Anonymous Revocable Identity-Based Encryption Supporting Anonymous Revocation

Kwangsu Lee*

Abstract

Anonymous identity-based encryption (AIBE) is an extension of identity-based encryption (IBE) that enhances the privacy of a ciphertext by providing ciphertext anonymity. In this paper, we introduce the concept of revocable IBE with anonymous revocation (RIBE-AR), which is capable of issuing an update key and hiding the revoked set of the update key that efficiently revokes private keys of AIBE. We first define the security models of RIBE-AR and propose an efficient RIBE-AR scheme in bilinear groups. Our RIBE-AR scheme is similar to the existing RIBE scheme in terms of efficiency, but is the first RIBE scheme to provide additional ciphertext anonymity and revocation privacy. We show that our RIBE-AR scheme provides the selective message privacy, selective identity privacy, and selective revocation privacy.

Keywords: Identity-based encryption, Ciphertext anonymity, Key revocation, Revocation privacy.

*Sejong University, Seoul, Korea. Email: `kwangsu@sejong.ac.kr`.

Contents

1 Introduction

An anonymous identity-based encryption (AIBE) scheme that provides ciphertext anonymity is an extension of an identity-based encryption (IBE) scheme that uses an identity string as a public key, and strengthens the privacy of a ciphertext by providing anonymity for the recipient identity included in the ciphertext. The first IBE scheme was proposed by Boneh and Franklin using pairing [10], and their IBE scheme was later found to provide anonymity [1]. Because an AIBE scheme can provide ciphertext anonymity in communication between users, it can be a very useful tool in fully private communication environments [1, 13]. In addition, an AIBE scheme also can be used for a searchable public-key encryption scheme, which is a new type of public key encryption that allows searching keywords in encrypted data [9, 41].

In order to manage the private key of a user in a public-key system, how to effectively revoke the private key is a very important issue. One way to handle the revocation of private keys in an AIBE scheme is to periodically issue an update key similar to a revocable IBE (RIBE) scheme, which efficiently revokes the private key of an IBE scheme [7]. In the RIBE scheme, each user which has a private key SK periodically obtain an update key UK issued by a trusted center. When a sender creates a ciphertext CT associated with a receiver identity ID and current time T for a receiver, the receiver can decrypt the ciphertext by combining his private key with an update key at that time. If the private key of the receiver is not revoked in the update key, the receiver can derive a valid decryption key DK to decrypt the ciphertext. However, in this revocation method, since the update key exposes the revoked set R of users, there is a problem in that the anonymity of the ciphertext can be breached by using this revoked set in the update key since the receiver identity of the ciphertext will not belong to the revoked set.

In this paper, we would like to solve the problem of designing an RIBE scheme that supports the efficient revocation of user private keys in an AIBE scheme, which provides anonymity of ciphertext, while not revealing additional revoked information of users in an update key.

1.1 Our Contributions

We first introduce RIBE with revocation privacy (RIBE-AR) which provides ciphertext anonymity and revocation privacy, and define three security models of RIBE-AR. The syntax of RIBE-AR is almost identical to the syntax of RIBE. In other words, a ciphertext is associated with an identity ID and time T, a private key is associated with an identity ID, and an update key is associated with a revocation set R and time T. As for the security model, the existing RIBE defines only message privacy which is IND-CPA, but the RIBE-AR defines three security models: message privacy (IND-CPA), identity privacy (ANO-CPA), and revocation privacy (REV-PRV). The IND-CPA security guarantees that the message M of a ciphertext is hidden as before, and the ANO-CPA security model guarantees that the identity ID of a ciphertext is hidden. Lastly, the REV-PRV security model guarantees that the revocation set R of an update key is hidden.

Next, we propose an RIBE-AR scheme that provides ciphertext anonymity and revocation privacy in asymmetric bilinear groups and prove the security of our scheme. In order to devise an RIBE-AR scheme that provides ciphertext anonymity only, we first constructed a basic RIBE scheme by combining an AIBE scheme, an IBE scheme, and a tree-based broadcast encryption scheme. However, this basic RIBE scheme derived in this way does not hide the revocation set of an update key. To provide revocation privacy, we additionally encrypt all node update keys in the update key by using a symmetric-key encryption scheme and provide hint values to quickly search for matching nodes. Compared to the existing RIBE scheme, our RIBE-AR scheme has similar efficiency in terms of private key size and update key size, but it requires slightly increased computation in the decryption key derivation. The detailed comparison of RIBE schemes is given in Table 1. Finally, we show that our RIBE-AR scheme satisfies the selective IND-CPA security,

Table 1: Comparison of RIBE schemes in bilinear groups

Scheme	PP Size	SK Size	UK Size	CT Size	ANO, RP, DKER
BF [10]	$O(1)$	$O(1)$	$O(N-r)$	$O(1)$	Yes, No, Yes
BGK [7]	$O(1)$	$O(\log N)$	$O(r\log \frac{N}{r})$	$O(1)$	No, No, No
LV [30]	$O(\lambda)$	$O(\log N)$	$O(r\log \frac{N}{r})$	$O(1)$	No, No, No
SE [38]	$O(\lambda)$	$O(\log N)$	$O(r\log \frac{N}{r})$	$O(1)$	No, No, Yes
LLP [27]	$O(1)$	$O(\log^2 N)$	$O(r)$	$O(1)$	No, No, Yes
WES [40]	$O(1)$	$O(\log N)$	$O(r\log \frac{N}{r})$	$O(1)$	No, No, Yes
Ours	$O(1)$	$O(\log N)$	$O(\lceil r\log \frac{N}{r}\rceil)$	$O(1)$	Yes, Yes, Yes

Let λ be a security parameter, N be the number of all users, and r be the number of revoked users. We count the number of group elements to measure the size. We use symbols ANO for ciphertext anonymity, RP for revocation privacy, and DKER for decryption key exposure resistance.

the selective ANO-CPA security, and the selective REV-PRV security.

1.2 Our Techniques

The well-known method of designing an RIBE scheme using IBE schemes is to combine two IBE schemes and the complete subtree (CS) method [7]. At this time, the private key SK of the RIBE scheme is composed of IBE private keys corresponding to the path nodes of a binary tree, and the update key UK of the RIBE scheme is composed of IBE private keys corresponding to the cover nodes that include non-revoked leaf nodes in the binary tree. The main advantages of this approach to build an RIBE scheme are that a sender can create a ciphertext CT without knowing revoked users, and the size of an update key UK periodically issued by a center does not increase linearly with the number of non-revoked users.

Our RIBE-AR scheme also employs this methodology used to design the existing RIBE scheme. At this time, we combine an AIBE scheme, an IBE scheme, and the CS method to provide ciphertext anonymity. We modify the AHIBE scheme proposed by Ducas [16] as the underlying AIBE scheme to allow private key randomization, and use the IBE scheme of Boneh and Boyen [8] as the underlying IBE scheme. However, this basic RIBE scheme, which simply combines AIBE and IBE schemes with the CS method, does not provide revocation privacy. The reason is that since an update key is publicly known, if we check which update key node is used for decryption in the process of deriving a decryption key by combining the update key and a private key, we can derive a matching node in the update key, which can be used to distinguish revocation sets.

To ensure revocation privacy, we encrypt each node update keys in an update key by using symmetric key encryption. To do this, we set each node v_i in a binary tree to have a symmetric key κ_i and encrypts the corresponding node update key NUK_i with the key κ_i. We also randomly mix all node update keys included in the update key so that additional information of tree nodes is not exposed. In order to decrypt the encrypted node update key, we set each node private key NSK_i associated with a node v_i in a private key to have a symmetric key κ_i. If the node private key NSK_i of SK and the node update key NUK_i of UK relate to the same node v_i, then we can decrypt the encrypted NUK_i with κ_i included in NSK_i and checks the correctness of the decryption. However, this method has the disadvantage of being too slow in deriving a decryption key because all possible combinations of all node private keys and all node update keys should be attempted.

In order to quickly derive a decryption key, we include hint values that allow checking for matching nodes in each node update key. For reference, this method of using hints has already been used in anonymous broadcast encryption schemes [18, 29]. To use hint values, the node private key of a private key includes a random ω_i associated with a node v_i, and the node update key of an update key also includes a hint element $Y_i = V^{\omega_i}$ associated with a node v_i where the element V is a unique random element for each update key. In this case, a receiver who owns a private key has exponent values $(\omega_1, \dots, \omega_n)$ corresponding to the path nodes of a binary tree, and uses the element V of the update key to derive elements $(V^{\omega_1}, \dots, V^{\omega_n})$. Afterwards, the receiver can quickly find a matching node by comparing these derived elements with the hint values $(Y_1, \dots, Y_m)$ included in node update keys of an update key. Note that an attacker who has access to only a limited set of revoked private keys will not be able to distinguish revoked sets even if he obtains hint values.

1.3 Related Work

IBE and AIBE. An identity-based encryption (IBE) scheme was first proposed by Boneh and Franklin in bilinear groups [10], and the security of their scheme was proven under the BDH assumption. The BF-IBE scheme is also the first anonymous IBE (AIBE) scheme that provides ciphertext anonymity by hiding the identity of a ciphertext [1]. Boyen proposed an identity-based sign/encryption (IBSE) scheme with ciphertext anonymity by modifying the BF-IBE scheme [12]. An AIBE scheme can be used for the construction of a public-key encryption with keyword search (PEKS) scheme that supports keyword searches on encrypted data [9,41]. The first AIBE and anonymous HIBE schemes without random oracles were proposed by Boyen and Waters under the decision linear assumption [13]. Gentry also proposed an AIBE scheme without random oracles using a strong q-type assumption [19]. Ducas proposed AIBE and anonymous HIBE schemes using asymmetric bilinear groups [16]. The first lattice-based IBE scheme proposed by Gentry et al. is also an AIBE scheme because it provides anonymity [20]. Both hidden-vector encryption and inner-product encryption schemes, which are extensions of IBE, can be easily converted to AIBE schemes through simple conversion [11, 22].

Revocable IBE. Boneh and Franklin first noticed the private key revocation problem of IBE and proposed a revocation method of periodically reissuing private keys [10]. However, this method has the disadvantage that the private key reissuing increases in proportion to the number of users because each individual user must be issued a new private key periodically. Boldyreva et al. proposed a different method in which a trusted center periodically generates an update key and broadcasts it for non-revoked users [7]. This method has the advantage that the update key size does not proportionally increases depending on the number of users because it uses a tree based broadcast encryption scheme. Seo and Emura updated the previous security model of RIBE schemes by allowing the decryption key queries of an adversary [38]. Lee et al. proposed an efficient RIBE scheme with an improved update key size by using the subset difference method instead of the previous complete subtree method [27]. Many additional studies have been conducted to improve the security or efficiency of RIBE schemes [30, 34, 35, 40]. Another research direction is being conducted on RHIBE schemes that can efficiently handle the revocation of the private key in HIBE schemes by using the similar methods of the RIBE scheme [17, 24, 28, 37, 39]. Recently, generic conversion methods of designing RIBE or RHIBE schemes using IBE or HIBE schemes in a black-box way have been proposed [23, 25, 26, 31], but these approaches are somewhat inefficient compared to direct design approaches.

2 Preliminaries

In this section we review pseudo-random functions, symmetric-key encryption, and the complete subtree method which is one instance of the subset cover framework.

2.1 Pseudo-Random Function

A pseudo-random function (PRF) is an efficiently computable function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ where $\mathcal{K}$ is the key space, $\mathcal{X}$ is the domain, and $\mathcal{Y}$ is the range. Let $F(k, \cdot)$ be an oracle for a uniformly chosen $k \in \mathcal{K}$ and $f(\cdot)$ be an oracle for a uniformly chosen function $f : \mathcal{X} \to \mathcal{Y}$. We say that a PRF is secure if for all probabilistic polynomial-time adversaries $\mathcal{A}$ the advantage $\mathbf{Adv}^{PRF}_{\mathcal{A}}(\lambda) = \big|\Pr[\mathcal{A}^{F(k,\cdot)} = 1] - \Pr[\mathcal{A}^{f(\cdot)} = 1]\big|$ is negligible.

2.2 Symmetric-Key Encryption

Symmetric-key encryption (SKE) is an encryption method in which the sender and receiver use the same symmetric key for encryption and decryption. The syntax of SKE is defined as follows.

Definition 2.1 (Symmetric-Key Encryption, SKE)**.** A symmetric-key encryption (SKE) scheme consists of three algorithms **GenKey**, **Encrypt**, and **Decrypt**, which are defined as follows:

GenKey(1^λ). The key generation algorithm takes as input a security parameter λ. It outputs a symmetric key K.

Encrypt(K,M). The encryption algorithm takes as input a message $M \in \mathcal{M}$ and the symmetric key K. It outputs a ciphertext C.

Decrypt(K,C). The decryption algorithm takes as input a ciphertext CT and the symmetric key K. It outputs a message M or a symbol $\perp$.

The correctness of the SKE scheme is defined as follows: For all K generated by **GenKey** and any message $M \in \mathcal{M}$, it is required that **Decrypt**$(K,$**Encrypt**$(K,M)) = M$.

In general, the IND-CPA security model of the SKE scheme allows multiple challenge ciphertext queries, but we use a simple security model that allows only one challenge ciphertext query. Note that an SKE scheme, which is IND-CPA secure for the model that allows multiple challenge ciphertext queries, is naturally guaranteed to be IND-CPA secure even for the simple model that allow one challenge ciphertext query.

Definition 2.2 (Message Privacy, IND-CPA)**.** The IND-CPA security of SKE is defined in the following experiment $\mathbf{EXP}^{IND\text{-}CPA}_{\mathcal{A}}(1^\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Setup**: $\mathcal{C}$ generates a symmetric key K by running **GenKey**(1^λ). It keeps K to itself.
2. **Phase 1**: $\mathcal{A}$ adaptively request a polynomial number of encryption queries. For each encryption query for a message M, $\mathcal{C}$ generates a ciphertext CT by running **Encrypt**(K,M) and gives CT to $\mathcal{A}$.
3. **Challenge**: $\mathcal{A}$ submits challenge messages M_0^*, M_1^* where $|M_0^*| = |M_1^*|$. $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$ and gives a challenge ciphertext CT^* to $\mathcal{A}$ by running **Encrypt**(K, M_μ^*).
4. **Phase 2**: $\mathcal{A}$ may continues to request additional encryption queries and $\mathcal{C}$ handles these queries as the same as the phase 1.

5. **Guess**: $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ of μ. $\mathcal{C}$ outputs 1 if $\mu = \mu'$ or 0 otherwise.

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{SKE,\mathcal{A}}^{IND\text{-}CPA}(\lambda) = \big|\Pr[\mathbf{EXP}_{\mathcal{A}}^{IND\text{-}CPA}(1^\lambda) = 1] - \frac{1}{2}\big|$. An SKE scheme is IND-CPA secure if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

The key privacy (KEY-PRV) security model of SKE is that an attacker cannot distinguish which of two symmetric keys K_0 and K_1 is used for encryption. We use a simple security model that allows one challenge ciphertext query by modifying the definition of Bellare et al. [6] for PKE into the definition for SKE.

Definition 2.3 (Key Privacy, KEY-PRV [6])**.** The KEY-PRV security of SKE is defined in the following experiment $\mathbf{EXP}_{\mathcal{A}}^{KEY\text{-}PRV}(1^\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Setup**: $\mathcal{C}$ generates symmetric keys K_0, K_1 by running $\mathbf{GenKey}(1^\lambda)$. It keeps K_0, K_1 to itself.

2. **Phase 1**: $\mathcal{A}$ adaptively request a polynomial number of encryption queries. For each encryption query for a message M and a choice $b \in \{0,1\}$, $\mathcal{C}$ generates a ciphertext CT by running $\mathbf{Encrypt}(K_b, M)$ and gives CT to $\mathcal{A}$.

3. **Challenge**: $\mathcal{A}$ submits a challenge message M^*. $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$ and gives a challenge ciphertext CT^* to $\mathcal{A}$ by running $\mathbf{Encrypt}(K_\mu, M^*)$.

4. **Phase 2**: $\mathcal{A}$ may continues to request additional encryption queries and $\mathcal{C}$ handles these queries as the same as the phase 1.

5. **Guess**: $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ of μ. $\mathcal{C}$ outputs 1 if $\mu = \mu'$ or 0 otherwise.

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{SKE,\mathcal{A}}^{KEY\text{-}PRV}(\lambda) = \big|\Pr[\mathbf{EXP}_{\mathcal{A}}^{KEY\text{-}PRV}(1^\lambda) = 1] - \frac{1}{2}\big|$. An SKE scheme is KEY-PRV secure if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

2.3 The Complete Subtree Method

The complete subtree method is one instance of the subset cover revocation framework of Naor et al. [33]. The simplified CS method is given as follows:

CS.Setup(N)**:** Let N be the number of all users where $N = 2^n$ for simplicity. It sets a perfect binary tree $\mathcal{BT}$ of depth n. Each user is assigned to a different leaf node in $\mathcal{BT}$. It outputs the binary tree $\mathcal{BT}$.

CS.Assign$(\mathcal{BT}, v)$**:** Let v be a leaf node assigned to a user. Let $(v_{k_0}, v_{k_1}, \ldots, v_{k_n})$ be the path from the root node $v_{k_0} = v_0$ to the leaf node $v_{k_n} = v$. It initializes a private set $PV = \emptyset$. For all $j \in \{k_0, \ldots, k_n\}$, it adds v_j into PV. It outputs the private set $PV = \{v_j\}$.

CS.Cover$(\mathcal{BT}, R)$**:** Let R be a set of revoked users which consists of leaf nodes. It computes the Steiner tree $\mathcal{ST}_R$. Let $\mathcal{T}_{k_1}, \ldots \mathcal{T}_{k_m}$ be all the subtrees of $\mathcal{BT}$ that hang off $\mathcal{ST}_R$, that is all subtrees whose roots $v_{k_1}, \ldots v_{k_m}$ are not in $\mathcal{ST}_R$ but adjacent to nodes of outdegree 1 in $\mathcal{ST}_R$. It initializes a cover set $CV = \emptyset$. For all $i \in \{k_1, \ldots, k_m\}$, it adds v_i into CV. It outputs the cover set $CV = \{v_i\}$.

CS.Match(CV, PV)**:** It finds a common node v_k with $v_k \in CV$ and $v_k \in PV$. If there exists a common node, it outputs (v_k, v_k). Otherwise, it outputs $\perp$.

The correctness of the CS scheme requires that if $v \notin R$, then $\mathbf{CS.Match}(CV, PV) = (v_k, v_k)$ for the same v_k where CV and PV are associated with R and v respectively.

3 Anonymous RIBE with Anonymous Revocation

In this section, we define the syntax and security models of RIBE-AR that provides message privacy, identity privacy, and revocation privacy.

3.1 Definition

Revocable IBE with anonymous revocation (RIBE-AR) is an extension of IBE and supports the revocation of user private keys through the issuance of update keys [7]. In an RIBE-AR scheme, a trusted center creates a user's private key using the master key and status information and maintains a revocation list. If the private key of a user is revealed or expired, then the trusted center updates the revocation list by including the identity of the revoked user. And the trusted center periodically issues an update key using the revocation list and broadcasts it for non-revoked users. A sender creates a ciphertext by specifying the identity of a recipient and current time. A receiver can derive a decryption key by combining his private key with the update key if his private key is not revoked in the update key. By using the decryption key, the recipient can decrypt the corresponding ciphertext. The more detailed syntax of RIBE is given as follows.

Definition 3.1 (RIBE with Anonymous Revocation, RIBE-AR)**.** An RIBE-AR scheme consists of seven algorithms **Setup**, **GenKey**, **UpdateKey**, **DeriveKey**, **Encrypt**, **Decrypt**, and **Revoke**, which are defined as follows:

Setup$(1^\lambda, N)$: The setup algorithm takes as input a security parameter 1^λ and the maximum number of users N. It outputs a master key MK, an (empty) revocation list RL, and public parameters PP.

GenKey(ID, MK, ST, PP): The private key generation algorithm takes as input an identity $ID \in \mathcal{I}$, the master key MK, and public parameters PP. It outputs a private key SK_{ID}.

UpdateKey(T, RL, MK, ST, PP): The update key generation algorithm takes as input update time $T \in \mathcal{T}$, the revocation list RL, the master key MK, and public parameters PP. It outputs an update key UK_T.

DeriveKey(SK_{ID}, UK_T, PP): The decryption key derivation algorithm takes as input a private key SK_{ID}, an update key UK_T, and public parameters PP. It outputs a decryption key $DK_{ID,T}$.

Encrypt(ID, T, M, PP): The encryption algorithm takes as input an identity $ID \in \mathcal{I}$, time T, a message $M \in \mathcal{M}$, and public parameters PP. It outputs a ciphertext CT.

Decrypt$(CT, DK_{ID',T'}, PP)$: The decryption algorithm takes as input a ciphertext CT, a decryption key $DK_{ID',T'}$, and public parameters PP. It outputs a message M.

Revoke(ID, T, RL): The revocation algorithm takes as input an identity ID to be revoked and revocation time T, and a revocation list RL. It outputs an updated revocation list RL.

The correctness of RIBE-AR is defined as follows: For all MK, RL, and PP generated by **Setup**$(1^\lambda, N)$, SK_{ID} generated by **GenKey**(ID, MK, PP) for any ID, UK_T generated by **UpdateKey**(T, RL, MK, PP) for any T and RL such that $(ID, T_j) \notin RL$ for all $T_j \leq T$, CT generated by **Encrypt**(ID, T, M, PP) for ID, T, and M, it is required that

- **Decrypt**$(CT,$**DeriveKey**$(SK_{ID}, UK_T, PP), PP) = M$.

3.2 Security Model

The selective IND-CPA security model of RIBE-AR is an extension of the selective IND-CPA security model of IBE that allows the revocation of private keys [7, 38]. In this model, an attacker first submits the challenge identity ID^* and challenge time T^* and receives public parameters. Afterwards, the attacker can request private key, update key, decryption key, and revocation queries that satisfy some restrictions to prevent trivial attacks. In the challenge phase, the attacker submits challenge messages M_0^*, M_1^* and receives a challenge ciphertext for ID^* and T^* that encrypts M_0^* or M_1^*. The attacker can then make additional queries and succeed if he correctly guesses the challenge ciphertext message. The detailed definition of this security model is given as follows:

Definition 3.2 (Message Privacy, SE-IND-CPA)**.** The selective IND-CPA (SE-IND-CPA) security of RIBE-AR is defined in terms of the following experiment $\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}IND\text{-}CPA}(1^\lambda)$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ submits a challenge identity ID^* and challenge time T^*.

2. **Setup**: $\mathcal{C}$ generates a master key MK, a revocation list RL, a state ST, and public parameters PP by running $\mathbf{Setup}(1^\lambda, N)$. It initializes a current time period $T_c = 1$ and gives PP to $\mathcal{A}$.

3. **Phase 1**: $\mathcal{A}$ adaptively request a polynomial number of queries. $\mathcal{C}$ handles these queries as follows:

   - Private key query for an identity ID: It checks that if $ID = ID^*$, then $(ID^*, T) \in RL$ for some $T \leq T^*$. It generates SK_{ID} by running $\mathbf{GenKey}(ID, MK, PP)$ and gives SK_{ID} to $\mathcal{A}$.
   - Update key query: It generates UK_{T_c} by running $\mathbf{UpdateKey}(T_c, RL, MK, PP)$, sets $T_c = T_c + 1$, and gives UK_{T_c} to $\mathcal{A}$.
   - Decryption key query for an identity ID and time T: It checks that $(ID, T) \neq (ID^*, T^*)$ and $1 \leq T < T_c$. It generates $DK_{ID,T}$ by running $\mathbf{DeriveKey}(SK_{ID}, UK_T, PP)$ and gives $DK_{ID,T}$ to $\mathcal{A}$.
   - Revocation query for an identity ID: It updates RL by running $\mathbf{Revoke}(ID, T_c, RL)$.

4. **Challenge**: $\mathcal{A}$ submits two challenge messages M_0^*, M_1^* with equal length. $\mathcal{C}$ flips random $\mu \in \{0,1\}$. It obtains a challenge ciphertext CT^* by running $\mathbf{Encrypt}(ID_\mu^*, T^*, M_\mu^*, PP)$ and gives CT^* to $\mathcal{A}$.

5. **Phase 2**: $\mathcal{A}$ may continue to request additional queries. $\mathcal{B}$ handles these queries as the same as the phase 1.

6. **Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$, and wins the game if $\mu = \mu'$.

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{RIBE\text{-}AR,\mathcal{A}}^{SE\text{-}IND\text{-}CPA}(\lambda) = \left|\Pr[\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}IND\text{-}CPA}(1^\lambda) = 1] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the experiment. An RIBE-AR scheme is SE-IND-CPA secure if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

The selective ANO-CPA security of RIBE-AR is an extension of the selective ANO-CPA security model of AIBE to allow the revocation of private keys. In this model, an attacker first submits challenge identities ID_0^*, ID_1^* and challenge time T^* and receives public parameters. Afterwards, the attacker can request private key, update key, decryption key, and revocation queries that satisfies some restrictions to prevent trivial attacks. In the challenge phase, the attacker submits a challenge message M^* and receives a challenge ciphertext encrypted with ID_0^* or ID_1^* for T^* and M^*. The attacker can then make additional queries and succeed by correctly guessing the challenge identity in the challenge ciphertext. The detailed definition of this security model is given as follows:

Definition 3.3 (Identity Privacy, SE-ANO-CPA)**.** The selective ANO-CPA (SE-ANO-CPA) security of RIBE-AR is defined in terms of the following experiment $\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}ANO\text{-}PRV}(1^{\lambda})$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ submits two challenge identities ID_0^*, ID_1^* and challenge time T^*.
2. **Setup**: $\mathcal{C}$ generates a master key MK, a revocation list RL, a state ST, and public parameters PP by running **Setup**$(1^{\lambda}, N)$. It initializes a current time period $T_c = 1$ and gives PP to $\mathcal{A}$.
3. **Phase 1**: $\mathcal{A}$ adaptively request a polynomial number of queries. $\mathcal{C}$ handles these queries as follows:
   - Private key query for an identity ID: It checks that if $ID = ID_b^*$ for some $b \in \{0,1\}$, then $(ID_b^*, T) \in RL$ for some $T \leq T^*$. It generates SK_{ID} by running **GenKey**(ID, MK, PP) and gives SK_{ID} to $\mathcal{A}$.
   - Update key query: It generates UK_{T_c} by running **UpdateKey**(T_c, RL, MK, PP), sets $T_c = T_c + 1$, and gives UK_{T_c} to $\mathcal{A}$.
   - Decryption key query for an identity ID and time T: It checks that $(ID, T) \neq (ID_b^*, T^*)$ for all $b \in \{0,1\}$ and $1 \leq T < T_c$. It generates $DK_{ID,T}$ by running **DeriveKey**(SK_{ID}, UK_T, PP) and gives $DK_{ID,T}$ to $\mathcal{A}$.
   - Revocation query for an identity ID: It updates RL by running **Revoke**(ID, T_c, RL).
4. **Challenge**: $\mathcal{A}$ submits a challenge message M^*. $\mathcal{C}$ flips random $\mu \in \{0,1\}$. It obtains a challenge ciphertext CT^* by running **Encrypt**$(ID_{\mu}^*, T^*, M^*, PP)$ and gives CT^* to $\mathcal{A}$.
5. **Phase 2**: $\mathcal{A}$ may continue to request additional queries. $\mathcal{B}$ handles these queries as the same as the phase 1.
6. **Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$, and wins the game if $\mu = \mu'$.

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{RIBE\text{-}AR,\mathcal{A}}^{SE\text{-}ANO\text{-}CPA}(\lambda) = \left|\Pr[\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}ANO\text{-}PRV}(1^{\lambda}) = 1] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the experiment. An RIBE-AR scheme is SE-ANO-CPA secure if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

The selective REV-PRV security of RIBE-AR models that external users who only have revoked private keys cannot get revocation information from update keys. In this model, an attacker first submits challenge revocation sets R_0^*, R_1^* and challenge time T^* and receives public parameters. Afterwards, the attacker can request private key, update key, decryption key, and revocation queries that satisfy some restrictions. One restriction is that an attacker can only query private keys belonging to $R_0^* \cap R_1^*$. In the challenge phase, the attacker receives a challenge update key for R_0^* or R_1^*. Afterwards, the attacker can make additional queries and finally succeed if he can guess the revoked set of the challenge update key. The detailed definition of this security model is given as follows:

Definition 3.4 (Revocation Privacy, SE-REV-PRV)**.** The selective REV-PRV (SE-REV-PRV) security of RIBE-AR is defined in terms of the following experiment $\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}REV\text{-}PRV}(1^{\lambda})$ between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ submits two challenge revoked sets R_0^*, R_1^* and challenge time T^* such that $|R_0^*| = |R_1^*|$.
2. **Setup**: $\mathcal{C}$ generates a master key MK, a revocation list RL, a state ST, and public parameters PP by running **Setup**$(1^{\lambda}, N)$. It initializes a current time period $T_c = 1$ and gives PP to $\mathcal{A}$.

3. **Phase 1**: $\mathcal{A}$ adaptively request a polynomial number of queries. $\mathcal{C}$ handles these queries as follows:
   - Private key query for an identity ID: It checks that $ID \in R_0^* \cap R_1^*$. It generates SK_{ID} by running **GenKey**(ID, MK, PP) and gives SK_{ID} to $\mathcal{A}$.
   - Update key query: It checks that $T_c \neq T^*$. It generates UK_{T_c} by running **UpdateKey**(T_c, RL, MK, PP), sets $T_c = T_c + 1$, and gives UK_{T_c} to $\mathcal{A}$.
   - Decryption key query for an identity ID and time T: It checks that $1 \leq T < T_c$. It generates $DK_{ID,T}$ by running **DeriveKey**(SK_{ID}, UK_T, PP) and gives $DK_{ID,T}$ to $\mathcal{A}$.
   - Revocation query for an identity ID: It updates RL by running **Revoke**(ID, T_c, RL).
4. **Challenge**: $\mathcal{C}$ flips random $\mu \in \{0,1\}$ and proceeds as follows:
   (a) For each $ID \in R_\mu^*$, it updates RL by adding (ID, T^*) if $(ID, T) \notin RL$ for some $T < T^*$.
   (b) It obtains a challenge update key UK^* by running **UpdateKey**(T^*, RL, MK, PP). It sets $T_c = T^* + 1$ and gives UK^* to $\mathcal{A}$.
5. **Phase 2**: $\mathcal{A}$ may continue to request additional queries. $\mathcal{B}$ handles these queries as the same as the phase 1.
6. **Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$, and wins the game if $\mu = \mu'$.

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{RIBE\text{-}AR,\mathcal{A}}^{SE\text{-}REV\text{-}PRV}(\lambda) = \left|\Pr[\mathbf{EXP}_{\mathcal{A}}^{SE\text{-}REV\text{-}PRV}(1^\lambda) = 1] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the experiment. An RIBE-AR scheme is SE-REV-PRV secure if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

Remark 1. The selective REV-PRV security model we defined is a weak security model because it considers outsider attackers that only query private keys belonging to $R_0^* \cap R_1^*$ (private keys of revoked users). A stronger security model is to consider inside attackers that can additionally query private keys belonging to $(\mathcal{U} \setminus R_0^*) \cap (\mathcal{U} \setminus R_1^*)$ (private keys of non-revoked users) where $\mathcal{U}$ is the set of all users. However, our RIBE-AR scheme in this paper only satisfies this weak security model.

4 Construction from Bilinear Maps

In this section, we propose an anonymous RIBE-AR scheme that provides revocation privacy in bilinear groups.

4.1 Bilinear Groups

A bilinear group generator $\mathcal{G}$ takes as input a security parameter λ and outputs a tuple $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ where p is a random prime and $\mathbb{G}, \hat{\mathbb{G}}$, and $\mathbb{G}_T$ be three cyclic groups of prime order p. Let g and $\hat{g}$ be generators of $\mathbb{G}$ and $\hat{\mathbb{G}}$, respectively. The bilinear map $e : \mathbb{G} \times \hat{\mathbb{G}} \to \mathbb{G}_T$ has the following properties:

1. Bilinearity: $\forall u \in \mathbb{G}, \forall \hat{v} \in \hat{\mathbb{G}}$ and $\forall a, b \in \mathbb{Z}_p$, $e(u^a, \hat{v}^b) = e(u, \hat{v})^{ab}$.
2. Non-degeneracy: $\exists g \in \mathbb{G}, \hat{g} \in \hat{\mathbb{G}}$ such that $e(g, \hat{g})$ has order p in $\mathbb{G}_T$.

We say that $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ are asymmetric bilinear groups with no efficiently computable isomorphisms if the group operations in $\mathbb{G}, \hat{\mathbb{G}}$, and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable, but there are no efficiently computable isomorphisms between $\mathbb{G}$ and $\hat{\mathbb{G}}$.

4.2 Complexity Assumptions

In this section, we introduce complexity assumptions for the security proof of our RIBE scheme.

Assumption 1 (Decisional Bilinear Diffie-Hellman, DBDH [13])**.** Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be an asymmetric bilinear group generated by $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. The decisional bilinear Diffie-Hellman (DBDH) assumption is that if the challenge tuple

$$D = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^a, g^c, \hat{g}, \hat{g}^a, \hat{g}^b\big) \text{ and } Z$$

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = e(g, \hat{g})^{abc}$ from $Z = Z_1 = e(g, \hat{g})^d$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{DBDH}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b, c, d \in \mathbb{Z}_p$.

Assumption 2 (Decisional eXternal Diffie-Hellman, XDH)**.** Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be an asymmetric bilinear group generated by $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. The decisional XDH assumption in $\hat{\mathbb{G}}$ is that if the challenge tuple

$$D = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, \hat{g}^a, \hat{g}^b\big) \text{ and } Z$$

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = \hat{g}^{ab}$ from $Z = Z_1 = \hat{g}^c$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{XDH}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b, c \in \mathbb{Z}_p$.

Assumption 3 (Decisional P-Bilinear Diffie-Hellman, PBDH, [16])**.** Let $(p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e)$ be an asymmetric bilinear group generated by $\mathcal{G}(1^\lambda)$. Let $g, \hat{g}$ be random generators of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. The decisional PBDH assumption is that if the challenge tuple

$$D = \big((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, g^a, g^{ab}, g^c, \hat{g}, \hat{g}^a, \hat{g}^b\big) \text{ and } Z$$

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = g^{abc}$ from $Z = Z_1 = g^d$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{PBDH}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b, c, d \in \mathbb{Z}_p$.

4.3 AIBE and IBE Schemes

We describe AIBE and IBE schemes, which are the basis for the design of our RIBE-AR scheme. The underlying AIBE scheme we describe is a modified key encapsulation mechanism (KEM) version of the AHIBE scheme proposed by Ducas that supports private key re-randomization [16]. And the underlying IBE scheme we described is a modified KEM version of the IBE scheme of Boneh and Boyen [8]. In addition, we added RandKey and ChangeKey algorithms to the existing AIBE and IBE schemes to enable private key randomization and master-key part randomization in private keys. Using this modification, our RIBE-AR scheme can be simply described by using the AIBE and IBE schemes in a modular way.

The AIBE scheme for $\mathcal{I} = \mathbb{Z}_p$ from the AHIBE scheme of Ducas is described as follows:

AIBE.Setup(GDS): Let $GDS = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g})$ be the description of a bilinear group with generators $g \in \mathbb{G}, \hat{g} \in \hat{\mathbb{G}}$. It selects random exponents $u_1', h_1', w_1' \in \mathbb{Z}_p$ and sets $u_1 = g^{u_1'}, h_1 = g^{h_1'}, w = g^{w_1'}, \hat{u}_1 = \hat{g}^{u_1'}, \hat{h}_1 = \hat{g}^{h_1'}, \hat{w} = \hat{g}^{w_1'}$. It chooses a random exponent $\gamma \in \mathbb{Z}_p$ and outputs a master key $MK = \hat{g}^\gamma$, master parameters $MP = (\hat{u}_1, \hat{h}_1, \hat{w})$, and public parameters $PP = \big(GDS, u_1, h_1, w, \Lambda = e(g, \hat{g})^\gamma\big)$.

AIBE.GenKey(ID, MK, MP, PP)**:** It chooses random exponents $r_1, \ldots, r_6 \in \mathbb{Z}_p$ and outputs a private key $SK_{ID} = \big(K_0 = MK(\hat{u}_1^{ID}\hat{h}_1)^{r_1}\hat{w}^{r_2}, K_1 = \hat{g}^{-r_1}, K_2 = \hat{g}^{-r_2}\big)$ with a rand key $RK_{ID} = \big(L_{0,1} = (\hat{u}_1^{ID}\hat{h}_1)^{r_3}\hat{w}^{r_4}, L_{1,1} = \hat{g}^{-r_3}, L_{2,1} = \hat{g}^{-r_4}, L_{0,2} = (\hat{u}_1^{ID}\hat{h}_1)^{r_5}\hat{w}^{r_6}, L_{1,2} = \hat{g}^{-r_5}, L_{2,2} = \hat{g}^{-r_6}\big)$.

AIBE.RandKey(SK_{ID}, RK_{ID}, PP)**:** Let $SK_{ID} = (K_0', K_1', K_2')$ and $RK_{ID} = (L_{0,1}, L_{1,1}, L_{2,1}, L_{0,2}, L_{1,2}, L_{2,2})$. It chooses random exponents $r_1, r_2 \in \mathbb{Z}_p$ and outputs a new private key $SK_{ID} = \big(K_0 = K_0' \cdot L_{0,1}^{r_1} L_{0,2}^{r_2}, K_1 = K_1' \cdot L_{1,1}^{-r_1} L_{1,2}^{-r_2}, K_2 = K_2' \cdot L_{2,1}^{-r_1} L_{2,2}^{-r_2}\big)$.

AIBE.ChangeKey($SK_{ID}, \delta, RK_{ID}, PP$)**:** Let $SK_{ID} = (K_0', K_1', K_2')$. It sets $TK = (K_0 = K_0' \cdot \hat{g}^{\delta}, K_1 = K_1', K_2 = K_2')$. It outputs a new private key SK_{ID} by running **AIBE.RandKey**(TK, RK_{ID}, PP).

AIBE.Encaps(ID, t, PP)**:** It outputs a ciphertext header $CH = \big(C_0 = g^t, C_1 = (u_1^{ID}h_1)^t, C_2 = w^t\big)$ and a session key $EK = \Lambda^t$.

AIBE.Decaps(CH, SK_{ID}, PP)**:** Let $CH = (C_0, C_1, C_2)$ and $SK_{ID} = (K_0, K_1, K_2)$. It outputs a session key EK by calculating $EK = \prod_{i=0}^{2} e(C_i, K_i)$.

The IBE scheme for $\mathcal{I} = \mathbb{Z}_p$ from the IBE scheme of Boneh and Boyen is described as follows:

IBE.Setup(GDS)**:** Let $GDS = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g})$ be the group description string of a bilinear group with a generator $g \in \mathbb{G}$. It selects random exponents $u_2', h_2' \in \mathbb{Z}_p$ and sets $u_2 = g^{u_2'}, h_2 = g^{h_2'}, \hat{u}_2 = \hat{g}^{u_2'}, \hat{h}_2 = \hat{g}^{h_2'}$. It chooses a random exponent $\beta \in \mathbb{Z}_p$ and outputs a master key $MK = \hat{g}^{\beta}$ and public parameters $PP = \big(GDS, u_2, h_2, \hat{u}_2, \hat{h}_2, \Lambda = e(g, \hat{g})^{\beta}\big)$.

IBE.GenKey(T, MK, PP)**:** It chooses a random exponent $r \in \mathbb{Z}_p$ and outputs a private key $SK_T = \big(U_0 = MK(\hat{u}_2^T\hat{h}_2)^r, U_1 = \hat{g}^{-r}\big)$.

IBE.RandKey(SK_T, PP)**:** Let $SK_T = (U_0', U_1')$. It chooses a random exponent $r \in \mathbb{Z}_p$ and outputs a randomized private key $SK_T = \big(U_0 = U_0' \cdot (\hat{u}_2^T\hat{h}_2)^r, U_1 = U_1' \cdot \hat{g}^{-r}\big)$.

IBE.ChangeKey(SK_T, δ, PP)**:** Let $SK_T = (U_0', U_1')$. It sets $TK = (U_0 = U_0' \cdot \hat{g}^{\delta}, U_1 = U_1')$. It outputs a new private key SK_T by running **IBE.RandKey**(TK, PP).

IBE.Encaps(T, t, PP)**:** Let t be a random exponent in $\mathbb{Z}_p$. It outputs a ciphertext header $CH_T = \big(C_0 = g^t, C_1 = (u_2^T h_2)^t\big)$ and a session key $EK = \Lambda^t$.

IBE.Decaps($CH_T, SK_{T'}, PP$)**:** Let $CH_T = (C_0, C_1)$ and $SK_{T'} = (U_0, U_1)$. If $T = T'$, then it outputs a session key EK by computing $EK = e(C_0, U_0) \cdot e(C_1, U_1)$. Otherwise, it outputs $\perp$.

4.4 RIBE-AR Construction

The basic idea of designing our RIBE-AR scheme is to follow the existing design method of combining two IBE schemes and a tree-based revocation system [7, 38]. For ciphertext anonymity, we use an AIBE scheme instead of the first IBE scheme. For revocation privacy, we encrypt each node update key in an RIBE-AR update key by using an SKE scheme. At this time, a symmetric key used for the encryption of the node update key is uniquely associated with each node of a binary tree, and an RIBE-AR private key has symmetric keys corresponding to the path nodes of the binary tree. In this case, if a common node associated with the path node in the private key and the cover nodes in the update key exists, it is possible to decrypt the node update key by using the common symmetric key associated with the common node. However, the algorithm for deriving a decryption key has the problem of being slow since it is needed to try all tree nodes included in the update key. To overcome this, we use an efficient method that can quickly find the matching node by providing hint information in an update key, which was devised for anonymous broadcast encryption [18, 29]. Additionally, we randomly mix all node update keys in the update key.

Let **PRF** be a pseudo-random function for $\mathcal{K} = \{0,1\}^{\lambda}$, $\mathcal{X} = \{0,1\}^{*}$, and $\mathcal{Y} = \mathbb{Z}_p^2 \times \{0,1\}^{\lambda}$. Let **Label** be a function that uniquely maps a leaf node v_i to a bit string in $\{0,1\}^{*}$. Our RIBE-AR scheme for $\mathcal{I} = \mathbb{Z}_p$, $\mathcal{T} = \mathbb{Z}_p$, and $\mathcal{M} \in \mathbb{G}_T$ is described as follows:

RIBE-AR.Setup$(1^{\lambda}, N)$**:** Let λ be a security parameter and N be the maximum number of users.

1. It generates asymmetric bilinear groups $\mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T$ of prime order p with two random generators $g, \hat{g}$ of $\mathbb{G}, \hat{\mathbb{G}}$ respectively. It sets $GDS = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g})$.
2. It obtains MK_{AIBE}, MP_{AIBE}, and PP_{AIBE} by running **AIBE.Setup**(GDS). It also obtains MK_{IBE} and PP_{IBE} by running **IBE.Setup**(GDS). It obtains $\mathcal{BT}$ by running **CS.Setup**(N) and selects a random PRF key z.
3. Finally, it chooses a random exponent $\alpha \in \mathbb{Z}_p$ and outputs a master key $MK = (MK_{AIBE}, MP_{AIBE}, MK_{IBE}, \alpha)$, an empty revocation list RL, a state $ST = (\mathcal{BT}, z)$, and public parameters $PP = \big(PP_{AIBE}, PP_{IBE}, \Omega = e(g, \hat{g})^{\alpha}\big)$.

RIBE-AR.GenKey(ID, MK, ST, PP)**:** Let $MK = (MK_{AIBE}, MP_{AIBE}, MK_{IBE}, \alpha)$ and $ST = (\mathcal{BT}, z)$.

1. It assigns ID to a random leaf node $v \in \mathcal{BT}$ and obtains a private set $PV = \{v_0, \ldots, v_n\}$ by running **CS.Assign**$(\mathcal{BT}, v)$.
2. For $0 \leq j \leq n$, it computes $(\gamma_j, \omega_j, \kappa_j) = \mathbf{PRF}(z, \mathbf{Label}(v_j))$ and proceeds as follows: It obtains $SK_{AIBE,j}$ and $RK_{AIBE,j}$ by running **AIBE.GenKey**$(ID, \hat{g}^{\gamma_j}, MP_{AIBE}, PP_{AIBE})$. It creates a node private key $NSK_j = \big(SK_{AIBE,j}, \omega_j, \kappa_j\big)$.
3. Finally, it sets $RK_{ID} = RK_{AIBE,0}$ and outputs a private key $SK_{ID} = \big(NSK_0, \ldots, NSK_n, RK_{ID}\big)$.

RIBE-AR.UpdateKey(T, RL, MK, ST, PP)**:** Let $MK = (MK_{AIBE}, MP_{AIBE}, MK_{AIBE}, \alpha)$ and $ST = (\mathcal{BT}, z)$.

1. It derives a revoked set R on time T from RL and obtains a cover set $CV = \{v_1, \ldots, v_{\ell}\}$ by running **CS.Cover**$(\mathcal{BT}, R)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It selects a random exponent $s \in \mathbb{Z}_p$ and sets $V = \hat{g}^{s}$.
2. For $1 \leq i \leq \ell$, it computes $(\gamma_i, \omega_i, \kappa_i) = \mathbf{PRF}(z, \mathbf{Label}(v_i))$ and proceeds as follows: It obtains $SK_{IBE,i}$ by running **IBE.GenKey**$(T, \hat{g}^{\alpha - \gamma_i}, PP_{IBE})$. It computes $CU_i = \mathbf{SKE.Encrypt}(\kappa_i, SK_{IBE,i})$. It creates a node update key $NUK_i = \big(CU_i, Y_i\big)$ by setting a hint value $Y_i = V^{\omega_i}$.
3. For $\ell + 1 \leq i \leq \ell_m$, it proceeds as follows: It sets a random $\tilde{SK}_{IBE,i} = (\tilde{U}_{i,0}, \tilde{U}_{i,1})$ by selecting random $\tilde{U}_{i,0}, \tilde{U}_{i,1} \in \hat{\mathbb{G}}$. It computes $CU_i = \mathbf{SKE.Encrypt}(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random $\tilde{\kappa}_i$. It creates a node update key $NUK_i = \big(CU_i, \tilde{Y}_i\big)$ by selecting a random $\tilde{Y}_i \in \hat{\mathbb{G}}$.
4. Finally, it selects a random permutation $\pi : \{1, \ldots, \ell_m\} \to \{1, \ldots, \ell_m\}$ and outputs an update key $UK_T = \big(NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V\big)$.

RIBE-AR.DeriveKey$(ID, T, SK_{ID}, UK_T, PP)$**:** Let $SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{ID})$ and $UK_T = (NUK_1, \ldots, NUK_{\ell_m}, V)$.

1. It retrieves ω_j from NSK_j for all $j \in \{0, \ldots, n\}$ and sets a list $(Y_0' = V^{\omega_0}, \ldots, Y_n' = V^{\omega_n})$. It also retrieves Y_i from NUK_i for all $i \in \{1, \ldots, \ell_m\}$ and sets a list $(Y_1, \ldots, Y_{\ell_m})$.

2. It finds two indexes $j \in \{0,\dots,n\}$ and $i \in \{1,\dots,\ell_m\}$ such that $Y'_j = Y_i$ in the lists, then it retrieves the corresponding $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$ and $NUK_i = (CU_i, Y_i)$. Next, it computes $SK_{IBE,i} = \textbf{SKE.Decrypt}(\kappa_j, CU_i)$.
3. It selects a random exponent $\delta \in \mathbb{Z}_p$. It obtains SK_{AIBE} by running $\textbf{AIBE.ChangeKey}(SK_{AIBE,j}, \delta, RK_{ID}, PP_{AIBE})$. It also obtains SK_{IBE} by running $\textbf{IBE.ChangeKey}(SK_{IBE,i}, -\delta, PP_{IBE})$. Finally, it outputs a decryption key $DK_{ID,T} = \big(SK_{AIBE}, SK_{IBE}\big)$.

RIBE-AR.Encrypt(ID,T,M,PP)**:** It chooses a random exponent $t \in \mathbb{Z}_p$. It obtains CH_{AIBE} and EK_{AIBE} by running $\textbf{AIBE.Encaps}(ID,t,PP_{AIBE})$. Next it obtains CH_{IBE} and EK_{IBE} by running $\textbf{IBE.Encaps}(T,t, PP_{IBE})$. It outputs a ciphertext $CT = \big(CH_{AIBE}, CH_{IBE}, C = \Omega^t \cdot M\big)$.

RIBE-AR.Decrypt$(CT, DK_{ID,T}, PP)$**:** Let $CT = (CH_{AIBE}, CH_{IBE}, C)$ and $DK_{ID,T} = (SK_{AIBE}, SK_{IBE})$. It derives EK_{AIBE} and EK_{IBE} by running $\textbf{AIBE.Decaps}(CH_{AIBE}, SK_{AIBE}, PP_{AIBE})$ and $\textbf{IBE.Decaps}(CH_{IBE}, SK_{IBE}, PP_{IBE})$ respectively. It outputs a decrypted message $M = C \cdot \big(EK_{AIBE} \cdot EK_{IBE}\big)^{-1}$.

RIBE-AR.Revoke(ID,T,RL)**:** If ID is not assigned in $\mathcal{BT}$, then it outputs $\perp$. Otherwise, it updates RL by adding (ID,T) to RL.

4.5 Correctness

To show the correctness of the above RIBE-AR scheme, we first show that a decryption key $DK_{ID,T}$ is correctly derived from a private key SK_{ID} and an update key UK_T. Let $SK_{ID} = (NSK_0,\dots,NSK_n,RK_{ID})$ for PV and $UK_T = (NUK_1,\dots,NUK_{\ell_m},V)$ for CV where $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$ and $NUK_i = (CU_i, Y_i = V^{\omega_i})$. If $ID \notin R$, then $PV \cap CV \neq \emptyset$. Thus, there exist a node $v_k \in PV$ and $v_k \in CV$ such that $V^{\omega_k} = Y_k$ where $\omega_k \in NSK_k = (SK_{AIBE,k}, \omega_k, \kappa_k)$ and $Y_k \in NUK_k = (CU_k, Y_k)$. By decrypting CU_k with a valid key κ_k, $SK_{IBE,k}$ is derived. From the **RIBE-AR.GenKey** and **RIBE-AR.UpdateKey** algorithms, the master key parts of $SK_{AIBE,k}$ and $SK_{IBE,k}$ are associated with γ_k and $\alpha - \gamma_k$ respectively. The master key part of SK_{AIBE} derived from **AIBE.ChangeKey** still associated with $\gamma_k + \delta$ and the master key part of SK_{IBE} derived from **IBE.ChangeKey** still associated with $\alpha - \gamma_k - \delta$. Thus the **RIBE-AR.DeriveKey** algorithm is correct since we have α if we add two master key parts of the decryption key.

Next, we show that a message is correctly decrypted by the decryption algorithm. The correctness of the **RIBE-AR.Decrypt** algorithm can be shown by the correctness of the **AIBE.Decaps** and **IBE.Decaps**. That is, we have $e(g,g)^{(\gamma_k+\delta)t}$ from the correctness of AIBE and $e(g,g)^{(\alpha-\gamma_k-\delta)t}$ from the correctness of IBE. Thus, the message M can be easily obtained by using these session keys.

4.6 Discussion

Construction from Lattices. It is possible to design an RIBE-AR scheme that provides revocation privacy based on lattices. A lattice-based RIBE-AR scheme was previously proposed [15,21], but revocation privacy was not provided. To provide revocation privacy, the similar method of our pairing based RIBE-AR scheme that uses symmetric key encryption to hide node update keys can be used. And to quickly search for tree nodes that match in a private key and an update key, a hint system based on the LWR assumption that does not include noise can be used. A detailed description of the lattice-based RIBE-AR scheme is provided in Appendix A.

5 Security Analysis

In this section, we prove the three security features that our RIBE-AR scheme must satisfy: message privacy, identity privacy, and revocation privacy.

5.1 AIBE and IBE Security

The underlying AIBE scheme in this paper is an AIBE scheme capable of private key re-randomization by modifying the AHIBE scheme of Ducas [16]. For reference, Ducas also described the AIBE scheme, but it is not suitable for the RIBE-AR scheme which requires decryption key derivation since this AIBE scheme does not support private key re-randomization. The AHIBE scheme of Ducas provides selective IND-CPA security under the DBDH assumption and selective ANO-CPA security under the PBDH assumption.

Theorem 5.1 ([16]). *The above AIBE scheme is selectively IND-CPA secure if the DBDH assumption holds.*

Theorem 5.2 ([16]). *The above AIBE scheme is selectively ANO-CPA secure if the PBDH assumption holds.*

The underlying IBE scheme in this paper is a KEM version of the IBE scheme of Boneh and Boyen [8], which provides selective IND-CPA security under the DBDH assumption.

Theorem 5.3 ([8]). *The above IBE scheme is selectively IND-CPA secure if the DBDH assumption holds.*

5.2 IND-CPA Security

The IND-CPA security proof of our RIBE-AR scheme is almost similar to the IND-CPA security proof of the existing RIBE scheme. In other words, the simulator of the security proof divides attackers into two types: one that do not query the private key corresponding to the challenge identity ID^* and another that queries the private key for ID^*. And then the simulator handles the simulation of private keys and update keys differently for each type of attackers. To simplify the simulation of the RIBE-AR security proof, we use simulators of the AIBE and IBE schemes as sub-simulators.

Theorem 5.4. *The above RIBE-AR scheme is SE-IND-CPA secure if the PRF scheme is secure and the DBDH assumption holds.*

Proof. Let ID^* be the challenge identity submitted by an adversary. To prove the SE-IND-CPA security of our RIBE-AR scheme, we classify the type of adversaries into two types.

Type-1. An adversary is Type-1 if it does not request a private key for ID^*.

Type-2. An adversary is Type-2 if it requests a private key for ID^*.

Suppose that an adversary is τ-type. The security proof for the τ-type adversary $\mathcal{A}$ consists of the sequence of hybrid games. We define the games as follows:

Game $\mathbf{G}_0$. This game is the original security game. That is, a simulator $\mathcal{B}$ obtains $(\gamma_i, \omega_i, \kappa_i)$ for a node v_i by running PRF.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is similar to the game $\mathbf{G}_0$ except that the PRF is replaced by a truly random function. That is, $\mathcal{B}$ selects fixed random $(\gamma_i, \omega_i, \kappa_i)$ for a node v_i if v_i is used for the generation of a private key (or an update key).

Game $\mathbf{G}_2$. This final game $\mathbf{G}_2$ is similar to the game $\mathbf{G}_1$ except the generation of the challenge ciphertext CT^*. Let $CT^* = (CH^*_{AIBE}, CH^*_{IBE}, C^*)$ be the challenge ciphertext. In this game $\mathbf{G}_2$, C^* is replaced by a random element in $\mathbb{G}_T$. Note that the advantage of $\mathcal{A}$ in this game is zero since the challenge ciphertext is not related to μ.

Let $\mathbf{Adv}^{G_i}_{\mathcal{A}}$ be the advantage of $\mathcal{A}$ in a game $\mathbf{G}_i$. Let E_τ be the event that the adversary behaves like τ-type. From the Lemmas 5.5 and 5.6, we obtain the following equation

$$\mathbf{Adv}^{G_0}_{\mathcal{A}} \leq \sum_{\tau=1}^{2} \Pr[E_\tau] \cdot \left|\mathbf{Adv}^{G_0}_{\mathcal{A}} - \mathbf{Adv}^{G_2}_{\mathcal{A}}\right| \leq 2\mathbf{Adv}^{PRF}_{\mathcal{B}}(\lambda) + 2\mathbf{Adv}^{DBDH}_{\mathcal{B}}(\lambda).$$

This completes our proof. □

Lemma 5.5. *If the PRF scheme is secure, then no PPT type-τ adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

Proof. The proof is relatively straightforward if a simulator that distinguishes whether an oracle is PRF or not simply generates the master key MK of RIBE-AR by himself. We omit the details of this proof. □

Lemma 5.6. *If the DBDH assumption holds, then no PPT type-τ adversary can distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that attacks the above RIBE-AR scheme with a non-negligible advantage. A meta-simulator $\mathcal{B}$ that solves the DBDH assumption using $\mathcal{A}$ is given: a challenge tuple $D = (g, g^a, g^b, g^c, \hat{g}, \hat{g}^a, \hat{g}^b)$ and Z where $Z = Z_0 = e(g, \hat{g})^{abc}$ or $Z = Z_1 = e(g, \hat{g})^d$. Let $\mathcal{B}_{AIBE}$ be the simulator for AIBE and $\mathcal{B}_{IBE}$ be the simulator for IBE. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits a challenge identity ID^* and challenge time T^*. $\mathcal{B}$ runs $\mathcal{B}_{AIBE}$ by giving D and Z, and runs $\mathcal{B}_{IBE}$ by giving D and Z.

Setup: $\mathcal{B}$ proceeds as follows:

1. It submits ID^* to $\mathcal{B}_{AIBE}$ and receives PP_{AIBE}. It also submits T^* to $\mathcal{B}_{IBE}$ and receives PP_{IBE}.
2. It obtains $\mathcal{BT}$ by running $\mathbf{CS.Setup}(N)$. It sets RL as an empty one and sets $ST = (\mathcal{BT})$. It fixes a random leaf node $v^* \in \mathcal{BT}$ that will be assigned to ID^*.
3. If $\tau = 1$, it sets $\mathbf{ChalPath} = \emptyset$. Otherwise $(\tau = 2)$, it sets $\mathbf{ChalPath} = \mathbf{Path}(v^*)$.
4. It publishes public parameters $PP = \big(PP_{AIBE}, PP_{IBE}, \Omega = e(g^a, \hat{g}^b)\big)$.

Phase 1: $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, and decryption key queries. If this is a private key query for an identity ID, then $\mathcal{B}$ proceeds as follows:

- **Case** $\tau = 1$: In this case, we have $ID \neq ID^*$ and $\mathbf{ChalPath} = \emptyset$.
  1. It queries an AIBE private key for $ID \neq ID^*$ to $\mathcal{B}_{AIBE}$ and receives $SK_{AIBE,ID}$ and $RK_{AIBE,ID}$.
  2. It assigns ID to a new leaf node $v \neq v^*$ and obtains $PV = \{v_0, \ldots, v_n\}$ by running $\mathbf{CS.Assign}(\mathcal{BT}, v)$.
  3. For $0 \leq j \leq n$, it retrieves $(\gamma_j, \omega_j, \kappa_j)$ of the node v_j and performs the steps: It obtains $SK_{AIBE,j}$ by running $\mathbf{AIBE.ChangeKey}(SK_{AIBE,ID}, -\gamma_j, RK_{AIBE,ID}, PP_{AIBE})$. It sets $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$.
  4. It creates $SK_{ID} = \big(NSK_0, \ldots, NSK_n, RK_{AIBE,ID}\big)$.

- **Case** $\tau = 2 \wedge ID \neq ID^*$: In this case, we have $PV \cap$ **ChalPath** $\neq \emptyset$.

  1. It queries an AIBE private key for $ID \neq ID^*$ to $\mathcal{B}_{AIBE}$ and receives $SK_{AIBE,ID}$ and $RK_{AIBE,ID}$.
  2. It assigns ID to a new leaf node $v \neq v^*$ and obtains $PV = \{v_0, \dots, v_n\}$ by running **CS.Assign**$(\mathcal{BT}, v)$.
  3. For $0 \leq j \leq n$, it retrieves $(\gamma_j, \omega_j, \kappa_j)$ of the node v_j and performs the steps:
     (a) If $v_j \in$ **ChalPath**, then it obtains $SK_{AIBE,j}$ and $RK_{AIBE,j}$ by running **AIBE.GenKey**$(ID, \hat{g}^{\gamma_j}, MP_{AIBE}, PP_{AIBE})$.
     (b) Otherwise ($v_j \notin$ **ChalPath**), it obtains $SK_{AIBE,j}$ by running **AIBE.ChangeKey**$(SK_{AIBE,ID}, -\gamma_j, RK_{AIBE,ID}, PP_{AIBE})$.
     (c) It sets $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$.
  4. It creates $SK_{ID} = \big(NSK_0, \dots, NSK_n, RK_{AIBE,ID}\big)$.

- **Case** $\tau = 2 \wedge ID = ID^*$: In this case, we have $PV =$ **ChalPath**.

  1. It retrieves the leaf node v^* and obtains $PV = \{v_0, \dots, v_n\}$ by running **CS.Assign**$(\mathcal{BT}, v^*)$.
  2. For $0 \leq j \leq n$, it retrieves $(\gamma_j, \omega_j, \kappa_j)$ of the node v_j and performs the steps: It obtains $SK_{AIBE,j}$ and $RK_{AIBE,j}$ by running **AIBE.GenKey**$(ID, \hat{g}^{\gamma_j}, MP_{AIBE}, PP_{AIBE})$. It sets $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$.
  3. It creates $SK_{ID} = \big(NSK_0, \dots, NSK_n, RK_{AIBE,0}\big)$.

If this is an update key query for time T, then $\mathcal{B}$ proceeds as follows:

- **Case** $\tau = 1$: In this case, we have **ChalPath** $= \emptyset$.

  1. It defines a revoked set R on time T from RL and obtains $CV = \{v_1, \dots, v_\ell\}$ by running **CS.Cover** $(\mathcal{BT}, R)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It sets $V = \hat{g}^s$ by selecting a random $s \in \mathbb{Z}_p$.
  2. For $1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and performs the steps: It obtains $SK_{IBE,i}$ by running **IBE.GenKey**$(T, \hat{g}^{\gamma_i}, PP_{IBE})$. It computes $CU_i =$ **SKE.Encrypt**$(\kappa_i, SK_{IBE,i})$ and sets $NUK_i = (CU_i, Y_i = V^{\omega_i})$.
  3. For $\ell + 1 \leq i \leq \ell_m$, it performs the steps: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random elements and computes $CU_i =$ **SKE.Encrypt**$(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{\kappa}_i$. It selects a random $\tilde{Y}_i$ and creates $NUK_i = (CU_i, \tilde{Y}_i)$.
  4. It creates $UK_T = \big(NUK_{\pi(1)}, \dots, NUK_{\pi(\ell_m)}, V\big)$ where π is a random permutation.

- **Case** $\tau = 2$ and $T \neq T^*$: In this case, we have $CV \cap$ **ChalPath** $\neq \emptyset$.

  1. It queries an IBE private key for $T \neq T^*$ to $\mathcal{B}_{IBE}$ and receives $SK_{IBE,T}$.
  2. It defines a revoked set R on time T from RL obtains $CV = \{v_1, \dots, v_\ell\}$ by running **CS.Cover** $(\mathcal{BT}, R)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It sets $V = \hat{g}^s$ by selecting a random $s \in \mathbb{Z}_p$.
  3. For $1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and performs the steps:
     (a) If $v_i \in$ **ChalPath**, then it obtains $SK_{IBE,i}$ by running **IBE.ChangeKey**$(SK_{IBE,T}, \hat{g}^{-\gamma_i}, PP_{IBE})$.
     (b) Otherwise ($v_i \notin$ **ChalPath**), it obtains $SK_{IBE,i}$ by running **IBE.GenKey**$(T, \hat{g}^{\gamma_i}, PP_{IBE})$.
     (c) It computes $CU_i =$ **SKE.Encrypt**$(\kappa_i, SK_{IBE,i})$ and sets $NUK_i = (CU_i, Y_i = V^{\omega_i})$.

4. For $\ell+1 \leq i \leq \ell_m$, it performs the steps: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random elements and computes $CU_i = \textbf{SKE.Encrypt}(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{\kappa}_i$. It selects a random $\tilde{Y}_i$ and creates $NUK_i = (CU_i, \tilde{Y}_i)$.
5. It creates $UK_T = \big(NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V\big)$ where π is a random permutation.

- **Case** $\tau = 2$ and $T = T^*$: In this case, we have $CV \cap \textbf{ChalPath} = \emptyset$ since ID^* is revoked on time T^*.

  1. It defines a revoked set R^* on time T^* from RL and obtains $CV = \{v_1, \ldots, v_\ell\}$ by running $\textbf{CS.Cover}(\mathcal{BT}, R^*)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It sets $V = \hat{g}^s$ by selecting a random $s \in \mathbb{Z}_p$.
  2. For $1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and performs the steps: It obtains $SK_{IBE,i}$ by running $\textbf{IBE.GenKey}(T, \hat{g}^{\gamma_i}, PP_{IBE})$. It computes $CU_i = \textbf{SKE.Encrypt}(\kappa_i, SK_{IBE,i})$ and sets $NUK_i = (CU_i, Y_i = V^{\omega_i})$.
  3. For $\ell+1 \leq i \leq \ell_m$, it performs the steps: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random elements and computes $CU_i = \textbf{SKE.Encrypt}(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{\kappa}_i$. It selects a random $\tilde{Y}_i$ and creates $NUK_i = (CU_i, \tilde{Y}_i)$.
  4. It creates $UK_T = \big(NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V\big)$ where π is a random permutation.

If this is a decryption key query for an identity ID and time T, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID \neq ID^*$: It queries an AIBE private key for $ID \neq ID^*$ to $\mathcal{B}_{AIBE}$ and receives $SK_{AIBE,ID}$ and $RK_{AIBE,ID}$. It selects a random exponent $\delta \in \mathbb{Z}_p$. It obtains SK_{AIBE} by running **AIBE.ChangeKey** $(SK_{AIBE,ID}, -\delta, RK_{AIBE,ID}, PP_{AIBE})$. It obtains SK_{IBE} by running $\textbf{IBE.GenKey}(T, \hat{g}^{\delta}, PP_{IBE})$. It creates $DK_{ID,T} = (SK_{AIBE}, SK_{IBE})$.

- **Case** $ID = ID^*$ and $T \neq T^*$: It queries an IBE private key for $T \neq T^*$ to $\mathcal{B}_{IBE}$ and receives $SK_{IBE,T}$. It selects a random exponent $\delta \in \mathbb{Z}_p$. It obtains SK_{AIBE} and RK_{AIBE} by running $\textbf{AIBE.GenKey}(ID, \hat{g}^{\delta}, MP_{AIBE}, PP_{AIBE})$. It obtains SK_{IBE} by running $\textbf{IBE.ChangeKey}(SK_{IBE,T}, -\delta, PP_{IBE})$. It creates $DK_{ID,T} = (SK_{AIBE}, SK_{IBE})$.

If this is a revocation query for an identity ID and time T, then $\mathcal{B}$ updates RL by running $\textbf{RIBE-AR.Revoke}(ID, T, RL, ST)$.

Challenge: $\mathcal{A}$ submits two challenge messages M_0^*, M_1^*. $\mathcal{B}$ flips a random bit $\mu \in \{0,1\}$ and proceeds as follows:

1. It submits M_0^*, M_1^* to $\mathcal{B}_{AIBE}$ and receives a challenge CH_{AIBE}^* and EK_{AIBE}^*. It also submits M_0^*, M_1^* to $\mathcal{B}_{IBE}$ and receives a challenge CH_{IBE}^* and EK_{IBE}^*.

2. It creates $CT^* = \big(CH_{AIBE}^*, CH_{IBE}^*, Z \cdot M_\mu^*\big)$ and gives it to $\mathcal{A}$.

Phase 2: Same as Phase 1.

Guess: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ outputs 0 if $\mu = \mu'$ or 1 otherwise. This completes our proof. □

5.3 ANO-CPA Security

In order to argue the selective ANO-CPA security proof of our RIBE-AR scheme, we first classify attackers into four types. And we use the simulators of AIBE and IBE schemes as sub-simulators to configure an RIBE-AR simulator to easily handle private keys and update keys for individual types of attackers. Then, we play hybrid games that change the elements of an AIBE ciphertext header into random elements, showing that the challenge identity ID_μ^* is not revealed in the ciphertext.

Theorem 5.7. *The above RIBE-AR scheme is SE-ANO-CPA secure if the PRF scheme is secure and the PBDH assumption holds.*

Proof. Let ID_0^*, ID_1^* be the challenge identities submitted by an adversary. To prove the SE-ANO-CPA security of our RIBE-AR scheme, we classify the type of adversaries into four types.

Type-1. An adversary is Type-1 if it does not request a private key for ID_0^* and ID_1^*.

Type-2. An adversary is Type-2 if it does not request a private key for ID_0^*, but it requests a private key for ID_1^*.

Type-3. An adversary is Type-3 if it requests a private key for ID_0^*, but it does not request a private key for ID_1^*.

Type-4. An adversary is Type-4 if it requests private keys for ID_0^* and ID_1^*.

Suppose that an adversary is type-τ. The security proof for the type-τ adversary $\mathcal{A}$ consists of the sequence of hybrid games. We define the games as follows:

Game $\mathbf{G}_0$. This game is the original security game. That is, a simulator $\mathcal{B}$ obtains $(\gamma_i, \omega_i, \kappa_i)$ for a node v_i by running PRF.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is similar to the game $\mathbf{G}_0$ except that the PRF is replaced by a truly random function. That is, $\mathcal{B}$ selects fixed random $(\gamma_i, \omega_i, \kappa_i)$ for a node v_i if v_i is used for the generation of a private key (or an update key).

Game $\mathbf{G}_2$. This game $\mathbf{G}_2$ is similar to the game $\mathbf{G}_1$ except the generation of the challenge ciphertext CT^*. Let $CT^* = (CH_{AIBE}^*, CH_{IBE}^*, C^*)$ be the challenge ciphertext. In this game $\mathbf{G}_2$, C^* is replaced by a random element in $\mathbb{G}_T$.

Game $\mathbf{G}_3$. This final game $\mathbf{G}_3$ is similar to the game $\mathbf{G}_2$ except the generation of the challenge AIBE ciphertext header CH_{AIBE}^*. Let $CH_{AIBE}^* = (C_0^*, C_1^*, C_2^*)$ be the challenge AIBE ciphertext header. In this game, C_1^* and C_2^* are replaced by random elements in $\mathbb{G}$. Note that the advantage of $\mathcal{A}$ in this game is zero since the challenge ciphertext is not related to μ.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_i}$ be the advantage of $\mathcal{A}$ in a game $\mathbf{G}_i$. Let E_τ be the event that the adversary behaves like type-τ. From the Lemmas 5.8, 5.9, and 5.10, we obtain the following equation

$$\mathbf{Adv}_{\mathcal{A}}^{G_0} \le \sum_{\tau=1}^{4} \Pr[E_\tau] \cdot \left| \mathbf{Adv}_{\mathcal{A}_\tau}^{G_0} - \mathbf{Adv}_{\mathcal{A}_\tau}^{G_2} \right| \le 4\mathbf{Adv}_{\mathcal{B}}^{PRF}(\lambda) + 4\mathbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda) + 4\mathbf{Adv}_{\mathcal{B}}^{A3DH}(\lambda).$$

This completes our proof. □

Lemma 5.8. *If the PRF scheme is secure, then no PPT type-τ adversary can distinguish between $\boldsymbol{G}_0$ and $\boldsymbol{G}_1$ with a non-negligible advantage.*

We omit the proof of this lemma since it is the same as Lemma 5.5.

Lemma 5.9. *If the DBDH assumption holds, then no PPT type-τ adversary can distinguish between $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ with a non-negligible advantage.*

We omit the proof of this lemma since it is the same as Lemma 5.6.

Lemma 5.10. *If the PBDH assumption holds, then no PPT type-τ adversary can distinguish between $\boldsymbol{G}_2$ and $\boldsymbol{G}_3$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that attacks the above RIBE-AR scheme with a non-negligible advantage. A meta-simulator $\mathcal{B}$ that solves the A3DH assumption using $\mathcal{A}$ is given: a challenge tuple $D = (g, g^a, g^{ab}, g^c, \hat{g}, \hat{g}^a, \hat{g}^b)$ and Z where $Z = Z_0 = e(g, \hat{g})^{abc}$ or $Z = Z_1 = e(g, \hat{g})^d$. Note that a challenge tuple $D_{DBDH} = (g, g^a, g^c, \hat{g}, \hat{g}^a, \hat{g}^b)$ for the DBDH assumption can be derived from the given D. Let $\mathcal{B}_{AIBE}$ be the simulator for AIBE and $\mathcal{B}_{IBE}$ be the simulator for IBE. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits challenge identities ID_0^*, ID_1^* and challenge time T^*. $\mathcal{B}$ runs $\mathcal{B}_{AIBE}$ by giving D_{PBDH} and Z, and runs $\mathcal{B}_{IBE}$ by giving D_{DBDH} and Z.

Setup: $\mathcal{B}$ proceeds as follows:

1. It submits ID_0^*, ID_1^* to $\mathcal{B}_{AIBE}$ and receives PP_{AIBE}. It also submits T^* to $\mathcal{B}_{IBE}$ and receives PP_{IBE}.
2. It obtains $\mathcal{BT}$ by running **CS.Setup**(N). It initializes UL as an empty one. It sets RL as an empty one and sets $ST = (\mathcal{BT}, UL)$. It fixes random leaf nodes $v_0^*, v_1^* \in \mathcal{BT}$ that will be assigned to ID_0^*, ID_1^*, respectively.
3. If $\tau = 1$, it sets **ChalPath** $= \emptyset$. If $\tau = 2$, it sets **ChalPath** $=$ **Path**(v_1^*). If $\tau = 3$, it sets **ChalPath** $=$ **Path**(v_0^*). If $\tau = 4$, it sets **ChalPath** $=$ **Path**$(v_0^*) \cup$ **Path**(v_1^*).
4. It publishes public parameters $PP = \big(PP_{AIBE}, PP_{IBE}, \Omega = e(g^a, \hat{g}^b)\big)$.

Phase 1: $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, and decryption key queries. If this is a private key query for an identity ID, then $\mathcal{B}$ proceeds as follows:

- **Case** $\tau = 1$: In this case, we have $ID \neq ID_b^*$ for any $b \in \{0,1\}$ and **ChalPath** $= \emptyset$.
  1. It queries an AIBE private key for $ID \neq ID_b^*$ to $\mathcal{B}_{AIBE}$ and receives $SK_{AIBE,ID}$ and $RK_{AIBE,ID}$.
  2. It assigns ID to a new leaf node $v \neq v^*$ and obtains $PV = \{v_0, \ldots, v_n\}$ by running **CS.Assign**$(\mathcal{BT}, v)$.
  3. For $0 \leq j \leq n$, it retrieves $(\gamma_j, \omega_j, \kappa_j)$ of the node v_j and performs the steps: It obtains $SK_{AIBE,j}$ by running **AIBE.ChangeKey**$(SK_{AIBE,ID}, -\gamma_j, RK_{AIBE,ID}, PP_{AIBE})$. It sets $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$.
  4. It creates $SK_{ID} = \big(NSK_0, \ldots, NSK_n, RK_{AIBE,ID}\big)$.
- **Case** $(\tau = 2 \wedge ID \neq ID_1^*) \vee (\tau = 3 \wedge ID \neq ID_0^*) \vee (\tau = 4 \wedge ID \neq ID_0^*)$: In this case, we have $ID \neq ID_b^*$ for any $b \in \{0,1\}$ and $PV \cap$ **ChalPath** $\neq \emptyset$.

1. It queries an AIBE private key for $ID \neq ID_b^*$ to $\mathcal{B}_{AIBE}$ and receives $SK_{AIBE,ID}$ and $RK_{AIBE,ID}$.
2. It assigns ID to a new leaf node $v \neq v_b^*$ and obtains $PV = \{v_0, \ldots, v_n\}$ by running **CS.Assign**$(\mathcal{BT}, v)$.
3. For $0 \leq j \leq n$, it retrieves $(\gamma_j, \omega_j, \kappa_j)$ of the node v_j and performs the steps:
   (a) If $v_j \in$ **ChalPath**, then it obtains $SK_{AIBE,j}$ by running **AIBE.GenKey**$(ID, \hat{g}^{\gamma_j}, MP_{AIBE}, PP_{AIBE})$.
   (b) Otherwise ($v_j \notin$ **ChalPath**), it obtains $SK_{AIBE,j}$ by running **AIBE.ChangeKey**$(SK_{AIBE,ID}, -\gamma_j, RK_{AIBE,ID}, PP_{AIBE})$.
   (c) It sets $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$.
4. It creates $SK_{ID} = \big(NSK_0, \ldots, NSK_n, RK_{AIBE,ID}\big)$.

- **Case** $(\tau = 2 \wedge ID = ID_1^*)$ or $(\tau = 3 \wedge ID = ID_0^*)$ or $(\tau = 4 \wedge (ID = ID_0^* \vee ID = ID_1^*))$: In this case, we have $PV \subseteq$ **ChalPath**.

1. It retrieves the leaf node v_b^* of ID and obtains $PV = \{v_0, \ldots, v_n\}$ by running **CS.Assign**$(\mathcal{BT}, v_b^*)$.
2. For $0 \leq j \leq n$, it retrieves $(\gamma_j, \omega_j, \kappa_j)$ of the node v_j and performs the steps: It obtains $SK_{AIBE,j}$ and $RK_{AIBE,j}$ by running **AIBE.GenKey**$(ID, \hat{g}^{\gamma_j}, MP_{AIBE}, PP_{AIBE})$. It sets $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$.
3. It creates $SK_{ID} = \big(NSK_0, \ldots, NSK_n, RK_{AIBE,0}\big)$.

If this is an update key query for time T, then $\mathcal{B}$ proceeds as follows:

- **Case** $\tau = 1$: In this case, we have $CV \cap$ **ChalPath** $= \emptyset$ since **ChalPath** $= \emptyset$.

1. It defines a revoked set R on time T from RL and obtains $CV = \{v_1, \ldots, v_\ell\}$ by running **CS.Cover** $(\mathcal{BT}, R)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It sets $V = \hat{g}^s$ by selecting a random $s \in \mathbb{Z}_p$.
2. For $1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and performs the steps: It obtains $SK_{IBE,i}$ by running **IBE.GenKey**$(T, \hat{g}^{\gamma_i}, PP_{IBE})$. It computes $CU_i =$ **SKE.Encrypt**$(\kappa_i, SK_{IBE,i})$ and sets $NUK_i = (CU_i, Y_i = V^{\omega_i})$.
3. For $\ell + 1 \leq i \leq \ell_m$, it performs the steps: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random elements and computes $CU_i =$ **SKE.Encrypt**$(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{\kappa}_i$. It selects a random $\tilde{Y}_i$ and creates $NUK_i = (CU_i, \tilde{Y}_i)$.
4. It creates $UK_T = \big(NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V\big)$ where π is a random permutation.

- **Case** $(\tau = 2 \wedge T \neq T^*) \vee (\tau = 3 \wedge T \neq T^*) \vee (\tau = 4 \wedge T \neq T^*)$: In this case, we have $CV \cap$ **ChalPath** $\neq \emptyset$ and $CV \not\subseteq$ **ChalPath**.

1. It queries an IBE private key for $T \neq T^*$ to $\mathcal{B}_{IBE}$ and receives $SK_{IBE,T}$.
2. It defines a revoked set R on time T from RL and obtains $CV = \{v_1, \ldots, v_\ell\}$ by running **CS.Cover** $(\mathcal{BT}, R)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It sets $V = \hat{g}^s$ by selecting a random $s \in \mathbb{Z}_p$.
3. For $1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and performs the steps:
   (a) If $v_i \in$ **ChalPath**, then it obtains $SK_{IBE,i}$ by running **IBE.ChangeKey**$(SK_{IBE,T}, \hat{g}^{-\gamma_i}, PP_{IBE})$.
   (b) Otherwise ($v_i \notin$ **ChalPath**), it obtains $SK_{IBE,i}$ by running **IBE.GenKey**$(T, \hat{g}^{\gamma_i}, PP_{IBE})$.
   (c) It computes $CU_i =$ **SKE.Encrypt**$(\kappa_i, SK_{IBE,i})$ and sets $NUK_i = (CU_i, Y_i = V^{\omega_i})$.

4. For $\ell+1 \leq i \leq \ell_m$, it performs the steps: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random elements and computes $CU_i = \textbf{SKE.Encrypt}(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{\kappa}_i$. It selects a random $\tilde{Y}_i$ and creates $NUK_i = (CU_i, \tilde{Y}_i)$.
5. It creates $UK_T = \big(NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V\big)$ where π is a random permutation.

- **Case** $(\tau = 2 \wedge T = T^*) \vee (\tau = 3 \wedge T = T^*) \vee (\tau = 4 \wedge T = T^*)$: In this case, we have $CV \cap \textbf{ChalPath} = \emptyset$ since ID_b^* is revoked on time T^*.

  1. It defines a revoked set R^* on time T^* from RL and obtains $CV = \{v_1, \ldots, v_\ell\}$ by running $\textbf{CS.Cover}(\mathcal{BT}, R^*)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It sets $V = \hat{g}^s$ by selecting a random $s \in \mathbb{Z}_p$.
  2. For $1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and performs the steps: It obtains $SK_{IBE,i}$ by running $\textbf{IBE.GenKey}(T, \hat{g}^{\gamma_i}, PP_{IBE})$. It computes $CU_i = \textbf{SKE.Encrypt}(\kappa_i, SK_{IBE,i})$ and sets $NUK_i = (CU_i, Y_i = V^{\omega_i})$.
  3. For $\ell+1 \leq i \leq \ell_m$, it performs the steps: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random elements and computes $CU_i = \textbf{SKE.Encrypt}(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{\kappa}_i$. It selects a random $\tilde{Y}_i$ and creates $NUK_i = (CU_i, \tilde{Y}_i)$.
  4. It creates $UK_T = \big(NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V\big)$ where π is a random permutation.

If this is a decryption key query for an identity ID and time T, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID \neq ID^*$: It queries an AIBE private key for $ID \neq ID^*$ to $\mathcal{B}_{AIBE}$ and receives $SK_{AIBE,ID}$ and $RK_{AIBE,ID}$. It selects a random exponent $\delta \in \mathbb{Z}_p$. It obtains SK_{AIBE} by running **AIBE.ChangeKey** $(SK_{AIBE,ID}, -\delta, RK_{AIBE,ID}, PP_{AIBE})$. It obtains SK_{IBE} by running $\textbf{IBE.GenKey}(T, \hat{g}^{\delta}, PP_{IBE})$. It creates $DK_{ID,T} = (SK_{AIBE}, SK_{IBE})$.

- **Case** $ID = ID^*$ and $T \neq T^*$: It queries an IBE private key for $T \neq T^*$ to $\mathcal{B}_{IBE}$ and receives $SK_{IBE,T}$. It selects a random exponent $\delta \in \mathbb{Z}_p$. It obtains SK_{AIBE} and RK_{AIBE} by running $\textbf{AIBE.GenKey}(ID, \hat{g}^{\delta}, MP_{AIBE}, PP_{AIBE})$. It obtains SK_{IBE} by running $\textbf{IBE.ChangeKey}(SK_{IBE,T}, -\delta, PP_{IBE})$. It creates $DK_{ID,T} = (SK_{AIBE}, SK_{IBE})$.

If this is a revocation query for an identity ID and time T, then $\mathcal{B}$ updates RL by running $\textbf{RIBE-AR.Revoke}(ID, T, RL, ST)$.

Challenge: $\mathcal{A}$ submits a challenge message M^*. $\mathcal{B}$ proceeds as follows:

1. It submits M^* to $\mathcal{B}_{AIBE}$ and receives a challenge CH^*_{AIBE} and EK^*_{AIBE}. It also submits M^* to $\mathcal{B}_{IBE}$ and receives a challenge CH^*_{IBE} and EK^*_{IBE}.
2. It creates $CT^* = \big(CH^*_{AIBE}, CH^*_{IBE}, e(g, \hat{g})^d \cdot M^*\big)$ by selecting a random d and gives it to $\mathcal{A}$.

Phase 2: Same as Phase 1.

Guess: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ also outputs μ'. This completes our proof. □

5.4 REV-PRV Security

In order to prove the selective REV-PRV security of our RIBE-AR scheme, we construct hybrid games that change all challenge node update keys included in the challenge update key UK^* to random elements one by one. In this way, if all challenge node update keys are changed to random elements, an attacker will not be able to distinguish the challenge update key because the challenge revoked set R^*_μ is not exposed. The reason why all elements of the challenge update key can be converted into random elements is because the security model does not allow querying a private key that can be derived to a correct decryption key in combination with the challenge update key.

Theorem 5.11. *The above RIBE-AR scheme is SE-REV-PRV secure if the PRF is secure, the SKE scheme is IND-CPA and KEY-PRV secure, and the XDH assumption holds.*

Proof. The security proof consists of a sequence of hybrid games $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2$. The first game will be the original security game and the last one will be a game in which an adversary has no advantage. We define the games as follows:

Game $\mathbf{G}_0$. This game is the original security game. That is, a simulator $\mathcal{B}$ obtains $(\gamma_i, \omega_i, \kappa_i)$ for a node v_i by running PRF.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is similar to the game $\mathbf{G}_0$ except that the PRF is replaced by a truly random function. That is, $\mathcal{B}$ selects fixed random $(\gamma_i, \omega_i, \kappa_i)$ for a node v_i if v_i is used for the generation of a private key (or an update key).

Game $\mathbf{G}_2$. This final game $\mathbf{G}_2$ is similar to the game $\mathbf{G}_1$ except that generation of the challenge update key UK^*. Let $UK^* = (NUK^*_1, \dots, NUK^*_{\ell_m}, V^*)$ be the challenge update key where $NUK^*_i = (CU_i, Y_i)$ is the challenge node update key. In this game, each NUK_i is replaced by random. That is, CU_i is replaced by SKE encryption of a random message with a random key and Y_i is replaced by a random element in $\hat{\mathbb{G}}$. Note that the advantage of $\mathcal{A}$ in this game is zero since the challenge update key is not related to μ.

To argue that the adversary cannot distinguish $\mathbf{G}_1$ from $\mathbf{G}_2$, we also define a sequence of hybrid games $\mathbf{G}_{1,0} = \mathbf{G}_0, \dots, \mathbf{G}_{1,k}, \dots, \mathbf{G}_{1,\ell} = \mathbf{G}_2$ which are defined as follows:

Game $\mathbf{G}_{1,0}$. This game is equal to the game $\mathbf{G}_1$. That is, all challenge node update keys in the challenge update key are generated normally.

Game $\mathbf{G}_{1,k}$. In this game $\mathbf{G}_{1,k}$, each challenge node update key NUK^*_i for $1 \le i \le k$ is generated randomly, but each challenge node update key NUK^*_i for $k+1 \le i$ is generated normally.

Game $\mathbf{G}_{1,\ell}$. This game $\mathbf{G}_{1,\ell}$ is equal to the game $\mathbf{G}_2$. That is, all challenge node update keys in the challenge update key are generated randomly.

To argue the indistinguishability of $\mathbf{G}_{1,k-1}$ and $\mathbf{G}_{1,k}$, we additionally define a sequence of hybrid games $\mathbf{H}_{k,0}, \mathbf{H}_{k,1}, \mathbf{H}_{k,2}, \mathbf{H}_{k,3}$. Let $NUK^*_k = (CU_k, Y_k)$ be the challenge node update key of the node $v_k \in CV^*$. Through these hybrid games, we change the generation of this node update key. We define the games as follows:

Game $\mathbf{H}_{k,0}$. This game is equal to the game $\mathbf{G}_{1,k-1}$. That is, CU_k and Y_k are generated normally.

Game $\mathbf{H}_{k,1}$. In this game $\mathbf{H}_{k,1}$, CU_k is generated by encrypting random elements with a valid key, but Y_k is generated normally. That is, $CU_k = \textbf{SKE.Encrypt}(\kappa_k, (\tilde{U}_{k,0}, \tilde{U}_{k,1}))$ by selecting random $\tilde{U}_{k,0}, \tilde{U}_{k,1}$.

Game $\mathbf{H}_{k,2}$. In this game $\mathbf{H}_{k,2}$, CU_k is generated by encrypting random elements with a random key, but Y_k is generated normally. That is, $CU_k = \textbf{SKE.Encrypt}(\tilde{\kappa}_k, (\tilde{U}_{k,0}, \tilde{U}_{k,1}))$ by selecting random $\tilde{U}_{k,0}, \tilde{U}_{k,1}$ and a random key $\tilde{\kappa}_k$.

Game $\mathbf{H}_{k,3}$. This game $\mathbf{H}_{k,3}$ is equal to the game $\mathbf{G}_{1,k}$. In this game, CU_k is generated by encrypting random elements with a random key and Y_k is also generated randomly. That is, Y_k is replaced by a random element $\tilde{Y}_k$.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_j}$ be the advantage of $\mathcal{A}$ in the game $\mathbf{G}_j$. We have that $\mathbf{Adv}_{RIBE\text{-}AR,\mathcal{A}}^{SE\text{-}REV\text{-}PRV}(\lambda) = \mathbf{Adv}_{\mathcal{A}}^{G_0}$ and $\mathbf{Adv}_{\mathcal{A}}^{G_2} = 0$. From the following Lemmas 5.12, 5.13, 5.14, and 5.15, we obtain the equation

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}}^{G_0}(\lambda) &\leq \sum_{j=1}^{2} \left|\mathbf{Adv}_{\mathcal{A}}^{G_{j-1}} - \mathbf{Adv}_{\mathcal{A}}^{G_j}\right| + \mathbf{Adv}_{\mathcal{A}}^{G_2} \\
&\leq \left|\mathbf{Adv}_{\mathcal{A}}^{G_0} - \mathbf{Adv}_{\mathcal{A}}^{G_1}\right| + \sum_{k=1}^{\ell} \left|\mathbf{Adv}_{\mathcal{A}}^{G_{1,k-1}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,k}}\right| \\
&\leq \left|\mathbf{Adv}_{\mathcal{A}}^{G_0} - \mathbf{Adv}_{\mathcal{A}}^{G_1}\right| + \sum_{k=1}^{\ell} \sum_{j=1}^{3} \left|\mathbf{Adv}_{\mathcal{A}}^{H_{k,j-1}} - \mathbf{Adv}_{\mathcal{A}}^{H_{k,j}}\right| \\
&\leq \mathbf{Adv}_{\mathcal{B}}^{PRF}(\lambda) + r \log N \big(\mathbf{Adv}_{SKE,\mathcal{B}}^{IND\text{-}CPA}(\lambda) + \mathbf{Adv}_{SKE,\mathcal{B}}^{KEY\text{-}PRV}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{XDH}(\lambda)\big)
\end{aligned}$$

where r is the maximum number of revoked users in an update key. This completes the proof. □

Lemma 5.12. *If the PRF is secure, then no PPT adversary can distinguish $\mathbf{G}_0$ from $\mathbf{G}_1$ with a non-negligible advantage.*

The proof of this lemma is the same as Lemma 5.5.

Lemma 5.13. *If the SKE scheme is IND-CPA secure, then no PPT adversary can distinguish $\mathbf{H}_{k,0}$ from $\mathbf{H}_{k,1}$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that attacks the above RIBE-AR scheme with a non-negligible advantage. A simulator $\mathcal{B}$ that breaks the IND-CPA security of an SKE scheme using $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits challenge revoked sets R_0^*, R_1^* and challenge time T^*. $\mathcal{B}$ chooses a random bit $\mu \in \{0,1\}$ and obtains a challenge cover set $CV^* = \{v_1, \ldots, v_k, \ldots, v_\ell\}$ by running **CS.Cover**$(\mathcal{BT}, R_\mu^*)$. Let $r = |R_\mu^*|$, $\ell = |CV^*|$, and $\ell_m = \lceil r \log(N/r) \rceil$.

Setup: $\mathcal{B}$ obtains MK, RL, ST by running **RIBE-AR.Setup**$(1^\lambda, N)$. Note that the random κ_k of the node $v_k \in CV^*$ is unknown to $\mathcal{B}$ since κ_k is implicitly associated with the encryption key of the SKE scheme.

Phase 1: $\mathcal{B}$ handles private key, update key, and decryption key queries as follows: It can create a private key SK_{ID} by using MK and ST although κ_k of the node v_k is unknown since $PV \cap CV^* = \emptyset$ by the restriction of the security model. It can create an update key UK_T by using MK and ST by using the encryption oracle of the SKE scheme for the node v_k. It can also easily create a decryption key DK by using MK.

Challenge: $\mathcal{B}$ creates a challenge update key UK^* as follows:

1. It selects a random exponent $s \in \mathbb{Z}_p$ and sets $V^* = \hat{g}^s$.

2. For $1 \leq i \leq k-1$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and proceeds as follows: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random $\tilde{U}_{i,0}, \tilde{U}_{i,1}$. It computes $CU_i = \mathbf{SKE.Encrypt}(\kappa_i, \tilde{SK}_{IBE,i})$. It creates $NUK_i^* = \big(CU_i, Y_i = (V^*)^{\omega_i}\big)$.

3. For $i = k$, it retrieves $(\gamma_k, \omega_k, -)$ of the node v_k and proceeds as follows:

(a) It sets a random $\tilde{SK}_{IBE,k}$ by selecting random $\tilde{U}_{k,0}, \tilde{U}_{k,1}$. It obtains a normal $SK_{IBE,k}$ by running **IBE.GenKey**$(T, \hat{g}^{\alpha-\gamma_k}, PP_{IBE})$.

(b) It submits challenge messages $M_0^* = \tilde{SK}_{IBE,k}, M_1^* = SK_{IBE,k}$ to the challenge oracle of the SKE scheme and obtains a challenge ciphertext CT^* from the challenge oracle of SKE.

(c) It creates $NUK_k^* = \big(CU_k = CT^*, Y_k = (V^*)^{\omega_k}\big)$.

4. For $k+1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and proceeds as follows: It obtains a normal $SK_{IBE,i}$ by running **IBE.GenKey**$(T, \hat{g}^{\alpha-\gamma_i}, PP_{IBE})$. It computes $CU_i =$ **SKE.Encrypt**$(\kappa_i, SK_{IBE,i})$. It creates a normal $NUK_i^* = \big(CU_i, Y_i = (V^*)^{\omega_i}\big)$.

5. For $\ell+1 \leq i \leq \ell_m$, it proceeds as follows: It sets a random $\tilde{SK}_{IBE,i}$ by selecting random $\tilde{U}_{i,0}, \tilde{U}_{i,1}$. It computes $CU_i =$ **SKE.Encrypt**$(\tilde{\kappa}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{\kappa}_i$. It creates a random $NUK_i^* = \big(CU_i, \tilde{Y}_i\big)$ by selecting a random $\tilde{Y}_i$.

6. It creates $UK^* = \big(NUK^*_{\pi(1)}, \ldots, NUK^*_{\pi(\ell_m)}, V^*\big)$ where π is a random permutation.

Phase 2: Same as Phase 1.

Guess: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ also outputs μ'.

If a right message M_1^* is encrypted in the IND-CPA game, then $CU_k =$ **SKE.Encrypt**$(\kappa_k, SK_{IBE,k})$ which is equal to the game $\mathbf{H}_{k,0}$. Otherwise (a left message M_0^* is encrypted), $CU_k =$ **SKE.Encrypt**$(\kappa_k, \tilde{SK}_{IBE,k})$ which is equal to the game $\mathbf{H}_{k,1}$. This completes our proof. □

Lemma 5.14. *If the SKE scheme is KEY-PRV secure, then no PPT adversary can distinguish $\mathbf{H}_{k,1}$ from $\mathbf{H}_{k,2}$ with a non-negligible advantage.*

Proof. The proof of this lemma is almost similar to that of Lemma 5.13 except the generation of CU_k in NUK_k^*. Let $\mathcal{A}$ be an adversary that attacks the above RIBE-AR scheme with a non-negligible advantage and $\mathcal{B}$ be a simulator that breaks the KEY-PRV security of an SKE scheme using $\mathcal{A}$. The generation of private keys, update keys, and decryption keys are the same since the encryption oracle of the SKE scheme is given. The generation of NUK_i^* in a challenge update key UK^* is also similar except the generation of NUK_k^*. The k-th node update key NUK_k^* is generated as follows:

- For $i = k$, it retrieves $(\gamma_k, \omega_k, -)$ of the node v_k and proceeds as follows:
  1. It sets a random $\tilde{SK}_{IBE,k}$ by selecting random $\tilde{U}_{k,0}, \tilde{U}_{k,1}$.
  2. It submits a challenge message $M^* = \tilde{SK}_{IBE,k}$ to the challenge oracle of the SKE scheme and receives a challenge ciphertext CT^* from the challenge oracle of SKE.
  3. It creates $NUK_k^* = \big(CU_k = CT^*, Y_k = (V^*)^{\omega_k}\big)$.

If a valid encryption key κ_k is used in the KEY-PRV game, then $CU_k =$ **SKE.Encrypt**$(\kappa_k, \tilde{SK}_{IBE,k})$ which is equal to the game $\mathbf{H}_{k,1}$. Otherwise (a random encryption key $\tilde{\kappa}_k$ is used), $CU_k =$ **SKE.Encrypt**$(\tilde{\kappa}_k, \tilde{SK}_{IBE,k})$ which is equal to the game $\mathbf{H}_{k,2}$. This completes our proof. □

Lemma 5.15. *If the XDH assumption holds, then no PPT adversary can distinguish $\mathbf{H}_{k,2}$ from $\mathbf{H}_{k,3}$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that attacks the above RIBE-AR scheme with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the XDH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \hat{\mathbb{G}}, \mathbb{G}_T, e), g, \hat{g}, \hat{g}^a, \hat{g}^b)$ and Z where $Z = Z_0 = \hat{g}^{ab}$ or $Z = Z_1 \in_R \hat{\mathbb{G}}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits challenge revoked sets R_0^*, R_1^* and challenge time T^*. $\mathcal{B}$ chooses a random bit $\mu \in \{0,1\}$ and obtains a challenge cover set $CV^* = \{v_1, \ldots, v_k, \ldots, v_\ell\}$ by running **CS.Cover**$(\mathcal{BT}, R_\mu^*)$. Let $r = |R_\mu^*|$, $\ell = |CV^*|$, and $\ell_m = \lceil r \log(N/r) \rceil$.

Setup: $\mathcal{B}$ obtains MK, RL, ST by running **RIBE-AR.Setup**$(1^\lambda, N)$. Note that the random ω_k of the node $v_k \in CV^*$ is unknown to $\mathcal{B}$ since ω_k is implicitly associated with $\text{dlog}(\hat{g}^b)$ of the XDH assumption.

Phase 1: $\mathcal{B}$ handles private key, update key, and decryption key queries as follows: It can create a private key SK_{ID} by using MK and ST although ω_k of the node v_k is unknown since $PV \cap CV^* = \emptyset$ by the restriction of the security model. It can create an update key UK_T by using MK and ST by using $\hat{g}^{\omega_k} = \hat{g}^b$ for the node v_k and selecting a random exponent s.

Challenge: Let UK^* be the challenge update key that should be created in this step. $\mathcal{B}$ sets $V^* = \hat{g}^a$ by implicitly setting $s = a$ and creates all CU_i in UK^* randomly. Next it creates the hint values of UK^* as follows:

1. For $1 \le i \le k-1$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and sets a random Y_i.
2. For $i = k$, it retrieves $(\gamma_k, -, \kappa_k)$ of the node v_k and sets $Y_k = Z$ by implicitly setting $\omega_k = b$.
3. For $k+1 \le i \le \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and sets $Y_i = (\hat{g}^a)^{\omega_i}$.
4. For $\ell+1 \le i \le \ell_m$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node v_i and sets a random Y_i.

Phase 2: Same as Phase 1.

Guess: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ also outputs μ'.

If a valid $Z = g^{ab}$ is given, then a valid hint $Y_k = \hat{g}^{ab} = (\hat{g}^{\omega_k})^s$ is created like in the game $\mathbf{H}_{k,1}$. Otherwise (a random $Z = g^c$ is given), a random hint $Y_k = \hat{g}^c$ is created like in the game $\mathbf{H}_{k,2}$. This completes our proof. □

6 Conclusion

In this paper, we introduced the concept of RIBE-AR that provides ciphertext anonymity with revocation privacy, and proposed an efficient RIBE-AR scheme by combining AIBE and IBE schemes with the CS method in bilinear groups. Our RIBE-AR scheme has similar private key size, update key size, and ciphertext size compared to the previous efficient RIBE schemes, despite the revocation set of an update key is hidden. We proved the selective IND-CPA, selective ANO-CPA, and selective REV-PRV security of our RIBE-AR scheme under complexity assumptions in bilinear groups with the security of underlying PRF and SKE schemes. Since our RIBE-AR scheme can provide the weak revocation privacy that hides revocation set against outsider attackers who can only obtain revoked private keys, it is an interesting problem to design an RIBE-AR scheme that provides the strong revocation privacy that hides the revoked set against internal attackers who have access to private keys that were not revoked.

References

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2005.

[2] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 553–572. Springer, 2010.

[3] Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming - ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 1–9. Springer, 1999.

[4] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory Comput. Syst.*, 48(3):535–553, 2011.

[5] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, 2012.

[6] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 566–582. Springer, 2001.

[7] Alexandra Boldyreva, Vipul Goyal, and Virendra Kumar. Identity-based encryption with efficient revocation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM Conference on Computer and Communications Security - CCS 2008*, pages 417–426. ACM, 2008.

[8] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[9] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[10] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer, 2001.

[11] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In Salil P. Vadhan, editor, *Theory of Cryptography - TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554. Springer, 2007.

[12] Xavier Boyen. Multipurpose identity-based signcryption (A swiss army knife for identity-based cryptography). In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2003.

[13] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In Cynthia Dwork, editor, *Advances in Cryptology - CRYPTO 2006*, volume 4117 of *Lecture Notes in Computer Science*, pages 290–307. Springer, 2006.

[14] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 523–552. Springer, 2010.

[15] Jie Chen, Hoon Wei Lim, San Ling, Huaxiong Wang, and Khoa Nguyen. Revocable identity-based encryption from lattices. In Willy Susilo, Yi Mu, and Jennifer Seberry, editors, *Information Security and Privacy - ACISP 2012*, volume 7372 of *Lecture Notes in Computer Science*, pages 390–403. Springer, 2012.

[16] Léo Ducas. Anonymity from asymmetry: New constructions for anonymous HIBE. In Josef Pieprzyk, editor, *Topics in Cryptology - CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 148–164. Springer, 2010.

[17] Keita Emura, Atsushi Takayasu, and Yohei Watanabe. Adaptively secure revocable hierarchical IBE from k-linear assumption. *Des. Codes Cryptogr.*, 89(7):1535–1574, 2021.

[18] Nelly Fazio and Irippuge Milinda Perera. Outsider-anonymous broadcast encryption with sublinear ciphertexts. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public-Key Cryptography - PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 225–242. Springer, 2012.

[19] Craig Gentry. Practical identity-based encryption without random oracles. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 445–464. Springer, 2006.

[20] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Cynthia Dwork, editor, *ACM Symposium on Theory of Computing - STOC 2008*, pages 197–206. ACM, 2008.

[21] Shuichi Katsumata, Takahiro Matsuda, and Atsushi Takayasu. Lattice-based revocable (hierarchical) IBE with decryption key exposure resistance. In Dongdai Lin and Kazue Sako, editors, *Public-Key Cryptography - PKC 2019*, volume 11443 of *Lecture Notes in Computer Science*, pages 441–471. Springer, 2019.

[22] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *Advances in Cryptology - EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162. Springer, 2008.

[23] Kwangsu Lee. A generic construction for revocable identity-based encryption with subset difference methods. Cryptology ePrint Archive, Report 2019/798, 2019. `http://eprint.iacr.org/2019/798`.

[24] Kwangsu Lee. Revocable hierarchical identity-based encryption with adaptive security. *Theoretical Computer Science*, 880:37–68, 2021.

[25] Kwangsu Lee. Delegate and verify the update keys of revocable identity-based encryption. *IEEE Access*, 11:52636–52652, 2023.

[26] Kwangsu Lee and Joon Sik Kim. A generic approach to build revocable hierarchical identity-based encryption. *IEEE Access*, 10:44178–44199, 2022.

[27] Kwangsu Lee, Dong Hoon Lee, and Jong Hwan Park. Efficient revocable identity-based encryption via subset difference methods. *Des. Codes Cryptogr.*, 85(1):39–76, 2017.

[28] Kwangsu Lee and Seunghwan Park. Revocable hierarchical identity-based encryption with shorter private keys and update keys. *Des. Codes Cryptogr.*, 86(10):2407–2440, 2018.

[29] Benoît Libert, Kenneth G. Paterson, and Elizabeth A. Quaglia. Anonymous broadcast encryption: Adaptive security and efficient constructions in the standard model. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *Public-Key Cryptography - PKC 2012*, volume 7293 of *Lecture Notes in Computer Science*, pages 206–224. Springer, 2012.

[30] Benoît Libert and Damien Vergnaud. Adaptive-ID secure revocable identity-based encryption. In Marc Fischlin, editor, *Topics in Cryptology - CT-RSA 2009*, volume 5473 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2009.

[31] Xuecheng Ma and Dongdai Lin. Generic constructions of revocable identity-based encryption. In Zhe Liu and Moti Yung, editors, *Information Security and Cryptology - Inscrypt 2019*, volume 12020 of *Lecture Notes in Computer Science*, pages 381–396. Springer, 2019.

[32] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718. Springer, 2012.

[33] Dalit Naor, Moni Naor, and Jeffery Lotspiech. Revocation and tracing schemes for stateless receivers. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 41–62. Springer, 2001.

[34] Seunghwan Park, Kwangsu Lee, and Dong Hoon Lee. New constructions of revocable identity-based encryption from multilinear maps. *IEEE Trans. Inf. Forensic Secur.*, 10(8):1564–1577, 2015.

[35] Baodong Qin, Robert H. Deng, Yingjiu Li, and Shengli Liu. Server-aided revocable identity-based encryption. In Günther Pernul, Peter Y. A. Ryan, and Edgar R. Weippl, editors, *Computer Security - ESORICS 2015*, volume 9326 of *Lecture Notes in Computer Science*, pages 286–304. Springer, 2015.

[36] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *ACM Symposium on Theory of Computing - STOC 2005*, pages 84–93. ACM, 2005.

[37] Jae Hong Seo and Keita Emura. Efficient delegation of key generation and revocation functionalities in identity-based encryption. In Ed Dawson, editor, *Topics in Cryptology - CT-RSA 2013*, volume 7779 of *Lecture Notes in Computer Science*, pages 343–358. Springer, 2013.

[38] Jae Hong Seo and Keita Emura. Revocable identity-based encryption revisited: Security model and construction. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography - PKC 2013*, volume 7778 of *Lecture Notes in Computer Science*, pages 216–234. Springer, 2013.

[39] Jae Hong Seo and Keita Emura. Revocable hierarchical identity-based encryption: History-free update, security against insiders, and short ciphertexts. In Kaisa Nyberg, editor, *Topics in Cryptology - CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 106–123. Springer, 2015.

[40] Yohei Watanabe, Keita Emura, and Jae Hong Seo. New revocable IBE in prime-order groups: Adaptively secure, decryption key exposure resistant, and with short public parameters. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017*, volume 10159 of *Lecture Notes in Computer Science*, pages 432–449. Springer, 2017.

[41] Brent R. Waters, Dirk Balfanz, Glenn Durfee, and Diana K. Smetters. Building an encrypted and searchable audit log. In *NDSS 2004*. The Internet Society, 2004.

A Construction from Lattices

In this section, we present an RIBE-AR scheme from lattices without DKER. By following the method of Katsumata et al. [21], our RIBE-AR scheme can be modified to provide DKER.

A.1 Lattices

Lemma A.1. *Let $n,m,\tilde{m},q$ be positive integers with $m \geq 2n\log q$ and q prime. There are polynomial time algorithms for generating short basis of lattices as follows:*

GenTrap$(1^n, 1^m, q)$**:** *It is a randomized algorithm that outputs a full rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a trapdoor 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 $\|\mathbf{T_A}\|_{GS} = O(\sqrt{n\log q})$ with overwhelming probability in n [3, 4, 32].*

ExtendRandLeft$(\mathbf{A}, \mathbf{F}, \mathbf{T_A}, q)$**:** *It is a randomized algorithm that, given as input matrices $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{F} \in \mathbb{Z}_q^{n\times\tilde{m}}$, a basis $\mathbf{T_A}$ of $\Lambda_q^{\perp}(\mathbf{A})$, and a Gaussian parameter $\sigma \geq \|\mathbf{T_A}\|_{GS}\cdot\omega(\sqrt{\log n})$, outputs a matrix $\mathbf{T}_{[\mathbf{A}|\mathbf{F}]} \in \mathbb{Z}^{(m+\tilde{m})\times(m+\tilde{m})}$ distributed statistically close to D [14].*

ExtendRandRight$(\mathbf{A}, \mathbf{G}, \mathbf{R}, \mathbf{T_G}, \sigma)$**:** *It is a randomized algorithm that, given as input a full rank matrix $\mathbf{A}, \mathbf{G} \in \mathbb{Z}_q^{n\times m}$, 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 D [2].*

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 $\mathbf{T_G}$ with $\|\mathbf{T_G}\|_{GS} \leq \sqrt{5}$ [32].

Lemma A.2. *There are polynomial time algorithms for sampling a short vector as follows:*

SampleLeft$(\mathbf{A}, \mathbf{F}, \mathbf{T_A}, \mathbf{u}, \sigma)$**:** *It is a randomized algorithm that, given as input a full rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, a matrix $\mathbf{F} \in \mathbb{Z}_q^{n\times m}$, a basis $\mathbf{T_A} \in \mathbb{Z}^{m\times m}$ of $\Lambda_q^{\perp}(\mathbf{A})$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter $\sigma \geq \|\mathbf{T_A}\|_{GS}\cdot\omega(\sqrt{\log n})$, outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+\tilde{m}}$ sampled from a distribution statistically close to $\mathcal{D}_{\Lambda_q^{\mathbf{u}}([\mathbf{A}|\mathbf{F}]),\sigma}$ [2, 32].*

Assumption 4 (Learning with Errors, LWE [36])**.** The LWE problem is to distinguish the following distributions:

$$(\mathbf{A}, \mathbf{A}^{\top}\mathbf{s}+\mathbf{x}) \quad \text{and} \quad (\mathbf{A}, \mathbf{u})$$

where $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{x} \in \mathcal{D}$, $\mathbf{u} \in \mathbb{Z}_q^m$ are independently sampled. The advantage of LWE is defined as $\mathbf{Adv}_{\mathcal{A}}^{LWE} = |\Pr[\mathcal{A}(\mathbf{A}, \mathbf{A}^{\top}\mathbf{s}+\mathbf{x}) = 1] - \Pr[\mathcal{A}(\mathbf{A}, \mathbf{u}) = 1]|$. We say that the LWE assumption holds if the above advantage is negligible for all PPT adversaries.

Assumption 5 (Learning with Rounding, LWR [5])**.** The LWR problem is to distinguish the following distributions:

$$(\mathbf{A}, \lfloor\mathbf{A}^{\top}\mathbf{s}\rceil_p) \quad \text{and} \quad (\mathbf{A}, \mathbf{v})$$

where $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{x} \in \mathcal{D}$, $\mathbf{v} \in \mathbb{Z}_p^m$ are independently sampled. The advantage of LWR is defined as $\mathbf{Adv}_{\mathcal{A}}^{LWR} = |\Pr[\mathcal{A}(\mathbf{A}, \lfloor\mathbf{A}^{\top}\mathbf{s}\rceil_p) = 1] - \Pr[\mathcal{A}(\mathbf{A}, \mathbf{v}) = 1]|$. We say that the LWR assumption holds if the above advantage is negligible for all PPT adversaries.

A.2 Construction

Let PRF be a pseudo-random function for $\mathcal{K} = \{0,1\}^\lambda$, $\mathcal{X} = \{0,1\}^*$, and $\mathcal{Y} = \mathbb{Z}_p$. Our RIBE-AR scheme for $\mathcal{I} = \{0,1\}^n$, $\mathcal{T} = \{0,1\}^n$, and $\mathcal{M} \in \{0,1\}$ is described as follows:

RIBE-AR.Setup$(1^\lambda, N)$**:** Let λ be a security parameter and N be the maximum number of users. Let $n = \lambda$.

1. It obtains $(\mathbf{A}_1, \mathbf{T}_{\mathbf{A}_1})$ and $(\mathbf{A}_2, \mathbf{T}_{\mathbf{A}_2})$ by running **GenTrap**$(1^n, 1^m, q)$ and **GenTrap**$(1^n, 1^m, q)$ respectively. It also samples uniformly random matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}$ and vector $\mathbf{u} \leftarrow \mathbb{Z}_q^n$.
2. It generates two PRF keys z_1, z_2 and obtains $\mathcal{BT}$ by running **CS.Setup**(N).
3. Finally, it outputs a master key $MK = (\mathbf{T}_{\mathbf{A}_1}, \mathbf{T}_{\mathbf{A}_2}, z_1, z_2)$, a revocation list $RL = \emptyset$, a state $ST = (\mathcal{BT})$, and public parameters $PP = \big(\mathbf{A}_1, \mathbf{A}_2, \mathbf{B}, \mathbf{u}\big)$.

RIBE-AR.GenKey(ID, MK, ST, PP)**:** Let $MK = (\mathbf{T}_{\mathbf{A}_1}, \mathbf{T}_{\mathbf{A}_2}, z_1, z_2)$ and $ST = (\mathcal{BT})$.

1. It assigns ID to a random leaf node $v \in \mathcal{BT}$ and obtains a private set $PV = \{S_0, \ldots, S_n\}$ by running **CS.Assign**$(\mathcal{BT}, v)$.
2. For $j = 0$ to $j = n$, it proceeds as follows: Let $L_j = \text{Label}(S_j)$ be a label string. It computes $(\mathbf{u}_j, \mathbf{w}_j) = \mathbf{PRF}_1(z_1, L_j)$ and $\kappa_j = \mathbf{PRF}_2(z_2, L_j)$ where $\mathbf{u}_j, \mathbf{w}_j \in \mathbb{Z}_q^n$. It samples a vector $\mathbf{e}_j$ by running **SampleLeft**$(\mathbf{A}_1, \mathbf{E}(ID), \mathbf{u}_j, \mathbf{T}_{\mathbf{A}_1}, \sigma)$ such that
$$[\mathbf{A}_1 | \mathbf{E}(ID)]^\top \mathbf{e}_j = \mathbf{u}_j.$$
Next, it creates a node private key $NSK_j = \big(\mathbf{e}_j, \mathbf{w}_j, \kappa_j\big)$.
3. It obtains an extended basis $\mathbf{T}_{[\mathbf{A}_2|\mathbf{E}(ID)]}$ by running **ExtendRandLeft**$(\mathbf{A}_2, \mathbf{E}(ID), \mathbf{T}_{\mathbf{A}_2}, \sigma)$.
4. Finally, it outputs a private key $SK_{ID} = \big(NSK_0, \ldots, NSK_n, \mathbf{T}_{[\mathbf{A}_2|\mathbf{E}(ID)]}\big)$.

RIBE-AR.UpdateKey(T, RL, MK, ST, PP)**:** Let $MK = (\mathbf{T}_{\mathbf{A}_1}, \mathbf{T}_{\mathbf{A}_2}, z_1, z_2)$ and $ST = (\mathcal{BT})$.

1. It derives a revoked set R on time T from RL and obtains a cover set $CV = \{S_1, \ldots, S_\ell\}$ by running **CS.Cover**$(\mathcal{BT}, R)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$.
2. It selects a uniformly random matrix $\mathbf{V} \in \mathbb{Z}_q^{n \times n}$.
3. For $1 \le i \le \ell$, it proceeds as follows: Let $L_i = \text{Label}(S_i)$ be a label string. It computes $(\mathbf{u}_j, \mathbf{w}_j) = \mathbf{PRF}_1(z_1, L_j)$ and $\kappa_j = \mathbf{PRF}_2(z_2, L_j)$ where $\mathbf{u}_j, \mathbf{w}_j \in \mathbb{Z}_q^n$. It samples a vector $\mathbf{f}_i$ by running **SampleLeft**$(\mathbf{A}_1, \mathbf{F}(T), \mathbf{u} - \mathbf{u}_i, \mathbf{T}_{\mathbf{A}_1}, \sigma)$ such that
$$[\mathbf{A}_1 | \mathbf{F}(T)]^\top \mathbf{f}_i = \mathbf{u} - \mathbf{u}_i.$$
It obtains a ciphertext $CU_i = \mathbf{SKE.Encrypt}(\kappa_i, \mathbf{f}_i)$. Next, it sets a hint vector $\mathbf{y}_i = \lfloor \mathbf{V}^\top \mathbf{w}_i \rceil_p$ and creates a node update key $NUK_i = \big(CU_i, \mathbf{y}_i\big)$.
4. For $\ell + 1 \le i \le \ell_m$, it proceeds as follows: It selects random $\mathbf{f}_i$ and obtains $CU_i = \mathbf{SKE.Encrypt}(\kappa_i, \mathbf{f}_i)$ by using a random key κ_i. It creates a node update key $NUK_i = \big(CU_i, \mathbf{y}_i\big)$.
5. Finally, it selects a random permutation $\pi : \{1, \ldots, \ell_m\} \to \{1, \ldots, \ell_m\}$ and outputs an update key $UK_T = \big(NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, \mathbf{V}\big)$.

RIBE-AR.DeriveKey$(ID, T, SK_{ID}, UK_T, PP)$**:** Let $SK_{ID} = (NSK_0, \ldots, NSK_n, \mathbf{T}_{[\mathbf{A}_2|\mathbf{E}(ID)]})$ and $UK_T = (NUK_1, \ldots, NUK_{\ell_m}, \mathbf{V})$.

1. It retrieves $\mathbf{w}_j$ from NSK_j for all $j \in \{0, \ldots, n\}$ and sets a list $(\mathbf{y}'_0 = \lfloor \mathbf{V}^\top \mathbf{w}_0 \rceil_p, \ldots, \mathbf{y}'_n = \lfloor \mathbf{V}^\top \mathbf{w}_n \rceil_p)$. It also retrieves $\mathbf{y}_i$ from NUK_i for all $i \in \{1, \ldots, \ell_m\}$ and sets a list $(\mathbf{y}_1, \ldots, \mathbf{y}_{\ell_m})$.
2. It finds two indexes $j \in \{0, \ldots, n\}$ and $i \in \{1, \ldots, \ell_m\}$ such that $\mathbf{y}'_j = \mathbf{y}_i$ in the lists, then it retrieves the corresponding $NSK_j = (\mathbf{e}_j, \mathbf{w}_j, \kappa_j)$ and $NUK_i = (CU_i, \mathbf{y}_i)$. Next, it decrypts $\mathbf{f}_i =$ **SKE.Decrypt**(κ_j, CU_i).
3. It parses $\mathbf{e}_j = [\mathbf{e}_{j,L} | \mathbf{e}_{j,R}]$ and $\mathbf{f}_i = [\mathbf{f}_{i,L} | \mathbf{f}_{i,R}]$ where $\mathbf{e}_{j,L}, \mathbf{f}_{i,L}, \mathbf{e}_{j,R}, \mathbf{f}_{i,R} \in \mathbb{Z}^m$. Then it computes $\mathbf{d}_1 = [\mathbf{e}_{j,L} + \mathbf{f}_{i,L} | \mathbf{e}_{j,R} | \mathbf{f}_{i,R}]$ such that
$$[\mathbf{A}_1 | \mathbf{E}(ID) | \mathbf{F}(T)]^\top \mathbf{d}_1 = \mathbf{u}.$$
4. It also samples $\mathbf{d}_2$ by running **SampleLeft**$([\mathbf{A}_2 | \mathbf{E}(ID)], \mathbf{F}(T), \mathbf{u}, \mathbf{T}_{[\mathbf{A}_2 | \mathbf{E}(ID)]}, \sigma)$ such that
$$[\mathbf{A}_2 | \mathbf{E}(ID) | \mathbf{F}(T)]^\top \mathbf{d}_2 = \mathbf{u}.$$
5. Finally, it outputs a decryption key $DK_{ID,T} = \big(\mathbf{d}_1, \mathbf{d}_2\big)$.

RIBE-AR.Encrypt(ID, T, M, PP)**:** It first samples uniformly random vectors $\mathbf{s}_1, \mathbf{s}_2 \in \mathbb{Z}_q^n$. It also samples $x \leftarrow D_{\mathbb{Z}, \alpha q}$, $\mathbf{x}_1, \mathbf{x}_2 \leftarrow D_{\mathbb{Z}^{3m}, \alpha' q}$ and sets

$$c_0 = \mathbf{u}^\top (\mathbf{s}_1 + \mathbf{s}_2) + x + M \lfloor q/2 \rfloor,$$
$$\mathbf{c}_1 = [\mathbf{A}_1 | \mathbf{E}(ID) | \mathbf{F}(T)]^\top \mathbf{s}_1 + \mathbf{x}_1,$$
$$\mathbf{c}_2 = [\mathbf{A}_2 | \mathbf{E}(ID) | \mathbf{F}(T)]^\top \mathbf{s}_2 + \mathbf{x}_2.$$

Finally, it outputs a ciphertext $CT = \big(c_0, \mathbf{c}_1, \mathbf{c}_2\big)$.

RIBE-AR.Decrypt$(CT, DK_{ID,T}, PP)$**:** Let $CT = (c_0, \mathbf{c}_1, \mathbf{c}_2)$ and $DK_{ID,T} = (\mathbf{d}_1, \mathbf{d}_2)$. From the ciphertext and the decryption key, it computes

$$c' = c_0 - \mathbf{c}_1^\top \mathbf{d}_1 - \mathbf{c}_2^\top \mathbf{d}_2.$$

It outputs 1 if $|c' - \lfloor q/2 \rfloor| < \lfloor q/4 \rfloor$ and 0 otherwise.

RIBE-AR.Revoke(ID, T, RL)**:** If ID is not assigned in $\mathcal{BT}$, then it outputs $\perp$. Otherwise, it updates RL by adding (ID, T) to RL.