal feature. However, unfortunately, due to the difference in the algebraic structure of bilinear, multilinear maps and lattices, we are currently unaware of any way of achieving the key re-randomization property from lattices.[3] Therefore, to construct lattice-based R(H)IBE schemes with DKER, it seems that we must deviate from prior methodologies and develop new techniques.

Our Contributions. In this paper, we propose the first lattice-based R(H)IBE scheme with DKER secure under the learning with errors (LWE) assumption. The techniques used in this work highly depart from previous works that rely on the key re-randomization property for achieving DKER and the key delegation functionality. Specifically, we show a generic construction of an RIBE scheme with DKER from any two-level standard HIBE scheme and RIBE scheme without DKER, thus bypassing the necessity of the key re-randomization property. Then, building on top of the idea of our generic construction, we further exploit the algebraic structure of lattices to construct an RHIBE scheme with DKER.

[3] A knowledgeable reader familiar with lattice-based cryptography may wonder why the existing RIBE schemes [10,36] cannot be easily modified to support the property by using short trapdoor bases. We provide detailed discussions on why this simple modification is insufficient in Section 2.

We provide a brief summary of our work below and refer the detailed technical overview to Section 2.

Our first contribution is a generic construction of RIBE *with* DKER from any RIBE *without* DKER and two-level HIBE. The new tools we introduce to circumvent the necessity of the key re-randomization property are called *leveled ciphertexts* and *leveled decryption keys*. At a high level, each "level" for the leveled ciphertexts and decryption keys is associated to the RIBE scheme without DKER and the two-level HIBE scheme, respectively; one level is responsible for achieving the revocation mechanism and the other is responsible for the key re-randomization mechanism. Therefore, informally, our leveled structure allows for a *partial* key re-randomization mechanism. Using the lattice-based RIBE scheme without DKER of Chen et al. [10] and any lattice-based HIBE scheme, e.g., [1,8], our result implies the first lattice-based RIBE scheme with DKER. Furthermore, since any IBE schemes can be converted to an HIBE scheme [12] (in the selective-identity model) and any RIBE scheme without DKER implies an IBE scheme, our result also implies a generic conversion of any RIBE scheme without DKER into an RIBE scheme with DKER.

Our second contribution is the construction of the first lattice-based RHIBE scheme with DKER. It is built on top of the idea of our generic construction and further exploits the algebraic structure unique to lattices. Namely, to achieve the key delegation functionality, i.e., hierarchal feature, we additionally introduce a tool called *level conversion keys*. In essence, this tool enables a user to convert his (secret) decryption key to a (public) key update for users of different hierarchal levels. In other words, the level conversion key allows one to delegate his key to its children without re-randomizing his key. Although the idea is simple, the concrete machinery to blend the level conversion keys securely into the construction is rather contrived and we refer the details to Section 2.

Finally, we state some side contributions worth highlighting in our paper. Firstly, we re-formalize the syntax and security definitions for R(H)IBE. For instance, since previous security definitions [6,33,34,35] had some ambiguity (e.g. in some cases it is not clear when the values such as secret keys and key updates are generated during the security game), it was up to the readers to interpret the definitions and the proofs. Therefore, in our work we provide a refined security definition for R(H)IBE which in particular is a more rigorous and explicit treatment than the previous definitions. Secondly, we provide a formal treatment on an implicit argument that has been frequently adopted in the R(H)IBE literature. In particular, we introduce a simple yet handy "strategy-dividing lemma", which helps us simplify the security proofs for R(H)IBE schemes in general. For the details, see Section 4.

Related Works. Boldyreva et al. [6] proposed the first RIBE scheme that achieved selective-identity security from bilinear maps and Libert and Vergnaud [20] extended their results to the adaptive setting. The first lattice-based RIBE scheme was proposed by Chen et al. [10] and the first RHIBE scheme was proposed by Seo and Emura [34] based on bilinear maps. Recently, Chang

et al. [9] proposed an RIBE scheme from codes with rank metric in the random oracle model.

After Seo and Emura [33] introduced the security notion of DKER and proposed the first RIBE scheme with DKER, several improvements and variants have been proposed. These works consist of RIBE [15,18,37] and RHIBE [13,17,19,35] from bilinear maps, and those from multilinear maps [23,28,29]. From lattices, Takayasu and Watanabe [36] proposed an RIBE scheme with bounded DKER; a strictly weaker notion then DKER.

Server-aided RIBE [11,26,30] is a variant of RIBE where most of the computation of the users are delegated to an untrusted server. The revocation mechanism we study in this paper is sometimes referred to as indirect revocation. A direct revocation mechanism does not require key updates and has been discussed for attribute-based encryption [4,5] and predicate encryption [27]. Recently, Ling et al. proposed the first lattice-based directly revocable predicate encryption scheme [21] and its server-aided variant [22].

Roadmap. In Section 2, we provide an overview of our constructions. In Section 3, we recall basic tools for lattice-based cryptography. In Section 4, we introduce formal definitions for RHIBE. In Section 5, we show a generic construction of RIBE with DKER. Finally, in Section 6, we show our main result concerning the first lattice-based RHIBE scheme with DKER.

Notations. Before diving into the technical details, we prepare some notations. Let $\mathbb{N}$ be the set of all natural numbers. For non-negative integers $n, n' \in \mathbb{N}$ with $n \leq n'$, we define $[n, n'] := \{n, n+1, \ldots, n'\}$, and we extend the definition for $n > n'$ by $[n, n'] = \emptyset$. For notational convenience, for $n \in \mathbb{N}$, we define $[n] := [1, n]$. Throughout the paper, $\lambda \in \mathbb{N}$ denotes the security parameter.

As usual in the literature of (R)HIBE, an identity ID of a user at level ℓ in the hierarchy in an RHIBE scheme is expressed as a length-ℓ vector $\mathsf{ID} = (\mathsf{id}_1, \cdots, \mathsf{id}_\ell)$. In order not to mix up with an identity $\mathsf{ID} = (\mathsf{id}_1, \mathsf{id}_2, \ldots)$ treated in an RHIBE scheme and its element id_i, we sometimes call the former a *hierarchical identity* and the latter an *element identity*. We refer to the set of all element identities as the *element identity space* and denote it by $\mathcal{ID}$. We assume the element identity space is determined only by the security parameter λ. Thus, for example, the space to which level-ℓ identities belong is expressed as $(\mathcal{ID})^\ell$. For notational convenience, for $\ell \in \mathbb{N}$ we define $(\mathcal{ID})^{\leq \ell} := \bigcup_{i \in [\ell]} (\mathcal{ID})^i$, and the hierarchal identity space $\mathcal{ID}_{\mathsf{h}} := (\mathcal{ID})^{\leq L}$. We denote by "$\mathsf{kgc}$" the special hierarchical identity for the level-0 user, i.e., the key generation center (KGC).

Like an ordinary vector, we consider a prefix of hierarchical identities. For example, for a level-ℓ hierarchical identity $\mathsf{ID} = (\mathsf{id}_1, \ldots, \mathsf{id}_\ell)$ and $t \leq \ell$, $\mathsf{ID}_{[t]}$ represents the length-t prefix of ID, i.e., $\mathsf{ID}_{[t]} = (\mathsf{id}_1, \ldots, \mathsf{id}_t)$. We denote by "$\mathsf{pa}(\mathsf{ID})$" the identity of its parent (i.e. the direct ancestor), namely, if $\mathsf{ID} \in (\mathcal{ID})^\ell$, then $\mathsf{pa}(\mathsf{ID}) := \mathsf{ID}_{[\ell-1]} = (\mathsf{id}_1, \ldots, \mathsf{id}_{\ell-1})$, and $\mathsf{pa}(\mathsf{ID})$ for a level-1 identity $\mathsf{ID} \in \mathcal{ID}$ is defined to be kgc. Furthermore, we denote by "$\mathsf{prefix}(\mathsf{ID})$" the set consisting of itself and all of its ancestors, namely, $\mathsf{prefix}(\mathsf{ID}) := \{\mathsf{ID}_{[1]}, \mathsf{ID}_{[2]}, \ldots, \mathsf{ID}_{[|\mathsf{ID}|]} = \mathsf{ID}\}$. Also, for $\mathsf{ID} \in (\mathcal{ID})^\ell$, we denote by "$\mathsf{ID} \| \mathcal{ID}$" as the subset of $(\mathcal{ID})^{\ell+1}$ that contains all the members who have ID as its parent.

2 Technical Overview

In this section, we provide the technical overview of our results. In order to make the lattice-based RHIBE overview easier to follow, we present the details of our generic construction of RIBE with DKER using lattice terminologies. The general idea presented below translates naturally to our generic construction. To this end, we first prepare two standard hash functions used in lattice-based cryptography: one for the users $\mathsf{ID} \in \mathcal{ID}_{\mathtt{h}} = \mathcal{ID}^{\leq L}$, where each element identity space is defined by $\mathcal{ID} = \mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}$, and another for the time period[4] $\mathsf{t} \in \mathcal{T} \subset \mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}$. In particular, for a user $\mathsf{ID} = (\mathsf{id}_1, \ldots, \mathsf{id}_\ell) \in (\mathbb{Z}_q^n \setminus \{\mathbf{0}_n\})^{\leq L}$ and time period $\mathsf{t} \in \mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}$ we use the following hash functions $\mathbf{E}(\cdot)$ and $\mathbf{F}(\cdot)$:

$$\begin{aligned} \mathbf{E}(\mathsf{ID}) &:= [\mathbf{B}_1 + H(\mathsf{id}_1)\mathbf{G} | \cdots | \mathbf{B}_\ell + H(\mathsf{id}_\ell)\mathbf{G}] \in \mathbb{Z}_q^{n \times \ell m}, \\ \mathbf{F}(\mathsf{t}) &:= \mathbf{B}_{L+1} + H(\mathsf{t})\mathbf{G} \in \mathbb{Z}_q^{n \times m}, \end{aligned} \tag{1}$$

where $(\mathbf{B}_j)_{j \in [L+1]}$ are random matrices in $\mathbb{Z}_q^{n \times m}$ chosen at setup of the scheme and $\mathbf{G}$ is the gadget matrix [24]. Here, $H : \mathbb{Z}_q^n \to \mathbb{Z}_q^{n \times n}$ is a specific hash function used to encode an identity to a matrix, and its definition is provided in Section 3. Notice that for any $\mathsf{ID} \in (\mathbb{Z}_q^n \setminus \{\mathbf{0}_n\})^\ell$ and $\mathsf{id}_{\ell+1} \in \mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}$, we have $\mathbf{E}(\mathsf{ID} \| \mathsf{id}_{\ell+1}) = [\mathbf{E}(\mathsf{ID}) | \mathbf{B}_{\ell+1} + H(\mathsf{id}_{\ell+1})\mathbf{G}]$. Finally, we define $\mathbf{E}(\mathsf{kgc}) := \emptyset$.

Review of RIBE *without* DKER. We first recall Chen et al.'s lattice-based RIBE scheme *without* DKER [10] in Figure 1. Here, $\mathbf{A}$ and $\mathbf{u}$ in the master public key PP are a matrix in $\mathbb{Z}_q^{n \times m}$ and a vector in $\mathbb{Z}_q^n$, respectively, and $\mathbf{T}_{\mathbf{A}}$ is the trapdoor associated with $\mathbf{A}$. Other terms will be explained as we proceed with our technical overview. Below, we see why the scheme realizes the revocation

$\mathsf{PP} := (\mathbf{A}, \mathbf{u}, \text{hash functions } \mathbf{E}(\cdot), \mathbf{F}(\cdot))$,	$\mathsf{sk}_{\mathsf{kgc}} := \mathbf{T}_{\mathbf{A}}$
$\mathsf{ct} := (c_0 := \mathbf{u}^\top \mathbf{s} + \mathsf{noise} + \mathsf{M} \lfloor \frac{q}{2} \rfloor, \mathbf{c}_1 := [\mathbf{A} \| \mathbf{E}(\mathsf{ID}) \| \mathbf{F}(\mathsf{t})]^\top \mathbf{s} + \mathsf{noise})$	
$\mathsf{sk}_{\mathsf{ID}} := (\mathbf{e}_{\mathsf{ID},\theta})_\theta$	s.t. $[\mathbf{A} \| \mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_\theta$
$\mathsf{ku}_{\mathsf{t}} := (\mathbf{e}_{\mathsf{t},\theta})_\theta$	s.t. $[\mathbf{A} \| \mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{t},\theta} = \mathbf{u} - \mathbf{u}_\theta$
$\mathsf{dk}_{\mathsf{ID},\mathsf{t}} := \mathbf{d}_{\mathsf{ID},\mathsf{t}}$	s.t. $[\mathbf{A} \| \mathbf{E}(\mathsf{ID}) \| \mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}$

Fig. 1: **Chen et al.'s RIBE Scheme**

mechanism while it does not satisfy DKER. One feature of RIBE construction is that the KGC maintains a binary tree where each user is assigned to a randomly selected leaf. Furthermore, a random vector $\mathbf{u}_\theta \in \mathbb{Z}_q^n$ is uniquely assigned to each node θ of the binary tree. Below, we explain the three types of keys which are core tools to realize the revocation mechanism: A *secret key* for a user ID is

[4] As we will show in Section 4, the time period space is a set of natural numbers $\{1, 2, \ldots\}$. Here, we assume that there is an efficient hash function that maps each natural number to a distinct vector in $\mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}$.

a tuple of short vectors $\mathsf{sk}_{\mathsf{ID}} = (\mathbf{e}_{\mathsf{ID},\theta})_\theta$, where each *short* vector $\mathbf{e}_{\mathsf{ID},\theta} \in \mathbb{Z}^{2m}$ is associated to a random vector $\mathbf{u}_\theta$ such that

$$[\mathbf{A}|\mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_\theta.$$

Since $\mathbf{u}_\theta$ is an independent random vector and the ciphertext c_0 only depends on $\mathbf{u}$, the vector $\mathbf{e}_{\mathsf{ID},\theta}$ in $\mathsf{sk}_{\mathsf{ID}}$ itself is useless for decrypting a ciphertext ct. Hence, in each time period the KGC broadcasts a *key update* which is also a tuple of short vectors $\mathsf{ku}_{\mathsf{t}} = (\mathbf{e}_{\mathsf{t},\theta})_\theta$, where each short vector $\mathbf{e}_{\mathsf{t},\theta}$ is associated to a random vector $\mathbf{u}_\theta$ such that

$$[\mathbf{A}|\mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{t},\theta} = \mathbf{u} - \mathbf{u}_\theta.$$

Similarly to above, $\mathbf{e}_{\mathsf{t},\theta}$ in ku_{t} itself is useless for decrypting a ciphertext ct. Now, we explain how the revocation mechanism works. By utilizing the complete subtree (CS) method [25], the KGC is able to broadcast key updates so that there is no common node θ in ku_{t} and $\mathsf{sk}_{\mathsf{ID}}$ of *revoked* IDs, while there is at least one common node θ in ku_{t} and $\mathsf{sk}_{\mathsf{ID}}$ of *non-revoked* IDs. Then, $\mathbf{e}_{\mathsf{ID},\theta}$ in $\mathsf{sk}_{\mathsf{ID}}$ and $\mathbf{e}_{\mathsf{t},\theta}$ in ku_{t} of the common node θ enable a non-revoked ID to derive a well-formed *decryption key* $\mathbf{d}_{\mathsf{ID},\mathsf{t}} \in \mathbb{Z}^{3m}$ which is a *short* vector satisfying

$$[\mathbf{A}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}.$$

It can be easily checked that $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ can be obtained by simply adding $\mathbf{e}_{\mathsf{ID},\theta}$ and $\mathbf{e}_{\mathsf{t},\theta}$ in a component-wise fashion. Note that if $\mathbf{e}_{\mathsf{ID},\theta}$ and $\mathbf{e}_{\mathsf{t},\theta}$ are short vectors, then so is $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$. Then, the vector enables us to recover the plaintext by computing

$$c_0 - \mathbf{c}_1^\top \mathbf{d}_{\mathsf{ID},\mathsf{t}} \approx \mathsf{M} \left\lfloor \frac{q}{2} \right\rfloor.$$

The main insight of this construction is that only non-revoked users can use the key updates to eliminate the random factor $\mathbf{u}_\theta$ to obtain a short vector $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ that is bound to the the public matrix $[\mathbf{A}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]$ and public vector $\mathbf{u}$ with which a ciphertexts ct is created.

Although the scheme is proven to be a secure RIBE scheme *without* DKER, it clearly does not satisfy DKER. Indeed, there is a concrete attack even with a single decryption key query (i.e., decryption key exposure) on the target ID^*. The attack is as follows: assume that the adversary obtains a decryption key $\mathsf{dk}_{\mathsf{ID}^*,\mathsf{t}}$ for the target ID^* and a time period $\mathsf{t} \neq \mathsf{t}^*$. Since key updates are publicly broadcast, the adversary also obtains ku_{t} and $\mathsf{ku}_{\mathsf{t}^*}$. Since user ID^* will not be revoked unless $\mathsf{sk}_{\mathsf{ID}^*}$ was revealed to the adversary, the key updates ku_{t} and $\mathsf{ku}_{\mathsf{t}^*}$ will share a common node θ^* with the secret key.[5] Therefore, recalling that $\mathsf{dk}_{\mathsf{ID}^*,\mathsf{t}}$ was a simple component-wise addition of $\mathbf{e}_{\mathsf{ID}^*,\theta^*}$ in $\mathsf{sk}_{\mathsf{ID}^*}$ and $\mathbf{e}_{\mathsf{t},\theta^*}$ in ku_{t}, $\mathcal{A}$ can first recover the secret key component $\mathbf{e}_{\mathsf{ID}^*,\theta^*}$ from $(\mathsf{dk}_{\mathsf{ID}^*,\mathsf{t}}, \mathbf{e}_{\mathsf{t},\theta^*})$, which he can then combine it with $\mathbf{e}_{\mathsf{t}^*,\theta^*}$ in $\mathsf{ku}_{\mathsf{t}^*}$ to create the decryption key $\mathbf{d}_{\mathsf{ID}^*,\mathsf{t}^*}$ for the

[5] To be more precise, there are cases ku_{t} and $\mathsf{ku}_{\mathsf{t}^*}$ might not share a common node, however, $\mathcal{A}$ can always adaptively revoke other users so that this holds.

challenge time period t^*. Specifically, this decryption key allows the adversary to completely break the scheme. In reality, this corresponds to the fact that once a decryption key for a certain time period is exposed to an adversary, then all the messages of distinct time periods may also be compromised. In essence, this attack relies on the fact that the decryption key leaks partial information on the secret key, which can then be used to construct decryption keys of all distinct time periods.

In all the previous bilinear map-based constructions, the above problem was circumvented by relying on the so-called *key re-randomization property*. Informally, this property allows one to re-randomize the decryption key, hence even if the decryption key is leaked, it would be impossible to restore the original secret key. In the above construction, this idea would correspond to re-sampling a short random vector $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ such that

$$[\mathbf{A}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}$$

using his original decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$. Indeed, if the distribution of $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ is independent of the original decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$, this modification would prevent the above attack, since the adversary will not be able to recover the secret key component $\mathbf{e}_{\mathsf{ID}^*,\theta^*}$ anymore using the above strategy. However, such a re-sampling procedure is computationally infeasible, since otherwise we would be able to trivially solve the small integer solution (SIS) problem.

Readers familiar with lattice-based constructions of (non-revocable) HIBE may think that we could achieve the key re-randomization property by simply using a short trapdoor basis as the secret key instead of a vector. Indeed, if we add a short trapdoor basis $\mathbf{T}_{[\mathbf{A}|\mathbf{E}(\mathsf{ID})]}$ as a part of the secret key $\mathsf{sk}_{\mathsf{ID}}$, the user ID will be able to sample a short vector $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}} \neq \mathbf{d}_{\mathsf{ID},\mathsf{t}}$, since anybody can efficiently extend the trapdoor basis $\mathbf{T}_{[\mathbf{A}|\mathbf{E}(\mathsf{ID})]}$ to $\mathbf{T}_{[\mathbf{A}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]}$ and thus sample a random vector $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ such that $[\mathbf{A}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}$. However, this approach does not mesh well with the above revocation mechanism, since now the user ID can derive decryption keys $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ for every time period without requiring the key updates ku_{t}. Therefore, adding a short trapdoor basis to the secret key provides too much flexibility to the users and we completely lose the mechanism for supporting revocation.

Constructing RIBE *with* DKER. To summarize so far, the main bottleneck of Chen et al.'s RIBE scheme without DKER is that it satisfies the key revocation mechanism, but seems challenging to extend it to satisfy DKER. On the other hand, adding a short trapdoor basis would definitely be useful for achieving DKER, however, it seems to contradict with the revocation mechanism. In the following, we show that we can carefully combine these two seemingly conflicting ideas together. The concrete construction of our lattice-based RIBE scheme *with* DKER is illustrated in Figure 2. The boxed items denote the changes made from the previous figure.

Our construction relies on a tool we call *leveled ciphertexts* and *leveled decryption keys*; the terminology should become more intuitive and helpful in the hierarchical setting that we explaine later. Here, we call an element associated

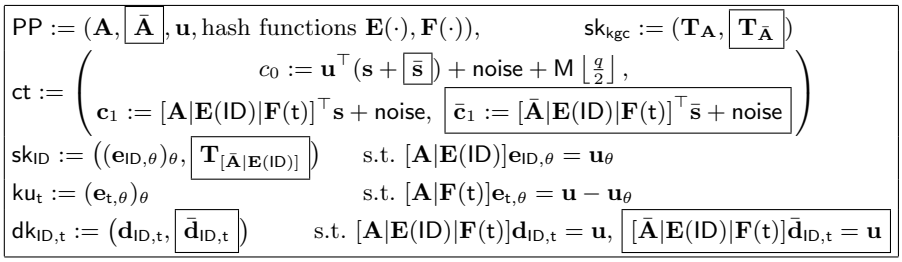

Fig. 2: **Our RIBE Scheme with DKER**

with a matrix $\mathbf{A}$ and $\bar{\mathbf{A}}$ level-1 and level-2, respectively. In particular, $\mathbf{c}_1, \bar{\mathbf{c}}_1$ and $\mathbf{d}_{\mathsf{ID},\mathsf{t}}, \bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ in Figure 2 are the level-1, level-2 ciphertexts and decryption keys, respectively. Here, the level-1 components $\mathbf{c}_1$ and $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ correspond to Chen et al.'s RIBE scheme without DKER and are responsible for achieving the revocation mechanism. On the other hand, the level-2 components $\bar{\mathbf{c}}_1$ and $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ are the newly introduced elements that will help us achieve DKER. Since the two decryption keys for levels-1 and 2 are in one-to-one correspondence with the ciphertexts $(\mathbf{c}_1, \bar{\mathbf{c}}_1)$ for levels-1 and 2, both of the decryption keys are required to recover the underlying message as follows:

$$c_0 - \underbrace{\mathbf{c}_1^\top \mathbf{d}_{\mathsf{ID},\mathsf{t}}}_{\text{level-1 component}} - \underbrace{\bar{\mathbf{c}}_1^\top \bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}}_{\text{level-2 component}} \approx \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor.$$

In particular, if either level of the decryption key is missing, the message cannot be recovered. Separating the role of the decryption keys is the main idea that allows us to associate the two seemingly conflicting properties of revocation and key re-randomization to each level of the decryption keys.

First, we observe that the above RIBE scheme achieves the revocation mechanism since it simply inherits this property from the underlying Chen et al.'s RIBE scheme without DKER. Furthermore, we achieve DKER by incorporating the aforementioned trapdoor idea; we add a trapdoor $\mathbf{T}_{[\bar{\mathbf{A}}|\mathbf{E}(\mathsf{ID})]}$ to the secret key $\mathsf{sk}_{\mathsf{ID}}$. Using this short trapdoor basis $\mathbf{T}_{[\bar{\mathbf{A}}|\mathbf{E}(\mathsf{ID})]}$, we can now sample a level-2 decryption key $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ for each time period independently from the previous time periods. Namely, using $\mathbf{T}_{[\bar{\mathbf{A}}|\mathbf{E}(\mathsf{ID})]}$, we can sample a short vector $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ such that

$$[\bar{\mathbf{A}}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}} = \mathbf{u},$$

where $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ leaks no information of the secret key $\mathsf{sk}_{\mathsf{ID}}$. Hence, although we are not able to completely re-randomize the decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} = (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, \bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}})$, we can *partially* re-randomize the decryption key by sampling a new level-2 decryption key $\bar{\mathbf{d}}_{\mathsf{ID},\mathsf{t}}$ for each time period; even if $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ is compromised, this alone will not be sufficient for constructing decryption keys for other time periods. Indeed, we show that this partial key re-randomization property is sufficient to prove the DKER security.

In Section 5, we formalize and prove the above idea by providing a generic construction of RIBE with DKER, using as building blocks any RIBE without DKER and 2-level HIBE. At a high level, the 2-level HIBE scheme is responsible for the key re-randomization property and is the core component that allows us to convert non-DKER secure RIBE schemes into DKER secure RIBE schemes.

Constructing RHIBE from Lattices. Next, we show an overview of our lattice-based RHIBE construction. For simplicity of presentation and since we can add DKER via the above idea, we do not take into account DKER in the following RHIBE construction. Specifically, we explain how to construct an RHIBE scheme *without* DKER by modifying Chen et al.'s RIBE scheme.

Before getting into detail, we prepare some notations used for the hierarchal setting. In the following, let L be the maximum depth of the hierarchy, where we treat the KGC as level-0. In RHIBE, all level-i users ID for $i \in [0, L-1]$, including the KGC, maintain a binary tree $\mathtt{BT}_{\mathsf{ID}}$ to manage their children users in $\mathsf{ID}\|\mathcal{ID}$. Furthermore, a random vector $\mathbf{u}_{\mathsf{ID},\theta} \in \mathbb{Z}_q^n$ is uniquely assigned to each node θ of the binary tree $\mathtt{BT}_{\mathsf{ID}}$. The level-$(\ell-1)$ user $\mathsf{pa}(\mathsf{ID})$ creates the secret key $\mathsf{sk}_{\mathsf{ID}}$ of the level-ℓ user ID, and the user ID derives his own decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ by combining his own secret key $\mathsf{sk}_{\mathsf{ID}}$ and the key updates $\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}}$ that are broadcast by the parent user $\mathsf{pa}(\mathsf{ID})$. Throughout the overview, we assume ID represents an level-ℓ user.

Introducing Leveled Secret Keys: Due to the complex nature of our scheme, we believe it to be helpful to provide the intuition of our scheme following a series of modifications, where our final scheme without DKER is depicted in Figure 6. Our starting point is illustrated in Figure 3, where as before, the box indicates the changes made from the prior scheme.

$\mathsf{PP} := (\boxed{(\mathbf{A}_i)_{i\in[L]}}, \mathbf{u}, \text{hash functions } \mathbf{E}(\cdot), \mathbf{F}(\cdot))$, $\quad \mathsf{sk}_{\mathsf{kgc}} := \boxed{(\mathbf{T}_{\mathbf{A}_i})_{i\in[L]}}$

$\mathsf{ct} := \left(c_0 := \mathbf{u}^\top \mathbf{s} + \mathsf{noise} + \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor, \mathbf{c}_1 := [\boxed{\mathbf{A}_\ell}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]^\top \mathbf{s} + \mathsf{noise}\right)$

$\mathsf{sk}_{\mathsf{ID}} := \left((\mathbf{e}_{\mathsf{ID},\theta})_\theta, \boxed{(\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]})_{i\in[\ell+1,L]}}\right)$ $\quad$ s.t. $[\boxed{\mathbf{A}_\ell}|\mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_{\mathsf{pa}(\mathsf{ID}),\theta}$

$\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}} := (\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta})_\theta$ $\quad$ s.t. $[\boxed{\mathbf{A}_\ell}|\mathbf{E}(\mathsf{pa}(\mathsf{ID}))|\mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta} = \mathbf{u} - \mathbf{u}_{\mathsf{pa}(\mathsf{ID}),\theta}$

$\mathsf{dk}_{\mathsf{ID},\mathsf{t}} := \mathbf{d}_{\mathsf{ID},\mathsf{t}}$ $\quad$ s.t. $[\boxed{\mathbf{A}_\ell}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}$

Fig. 3: **Leveled Secret Key and i-Leveled Ciphertext**

Toward resolving the incompatibility of the key delegation property and the key revocation mechanism, the scheme in Figure 3 utilizes leveled ciphertexts as done in the prior non-hierarchal scheme in Figure 2. Furthermore, we introduce a new tool called *leveled secret keys* in this scheme. Here, we call an element associated with a matrix $\mathbf{A}_i$ level-i, respectively. In particular, the ciphertext ct of a level-ℓ user ID is a level-ℓ ciphertext since $\mathbf{c}_1$ is associated with $\mathbf{A}_\ell$. The main trick of the scheme in Figure 3 is that a secret key $\mathsf{sk}_{\mathsf{ID}}$ for a level-ℓ user consists of level-i secret keys for $i \in [\ell, L]$, where the level-ℓ secret key $(\mathbf{e}_{\mathsf{ID},\theta})_\theta$

and the other level-i secret keys $\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]}$ for $i \in [\ell+1, L]$ serve a different purpose. The level-ℓ secret key in $\mathsf{sk}_{\mathsf{ID}}$ is a tuple of short vectors of the form $(\mathbf{e}_{\mathsf{ID},\theta})_\theta$ each of which satisfies

$$[\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = [\mathbf{A}_\ell|\mathbf{E}(\mathsf{pa}(\mathsf{ID}))|\mathbf{B}_\ell + H(\mathsf{id}_\ell)\mathbf{G}]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_{\mathsf{pa}(\mathsf{ID}),\theta}, \tag{2}$$

and serves the same purpose as the original Chen et al.'s RIBE scheme. Namely, the level-ℓ secret key of a level-ℓ user is used for decrypting its own level-ℓ ciphertext, where the detailed procedure will be explained later. The remaining level-i secret keys in $\mathsf{sk}_{\mathsf{ID}}$ for $i \in [\ell+1, L]$ are trapdoors of the form $\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]}$ in $\mathsf{sk}_{\mathsf{ID}}$ and serves the purpose of delegation. Concretely, using the trapdoor $\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]}$ for $i \in [\ell+1, L]$, the level-ℓ user ID can sample all level-i secret keys for his children $\mathsf{ID}\|\mathsf{id}_{\ell+1} \in \mathsf{ID}\|\mathcal{ID}$; a set of short vectors $(\mathbf{e}_{\mathsf{ID}\|\mathsf{id}_{\ell+1},\theta})_\theta$ such that $[\mathbf{A}_i|\mathbf{E}(\mathsf{ID}\|\mathsf{id}_{\ell+1})]\mathbf{e}_{\mathsf{ID}\|\mathsf{id}_{\ell+1},\theta} = \mathbf{u}_{\mathsf{ID},\theta}$ and trapdoors $\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID}\|\mathsf{id}_{\ell+1})]}$ for $i \in [\ell+2, L]$. In addition, the level-ℓ user ID can also use the level-$(\ell+1)$ trapdoor $\mathbf{T}_{[\mathbf{A}_{\ell+1}|\mathbf{E}(\mathsf{ID})]}$ in $\mathsf{sk}_{\mathsf{ID}}$ to derive key updates $\mathsf{ku}_{\mathsf{ID},\mathsf{t}}$. Here, a level-$(\ell-1)$ user $\mathsf{pa}(\mathsf{ID})$'s key update $\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}}$ is a tuple of short vectors $(\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta})_\theta$ such that

$$[\mathbf{A}_\ell|\mathbf{E}(\mathsf{pa}(\mathsf{ID}))|\mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta} = \mathbf{u} - \mathbf{u}_{\mathsf{pa}(\mathsf{ID}),\theta}. \tag{3}$$

Then, from Eq. (2) and (3), the level-ℓ user ID can derive a well-formed decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ which is a short vector of the form $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ satisfying

$$[\mathbf{A}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = [\mathbf{A}|\mathbf{E}(\mathsf{pa}(\mathsf{ID}))|\mathbf{B}_\ell + H(\mathsf{id}_\ell)\mathbf{G}|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}.$$

Hence, the scheme in Figure 3 properly supports the key delegation functionality.

Furthermore, at first glance, the scheme also supports the key revocation mechanism. Since the level-ℓ secret key $(\mathbf{e}_{\mathsf{ID},\theta})_\theta$ of the level-ℓ user ID is exactly the same as the secret key used by user ID in Chen et al.'s RIBE scheme, it simply inherits the revocation mechanism. In particular, user ID will not be able to decrypt his level-ℓ ciphertext without his parent's key update $\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}}$, which will no longer be provided once user ID is revoked. However, unfortunately, this scheme is trivially flawed and does not meet the security notion of RHIBE. In RHIBE, we require the user ID to be revoked once any of his ancestors $\mathsf{ID}_{[i]} \in \mathsf{prefix}(\mathsf{ID})$ for $i \in [\ell-1]$ is revoked. In other words, once a user is revoked from the system, then all of its descendants must also be revoked. It can be easily checked that this requirement is not met by our above RHIBE scheme. Since the level-ℓ user ID has the full trapdoor $\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]}$ for $i \in [\ell+1, L]$ as part of its secret key, nothing is preventing user ID from continuing on generating secret keys and key updates for his children.

Introducing Leveled Decryption Keys: To fix the above issue concerning key revocation, we further modify the scheme as in Figure 4. From now on, we further modify the definition of level-i ciphertext, and call a tuple

$$(\mathbf{u}^\top\mathbf{s}_i + \mathsf{noise}, \quad \mathbf{c}_i = [\mathbf{A}_i|\mathbf{E}(\mathsf{ID}_{[i]})|\mathbf{F}(\mathsf{t})]^\top\mathbf{s}_i + \mathsf{noise})$$

a level-i ciphertext since $\mathbf{c}_i$ is associated with the public matrix $\mathbf{A}_i$ and both components are associated with the same secret vector $\mathbf{s}_i$. In this scheme, we

$$\mathsf{PP} := ((\mathbf{A}_i)_{i\in[L]}, \mathbf{u}, \text{hash functions } \mathbf{E}(\cdot), \mathbf{F}(\cdot)), \qquad \mathsf{sk}_{\mathsf{kgc}} := (\mathbf{T}_{\mathbf{A}_i})_{i\in[L]}$$

$$\mathsf{ct} := \left(\begin{array}{l} c_0 := \mathbf{u}^\top \boxed{(\mathbf{s}_1 + \cdots + \mathbf{s}_\ell)} + \mathsf{noise} + \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor, \\ \boxed{(\mathbf{c}_i := [\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]}) | \mathbf{F}(\mathsf{t})]^\top \mathbf{s}_i + \mathsf{noise})_{i\in[\ell]}} \end{array} \right)$$

$$\mathsf{sk}_{\mathsf{ID}} := ((\mathbf{e}_{\mathsf{ID},\theta}), (\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]})_{i\in[\ell+1,L]}) \quad \text{s.t. } [\mathbf{A}_\ell | \mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_\theta$$

$$\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}} := ((\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta}), \boxed{(\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}})_{i\in[\ell-1]}}) \quad \text{s.t. } [\mathbf{A}_\ell | \mathbf{E}(\mathsf{pa}(\mathsf{ID}))|\mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta} = \mathbf{u} - \mathbf{u}_\theta,$$
$$\boxed{[\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]})|\mathbf{F}(\mathsf{t})]\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}} = \mathbf{u}}$$

$$\mathsf{dk}_{\mathsf{ID},\mathsf{t}} := (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, \boxed{(\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}})_{i\in[\ell-1]}}) \quad \text{s.t. } [\mathbf{A}_\ell | \mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}$$

Fig. 4: **Multiple Leveled Ciphertext and Key Update**

modify the ciphertext for a level-ℓ user ID to contain all the level-i ciphertexts for $i \in [\ell]$, where each level-i ciphertext is associated with the public matrix $\mathbf{A}_i$ and an identity $\mathsf{ID}_{[i]}$. The idea behind this modification is to revoke any user ID whose ancestors were revoked by including some information specific to the ancestors in the ciphertext. In particular, if some ancestor at level $i \in [\ell - 1]$ were to be revoked, then the level-i ciphertext $\mathbf{c}_i$ should become undecryptable, hence maintaining the secrecy of the plaintext M. To make this idea work, we must now provide user ID with new components to allow decryption of the level-i ciphertexts for $i \in [\ell - 1]$. We achieve this by introducing a new tool called *leveled decryption keys*. A leveled decryption key for a level-ℓ user ID consists of level-i decryption keys for $i \in [\ell]$. Similarly to leveled secret keys, the level-ℓ decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ and the other level-i decryption keys $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}}$ for $i \in [\ell - 1]$ serve a different purpose. The level-ℓ decryption key denoted as $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ in $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ serves the same purpose as in the previous schemes. The level-i decryption key for $i \in [\ell - 1]$ denoted as $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}}$ in $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ is the actual decryption key used by its ancestor at level-i. Although we use a different notation, $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}}$ is equivalent to $\mathbf{d}_{\mathsf{ID}_{[i]},\mathsf{t}}$ such that

$$[\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]})|\mathbf{F}(\mathsf{t})]\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}} = [\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]})|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID}_{[i]},\mathsf{t}} = \mathbf{u}. \tag{4}$$

In particular, each ancestor at level-i for $i \in [\ell - 1]$ broadcasts their own decryption key $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}}$ (See $\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}}$ in Figure 4) and the user ID sets the level-i decryption key for $i \in [\ell - 1]$ as $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}}$. It can be easily verified that user ID can correctly decrypt his ciphertext as follows:

$$c_0 - \underbrace{\mathbf{c}_\ell^\top \mathbf{d}_{\mathsf{ID},\mathsf{t}}}_{\text{level-}\ell\text{ component}} - \sum_{i=1}^{\ell-1} \underbrace{\mathbf{c}_i^\top \mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}}}_{\text{level-}i\text{ component}} \approx \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor.$$

However, this scheme is obviously insecure, since the level-i ancestors are required to publicly broadcast their level-i decryption key $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t}} (= \mathbf{d}_{\mathsf{ID}_{[i]},\mathsf{t}})$, which can in turn be used by anybody to decrypt the level-i ciphertexts of that particular ancestor.

Making the Levels Two-Dimensional: For the scheme in Figure 4 to be secure, decryption keys of the ancestors should not be made public via the key updates. Specifically, a ciphertext aimed for a user should not contain the same level as of his ancestors, since otherwise the decryption keys of the ancestors must be made public. For the purpose, we further modify the scheme as in Figure 5. To this

$$\mathsf{PP} := ((\mathbf{A}_i)_{i\in[L]}, \boxed{(\mathbf{u}_k)_{k\in[L]}}, \text{hash functions } \mathbf{E}(\cdot), \mathbf{F}(\cdot)), \quad \mathsf{sk}_{\mathsf{kgc}} := (\mathbf{T}_{\mathbf{A}_i})_{i\in[L]}$$

$$\mathsf{ct} := \begin{pmatrix} c_0 := \boxed{\mathbf{u}_\ell}^\top (\mathbf{s}_1 + \cdots \mathbf{s}_\ell) + \mathsf{noise} + \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor, \\ (\mathbf{c}_i := [\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]}) | \mathbf{F}(\mathsf{t})]^\top \mathbf{s}_i + \mathsf{noise})_{i\in[\ell]} \end{pmatrix}$$

$$\mathsf{sk}_{\mathsf{ID}} := ((\mathbf{e}_{\mathsf{ID},\theta}), (\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]})_{i\in[\ell+1,L]}) \qquad \text{s.t. } [\mathbf{A}_\ell | \mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_\theta,$$

$$\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}} := ((\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta}), \boxed{(\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},k})_{i\in[\ell-1],k\in[\ell,L]}})$$

$$\text{s.t. } [\mathbf{A}_\ell | \mathbf{E}(\mathsf{pa}(\mathsf{ID})) | \mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t},\theta} = \mathbf{u}_\ell - \mathbf{u}_\theta, \qquad \boxed{[\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]}) | \mathbf{F}(\mathsf{t})]\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},k} = \mathbf{u}_k}$$

$$\mathsf{dk}_{\mathsf{ID},\mathsf{t}} := (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, \boxed{(\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell})_{i\in[\ell-1]}}) \qquad \text{s.t. } [\mathbf{A}_\ell | \mathbf{E}(\mathsf{ID}) | \mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \boxed{\mathbf{u}_\ell}$$

Fig. 5: (k, i)-**Leveled Ciphertext and Decryption Key**

end, we incorporate multiple public vectors $(\mathbf{u}_k)_{k\in[L]}$, and redefine the notion of leveled ciphertexts and leveled decryption keys to be *two-dimensional*. Here, we refer to an element associated with a vector $\mathbf{u}_k$ and a matrix $\mathbf{A}_i$ as level-(k, i), respectively. For example, we call a tuple

$$(\mathbf{u}_k^\top \mathbf{s}_i + \mathsf{noise}, \quad \mathbf{c}_i = [\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]}) | \mathbf{F}(\mathsf{t})]^\top \mathbf{s}_i + \mathsf{noise})$$

a level-(k, i) ciphertext since the first component is associated with the public vector $\mathbf{u}_k$, and the latter component $\mathbf{c}_i$ is associated with the public matrix $\mathbf{A}_i$, and both components are associated with the same secret vector $\mathbf{s}_i$. In particular, a ciphertext for a level-ℓ user ID consists of level-(ℓ, i) ciphertexts for $i \in [\ell]$. Accordingly, we must provide user ID with a redefined leveled decryption key to allow decryption of the two-dimensional leveled ciphertexts. Specifically, we provide a level-ℓ user ID with level-(ℓ, i) decryption keys for $i \in [\ell]$, where again the level-(ℓ, ℓ) decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ and the other level-(ℓ, i) decryption keys $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$ for $i \in [\ell - 1]$ serve a different purpose. The level-(ℓ, ℓ) decryption key denoted as $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ is constructed and serves the exact same purpose as in the previous scheme. The level-(ℓ, i) decryption keys for $i \in [\ell - 1]$ are denoted as $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$. As before, these decryption keys $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$ are broadcast as part of the parent's key updates $\mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}}$, however, the way they are defined is slightly different from the previous scheme. Namely, the level-(ℓ, i) decryption key $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$ satisfies

$$[\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]}) | \mathbf{F}(\mathsf{t})]\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell} = \mathbf{u}_\ell,$$

Note that it is $\mathbf{u}_\ell$ and not $\mathbf{u}$ as in Eq. (4). Using this, a level-ℓ user ID can decrypt its ciphertext as follows:

$$c_0 - \underbrace{\mathbf{c}_\ell^\top \mathbf{d}_{\mathsf{ID},\mathsf{t}}}_{\text{level-}(\ell,\ell)\text{ component}} - \sum_{i=1}^{\ell-1} \underbrace{\mathbf{c}_i^\top \mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}}_{\text{level-}(\ell,i)\text{ component}} \approx \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor,$$

where each level of the ciphertext and decryption keys are in one-to-one correspondence with each other. Note that the level-ℓ user ID uses only level-(ℓ,i) decryption keys $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$ for $i \in [\ell-1]$ provided in the key update $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ to decrypt his own ciphertext. He simply forwards the remaining level-(k,i) decryption keys $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},k}$ for $(k,i) \in [\ell+1, L] \times [\ell-1]$ as part of his key update $\mathsf{ku}_{\mathsf{ID},\mathsf{t}}$.

One can see that the problem in the previous scheme of Figure 4 is now resolved, since the public term $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$ can only be used in combination with the level-(ℓ,i) ciphertext. In other words, due to the two-dimensional level, $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$ is only useful for decrypting ciphertexts of level-ℓ users. Furthermore, since the level-(ℓ,ℓ) decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ still remains secret, the publicly broadcast decryption keys $\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}$ for $i \in [\ell-1]$ alone are insufficient for decrypting the ciphertexts sent to user ID. The remaining problem with this approach is that there is currently no way for the level-$(\ell-1)$ ancestors $\mathsf{pa(ID)}$ to create the level-$(k,\ell-1)$ decryption keys $(\mathbf{f}_{\mathsf{ID}_{[\ell-1]},\mathsf{t},k})_{k\in[\ell,L]}$ which they must broadcast as part of the key updates $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$. Specifically, since they do not have the trapdoor $\mathbf{T}_{[\mathbf{A}_{\ell-1}|\mathbf{E}(\mathsf{ID}_{[\ell-1]})]}$, they cannot simply sample the level-$(k,\ell-1)$ decryption keys $(\mathbf{f}_{\mathsf{ID}_{[\ell-1]},\mathsf{t},k})_{k\in[\ell,L]}$ for every time period.

Introducing Level Conversion Keys: Finally, we arrive at our proposed RHIBE scheme (without DKER) illustrated in Figure 6. We overcome our final obstacle by introducing a tool called *level conversion keys*. In the scheme of Figure 5, a level-ℓ parent user ID is able to create his level-(ℓ,ℓ) decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ by himself although he cannot compute the level-(k,ℓ) decryption keys $(\mathbf{f}_{\mathsf{ID},\mathsf{t},k})_{k\in[\ell+1,L]}$ in the key updates $\mathsf{ku}_{\mathsf{ID},\mathsf{t}}$ (which corresponds to $(\mathbf{f}_{\mathsf{ID}_{[\ell-1]},\mathsf{t},k})_{k\in[\ell,L]}$ in $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ of level-$(\ell-1)$ users in the figure). To overcome the issue, we define a level-$[\ell,k]$

$$\mathsf{PP} := ((\mathbf{A}_i)_{i\in[L]}, (\mathbf{u}_k)_{k\in[L]}, \text{hash functions } \mathbf{E}(\cdot), \mathbf{F}(\cdot)), \qquad \mathsf{sk}_{\mathsf{kgc}} := (\mathbf{T}_{\mathbf{A}_i})_{i\in[L]}$$

$$\mathsf{ct} := \begin{pmatrix} c_0 := \mathbf{u}_\ell^\top(\mathbf{s}_1 + \cdots \mathbf{s}_\ell) + \mathsf{noise} + \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor, \\ (\mathbf{c}_i := [\mathbf{A}_i|\mathbf{E}(\mathsf{ID}_{[i]})|\mathbf{F}(\mathsf{t})]^\top \mathbf{s}_i + \mathsf{noise})_{i\in[\ell]} \end{pmatrix}$$

$$\mathsf{sk}_{\mathsf{ID}} := ((\mathbf{e}_{\mathsf{ID},\theta}), \boxed{(\mathbf{f}_{\mathsf{ID},k})_{k\in[\ell+1,L]}}, (\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]})_{i\in[\ell+1,L]}) \quad \text{s.t. } [\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_\theta,$$
$$\boxed{[\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})]\mathbf{f}_{\mathsf{ID},k} = \mathbf{u}_k - \mathbf{u}_\ell}$$

$$\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}} := ((\mathbf{e}_{\mathsf{pa(ID)},\mathsf{t},\theta}), (\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},k})_{i\in[\ell-1],k\in[\ell,L]})$$
$$\text{s.t. } [\mathbf{A}_\ell|\mathbf{E}(\mathsf{pa(ID)})|\mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{pa(ID)},\mathsf{t},\theta} = \mathbf{u}_\ell - \mathbf{u}_\theta, \qquad [\mathbf{A}_i|\mathbf{E}(\mathsf{ID}_{[i]})|\mathbf{F}(\mathsf{t})]\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},k} = \mathbf{u}_k$$

$$\mathsf{dk}_{\mathsf{ID},\mathsf{t}} := (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, (\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell})_{i\in[\ell-1]}) \qquad \text{s.t. } [\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}_\ell$$

Fig. 6: **Level Conversion Key**

conversion key $(\mathbf{f}_{\mathsf{ID},k})_{k\in[\ell+1,L]}$ of a level-ℓ user ID satisfying

$$[\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})]\mathbf{f}_{\mathsf{ID},k} = \mathbf{u}_k - \mathbf{u}_\ell.$$

To compute level-(k,ℓ) decryption keys $(\mathbf{f}_{\mathsf{ID},\mathsf{t},k})_{k\in[\ell+1,L]}$ in key updates $\mathsf{ku}_{\mathsf{ID},\mathsf{t}}$, the level-$[\ell,k]$ conversion key allows the user ID to convert his *secret* level-(ℓ,ℓ) decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ which satisfies

$$[\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{d}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}_\ell$$

into a *public* level-(k,ℓ) decryption key $\mathbf{f}_{\mathsf{ID},\mathsf{t},k}$ which satisfies

$$[\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{f}_{\mathsf{ID},\mathsf{t},k} = \mathbf{u}_k,$$

where the conversion is a simple component-wise addition. Since the scheme supports both the key delegation functionality and the key revocation mechanism, it can be shown to be a secure RHIBE scheme *without* DKER.

Adding DKER to the Construction: To make the above lattice-based RHIBE scheme in Figure 6 satisfy DKER, we will use the same idea incorporated in our generic construction of RIBE with DKER. Specifically, we add one more level to the above scheme and wrap a standard HIBE scheme around it to manage the partial key re-randomization property. The concrete construction appears in Section 6.

3 Preliminaries

In this section, we briefly summarize the basic tools used in lattice-based cryptography. We treat vectors in their column form. For a vector $\mathbf{v}\in\mathbb{R}^n$, denote $\|\mathbf{v}\|$ as the standard Euclidean norm. For a matrix $\mathbf{R}\in\mathbb{R}^{n\times n}$, denote $\|\mathbf{R}\|_{\text{GS}}$ as the longest column of the Gram-Schmidt orthogonalization of $\mathbf{R}$ and denote $\|\mathbf{R}\|_2$ as the largest singular value. We denote $\mathbf{I}_m$ as the $m\times m$ identity matrix and $\mathbf{0}_{n\times m}$ as the $n\times m$ zero matrix. We sometimes simply write $\mathbf{0}_n$ to denote (column) zero vectors.

Lattices. A (full-rank-integer) m-dimensional lattice Λ in $\mathbb{Z}^m$ is a set of the form $\{\sum_{i\in[m]} x_i\mathbf{b}_i | x_i\in\mathbb{Z}\}$, where $\mathbf{B}=\{\mathbf{b}_1,\cdots,\mathbf{b}_m\}$ are m linearly independent vectors in $\mathbb{Z}^m$. We call $\mathbf{B}$ the basis of the lattice Λ. For any positive integers n, m and $q\geq 2$, a matrix $\mathbf{A}\in\mathbb{Z}_q^{n\times m}$ and a vector $\mathbf{u}\in\mathbb{Z}_q^n$, we define $\Lambda_q^\perp(\mathbf{A}) = \{\mathbf{z}\in\mathbb{Z}^m|\mathbf{A}\mathbf{z}=\mathbf{0}_n \mod q\}$ and $\Lambda_q^{\mathbf{u}}(\mathbf{A}) = \{\mathbf{z}\in\mathbb{Z}^m|\mathbf{A}\mathbf{z}=\mathbf{u} \mod q\}$.

Gaussian Measures. Let $\mathcal{D}_{\Lambda,\sigma}$ denote the standard discrete Gaussian distribution over Λ with a Gaussian parameter σ. We summarize some basic properties of discrete Gaussian distributions.

Lemma 1 ([14]). *Let Λ be an m-dimensional lattice. Let $\mathbf{T}$ be a basis for Λ, and suppose $\sigma\geq\|\mathbf{T}\|_{\text{GS}}\cdot\omega(\sqrt{\log m})$. Then $\Pr[\|\mathbf{x}\|_2>\sigma\sqrt{m}: \mathbf{x}\leftarrow\mathcal{D}_{\Lambda,\sigma}]\leq\mathsf{negl}(m)$.*

Lemma 2 ([14]). *Let n, m, q be positive integers such that $m \geq 2n \log q$ and q a prime. Let σ be any positive real such that $\sigma \geq \omega(\sqrt{\log n})$. Then for $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ and $\mathbf{e} \leftarrow D_{\mathbb{Z}^m, \sigma}$, the distribution of $\mathbf{u} = \mathbf{A}\mathbf{e} \mod q$ is statistically close to uniform over $\mathbb{Z}_q^n$. Furthermore, for a fixed $\mathbf{u} \in \mathbb{Z}_q^n$, the conditional distribution of $\mathbf{e} \leftarrow D_{\mathbb{Z}^m, \sigma}$, given $\mathbf{A}\mathbf{e} = \mathbf{u} \mod q$ for a uniformly random $\mathbf{A}$ in $\mathbb{Z}_q^{n \times m}$ is $D_{\Lambda_q^{\mathbf{u}}(\mathbf{A}), \sigma}$ with all but negligible probability.*

Sampling Algorithms. We review some of the algorithms for sampling short vectors from a given lattice.

Lemma 3. *Let $n, m, \bar{m}, q > 0$ be positive integers with $m \geq 2n\lceil \log q \rceil$ and q a prime. Then, we have the following polynomial time algorithms:*

$\mathsf{TrapGen}(1^n, 1^m, q) \rightarrow (\mathbf{A}, \mathbf{T_A})$ *([2,3,24]): a randomized algorithm that outputs a full rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a basis $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^{\perp}(\mathbf{A})$ such that $\mathbf{A}$ is statistically close to uniform and $\|\mathbf{T_A}\|_{\mathrm{GS}} = O\left(\sqrt{n \log q}\right)$ with overwhelming probability in n.*

$\mathsf{SampleLeft}(\mathbf{A}, \mathbf{F}, \mathbf{u}, \mathbf{T_A}, \sigma) \rightarrow \mathbf{e}$ *([1,24]): a randomized algorithm that, given as input a full rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a matrix $\mathbf{F} \in \mathbb{Z}_q^{n \times \bar{m}}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, a basis $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ of $\Lambda_q^{\perp}(\mathbf{A})$, and a Gaussian parameter $\sigma \geq \|\mathbf{T_A}\|_{\mathrm{GS}} \cdot \omega\left(\sqrt{\log m}\right)$, outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+\bar{m}}$ sampled from a distribution statistically close to $\mathcal{D}_{\Lambda_q^{\mathbf{u}}([\mathbf{A}|\mathbf{F}]), \sigma}$.*

([24]): There exists a fixed full rank matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ such that the lattice $\Lambda_q^{\perp}(\mathbf{G})$ has a publicly known basis $\mathbf{T_G} \in \mathbb{Z}^{m \times m}$ with $\|\mathbf{T_G}\|_{\mathrm{GS}} \leq \sqrt{5}$.

For simplicity, we omit the $\mathsf{SamplePre}$ algorithm of [1], since in our paper it will be used as a public algorithm to sample from the lattice $\mathbb{Z}^m$. The following algorithms allow one to securely delegate a trapdoor of a lattice to an arbitrary higher-dimensional extension, with a slight loss in quality. It can be obtained by combining the works of [8] and [1] in a straightforward manner.

Lemma 4. *Let $n, m, \bar{m}, q > 0$ be positive integers with $m > n$ and q a prime. Then, we have the following polynomial time algorithms:*

$\mathsf{ExtRndLeft}(\mathbf{A}, \mathbf{F}, \mathbf{T_A}, \sigma) \rightarrow \mathbf{T}_{[\mathbf{A}|\mathbf{F}]}$: *a randomized algorithm that, given as input matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{F} \in \mathbb{Z}_q^{n \times \bar{m}}$, a basis $\mathbf{T_A}$ of $\Lambda_q^{\perp}(\mathbf{A})$, and a Gaussian parameter $\sigma \geq \|\mathbf{T_A}\|_{\mathrm{GS}} \cdot \omega(\sqrt{\log n})$, outputs a matrix $\mathbf{T}_{[\mathbf{A}|\mathbf{F}]} \in \mathbb{Z}^{(m+\bar{m}) \times (m+\bar{m})}$ distributed statistically close to $(\mathcal{D}_{\Lambda_q^{\perp}([\mathbf{A}|\mathbf{F}]), \sigma})^{m+\bar{m}}$.*

$\mathsf{ExtRndRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}, \mathbf{T_G}, \sigma) \rightarrow \mathbf{T}_{[\mathbf{A}|\mathbf{AR}+\mathbf{G}]}$: *a randomized algorithm that, given as input full rank matrices $\mathbf{A}, \mathbf{G} \in \mathbb{Z}_q^{n \times m}$, a matrix $\mathbf{R} \in \mathbb{Z}^{m \times m}$, a basis $\mathbf{T_G}$ of $\Lambda_q^{\perp}(\mathbf{G})$, and a Gaussian parameter $\sigma \geq \|\mathbf{R}\|_2 \cdot \|\mathbf{T_G}\|_2 \cdot \omega(\sqrt{\log n})$ outputs a matrix $\mathbf{T}_{[\mathbf{A}|\mathbf{AR}+\mathbf{G}]} \in \mathbb{Z}^{2m \times 2m}$ distributed statistically close to $(\mathcal{D}_{\Lambda_q^{\perp}([\mathbf{A}|\mathbf{AR}+\mathbf{G}]), \sigma})^{2m}$.*

We use the standard map to encode identities as matrices in $\mathbb{Z}_q^{n \times n}$.

Definition 1 ([1]). *Let n, q be positive integers with q a prime. We say that a function $H : \mathbb{Z}_q^n \rightarrow \mathbb{Z}_q^{n \times n}$ is a full-rank difference (FRD) map if: for all distinct $\mathsf{ID}, \mathsf{ID}' \in \mathbb{Z}_q^n$, the matrix $H(\mathsf{ID}) - H(\mathsf{ID}') \in \mathbb{Z}_q^{n \times n}$ is full rank, and H is computable in polynomial time in $n \log q$.*

Hardness Assumption. The security of our RIBE scheme is reduced to the learning with errors (LWE) assumption introduced by Regev [31].

Assumption 1 (Learning with Errors) *For integers n, m, a prime q, a real $\alpha \in (0,1)$ such that $\alpha q > 2\sqrt{n}$, and a PPT algorithm $\mathcal{A}$, the advantage for the learning with errors problem $\mathsf{LWE}_{n,m,q,\mathcal{D}_{\mathbb{Z}^m,\alpha q}}$ of $\mathcal{A}$ is defined as $\left|\Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{A}^\top \mathbf{s} + \mathbf{x}) = 1\right] - \Pr\left[\mathcal{A}(\mathbf{A}, \mathbf{v} + \mathbf{x}) = 1\right]\right|$, where $\mathbf{A} \leftarrow \mathbb{Z}^{n \times m}$, $\mathbf{s} \leftarrow \mathbb{Z}^n$, $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\alpha q}$, $\mathbf{v} \leftarrow \mathbb{Z}^m$. We say that the LWE assumption holds if the above advantage is negligible for all PPT $\mathcal{A}$.*

4 Formal Definitions for Revocable Hierarchical Identity-Based Encryption and a Supporting Lemma

In this section, we give formal definitions for RHIBE in Section 4.1. Then, in Section 4.2, we explain a simple and yet handy lemma that we call the "strategy-dividing lemma", which helps us simplify security proofs of R(H)IBE schemes in general.

4.1 Revocable Hierarchical Identity-Based Encryption

As mentioned in the introduction, we re-formalize the syntax of RHIBE. Compared to the existing works on RHIBE, our syntax of RHIBE treats each user's secret key, state information, and revocation list in a simplified manner. Thus, we first explain our treatments of them, and then proceed to introducing the formal syntax and security definitions.

On the Role of a Secret Key. In the literature of R(H)IBE, typically, the entity who has the power to derive a secret key for lower-level users (i.e., the KGC in RIBE, and non-leaf users in RHIBE), is modeled as a stateful entity, and is supposed to maintain a so-called "state", in addition to its own secret key. The state information typically contains the information with which the revocation mechanism is realized, and needs to be treated confidentially. Since it is after all another type of secret information, in our syntax, we merge the roles of the state information and a secret key. Hence, in our model, each user is supposed to maintain its own secret key that is generated by its parent, and it could be updated after performing the key generation algorithm (for generating a secret key for its child) and the key update information generation algorithm.

On the Treatment of Revocation Lists. Note that unlike in standard revocable (non-hierarchical) IBE, the key update information and revocation lists of users are maintained individually by their corresponding parent users in RHIBE. In our syntax of R(H)IBE, we treat a revocation list just as a subset of (the corresponding children's) identity space. More specifically, the revocation list of a user with identity $\mathsf{ID} \in (\mathcal{ID})^\ell$ contains identities that belong to the set $\mathsf{ID} \| \mathcal{ID} \subseteq (\mathcal{ID})^{\ell+1}$.

In the literature, for R(H)IBE, it is typical to consider the "revoke" algorithm whose role is to add an identity of a user to be revoked into the revocation list.

We do not explicitly introduce such an algorithm as part of our syntax, since it is a simple operation of appending revoked users to a list.

Syntax. An RHIBE scheme Π consists of the six algorithms ($\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{GenSK}, \mathsf{KeyUp}, \mathsf{GenDK}, \mathsf{Decrypt}$) with the following interface:

$\mathsf{Setup}(1^\lambda, L) \to (\mathsf{PP}, \mathsf{sk}_{\mathsf{kgc}})$: This is the *setup* algorithm that takes the security parameter 1^λ and the maximum depth of the hierarchy $L \in \mathbb{N}$ as input, and outputs a public parameter PP and the KGC's secret key $\mathsf{sk}_{\mathsf{kgc}}$ (also called a master secret key).
We assume that the plaintext space $\mathcal{M}$, the time period space $\mathcal{T} := \{1, 2, \ldots, \mathsf{t}_{\max}\}$, where $\mathsf{t}_{\max}$ is polynomial in λ, the element identity space $\mathcal{ID}$, and the hierarchical identity space $\mathcal{ID}_{\mathsf{h}} := (\mathcal{ID})^{\leq L}$ are determined only by the security parameter λ, and their descriptions are contained in PP.

$\mathsf{Encrypt}(\mathsf{PP}, \mathsf{ID}, \mathsf{t}, \mathsf{M}) \to \mathsf{ct}$: This is the *encryption* algorithm that takes a public parameter PP, an identity ID, a time period t, and a plaintext M as input, and outputs a ciphertext ct.

$\mathsf{GenSK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{pa(ID)}}, \mathsf{ID}) \to (\mathsf{sk}_{\mathsf{ID}}, \mathsf{sk}'_{\mathsf{pa(ID)}})$: This is the *secret key generation* algorithm that takes a public parameter PP, a parent's secret key $\mathsf{sk}_{\mathsf{pa(ID)}}$, and an identity $\mathsf{ID} \in \mathcal{ID}_{\mathsf{h}}$ as input, and may update the parent's secret key $\mathsf{sk}_{\mathsf{pa(ID)}}$. Then, it outputs a secret key $\mathsf{sk}_{\mathsf{ID}}$ for the identity ID and also the parent's "updated" secret key $\mathsf{sk}'_{\mathsf{pa(ID)}}$.

$\mathsf{KeyUp}(\mathsf{PP}, \mathsf{t}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}}, \mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}) \to (\mathsf{ku}_{\mathsf{ID},\mathsf{t}}, \mathsf{sk}'_{\mathsf{ID}})$: This is the *key update information generation* algorithm that takes a public parameter PP, a time period t, a secret key $\mathsf{sk}_{\mathsf{ID}}$ (of a user with $\mathsf{ID} \in (\mathcal{ID})^{\leq L-1} \cup \{\mathsf{kgc}\}$), a revocation list $\mathsf{RL}_{\mathsf{ID},\mathsf{t}} \subseteq \mathsf{ID} \| \mathcal{ID}$, and a parent's key update $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ as input, and may update the secret key $\mathsf{sk}_{\mathsf{ID}}$. Then, it outputs a key update $\mathsf{ku}_{\mathsf{ID},\mathsf{t}}$ and also the "updated" secret key $\mathsf{sk}'_{\mathsf{ID}}$.
In the special case $\mathsf{ID} = \mathsf{kgc}$, we define $\mathsf{ku}_{\mathsf{pa(kgc)},\mathsf{t}} := \bot$ for all $\mathsf{t} \in \mathcal{T}$, i.e., a key update is not needed for generating the KGC's key update $\mathsf{ku}_{\mathsf{kgc},\mathsf{t}}$.

$\mathsf{GenDK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}) \to \mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ or $\bot$: This is the *decryption key generation* algorithm that takes a public parameter PP, a secret key $\mathsf{sk}_{\mathsf{ID}}$ (of a user with $\mathsf{ID} \in (\mathcal{ID})^{\leq L}$), and a parent's key update $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ as input, and outputs a decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ for time period t or the special "invalid" symbol $\bot$ indicating that ID or some of its ancestor has been revoked.

$\mathsf{Decrypt}(\mathsf{PP}, \mathsf{dk}_{\mathsf{ID},\mathsf{t}}, \mathsf{ct}) \to \mathsf{M}$: This is the *decryption* algorithm that takes a public parameter PP, a decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$, and a ciphertext ct as input, and outputs the decryption result M.

Correctness. We require the following to hold for an RHIBE scheme. Informally, we require a ciphertext corresponding to a user ID for time t to be properly decrypted by user ID if the user or any of its ancestor is not revoked on time t. To fully capture this, we consider all the possible scenarios of creating the secret key for user ID. Namely, for all $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, $(\mathsf{PP}, \mathsf{sk}_{\mathsf{kgc}}) \leftarrow \mathsf{Setup}(1^\lambda, L)$, $\ell \in [L]$, $\mathsf{ID} \in (\mathcal{ID})^\ell$, $\mathsf{t} \in \mathcal{T}$, $\mathsf{M} \in \mathcal{M}$, $\mathsf{RL}_{\mathsf{kgc},\mathsf{t}} \subseteq \mathcal{ID}$, $\mathsf{RL}_{\mathsf{ID}_{[1]},\mathsf{t}} \subseteq \mathsf{ID}_{[1]} \| \mathcal{ID}, \ldots, \mathsf{RL}_{\mathsf{ID}_{[\ell-1]},\mathsf{t}} \subseteq \mathsf{ID}_{[\ell-1]} \| \mathcal{ID}$, if $\mathsf{ID}' \notin \mathsf{RL}_{\mathsf{pa(ID')},\mathsf{t}}$ holds for all

$\mathsf{ID}' \in \mathsf{prefix}(\mathsf{ID})$, then we require $\mathsf{M}' = \mathsf{M}$ to hold after executing the following procedures:

(1) $(\mathsf{ku}_{\mathsf{kgc},\mathsf{t}}, \mathsf{sk}_{\mathsf{kgc}}) \leftarrow \mathsf{KeyUp}(\mathsf{PP}, \mathsf{t}, \mathsf{sk}_{\mathsf{kgc}}, \mathsf{RL}_{\mathsf{kgc},\mathsf{t}}, \perp)$.
(2) For all $\mathsf{ID}' \in \mathsf{prefix}(\mathsf{ID})$ (in the short-to-long order), execute (2.1) and (2.2):
  (2.1) $(\mathsf{sk}_{\mathsf{ID}'}, \mathsf{sk}'_{\mathsf{pa}(\mathsf{ID}')}) \leftarrow \mathsf{GenSK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{pa}(\mathsf{ID}')}, \mathsf{ID}')$.
  (2.2) $(\mathsf{ku}_{\mathsf{ID}',\mathsf{t}}, \mathsf{sk}'_{\mathsf{ID}'}) \leftarrow \mathsf{KeyUp}(\mathsf{PP}, \mathsf{t}, \mathsf{sk}_{\mathsf{ID}'}, \mathsf{RL}_{\mathsf{ID}',\mathsf{t}}, \mathsf{ku}_{\mathsf{pa}(\mathsf{ID}'),\mathsf{t}})$.[6]
(3) $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} \leftarrow \mathsf{GenDK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}})$.[7]
(4) $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{PP}, \mathsf{ID}, \mathsf{t}, \mathsf{M})$.
(5) $\mathsf{M}' \leftarrow \mathsf{Decrypt}(\mathsf{PP}, \mathsf{dk}_{\mathsf{ID},\mathsf{t}}, \mathsf{ct})$.

We note that, the most stringent way to define correctness would be to also capture the fact that the secret keys $\mathsf{sk}_{\mathsf{ID}}$ can be further updated after executing GenSK. In particular, the output of KeyUp, which takes as input the secret key $\mathsf{sk}_{\mathsf{ID}}$, may differ in general before and after GenSK is run. Therefore, to be more precise, we should also allow an arbitrary (polynomial) number of executions of GenSK in between steps (2.1) and (2.2). However, we defined correctness as above for the sake of simplicity and readability. We note that our scheme satisfies the more stringent correctness (which will be obvious from the construction).

Security Definition. Here, we give a formal security definition for RHIBE.

It seems to us that since the previous security definitions [6,33,34,35] have some ambiguous treatment in the security game, it was up to the readers to interpret the definitions and the proofs. Therefore, in our work, we provide a refined security definition for RHIBE which in particular is a more rigorous and explicit treatment than the previous definitions.

Specifically, we explicitly separate the secret key generation and secret key reveal queries, so that we can capture a situation where some $\mathsf{sk}_{\mathsf{ID}}$ has been generated but not revealed to an adversary. Furthermore, we combine the "revoke" and "key update" queries in the previous definitions into the single "revoke & key update" query, and introduce the notion of the "current time period" $\mathsf{t}_{\mathsf{cu}} \in \mathcal{T}$ which is coordinated with the adversary's revoke & key update query. These make all the key updates of non-revoked users to be well-defined throughout the security game.

Formally, let $\Pi = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{GenSK}, \mathsf{KeyUp}, \mathsf{GenDK}, \mathsf{Decrypt})$ be an RHIBE scheme. We will only consider selective-identity security, which is defined via a game between an adversary $\mathcal{A}$ and the challenger $\mathcal{C}$. The game is parameterized by the security parameter λ and a polynomial $L = L(\lambda)$ representing the maximum depth of the identity hierarchy. Moreover, the game has the global counter t_{cu}, initialized with 1, that denotes the "current time period" with which $\mathcal{C}$'s responses to $\mathcal{A}$'s queries are controlled. The game proceeds as follows:

At the beginning, $\mathcal{A}$ sends the challenge identity/time period pair $(\mathsf{ID}^*, \mathsf{t}^*) \in (\mathcal{ID})^{\leq L} \times \mathcal{T}$ to $\mathcal{C}$. Next, $\mathcal{C}$ runs $(\mathsf{PP}, \mathsf{sk}_{\mathsf{kgc}}) \leftarrow \mathsf{Setup}(1^\lambda, L)$, and prepares a list

[6] If $|\mathsf{ID}'| = L$, then this step is skipped.
[7] Here, $\mathsf{sk}_{\mathsf{ID}}$ is the latest secret key that is the result of the step (2).

$\mathtt{SKList}$ that initially contains $(\mathsf{kgc}, \mathsf{sk}_{\mathsf{kgc}})$, and into which identity/secret key pairs $(\mathsf{ID}, \mathsf{sk}_{\mathsf{ID}})$ generated during the game will be stored. From this point on, whenever a new secret key is generated or an existing secret key is updated for an identity $\mathsf{ID} \in (\mathcal{ID})^{\leq L} \cup \{\mathsf{kgc}\}$ due to the execution of GenSK or KeyUp, $\mathcal{C}$ will store $(\mathsf{ID}, \mathsf{sk}_{\mathsf{ID}})$ or update the corresponding entry $(\mathsf{ID}, \mathsf{sk}_{\mathsf{ID}})$ in $\mathtt{SKList}$, and we will not explicitly mention this addition/update. Then, $\mathcal{C}$ executes $(\mathsf{ku}_{\mathsf{kgc},1}, \mathsf{sk}'_{\mathsf{kgc}}) \leftarrow \mathsf{KeyUp}(\mathsf{PP}, \mathsf{t}_{\mathsf{cu}} = 1, \mathsf{sk}_{\mathsf{kgc}}, \mathsf{RL}_{\mathsf{kgc},1} = \emptyset, \perp)$ for generating a key update for the initial time period $\mathsf{t}_{\mathsf{cu}} = 1$. After that, $\mathcal{C}$ gives PP and $\mathsf{ku}_{\mathsf{kgc},1}$ to $\mathcal{A}$.

From this point on, $\mathcal{A}$ may adaptively make the following five types of queries to $\mathcal{C}$:

Secret Key Generation Query: Upon a query $\mathsf{ID} \in (\mathcal{ID})^{\leq L}$ from $\mathcal{A}$, $\mathcal{C}$ checks if $(\mathsf{ID}, *) \notin \mathtt{SKList}$ and $(\mathsf{pa}(\mathsf{ID}), \mathsf{sk}_{\mathsf{pa}(\mathsf{ID})}) \in \mathtt{SKList}$ for some $\mathsf{sk}_{\mathsf{pa}(\mathsf{ID})}$, and returns $\perp$ to $\mathcal{A}$ if this is *not* the case. Otherwise, $\mathcal{C}$ executes $(\mathsf{sk}_{\mathsf{ID}}, \mathsf{sk}'_{\mathsf{pa}(\mathsf{ID})}) \leftarrow \mathsf{GenSK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{pa}(\mathsf{ID})}, \mathsf{ID})$. If $\mathsf{ID} \in (\mathcal{ID})^{\leq L-1}$, then $\mathcal{C}$ furthermore executes $(\mathsf{ku}_{\mathsf{ID},\mathsf{t}_{\mathsf{cu}}}, \mathsf{sk}'_{\mathsf{ID}}) \leftarrow \mathsf{KeyUp}(\mathsf{PP}, \mathsf{t}_{\mathsf{cu}}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}_{\mathsf{cu}}} = \emptyset, \mathsf{ku}_{\mathsf{pa}(\mathsf{ID}),\mathsf{t}_{\mathsf{cu}}})$. Then, $\mathcal{C}$ returns $\mathsf{ku}_{\mathsf{ID},\mathsf{t}_{\mathsf{cu}}}$ to $\mathcal{A}$ if $\mathsf{ID} \in (\mathcal{ID})^{\leq L-1}$, or returns nothing to $\mathcal{A}$ if $\mathsf{ID} \in (\mathcal{ID})^{L}$.[8]

We require that all identities ID appearing in the following queries (except the challenge query) be "activated", in the sense that $\mathsf{sk}_{\mathsf{ID}}$ is generated via this query and hence $(\mathsf{ID}, \mathsf{sk}_{\mathsf{ID}}) \in \mathtt{SKList}$.

Secret Key Reveal Query: Upon a query $\mathsf{ID} \in (\mathcal{ID})^{\leq L}$ from $\mathcal{A}$, $\mathcal{C}$ checks if the following condition is satisfied:

- If $\mathsf{t}_{\mathsf{cu}} \geq \mathsf{t}^*$ and $\mathsf{ID}' \notin \mathsf{RL}_{\mathsf{pa}(\mathsf{ID}'),\mathsf{t}^*}$ for all $\mathsf{ID}' \in \mathsf{prefix}(\mathsf{ID}^*)$, then $\mathsf{ID} \notin \mathsf{prefix}(\mathsf{ID}^*)$.[9]

If this condition is *not* satisfied, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ finds $\mathsf{sk}_{\mathsf{ID}}$ from $\mathtt{SKList}$, and returns it to $\mathcal{A}$.

Revoke & Key Update Query: Upon a query $\mathsf{RL} \subseteq (\mathcal{ID})^{\leq L}$ (which denotes the set of identities that are going to be revoked in the next time period) from $\mathcal{A}$, $\mathcal{C}$ checks if the following conditions are satisfied simultaneously:

- $\mathsf{RL}_{\mathsf{ID},\mathsf{t}_{\mathsf{cu}}} \subseteq \mathsf{RL}$ for all $\mathsf{ID} \in \mathcal{ID}^{\leq L-1} \cup \{\mathsf{kgc}\}$ that appear in $\mathtt{SKList}$.[10]
- For all identities ID such that $(\mathsf{ID}, *) \in \mathtt{SKList}$ and $\mathsf{ID}' \in \mathsf{prefix}(\mathsf{ID})$, if $\mathsf{ID}' \in \mathsf{RL}$ then $\mathsf{ID} \in \mathsf{RL}$.[11]

[8] We stress that just making this query does not give the secret key $\mathsf{sk}_{\mathsf{ID}}$ to $\mathcal{A}$. It is captured by the "Secret Key Reveal Query" explained next. Furthermore, we provide the key updates to $\mathcal{A}$ unconditionally, since they are typically broadcast via an insecure channel and are not meant to be secret.

[9] In other words, this check ensures that if ID^* or any of its ancestors was *not* revoked before the challenge time period t^*, then $\mathsf{sk}_{\mathsf{ID}}$ will not be revealed for any $\mathsf{ID} \in \mathsf{prefix}(\mathsf{ID}^*)$. Without this condition, there is a trivial attack on any RHIBE scheme.

[10] This check ensures that the identities that have already been revoked will remain revoked in the next time period.

[11] In other words, this check ensures that if some ID is revoked, then all of its descendants are also revoked.

- If $\mathsf{t_{cu}} = \mathsf{t}^* - 1$ and $\mathsf{sk_{ID'}}$ for some $\mathsf{ID'} \in \mathsf{prefix}(\mathsf{ID}^*)$ has already been revealed by the secret key reveal query $\mathsf{ID'}$, then $\mathsf{ID'} \in \mathsf{RL}$. [12]

If these conditions are *not* satisfied, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$.
Otherwise $\mathcal{C}$ increments the current time period by $\mathsf{t_{cu}} \leftarrow \mathsf{t_{cu}} + 1$. Then, $\mathcal{C}$ executes the following operations (1) and (2) for all "activated" and non-revoked identities ID, i.e., $\mathsf{ID} \in (\mathcal{ID})^{\leq L-1} \cup \{\mathsf{kgc}\}$, $(\mathsf{ID}, *) \in \mathtt{SKList}$, and $\mathsf{ID} \notin \mathsf{RL}$, in the breadth-first order in the identity hierarchy:

(1) Set $\mathsf{RL_{ID,t_{cu}}} \leftarrow \mathsf{RL} \cap (\mathsf{ID} \| \mathcal{ID})$, where we define $\mathsf{kgc} \| \mathcal{ID} := \mathcal{ID}$.
(2) Run $(\mathsf{ku_{ID,t_{cu}}}, \mathsf{sk'_{ID}}) \leftarrow \mathsf{KeyUp}(\mathsf{PP}, \mathsf{t_{cu}}, \mathsf{sk_{ID}}, \mathsf{RL_{ID,t_{cu}}}, \mathsf{ku_{pa(ID),t_{cu}}})$, where $\mathsf{ku_{pa(kgc),t_{cu}}} := \perp$.

Finally, $\mathcal{C}$ returns all the generated key updates $\{\mathsf{ku_{ID,t_{cu}}}\}_{(\mathsf{ID},*) \in \mathtt{SKList}}$ to $\mathcal{A}$.

Decryption Key Reveal Query: Upon a query $(\mathsf{ID}, \mathsf{t}) \in (\mathcal{ID})^{\leq L} \times \mathcal{T}$ from $\mathcal{A}$, $\mathcal{C}$ checks if the following conditions are simultaneously satisfied:

- $\mathsf{t} \leq \mathsf{t_{cu}}$.
- $\mathsf{ID} \notin \mathsf{RL_{pa(ID),t}}$
- $(\mathsf{ID}, \mathsf{t}) \neq (\mathsf{ID}^*, \mathsf{t}^*)$.[13]

If these conditions are *not* satisfied, then $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ finds $\mathsf{sk_{ID}}$ from $\mathtt{SKList}$, runs $\mathsf{dk_{ID,t}} \leftarrow \mathsf{GenDK}(\mathsf{PP}, \mathsf{sk_{ID}}, \mathsf{ku_{pa(ID),t}})$, and returns $\mathsf{dk_{ID,t}}$ to $\mathcal{A}$.[14]

Challenge Query: $\mathcal{A}$ is allowed to make this query only once. Upon a query $(\mathsf{M}_0, \mathsf{M}_1)$ from $\mathcal{A}$, where it is required that $|\mathsf{M}_0| = |\mathsf{M}_1|$, $\mathcal{C}$ picks the challenge bit $b \in \{0, 1\}$ uniformly at random, runs $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{PP}, \mathsf{ID}^*, \mathsf{t}^*, \mathsf{M}_b)$, and returns the challenge ciphertext ct^* to $\mathcal{A}$.

At some point, $\mathcal{A}$ outputs $b' \in \{0, 1\}$ as its guess for b and terminates.

The above completes the description of the game. In this game, $\mathcal{A}$'s selective-identity security advantage $\mathsf{Adv}^{\mathtt{RHIBE\text{-}sel}}_{\Pi,L,\mathcal{A}}(\lambda)$ is defined by $\mathsf{Adv}^{\mathtt{RHIBE\text{-}sel}}_{\Pi,L,\mathcal{A}}(\lambda) := 2 \cdot |\Pr[b' = b] - 1/2|$.

Definition 2. *We say that an RHIBE scheme Π with depth L satisfies* selective-identity security*, if the advantage $\mathsf{Adv}^{\mathtt{RHIBE\text{-}sel}}_{\Pi,L,\mathcal{A}}(\lambda)$ is negligible for all PPT adversaries $\mathcal{A}$.*

[12] In other words, this check is to ensure that if the secret key $\mathsf{sk_{ID'}}$ of some ancestor $\mathsf{ID'}$ of ID^* (or ID^* itself) has been revealed to $\mathcal{A}$, then $\mathsf{ID'}$ is revoked in the next time period.

[13] In previous works [33,35], $\mathcal{A}$ is disallowed to obtain not only $\mathsf{dk_{ID^*,t^*}}$ (which is clearly necessary to avoid a trivial attack), but also decryption keys $\mathsf{dk_{ID',t^*}}$ for all $\mathsf{ID'} \in \mathsf{prefix}(\mathsf{ID}^*)$. Our relaxed condition here makes the defined security stronger since $\mathcal{A}$ is able to obtain additional information without any restrictions.

[14] Note that $\mathsf{ku_{pa(ID),t}}$ must have been already generated at this point due to the condition $\mathsf{t} \leq \mathsf{t_{cu}}$.

4.2 Strategy-Dividing Lemma

In the literature of R(H)IBE, a typical security proof for an R(H)IBE scheme goes as follows:

(1) classify an adversary's strategies into multiple pre-determined types, say Type-1 to Type-n for some $n \in \mathbb{N}$ that cover all possible strategies, and
(2) for each $i \in [n]$, prove that any adversary that is promised to follow the Type-i strategy (and never break the promise) has negligible advantage in attacking the considered scheme.

Here, it is implicitly assumed that the above mentioned "type-classification-based" security proof is sufficient for proving security against arbitrary adversaries that may decide their attack strategies adaptively during the game.

For completeness, we formalize the above implicit argument as a simple yet handy "strategy-dividing lemma", which helps us simplify security proofs for R(H)IBE schemes in general. Since this is an implicit argument that has been frequently adopted in the R(H)IBE literatures, we provide it in the full version.

5 Generic Construction of RIBE with DKER

In this section, we show a "security-enhancing" generic construction for RIBE. Namely, we show how to construct an RIBE scheme with DKER by combining an RIBE scheme without DKER and a 2-level (non-revocable) HIBE scheme.

Let $\mathsf{r}.\Pi = (\mathsf{r.Setup}, \mathsf{r.Encrypt}, \mathsf{r.GenSK}, \mathsf{r.KeyUp}, \mathsf{r.GenDK}, \mathsf{r.Decrypt})$ be an RIBE scheme (without DKER) with identity space $\mathsf{r}.\mathcal{ID}$, plaintext space $\mathsf{r}.\mathcal{M}$, and time period space $\mathsf{r}.\mathcal{T}$. Let $\mathsf{h}.\Pi = (\mathsf{h.Setup}, \mathsf{h.Encrypt}, \mathsf{h.GenSK}, \mathsf{h.Delegate}, \mathsf{h.Decrypt})$ be a 2-level HIBE scheme with element identity space $\mathsf{h}.\mathcal{ID}$ and plaintext space $\mathsf{h}.\mathcal{M}$. We assume $\mathsf{r}.\mathcal{ID} = \mathsf{h}.\mathcal{ID}$, $\mathsf{r}.\mathcal{M} = \mathsf{h}.\mathcal{M}$, and $\mathsf{r}.\mathcal{T} \subseteq \mathsf{h}.\mathcal{ID}$. Furthermore, we assume that the plaintext space is finite and forms an abelian group with the addition "+" as the group operation.

Using these ingredients, we construct an RIBE scheme $\Pi = (\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{GenSK}, \mathsf{KeyUp}, \mathsf{GenDK}, \mathsf{Decrypt})$ with DKER as follows. The identity space $\mathcal{ID}$, the plaintext space $\mathcal{M}$, and the time period space $\mathcal{T}$ of the constructed RIBE scheme Π are, respectively, $\mathcal{ID} = \mathsf{r}.\mathcal{ID} = \mathsf{h}.\mathcal{ID}$, $\mathcal{M} = \mathsf{r}.\mathcal{M} = \mathsf{h}.\mathcal{M}$, and $\mathcal{T} = \mathsf{r}.\mathcal{T} \subseteq \mathsf{h}.\mathcal{ID}$.

$\mathsf{Setup}(1^\lambda) \to (\mathsf{PP}, \mathsf{sk}_{\mathsf{kgc}})$: It takes the security parameter 1^λ as input, and runs $(\mathsf{r.PP}, \mathsf{r.sk}_{\mathsf{kgc}}) \leftarrow \mathsf{r.Setup}(1^\lambda)$ and $(\mathsf{h.PP}, \mathsf{h.sk}_{\mathsf{kgc}}) \leftarrow \mathsf{h.Setup}(1^\lambda)$. Then, it outputs a public parameter $\mathsf{PP} := (\mathsf{r.PP}, \mathsf{h.PP})$ and the KGC's secret key $\mathsf{sk}_{\mathsf{kgc}} := (\mathsf{r.sk}_{\mathsf{kgc}}, \mathsf{h.sk}_{\mathsf{kgc}})$.

$\mathsf{Encrypt}(\mathsf{PP}, \mathsf{ID}, \mathsf{t}, \mathsf{M}) \to \mathsf{ct}$: It takes a public parameter $\mathsf{PP} = (\mathsf{r.PP}, \mathsf{h.PP})$, an identity $\mathsf{ID} \in \mathcal{ID}$, a time period $\mathsf{t} \in \mathcal{T}$, and a plaintext $\mathsf{M} \in \mathcal{M}$ as input, and samples a pair $(\mathsf{r.M}, \mathsf{h.M}) \in \mathcal{M}^2$ uniformly at random, subject to $\mathsf{r.M} + \mathsf{h.M} = \mathsf{M}$. Then, it runs $\mathsf{r.ct} \leftarrow \mathsf{r.Encrypt}(\mathsf{r.PP}, \mathsf{ID}, \mathsf{t}, \mathsf{r.M})$ and $\mathsf{h.ct} \leftarrow \mathsf{h.Encrypt}(\mathsf{h.PP}, (\mathsf{ID}, \mathsf{t}), \mathsf{h.M})$. Finally, it outputs a ciphertext $\mathsf{ct} := (\mathsf{r.ct}, \mathsf{h.ct})$.

$\mathsf{GenSK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{kgc}}, \mathsf{ID}) \to (\mathsf{sk}_{\mathsf{ID}}, \mathsf{sk}'_{\mathsf{kgc}})$: It takes a public parameter $\mathsf{PP} = (\mathsf{r.PP}, \mathsf{h.PP})$, the KGC's secret key $\mathsf{sk}_{\mathsf{kgc}} = (\mathsf{r.sk}_{\mathsf{kgc}}, \mathsf{h.sk}_{\mathsf{kgc}})$, and an identity $\mathsf{ID} \in \mathcal{ID}$ as input, and runs $(\mathsf{r.sk}_{\mathsf{ID}}, \mathsf{r.sk}'_{\mathsf{kgc}}) \leftarrow \mathsf{r.GenSK}(\mathsf{r.PP}, \mathsf{r.sk}_{\mathsf{kgc}}, \mathsf{ID})$ and $\mathsf{h.sk}_{\mathsf{ID}} \leftarrow \mathsf{h.GenSK}(\mathsf{h.PP}, \mathsf{h.sk}_{\mathsf{kgc}}, \mathsf{ID})$. Then, it outputs a secret key $\mathsf{sk}_{\mathsf{ID}} := (\mathsf{r.sk}_{\mathsf{ID}}, \mathsf{h.sk}_{\mathsf{ID}})$ for the identity ID and also the KGC's updated secret key $\mathsf{sk}'_{\mathsf{kgc}} := (\mathsf{r.sk}'_{\mathsf{kgc}}, \mathsf{h.sk}_{\mathsf{kgc}})$.

$\mathsf{KeyUp}(\mathsf{PP}, \mathsf{t}, \mathsf{sk}_{\mathsf{kgc}}, \mathsf{RL}_{\mathsf{t}}) \to (\mathsf{ku}_{\mathsf{t}}, \mathsf{sk}'_{\mathsf{kgc}})$: It takes a public parameter $\mathsf{PP} = (\mathsf{r.PP}, \mathsf{h.PP})$, a time period $\mathsf{t} \in \mathcal{T}$, the KGC's secret key $\mathsf{sk}_{\mathsf{kgc}} = (\mathsf{r.sk}_{\mathsf{kgc}}, \mathsf{h.sk}_{\mathsf{kgc}})$, and a revocation list $\mathsf{RL}_{\mathsf{t}} \subseteq \mathcal{ID}$ as input, and, runs $(\mathsf{r.ku}_{\mathsf{t}}, \mathsf{r.sk}'_{\mathsf{kgc}}) \leftarrow \mathsf{r.KeyUp}(\mathsf{r.PP}, \mathsf{t}, \mathsf{r.sk}_{\mathsf{kgc}}, \mathsf{RL}_{\mathsf{t}})$. Then, it outputs a key update $\mathsf{ku}_{\mathsf{t}} := \mathsf{r.ku}_{\mathsf{t}}$ and also the KGC's updated secret key $\mathsf{sk}'_{\mathsf{kgc}} := (\mathsf{r.sk}'_{\mathsf{kgc}}, \mathsf{h.sk}_{\mathsf{kgc}})$.

$\mathsf{GenDK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{ku}_{\mathsf{t}}) \to \mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ or $\perp$: It takes a public parameter $\mathsf{PP} = (\mathsf{r.PP}, \mathsf{h.PP})$, a secret key $\mathsf{sk}_{\mathsf{ID}} = (\mathsf{r.sk}_{\mathsf{ID}}, \mathsf{h.sk}_{\mathsf{ID}})$, and a key update $\mathsf{ku}_{\mathsf{t}} = \mathsf{r.ku}_{\mathsf{t}}$ as input, and runs $\mathsf{r.dk}_{\mathsf{ID},\mathsf{t}} \leftarrow \mathsf{r.GenDK}(\mathsf{r.PP}, \mathsf{r.sk}_{\mathsf{ID}}, \mathsf{r.ku}_{\mathsf{t}})$ and $\mathsf{h.sk}_{\mathsf{ID},\mathsf{t}} \leftarrow \mathsf{h.Delegate}(\mathsf{h.PP}, \mathsf{h.sk}_{\mathsf{ID}}, \mathsf{t})$. Then, it outputs a decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} := (\mathsf{r.dk}_{\mathsf{ID},\mathsf{t}}, \mathsf{h.sk}_{\mathsf{ID},\mathsf{t}})$ for time period t, except that if $\mathsf{r.dk}_{\mathsf{ID},\mathsf{t}} = \perp$, then it returns the special "invalid" symbol $\perp$ indicating that ID has been revoked.

$\mathsf{Decrypt}(\mathsf{PP}, \mathsf{dk}_{\mathsf{ID},\mathsf{t}}, \mathsf{ct}) \to \mathsf{M}$: It takes a public parameter $\mathsf{PP} = (\mathsf{r.PP}, \mathsf{h.PP})$, a decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} = (\mathsf{r.dk}_{\mathsf{ID},\mathsf{t}}, \mathsf{h.sk}_{\mathsf{ID},\mathsf{t}})$, and a ciphertext $\mathsf{ct} = (\mathsf{r.ct}, \mathsf{h.ct})$ as input, and then runs $\mathsf{r.M} \leftarrow \mathsf{r.Decrypt}(\mathsf{r.PP}, \mathsf{r.dk}_{\mathsf{ID},\mathsf{t}}, \mathsf{r.ct})$ and $\mathsf{h.M} \leftarrow \mathsf{h.Decrypt}(\mathsf{h.PP}, \mathsf{h.sk}_{\mathsf{ID},\mathsf{t}}, \mathsf{h.ct})$. If $\mathsf{r.M} = \perp$ or $\mathsf{h.M} = \perp$, then it returns $\perp$. Otherwise, it outputs the decryption result $\mathsf{M} := \mathsf{r.M} + \mathsf{h.M}$.

It is immediate to see that the correctness of the constructed RIBE scheme Π follows from that of the building blocks. The security of Π is guaranteed by the following theorem.

Theorem 1. *If the underlying RIBE scheme* $\mathsf{r}.\Pi$ *satisfies weak selective-identity (resp. weak adaptive-identity) security and the underlying 2-level HIBE scheme* $\mathsf{h}.\Pi$ *satisfies selective-identity (resp. adaptive-identity) security, then the resulting RIBE scheme* Π *satisfies selective-identity (resp. adaptive-identity) security.*

Proof Overview. Here, we explain an overview of the proof. In the actual proof, we consider the following two attack strategies of an adversary against the RIBE scheme Π that are mutually exclusive and cover all possibilities:

- Type-I: The adversary issues a valid secret key reveal query on ID^*.
- Type-II: The adversary does not issue a valid secret key reveal query on ID^*.

Whether an adversary has deviated from one strategy is easy to detect. Due to the strategy-dividing lemma, it suffices to show that for each type of adversary (that is promised to follow the attack strategy), its advantage is negligible. In particular, we show that the security of the RIBE scheme Π against Type-I (resp. Type-II) adversary is guaranteed by the security of the underlying RIBE scheme $\mathsf{r}.\Pi$ (resp. 2-level HIBE scheme $\mathsf{h}.\Pi$).

6 RHIBE from Lattices

In this section, we first explain our treatment on binary trees, the CS method, and the parameters used in the scheme. Then, we show our proposed scheme in Section 6.1 and discuss the security in Section 6.2.

On the Treatment of Binary Trees and the CS Method. Every user ID such that $|\mathsf{ID}| \leq L-1$ (including KGC) maintains a binary tree $\mathtt{BT}_{\mathsf{ID}}$ as part of his secret key $\mathsf{sk}_{\mathsf{ID}}$. We assume that auxiliary information such as user identities ID and vectors in $\mathbb{Z}_q^n$ can be stored in the nodes of binary trees. The binary tree along with the CS method is the mechanism used by the parent to manage its children, i.e., keep track whether a child is revoked or not. We use θ to denote a node in a binary tree. We use η when we emphasize that the node θ is a leaf node. Let $\mathsf{Path}(\mathtt{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}})$ denote the set of nodes which are on the path along the root of $\mathtt{BT}_{\mathsf{pa(ID)}}$ to the leaf η_{ID}. Note that the size of $\mathsf{Path}(\mathtt{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}})$ is $O(\log N)$. We define the CS method by the following four algorithms:

- $\mathsf{CS.SetUp}(N) \to \mathtt{BT}_{\mathsf{pa(ID)}}$: It takes the number of users N as input, and outputs a binary tree $\mathtt{BT}_{\mathsf{pa(ID)}}$ with at least N and at most $2N$ leaves.
- $\mathsf{CS.Assign}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{ID}) \to (\eta_{\mathsf{ID}}, \mathtt{BT}_{\mathsf{pa(ID)}})$: It takes a binary tree $\mathtt{BT}_{\mathsf{pa(ID)}}$ and an identity ID as inputs, and randomly assigns the user identity ID to a leaf node η_{ID}, to which no other IDs have been assigned yet. Then, it outputs a leaf η_{ID} and an "updated" binary tree $\mathtt{BT}_{\mathsf{pa(ID)}}$.
- $\mathsf{CS.Cover}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}}) \to \mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$: It takes a binary tree $\mathtt{BT}_{\mathsf{pa(ID)}}$ and a revocation list $\mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}}$ as inputs, and outputs a set of nodes $\mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$. Here, the subtrees with root $\theta \in \mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$ cover all leaves η_{ID} in $\mathtt{BT}_{\mathsf{pa(ID)}}$ for $\mathsf{ID} \notin \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}}$ and do not cover any leaves η_{ID} for $\mathsf{ID} \in \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}}$.
- $\mathsf{CS.Match}(\mathsf{Path}(\mathtt{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}}), \mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})) \to \theta$ or $\emptyset$: It takes $\mathsf{Path}(\mathtt{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}})$ and $\mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$ as inputs, and outputs an arbitrary node $\theta \in \mathsf{Path}(\mathtt{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}}) \cap \mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$ if it exists. Otherwise, it outputs $\emptyset$.

Looking ahead, at a high level, all parents maintain the children to whom it has generated secret keys by the binary tree $\mathtt{BT}_{\mathsf{pa(ID)}}$. The secret keys $\mathsf{sk}_{\mathsf{ID}}$ will include some (partial) secret information that are associated with a node in $\mathsf{Path}(\mathtt{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}})$. To revoke a set of users $\mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}}$, the parent constructs the key update $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ by running $\mathsf{CS.Cover}$ and generates a set of nodes $\mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$, which represents the set of users that are *not* revoked. Similarly to above, each node in $\mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$ will include some (partial) secret information. We note that the size of $\mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$ is $O(R\log(N/R))$, where $R = \left|\mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}}\right|$. Notably, the size of the key update $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ will be logarithmic in N. Then, any user ID who is not revoked can run the $\mathsf{CS.Match}$ algorithm to obtain a node θ which is included both in $\mathsf{Path}(\mathtt{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}})$ and $\mathsf{KUNode}(\mathtt{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$. Combining the two partial secret information embedded in the nodes, user ID will be able to construct the decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ which allows him to decrypt the ciphertext.

Parameters. Let L denote the maximum depth of the hierarchy and N denote the maximum number of children each parent manages. Furthermore, let n, m, q be positive integers such that q is a prime and $\alpha, \alpha', (\sigma_i)_{i=0}^{L}$ be positive reals denoting the Gaussian parameters. Finally, we set the plaintext space as $\mathcal{M} = \{0,1\}$, the element identity space as $\mathcal{ID} = \mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}$, and the hierarchal identity space as $\mathcal{ID}_{\mathtt{h}} := (\mathbb{Z}_q^n \setminus \{\mathbf{0}_n\})^{\leq L}$. We also encode the time period space $\mathcal{T} = \{1, 2, \cdots, \mathsf{t}_{\max}\}$ into a polynomial sized subset of $\mathbb{Z}_q^n$. In the following, for readability, we may simply address each space $\mathcal{ID}, \mathcal{ID}_{\mathtt{h}}, \mathcal{T}$ as $\mathcal{T} = \mathcal{ID} = \mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}, \mathcal{ID}_{\mathtt{h}} = (\mathbb{Z}_q^n \setminus \{\mathbf{0}_n\})^{\leq L}$, unless stated otherwise.

6.1 Construction

We provide our RHIBE scheme below. The intuition of the construction follows the explanation given in Section 2. Due to the complex nature of our scheme, we encourage readers to go back to Section 2 whenever needed.

$\mathsf{Setup}(1^n, L) \to (\mathsf{PP}, \mathsf{sk}_{\mathsf{kgc}})$: The setup algorithm is run by the KGC. It takes the security parameter 1^n and the maximum depth of the hierarchy L as input, and runs $(\mathbf{A}_i, \mathbf{T}_{\mathbf{A}_i}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$ for $i \in [L+1]$. It also samples uniformly random matrices $(\mathbf{B}_j)_{j\in[L+1]} \leftarrow (\mathbb{Z}_q^{n\times m})^{(L+1)}$ and vectors $(\mathbf{u}_k)_{k\in[L]} \leftarrow (\mathbb{Z}_q^n)^L$. Finally, it creates a binary tree by running $\mathsf{BT}_{\mathsf{kgc}} \leftarrow \mathsf{CS.SetUp}(N)$ and outputs

$$\mathsf{PP} := \Big((\mathbf{A}_i)_{i\in[L+1]}, (\mathbf{B}_j)_{j\in[L+1]}, (\mathbf{u}_k)_{k\in[L]}\Big), \quad \mathsf{sk}_{\mathsf{kgc}} := \Big(\mathsf{BT}_{\mathsf{kgc}}, (\mathbf{T}_{\mathbf{A}_i})_{i\in[L+1]}\Big).$$

Recall here that the matrices $\mathbf{B}_j$ define the hash functions $\mathbf{E}(\cdot)$ and $\mathbf{F}(\cdot)$ stated in Eq. (1) in Section 2.

$\mathsf{Encrypt}(\mathsf{PP}, \mathsf{ID} = (\mathsf{id}_1, \ldots, \mathsf{id}_\ell), \mathsf{t}, \mathsf{M}) \to \mathsf{ct}$: On input an identity $\mathsf{ID} \in (\mathbb{Z}_q^n)^\ell$ at depth $\ell \in [L]$ and time period $\mathsf{t} \in \mathbb{Z}_q^n$, it first samples $\ell+1$ uniformly random vectors $(\mathbf{s}_i)_{i\in[\ell]}, \mathbf{s}_{L+1} \in \mathbb{Z}_q^n$. Then it samples $x \leftarrow D_{\mathbb{Z},\alpha q}, \mathbf{x}_i \leftarrow D_{\mathbb{Z}^{(i+2)m},\alpha' q}$ for $i \in [\ell]$ and $\mathbf{x}_{L+1} \leftarrow D_{\mathbb{Z}^{(\ell+2)m},\alpha' q}$, and sets

$$\begin{cases} c_0 = \mathbf{u}_\ell^\top(\mathbf{s}_1 + \cdots + \mathbf{s}_\ell + \mathbf{s}_{L+1}) + x + \mathsf{M}\left\lfloor \frac{q}{2} \right\rfloor, \\ \mathbf{c}_i = [\mathbf{A}_i | \mathbf{E}(\mathsf{ID}_{[i]}) | \mathbf{F}(\mathsf{t})]^\top \mathbf{s}_i + \mathbf{x}_i \quad \text{for} \quad i \in [\ell], \\ \mathbf{c}_{L+1} = [\mathbf{A}_{L+1} | \mathbf{E}(\mathsf{ID}) | \mathbf{F}(\mathsf{t})]^\top \mathbf{s}_{L+1} + \mathbf{x}_{L+1}. \end{cases}$$

Finally, it outputs a ciphertext $\mathsf{ct} := (c_0, \mathbf{c}_1, \ldots, \mathbf{c}_\ell, \mathbf{c}_{L+1}) \in \mathbb{Z}_q \times \mathbb{Z}_q^{3m} \times \cdots \times \mathbb{Z}_q^{(\ell+2)m} \times \mathbb{Z}_q^{(\ell+2)m}$.

$\mathsf{GenSK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{pa}(\mathsf{ID})}, \mathsf{ID}) \to (\mathsf{sk}_{\mathsf{ID}}, \mathsf{sk}'_{\mathsf{pa}(\mathsf{ID})})$: The secret key generation algorithm is run by a parent user $\mathsf{pa}(\mathsf{ID})$ at level $\ell - 1$, where $1 \leq \ell \leq L$, to create a secret key for its child ID.[15] It first runs $(\mathsf{BT}_{\mathsf{pa}(\mathsf{ID})}, \eta_{\mathsf{ID}}) \leftarrow \mathsf{CS.Assign}(\mathsf{BT}_{\mathsf{pa}(\mathsf{ID})}, \mathsf{ID})$. Then, for each node $\theta \in \mathsf{Path}(\mathsf{BT}_{\mathsf{pa}(\mathsf{ID})}, \eta_{\mathsf{ID}})$, it checks whether a vector

[15] Recall that a user at level 0 corresponds to the kgc, i.e., for any level-1 user $\mathsf{ID} \in \mathbb{Z}_q^n \setminus \{\mathbf{0}_n\}$, $\mathsf{pa}(\mathsf{ID}) = \mathsf{kgc}$.

$\mathbf{u}_{\mathsf{pa(ID)},\theta} \in \mathbb{Z}_q^n$ has already been assigned. If not, pick a uniformly random vector $\mathbf{u}_{\mathsf{pa(ID)},\theta} \in \mathbb{Z}_q^n$ and update $\mathsf{sk}_{\mathsf{pa(ID)}}$ by storing $\mathbf{u}_{\mathsf{pa(ID)},\theta}$ in node $\theta \in \mathsf{BT}_{\mathsf{pa(ID)}}$. Next, it samples vectors $\mathbf{e}_{\mathsf{ID},\theta}, \mathbf{f}_{\mathsf{ID},k} \in \mathbb{Z}^{(\ell+1)m}$ for $\theta \in \mathsf{Path}(\mathsf{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}}), k \in [\ell{+}1, L]$, respectively, such that $[\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})]\mathbf{e}_{\mathsf{ID},\theta} = \mathbf{u}_{\mathsf{pa(ID)},\theta}, [\mathbf{A}_\ell|\mathbf{E}(\mathsf{ID})]\mathbf{f}_{\mathsf{ID},k} = \mathbf{u}_k - \mathbf{u}_\ell$by running $\mathsf{SampleLeft}(\cdot)$ with trapdoor $\mathbf{T}_{[\mathbf{A}_\ell|\mathbf{E}(\mathsf{pa(ID)})]}$[16] and Gaussian parameter σ_ℓ. Then, it extends its bases by running the following algorithm for $i \in [\ell+1, L+1]$: $\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]} \leftarrow \mathsf{ExtRndLeft}([\mathbf{A}_i|\mathbf{E}(\mathsf{pa(ID)})], \mathbf{B}_\ell + H(\mathsf{id}_\ell)\mathbf{G}, \mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{pa(ID)})]}, \sigma_{\ell-1})$,where $\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]} \in \mathbb{Z}^{(\ell+1)m\times(\ell+1)m}$. Here, recall that $\mathbf{E}(\mathsf{ID}) = [\mathbf{E}(\mathsf{pa(ID)})|\mathbf{B}_\ell + H(\mathsf{id}_\ell)\mathbf{G}]$. Finally, it runs $\mathsf{BT}_{\mathsf{ID}} \leftarrow \mathsf{CS.SetUp}(N)$ and outputs,

$$\mathsf{sk}_{\mathsf{ID}} = \begin{pmatrix} \mathsf{BT}_{\mathsf{ID}}, \mathsf{Path}(\mathsf{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}}), \ (\mathbf{e}_{\mathsf{ID},\theta})_{\theta\in\mathsf{Path}(\mathsf{BT}_{\mathsf{pa(ID)}},\eta_{\mathsf{ID}})}, \\ (\mathbf{f}_{\mathsf{ID},k})_{k\in[\ell+1,L]}, \ (\mathbf{T}_{[\mathbf{A}_i|\mathbf{E}(\mathsf{ID})]})_{i\in[\ell+1,L+1]} \end{pmatrix}$$

along with its updated secret key $\mathsf{sk}'_{\mathsf{pa(ID)}}$.

$\mathsf{KeyUp}(\mathsf{PP}, \mathsf{t}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}}, \mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}) \to (\mathsf{ku}_{\mathsf{ID},\mathsf{t}}, \mathsf{sk}'_{\mathsf{ID}})$: The key update information generation algorithm is run by user ID at level ℓ, where $0 \le \ell \le L-1$, to create a key update $\mathsf{ku}_{\mathsf{ID},\mathsf{t}}$ for time period t for its children. It first runs $\mathsf{KUNode}(\mathsf{BT}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}}) \leftarrow \mathsf{CS.Cover}(\mathsf{BT}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}})$, and checks whether $\mathbf{u}_{\mathsf{ID},\theta}$ is defined for each node $\theta \in \mathsf{KUNode}(\mathsf{BT}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}})$. If not, it picks a random $\mathbf{u}_{\mathsf{ID},\theta} \in \mathbb{Z}_q^n$ and updates $\mathsf{sk}_{\mathsf{ID}}$ by storing $\mathbf{u}_{\mathsf{ID},\theta}$ in the node $\theta \in \mathsf{BT}_{\mathsf{ID}}$. Then, for each node θ, it samples $\mathbf{e}_{\mathsf{ID},\mathsf{t},\theta} \in \mathbb{Z}^{(\ell+2)m}$ such that $[\mathbf{A}_{\ell+1}|\mathbf{E}(\mathsf{ID})|\mathbf{F}(\mathsf{t})]\mathbf{e}_{\mathsf{ID},\mathsf{t},\theta} = \mathbf{u}_{\ell+1} - \mathbf{u}_{\mathsf{ID},\theta}$ by running $\mathsf{SampleLeft}(\cdot)$ with trapdoor $\mathbf{T}_{[\mathbf{A}_{\ell+1}|\mathbf{E}(\mathsf{ID})]}$ and Gaussian parameter $\sigma_{\ell+1}$.

At this point, the algorithm behaves differently depending on $\ell \ge 1$ or $\ell = 0$ (i.e., $\mathsf{ID} = \mathsf{kgc}$). In case $\ell \ge 1$, it computes its own decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$, which includes a vector $\mathbf{d}_{\mathsf{ID},\mathsf{t}} \in \mathbb{Z}^{(\ell+2)m}$, using the decryption key generation algorithm $\mathsf{GenDK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{ku}_{\mathsf{pa(ID,t)}})$ defined below, and computes the following vectors for $k \in [\ell+1, L]$: $\mathbf{f}_{\mathsf{ID},\mathsf{t},k} = \mathbf{d}_{\mathsf{ID},\mathsf{t}} + [\mathbf{f}_{\mathsf{ID},k} \| \mathbf{0}_m] \in \mathbb{Z}^{(\ell+2)m}$. Here, $[\cdot\|\cdot]$ denotes vertical concatenation of vectors.

Finally, it extracts $(\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},k} \in \mathbb{Z}^{(i+2)m})_{(i,k)\in[\ell-1]\times[\ell+1,L]}$ from its ancestor's key update information $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ and outputs

$$\mathsf{ku}_{\mathsf{ID},\mathsf{t}} = \begin{pmatrix} \mathsf{KUNode}(\mathsf{BT}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}}), (\mathbf{e}_{\mathsf{ID},\mathsf{t},\theta})_{\theta\in\mathsf{KUNode}(\mathsf{BT}_{\mathsf{ID}},\mathsf{RL}_{\mathsf{ID},\mathsf{t}})}, \\ (\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},k})_{(i,k)\in[\ell]\times[\ell+1,L]} \end{pmatrix}$$

and the possibly updated $\mathsf{sk}'_{\mathsf{ID}}$.

In case $\ell = 0$, it skips all the above procedures and simply outputs

$$\mathsf{ku}_{\mathsf{ID},\mathsf{t}} = (\mathsf{KUNode}(\mathsf{BT}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}}), (\mathbf{e}_{\mathsf{ID},\mathsf{t},\theta})_{\theta\in\mathsf{KUNode}(\mathsf{BT}_{\mathsf{ID}},\mathsf{RL}_{\mathsf{ID}})})$$

and the possibly updated $\mathsf{sk}'_{\mathsf{ID}}$.[17]

[16] There are two exceptions for this algorithm. In the special case $\mathsf{ID} = \mathsf{kgc}$, recall that we set $\mathbf{T}_{[\mathbf{A}_1|\mathbf{E}(\mathsf{kgc})]}$ as $\mathbf{T}_{\mathbf{A}_1}$, which is included in the $\mathsf{sk}_{\mathsf{kgc}}$. In the other special case when $\ell = L$, we no longer sample $\mathbf{f}_{\mathsf{ID},k}$, since this vector is only required for delegating key updates to its children, which users at level L do not have.

[17] The branch in the algorithm is due to the fact that for the special case $\ell = 0$, i.e., $\mathsf{ID} = \mathsf{kgc}$, we have $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}} = \perp$ for all $\mathcal{T}$ and there exists no decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}}$.

$\mathsf{GenDK}(\mathsf{PP}, \mathsf{sk}_{\mathsf{ID}}, \mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}) \to \mathsf{dk}_{\mathsf{ID},\mathsf{t}}$ or $\perp$: The decryption key generation algorithm is run by user ID at level ℓ, where $1 \leq \ell \leq L$. It extracts $\mathsf{Path}(\mathsf{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}})$ in $\mathsf{sk}_{\mathsf{ID}}$ and $\mathsf{KUNode}(\mathsf{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}})$ in $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$, and runs $\theta/\emptyset \leftarrow \mathsf{CS.Match}(\mathsf{Path}(\mathsf{BT}_{\mathsf{pa(ID)}}, \eta_{\mathsf{ID}}), \mathsf{KUNode}(\mathsf{BT}_{\mathsf{pa(ID)}}, \mathsf{RL}_{\mathsf{pa(ID)},\mathsf{t}}))$. If the output is $\emptyset$, it outputs $\perp$. Otherwise, it extracts $\mathbf{e}_{\mathsf{ID},\theta}, \mathbf{e}_{\mathsf{pa(ID)},\mathsf{t},\theta} \in \mathbb{Z}^{(\ell+1)m}$ in $\mathsf{sk}_{\mathsf{ID}}, \mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$, respectively, and parses it as

$$\mathbf{e}_{\mathsf{ID},\theta} = [\mathbf{e}^{\mathsf{L}}_{\mathsf{ID},\theta} \| \mathbf{e}^{\mathsf{R}}_{\mathsf{ID},\theta}], \quad \mathbf{e}_{\mathsf{pa(ID)},\mathsf{t},\theta} = [\mathbf{e}^{\mathsf{L}}_{\mathsf{pa(ID)},\mathsf{t},\theta} \| \mathbf{e}^{\mathsf{R}}_{\mathsf{pa(ID)},\mathsf{t},\theta}],$$

where $\mathbf{e}^{\mathsf{L}}_{\mathsf{ID},\theta}, \mathbf{e}^{\mathsf{L}}_{\mathsf{pa(ID)},\mathsf{t},\theta} \in \mathbb{Z}^{\ell m}$ and $\mathbf{e}^{\mathsf{R}}_{\mathsf{ID},\theta}, \mathbf{e}^{\mathsf{R}}_{\mathsf{pa(ID)},\mathsf{t},\theta} \in \mathbb{Z}^{m}$. Then, it computes

$$\mathbf{d}_{\mathsf{ID},\mathsf{t}} = [\mathbf{e}^{\mathsf{L}}_{\mathsf{ID},\theta} + \mathbf{e}^{\mathsf{L}}_{\mathsf{pa(ID)},\mathsf{t},\theta} \| \mathbf{e}^{\mathsf{R}}_{\mathsf{ID},\theta} \| \mathbf{e}^{\mathsf{R}}_{\mathsf{pa(ID)},\mathsf{t},\theta}] \in \mathbb{Z}^{(\ell+2)m}.$$

It further samples $\mathbf{g}_{\mathsf{ID},\mathsf{t}} \in \mathbb{Z}^{(\ell+2)m}$ such that $[\mathbf{A}_{L+1} | \mathbf{E}(\mathsf{ID}) | \mathbf{F}(\mathsf{t})]\mathbf{g}_{\mathsf{ID},\mathsf{t}} = \mathbf{u}_\ell$ by running $\mathsf{SampleLeft}(\cdot)$ with trapdoor $\mathbf{T}_{[\mathbf{A}_{L+1}|\mathbf{E}(\mathsf{ID})]}$ and Gaussian parameter σ_ℓ.
Finally, in case $\ell \geq 2$, it extracts $(\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell})_{i\in[\ell-1]}$ from $\mathsf{ku}_{\mathsf{pa(ID)},\mathsf{t}}$ and outputs $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} = (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, (\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell})_{i\in[\ell-1]}, \mathbf{g}_{\mathsf{ID},\mathsf{t}})$. Otherwise, in case $\ell = 1$, it simply outputs $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} = (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, \mathbf{g}_{\mathsf{ID},\mathsf{t}})$.

$\mathsf{Decrypt}(\mathsf{PP}, \mathsf{dk}_{\mathsf{ID},\mathsf{t}}, \mathsf{ct}) \to \mathsf{M}$: The decryption algorithm is run by user ID at level ℓ, where $1 \leq \ell \leq L$. It first parses the ciphertext ct as $(c_0, \mathbf{c}_1, \cdots, \mathbf{c}_\ell, \mathbf{c}_{L+1})$. Then, in case $\ell \geq 2$, it uses its decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} = (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, (\mathbf{f}_{\mathsf{ID}_{[i]},\mathsf{t},\ell})_{i\in[\ell-1]}, \mathbf{g}_{\mathsf{ID},\mathsf{t}})$ and computes

$$c' = c_0 - \sum_{i=1}^{\ell-1} \mathbf{f}^{\top}_{\mathsf{ID}_{[i]},\mathsf{t},\ell}\mathbf{c}_i - \mathbf{d}^{\top}_{\mathsf{ID},\mathsf{t}}\mathbf{c}_\ell - \mathbf{g}^{\top}_{\mathsf{ID},\mathsf{t}}\mathbf{c}_{L+1} \in \mathbb{Z}_q. \tag{5}$$

Otherwise, in case $\ell = 1$, it uses its decryption key $\mathsf{dk}_{\mathsf{ID},\mathsf{t}} = (\mathbf{d}_{\mathsf{ID},\mathsf{t}}, \mathbf{g}_{\mathsf{ID},\mathsf{t}})$ and computes

$$c' = c_0 - \mathbf{d}^{\top}_{\mathsf{ID},\mathsf{t}}\mathbf{c}_1 - \mathbf{g}^{\top}_{\mathsf{ID},\mathsf{t}}\mathbf{c}_{L+1} \in \mathbb{Z}_q.$$

Finally, it compares c' and $\lfloor \frac{q}{2} \rfloor$ treating them as integers in $\mathbb{Z}$, and outputs 1 in case $|c' - \lfloor \frac{q}{2} \rfloor| < \lfloor \frac{q}{4} \rfloor$ and 0 otherwise.

Correctness. Let a ciphertext be aimed for user ID and time period t. To check correctness, we only need to consider the case where all the ancestors of ID are not revoked. In other words, we check that user ID will be able to obtain all the required components to construct the decryption key $\mathbf{d}_{\mathsf{ID},\mathsf{t}}$ when provided with all the key updates $\mathsf{ku}_{\mathsf{ID}',\mathsf{t}}$ from $\mathsf{ID}' \in \mathsf{prefix}(\mathsf{ID}) \backslash \{\mathsf{ID}\}$.

Lemma 5. *Assume $O((\alpha + mL^2\sigma_L\alpha')q) \leq q/5$ holds with overwhelming probability. Then the above scheme has negligible decryption error.*

Remarks. Note that for simplicity we defined correctness of RHIBE to hold with probability one in Section 4. Therefore, to be consistent with our definition, we can use standard techniques to modify our lattice-based construction to have no decryption error by considering a bound on the secret/noise vectors.

6.2 Security

Theorem 2. *The above RHIBE scheme Π is selective-identity secure assuming the hardness of the* $\mathsf{LWE}_{n,m+1,q,\chi}$ *problem, where* $\chi = D_{\mathbb{Z}^{m+1},\alpha q}$.

Proof Overview. Here, we provide an overview of the proof. Let $\mathcal{A}$ be a PPT adversary that attacks the selective-identity security of the RHIBE scheme Π with non-negligible advantage. In addition, let $(\mathsf{ID}^* = (\mathsf{id}_1^*, \ldots, \mathsf{id}_{\ell^*}^*), \mathsf{t}^*)$ be the challenge identity/time period pair that $\mathcal{A}$ sends to the challenger at the beginning of the game. Similarly to the RIBE adversary in Section 5, the strategy taken by $\mathcal{A}$ can be divided into the following two types that are mutually exclusive, where the first type can be further divided into ℓ types of strategies that are mutually exclusive:

- Type-I: $\mathcal{A}$ issues secret key reveal queries on at least one $\mathsf{ID} \in \mathsf{prefix}(\mathsf{ID}^*)$.
  - Type-I-i^*: $\mathcal{A}$ issues a secret key reveal query on $\mathsf{ID}^*_{[i^*]}$ but not on any $\mathsf{ID} \in \mathsf{prefix}(\mathsf{ID}^*_{[i^*-1]})$.
- Type-II: $\mathcal{A}$ does not issue secret key reveal queries on any $\mathsf{ID} \in \mathsf{prefix}(\mathsf{ID}^*)$.

Due to the strategy-dividing lemma, it suffices to prove security against each type of adversary independently. In our proof we provide two types of security reduction: one for when $\mathcal{A}$ follows the Type-I-i^* ($1 \leq i^* \leq \ell^*$) strategy and another for when $\mathcal{A}$ follows the Type-II strategy. Let us provide a brief overview of the reduction when we are against a Type-I-i^* adversary $\mathcal{A}$. The general idea holds for Type-II adversaries as well.

Our goal is to modify the challenger through a sequence of games so that in the end he would be able to simulate the game against the Type-I-i^* adversary $\mathcal{A}$ using only the trapdoors $\{\mathbf{T}_{\mathbf{A}}\}_{i \in [L+1] \setminus \{i^*\}}$. At a high level, this allows the challenger to embed his LWE challenge into the matrix $\mathbf{A}_{i^*}$ included in the public parameter PP. The following Table 1 depicts all the possible scenarios where the challenger requires the trapdoor $\mathbf{T}_{\mathbf{A}_{i^*}}$, either implicitly or explicitly, in the real game to respond to $\mathcal{A}$'s queries. For readers familiar with the RIBE

	$\mathsf{ID} \in (\mathcal{ID})^{i^*}$	$\mathsf{ID} \in (\mathcal{ID})^{i^*-1}$	(In case $i^* \geq 3$) $\mathsf{ID} \in (\mathcal{ID})^{\leq i^*-2}$
Secret Key Generation $(\mathsf{sk}_{\mathsf{ID}})$	$(\mathbf{e}_{\mathsf{ID},\theta})_{\theta \in \mathsf{Path}(\mathsf{BT}_{\mathsf{pa}(\mathsf{ID})}, \eta_{\mathsf{ID}})}$ $(\mathbf{f}_{\mathsf{ID},k})_{k \in [i^*+1, L]}$	$\mathbf{T}_{[\mathbf{A}_{i^*} \mid \mathbf{E}(\mathsf{ID})]}$	$\mathbf{T}_{[\mathbf{A}_{i^*} \mid \mathbf{E}(\mathsf{ID})]}$
Revoke & Key Update $(\mathsf{ku}_{\mathsf{ID},\mathsf{t}})$	$(\mathbf{f}_{\mathsf{ID},\mathsf{t},k})_{k \in [i^*+1, L]}$	$(\mathbf{e}_{\mathsf{ID},\mathsf{t},\theta})_{\theta \in \mathsf{KUNode}(\mathsf{BT}_{\mathsf{ID}}, \mathsf{RL}_{\mathsf{ID},\mathsf{t}})}$	–
Decryption Key Reveal $(\mathsf{dk}_{\mathsf{ID},\mathsf{t}})$	$\mathbf{d}_{\mathsf{ID},\mathsf{t}}$	–	–

Table 1: Items for which the challenger requires $\mathbf{T}_{\mathbf{A}_{i^*}}$ to construct.

scheme without DKER of Chen et al. [10], it may be helpful to point out that the way we modify the challenger so that he no longer requires $\mathbf{T}_{\mathbf{A}_{i^*}}$ to construct $(\mathbf{e}_{\mathsf{ID},\theta})_\theta$ in the secret key generation query and $(\mathbf{e}_{\mathsf{ID},\mathsf{t},\theta})_\theta$ in the revoke & key

update query is very similar to the technique used in [10]. This is mainly because these components are those responsible for achieving the revocation mechanism. Our proof deviates from prior works when we modify the challenger so that he no longer requires $\mathbf{T}_{\mathbf{A}_{i^*}}$ to construct $(\mathbf{f}_{\mathsf{ID},k})_k$ in the secret key generation query and $(\mathbf{f}_{\mathsf{ID},\mathsf{t},k})_k$ in the revoke & key update query, since these are the newly added components for achieving DKER.

Acknowledgement. The first author was partially supported by JST CREST Grant Number JPMJCR1302 and JSPS KAKENHI Grant Number 17J05603. The second author was partially supported by JST CREST Grant Number JPMJCR1688. The third author was partially supported by JST CREST Grant Number JPMJCR14D6.

References

1. Agrawal, S., Boneh, D., Boyen, X.: Efficient lattice (H)IBE in the standard model. EUROCRYPT 2010. LNCS 6110, pp. 553–572. Springer (2010)
2. Ajtai, M.: Generating hard instances of the short basis problem. ICALP'99. LNCS 1644, pp. 1–9. Springer (1999)
3. Alwen, J., Peikert, C.: Generating shorter bases for hard random lattices. Theory Comput. Syst. 48(3), 535–553 (2011)
4. Attrapadung, N., Imai, H.: Attribute-based encryption supporting direct/indirect revocation modes. Cryptography and Coding 2009. LNCS 5921, pp. 278–300. Springer (2009)
5. Attrapadung, N., Imai, H.: Conjunctive broadcast and attribute-based encryption. Pairing 2009. LNCS 5671, pp. 248–265. Springer (2009)
6. Boldyreva, A., Goyal, V., Kumar, V.: Identity-based encryption with efficient revocation. CCS 2008. pp. 417–426. ACM (2008)
7. Boneh, D., Franklin, M.K.: Identity-based encryption from the weil pairing. SIAM J. Comput. 32(3), 586–615 (2003)
8. Cash, D., Hofheinz, D., Kiltz, E., Peikert, C.: Bonsai trees, or how to delegate a lattice basis. J. Cryptology 25(4), 601–639 (2012)
9. Chang, D., Chauhan, A.K., Kumar, S., Sanadhya, S.K.: Revocable identity-based encryption from codes with rank metric. CT-RSA 2018. LNCS 10808, pp. 435–451. Springer (2018)
10. Chen, J., Lim, H.W., Ling, S., Wang, H., Nguyen, K.: Revocable identity-based encryption from lattices. ACISP 2012. LNCS 7372, pp. 390–403. Springer (2012)
11. Cui, H., Deng, R.H., Li, Y., Qin, B.: Server-aided revocable attribute-based encryption. ESORICS 2016. LNCS 9879, pp. 570–587. Springer (2016)
12. Döttling, N., Garg, S.: From selective IBE to full IBE and selective HIBE. TCC 2017. LNCS 10677, pp. 372–408. Springer (2017)
13. Emura, K., Seo, J.H., Youn, T.: Semi-generic transformation of revocable hierarchical identity-based encryption and its DBDH instantiation. IEICE Transactions 99-A(1), 83–91 (2016)
14. Gentry, C., Peikert, C., Vaikuntanathan, V.: Trapdoors for hard lattices and new cryptographic constructions. STOC 2008. pp. 197–206. ACM (2008)
15. Ishida, Y., Shikata, J., Watanabe, Y.: CCA-secure revocable identity-based encryption schemes with decryption key exposure resistance. IJACT 3(3), 288–311 (2017)

16. Katsumata, S., Yamada, S.: Partitioning via non-linear polynomial functions: More compact ibes from ideal lattices and bilinear maps. ASIACRYPT 2016. LNCS 10032, pp. 682–712 (2016)
17. Lee, K.: Revocable hierarchical identity-based encryption with adaptive security. IACR Cryptology ePrint Archive 2016, 749 (2016)
18. Lee, K., Lee, D.H., Park, J.H.: Efficient revocable identity-based encryption via subset difference methods. Des. Codes Cryptography 85(1), 39–76 (2017)
19. Lee, K., Park, S.: Revocable hierarchical identity-based encryption with shorter private keys and update keys. IACR Cryptology ePrint Archive 2016, 460 (2016)
20. Libert, B., Vergnaud, D.: Adaptive-id secure revocable identity-based encryption. CT-RSA 2009. LNCS 5473, pp. 1–15. Springer (2009)
21. Ling, S., Nguyen, K., Wang, H., Zhang, J.: Revocable predicate encryption from lattices. ProvSec 2017. LNCS 10592, pp. 305–326. Springer (2017)
22. Ling, S., Nguyen, K., Wang, H., Zhang, J.: Server-aided revocable predicate encryption: Formalization and lattice-based instantiation. CoRR abs/1801.07844 (2018)
23. Mao, X., Lai, J., Chen, K., Weng, J., Mei, Q.: Efficient revocable identity-based encryption from multilinear maps. Security and Communication Networks 8(18), 3511–3522 (2015)
24. Micciancio, D., Peikert, C.: Trapdoors for lattices: Simpler, tighter, faster, smaller. EUROCRYPT 2012. LNCS 7237, pp. 700–718. Springer (2012)
25. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. CRYPTO 2001. LNCS 2139, pp. 41–62. Springer (2001)
26. Nguyen, K., Wang, H., Zhang, J.: Server-aided revocable identity-based encryption from lattices. CANS 2016. LNCS 10052, pp. 107–123 (2016)
27. Nieto, J.M.G., Manulis, M., Sun, D.: Fully private revocable predicate encryption. ACISP 2012. LNCS 7372, pp. 350–363. Springer (2012)
28. Park, S., Lee, D.H., Lee, K.: Revocable hierarchical identity-based encryption from multilinear maps. CoRR abs/1610.07948 (2016)
29. Park, S., Lee, K., Lee, D.H.: New constructions of revocable identity-based encryption from multilinear maps. IEEE Trans. Information Forensics and Security 10(8), 1564–1577 (2015)
30. Qin, B., Deng, R.H., Li, Y., Liu, S.: Server-aided revocable identity-based encryption. ESORICS 2015. LNCS 9326, pp. 286–304. Springer (2015)
31. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. STOC 2005. pp. 84–93. ACM (2005)
32. Ryu, G., Lee, K., Park, S., Lee, D.H.: Unbounded hierarchical identity-based encryption with efficient revocation.WISA 2015. LNCS 9503, pp. 122–133. Springer (2015)
33. Seo, J.H., Emura, K.: Revocable identity-based encryption revisited: Security model and construction. PKC 2013. LNCS 7778, pp. 216–234. Springer (2013)
34. Seo, J.H., Emura, K.: Revocable hierarchical identity-based encryption. Theor. Comput. Sci. 542, 44–62 (2014)
35. Seo, J.H., Emura, K.: Revocable hierarchical identity-based encryption via history-free approach. Theor. Comput. Sci. 615, 45–60 (2016)
36. Takayasu, A., Watanabe, Y.: Lattice-based revocable identity-based encryption with bounded decryption key exposure resistance. ACISP 2017. LNCS 10342, pp. 184–204. Springer (2017)
37. Watanabe, Y., Emura, K., Seo, J.H.: New revocable IBE in prime-order groups: Adaptively secure, decryption key exposure resistant, and with short public parameters. CT-RSA 2017. LNCS 10159, pp. 432–449. Springer (2017)