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.
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

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

Let \( \lambda \) 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.

### 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 \( \kappa_i \) and encrypts the corresponding node update key \( NUK_i \) with the key \( \kappa_i \). 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 \( \kappa_i \). 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 \( \kappa_i \) 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 $\omega_i$ 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 $(\omega_1, \ldots, \omega_n)$ corresponding to the path nodes of a binary tree, and uses the element $V$ of the update key to derive elements $(V^{\omega_1}, \ldots, V^{\omega_n})$. Afterwards, the receiver can quickly find a matching node by comparing these derived elements with the hint values $(Y_1, \ldots, Y_m)$ 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 \( F: K \times X \rightarrow Y \) where \( K \) is the key space, \( X \) is the domain, and \( Y \) is the range. Let \( F(k, \cdot) \) be an oracle for a uniformly chosen \( k \in K \) and \( f(\cdot) \) be an oracle for a uniformly chosen function \( f: X \rightarrow Y \). We say that a PRF is secure if for all probabilistic polynomial-time adversaries \( A \) the advantage \( \text{Adv}^\text{PRF}_A(\lambda) = \Pr[A_{F(k, \cdot)} = 1] - \Pr[A_{f(\cdot)} = 1] \) 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 \( \text{GenKey}, \text{Encrypt}, \) and \( \text{Decrypt}, \) which are defined as follows:

- \( \text{GenKey}(1^\lambda) \). The key generation algorithm takes as input a security parameter \( \lambda \). It outputs a symmetric key \( K \).
- \( \text{Encrypt}(K, M) \). The encryption algorithm takes as input a message \( M \in M \) and the symmetric key \( K \). It outputs a ciphertext \( C \).
- \( \text{Decrypt}(K, C) \). 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 \( \text{GenKey} \) and any message \( M \in M \), it is required that \( \text{Decrypt}(K, \text{Encrypt}(K, M)) = M \).

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 \( \text{EXP}_{A,\text{IND-CPA}}(1^\lambda) \) between a challenger \( C \) and a PPT adversary \( A \):

1. **Setup**: \( C \) generates a symmetric key \( K \) by running \( \text{GenKey}(1^\lambda) \). It keeps \( K \) to itself.
2. **Phase 1**: \( A \) adaptively request a polynomial number of encryption queries. For each encryption query for a message \( M \), \( C \) generates a ciphertext \( CT \) by running \( \text{Encrypt}(K, M) \) and gives \( CT \) to \( A \).
3. **Challenge**: \( A \) submits challenge messages \( M^*_0, M^*_1 \) where \( |M^*_0| = |M^*_1| \). \( C \) flips a random coin \( \mu \in \{0, 1\} \) and gives a challenge ciphertext \( CT^* \) to \( A \) by running \( \text{Encrypt}(K, M^*_\mu) \).
4. **Phase 2**: \( A \) may continues to request additional encryption queries and \( C \) handles these queries as the same as the phase 1.
5. **Guess:** $\mathcal{A}$ outputs a guess $\mu' \in \{0, 1\}$ of $\mu$. $C$ outputs 1 if $\mu = \mu'$ or 0 otherwise.

The advantage of $\mathcal{A}$ is defined as $\text{Adv}^{\text{IND-CPA}}_{\mathcal{A}}(\lambda) = \left| \Pr[\text{EXP}_{\mathcal{A}}^{\text{IND-CPA}}(1^\lambda) = 1] - \frac{1}{2} \right|$. 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 $\lambda$.

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 $\text{EXP}_{\mathcal{A}}^{\text{KEY-PRV}}(1^\lambda)$ between a challenger $C$ and a PPT adversary $\mathcal{A}$:

1. **Setup:** $C$ generates symmetric keys $K_0, K_1$ by running $\text{GenKey}(1^\lambda)$. It keeps $K_0, K_1$ to itself.
2. **Phase 1:** $\mathcal{A}$ adaptively requests a polynomial number of encryption queries. For each encryption query for a message $M$ and a choice $b \in \{0, 1\}$, $C$ generates a ciphertext $CT$ by running $\text{Encrypt}(K_b, M)$ and gives $CT$ to $\mathcal{A}$.
3. **Challenge:** $\mathcal{A}$ submits a challenge message $M^\ast$. $C$ flips a random coin $\mu \in \{0, 1\}$ and gives a challenge ciphertext $CT^\ast$ to $\mathcal{A}$ by running $\text{Encrypt}(K_\mu, M^\ast)$.
4. **Phase 2:** $\mathcal{A}$ may continue to request additional encryption queries and $C$ handles these queries as the same as the phase 1.
5. **Guess:** $\mathcal{A}$ outputs a guess $\mu' \in \{0, 1\}$ of $\mu$. $\mathcal{A}$ outputs 1 if $\mu = \mu'$ or 0 otherwise.

The advantage of $\mathcal{A}$ is defined as $\text{Adv}^{\text{KEY-PRV}}_{\mathcal{A}}(\lambda) = \left| \Pr[\text{EXP}_{\mathcal{A}}^{\text{KEY-PRV}}(1^\lambda) = 1] - \frac{1}{2} \right|$. 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 $\lambda$.

### 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 $BT$ of depth $n$. Each user is assigned to a different leaf node in $BT$. It outputs the binary tree $BT$.

**CS.Assign($BT, v$):** 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($BT, R$):** Let $R$ be a set of revoked users which consists of leaf nodes. It computes the Steiner tree $ST_R$. Let $T_{k_1}, \ldots, T_{k_m}$ be all the subtrees of $BT$ that hang off $ST_R$, that is all subtrees whose roots $v_{k_1}, \ldots, v_{k_m}$ are not in $ST_R$ but adjacent to nodes of outdegree 1 in $ST_R$. 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 $\bot$.

The correctness of the CS scheme requires that if $v \not\in R$, then $\text{CS.Match}(CV, PV) = (v_k, v_k)$ 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^λ,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 ∈ 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 ∈ 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 ∈ I, time T, a message M ∈ 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^λ,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) ∉ RL for all T_j ≤ 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 \(\text{EXP}^{\text{SE-IND-CPA}}(\lambda)\) between a challenger \(C\) and a PPT adversary \(A\):

1. **Init**: \(A\) submits a challenge identity \(ID^*\) and challenge time \(T^*\).
2. **Setup**: \(C\) generates a master key \(MK\), a revocation list \(RL\), a state \(ST\), and public parameters \(PP\) by running \(\text{Setup}(1^\lambda, N)\). It initializes a current time period \(T_e = 1\) and gives \(PP\) to \(A\).
3. **Phase 1**: \(A\) adaptively request a polynomial number of queries. \(C\) 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 \(\text{GenKey}(ID, MK, PP)\) and gives \(SK_{ID}\) to \(A\).
   - Update key query: It generates \(UK_T\) by running \(\text{UpdateKey}(T_c, RL, MK, PP)\), sets \(T_c = T_e + 1\), and gives \(UK_T\) to \(A\).
   - Decryption key query for an identity \(ID\) and time \(T\): It checks that \((ID, T) \neq (ID^*, T^*)\) and \(1 \leq T < T_e\). It generates \(DK_{ID,T}\) by running \(\text{DeriveKey}(SK_{ID}, UK_T, PP)\) and gives \(DK_{ID,T}\) to \(A\).
   - Revocation query for an identity \(ID\): It updates \(RL\) by running \(\text{Revoke}(ID, T_e, RL)\).
4. **Challenge**: \(A\) submits two challenge messages \(M_0^*, M_1^*\) with equal length. \(C\) flips random \(\mu \in \{0, 1\}\). It obtains a challenge ciphertext \(CT^*\) by running \(\text{Encrypt}(ID^*, T^*, M_\mu^*, PP)\) and gives \(CT^*\) to \(A\).
5. **Phase 2**: \(A\) may continue to request additional queries. \(B\) handles these queries as the same as the phase 1.
6. **Guess**: Finally, \(A\) outputs a guess \(\mu' \in \{0, 1\}\), and wins the game if \(\mu = \mu'\).

The advantage of \(A\) is defined as \(\text{Adv}^{\text{SE-IND-CPA}}_{\text{RIBE-AR}, A}(\lambda) = | \Pr[\text{EXP}^{\text{SE-IND-CPA}}_{A}(1^\lambda) = 1] - \frac{1}{2} |\) 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 \(A\), the advantage of \(A\) is negligible in the security parameter \(\lambda\).

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 $\text{EXP}_A^{SE-ANO-CPA}(\lambda)$ between a challenger $C$ and a PPT adversary $A$:

1. **Init**: $A$ submits two challenge identities $ID_0^*, ID_1^*$ and challenge time $T^*$.

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

3. **Phase 1**: $A$ adaptively request a polynomial number of queries. $C$ 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 $\text{GenKey}(ID, MK, PP)$ and gives $SK_{ID}$ to $A$.
   - Update key query: It generates $UK_{T_c}$ by running $\text{UpdateKey}(T_c, RL, MK, PP)$, sets $T_c = T_c + 1$, and gives $UK_{T_c}$ to $A$.
   - 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 $\text{DeriveKey}(SK_{ID}, UK_{T}, PP)$ and gives $DK_{ID, T}$ to $A$.
   - Revocation query for an identity $ID$: It updates $RL$ by running $\text{Revoke}(ID, T_c, RL)$.

4. **Challenge**: $A$ submits a challenge message $M^*$. $C$ flips random $\mu \in \{0, 1\}$. It obtains a challenge ciphertext $CT^*$ by running $\text{Encrypt}(ID_\mu^*, T^*, M^*, PP)$ and gives $CT^*$ to $A$.

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

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

The advantage of $A$ is defined as $\text{Adv}_{A}^{SE-ANO-CPA}(\lambda) = \Pr[\text{EXP}_A^{SE-ANO-CPA}(\lambda) = 1] - \frac{1}{2}$ 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 $A$, the advantage of $A$ is negligible in the security parameter $\lambda$.

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 $\text{EXP}_A^{SE-REV-PRV}(\lambda)$ between a challenger $C$ and a PPT adversary $A$:

1. **Init**: $A$ submits two challenge revoked sets $R_0^*, R_1^*$ and challenge time $T^*$ such that $|R_0^*| = |R_1^*|$.

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

10
3. **Phase 1:** \(A\) adaptively request a polynomial number of queries. \(C\) 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 \(A\).
- 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 \(A\).
- 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 \(DeriveKey(SK_{ID}, UK_{T_c}, PP)\) and gives \(DK_{ID,T}\) to \(A\).
- Revocation query for an identity \(ID\): It updates \(RL\) by running \(Revoke(ID, T_c, RL)\).

4. **Challenge:** \(C\) flips random random \(\mu \in \{0, 1\}\) and proceeds as follows:

   (a) For each \(ID \in R_0^*\), 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 \(A\).

5. **Phase 2:** \(A\) may continue to request additional queries. \(B\) handles these queries as the same as the phase 1.

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

The advantage of \(A\) is defined as \(Adv_{\text{SE-REV-PRV}}^{\text{RIBE-AR}, A}(\lambda) = \left| \Pr_{\text{EXP}_{\text{SE-REV-PRV}}^{\text{RIBE-AR}, A}}(1^\lambda = 1) - \frac{1}{2} \right|\) 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 \(A\), the advantage of \(A\) is negligible in the security parameter \(\lambda\).

**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 \((U \setminus R_0^*) \cap (U \setminus R_1^*)\) (private keys of non-revoked users) where \(U\) 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 \(\lambda\) and outputs a tuple \((p, G, \hat{G}, G_T, e)\) where \(p\) is a random prime and \(G, \hat{G}, \) and \(G_T\) be three cyclic groups of prime order \(p\). Let \(g\) and \(\hat{g}\) be generators of \(G\) and \(\hat{G}\), respectively. The bilinear map \(e : G \times \hat{G} \rightarrow G_T\) has the following properties:

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

We say that \(G, \hat{G}, G_T\) are asymmetric bilinear groups with no efficiently computable isomorphisms if the group operations in \(G, \hat{G}, \) and \(G_T\) as well as the bilinear map \(e\) are all efficiently computable, but there are no efficiently computable isomorphisms between \(G\) and \(\hat{G}\).
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 \((p, G, \hat{G}, G_T, e)\) be an asymmetric bilinear group generated by \(G(1^k)\). Let \(g, \hat{g}\) be random generators of \(G, \hat{G}\) respectively. The decisional bilinear Diffie-Hellman (DBDH) assumption is that if the challenge tuple

\[
D = ((p, G, \hat{G}, G_T, e), g, g^a, g^c, \hat{g}, \hat{g}^{a^b}, \hat{g}^b) \quad \text{and} \quad Z
\]

are given, no PPT algorithm \(A\) 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 \(A\) is defined as \(\text{Adv}_{\text{DBDH}}^A(\lambda) = \left| \Pr[A(D, Z_0) = 0] - \Pr[A(D, Z_1) = 0] \right|\) 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 \((p, G, \hat{G}, G_T, e)\) be an asymmetric bilinear group generated by \(G(1^k)\). Let \(g, \hat{g}\) be random generators of \(G, \hat{G}\) respectively. The decisional XDH assumption in \(\hat{G}\) is that if the challenge tuple

\[
D = ((p, G, \hat{G}, G_T, e), g, \hat{g}, \hat{g}^a, \hat{g}^b) \quad \text{and} \quad Z
\]

are given, no PPT algorithm \(A\) can distinguish \(Z = Z_0 = \hat{g}^{ab}\) from \(Z = Z_1 = \hat{g}^c\) with more than a negligible advantage. The advantage of \(A\) is defined as \(\text{Adv}_{\text{XDH}}^A(\lambda) = \left| \Pr[A(D, Z_0) = 0] - \Pr[A(D, Z_1) = 0] \right|\) 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 \((p, G, \hat{G}, G_T, e)\) be an asymmetric bilinear group generated by \(G(1^k)\). Let \(g, \hat{g}\) be random generators of \(G, \hat{G}\) respectively. The decisional PBDH assumption is that if the challenge tuple

\[
D = ((p, G, \hat{G}, G_T, e), g, g^a, g^{ab}, g^c, \hat{g}, \hat{g}^a, \hat{g}^b) \quad \text{and} \quad Z
\]

are given, no PPT algorithm \(A\) can distinguish \(Z = Z_0 = g^{abc}\) from \(Z = Z_1 = g^d\) with more than a negligible advantage. The advantage of \(A\) is defined as \(\text{Adv}_{\text{PBDH}}^A(\lambda) = \left| \Pr[A(D, Z_0) = 0] - \Pr[A(D, Z_1) = 0] \right|\) 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 \(I = \mathbb{Z}_p\) from the AHIBE scheme of Ducas is described as follows:

**AIBE.Setup(GDS):** Let \(GDS = ((p, G, \hat{G}, G_T, e), g, \hat{g})\) be the description of a bilinear group with generators \(g \in G, \hat{g} \in \hat{G}\). It selects random exponents \(u_1, h_1, w \in \mathbb{Z}_p\) 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 \(\text{MP} = (\hat{u}_1, \hat{h}_1, \hat{w})\), and public parameters \(\text{PP} = (GDS, u_1, h_1, w, \Lambda = e(g, \hat{g})^\gamma)\).
AIBE.GenKey(ID, MK, MP, PP): It chooses random exponents \( r_1, \ldots, r_6 \in \mathbb{Z}_p \) and outputs a private key \( SK_{ID} = (K_0 = MK(\hat{a}_1^{r_1}h_1)^{r_3}\hat{w}_1^2, K_1 = \hat{g}^{-r_1}, K_2 = \hat{g}^{-r_2}) \) with a random key \( RK_{ID} = (L_{0, 1} = (\hat{a}_1^{r_1}h_1)^{r_1}\hat{w}_1^2, L_{1, 1} = \hat{g}^{-r_1}, L_{2, 1} = \hat{g}^{-r_2}, L_{0, 2} = (\hat{a}_1^{r_1}h_1)^{r_3}\hat{w}_1^2, L_{1, 2} = \hat{g}^{-r_1}, L_{2, 2} = \hat{g}^{-r_2}) \).

AIBE.RandKey(SK_{ID}, RK_{ID}, PP): Let \( SK_{ID} = (K'_0, K'_1, K'_2) \) and \( RK_{ID} = (L_{0, 1, 1, 1, 1, 1}, L_{0, 2, 1, 1, 2, 2}, L_{1, 1, 2, 2}, K_1 = K'_1, L_{1, 2}, K_2 = K'_2) \). It chooses random exponents \( r_1, r_2 \in \mathbb{Z}_p \) and outputs a new private key \( SK_{ID} = (K_0 = K'_0, L_{0, 1, 1, 1, 1, 1} = K'_1 \cdot L_{1, 1, 2, 2}, L_{1, 2} = K'_1 \cdot L_{1, 1, 2, 2}) \).

AIBE.ChangeKey(SK_{ID}, \delta, RK_{ID}, PP): Let \( SK_{ID} = (K'_0, K'_1, K'_2) \). It sets \( TK = (K_0 = K'_0, \hat{g}^\delta, K_1 = K'_1, K_2 = K'_2) \). 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 \( CH = (C_0 = g', C_1 = (u_t^{r_1}h_1)^{r_1}, C_2 = w') \) and a session key \( EK = N' \).

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 \( EK = \prod_{i=0}^{3} e(C_i, K_i) \).

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 \( GDS = ((p, G, \hat{G}, \mathbb{G}_t, e), g, \hat{g}) \) be the group description string of a bilinear group with a generator \( g \in \mathbb{G} \). It selects random exponents \( u_0, h_0 \in \mathbb{Z}_q \) and sets \( u_2 = g^{h_0}, h_2 = g^{h_0}, \hat{u}_2 = \hat{g}^{h_0}, \hat{h}_2 = \hat{g}^{h_0} \). It chooses a random exponent \( \beta \in \mathbb{Z}_p \) and outputs a master key \( MK = \hat{g}^\beta \) and public parameters \( PP = (GDS, u_2, h_2, \hat{u}_2, \hat{h}_2, \mathbb{G}_t, e(g, \hat{g})^\beta) \).

IBE.GenKey(T, MK, PP): It chooses a random exponent \( r \in \mathbb{Z}_p \) and outputs a private key \( SK_T = (U_0 = MK(\hat{a}_1^{r}h_2)^{r_1}, U_1 = \hat{g}^{-r}) \).

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 \( SK_T = (U_0 = U'_0 \cdot (\hat{a}_1^{r}h_2)^{r_1}, U_1 = U'_1 \cdot \hat{g}^{-r}) \).

IBE.ChangeKey(SK_T, \delta, 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 \( CH_T = (C_0 = g', C_1 = (u_t^{r_1}h_2)^{r_1}) \) and a session key \( EK = N' \).

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 [5][38]. For ciphertext anonymity, we use an AIBE scheme instead of the first IBE scheme. For revocation privacy, we encrypt each node 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 [13][29]. Additionally, we randomly mix all node update keys in the update key.

Let PRF be a pseudo-random function for \( K = \{0, 1\}^k \), \( X = \{0, 1\}^* \), and \( Y = \mathbb{Z}_p^2 \times \{0, 1\}^k \). 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 \( T = \mathbb{Z}_p \), \( T = \mathbb{Z}_p \), and \( M = G_T \) is described as follows:

**RIBE-AR.Setup** \((1^k, N)\): Let \( \lambda \) 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 \( GDS = ((p, \mathbb{G}, \hat{\mathbb{G}}, G_T, e), g, \hat{g}) \).
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 \( B_T \) 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 \( ST = (B_T, z) \), and public parameters \( PP = (PP_{AIBE}, PP_{IBE}, \Omega = e(g, \hat{g})^\alpha) \).

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

1. It assigns \( ID \) to a random leaf node \( v \in B_T \) and obtains a private set \( PV = \{v_0, \ldots, v_n\} \) by running \( CS.Assign(B_T, v) \).
2. For \( 0 \leq j \leq n \), it computes \( (\gamma_j, \omega_j, \kappa_j) = PRF(z, Label(v_j)) \) 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 \( NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j) \).
3. Finally, it sets \( RK_{ID} = RK_{AIBE,0} \) and outputs a private key \( SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{ID}) \).

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

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 \( CS.Cover(B_T, R) \). 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 \( (\gamma_i, \omega_i, \kappa_i) = PRF(z, Label(v_i)) \) and proceeds as follows: It obtains \( SK_{IBE,i} \) by running \( IBE.GenKey(T, \hat{g}^{s\alpha-\gamma_i}, PP_{IBE}) \). It computes \( CU_i = SKE.Encrypt(\kappa_i, SK_{IBE,i}) \). It creates a node update key \( NUK_i = (CU_i, Y_i) \) 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 \( CU_i = SKE.Encrypt(\tilde{K}_i, \tilde{SK}_{IBE,i}) \) by using a random \( \tilde{K}_i \). It creates a node update key \( NUK_i = (CU_i, \tilde{Y}_i) \) 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 \( UK_T = (NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V) \).

**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 \( \omega_j \) 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 $j \in \{0, \ldots, n\}$ and $i \in \{1, \ldots, \ell\}$ 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 $SK_{IBE,i} = \text{SKE.Decrypt}(\kappa_j, CU_i)$.

3. It selects a random exponent $\delta \in \mathbb{Z}_p$. It obtains $SK_{AIBE}$ by running $\text{AIBE.ChangeKey}(SK_{AIBE,j}, \delta, RK_{ID}, PP_{AIBE})$. It also obtains $SK_{IBE}$ by running $\text{IBE.ChangeKey}(SK_{IBE,i}, -\delta, PP_{IBE})$. Finally, it outputs a decryption key $DK_{ID,T} = (SK_{AIBE}, SK_{IBE})$.

\textbf{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 $\text{AIBE.Encaps}(ID, t, PP_{AIBE})$. Next it obtains $CH_{IBE}$ and $EK_{IBE}$ by running $\text{IBE.Encaps}(T, t, PP_{IBE})$. It outputs a ciphertext $CT = (CH_{AIBE}, CH_{IBE}, C = \Omega \cdot M)$.

\textbf{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 $\text{AIBE.Decaps}(CH_{AIBE}, SK_{AIBE}, PP_{AIBE})$ and $\text{IBE.Decaps}(CH_{IBE}, SK_{IBE}, PP_{IBE})$ respectively. It outputs a decrypted message $M = C \cdot (EK_{AIBE} \cdot EK_{IBE})^{-1}$.

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

\subsection{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 $SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{ID})$ for $PV$ and $UK_T = (NUK_1, \ldots, NUK_m, V)$ for $CV$ where $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$ and $NUK_i = (CU_i, Y_i = V^\omega)$. If ID $\not\in R$, 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 $\omega_k \in NSK_k = (SK_{AIBE,k}, \omega_h, \kappa_h)$ and $Y_k \in NUK_k = (CU_k, Y_k)$. By decrypting $CU_k$ with a valid key $\kappa_k$, $SK_{IBE,k}$ is derived. From the \textbf{RIBE-AR.GenKey} and \textbf{RIBE-AR.UpdateKey} algorithms, the master key parts of $SK_{AIBE,k}$ and $SK_{IBE,k}$ are associated with $\gamma_k$ and $\alpha - \gamma_k$ respectively. The master key part of $SK_{AIBE}$ derived from \textbf{AIBE.ChangeKey} still associated with $\gamma_k + \delta$ and the master key part of $SK_{IBE}$ derived from \textbf{IBE.ChangeKey} still associated with $\alpha - \gamma_k - \delta$. Thus the \textbf{RIBE-AR.DeriveKey} algorithm is correct since we have $\alpha$ 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 \textbf{RIBE-AR.Decrypt} algorithm can be shown by the correctness of the \textbf{AIBE.Decaps} and \textbf{IBE.Decaps}. That is, we have $e(g, g)^{\gamma_k + \delta}$ from the correctness of AIBE and $e(g, g)^{\alpha - \gamma_k - \delta}$ from the correctness of IBE. Thus, the message $M$ can be easily obtained by using these session keys.

\subsection{4.6 Discussion}

\textbf{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\cite{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 A consists of the sequence of hybrid games. We define the games as follows:

Game G_0. This game is the original security game. That is, a simulator B obtains (γ_i, ω_i, κ_i) for a node v_i by running PRF.

Game G_1. This game G_1 is similar to the game G_0 except that the PRF is replaced by a truly random function. That is, B selects fixed random (γ_i, ω_i, κ_i) for a node v_i if v_i is used for the generation of a private key (or an update key).
Game $G_2$. This final game $G_2$ is similar to the game $G_1$ except the generation of the challenge ciphertext $CT^*$. Let $CT^* = (CH_{ABE}^*, CH_{IBE}^*, C^*)$ be the challenge ciphertext. In this game $G_2$, $C^*$ is replaced by a random element in $G_T$. Note that the advantage of $A$ in this game is zero since the challenge ciphertext is not related to $\mu$.

Let $Adv_{\mathcal{A}}^{G_2}$ be the advantage of $\mathcal{A}$ in a game $G_i$. Let $E_\tau$ be the event that the adversary behaves like $\tau$-type. From the Lemmas 5.5 and 5.6, we obtain the following equation

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

This completes our proof. 

Lemma 5.5. If the PRF scheme is secure, then no PPT type-$\tau$ adversary can distinguish between $G_0$ and $G_1$ 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-$\tau$ adversary can distinguish between $G_1$ and $G_2$ with a non-negligible advantage.

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

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

Setup: $B$ proceeds as follows:

1. It submits $ID^*$ to $B_{AIBE}$ and receives $PP_{AIBE}$. It also submits $T^*$ to $B_{IBE}$ and receives $PP_{IBE}$.

2. It obtains $BT$ by running $CS.Setup(N)$. It sets $RL$ as an empty one and sets $ST = (BT)$. It fixes a random leaf node $v^* \in BT$ that will be assigned to $ID^*$.

3. If $\tau = 1$, it sets $ChalPath = \emptyset$. Otherwise ($\tau = 2$), it sets $ChalPath = Path(v^*)$.

4. It publishes public parameters $PP = (PP_{AIBE}, PP_{IBE}, \Omega = e(g^a, \hat{g}^b))$.

Phase 1: $A$ 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 $B$ proceeds as follows:

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

  1. It queries an AIBE private key for $ID \neq ID^*$ to $B_{AIBE}$ 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 $CS.Assign(BT, v)$.

  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 $SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{AIBE, ID})$. 

17
• Case $\tau = 2 \land ID \neq ID^*$: In this case, we have $PV \cap \text{ChalPath} \neq \emptyset$.

1. It queries an AIBE private key for $ID \neq ID^*$ to $B_{AIBE}$ 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 $CS.Assign(B_T,v)$.
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 $v_j \in \text{ChalPath}$, 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 ($v_j \notin \text{ChalPath}$), 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 $SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{AIBE,ID})$.

• Case $\tau = 2 \land ID = ID^*$: In this case, we have $PV = \text{ChalPath}$.

1. It retrieves the leaf node $v^*$ and obtains $PV = \{v_0, \ldots, v_n\}$ by running $CS.Assign(B_T,v^*)$.
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 $SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{AIBE,0})$.

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

• Case $\tau = 1$: In this case, we have $\text{ChalPath} = \emptyset$.

1. It defines a revoked set $R$ on time $T$ from $RL$ and obtains $CV = \{v_1, \ldots, v_\ell\}$ by running $CS.Cover(B_T,R)$. 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 $\hat{S}K_{IBE,i}$ by selecting random elements and computes $CU_i = SKE.Encrypt(\hat{\kappa}_i, \hat{S}K_{IBE,i})$ by using a random key $\hat{\kappa}_i$. It selects a random $\hat{Y}_i$ and creates $NUK_i = (CU_i, \hat{Y}_i)$.
4. It creates $UK_T = (NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V)$ where $\pi$ is a random permutation.

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

1. It queries an IBE private key for $T \neq T^*$ to $B_{IBE}$ and receives $SK_{IBE,T}$.
2. It defines a revoked set $R$ on time $T$ from $RL$ obtains $CV = \{v_1, \ldots, v_\ell\}$ by running $CS.Cover(B_T,R)$. 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 $v_i \in \text{ChalPath}$, then it obtains $SK_{IBE,i}$ by running $IBE.ChangeKey(SK_{IBE,T}, \hat{g}^{-\gamma_i}, PP_{IBE})$.
   (b) Otherwise ($v_i \notin \text{ChalPath}$), it obtains $SK_{IBE,i}$ by running $IBE.GenKey(T, \hat{g}^{\omega_i}, PP_{IBE})$.
   (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 $\hat{S}K_{IBE,i}$ by selecting random elements and computes $CU_i = \text{SKE.Encrypt}(\hat{k}_i, \hat{S}K_{IBE,i})$ by using a random key $\hat{k}_i$. It selects a random $\hat{Y}_i$ and creates $NUK_i = (CU_i, \hat{Y}_i)$.

5. It creates $UK_T = (NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V)$ where $\pi$ is a random permutation.

- **Case** $\tau = 2$ and $T = T^*$: In this case, we have $CV \cap \text{ChalPath} = \emptyset$ 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 $\text{CS.Cover}(BT, R^*)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$. It sets $V = g^\delta$ by selecting a random $s \in \mathbb{Z}_p$.

  2. For $1 \leq i \leq \ell$, it retrieves $(y_i, \omega_i, \kappa_i)$ of the node $v_i$ and performs the steps: It obtains $SK_{IBE,i}$ by running $\text{IBE.GenKey}(T, g^\delta, PP_{IBE})$. It computes $CU_i = \text{SKE.Encrypt}(\kappa_i, SK_{IBE,i})$ 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 $\hat{S}K_{IBE,i}$ by selecting random elements and computes $CU_i = \text{SKE.Encrypt}(\hat{k}_i, \hat{S}K_{IBE,i})$ by using a random key $\hat{k}_i$. It selects a random $\hat{Y}_i$ and creates $NUK_i = (CU_i, \hat{Y}_i)$.

  4. It creates $UK_T = (NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V)$ where $\pi$ is a random permutation.

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

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

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

**Challenge:** $A$ submits two challenge messages $M^*_0, M^*_1$. $B$ flips a random bit $\mu \in \{0, 1\}$ and proceeds as follows:

1. It submits $M^*_\mu, M^*_1$ to $B_{AIBE}$ and receives a challenge $CH^*_A, EK^*_A$. It also submits $M^*_0, M^*_1$ to $B_{IBE}$ and receives a challenge $CH^*_B, EK^*_B$.

2. It creates $CT^* = (CH^*_A, CH^*_B, C^*, M^*_\mu)$ and gives it to $A$.

**Phase 2:** Same as Phase 1.

**Guess:** Finally, $A$ outputs a guess $\mu' \in \{0, 1\}$. $B$ 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 $I^*D_\mu$ 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 $I^*_0, I^*_1$ 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 $I^*_0$ and $I^*_1$.

**Type-2.** An adversary is Type-2 if it does not request a private key for $I^*_0$, but it requests a private key for $I^*_1$.

**Type-3.** An adversary is Type-3 if it requests a private key for $I^*_0$, but it does not request a private key for $I^*_1$.

**Type-4.** An adversary is Type-4 if it requests private keys for $I^*_0$ and $I^*_1$.

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

**Game $G_0$.** This game is the original security game. That is, a simulator $B$ obtains $(\gamma_i, \omega_i, \kappa_i)$ for a node $v_i$ by running PRF.

**Game $G_1$.** This game $G_1$ is similar to the game $G_0$ except that the PRF is replaced by a truly random function. That is, $B$ 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 $G_2$.** This game $G_2$ is similar to the game $G_1$ except the generation of the challenge ciphertext $CT^*$. Let $CT^* = (CH^*_{AIBE}, CH^*_{IBE}, C^*)$ be the challenge ciphertext. In this game $G_2$, $C^*$ is replaced by a random element in $G_T$.

**Game $G_3$.** This final game $G_3$ is similar to the game $G_2$ 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 $G$. Note that the advantage of $A$ in this game is zero since the challenge ciphertext is not related to $\mu$.

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

$$\text{Adv}_{\mathcal{A}}^{G_0} \leq 4 \sum_{\tau=1}^4 \Pr[E_\tau] \cdot |\text{Adv}_{\mathcal{A}}^{G_0} - \text{Adv}_{\mathcal{A}}^{G_2}| \leq 4\text{Adv}_{R}^{\text{PRF}}(\lambda) + 4\text{Adv}_{R}^{\text{PBDH}}(\lambda) + 4\text{Adv}_{R}^{\text{3DH}}(\lambda).$$

This completes our proof. \qed
Lemma 5.8. If the PRF scheme is secure, then no PPT type-\(\tau\) adversary can distinguish between \(G_0\) and \(G_1\) 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-\(\tau\) adversary can distinguish between \(G_1\) and \(G_2\) 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-\(\tau\) adversary can distinguish between \(G_2\) and \(G_3\) with a non-negligible advantage.

Proof. Suppose there exists an adversary \(A\) that attacks the above RIBE-AR scheme with a non-negligible advantage. A meta-simulator \(B\) that solves the A3DH assumption using \(A\) 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 \(B_{AIBE}\) be the simulator for IBE and \(B_{IBE}\) be the simulator for IBE. Then \(B\) that interacts with \(A\) is described as follows:

Init: \(A\) initially submits challenge identities \(ID_{0}^*\), \(ID_{1}^*\) and challenge time \(T^*\). \(B\) runs \(B_{AIBE}\) by giving \(D_{PBDH}\) and \(Z\), and runs \(B_{IBE}\) by giving \(D_{DBDH}\) and \(Z\).

Setup: \(B\) proceeds as follows:

1. It submits \(ID_{0}^*, ID_{1}^*\) to \(B_{AIBE}\) and receives \(PP_{AIBE}\). It also submits \(T^*\) to \(B_{IBE}\) and receives \(PP_{IBE}\).

2. It obtains \(BT\) by running \(CS.Setup(N)\). It initializes \(UL\) as an empty one. It sets \(RL\) as an empty one and sets \(ST = (BT, UL)\). It fixes random leaf nodes \(v_0^*, v_1^* \in BT\) that will be assigned to \(ID_{0}^*, ID_{1}^*\), respectively.

3. If \(\tau = 1\), it sets \(ChalPath = \emptyset\). If \(\tau = 2\), it sets \(ChalPath = Path(v_0^*)\). If \(\tau = 3\), it sets \(ChalPath = Path(v_0^*) \cup Path(v_1^*)\).

4. It publishes public parameters \(PP = (PP_{AIBE}, PP_{IBE}, \Omega = e(g^a, \hat{g}^b))\).

Phase 1: \(A\) 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 \(B\) 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 \(B_{AIBE}\) 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 \(CS.Assign(BT, v)\).
  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 \(SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{AIBE, ID})\).

- **Case** \((\tau = 2 \land ID \neq ID_{b}^*) \lor (\tau = 3 \land ID \neq ID_{b}^*) \lor (\tau = 4 \land ID \neq ID_{b}^*)\): 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 $ID \neq ID_j$ to $B_{AIBE}$ and receives $SK_{AIBE,ID}$ and $RK_{AIBE,ID}$.
2. It assigns $ID$ to a new leaf node $v \neq v_h$ and obtains $PV = \{v_0, \ldots, v_n\}$ by running $CS.Assign(BT, v)$.
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 $AIBE.GenKey(ID, \hat{g}_j, MP_{AIBE}, PP_{AIBE})$.
   (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 $SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{AIBE,ID})$.

- **Case** $(\tau = 2 \land ID = ID_1)$ or $(\tau = 3 \land ID = ID_0)$ or $(\tau = 4 \land (ID = ID_0 \lor ID = ID_1))$: In this case, we have $PV \subseteq ChalPath$.
  1. It retrieves the leaf node $v_0$ of $ID$ and obtains $PV = \{v_0, \ldots, v_n\}$ by running $CS.Assign(BT, v_0)$.
  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 $AIBE.GenKey(ID, \hat{g}_j, MP_{AIBE}, PP_{AIBE})$. It sets $NSK_j = (SK_{AIBE,j}, \omega_j, \kappa_j)$.
  3. It creates $SK_{ID} = (NSK_0, \ldots, NSK_n, RK_{AIBE,0})$.

If this is an update key query for time $T$, then $B$ 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 $CV = \{v_1, \ldots, v_l\}$ by running $CS.Cover(BT, R)$. 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 $IBE.GenKey(T, \hat{g}_i, PP_{IBE})$. 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 $SK_{IBE,i}$ by selecting random elements and computes $CU_i = SKE.Encrypt(\tilde{\kappa}_i, SK_{IBE,i})$ 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 $UK_T = (NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V)$ where $\pi$ is a random permutation.

- **Case** $(\tau = 2 \land T \neq T^*)$ or $(\tau = 3 \land T \neq T^*)$ or $(\tau = 4 \land T \neq T^*)$: 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 $B_{IBE}$ and receives $SK_{IBE,T}$.
  2. It defines a revoked set $R$ on time $T$ from $RL$ and obtains $CV = \{v_1, \ldots, v_l\}$ by running $CS.Cover(BT, R)$. 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 $IBE.GenKey(T, \hat{g}_i, PP_{IBE})$.
     (c) It computes $CU_i = SKE.Encrypt(\kappa_i, SK_{IBE,i})$ and sets $NUK_i = (CU_i, Y_i = V^{\omega_i})$. 

22
4. For \(\ell + 1 \leq i \leq \ell_m\), it performs the steps: It sets a random \(\hat{\SK}_{IBE,i}\) by selecting random elements and computes \(CU_i = \text{SKE.Encrypt}(\kappa_i, \hat{\SK}_{IBE,i})\) by using a random key \(\kappa_i\). It selects a random \(\hat{\gamma}_i\) and creates \(\text{NUK}_i = (CU_i, \hat{\gamma}_i)\).

5. It creates \(UK_T = (\text{NUK}_{\pi(1)}, \ldots, \text{NUK}_{\pi(\ell_m)}, V)\) where \(\pi\) is a random permutation.

- **Case** \((\tau = 2 \land T = T^*) \lor (\tau = 3 \land T = T^*) \lor (\tau = 4 \land T = T^*)\): In this case, we have \(CV \cap \text{ChalPath} = \emptyset\)

  1. It defines a revoked set \(R^*\) on time \(T^*\) from \(RL\) and obtains \(CV = \{v_1, \ldots, v_\ell\}\) by running \(\text{CS.Cover}(BT, R^*)\). Let \(r = |R|, \ell = |CV|, \text{and} \ell_m = \lceil r \log(N/r) \rceil\). It sets \(V = \tilde{g}^T\) 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 \(\text{IBE.GenKey}(T, \tilde{g}^{\gamma_i}, PP_{IBE})\). It computes \(CU_i = \text{SKE.Encrypt}(\kappa_i, \SK_{IBE,i})\) and sets \(\text{NUK}_i = (CU_i, \gamma_i)\).

4. For \(\ell + 1 \leq i \leq \ell_m\), it performs the steps: It sets a random \(\hat{\SK}_{IBE,i}\) by selecting random elements and computes \(CU_i = \text{SKE.Encrypt}(\kappa_i, \hat{\SK}_{IBE,i})\) by using a random key \(\kappa_i\). It selects a random \(\hat{\gamma}_i\) and creates \(\text{NUK}_i = (CU_i, \hat{\gamma}_i)\).

4. It creates \(UK_T = (\text{NUK}_{\pi(1)}, \ldots, \text{NUK}_{\pi(\ell_m)}, V)\) where \(\pi\) is a random permutation.

If this is a decryption key query for an identity \(ID\) and time \(T\), then \(B\) proceeds as follows:

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

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

**Challenge:** \(A\) submits a challenge message \(M^*\). \(B\) proceeds as follows:

1. It submits \(M^*\) to \(B_{AIBE}\) and receives a challenge \(CH_{AIBE}^*, EK_{AIBE}^*\). It also submits \(M^*\) to \(B_{IBE}\) and receives a challenge \(CH_{IBE}^*, EK_{IBE}^*\).

2. It creates \(CT^* = (CH_{AIBE}^*, CH_{IBE}^*, e(\tilde{g}, \tilde{g})^d \cdot M^*)\) by selecting a random \(d\) and gives it to \(A\).

**Phase 2:** Same as Phase 1.

**Guess:** Finally, \(A\) outputs a guess \(\mu' \in \{0, 1\}\). \(B\) also outputs \(\mu'\). 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 \( R^*_\mu \) 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 \( G_0, G_1, G_2 \). 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 \( G_0 \).** This game is the original security game. That is, a simulator \( B \) obtains \((\gamma_i, \omega_i, \kappa_i)\) for a node \( v_i \) by running PRF.

**Game \( G_1 \).** This game \( G_1 \) is similar to the game \( G_0 \) except that the PRF is replaced by a truly random function. That is, \( B \) 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 \( G_2 \).** This final game \( G_2 \) is similar to the game \( G_1 \) except that generation of the challenge update key \( UK^* \). Let \( UK^* = (NUK_1^*, ..., NUK_{k}^*, V^*) \) 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 \( \mathbb{G} \). Note that the advantage of \( A \) in this game is zero since the challenge update key is not related to \( \mu \).

To argue that the adversary cannot distinguish \( G_1 \) from \( G_2 \), we also define a sequence of hybrid games \( G_{1,0} = G_0, ..., G_{1,k-1}, G_{1,k} = G_2 \) which are defined as follows:

**Game \( G_{1,0} \).** This game is equal to the game \( G_1 \). That is, all challenge node update keys in the challenge update key are generated normally.

**Game \( G_{1,k} \).** In this game \( G_{1,k} \), each challenge node update key \( NUK_i^* \) for \( 1 \leq i \leq k \) is generated randomly, but each challenge node update key \( NUK_i^* \) for \( k + 1 \leq i \) is generated normally.

**Game \( G_{1,\ell} \).** This game \( G_{1,\ell} \) is equal to the game \( G_2 \). That is, all challenge node update keys in the challenge update key are generated randomly.

To argue the indistinguishability of \( G_{1,k-1} \) and \( G_{1,k} \), we additionally define a sequence of hybrid games \( H_{k,0}, H_{k,1}, H_{k,2}, H_{k,3} \). 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 \( H_{k,0} \).** This game is equal to the game \( G_{1,k-1} \). That is, \( CU_k \) and \( Y_k \) are generated normally.

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

**Game \( H_{k,2} \).** In this game \( H_{k,2} \), \( CU_k \) is generated by encrypting random elements with a random key, but \( Y_k \) is generated normally. That is, \( CU_k = \text{SKE.Encrypt}(\tilde{\kappa}_k, (\hat{U}_{k,0}, \hat{U}_{k,1})) \) by selecting random \( \hat{U}_{k,0}, \hat{U}_{k,1} \) and a random key \( \tilde{\kappa}_k \).
Game $H_{k,3}$. This game $H_{k,3}$ is equal to the game $G_{1,k}$. In this game, $CU_{j}$ 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 $\text{Adv}_{A}^{G_{j}}$ be the advantage of $A$ in the game $G_{j}$. We have that $\text{Adv}_{\text{RIBE-AR},A}^{\text{SE-REV-PPV}}(\lambda) = \text{Adv}_{A}^{G_{0}}$ and $\text{Adv}_{A}^{G_{2}} = 0$. From the following Lemmas 5.12, 5.13, 5.14, and 5.15, we obtain the equation

$$\text{Adv}_{A}^{G_{0}}(\lambda) \leq \sum_{j=1}^{2} |\text{Adv}_{A}^{G_{j-1}} - \text{Adv}_{A}^{G_{j}}| + \text{Adv}_{A}^{G_{2}}$$

$$\leq |\text{Adv}_{A}^{G_{0}} - \text{Adv}_{A}^{G_{1}}| + \sum_{k=1}^{\ell} |\text{Adv}_{A}^{G_{k-1}} - \text{Adv}_{A}^{G_{k}}|$$

$$\leq |\text{Adv}_{A}^{G_{0}} - \text{Adv}_{A}^{G_{1}}| + \sum_{k=1}^{\ell} \sum_{j=1}^{3} |\text{Adv}_{A}^{H_{k-1}} - \text{Adv}_{A}^{H_{k,j}}|$$

$$\leq \text{Adv}_{B}^{\text{PRF}}(\lambda) + r \log N \left( \text{Adv}_{SKE,B}^{\text{IND-CPA}}(\lambda) + \text{Adv}_{SKE,B}^{\text{KEY-PPV}}(\lambda) + \text{Adv}_{\text{XDH}}^{\text{IND-CPA}}(\lambda) \right)$$

where $r$ is the maximum number of revoked users in an update key. This completes the proof. \hfill $\square$

**Lemma 5.12.** If the PRF is secure, then no PPT adversary can distinguish $G_{0}$ from $G_{1}$ 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 $H_{k,0}$ from $H_{k,1}$ with a non-negligible advantage.

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

**Init:** $A$ initially submits challenge revoked sets $R_{\mu}^{*}, R_{1}$ and challenge time $T^{*}$. $B$ chooses a random bit $\mu \in \{0, 1\}$ and obtains a challenge cover set $CV^{*} = \{v_{1}, \ldots, v_{k}, \ldots, v_{l}\}$ by running $\text{CS.Cover}(BT, R_{\mu}^{*})$. Let $r = |R_{\mu}^{*}|$, $\ell = |CV^{*}|$, and $\ell_{m} = |r \log (N/r)|$.

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

**Phase 1:** $B$ 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 $\kappa_{k}$ 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:** $B$ creates a challenge update key $UK^{*}$ as follows:

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

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 $CU_{i} = \text{SKE.Encrypt}(\kappa_{i}, \tilde{SK}_{IBE,i})$. It creates $NUK^{*}_{i} = (CU_{i}, Y_{i} = (V^{*})^{\alpha_{i}})$.

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 $\text{IBE.GenKey}(T, g^{\alpha - \gamma}, PP_{IBE})$.

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

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 $\text{IBE.GenKey}(T, g^{\alpha - \gamma}, PP_{IBE})$. It computes $CU_i = \text{SKE.Encrypt}(\kappa_i, SK_{IBE,i})$. It creates a normal $NUK_i^* = (CU_i, Y_i = (V^*)^{\omega_i})$.

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 $CU_i = \text{SKE.Encrypt}(\tilde{k}_i, \tilde{SK}_{IBE,i})$ by using a random key $\tilde{k}_i$. It creates a random $NUK_i^* = (CU_i, \tilde{Y}_i)$ by selecting a random $\tilde{Y}_i$.

6. It creates $UK^* = (NUK_1^*_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}^*, V^*)$ where $\pi$ is a random permutation.

Phase 2: Same as Phase 1.

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

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

**Lemma 5.14.** If the SKE scheme is KEY-PRV secure, then no PPT adversary can distinguish $H_{k,1}$ from $H_{k,2}$ 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 $A$ be an adversary that attacks the above RIBE-AR scheme with a non-negligible advantage and $B$ be a simulator that breaks the KEY-PRV security of an SKE scheme using $A$. 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 $NUK_k^*$ 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 $NUK_k^* = (CU_k = CT^*, Y_k = (V^*)^{\omega_k})$.

If a valid encryption key $k_{\kappa}$ is used in the KEY-PRV game, then $CU_k = \text{SKE.Encrypt}(k_{\kappa}, \tilde{SK}_{IBE,k})$ which is equal to the game $H_{k,1}$. Otherwise (a random encryption key $k_{\kappa}$ is used), $CU_k = \text{SKE.Encrypt}(\kappa_{\kappa}, \tilde{SK}_{IBE,k})$ which is equal to the game $H_{k,2}$. This completes our proof.

**Lemma 5.15.** If the XDH assumption holds, then no PPT adversary can distinguish $H_{k,2}$ from $H_{k,3}$ with a non-negligible advantage.
Challenge: Let $UK^*$ be the challenge update key that should be created in this step. $B$ 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 $1 \leq i \leq k - 1$, 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 $k + 1 \leq i \leq \ell$, it retrieves $(\gamma_i, \omega_i, \kappa_i)$ of the node $v_i$ and sets $Y_i = (\hat{g}^a)^{\omega_i}$.
4. For $\ell + 1 \leq i \leq \ell_m$, 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, $A$ outputs a guess $\mu' \in \{0, 1\}$. $B$ also outputs $\mu'$.

If a valid $Z = g^{ab}$ is given, then a valid hint $Y_k = \hat{g}^{ab} = (\hat{g}^a)^{\omega_k}$ is created like in the game $H_{k,1}$. Otherwise (a random $Z = g^s$ is given), a random hint $Y_k = \hat{g}^s$ is created like in the game $H_{k,2}$. 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


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 \( n, m, \bar{m}, q \) be positive integers with \( m \geq 2n \log q \) and \( q \) prime. There are polynomial time algorithms for sampling a short vector as follows:

- \( \text{GenTrap}(1^n, 1^m, q) \): It is a randomized algorithm that outputs a full rank matrix \( A \in \mathbb{Z}^{n \times m}_q \) and a trapdoor basis \( T_A \in \mathbb{Z}^{m \times \bar{m}}_q \) for \( \Lambda_q^+ (A) \) such that \( A \) is statistically close to uniform and \( \|T_A\|_{GS} = O(\sqrt{n \log q}) \) with overwhelming probability in \( n \) \([3, 32]\).

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

- \( \text{ExtendRandRight}(A, G, R, T_G, \sigma) \): It is a randomized algorithm that, given as input a full rank matrix \( A, G \in \mathbb{Z}^{n \times m}_q \), a matrix \( R \in \mathbb{Z}^{m \times m}_q \), a basis \( T_G \) of \( \Lambda_q^+ (G) \), and a Gaussian parameter \( \sigma \geq \|R\|_2 \cdot \|T_G\|_2 \cdot \omega (\sqrt{\log n}) \), outputs a matrix \( T_{[A|R\cdot G]} \in \mathbb{Z}^{2m \times 2m} \) distributed statistically close to \( D \) \([2]\).

There exists a fixed full rank matrix \( G \in \mathbb{Z}^{n \times m}_q \) such that the lattice \( \Lambda_q^+ (G) \) has a publicly known basis \( T_G \) with \( \|T_G\|_{GS} \leq \sqrt{5} \) \([32]\).

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

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

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

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

where \( A \in \mathbb{Z}_q^{n \times m}, s \in \mathbb{Z}_q^n, x \in \mathcal{D}, u \in \mathbb{Z}_q^m \) are independently sampled. The advantage of LWE is defined as

\[
\text{Adv}_A^{\text{LWE}} = |\Pr[A(A, A^\top s + x) = 1] - \Pr[A(A, u) = 1]|.
\]

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:

\[
(A, |A^\top s|_p) \quad \text{and} \quad (A, v)
\]

where \( A \in \mathbb{Z}_q^{n \times m}, s \in \mathbb{Z}_q^n, x \in \mathcal{D}, v \in \mathbb{Z}_p^m \) are independently sampled. The advantage of LWR is defined as

\[
\text{Adv}_A^{\text{LWR}} = |\Pr[A(A, |A^\top s|_p) = 1] - \Pr[A(A, v) = 1]|.
\]

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

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

1. It obtains $(A_1, T_{A_1})$ and $(A_2, T_{A_2})$ by running GenTrap$(1^n, 1^m, q)$ and GenTrap$(1^n, 1^m, q)$ respectively. It also samples uniformly random matrix $B \leftarrow \mathbb{Z}_q^{n \times m}$ and vector $u \leftarrow \mathbb{Z}_q^n$.
2. It generates two PRF keys $z_1, z_2$ and obtains $BT$ by running CS.Setup$(N)$.
3. Finally, it outputs a master key $MK = (T_{A_1}, T_{A_2}, z_1, z_2)$, a revocation list $RL = \emptyset$, a state $ST = (BT)$, and public parameters $PP = (A_1, A_2, B, u)$.

RIBE-AR.GenKey$(ID, MK, ST, PP)$: Let $MK = (T_{A_1}, T_{A_2}, z_1, z_2)$ and $ST = (BT)$.

1. It assigns $ID$ to a random leaf node $v \in BT$ and obtains a private set $PV = \{S_0, \ldots, S_n\}$ by running CS.Assign$(BT, v)$.
2. For $j = 0$ to $j = n$, it proceeds as follows: Let $L_j = \text{Label}(S_j)$ be a label string. It computes $(u_j, w_j) = \text{PRF}_1(z_1, L_j)$ and $\kappa_j = \text{PRF}_2(z_2, L_j)$ where $u_j, w_j \in \mathbb{Z}_q^n$. It samples a vector $e_j$ by running SampleLeft$(A_1, E(ID), u_j, T_{A_1}, \sigma)$ such that

$$[A_1 | E(ID)]^\top e_j = u_j.$$ 

Next, it creates a node private key $NSK_j = (e_j, w_j, \kappa_j)$.
3. It obtains an extended basis $T_{A_1 | E(ID)}$ by running ExtendRandLeft$(A_2, E(ID), T_{A_2}, \sigma)$.
4. Finally, it outputs a private key $SK_{ID} = (NSK_0, \ldots, NSK_n, T_{A_1 | E(ID)})$.

RIBE-AR.UpdateKey$(T, RL, MK, ST, PP)$: Let $MK = (T_{A_1}, T_{A_2}, z_1, z_2)$ and $ST = (BT)$.

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$(BT, R)$. Let $r = |R|$, $\ell = |CV|$, and $\ell_m = \lceil r \log(N/r) \rceil$.
2. It selects a uniformly random matrix $V \in \mathbb{Z}_q^{n \times n}$.
3. For $1 \leq i \leq \ell$, it proceeds as follows: Let $L_i = \text{Label}(S_i)$ be a label string. It computes $(u_j, w_j) = \text{PRF}_1(z_1, L_j)$ and $\kappa_j = \text{PRF}_2(z_2, L_j)$ where $u_j, w_j \in \mathbb{Z}_q^n$. It samples a vector $f_i$ by running SampleLeft$(A_1, F(T), u_i, T_{A_1}, \sigma)$ such that

$$[A_1 | F(T)]^\top f_i = u_i.$$ 

It obtains a ciphertext $CU_i = \text{SKE.Encrypt}(\kappa_i, f_i)$. Next, it sets a hint vector $y_i = [V^\top w_i]_p$ and creates a node update key $NUK_i = (CU_i, y_i)$.
4. For $\ell + 1 \leq i \leq \ell_m$, it proceeds as follows: It selects random $f_i$ and obtains $CU_i = \text{SKE.Encrypt}(\kappa_i, f_i)$ by using a random key $\kappa_i$. It creates a node update key $NUK_i = (CU_i, y_i)$.
5. Finally, it selects a random permutation $\pi : \{1, \ldots, \ell_m\} \to \{1, \ldots, \ell_m\}$ and outputs an update key $UK_T = (NUK_{\pi(1)}, \ldots, NUK_{\pi(\ell_m)}, V)$.

RIBE-AR.DeriveKey$(ID, T, SK_{ID}, UK_T, PP)$: Let $SK_{ID} = (NSK_0, \ldots, NSK_n, T_{A_1 | E(ID)})$ and $UK_T = (NUK_1, \ldots, NUK_{\ell_m}, V)$.
1. It retrieves \( w_j \) from \( \text{NSK}_j \) for all \( j \in \{0,\ldots,n\} \) and sets a list \( (y'_0 = [V^\top w_0]_p,\ldots,y'_n = [V^\top w_n]_p) \). It also retrieves \( y_i \) from \( \text{NUK}_i \) for all \( i \in \{1,\ldots,\ell_m\} \) and sets a list \( (y_1,\ldots,y_{\ell_m}) \).

2. It finds two indexes \( j \in \{0,\ldots,n\} \) and \( i \in \{1,\ldots,\ell_m\} \) such that \( y'_j = y_i \) in the lists, then it retrieves the corresponding \( \text{NSK}_j = (e_j, w_j, \kappa_j) \) and \( \text{NUK}_i = (C_{U_i}, y_i) \). Next, it decrypts \( f_i = \text{SKE.Decrypt}(\kappa_j, C_{U_i}) \).

3. It parses \( e_j = [e_{j,L}, e_{j,R}] \) and \( f_i = [f_{i,L}, f_{i,R}] \) where \( e_{j,L}, f_{i,L}, e_{j,R}, f_{i,R} \in \mathbb{Z}^m \). Then it computes \( d_1 = [e_{j,L} + f_{i,L}, e_{j,R} f_{i,R}] \) such that

\[
A_1 [E(ID)] [F(T)]^\top d_1 = u.
\]

4. It also samples \( d_2 \) by running \( \text{SampleLeft}([A_2|E(ID)|F(T), u, T_{A_2|E(ID)|}, \sigma]) \) such that

\[
A_2 [E(ID)] [F(T)]^\top d_2 = u.
\]

5. Finally, it outputs a decryption key \( DK_{ID,T} = (d_1, d_2) \).

**RIBE-AR.Encrypt** \((ID, T, M, PP)\): It first samples uniformly random vectors \( s_1, s_2 \in \mathbb{Z}_q^n \). It also samples \( x \leftarrow D_{\mathbb{Z}, a_q}, x_1, x_2 \leftarrow D_{\mathbb{Z}_q, a_q} \) and sets

\[
c_0 = u^\top (s_1 + s_2) + x + M[q/2],
\]

\[
c_1 = [A_1|E(ID)|F(T)]^\top s_1 + x_1,
\]

\[
c_2 = [A_2|E(ID)|F(T)]^\top s_2 + x_2.
\]

Finally, it outputs a ciphertext \( CT = (c_0, c_1, c_2) \).

**RIBE-AR.Decrypt** \((CT, DK_{ID,T}, PP)\): Let \( CT = (c_0, c_1, c_2) \) and \( DK_{ID,T} = (d_1, d_2) \). From the ciphertext and the decryption key, it computes

\[
c' = c_0 - c_1 d_1 - c_2 d_2.
\]

It outputs 1 if \(|c' - [q/2]| < [q/4]| \) and 0 otherwise.

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