# A Generic Approach to Build Revocable Hierarchical Identity-Based Encryption

Kwangsu Lee[*] Joon Sik Kim[†]

**Abstract**

Revocable hierarchical identity-based encryption (RHIBE) is an extension of HIBE that provides the efficient key revocation function by broadcasting an update key per each time period. Many RHIBE schemes have been proposed by combining an HIBE scheme and the tree-based revocation method, but a generic method for constructing an RHIBE scheme has not been proposed. In this paper, we show for the first time that it is possible to construct RHIBE schemes by generically combining underlying cryptographic primitives and tree-based revocation methods. We first generically construct an RHIBE-CS scheme by combining HIBE scheme and the complete subtree (CS) method, and prove the adaptive security of this scheme by using the adaptive security of the HIBE schemes. Next, we generically construct an RHIBE-SD scheme by combining HIBE and hierarchical single revocation encryption (HSRE) schemes, and the subset difference (SD) method to reduce the size of an update key. Finally, we generically construct an RHIBE-CS scheme with shorter ciphertexts by combining HIBE schemes with constant-size ciphertext and the CS method.

**Keywords:** Hierarchical identity-based encryption, Key revocation, Subset cover, Generic construction.

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

[†]Korea University, Seoul, Korea.

## Contents

## 1 Introduction

Hierarchical identity-based encryption (HIBE) is an extension of IBE that represents an identity as a hierarchical identity vector and reduces the load of a key generation center by providing the key delegation function [19]. The HIBE scheme is suitable for use in an organization with a hierarchical structure, and can be applied to various fields such as identity-based signature, forward-secure encryption, chosen-ciphertext secure encryption, and public-key broadcast encryption [7, 8, 13, 18]. An important extension of HIBE is revocable HIBE (RHIBE) that supports efficient key revocation of users. In order to add the key revocation function to an HIBE scheme, Seo and Emura [34] proposed the first RHIBE scheme by following the design principle of the revocable IBE scheme of Boldyreva et al. [3] that associates the path of a binary tree with a private key and associates the cover of the binary tree with an update key. After the first RHIBE scheme, various RHIBE schemes with improved efficiency have been proposed and research in this field is actively conducted [15, 20, 21, 27, 33, 36].

Research on RHIBE that supports the key revocation is important because it is essential to efficiently revoke the private key of a user when the private key is exposed in a real environment. However, the existing RHIBE schemes have the disadvantage of having to modify the underlying HIBE scheme for building an RHIBE scheme, and perform a completely new analysis to prove the security of this RHIBE scheme. In the case of HIBE schemes from bilinear maps, the construction of RHIBE schemes is somewhat easier since the design methodology of RHIBE is already known [27, 34, 36], but in the case of the HIBE scheme from different mathematical structures, a completely different design methodology is required to build an RHIBE scheme. In order to overcome this problem, a generic design method of building an RHIBE scheme by using the underlying cryptographic schemes as black-box is required. In this paper, we ask whether it is possible to design an RHIBE scheme that supports efficient user key revocation by generically combining underlying cryptographic primitives.

### 1.1 Our Contributions

In this section, we show for the first time that it is possible to construct RHIBE schemes by generically combining underlying cryptographic schemes. The results of our RHIBE schemes are summarized as follows.

**RHIBE with Complete Subtree**. We first show that it is possible to design an RHIBE-CS scheme by generically combining HIBE schemes and the complete subtree (CS) method of a binary tree. We then prove the adaptive security of the proposed RHIBE-CS scheme based on the adaptive security of the HIBE schemes and the properties of the CS method. For the generic construction of an RHIBE-CS scheme, we extend the idea of Ma and Lin [30] used to design a generic RIBE scheme to an HIBE scheme. That is, since the hierarchical identity of HIBE has an identity for each level, each identity of the level is independently associated with a binary tree. At this time, a ciphertext is composed of HIBE ciphertexts associated with the path of the binary tree, and an update key is composed of the HIBE secret key associated with the cover of the binary tree. In our RHIBE-CS scheme, a ciphertext consists of approximately $O(\ell n)$ HIBE ciphertexts, a private key consists of only $O(1)$ HIBE private keys, and an update key consists of approximately $O(rn)$ HIBE private keys where $\ell$ is the level of hierarchical identity, $n$ is the depth of a binary tree, and $r$ is the number of revoked users.

**RHIBE with Subset Difference**. Next, we show that it is possible to design an RHIBE-SD scheme by generically combining an HIBE scheme, a hierarchical single revocation encryption (HSRE) scheme, and the SD method. The SD method is a type of the subset cover framework proposed by Naor et al. [31] to construct revocation and tracing schemes, and it enables to construct a cover set more efficiently compared to the CS method. We define HSRE in which the key delegation function was added to the SRE scheme which was introduced by Lee et al. [24] to build a public-key revocation scheme by using the SD method, and propose an HSRE scheme by combining HIBE and SRE schemes in bilinear groups. The design idea of the RHIBE-SD scheme is to associate a ciphertext with the path set of a binary tree and associate an update key with the cover set of a binary tree. In our RHIBE-SD scheme, a ciphertext consists of $O(\ell n^2)$ HSRE ciphertexts, a private key consists of $O(1)$ HIBE private keys and HSRE private keys, and an update key consists of $O(r)$ HSRE private keys.

**RHIBE with Shorter Ciphertexts**. Finally, we propose an RHIBE-CS scheme with shorter ciphertexts to reduce the ciphertext size of the previous RHIBE-CS scheme. The ciphertext of the RHIBE-CS scheme has a disadvantage of increasing the ciphertext size because the identity of each level is associated with the path of a binary tree, so the HIBE ciphertext is independently required for each path node. To solve this problem, we use a method of encoding and encrypting the information of the path nodes at once by using an HIBE scheme with a constant size ciphertext. In this case, it is possible to decrypt the ciphertext when a common node exists in the cover set and the path set by using the key delegation property of the HIBE scheme. The RHIBE-CS scheme with shorter ciphertexts has the advantage that the ciphertext size decreases compared to the previous RHIBE-CS scheme, but has the disadvantage that the update key size increases. In order to solve the problem of increasing update key size, we may consider that a cloud server stores all update keys and transmits the corresponding keys in update keys for each user.

## 1.2 Related Work

**IBE and Revocable IBE**. The concept of IBE was first introduced by Shamir [37] to solve the key management problem of existing public-key encryption and the first IBE scheme was proposed by Boneh and Franklin [6] by using bilinear maps. Since then, various IBE schemes have been proposed in bilinear groups [4, 16, 39], and a number of IBE schemes from different mathematical structures have also been proposed [12, 14, 17]. Revocable IBE (RIBE) is an extension of IBE that provides the revocation of private keys and the first efficient RIBE scheme that uses a binary tree was proposed by Boldyreva et al. [3]. After that, an RIBE scheme with adaptive security and an RIBE scheme that provides decryption key exposure resistance were proposed [29, 35]. To reduce the size of update keys, an RIBE scheme from multilinear maps was proposed [32]. Most of the previous RIBE schemes use the complete subtree (CS) method of a binary tree, but an RIBE scheme that reduces the size of update keys by using the subset difference (SD) method of a binary tree was proposed by Lee et al. [25]. In lattices, RIBE schemes that use binary trees have been proposed [10, 11, 20, 38]. Recently, Ma and Lin showed that an RIBE scheme can be constructed by generically combining HIBE and IBE schemes [30]. Lee also showed that a generic RIBE scheme that uses the SD method can be built by combining HIBE and single revocation encryption (SRE) schemes [22]. An RIBE scheme that delegates the generation of update keys to a cloud server and verifies the correctness of these update keys was also proposed [23].

**HIBE and Revocable HIBE**. Hierarchical IBE (HIBE) is an extension of IBE that represent an identity into a hierarchical structure in order to reduce the load of private key generation in IBE. Horwitz and Lin introduced the concept of HIBE and proposed a two-level HIBE scheme [19]. Since then, many HIBE schemes have been proposed in bilinear groups [4,5,18,28,40]. Different HIBE schemes have been proposed from lattices by using the basis delegation method of lattices [1, 2, 9]. The first revocable HIBE (RHIBE) scheme that provides the user key revocation function was proposed by Seo and Emura [34]. Similar to the RIBE scheme, they constructed an RHIBE scheme by using binary tree such that the path of a binary tree is associated with a private key and the cover of a binary tree is associated with an update key. Later, Seo and Emura proposed an efficient RHIBE scheme by using the history-free update method that reduces the size of private keys [36]. Lee and Park proposed RHIBE schemes with shorter private keys and update keys by carefully modifying the underlying HIBE scheme to support intermediate private keys [27]. Later, Lee proposed RHIBE schemes with adaptive security by using the dual system encryption method [21]. Recently, an adaptively secure RHIBE scheme under the standard assumption was proposed by Emura et al. [15].

## 2 Preliminaries

In this section we review the syntax and security model of the HIBE, HSRE, and RHIBE schemes.

### 2.1 Hierarchical Identity-Based Encryption

Hierarchical IBE (HIBE) is an extension of IBE that supports the identity of a user to have a hierarchical structure. The concept of HIBE was introduced by Horwitz and Lin [19] and an HIBE scheme supporting multiple levels was proposed by Gentry and Silverberg [18]. In an HIBE scheme, a private key is associated with a hierarchical identity $ID' = (I'_1, \dots, I'_k)$, and a ciphertext is associated with a hierarchical identity $ID = (I_1, \dots, I_\ell)$. If $ID' \in \text{Prefix}(ID)$ is established, a user having the corresponding private key can decrypt the ciphertext. The detailed syntax of HIBE is given as follows:

**Definition 2.1** (Hierarchical Identity-Based Encryption, HIBE)**.** An HIBE scheme consists of five algorithms **Setup**, **GenKey**, **Delegate**, **Encrypt**, and **Decrypt**, which are defined as follows:

**Setup**$(1^\lambda, L)$. The setup algorithm takes as input a security parameter $1^\lambda$ and maximum hierarchical depth $L$. It outputs a master key $MK$ and public parameters $PP$.

**GenKey**$(ID|_k, MK, PP)$. The key generation algorithm takes as input a hierarchical identity $ID|_k = (I_1, \dots, I_k)$, the master key $MK$, and the public parameters $PP$. It outputs a private key $SK_{ID|_k}$.

**Delegate**$(ID|_k, SK_{ID|_{k-1}}, PP)$. The delegation algorithm takes as input a hierarchical identity $ID|_k$, a private key $SK_{ID|_{k-1}}$ for $ID|_{k-1}$, and the public parameters $PP$. It outputs a delegated private key $SK_{ID|_k}$.

**Encrypt**$(ID|_\ell, M, PP)$. The encryption algorithm takes as input a hierarchical identity $ID|_\ell = (I_1, \dots, I_\ell)$, a message $M$, and public parameters $PP$. It outputs a ciphertext $CT_{ID|_\ell}$.

**Decrypt**$(CT_{ID|_\ell}, SK_{ID'|_k}, PP)$. The decryption algorithm takes as input a ciphertext $CT_{ID|_\ell}$, a private key $SK_{ID'_k}$, and public parameters $PP$. It outputs a message $M$ or $\perp$.

The correctness of HIBE is defined as follows: For all $MK, PP$ generated by **Setup**$(1^\lambda, L)$, all $ID|_\ell, ID'|_k$, any $SK_{ID'|_k}$ generated by **GenKey**$(ID'|_k, MK, PP)$, it is required that

- If $ID|_{k-1} \in \text{Prefix}(ID|_k)$, then **Delegate**$(ID|_k, SK_{ID|_{k-1}}, PP) = SK_{ID|_k}$.
- If $ID'|_k \in \text{Prefix}(ID|_\ell)$, then **Decrypt**(**Encrypt**$(ID|_\ell, M, PP), SK_{ID'|_k}, PP) = M$.

**Definition 2.2** (IND-CPA Security)**.** The security of HIBE is defined in terms of the indistinguishability under chosen plaintext attacks (IND-CPA). The security game is defined as the following game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Setup**: $\mathcal{C}$ runs **Setup**$(1^\lambda, L)$ to generate a master key $MK$ and public parameters $PP$. It keeps $MK$ to itself and gives $PP$ to $\mathcal{A}$.

2. **Query 1**: $\mathcal{A}$ adaptively requests private keys for hierarchical identities $ID_1, \ldots, ID_{q_1}$. In response, $\mathcal{C}$ gives the corresponding private keys $SK_1, \ldots, SK_{q_1}$ to $\mathcal{A}$ by running **GenKey**$(ID_i, MK, PP)$.

3. **Challenge**: $\mathcal{A}$ submits challenge labels $ID^*|_\ell$ and two messages $M_0^*, M_1^*$ with the equal length subject to the restriction: for all $ID_j$ of private key queries, it is required that $ID_j \notin \text{Prefix}(ID^*|_\ell)$. $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$ and gives the challenge ciphertext $CT^*$ to $\mathcal{A}$ by running **Encrypt**$(ID^*|_\ell, M_\mu^*, PP)$.

4. **Query 2**: $\mathcal{A}$ may continue to request private keys for hierarchical identities $ID_{q_1+1}, \ldots, ID_q$.

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

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{HIBE}(\lambda) = \left|\Pr[\mu = \mu'] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the game. An HIBE scheme is IND-CPA secure if for all probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter $\lambda$.

## 2.2 Hierarchical Single Revocation Encryption

Single revocation encryption (SRE) is a new type of public-key encryption introduced by Lee et al. [24] to design an efficient public key revocation scheme. In SRE, a private key is associated with group and member labels $(GL', ML')$, and a ciphertext is associated with group and revoked member labels $(GL, ML)$. At the decryption step, if $GL = GL' \wedge ML \neq ML'$ is established, a user having the corresponding private key can decrypt the ciphertext. We extend the concept of SRE to define hierarchical SRE (HSRE) that supports private key delegation. In HSRE, a key is divided into a delegate key and a private key. The delegate key of HSRE is associated with a hierarchical identity $ID = (I_1, \ldots, I_k)$ similar to the private key of HIBE that support the key delegation. The private key of HSRE is associated with a hierarchical identity $ID'$ and labels $(GL', ML')$, and a ciphertext is also associated with $ID$ and labels $(GL, ML)$. In this case, if $ID = ID' \wedge GL = GL' \wedge ML \neq ML'$ is established, a user having the corresponding private key can decrypt the ciphertext. The detailed syntax of HSRE is described as follows:

**Definition 2.3** (Hierarchical Single Revocation Encryption, HSRE)**.** An HSRE scheme consists of six algorithms **Setup**, **GenKey**, **Delegate**, **MakeKey**, **Encrypt**, and **Decrypt**, which are defined as follows:

**Setup**$(1^\lambda, L)$: The setup algorithm takes as input a security parameter $1^\lambda$. It outputs a master key $MK$ and public parameters $PP$.

**GenKey**$(ID|_k, (GL, ML), MK, PP)$: The private key generation algorithm takes as input a hierarchical identity $ID|_k$, labels $(GL, ML)$, the master key $MK$, and public parameters $PP$. It outputs a private key $SK_{ID|_k,(GL,ML)}$.

**Delegate**$(ID|_k, SK_{ID|_{k-1}}, PP)$: The delegation algorithm takes as input a hierarchical identity $ID|_k$, a private key $SK_{ID|_{k-1}}$, and public parameters $PP$. It outputs a delegate key $SK_{ID|_k}$.

**MakeKey**$((GL, ML), SK_{ID|_k}, PP)$: The key making algorithm takes as input labels $(GL, ML)$, a delegate key $SK_{ID|_k}$, and public parameters $PP$. It outputs a private key $SK_{ID|_k,(GL,ML)}$.

**Encrypt**$(ID|_{\ell-1}, (GL, ML), M, PP)$: The encryption algorithm takes as input a hierarchical identity $ID|_{\ell-1}$, labels $(GL, ML)$, a message $M \in \mathcal{M}$, and public parameters $PP$. It outputs a ciphertext $CT_{ID|_{\ell-1},(GL,ML)}$.

**Decrypt**$(CT_{ID|_{\ell-1},(GL,ML)}, SK_{ID'|_{\ell-1},(GL',ML')}, PP)$: The decryption algorithm takes as input a ciphertext $CT_{ID|_{\ell-1},(GL,ML)}$, a private key $SK_{ID'|_{\ell-1},(GL',ML')}$, and public parameters $PP$. It outputs a message $M$.

The correctness of HSRE is defined as follows: For all $MK$ and $PP$ generated by **Setup**$(1^\lambda, L)$, $SK_{ID}$ generated by **GenKey**$(ID'|_{\ell-1}, (GL',ML'), MK, PP)$ for any $(GL',ML')$, and any $(GL,ML)$ and any $M$, it is required that

- If $ID|_k \in \text{Prefix}(ID|_{\ell-1})$, **MakeKey**$((GL,ML),$ **Delegate**$(ID|_{\ell-1}, SK_{ID|_k}, PP), PP) = SK_{ID|_{\ell-1},(GL,ML)}$.
- If $(ID|_{\ell-1} = ID'|_{\ell-1}) \wedge (GL = GL') \wedge (ML \neq ML')$, **Decrypt**$(CT_{ID|_{\ell-1},(GL,ML)}, SK_{ID'|_{\ell-1},(GL',ML')}, PP) = M$.

**Definition 2.4** (IND-CPA Security)**.** The security of HSRE is defined in terms of the indistinguishability under chosen plaintext attacks (IND-CPA). The security game is defined as the following game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Setup**: $\mathcal{C}$ runs **Setup**$(1^\lambda, L)$ to generate a master key $MK$ and public parameters $PP$. It keeps $MK$ to itself and gives $PP$ to $\mathcal{A}$.
2. **Query 1**: $\mathcal{A}$ adaptively requests delegate keys and private keys. If a delegate key for $ID|_k$ is requested, then it generates $SK_{ID|_k}$ by running **Delegate**$(ID|_k, SK_{ID|_{k-1}}, PP)$. If a private key for $ID|_k, (GL,ML)$ is requested, then it generates $SK_{ID|_k,(GL,ML)}$ by running **GenKey**$(ID|_k, (GL,ML), MK, PP)$.
3. **Challenge**: $\mathcal{A}$ submits a challenge hierarchical identity $ID^*|_{\ell-1}$, challenge labels $(GL^*, ML^*)$ and two messages $M_0^*, M_1^*$ with the equal length subject to the restrictions: (1) For all $ID|_k$ of delegate keys, it is required that $ID|_k \notin \text{Prefix}(ID^*|_{\ell-1})$. (2) For all $ID|_k, (GL,ML)$ of private key queries, it is required that $(ID|_k, GL) \neq (ID^*|_{\ell-1}, GL^*)$ or $(ID|_k, GL) = (ID^*|_{\ell-1}, GL^*) \wedge (ML = ML^*)$.

   $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$ and gives the challenge ciphertext $CT^*$ to $\mathcal{A}$ by running **Encrypt**$(ID^*|_{\ell-1}, (GL^*, ML^*), M_\mu^*, PP)$.
4. **Query 2**: $\mathcal{A}$ may continue to request delegate keys and private keys.
5. **Guess**: $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ of $\mu$, and wins the game if $\mu = \mu'$.

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

## 2.3 Revocable Hierarchical Identity-Based Encryption

Revocable HIBE (RHIBE) is an extension of HIBE that additionally supports the revocation of private keys, and Seo and Emura first introduced the concept of RHIBE and proposed RHIBE schemes [34, 36]. RHIBE schemes are classified into history-preserving type and history-free type according to the method of generating an update key. Since the RHIBE schemes of this paper are history-preserving type that requires all update keys related with a user's private key when deriving a decryption key, we define the history-preserving RHIBE scheme. The detailed syntax of history-preserving RHIBE is described as follows:

**Definition 2.5** (Revocable HIBE)**.** An RHIBE scheme with history-preserving updates for the identity space $\mathcal{I}$, the time space $\mathcal{T}$, and the message space $\mathcal{M}$, consists of seven algorithms **Setup**, **GenKey**, **UpdateKey**, **DeriveKey**, **Encrypt**, **Decrypt**, and **Revoke**, which are defined as follows:

**Setup**$(1^\lambda, L)$: This algorithm takes as input a security parameter $1^\lambda$, the maximum depth $L$ of a hierarchical identity. It outputs a master key $MK$, an (empty) revocation list $RL_\varepsilon$, and public parameters $PP$.

**GenKey**$(ID|_k, SK_{ID|_{k-1}}, PP)$: This algorithm takes as input a hierarchical identity $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$, a private key $SK_{ID|_{k-1}}$, and public parameters $PP$. It outputs a private key $SK_{ID|_k}$.

**UpdateKey**$(T, RL_{ID|_{k-1}}, SK_{ID|_{k-1}}, PP)$: This algorithm takes as input time $T \in \mathcal{T}$, a revocation list $RL_{ID|_{k-1}}$, a private key $SK_{ID|_{k-1}}$, and public parameters $PP$. It outputs an update key $UK_{ID|_{k-1},T}$.

**DeriveKey**$(SK_{ID|_k}, UK_{ID|_0,T}, \ldots, UK_{ID|_{k-1},T}, PP)$: This algorithm takes as input a private key $SK_{ID|_k}$ for a hierarchical identity $ID|_k$, update keys $UK_{ID|_0,T}, \ldots, UK_{ID|_{k-1},T}$ for time $T$, and the public parameters $PP$. It outputs a decryption key $DK_{ID|_k,T}$.

**Encrypt**$(ID|_\ell, T, M, PP)$: This algorithm takes as input a hierarchical identity $ID|_\ell = (I_1, \ldots, I_\ell) \in \mathcal{I}^\ell$, time $T$, a message $M$, and the public parameters $PP$. It outputs a ciphertext $CT_{ID|_\ell,T}$.

**Decrypt**$(CT_{ID|_\ell,T}, DK_{ID'|_k,T'}, PP)$: This algorithm takes as input a ciphertext $CT_{ID|_\ell,T}$, a decryption key $DK_{ID'|_k,T'}$ and the public parameters $PP$. It outputs an encrypted message $M$.

**Revoke**$(ID|_k, T, RL_{ID|_{k-1}})$: This algorithm takes as input a hierarchical identity $ID|_k$, revocation time $T$, and a revocation list $RL_{ID|_{k-1}}$. It updates the revocation list $RL_{ID|_{k-1}}$.

The correctness of RHIBE is defined as follows: For all $MK$ and $PP$ generated by **Setup**$(1^\lambda, L, N_{max})$, $SK_{ID|_k}$ generated by **GenKey**$(ID|_k, SK_{ID|_{k-1}}, PP)$, $UK_{ID|_{k-1},T}$ generated by **UpdateKey**$(T, RL_{ID|_{k-1}}, SK_{ID|_{k-1}}, PP)$, $CT_{ID|_\ell,T}$ generated by **Encrypt**$(ID|_\ell, T, M, PP)$, it is required that

- If $ID|_j$ is not revoked in $RL_{ID|_{j-1}}$ to the time $T$ for all $j \in [k]$, then **DeriveKey**$(SK_{ID|_k}, UK_{ID|_{k-1},T}, PP) = DK_{ID|_k,T}$.
- If $(ID|_\ell = ID'|_\ell) \wedge (T = T')$, then **Decrypt**$(CT_{ID|_\ell,T}, DK_{ID'|_\ell,T'}, PP) = M$.

**Definition 2.6** (IND-CPA Security)**.** The IND-CPA security of RHIBE is defined in terms of the following experiment between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Setup**: $\mathcal{C}$ obtains a master key $MK$, a revocation list $RL_\varepsilon$, and public parameters $PP$ by running **Setup**$(1^\lambda, L)$. It keeps $MK, RL_\varepsilon$ to itself and gives $PP$ to $\mathcal{A}$.
2. **Phase 1**: $\mathcal{A}$ adaptively requests a polynomial number of queries. These queries are processed as follows:
   - Create key: If it is a create key query for a hierarchical identity $ID|_k$, then $\mathcal{C}$ creates a private key $SK_{ID|_k}$ by running **GenKey**$(ID|_k, SK_{ID|_{k-1}}, PP)$ with the restriction that the private key $SK_{ID|_{k-1}}$ was already created.
   - Private key: If it is a private key query for a hierarchical identity $ID|_k$, then $\mathcal{C}$ reveals the private key $SK_{ID|_k}$ that was already created.
   - Update key: If it is an update key query for a hierarchical identity $ID|_{k-1}$ and time $T$, then $\mathcal{C}$ gives an update key $UK_{ID|_{k-1},T}$ by running **UpdateKey**$(T, RL_{ID|_{k-1}}, SK_{ID|_{k-1}}, PP)$ with the restrictions that $SK_{ID|_{k-1}}$ was already created and $ID|_{k-1}$ or one of its ancestor was not revoked on time $T$. Although we described this update key as a key query, we can assume that all update keys for created private keys are broadcasted to $\mathcal{A}$.

- Decryption key: If it is a decryption key query for a hierarchical identity $ID|_k$ and time $T$, then $\mathcal{C}$ gives a decryption key $DK_{ID|_k,T}$ by running **DeriveKey**$(SK_{ID|_k}, UK_{ID|_0,T}, \dots, UK_{ID|_{k-1},T}, PP)$ with the restriction that $SK_{ID|_{k-1}}$ was already created and $ID_{ID|_k}$ is not revoked in $UK_{ID|_{k-1},T}$.
- Revocation: If it is a revocation query for a hierarchical identity $ID|_k$ and time $T$, then $\mathcal{C}$ updates a revocation list $RL_{ID|_{k-1}}$ by running **Revoke**$(ID|_k, T, RL_{ID|_{k-1}})$ with the restriction: A revocation query for $ID|_k$ on time $T$ cannot be requested if an update key query for $ID|_k$ on the time $T$ was requested.

Note that we assume that update key, decryption key, and revocation queries are requested in non-decreasing order of time.

3. **Challenge**: $\mathcal{A}$ submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*, \dots, I_\ell^*)$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$ with the following restrictions:

   - If a private key query for $ID|_k \in \text{Prefix}(ID^*|_\ell)$ was requested, then $ID|_k$ or one of its ancestors must be revoked at some time $T \le T^*$.
   - A decryption key query for $ID|_k \in \text{Prefix}(ID^*|_\ell)$ on the challenge time $T^*$ was not requested.

   $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$ and gives the challenge ciphertext $CT^*_{ID^*|_\ell,T^*}$ to $\mathcal{A}$ by running **Encrypt**$(ID^*|_\ell, T^*, M^*_\mu, PP)$.

4. **Phase 2**: $\mathcal{A}$ may continue to request a polynomial number of queries subject to the restrictions of the challenge step.

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

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

## 3 Revocable HIBE with CS

In this section, we generically construct an RHIBE scheme by combining HIBE schemes and the complete subtree method and prove the adaptive security of this scheme.

### 3.1 Binary Tree

A perfect binary tree $\mathcal{BT}$ is a tree data structure in which all internal nodes have two child nodes and all leaf nodes have the same depth. Let $N = 2^n$ be the number of leaf nodes in $\mathcal{BT}$. The number of all nodes in $\mathcal{BT}$ is $2N-1$ and we denote $v_i$ as a node in $\mathcal{BT}$ for any $1 \le i \le 2N-1$. The depth $d_i$ of a node $v_i$ is the length of the path from a root node to the node. The root node of a tree has depth zero. The depth of $\mathcal{BT}$ is the length of the path from the root node to a leaf node. A level of $\mathcal{BT}$ is a set of all nodes at given depth.

Each node $v_i \in \mathcal{BT}$ has an identifier $L_i \in \{0,1\}^*$ which is a fixed and unique string. An identifier of each node is assigned as follows: Each edge in the tree is assigned with 0 or 1 depending on whether it is connected to the left or right child node. The identifier $L_i$ of a node $v_i$ is obtained by reading all labels of edges in a path from the root node to the node $v_i$. The root node has an empty identifier $\varepsilon$. For a node $v_i$, we define Label$(v_i)$ be the identifier of $v_i$ and Depth$(v_i)$ be the depth $d_i$ of $v_i$.

A subtree $\mathcal{T}_i$ in $\mathcal{BT}$ is defined as a tree that is rooted at a node $v_i \in \mathcal{BT}$. A subset $S_i$ is defined as a set of all leaf nodes in $\mathcal{T}_i$. For any two nodes $v_i, v_j \in \mathcal{BT}$ where $v_j$ is a descendant of $v_i$, $\mathcal{T}_{i,j}$ is defined as a subtree $\mathcal{T}_i - \mathcal{T}_j$, that is, all nodes that are descendants of $v_i$ but not $v_j$. A subset $S_{i,j}$ is defined as a set of leaf nodes in $\mathcal{T}_{i,j}$, that is, $S_{i,j} = S_i \setminus S_j$. For a perfect binary tree $\mathcal{BT}$ and a subset $R$ of leaf nodes, $ST(R)$ is defined as the Steiner Tree induced by the set $R$ and the root node, that is, the minimal subtree of $\mathcal{BT}$ that connects all the leaf nodes in $R$ and the root node.

## 3.2 Complete Subtree Method

The complete subtree (CS) method is a type of the subset cover framework proposed by Naor et al. [31] to design revocation schemes for stateless receivers. The detailed description of the CS method is given as follows.

**CS.Setup(**$N$**):** Let $N = 2^n$ for simplicity. It sets a perfect binary tree $\mathcal{BT}$ of depth $n$. It outputs the binary tree $\mathcal{BT}$. Note that a user is assigned to a leaf node $v \in \mathcal{BT}$ and the collection $\mathcal{S}$ is defined as $\{S_i\}$ where $S_i$ is the set of all leaves in a subtree $\mathcal{T}_i$ with a subroot $v_i \in \mathcal{BT}$.

**CS.Assign(**$\mathcal{BT}, v_u$**):** Let $v_u$ be a leaf node of $\mathcal{BT}$ that is assigned to a user $u$. 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 $v_{k_n} = v_u$. It initializes a path set $PV$ as an empty one. For all $j \in \{k_0, \ldots, k_n\}$, it adds $S_j$ into $PV$. It outputs the path set $PV = \{S_j\}$.

**CS.Cover(**$\mathcal{BT}, R$**):** This algorithm partitions $\mathcal{N} \setminus R$ into disjoint subsets $S_{i_1}, \ldots, S_{i_m}$ as follows: It first builds the Steiner Tree $ST(R)$ which is the minimum subtree of $\mathcal{BT}$ that connects all the leaf nodes in $R$ and the root node. Let $\mathcal{T}_{k_1}, \ldots \mathcal{T}_{k_m}$ be all the subtrees of $\mathcal{BT}$ 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 $CV$ as empty one. For all $i \in \{k_1, \ldots, k_m\}$, it adds $S_i$ into $CV$. It outputs the cover set $CV = \{S_i\}$.

**CS.Match(**$CV, PV$**):** It finds a common subset $S_k$ such that $S_k \in CV$ and $S_k \in PV$. If $S_k$ exists, it outputs $(S_k, S_k)$. Otherwise, it outputs $\perp$.

**Lemma 3.1.** *Let $PV$ be a path set obtained by the **CS.Assign** algorithm for a leaf node $v$, and $CV$ be a cover set obtained by the **CS.Cover** algorithm for a set $R$ of leaf nodes. In this case, the CS method satisfies the following two properties: 1) If $v \notin R$, there is only one subset $S_k \in PV \cap CV$. 2) If $v \in R$, $PV \cap CV$ is an empty set.*

*Proof.* We prove the first property. If $v \notin R$, then $v$ belongs to only one subset $S_k \in CV$ since the **CS.Cover** algorithm outputs disjoint subsets $S_{i_1}, \ldots, S_{i_m}$. Since the subtree $T_k$ of $S_k$ contains $v$ as a leaf node, the root node $v_k$ of $T_k$ is an ancestor of $v$. Thus, there is only one $S_k \in CV \cap PV$ since $v_k$ is one of the path nodes of $v$.

Next, we prove the second property. If $v \in R$, the path nodes of $v$ belong to the tree $ST(R)$. Because disjoint subtrees $\mathcal{T}_{i_1}, \ldots, \mathcal{T}_{i_m}$ are created by removing $ST(R)$ by the **CS.Cover** algorithm, these subtrees cannot include $v$. Thus, $CV \cap PV$ is an empty set. □

## 3.3 Generic Construction

We first define three encoding functions $E_1, E_2$, and $E_3$. We define the function $E_i(x)$ takes a string $x$ as input and returns $i \| x$. In this case, if $i \neq j$, then $E_i(x) \neq E_j(y)$ for any strings $x$ and $y$. If $\vec{x} = (x_1, x_2, \ldots, x_\ell)$, the function $E_i(\vec{x})$ returns a vector $(E_i(x_1), \ldots, E_i(x_\ell))$. An RHIBE-CS scheme that is designed by generically combining HIBE schemes and the CS method is described as follows:

**RHIBE-CS.Setup($1^\lambda, L$):** Let $\mathcal{I} = \{0,1\}^n$ be the identity space and $L$ be the maximum depth of a hierarchical identity.

1. It first obtains $MK_{HIBE_1}, PP_{HIBE_1}$ by running **HIBE.Setup**$(1^\lambda, L+1)$. It obtains $MK_{HIBE_2}, PP_{HIBE_2}$ by running **HIBE.Setup**$(1^\lambda, L)$.
2. It defines a binary tree $\mathcal{BT}$ by running **CS.Setup**$(2^n)$ where an identity $I \in \mathcal{I}$ is uniquely assigned to a leaf node $v$ such that $\text{Label}(v) = I$.
3. It outputs a master key $MK = (MK_{HIBE_1}, MK_{HIBE_2})$, a revocation list $RL_\varepsilon = \emptyset$, and public parameters $PP = (PP_{HIBE_1}, PP_{HIBE_2}, \mathcal{BT})$.

**RHIBE-CS.GenKey($ID|_k, SK_{ID|_{k-1}}, PP$):** Let $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ where $k \geq 1$. Let $SK_{ID|_0} = MK$ where $SK_{HIBE_1, ID|_0} = MK_{HIBE_1}$ and $SK_{HIBE_2, ID|_0} = MK_{HIBE_2}$.

1. It obtains $SK_{HIBE_1, ID|_k}$ by running **HIBE.Delegate**$(E_1(ID|_k), SK_{HIBE_1, ID|_{k-1}}, PP_{HIBE_1})$, and obtains $SK_{HIBE_2, ID|_k}$ by running **HIBE.Delegate**$(E_1(ID|_k), SK_{HIBE_2, ID|_{k-1}}, PP_{HIBE_2})$.
2. Finally, it outputs a private key $SK_{ID|_k} = (SK_{HIBE_1, ID|_k}, SK_{HIBE_2, ID|_k})$.

**RHIBE-CS.UpdateKey($T, RL_{ID|_{k-1}}, SK_{ID|_{k-1}}, PP$):** Let $SK_{ID|_k} = (SK_{HIBE_1, ID|_k}, SK_{HIBE_2, ID|_k})$. To generate an update key for $ID|_{k-1}$ and $T$, it proceeds as follows:

1. It initializes $RV = \emptyset$. For each tuple $(ID_j, T_j) \in RL_{ID|_{k-1}}$ where $ID_j = (I_1, \ldots, I_{k-1}, I_k)$, it adds a leaf node $v_j \in \mathcal{BT}$ which is associated with $I_k$ such that $\text{Label}(v_j) = I_k$ into $RV$ if $T_j \leq T$. It obtains $CV_{k-1}$ by running **CS.Cover**$(\mathcal{BT}, RV)$.
2. For each $S_i \in CV_{k-1}$, it sets $L_i = \text{Label}(S_i)$ and obtains $SK_{HIBE_2, S_i}$ by running **HIBE.Delegate** $((E_1(ID|_{k-1}), E_2(L_i \| T)), SK_{HIBE_2, ID|_{k-1}}, PP_{HIBE_2})$.
3. Finally, it outputs an update key $UK_{ID|_{k-1}, T} = \big(CV_{k-1}, \{SK_{HIBE_2, S_i}\}_{S_i \in CV_{k-1}}\big)$.

**RHIBE-CS.DeriveKey($SK_{ID|_k}, UK_{ID|_0, T}, \ldots, UK_{ID|_{k-1}, T}, PP$):** Let $SK_{ID|_k} = (SK_{HIBE_1, ID|_k}, SK_{HIBE_2, ID|_k})$. To derive a decryption key for $ID|_k = (I_1, \ldots, I_k)$ and $T$, it proceeds as follows:

1. It obtains $SK_{HIBE_1, ID|_k, T}$ by running **HIBE.Delegate**$((E_1(ID|_k), E_3(T)), SK_{HIBE_1, ID|_k}, PP_{HIBE_1})$.
2. For each $u \in [k]$, it proceeds as follows:
   (a) It obtains $PV_u$ by running **CS.Assign**$(\mathcal{BT}, v_u)$ where $v_u$ is a leaf such that $\text{Label}(v_u) = I_u$.
   (b) It finds $(S_{i_u}, S_{i_u})$ by running **CS.Match**$(CV_{u-1}, PV_u)$. If it fails to find, it returns $\perp$.
   (c) It retrieves $SK_{HIBE_2, S_{i_u}}$ from $UK_{ID|_{u-1}, T}$ and sets $SK_{HIBE_2, u} = (S_{i_u}, SK_{HIBE_2, S_{i_u}})$.
3. Finally, it outputs a decryption key $DK_{ID|_k, T} = (SK_{HIBE_1, ID|_k, T}, SK_{HIBE_2, 1}, \ldots, SK_{HIBE_2, k})$.

**RHIBE-CS.Encrypt($ID|_\ell, T, M, PP$):** To generate a ciphertext for $ID|_\ell = (I_1, \ldots, I_\ell)$ and $T$, it proceeds as follows:

1. It selects random $R_{2,1}, \ldots, R_{2,\ell}$ and sets $R_1 = M \oplus (R_{2,1} \oplus \cdots \oplus R_{2,\ell})$. It obtains $CT_{HIBE_1}$ by running **HIBE.Encrypt**$((E_1(ID|_\ell), E_3(T)), R_1, PP_{HIBE_1})$.
2. For each $k \in [\ell]$, it performs the following steps.
   (a) It obtains $PV_k$ by running **CS.Assign**$(\mathcal{BT}, v_k)$ where $v_k$ is a leaf such that $\text{Label}(v_k) = I_k$.
   (b) For each $S_j \in PV_k$, it sets $L_j = \text{Label}(S_j)$ and obtains $CT_{HIBE_2, S_j}$ by running **HIBE.Encrypt** $((E_1(ID|_{k-1}), E_2(L_j \| T)), R_{2,k}, PP_{HIBE_2})$.

(c) It creates $CT_{PV_k} = \big(PV_k, \{CT_{HIBE_2,S_j}\}_{S_j \in PV_k}\big)$.

3. Finally, it outputs a ciphertext $CT_{ID|_\ell,T} = (CT_{HIBE_1}, CT_{PV_1}, \ldots CT_{PV_\ell})$.

**RHIBE-CS.Decrypt**($CT_{ID|_\ell,T}, DK_{ID'|_\ell,T'}, PP$)**:** Let $CT_{ID|_\ell,T} = (CT_{HIBE_1}, CT_{PV_1}, \ldots, CT_{PV_\ell})$ and $DK_{ID'|_\ell,T'} = (SK_{HIBE_1}, SK_{HIBE_2,1}, \ldots, SK_{HIBE_2,k})$ where $CT_{PV_k} = (PV_k, \{CT_{HIBE_2,S_j}\})$. If $(ID|_\ell \neq ID'|_\ell) \vee (T \neq T')$, then it returns $\perp$.

1. It first obtains $R_1$ by running **HIBE.Decrypt**($CT_{HIBE_1}, SK_{HIBE_1}, PP_{HIBE_1}$).
2. For each $k \in [\ell]$, it performs the following steps:
   (a) It gets $S_{i_k}$ from $SK_{HIBE_2,k} = (S_{i_k}, SK_{HIBE_2,S_{i_k}})$ and retrieves $CT_{HIBE_2,S_{i_k}}$ from $CT_{PV_k}$.
   (b) It obtains $R_{2,k}$ by running **HIBE.Decrypt**($CT_{HIBE_2,S_{i_k}}, SK_{HIBE_2,S_{i_k}}, PP_{HIBE_2}$).
3. Finally, it outputs a message $M = R_1 \oplus R_{2,1} \oplus \cdots \oplus R_{2,\ell}$.

**RHIBE-CS.Revoke**($ID|_k, T, RL_{ID|_{k-1}}$)**:** If $(ID|_k, *)$ already exists in $RL_{ID|_{k-1}}$, it outputs $RL_{ID|_{k-1}}$. Otherwise, it adds $(ID|_{k-1}, T)$ to $RL_{ID|_{k-1}}$ and outputs the updated $RL_{ID|_{k-1}}$.

## 3.4 Correctness

In order to show that our RHIBE-CS scheme is correct, we must show that the original message can be recovered when a ciphertext is decrypted with a valid decryption key. First, we show that if the hierarchical identity $ID|_k = (I_1, \ldots, I_k)$ of a user is not revoked in revocation lists $RL_{ID|_0}, \ldots, RL_{ID|_{k-1}}$ before time $T$, a valid decryption key can be derived. It is easy to derive a valid key component $SK_{HIBE_1,ID|_k,T}$ by using the delegation property of HIBE. For each $u \in [k]$, the path nodes $PV_u$ of a ciphertext can be determined since a private key is associated with $ID|_k$ and the leaf node $v_u$ for $I_u$ in a binary tree is fixed in advance, and we get a common subset $S_{i_u}$ of $CV_{u-1}$ and $PV_u$ by the first property of the CS method in Lemma 3.1. Thus, we can obtain valid decryption key components $SK_{HIBE_2,S_{i_1}}, \ldots, SK_{HIBE_2,S_{i_k}}$ associated with $S_{i_1}, \ldots, S_{i_k}$ respectively.

Next, we show that a ciphertext for $ID|_\ell$ and $T$ can be decrypted by using a decryption key for $ID'|_\ell$ and $T'$. At this time, a ciphertext $CT_{ID|_\ell,T}$ is composed of a ciphertext $CT_{HIBE_1}$ and multiple ciphertexts $CT_{PV_1}, \ldots, CT_{PV_\ell}$, and a decryption key $DK_{ID'|_\ell,T'}$ consists of a private key $SK_{HIBE_1}$ and a number of tuples $(S_{i_1}, SK_{HIBE_2,S_{i_1}}), \ldots, (S_{i_\ell}, SK_{HIBE_2,S_{i_\ell}})$. If $ID|_\ell = ID'|_\ell$ and $T = T'$ are satisfied, we can obtain the correct $R_1$ by decrypting $CT_{HIBE_1}$ with the private key $SK_{HIBE_1}$ from the correctness of the HIBE scheme. Next, for each $k \in [\ell]$, we can derive the correct $R_{2,k}$ by decrypting the ciphertext $CT_{HIBE_2,S_{i_k}} \in CT_{PV_k}$ with the private key $SK_{HIBE_2,S_{i_k}}$ by the correctness of the HIBE scheme. Therefore, we can correctly derive the original message $M$ by computing $R_1 \oplus R_{2,1} \oplus \cdots \oplus R_{2,\ell}$.

## 3.5 Security Analysis

In this section, we prove the security of our RHIBE-CS scheme by using the security of the underlying HIBE schemes and the properties of the CS method. The basic idea of this proof is to separate adversaries into $\ell+1$ types and perform independent proofs for these individual types.

**Theorem 3.2.** *The generic RHIBE-CS scheme is IND-CPA secure if the underlying HIBE schemes are IND-CPA secure.*

*Proof.* Let $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$ be the challenge hierarchical identity and $T^*$ be the challenge time. We divide the behavior of an adversary as $\ell+1$ types: Type-1, $\cdots$, Type-$\ell+1$, which are defined as follows:

**Type-$\tau$.** An adversary is Type-$\tau$ for $\tau \in \{1, \dots, \ell\}$ if it does not request a private key for $ID|_k \in \text{Prefix}(ID^*|_{\tau-1})$, but it must request a private key for $ID|_k = ID^*|_\tau$.

**Type-$\ell+1$.** An adversary is Type-$\ell+1$ if it does not request a private key for $ID|_k \in \text{Prefix}(ID^*|_\ell)$.

Let $E_\tau$ be the event that $\mathcal{A}$ behaves like Type-$\tau$ adversary. From Lemma 3.3 and Lemma 3.4, we obtain the following result

$$\mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq \sum_{\tau=1}^{\ell+1} \Pr[E_\tau] \mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq O(\ell n) \mathbf{Adv}_{\mathcal{B}}^{HIBE}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{HIBE}(\lambda).$$

This completes our proof. □

**Lemma 3.3.** *For the Type-$\tau$ adversary such that $\tau \in \{1, \dots, \ell\}$, the generic RHIBE-CS scheme is IND-CPA secure if the HIBE scheme is IND-CPA secure.*

*Proof.* Let $CT^* = (CT^*_{HIBE_1}, CT^*_{PV_1}, \dots, CT^*_{PV_\ell})$ be the challenge ciphertext. For the security proof, we define hybrid games $\mathbf{H}_0, \mathbf{H}_1, \dots, \mathbf{H}_{n+1}$ as follows:

**Game $\mathbf{H}_0$.** This game is the original security game defined in the security model except that the challenge bit $\mu$ is fixed to 0.

**Game $\mathbf{H}_\rho$** In this game, the generation of $CT^*_{PV_\tau} = (PV_\tau, \{CT^*_{HIBE_2, S_{j_d}}\}_{S_{j_d} \in PV_\tau})$ in the challenge ciphertext $CT^*$ is changed. The simulator of this game sets $R'_{2,\tau} = M^*_1 \oplus R_1 \bigoplus_{1 \leq k \neq \tau \leq \ell} R_{2,k}$ and $R_{2,\tau} = M^*_0 \oplus R_1 \bigoplus_{1 \leq k \neq \tau \leq \ell} R_{2,k}$. The game $\mathbf{H}_\rho$ is similar to the game $\mathbf{H}_{\rho-1}$ except that $CT^*_{HIBE_2, S_{j_\rho}}$ is an encryption on the value $R'_{2,\tau}$. Specifically, $CT^*_{HIBE_2, S_{j_d}}$ for $d \leq \rho$ is an encryption on $R'_{2,\tau}$ and $CT^*_{HIBE_2, S_{j_d}}$ for $d > \rho$ is an encryption on $R_{2,\tau}$.

**Game $\mathbf{H}_{n+1}$** This game is the original security game in the security model except that the challenge bit $\mu$ is fixed to 1.

Suppose there exists a Type-$\tau$ adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}_{\rho-1}$ and $\mathbf{H}_\rho$ of the RHIBE scheme with a non-negligible advantage. An algorithm $\mathcal{B}$ that attacks the HIBE scheme is initially given public parameters $PP_{HIBE_2}$ by a challenger $\mathcal{C}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

**Setup:** $\mathcal{B}$ generates $MK_{HIBE_1}, PP_{HIBE_1}$ by running $\textbf{HIBE.Setup}(1^\lambda, L+1)$. It initializes $RL_\varepsilon = \emptyset$ and gives $PP = (PP_{HIBE_1}, PP_{HIBE_2}, \mathcal{BT})$ to $\mathcal{A}$.

**Phase 1:** $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, decryption key, and revocation queries.

- For a private key query with $ID|_k$, $\mathcal{B}$ proceeds as follows: It generates $SK_{HIBE_1, ID|_k}$ by running $\textbf{HIBE.GenKey}(E_1(ID|_k), MK_{HIBE_1}, PP_{HIBE_1})$. It receives $SK_{HIBE_2, ID|_k}$ from $\mathcal{C}$ by querying a private key for $E_1(ID|_k)$. It gives $SK_{ID|_k} = (SK_{HIBE_1, ID|_k}, SK_{HIBE_2, ID|_k})$ to $\mathcal{A}$.

- For an update key query with $ID|_{k-1}$ and time $T$, $\mathcal{B}$ proceeds as follows:

  1. It initializes $RV = \emptyset$. For each $(ID_j, T_j) \in RL_{ID|_{k-1}}$, it adds a leaf node $v_j \in \mathcal{BT}$ into $RV$ if $T_j \leq T$. It obtains $CV_{k-1}$ by running $\textbf{CS.Cover}(\mathcal{BT}, RV)$.
  2. For each $S_i \in CV_{k-1}$, it sets $L_i = \text{Label}(S_i)$ and receives $SK_{HIBE_2, S_i}$ from $\mathcal{C}$ by querying a private key for $(E_1(ID|_{k-1}), E_2(L_i \| T))$.

3. It creates $UK_{ID|_{k-1},T} = \big(CV_{k-1}, \{SK_{HIBE_2,S_i}\}_{S_i \in CV_{k-1}}\big)$ and gives $UK_{ID|_{k-1},T}$ to $\mathcal{A}$.

- For a decryption key query with $ID|_k$ and time $T$, $\mathcal{B}$ proceeds as follows:
  1. It generates $SK_{HIBE_1,ID|_k,T}$ by running **HIBE.GenKey**$((E_1(ID|_k), E_3(T)), MK_{HIBE_1}, PP_{HIBE_1})$.
  2. Next, it obtains update keys $UK_{ID|_0,T}, \ldots, UK_{ID|_{k-1},T}$ by querying its own update key oracle for $(ID|_0, T), \ldots, (ID|_{k-1}, T)$. For each $u \in [k]$, it obtains $SK_{HIBE_2,u}$ if $I_u$ is not revoked in $UK_{ID|_{u-1},T}$ by following the procedure in the **RHIBE-CS.DeriveKey** algorithm.
  3. If $I_u$ was revoked in $UK_{ID|_{u-1},T}$ for any $u \in [k]$, then it gives $\perp$ to $\mathcal{A}$. Otherwise, t gives $DK_{ID|_k,T} = (SK_{HIBE_1,ID|_k,T}, SK_{HIBE_2,1}, \ldots, SK_{HIBE_2,k})$ to $\mathcal{A}$.
- For a revocation query with a hierarchical identity $ID|_k$ and time $T$, $\mathcal{B}$ adds $(ID|_k, T)$ to $RL_{ID|_{k-1}}$ if $ID|_k$ was not revoked before.

**Challenge**: $\mathcal{A}$ submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$. $\mathcal{B}$ proceeds as follows:

1. It first selects random $R_1, \{R_{2,i}\}_{1 \le i \ne \tau \le \ell}$ and computes $R_2 = \bigoplus_{1 \le i \ne \tau \le \ell} R_{2,i}$. It sets $R_{2,\tau,0} = M_0^* \oplus R_1 \oplus R_2$ and $R_{2,\tau,1} = M_1^* \oplus R_1 \oplus R_2$. Next, it generates $CT^*_{HIBE_1}$ by running **HIBE.Encrypt**$((E_1(ID^*|_\ell), E_3(T^*)), R_1, PP_{HIBE_1})$.
2. For each $k \in [\ell]$, it performs the following steps:
   (a) It obtains $PV_k$ by running **CS.Assign**$(\mathcal{BT}, v_k)$ where a leaf node $v_k$ is associated with $I_k^*$.
   (b) Case $k \ne \tau$: For each $S_j \in PV_k$, it obtains $L_j = \text{Label}(S_j)$ and generates $CT^*_{HIBE_2,S_j}$ by running **HIBE.Encrypt**$((E_1(ID^*|_{k-1}), E_2(L_j \| T^*)), R_{2,i}, PP_{HIBE_2})$.
   It creates $CT^*_{PV_k} = \big(PV_k, \{CT^*_{HIBE_2,S_j}\}_{S_j \in PV_k}\big)$.
   (c) Case $k = \tau$: For each $S_{j_d} \in PV_\tau = \{S_{j_1}, \ldots, S_{j_{n+1}}\}$, it obtains $L_{j_d} = \text{Label}(S_{j_d})$ and proceeds as follows:
   - If $d < \rho$, then it generates $CT^*_{HIBE_2,S_{j_d}}$ by running **HIBE.Encrypt**$((E_1(ID^*|_{\tau-1}), E_2(L_{j_d} \| T^*)), R_{2,\tau,1}, PP_{HIBE_2})$.
   - If $d = \rho$, then it receives $CT^*_{HIBE_2,S_{j_\rho}}$ from $\mathcal{C}$ by submitting a challenge hierarchical identity $(E_1(ID^*|_{\tau-1}), E_2(L_{j_\rho} \| T^*))$ and challenge messages $R_{2,\tau,0}, R_{2,\tau,1}$.
   - If $d > \rho$, then it generates $CT^*_{HIBE_2,S_{j_d}}$ by running **HIBE.Encrypt**$((E_1(ID^*|_{\tau-1}), E_2(L_{j_d} \| T^*)), R_{2,\tau,0}, PP_{HIBE_2})$.

   It creates $CT^*_{PV_\tau} = \big(PV_\tau, \{CT^*_{HIBE_2,S_{j_d}}\}_{S_{j_d} \in PV_\tau}\big)$.
3. It gives a challenge ciphertext $CT^* = (CT^*_{HIBE_1}, CT^*_{PV_1}, \ldots, CT^*_{PV_\ell})$ to $\mathcal{A}$.

**Phase 2**: Same as Phase 1.
**Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0, 1\}$. $\mathcal{B}$ also outputs $\mu'$.

To complete the proof of this lemma, we must show that the simulator can correctly handle private key, update key, and decryption key queries of the adversary. At this time, the HIBE$_2$ private key queries and the HIBE$_2$ challenge ciphertext requested by the simulator to the challenger must satisfy the constraint of the HIBE security model. To simplify the analysis, we first analyze the HIBE$_2$ challenge ciphertext of the challenge ciphertext and check whether the HIBE$_2$ private keys of private keys, update keys, and decryption keys satisfy the constraint.

- Challenge ciphertext: The challenge ciphertext consists of multiple HIBE$_2$ ciphertexts, and the HIBE$_2$ challenge ciphertext is $CT^*_{HIBE_2,S_{j_\rho}} \in CT^*_{PV_\tau}$. In this case, the HIBE$_2$ challenge ciphertext is associated with $(E_1(ID^*|_{\tau-1}), E_2(L_\rho \| T^*))$ where $L_\rho$ is the label of a path node defined by a leaf node $I^*_\tau$.

- Private key: Since the adversary is Type-$\tau$, he cannot request a private key for $ID^*|_k$ and $k < \tau$, but must request a private key for $ID^*|_\tau$. To simplify the analysis, we divide the private key generation of the simulator into the following cases:

  - Case $ID|_k \notin \text{Prefix}(ID^*|_\ell)$: In this case, the constraint of the HIBE$_2$ security model is satisfied, so it can generate a private key by using the HIBE$_2$ private key query.
  - Case $ID|_k \in \text{Prefix}(ID^*|_\ell) \wedge k = \tau$: In this case, we have $ID|_k = ID^*|_\tau = (I^*_1, \ldots, I^*_\tau)$, and the simulator requests an HIBE$_2$ private key for $E_1(ID^*|_\tau) = (E_1(I^*_1), \ldots, E_1(I^*_\tau))$ to the challenger. Since the encoding functions satisfy $E_1(x) \neq E_2(y)$ for any $x$ and $y$, it is established that $E_1(I^*|_\tau) \neq E_2(L_\rho \| T^*)$. Thus, it can generate a private key by using the HIBE$_2$ private key query since the constraint of the HIBE$_2$ security model is satisfied.
  - Case $ID|_k \in \text{Prefix}(ID^*|_\ell) \wedge k > \tau$: This case also includes the index $i = \tau$, so the same logic as in the previous case is applied. Thus, it can generate a private key by using the HIBE$_2$ private key query.

- Update key: The simulator must be able to generate an update key for any $ID|_{k-1}$ and $T$. To simplify the analysis, we divide the update key generation of the simulator into the following cases:

  - Case $T \neq T^*$: In this case, the hierarchical identity of an HIBE$_2$ private key in an update key contains a string $E_2(L_i \| T)$ and the challenge hierarchical identity of the HIBE$_2$ challenge ciphertext contains a string $E_2(L_j \| T^*)$, so it is established that $E_2(L_i \| T) \neq E_2(L_j \| T^*)$. Thus, it can generate an update key by using the HIBE$_2$ private key query.
  - Case $T = T^* \wedge ID|_{k-1} \notin \text{Prefix}(ID^*|_{\tau-1})$: In this case, it is established that $(E_1(ID|_{k-1}), E_2(L_i \| T^*))$ $\notin \text{Prefix}((E_1(ID^*|_{\tau-1}), E_2(L_\rho \| T^*)))$ in the HIBE$_2$ scheme. Thus, since this HIBE$_2$ private key query is allowed in the HIBE security model, it can generate an update key.
  - Case $T = T^* \wedge ID|_{k-1} \in \text{Prefix}(ID^*|_{\tau-1}) \wedge k \leq \tau - 1$: In this case, the HIBE$_2$ private key of an update key is associated with a string $(E_1(ID|_{k-1}), E_2(L_i \| T^*))$, and we have $E_2(L_i \| T^*) \neq E_1(I^*_k)$ by the encoding function. This means that $(E_1(ID|_{k-1}), E_2(L_i \| T^*)) \notin \text{Prefix}((E_1(ID^*|_{\tau-1}),$ $E_2(L_\rho \| T^*)))$. Thus, it can generate an update key since this HIBE$_2$ private key query is allowed in the HIBE security model.
  - Case $T = T^* \wedge ID|_{k-1} \in \text{Prefix}(ID^*|_{\tau-1}) \wedge k = \tau$: In this case, by the constraints of the RHIBE security model, the identity $I^*_\tau$ must be revoked in this update key. From the second property of the CS method in Lemma 3.1, we have that $PV_\tau \cap CV_{\tau-1}$ is an empty set. That is, a label $L_i$ of the cover $CV_{\tau-1}$ is different from the label $L_{j_\rho}$ of the path node $PV_\tau$, so we have $L_i \neq L_{j_\rho}$. This means that $(E_1(ID^*|_{\tau-1}), E_2(L_i \| T^*)) \neq (E_1(ID^*|_{\tau-1}), E_2(L_{j_\rho} \| T^*))$. Thus, it can generate an update key by using the HIBE$_2$ private key query.

- Decryption key: In the case of the decryption key, if the simulator can correctly generate update keys, then simulator can also correctly generate a decryption key.

Finally, we analyze the advantage of the simulation. Let $S_{\mathcal{A}}^{H_i}$ be the event that $\mathcal{A}$ outputs 0 in a game $\mathbf{H}_i$. Since this lemma consists of a number of hybrid games, we get the following result

$$\mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq \frac{1}{2}\left|\Pr[S_{\mathcal{A}}^{H_0}] - \Pr[S_{\mathcal{A}}^{H_{n+1}}]\right| \leq \frac{1}{2}\sum_{\rho=1}^{n+1}\left|\Pr[S_{\mathcal{A}}^{H_{\rho-1}}] - \Pr[S_{\mathcal{A}}^{H_{\rho}}]\right| \leq O(n)\mathbf{Adv}_{\mathcal{B}}^{HIBE}(\lambda).$$

This completes our proof. □

**Lemma 3.4.** *For the Type-$\ell+1$ adversary, the generic RHIBE scheme is IND-CPA secure if the HIBE scheme is IND-CPA secure.*

*Proof.* Suppose there exists a Type-$\ell+1$ adversary $\mathcal{A}$ that attacks the RHIBE scheme with a non-negligible advantage. An algorithm $\mathcal{B}$ that attacks the HIBE scheme is initially given public parameters $PP_{HIBE_1}$ by a challenger $\mathcal{C}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

**Setup:** $\mathcal{B}$ generates $MK_{HIBE_2}, PP_{HIBE_2}$ by running **HIBE.Setup**$(1^\lambda, L)$. It initializes $RL_\varepsilon = \emptyset$ and gives $PP = (PP_{HIBE_1}, PP_{HIBE_2}, \mathcal{BT})$ to $\mathcal{A}$.

**Phase 1:** $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, decryption key, and revocation queries.

- For a private key query with $ID|_k$, $\mathcal{B}$ proceeds as follows: It receives $SK_{HIBE_1,ID|_k}$ from $\mathcal{C}$ by querying a private key for $ID|_k$. It generates $SK_{HIBE_2,ID|_k}$ by running **HIBE.GenKey**$(E_1(ID|_k), MK_{HIBE_2}, PP_{HIBE_2})$. It gives $SK_{ID|_k} = (SK_{HIBE_1,ID|_k}, SK_{HIBE_2,ID|_k})$ to $\mathcal{A}$.
- For an update key query with $ID|_{k-1}$ and $T$, $\mathcal{B}$ proceeds as follows: It generates $UK_{ID|_{k-1},T}$ by running **RHIBE-CS.UpdateKey** algorithm since it knows $MK_{HIBE_2}$. It gives $UK_{ID|_{k-1},T}$ to $\mathcal{A}$.
- For a decryption key query with $ID|_k$ and $T$, $\mathcal{B}$ proceeds as follows:
  1. It receives $SK_{HIBE_1,ID|,T}$ from $\mathcal{C}$ by querying a private key for $(E_1(ID|_k), E_3(T))$.
  2. Next, it obtains update keys $UK_{ID|_0,T}, \ldots, UK_{ID|_{k-1},T}$ by querying its own update key oracle for $(ID|_0, T), \ldots, (ID|_{k-1}, T)$. For each $u \in [k]$, it obtains $SK_{HIBE_2,u}$ if $I_u$ is not revoked in $UK_{ID|_{u-1},T}$ by following the procedure in the **RHIBE-CS.DeriveKey** algorithm.
  3. If $I_u$ was revoked in $UK_{ID|_{u-1},T}$ for any $u \in [k]$, then it gives $\perp$ to $\mathcal{A}$. Otherwise, t gives $DK_{ID|_k,T} = (SK_{HIBE_1,ID|_k,T}, SK_{HIBE_2,1}, \ldots, SK_{HIBE_2,k})$ to $\mathcal{A}$.
- For a revocation query with a hierarchical identity $ID|_k$ and time $T$, $\mathcal{B}$ adds $(ID|_k, T)$ to $RL_{ID|_{k-1}}$ if $ID|_k$ was not revoked before.

**Challenge**: $\mathcal{A}$ submits a challenge hierarchical identity $ID^*|_\ell$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$. $\mathcal{B}$ proceeds as follows:

1. It first selects random $\{R_{2,i}\}_{1\leq i\leq \ell}$ and computes $R_2 = \bigoplus_{1\leq i\leq \ell} R_{2,i}$. It sets $R_{1,0} = M_0^* \oplus R_2$ and $R_{1,1} = M_1^* \oplus R_2$. It receives $CT_{HIBE_1}^*$ from $\mathcal{C}$ by submitting challenge identity $(E_1(ID^*|_\ell), E_3(T^*))$ and challenge messages $R_{1,0}, R_{1,1}$.
2. For each $k \in [\ell]$, it proceeds as follows:
   (a) It obtains $PV_k$ by running **CS.Assign**$(\mathcal{BT}, v_k)$ where a leaf node $v_k$ is associated with $I_k^*$.

(b) For each $S_j \in PV_k$, it obtains $L_j = \text{Label}(S_j)$ and generates $CT^*_{HIBE_2,S_j}$ by running **HIBE.Encrypt** $((E_1(ID^*|_{k-1}), E_2(L_j \| T^*)), R_{2,k}, PP_{HIBE_2})$.

(c) It creates $CT^*_{PV_k} = \big(PV_k, \{CT^*_{HIBE_2,S_j}\}_{S_j \in PV_k}\big)$.

3. It gives a challenge ciphertext $CT^* = (CT^*_{HIBE_1}, CT^*_{PV_1}, \dots, CT^*_{PV_\ell})$ to $\mathcal{A}$.

**Phase 2**: Same as Phase 1.
**Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ also outputs $\mu'$.

To complete the proof of this lemma, we must show that the simulator can correctly generate private keys, update keys, decryption keys, and the challenge ciphertext by using HIBE$_1$ private keys and HIBE$_1$ challenge ciphertext while satisfying the constraint of the HIBE security model.

- Challenge ciphertext: The adversary submits $ID^*|_\ell$ and $T^*$ as challenges, and the simulator requests the HIBE$_1$ challenge ciphertext after applying additional encoding. Therefore, the challenge hierarchical identity of the HIBE$_1$ scheme is set to $(E_1(ID^*|_\ell), E_3(T^*))$.

- Private key: Since the adversary is Type-$\ell+1$, he can query a private key for $ID|_k \notin \text{Prefix}(ID^*|_\ell)$, but he cannot query a private key for $ID|_k \in \text{Prefix}(ID^*|_\ell)$. Therefore, the HIBE private key satisfies $E_1(ID|_k) \notin \text{Prefix}(E_1(ID^*|_\ell))$, so the simulator can generate a private key by using the HIBE$_1$ private key query.

- Update key: Since an update key is composed of only HIBE$_2$ private keys, the simulator can easily generate an update key by using $MK_{HIBE_2}$.

- Decryption key: The adversary can query a decryption key for $ID|_k \notin \text{Prefix}(ID^*|_\ell)$ or $T \neq T^*$ due to the constraints of the RHIBE security model. At this time, a decryption key is composed of an HIBE$_1$ private key and a number of update keys, and update keys are easily generated as previously analyzed. To simplify the analysis, we divide the decryption key generation into the following cases and analyze the decryption key by checking whether it is possible to create an HIBE$_1$ private key.

  – Case $ID|_k \notin \text{Prefix}(ID^*|_\ell)$: In this case, we have $E_1(ID|_k) \notin \text{Prefix}(ID^*|_\ell)$, so an HIBE$_1$ private key satisfies $(E_1(ID|_k), E_3(T)) \notin \text{Prefix}(E_1(ID^*)|_\ell), E_3(T^*))$. Thus, the simulator can generate a decryption key since the constraint of the HIBE security model is satisfied.

  – Case $T \neq T^* \wedge k < \ell$: In this case, we have $E_3(T) \neq E_1(I^*|_{k+1})$ by the property of encoding functions, so an HIBE$_1$ private key satisfies $(E_1(ID|_k), E_3(T)) \notin \text{Prefix}(E_1(ID^*|_\ell), E_3(T^*))$. Thus, the simulator can generate a decryption key since the constraint of the HIBE security model is satisfied.

  – Case $T \neq T^* \wedge k = \ell$: In this case, we have $E_3(T) \neq E_3(T^*)$, so an HIBE$_1$ private key satisfies $(E_1(ID|_k), E_3(T)) \notin \text{Prefix}(E_1(ID^*|_\ell), E_3(T^*))$. Thus, the simulator can generate a decryption key since it satisfies the constraint of the HIBE security model.

  – Case $T \neq T^* \wedge k > \ell$: In this case, we have $E_1(I_{\ell+1}) \neq E_3(T^*)$, so an HIBE$_1$ private key satisfies $(E_1(ID|_k), E_3(T)) \notin \text{Prefix}(E_1(ID^*)|_\ell), E_3(T^*))$. Thus, the simulator can generate a decryption key since it satisfies the constraint of the HIBE security model.

This completes the proof. □

### 3.6 Discussions

**Efficiency Analysis**. We analyze the efficiency of our RHIBE-CS scheme. Let $n$ be the depth of a binary tree and $\ell$ be the level of a hierarchical identity. Note that in order to handle an arbitrary identity string, the depth $n$ must be the length of a hash output which is $2\lambda$. In our RHIBE-CS scheme, a private key is compact since it consists of two HIBE private keys. An update key is roughly composed of $O(rn)$ HIBE private keys since it is composed of HIBE private keys that are related to the cover nodes of the CS method where $r$ is the number of revoked users in a binary tree. A decryption key is composed of $\ell+1$ HIBE private keys since it only needs some HIBE private keys in update keys that match to the path nodes of the private key's identity. A ciphertext is consists of approximately $\ell(n+1)$ HIBE ciphertexts since it is composed of one HIBE ciphertext and HIBE ciphertexts associated with path nodes. The decryption algorithm is efficient because it only requires $\ell+1$ HIBE decryption operations.

**Reducing the Transmission of Update Keys**. In our RHIBE-CS scheme, an update key has a disadvantage of a large size since it is generated for each individual user and it is composed of approximately $O(rn)$ HIBE private keys. In order to improve the inefficiency of the update key transmission, we can consider to delegate the generation of update keys to a cloud server and to store these update keys on the cloud server. It is possible to delegate the generation of update keys to the cloud server since the HIBE$_2$ private key of a user's private keys is only used to generate an update key. If a user with a hierarchical identity $ID|_k$ requests update keys $UK_{ID|_0,T},\ldots,UK_{ID|_{k-1},T}$, the cloud server transmits HIBE$_2$ privates keys that match to path nodes associated with $ID|_k$ instead of transmitting all update keys. Recall that this approach is similar to the procedure of the decryption key algorithm. This approach has the disadvantage of requiring that each user have to query the cloud server, but it can reduce the transmission cost of update keys.

**Reducing the Size of Ciphertexts**. In our RHIBE-CS scheme, a ciphertext is composed of $\ell(n+1)$ HIBE ciphertexts since it consists of one HIBE$_1$ ciphertext and $\ell$ $CT_{PV}$ ciphertexts where each $CT_{PV}$ contains $n+1$ HIBE$_2$ ciphertexts. In this case, the number of HIBE ciphertexts included in the ciphertext is quite large since $n$ is the depth of a binary tree. One way to reduce the ciphertext size of our RHIBE-CS scheme is to change the underlying HIBE scheme to the BBG-HIBE scheme with constant size ciphertext [5]. That is, the ciphertext generation algorithm creates a BBG-HIBE ciphertext by setting a binary tree path as a hierarchical identity, and the update key generation algorithm generates BBG-HIBE private keys by setting the binary tree path up to a curve node as a hierarchical identity. If there is a matching node among the curve set of an update key and the path set of a ciphertext, decryption of the ciphertext can be done by using the key delegation property of the BBG-HIBE scheme. Thus, the ciphertext of this method is very efficient since it consists of only $\ell$ BBG-HIBE ciphertexts, but the size of the update key increases since the private key of the BBG-HIBE scheme contains additional elements for key delegation. The description of this scheme is given in Section 5.

## 4 Revocable HIBE with SD

In this section, we generically construct an RHIBE scheme by combining an HIBE scheme, an HSRE scheme, and the SD method and prove the adaptive security of this scheme.

### 4.1 Subset Difference Method

The subset difference (SD) method is a type of the subset cover framework proposed by Naor et al. [31]. We use the SD method newly defined by Lee et al. [24] that separate the subset definition and key assignment.

The detailed definition of the SD method is given as follows.

**SD.Setup**($N$)**:** Let $N = 2^n$ be the number of leaf nodes. It sets a perfect binary tree $\mathcal{BT}$ of depth $n$ and outputs $\mathcal{BT}$. Note that a user is assigned to a leaf node in $\mathcal{BT}$ and the collection $\mathcal{S}$ of SD is the set of all subsets $\{S_{i,j}\}$ where $v_i, v_j \in \mathcal{BT}$ and $v_j$ is a descendant of $v_i$.

**SD.Assign**($\mathcal{BT}, v$)**:** Let $v$ be the leaf node of $\mathcal{BT}$ that is assigned to a user $ID$. Let $(v_{k_0}, v_{k_1}, \ldots, v_{k_n})$ be a path from the root node $v_{k_0}$ to the leaf node $v_{k_n} = v$. It initializes a path set $PV$ as an empty one. For all $i, j \in \{k_0, \ldots, k_n\}$ such that $v_j$ is a descendant of $v_i$, it adds a subset $S_{i,j}$ defined by two nodes $v_i$ and $v_j$ in the path into $PV$. It outputs the path set $PV = \{S_{i,j}\}$.

**SD.Cover**($\mathcal{BT}, R$)**:** Let $R$ be a revoked set of leaf nodes (or users). It first sets a subtree $\mathcal{T}$ as $ST(\mathcal{BT}, R)$, and then it builds a cover set $CV$ iteratively by removing nodes from $\mathcal{T}$ until $\mathcal{T}$ consists of just a single node as follows:

1. It finds two leaf nodes $v_i$ and $v_j$ in $\mathcal{T}$ such that the least-common-ancestor $v$ of $v_i$ and $v_j$ does not contain any other leaf nodes of $\mathcal{T}$ in its subtree. Let $v_l$ and $v_k$ be the two child nodes of $v$ such that $v_i$ is a descendant of $v_l$ and $v_j$ is a descendant of $v_k$. If there is only one leaf node left, it makes $v_i = v_j$ to the leaf node, $v$ to be the root of $\mathcal{T}$ and $v_l = v_k = v$.
2. If $v_l \neq v_i$, then it adds the subset $S_{l,i}$ to $CV$; likewise, if $v_k \neq v_j$, it adds the subset $S_{k,j}$ to $CV_R$.
3. It removes from $\mathcal{T}$ all the descendants of $v$ and makes $v$ a leaf node.

It outputs the cover set $CV = \{S_{i,j}\}$.

**SD.Match**($CV, PV$)**:** Let $CV = \{S_{i,j}\}$ and $PV = \{S_{i,j}\}$. It finds two subsets $S_{i,j} \in CV$ and $S_{i',j'} \in PV$ such that $(v_i = v_{i'}) \wedge (d_j = d_{j'}) \wedge (v_j \neq v_{j'})$ where $d_j$ is the depth of $v_j$. If two subsets exist, then it outputs $(S_{i,j}, S_{i',j'})$. Otherwise, it outputs $\perp$.

**Lemma 4.1** ( [31])**.** *Given any set of revoked leaves $R$, the* ***SD.Cover*** *algorithm partitions $\mathcal{N} \setminus R$ into at most $2r-1$ disjoint subsets where $\mathcal{N}$ is the set of all leaves and $r$ is the size of $R$.*

**Lemma 4.2.** *Let $PV$ be a path set obtained by the* ***SD.Assign*** *algorithm for a leaf node $v$, and $CV$ be a cover set obtained by the* ***SD.Cover*** *algorithm for a revoked leaves $R$. In this case, the SD method satisfies the following two properties: 1) If $v \notin R$, there is only one subset pair $(S_{i,j}, S_{i',j'})$ of $S_{i,j} \in PV$ and $S_{i',j'} \in CV$ such that $v_i = v_{i'} \wedge d_j = d_{j'} \wedge v_j \neq v_{j'}$. 2) If $v \in R$, there is no subset pair $(S_{i,j}, S_{i',j'})$ of $S_{i,j} \in PV$ and $S_{i',j'} \in CV$ such that $v_i = v_{i'} \wedge d_j = d_{j'} \wedge v_j \neq v_{j'}$.*

*Proof.* We prove the first property. From Lemma 4.1, the **SD.Cover** algorithm partitions all leaf nodes except $R$ into disjoint subsets $S_{i'_1,j'_1}, \ldots, S_{i'_m,j'_m}$. It means that the leaf node $v \notin R$ must belong to one partitioned subset $S_{i',j'}$. According to the definition of the subset $S_{i',j'}$, if $v \in S_{i',j'}$, $v$ belongs to the leaves of the subtree $\mathcal{T}_{i'}$, but not the leaves of the subtree $\mathcal{T}_{j'}$. Thus, the path nodes of $v$ include the root node $v_{i'}$ of $\mathcal{T}_{i'}$, and not the root node $v_{j'}$ of $\mathcal{T}_{j'}$. Also, since $v_{j'}$ is a descendant of $v_{i'}$, there exists a node $v_j$ in the path nodes of $v$ such that the depth $d_j$ of $v_j$ is the same as the depth $d_{j'}$ of $v_{j'}$. Therefore, the subset $S_{i,j}$ defined by $v_i = v_{i'}$ and $v_j$ belongs to $PV$ and satisfies $v_i = v_{i'} \wedge d_j = d_{j'} \wedge v_j \neq v_{j'}$.

Next, we prove the second property. Suppose that there is a subset $S_{i',j'} \in CV$ that satisfies the condition $v_i = v_{i'} \wedge d_j = d_{j'} \wedge v_j \neq v_{j'}$. By the definition of subsets, the leaf node $v$ is the leaf node of the subtree $\mathcal{T}_{i'}$, but not the leaf node of the subtree $\mathcal{T}_{j'}$. Thus, $v$ belongs to the subset $S_{i',j'}$. However, this contradicts that $v$ belongs to $R$. Therefore, there is no subset pair that satisfies the condition if $v \in R$. □

## 4.2 Generic Construction

An RHIBE-SD scheme that is designed by generically combining an HIBE scheme, an HSRE scheme, and the SD method is described as follows:

**RHIBE-SD.Setup**($1^\lambda, L$)**:** Let $\mathcal{I} = \{0,1\}^n$ be the identity space and $L$ be the maximum number of hierarchical depth.

1. It first obtains $MK_{HIBE}, PP_{HIBE}$ by running **HIBE.Setup**$(1^\lambda, L+1)$. It obtains $MK_{HSRE}, PP_{HSRE}$ by running **HSRE.Setup**$(1^\lambda, L)$.
2. It defines a binary tree $\mathcal{BT}$ by running **SD.Setup**$(2^n)$ where an identity $I \in \mathcal{I}$ is uniquely assigned to a leaf node $v$ such that $\text{Label}(v) = I$.
3. It outputs a master key $MK = (MK_{HIBE}, MK_{HSRE})$, a revocation list $RL_\varepsilon = \emptyset$, and public parameters $PP = (PP_{HIBE}, PP_{HSRE}, \mathcal{BT})$.

**RHIBE-SD.GenKey**($ID|_k, SK_{ID|_{k-1}}, PP$)**:** Let $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ where $k \geq 1$.

1. It obtains $SK_{HIBE,ID|_k}$ by running **HIBE.Delegate**$(E_1(ID|_k), SK_{HIBE,ID|_{k-1}}, PP_{HIBE})$, and obtains $SK_{HSRE,ID|_k}$ by running **HSRE.Delegate**$(E_1(ID|_k), SK_{HSRE,ID|_{k-1}}, PP_{HSRE})$.
2. Finally, it outputs a private key $SK_{ID|_k} = (SK_{HIBE,ID|_k}, SK_{HSRE,ID|_k})$.

**RHIBE-SD.UpdateKey**($T, RL_{ID|_{k-1}}, SK_{ID|_{k-1}}, PP$)**:** To generate an update key for $T$, it proceeds as follows:

1. It initializes $RV = \emptyset$. For each tuple $(ID_j, T_j) \in RL_{ID|_{k-1}}$ where $ID_j = (I_1, \ldots, I_{k-1}, I_k)$, it adds a leaf node $v_j \in \mathcal{BT}$ which is associated with $I_k$ such that $\text{Label}(v_j) = I_k$ into $RV$ if $T_j \leq T$. It obtains $CV_{k-1}$ by running **SD.Cover**$(\mathcal{BT}, RV)$.
2. For each $S_{i,j} \in CV_{k-1}$, it sets $(GL, ML) = \text{GMLabels}(S_{i,j})$ and obtains $SK_{HSRE,S_{i,j}}$ by running **HSRE.MakeKey**$((E_2(GL\|T), E_2(ML)), SK_{HSRE,ID|_{k-1}}, PP_{HSRE})$.
3. Finally, it outputs an update key $UK_{ID|_{k-1},T} = \big(CV_{k-1}, \{SK_{HSRE,S_{i,j}}\}_{S_{i,j} \in CV_{k-1}}\big)$.

**RHIBE-SD.DeriveKey**($SK_{ID|_k}, UK_{ID|_0,T}, \ldots, UK_{ID|_{k-1},T}, PP$)**:** Let $SK_{ID|_k} = (SK_{HIBE,ID|_k}, SK_{HSRE,ID|_k})$. To derive a decryption key for $ID|_k = (I_1, \ldots, I_k)$ and $T$, it proceeds as follows:

1. It obtains $SK_{HIBE,ID|_k,T}$ by running **HIBE.Delegate**$((E_1(ID|_k), E_3(T)), SK_{HIBE,ID|_k}, PP_{HIBE})$.
2. For each $u \in [k]$, it proceeds as follows:
   (a) It obtains $PV_u$ by running **SD.Assign**$(\mathcal{BT}, v_u)$ where $v_u$ is a leaf such that $\text{Label}(v_u) = I_u$.
   (b) It finds $(S_{i_u,j_u}, S_{i'_u,j'_u})$ by running **SD.Match**$(CV_{u-1}, PV_u)$. If it fails to find, it returns $\perp$.
   (c) It retrieves $SK_{HSRE,S_{i_u,j_u}}$ from $UK_{ID|_{u-1},T}$ and sets $SK_{HSRE,u} = ((S_{i_u,j_u}, S_{i'_u,j'_u}), SK_{HSRE,S_{i_u,j_u}})$.
3. Finally, it outputs a decryption key $DK_{ID|_k,T} = (SK_{HIBE,ID|_k,T}, SK_{HSRE,1}, \ldots, SK_{HSRE,k})$.

**RHIBE-SD.Encrypt**($ID|_\ell, T, M, PP$)**:** To generate a ciphertext for $ID|_\ell = (I_1, \ldots, I_\ell)$ and $T$, it proceeds as follows:

1. It selects random $R_{2,1}, \ldots, R_{2,\ell}$ and sets $R_1 = M \oplus (R_{2,1} \oplus \cdots \oplus R_{2,\ell})$. It obtains $CT_{HIBE_1}$ by running **HIBE.Encrypt**$((E_1(ID|_\ell), E_3(T)), R_1, PP_{HIBE})$.
2. For each $k \in [\ell]$, it performs the following steps.

   (a) It obtains $PV_k$ by running **SD.Assign**$(\mathcal{BT}, v_k)$ where $v_k$ is a leaf such that $\text{Label}(v_k) = I_k$.
   (b) For each $S_{i,j} \in PV_k$, it sets $(GL, ML) = \text{GMLabels}(S_{i,j})$ and obtains $CT_{HSRE,S_{i,j}}$ by running **HSRE.Encrypt**$(E_1(ID|_{k-1}), (E_2(GL\|T), E_2(ML)), R_{2,k}, PP_{HSRE})$.
   (c) It creates $CT_{PV_k} = \big(PV_k, \{CT_{HSRE,S_{i,j}}\}_{S_{i,j} \in PV_k}\big)$.

3. Finally, it outputs a ciphertext $CT_{ID|_\ell,T} = (CT_{HIBE}, CT_{PV_1}, \ldots CT_{PV_\ell})$.

**RHIBE-SD.Decrypt($CT_{ID|_\ell,T}, DK_{ID'|_\ell,T'}, PP$):** Let $CT_{ID|_\ell,T} = (CT_{HIBE}, CT_{PV_1}, \ldots, CT_{PV_\ell})$ and $DK_{ID'|_\ell,T'} = (SK_{HIBE}, SK_{HSRE,1}, \ldots, SK_{HSRE,\ell})$. If $(ID|_\ell \neq ID'|_\ell) \vee (T \neq T')$, then it returns $\perp$.

1. It first obtains $R_1$ by running **HIBE.Decrypt**$(CT_{HIBE}, SK_{HIBE}, PP_{HIBE})$.
2. For each $k \in [\ell]$, it performs the following steps:
   (a) It gets $S_{i'_k,j'_k}$ from $SK_{HSRE,k} = ((S_{i_k,j_k}, S_{i'_k,j'_k}), SK_{HSRE,S_{i_k,j_k}})$ and retrieves $CT_{HSRE,S_{i'_k,j'_k}}$ from $CT_{PV_k} = (PV_k, \{CT_{HSRE,S_{i',j'}}\})$.
   (b) It obtains $R_{2,k}$ by running **HSRE.Decrypt**$(CT_{HSRE,S_{i'_k,j'_k}}, SK_{HSRE,S_{i_k,j_k}}, PP_{HSRE})$.
3. Finally, it outputs a message $M = R_1 \oplus R_{2,1} \oplus \cdots \oplus R_{2,\ell}$.

**RHIBE-SD.Revoke($ID|_k, T, RL_{ID|_{k-1}}$):** If $(ID|_k, *)$ already exists in $RL_{ID|_{k-1}}$, it outputs $RL_{ID|_{k-1}}$. Otherwise, it adds $(ID|_{k-1}, T)$ to $RL_{ID|_{k-1}}$ and outputs the updated $RL_{ID|_{k-1}}$.

## 4.3 Correctness

In order to show that our RHIBE-SD scheme is correct, we must show that the original message can be recovered when a ciphertext is decrypted with a valid decryption key. First, we show that if the hierarchical identity $ID|_k = (I_1, \ldots, I_k)$ of a user is not revoked in revocation lists $RL_{ID|_0}, \ldots, RL_{ID|_{k-1}}$ before time $T$, a valid decryption key can be derived. It is easy to derive a valid key component $SK_{HIBE,ID|_k,T}$ by using the delegation property of HIBE. For each $u \in [k]$, the path nodes $PV_u$ of a ciphertext can be determined since a private key is associated with $ID|_k$ and the leaf node $v_u$ for $I_u$ in a binary tree is fixed in advance, and we get subsets $S_{i,j} \in CV_{k-1}$ and $S_{i',j'} \in PV_u$ by the first property of the SD method in Lemma 4.2. Thus, we can obtain valid decryption key components $SK_{HSRE,S_{i_1,j_1}}, \ldots, SK_{HSRE,S_{i_k,j_k}}$ associated with $(S_{i_1,j_1}, S_{i'_1,j'_1}), \ldots, (S_{i_k,j_k}, S_{i'_k,j'_k})$ respectively.

Next, we show that a ciphertext for $ID|_\ell$ and $T$ can be decrypted by using a decryption key for $ID'|_\ell$ and $T'$. At this time, a ciphertext $CT_{ID|_\ell,T}$ is composed of a ciphertext $CT_{HIBE}$ and multiple ciphertexts $CT_{PV_1}, \ldots, CT_{PV_\ell}$, and a decryption key $DK_{ID'|_\ell,T'}$ consists of a private key $SK_{HIBE}$ and a number of tuples $((S_{i_1,j_1}, S_{i'_1,j'_1}), SK_{HSRE,S_{i_1,j_1}}), \ldots, ((S_{i_\ell,j_\ell}, S_{i'_\ell,j'_\ell}), SK_{HSRE,S_{i_\ell,j_\ell}})$. If $ID|_\ell = ID'|_\ell$ and $T = T'$ are satisfied, we can obtain the correct $R_1$ by decrypting $CT_{HIBE}$ with the private key $SK_{HIBE}$ from the correctness of the HIBE scheme. Next, for each $k \in [\ell]$, we can derive the correct $R_{2,k}$ by decrypting the ciphertext $CT_{HSRE,S_{i'_k,j'_k}} \in CT_{PV_k}$ with the private key $SK_{HSRE,S_{i_k,j_k}}$ by the correctness of the HSRE scheme. Therefore, we can correctly derive the original message $M$ by computing $R_1 \oplus R_{2,1} \oplus \cdots \oplus R_{2,\ell}$.

## 4.4 Security Analysis

**Theorem 4.3.** *The generic RHIBE-SD scheme is IND-CPA secure if the underlying HIBE and HSRE schemes are IND-CPA secure.*

*Proof.* Let $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$ be the challenge hierarchical identity and $T^*$ be the challenge time. We divide the behavior of an adversary as $\ell + 1$ types: Type-1, $\cdots$, Type-$\ell + 1$, which are defined as follows:

**Type-$\tau$.** An adversary is Type-$\tau$ for $\tau \in \{1, \dots, \ell\}$ if it does not request a private key for $ID|_k \in \text{Prefix}(ID^*|_{\tau-1})$, but it must request a private key for $ID|_k = ID^*|_\tau$.

**Type-$\ell+1$.** An adversary is Type-$\ell+1$ if it does not request a private key for $ID|_k \in \text{Prefix}(ID^*|_\ell)$.

Let $E_\tau$ be the event that $\mathcal{A}$ behaves like Type-$\tau$ adversary. From Lemma 4.4 and Lemma 4.5, we obtain the following result

$$\mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq \sum_{\tau=1}^{\ell+1} \Pr[E_\tau] \mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq O(\ell n^2) \mathbf{Adv}_{\mathcal{B}}^{HSRE}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{HIBE}(\lambda).$$

This completes our proof. □

**Lemma 4.4.** *For the Type-$\tau$ adversary, the generic RHIBE-SD scheme is IND-CPA secure if the HSRE scheme is IND-CPA secure.*

*Proof.* Let $CT^* = (CT^*_{HIBE}, CT^*_{PV_1}, \dots, CT^*_{PV_\ell})$ be the challenge ciphertext and $m = n(n-1)/2$ be the number of subsets in $PV_\tau$. For the security proof, we define hybrid games $\mathbf{H}_0, \mathbf{H}_1, \dots, \mathbf{H}_m$ as follows:

**Game $\mathbf{H}_0$.** This game is the original security game defined in the security model except that the challenge bit $\mu$ is fixed to 0.

**Game $\mathbf{H}_\rho$** In this game, the generation of $CT^*_{PV_\tau} = (PV_\tau, \{CT^*_{HSRE, S_{i_d,j_d}}\}_{S_{i_d,j_d} \in PV_\tau})$ in the challenge ciphertext $CT^*$ is changed. The simulator of this game sets $R'_{2,\tau} = M_1^* \oplus R_1 \bigoplus_{1 \leq k \neq \tau \leq \ell} R_{2,k}$ and $R_{2,\tau} = M_0^* \oplus R_1 \bigoplus_{1 \leq k \neq \tau \leq \ell} R_{2,k}$. The game $\mathbf{H}_\rho$ is similar to the game $\mathbf{H}_{\rho-1}$ except that $CT^*_{HSRE, S_{i_\rho, j_\rho}}$ is an encryption on the value $R'_{2,\tau}$. Specifically, $CT^*_{HSRE, S_{i_d,j_d}}$ for $d \leq \rho$ is an encryption on $R'_{2,\tau}$ and $CT^*_{HSRE, S_{i_d,j_d}}$ for $d > \rho$ is an encryption on $R_{2,\tau}$.

**Game $\mathbf{H}_m$** This game is the original security game in the security model except that the challenge bit $\mu$ is fixed to 1.

Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{H}_{\rho-1}$ and $\mathbf{H}_\rho$ of the RHIBE scheme with a non-negligible advantage. An algorithm $\mathcal{B}$ that attacks the HSRE scheme is initially given public parameters $PP_{HSRE}$ by a challenger $\mathcal{C}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

**Setup:** $\mathcal{B}$ generates $MK_{HIBE}, PP_{HIBE}$ by running **HIBE.Setup**$(1^\lambda, L+1)$. It initializes $RL_\varepsilon = \emptyset$ and gives $PP = (PP_{HIBE}, PP_{HSRE}, \mathcal{BT})$ to $\mathcal{A}$.

**Phase 1:** $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, decryption key, and revocation queries.

- For a private key query with $ID|_k$, $\mathcal{B}$ proceeds as follows: It first generates $SK_{HIBE, ID|_k}$ by running **HIBE.GenKey**$(E_1(ID|_k), MK_{HIBE}, PP_{HIBE})$. It receives $SK_{HSRE, ID|_k}$ from $\mathcal{C}$ by querying a private key for $E_1(ID|_k)$. It gives $SK_{ID|_k} = (SK_{HIBE, ID|_k}, SK_{HSRE, ID|_k})$ to $\mathcal{A}$.
- For an update key query with $ID|_{k-1}$ and time $T$, $\mathcal{B}$ proceeds as follows:
  1. It initializes $RV = \emptyset$. For each $(ID_j, T_j) \in RL$, it adds a leaf node $v_j \in \mathcal{BT}$ into $RV$ if $T_j \leq T$. It obtains $CV_{k-1}$ by running **SD.Cover**$(\mathcal{BT}, RV)$.
  2. For each $S_{i,j} \in CV_{k-1}$, it sets $(GL, ML) = \text{GMLabels}(S_{i,j})$ and receives $SK_{HSRE, S_{i,j}}$ from $\mathcal{C}$ by submitting $E_1(ID|_{k-1})$ and $(E_2(GL \| T), E_2(ML))$.

3. It creates $UK_{ID|_{k-1},T} = \big(CV_{k-1}, \{SK_{HSRE,S_{i,j}}\}_{S_{i,j} \in CV_{k-1}}\big)$ and gives $UK_{ID|_{k-1},T}$ to $\mathcal{A}$.

- For a decryption key query with $ID|_k$ and time $T$, $\mathcal{B}$ proceeds as follows:
  1. It generates $SK_{HIBE,ID|_k,T}$ by running **HIBE.GenKey**$((E_1(ID|_k), E_3(T)), MK_{HIBE}, PP_{HIBE})$.
  2. Next, it obtains update keys $UK_{ID|_0,T}, \ldots, UK_{ID|_{k-1},T}$ by querying its own update key oracle for $(ID|_0, T), \ldots, (ID|_{k-1}, T)$. For each $u \in [k]$, it obtains $SK_{HSRE,u}$ if $I_u$ is not revoked in $UK_{ID|_{u-1},T}$ by following the procedure in the **RHIBE-SD.DeriveKey** algorithm.
  3. If $I_u$ was revoked in $UK_{ID|_{u-1},T}$ for any $u \in [k]$, then it gives $\perp$ to $\mathcal{A}$. Otherwise, t gives $DK_{ID|_k,T} = (SK_{HIBE,ID|_k,T}, SK_{HSRE,1}, \ldots, SK_{HSRE,k})$ to $\mathcal{A}$.
- For a revocation query with a hierarchical identity $ID|_k$ and time $T$, $\mathcal{B}$ adds $(ID|_k, T)$ to $RL_{ID|_{k-1}}$ if $ID|_k$ was not revoked before.

**Challenge**: $\mathcal{A}$ submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$. $\mathcal{B}$ proceeds as follows:

1. It first selects random $R_1, \{R_{2,i}\}_{1 \leq i \neq \tau \leq \ell}$ and computes $R_2 = \bigoplus_{1 \leq i \neq \tau \leq \ell} R_{2,i}$. It sets $R_{2,\tau,0} = M_0^* \oplus R_1 \oplus R_2$ and $R_{2,\tau,1} = M_1^* \oplus R_1 \oplus R_2$. Next, it generates $CT_{HIBE}^*$ by running **HIBE.Encrypt**$((E_1(ID^*|_\ell), E_3(T^*)), R_1, PP_{HIBE})$.
2. For each $k \in [\ell]$, it performs the following steps:
   (a) It obtains $PV_k$ by running **SD.Assign**$(\mathcal{BT}, v_k)$ where a leaf node $v_k$ is associated with $I_k^*$.
   (b) Case $k \neq \tau$: For each $S_{i,j} \in PV_k$, it obtains $(GL, ML) = \text{GMLabels}(S_{i,j})$ and generates $CT_{HSRE,S_{i,j}}^*$ by running **HSRE.Encrypt**$((E_1(ID^*|_{k-1}), (E_2(GL \| T^*), E_2(ML))), R_{2,i}, PP_{HSRE})$.
   It creates $CT_{PV_k}^* = \big(PV_k, \{CT_{HSRE,S_{i,j}}^*\}_{S_{i,j} \in PV_k}\big)$.
   (c) Case $k = \tau$: For each $S_{i_d,j_d} \in PV_\tau = \{S_{i_1,j_1}, \ldots, S_{i_m,j_m}\}$, it obtains $(GL_d, ML_d) = \text{GMLabels}(S_{i_d,j_d})$ and proceeds as follows:
      - If $d < \rho$, then it generates $CT_{HSRE,S_{i_d,j_d}}^*$ by running **HSRE.Encrypt**$(E_1(ID^*|_{\tau-1}), (E_2(GL_d \| T^*), E_2(ML_d)), R_{2,\tau,1}, PP_{HSRE})$.
      - If $d = \rho$, then it receives $CT_{HSRE,S_{i_\rho,j_\rho}}^*$ from $\mathcal{C}$ by submitting a challenge hierarchical identity $E_1(ID^*|_{\tau-1})$, $(E_2(GL_\rho \| T^*), E_2(ML_\rho))$ and challenge messages $R_{2,\tau,0}, R_{2,\tau,1}$.
      - If $d > \rho$, then it generates $CT_{HSRE,S_{i_d,j_d}}^*$ by running **HSRE.Encrypt**$(E_1(ID^*|_{\tau-1}), (E_2(GL_d \| T^*), E_2(ML_d)), R_{2,\tau,0}, PP_{HSRE})$.

   It creates $CT_{PV_\tau}^* = \big(PV_\tau, \{CT_{HSRE,S_{i_d,j_d}}^*\}_{S_{i_d,j_d} \in PV_\tau}\big)$.
3. It gives a challenge ciphertext $CT^* = (CT_{HIBE}^*, CT_{PV_1}^*, \ldots, CT_{PV_\ell}^*)$ to $\mathcal{A}$.

**Phase 2**: Same as Phase 1.
**Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ also outputs $\mu'$.

To complete the proof of this lemma, we must show that the simulator can correctly handle private key, update key, and decryption key queries of the adversary. At this time, the HSRE private key queries and the HSRE challenge ciphertext requested by the simulator to the challenger must satisfy the constraints of the HSRE security model. To simplify the analysis, we first analyze the HSRE challenge ciphertext of the challenge ciphertext and check whether the HSRE private keys of private keys, update keys, and decryption keys satisfy the constraints.

- Challenge ciphertext: The challenge ciphertext consists of multiple HSRE ciphertexts, and the HSRE challenge ciphertext is $CT^*_{HSRE,S_{i_\rho,j_\rho}} \in CT^*_{PV_\tau}$. In this case, the HSRE challenge ciphertext is associated with a hierarchical identity $E_1(ID^*|_{\tau-1})$ and labels $(E_2(GL_\rho \| T^*), E_2(ML_\rho))$ where $GL_\rho, ML_\rho$ are group and member labels in path nodes defined by a leaf node $I^*_\tau$.
- Private key: By the restriction of the adversary type, the adversary of Type-$\tau$ cannot request a private key for $ID^*|_k \wedge k < \tau$, but must request a private key for $ID^*|_\tau$. To simplify the analysis, we divide the private key generation of the simulator into the following cases:
  - Case $ID|_k \notin \text{Prefix}(ID^*|_\ell)$: In this case, the constraint of the HSRE security model is satisfied, so it can generate a private key by using the HSRE delegate key query.
  - Case $ID|_k \in \text{Prefix}(ID^*|_\ell) \wedge k = \tau$: In this case, we have $ID|_k = ID^*|_\tau = (I^*_1, \ldots, I^*_\tau)$, and the simulator requests an HSRE delegate key for $E_1(ID^*|_\tau) = (E_1(I^*_1), \ldots, E_1(I^*_\tau))$ to the challenger. Since the encoding functions satisfy $E_1(x) \neq E_2(y)$ for any $x$ and $y$, it is established that $E_1(I^*|_\tau) \neq E_2(GL_\rho \| T^*)$. Thus, it can generate a private key by using the HSRE delegate key query since the constraint of the HSRE security model is satisfied.
  - Case $ID|_k \in \text{Prefix}(ID^*|_\ell) \wedge k > \tau$: This case also includes the index $i = \tau$, so the same logic as in the previous case is applied. Thus, it can generate a private key by using the HSRE delegate key query.
- Update key: The simulator must be able to generate an update key for any $ID|_{k-1}$ and $T$. To simplify the analysis, we divide the update key generation of the simulator into the following cases:
  - Case $T \neq T^*$: In this case, the hierarchical identity of an HSRE private key in an update key contains a string $E_2(GL_i \| T)$ and the challenge hierarchical identity of the HSRE challenge ciphertext contains a string $E_2(GL_d \| T^*)$, so it is established that $E_2(GL_i \| T) \neq E_2(GL_d \| T^*)$. Thus, it can generate an update key by using the HSRE private key query.
  - Case $T = T^* \wedge ID|_{k-1} \notin \text{Prefix}(ID^*|_{\tau-1})$: In this case, we have $E_1(ID|_{k-1}) \notin \text{Prefix}(E_1(ID^*|_{\tau-1})$. Thus, it can generate an update key since this HSRE private key query is allowed in the HSRE security model.
  - Case $T = T^* \wedge ID|_{k-1} \in \text{Prefix}(ID^*|_{\tau-1}) \wedge k \leq \tau - 1$: In this case, the HSRE private key of an update key is associated with $E_1(ID|_{k-1})$ and $(E_2(GL_i \| T^*), E_2(ML_i))$, and we have $E_2(GL_i \| T^*) \neq E_1(I^*_k)$ by the encoding functions. This means that $(E_1(ID|_{k-1}), E_2(GL_i \| T^*)) \notin \text{Prefix}(E_1(ID^*|_{\tau-1}))$. Thus, it can generate an update key since this HSRE private key query is allowed in the HSRE security model.
  - Case $T = T^* \wedge ID|_{k-1} \in \text{Prefix}(ID^*|_{\tau-1}) \wedge k = \tau$: In this case, by the constraints of the RHIBE security model, the identity $I^*_\tau$ must be revoked in this update key. From the second property of the SD method in Lemma 4.2, there is no subset tuple $(S_{i,j} \in CV_{\tau-1}, S_{i',j'} \in PV_\tau)$ such that $v_i = v_{i'} \wedge d_j = d_{j'} \wedge v_j \neq v_{j'}$. That is, $GL_i \neq GL_\rho$ or $GL_i = GL_\rho \wedge ML_i = ML_\rho$ where $(GL_\rho, ML_\rho) = \text{GMLabels}(S_{i',j'})$ and $(GL_i, ML_i) = \text{GMLabels}(S_{i,j})$. This means that $(E_1(ID^*|_{\tau-1}), E_2(GL_i \| T^*))$ $\neq (E_1(ID^*|_{\tau-1}), E_2(GL_\rho \| T^*))$ or $(E_1(ID^*|_{\tau-1}), E_2(GL_i \| T^*)) = (E_1(ID^*|_{\tau-1}), E_2(GL_\rho \| T^*)) \wedge (E_2(ML_i)) = (E_2(ML_\rho))$. Thus, it can generate an update key by using the HSRE private key query.
- Decryption key: In the case of the decryption key, if the simulator can correctly generate update keys, then simulator can also correctly generate a decryption key.

Finally, we analyze the advantage of the simulation. Let $S_{\mathcal{A}}^{H_i}$ be the event that $\mathcal{A}$ outputs 0 in a game $\mathbf{H}_i$. Since this lemma consists of a number of hybrid games, we get the following result

$$\mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq \frac{1}{2}\left|\Pr[S_{\mathcal{A}}^{H_0}] - \Pr[S_{\mathcal{A}}^{H_m}]\right| \leq \frac{1}{2}\sum_{\rho=1}^{m}\left|\Pr[S_{\mathcal{A}}^{H_{\rho-1}}] - \Pr[S_{\mathcal{A}}^{H_\rho}]\right| \leq O(n^2)\mathbf{Adv}_{\mathcal{B}}^{HSRE}(\lambda).$$

This completes our proof. □

**Lemma 4.5.** *For the Type-$\ell+1$ adversary, the generic RHIBE-SD scheme is IND-CPA secure if the HIBE scheme is IND-CPA secure.*

*Proof.* Suppose there exists a Type-$\ell+1$ adversary $\mathcal{A}$ that attacks the RHIBE scheme with a non-negligible advantage. An algorithm $\mathcal{B}$ that attacks the HIBE scheme is initially given public parameters $PP_{HIBE}$ by a challenger $\mathcal{C}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

**Setup:** $\mathcal{B}$ generates $MK_{HSRE}, PP_{HSRE}$ by running **HSRE.Setup**$(1^\lambda, L)$. It initializes $RL_\varepsilon = \emptyset$ and gives $PP = (PP_{HIBE}, PP_{HSRE}, \mathcal{BT})$ to $\mathcal{A}$.

**Phase 1:** $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, decryption key, and revocation queries.

- For a private key query with $ID|_k$, $\mathcal{B}$ proceeds as follows: It receives $SK_{HIBE,ID|_k}$ from $\mathcal{C}$ by querying a private key for $ID|_k$. It generates $SK_{HSRE,ID|_k}$ by running **HSRE.GenKey**$(E_1(ID|_k), MK_{HSRE}, PP_{HSRE})$. It gives $SK_{ID|_k} = (SK_{HIBE,ID|_k}, SK_{HSRE,ID|_k})$ to $\mathcal{A}$.
- For an update key query with $ID|_{k-1}$ and $T$, $\mathcal{B}$ proceeds as follows: It generates $UK_{ID|_{k-1},T}$ by running **RHIBE-SD.UpdateKey** algorithm since it knows $MK_{HSRE}$. It gives $UK_{ID|_{k-1},T}$ to $\mathcal{A}$.
- For a decryption key query with $ID|_k$ and $T$, $\mathcal{B}$ proceeds as follows:
  1. It receives $SK_{HIBE,ID|_k,T}$ from $\mathcal{C}$ by querying a private key for $(ID|_k, T)$.
  2. Next, it obtains update keys $UK_{ID|_0,T}, \ldots, UK_{ID|_{k-1},T}$ by querying its own update key oracle for $(ID|_0, T), \ldots, (ID|_{k-1}, T)$. For each $u \in [k]$, it obtains $SK_{HSRE,u}$ if $I_u$ is not revoked in $UK_{ID|_{u-1},T}$ by following the procedure in the **RHIBE-SD.DeriveKey** algorithm.
  3. If $I_u$ was revoked in $UK_{ID|_{u-1},T}$ for any $u \in [k]$, then it gives $\perp$ to $\mathcal{A}$. Otherwise, t gives $DK_{ID|_k,T} = (SK_{HIBE,ID|_k,T}, SK_{HSRE,1}, \ldots, SK_{HSRE,k})$ to $\mathcal{A}$.
- For a revocation query with a hierarchical identity $ID|_k$ and time $T$, $\mathcal{B}$ adds $(ID|_k, T)$ to $RL_{ID|_{k-1}}$ if $ID|_k$ was not revoked before.

**Challenge**: $\mathcal{A}$ submits a challenge hierarchical identity $ID^*|_\ell$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$. $\mathcal{B}$ proceeds as follows:

1. It first selects random $\{R_{2,i}\}_{1\leq i\leq \ell}$ and computes $R_2 = \bigoplus_{1\leq i\leq \ell} R_{2,i}$. It sets $R_{1,0} = M_0^* \oplus R_2, R_{1,1} = M_1^* \oplus R_2$. It receives $CT_{HIBE}^*$ from $\mathcal{C}$ by submitting challenge identity $(E_1(ID^*|_\ell), E_3(T^*))$ and challenge messages $R_{1,0}, R_{1,1}$.
2. For each $k \in [\ell]$, it proceeds as follows:
   (a) It obtains $PV_k$ by running **SD.Assign**$(\mathcal{BT}, v_k)$ where a leaf node $v_k$ is associated with $I_k^*$.

(b) For each $S_{i,j} \in PV_k$, it obtains $(GL, ML) = \text{GMLabels}(S_{i,j})$ and generates $CT^*_{HSRE,S_{i,j}}$ by running **HSRE.Encrypt**$(E_1(ID^*|_{k-1}), (E_2(GL \| T^*), E_2(ML)), R_{2,k}, PP_{HSRE})$.

(c) It creates $CT^*_{PV_k} = \big(PV_k, \{CT^*_{HSRE,S_{i,j}}\}_{S_{i,j} \in PV_k}\big)$.

3. It gives a challenge ciphertext $CT^* = (CT^*_{HIBE}, CT^*_{PV_1}, \ldots, CT^*_{PV_\ell})$ to $\mathcal{A}$.

**Phase 2**: Same as Phase 1.
**Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ also outputs $\mu'$.

To complete the proof of this lemma, we must show that the simulator can correctly generate private keys, update keys, decryption keys, and the challenge ciphertext while satisfying the constraints of the HIBE security model. Since the simulator sets the HSRE scheme by itself, the simulator can proceed with the simulation of the RHIBE queries while satisfying the constraints of the HIBE security model in the same way as Lemma 3.4. We omit the details of the analysis. □

### 4.5 Discussions

**Efficiency Analysis**. We analyze the efficiency of our RHIBE-SD scheme. Let $n$ be the depth of a binary tree and $\ell$ be the depth of a hierarchical identity. In our RHIBE-SD scheme, a private key is compact since it consists of an HIBE private key and an HSRE private key. An update key is roughly composed of $O(r)$ HSRE private keys since it is composed of HSRE private keys that are related to the cover nodes of the SD method where $r$ is the number of revoked users in a binary tree. A decryption key is composed of one HIBE private key and $\ell$ HSRE private keys since it only needs some HSRE private keys in update keys that match to the path nodes of the private key's identity. A ciphertext is consists of one HIBE ciphertext and $O(\ell n^2)$ HSRE ciphertexts since it is composed of one HIBE ciphertext and HSRE ciphertexts associated with path nodes. The decryption algorithm is efficient because it only requires one HIBE decryption operation and $\ell$ HSRE decryption operations.

## 5 RHIBE-CS with Shorter Ciphertexts

In this section, we construct an RHIBE scheme with shorter ciphertexts by combining HIBE schemes with constant size ciphertext and the CS method and prove the adaptive security of this scheme.

### 5.1 Generic Construction

An RHIBE-CS scheme with shorter ciphertexts that is designed by generically combining HIBE schemes with constant-size ciphertext and the CS method is described as follows:

**RHIBE-CS.Setup**$(1^\lambda, L)$**:** Let $\mathcal{I} = \{0,1\}^n$ be the identity space and $L$ be the maximum depth of a hierarchical identity.

1. It first obtains $MK_{HIBE_1}, PP_{HIBE_1}$ by running **HIBE.Setup**$(1^\lambda, L+1)$. It obtains $MK_{HIBE_2}, PP_{HIBE_2}$ by running **HIBE.Setup**$(1^\lambda, L+n)$.
2. It defines a binary tree $\mathcal{BT}$ by running **CS.Setup**$(2^n)$ where an identity $I \in \mathcal{I}$ is uniquely assigned to a leaf node $v$ such that $\text{Label}(v) = I$.

3. It outputs a master key $MK = (MK_{HIBE_1}, MK_{HIBE_2})$, a revocation list $RL_\varepsilon = \emptyset$, and public parameters $PP = (PP_{HIBE_1}, PP_{HIBE_2}, \mathcal{BT})$.

**RHIBE-CS.GenKey($ID|_k, SK_{ID|_{k-1}}, PP$):** Let $ID|_k = (I_1, \dots, I_k) \in \mathcal{I}^k$ where $k \geq 1$. Let $SK_{ID|_0} = MK$ where $SK_{HIBE_1,ID|_0} = MK_{HIBE_1}$ and $SK_{HIBE_2,ID|_0} = MK_{HIBE_2}$.

1. It obtains $SK_{HIBE_1,ID|_k}$ by running **HIBE.Delegate**$(E_1(ID|_k), SK_{HIBE_1,ID|_{k-1}}, PP_{HIBE_1})$, and obtains $SK_{HIBE_2,ID|_k}$ by running **HIBE.Delegate**$(E_1(ID|_k), SK_{HIBE_2,ID|_{k-1}}, PP_{HIBE_2})$.
2. Finally, it outputs a private key $SK_{ID|_k} = (SK_{HIBE_1,ID|_k}, SK_{HIBE_2,ID|_k})$.

**RHIBE-CS.UpdateKey($T, RL_{ID|_{k-1}}, SK_{ID|_{k-1}}, PP$):** Let $SK_{ID|_k} = (SK_{HIBE_1,ID|_k}, SK_{HIBE_2,ID|_k})$. To generate an update key for $ID|_{k-1}$ and $T$, it proceeds as follows:

1. It initializes $RV = \emptyset$. For each tuple $(ID_j, T_j) \in RL_{ID|_{k-1}}$ where $ID_j = (I_1, \dots, I_{k-1}, I_k)$, it adds a leaf node $v_j \in \mathcal{BT}$ which is associated with $I_k$ such that $\text{Label}(v_j) = I_k$ into $RV$ if $T_j \leq T$.
2. It obtains $CV_{k-1} = \{S_{i_1}, \dots, S_{i_m}\}$ by running **CS.Cover**$(\mathcal{BT}, RV)$.
3. For each $S_{i_u} \in CV_{k-1}$, it performs the following steps:
   (a) It obtains path nodes $PN = \{v_{i_0}, \dots, v_{i_u}\}$ from the root node $v_{i_0} = v_0$ to the node $v_{i_u}$ of the subset $S_{i_u}$.
   (b) It encodes a node identity vector $NID_u = (L_0 \| T, \dots, L_u \| T)$ where $L_i = \text{Label}(v_{i_u})$.
   (c) It obtains $SK_{HIBE_2,S_u}$ by running **HIBE.Delegate**$((E_1(ID|_{k-1}), E_2(NID_u)), SK_{HIBE_2,ID|_{k-1}}, PP_{HIBE_2})$.
4. Finally, it outputs an update key $UK_{ID|_{k-1},T} = \big(CV_{k-1}, \{SK_{HIBE_2,S_i}\}_{S_i \in CV_{k-1}}\big)$.

**RHIBE-CS.DeriveKey($SK_{ID|_k}, UK_{ID|_0,T}, \dots, UK_{ID|_{k-1},T}, PP$):** Let $SK_{ID|_k} = (SK_{HIBE_1,ID|_k}, SK_{HIBE_2,ID|_k})$. To derive a decryption key for $ID|_k = (I_1, \dots, I_k)$ and $T$, it proceeds as follows:

1. It obtains $SK_{HIBE_1,ID|_k,T}$ by running **HIBE.Delegate**$((E_1(ID|_k), E_3(T)), SK_{HIBE_1,ID|_k}, PP_{HIBE_1})$.
2. For each $u \in [k]$, it proceeds as follows:
   (a) It obtains $PV_u$ by running **CS.Assign**$(\mathcal{BT}, v_u)$ where $v_u$ is a leaf such that $\text{Label}(v_u) = I_u$.
   (b) It finds $(S_{i_u}, S_{i_u})$ by running **CS.Match**$(CV_{u-1}, PV_u)$. If it fails to find, it returns $\perp$.
   (c) It retrieves $SK_{HIBE_2,S_{i_u}}$ from $UK_{ID|_{u-1},T}$ and sets $SK_{HIBE_2,u} = (S_{i_u}, SK_{HIBE_2,S_{i_u}})$.
3. Finally, it outputs a decryption key $DK_{ID|_k,T} = (SK_{HIBE_1,ID|_k,T}, SK_{HIBE_2,1}, \dots, SK_{HIBE_2,k})$.

**RHIBE-CS.Encrypt($ID|_\ell, T, M, PP$):** To generate a ciphertext for $ID|_\ell = (I_1, \dots, I_\ell)$ and $T$, it proceeds as follows:

1. It selects random $R_{2,1}, \dots, R_{2,\ell}$ and sets $R_1 = M \oplus (R_{2,1} \oplus \cdots \oplus R_{2,\ell})$. It obtains $CT_{HIBE_1}$ by running **HIBE.Encrypt**$((E_1(ID|_\ell), E_3(T)), R_1, PP_{HIBE_1})$.
2. For each $k \in [\ell]$, it performs the following steps.
   (a) It obtains $PV_k = \{S_{j_0}, \dots, S_{j_n}\}$ by running **CS.Assign**$(\mathcal{BT}, v_k)$ where $v_k$ is a leaf such that $\text{Label}(v_k) = I_k$.
   (b) It encodes a path identity vector $PID_k = (L_{j_0} \| T, \dots, L_{j_n} \| T)$ where $L_{j_u} = \text{Label}(S_{j_u})$.
   (c) It obtains $CT_{HIBE_2,k}$ by running **HIBE.Encrypt**$((E_1(ID|_{k-1}), E_2(PID_k)), R_{2,k}, PP_{HIBE_2})$.

3. Finally, it outputs a ciphertext $CT_{ID|_\ell,T} = (CT_{HIBE_1}, CT_{HIBE_2,1}, \dots CT_{HIBE_2,\ell})$.

**RHIBE-CS.Decrypt**($CT_{ID|_\ell,T}, DK_{ID'|_\ell,T'}, PP$)**:** Let $CT_{ID|_\ell,T} = (CT_{HIBE_1}, CT_{HIBE_2,1}, \dots, CT_{HIBE_2,\ell})$ and $DK_{ID'|_\ell,T'} = (SK_{HIBE_1}, SK_{HIBE_2,1}, \dots, SK_{HIBE_2,k})$. If $(ID|_\ell \neq ID'|_\ell) \vee (T \neq T')$, then it returns $\perp$.

1. It first obtains $R_1$ by running **HIBE.Decrypt**$(CT_{HIBE_1}, SK_{HIBE_1}, PP_{HIBE_1})$.
2. For each $k \in [\ell]$, it performs the following steps:
   (a) Let $(E_1(ID|_{k-1}), E_2(NID_u))$ be the identity vector of $SK_{HIBE_2,k}$ and $(E_1(ID|_{k-1}), E_2(PID_k))$ be the identity vector of $CT_{HIBE_2,k}$.
   (b) It derives $SK'_{HIBE_2,k}$ by running **HIBE.Delegate**$((E_1(ID|_{k-1}), E_2(PID_k)), SK_{HIBE_2,k}, PP_{HIBE_2})$ since $NID_u \in \text{Prefix}(PID_k)$.
   (c) It obtains $R_{2,k}$ by running **HIBE.Decrypt**$(CT_{HIBE_2,k}, SK'_{HIBE_2,k}, PP_{HIBE_2})$.
3. Finally, it outputs a message $M = R_1 \oplus R_{2,1} \oplus \cdots \oplus R_{2,\ell}$.

**RHIBE-CS.Revoke**($ID|_k, T, RL_{ID|_{k-1}}$)**:** If $(ID|_k, *)$ already exists in $RL_{ID|_{k-1}}$, it outputs $RL_{ID|_{k-1}}$. Otherwise, it adds $(ID|_{k-1}, T)$ to $RL_{ID|_{k-1}}$ and outputs the updated $RL_{ID|_{k-1}}$.

## 5.2 Security Analysis

**Theorem 5.1.** *The generic RHIBE-CS scheme with shorter ciphertexts is IND-CPA secure if the underlying HIBE schemes are IND-CPA secure.*

*Proof.* Let $ID^*|_\ell = (I_1^*, \dots, I_\ell^*)$ be the challenge hierarchical identity and $T^*$ be the challenge time. We divide the behavior of an adversary as $\ell+1$ types: Type-1, $\cdots$, Type-$\ell+1$, which are defined as follows:

**Type-$\tau$.** An adversary is Type-$\tau$ for $\tau \in \{1, \dots, \ell\}$ if it does not request a private key for $ID|_k \in \text{Prefix}(ID^*|_{\tau-1})$, but it must request a private key for $ID|_k = ID^*|_\tau$.

**Type-$\ell+1$.** An adversary is Type-$\ell+1$ if it does not request a private key for $ID|_k \in \text{Prefix}(ID^*|_\ell)$.

Let $E_\tau$ be the event that $\mathcal{A}$ behaves like Type-$\tau$ adversary. From Lemma 5.2 and Lemma 5.3, we obtain the following result

$$\mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq \sum_{\tau=1}^{\ell+1} \Pr[E_\tau] \mathbf{Adv}_{\mathcal{A}}^{RHIBE}(\lambda) \leq (\ell+1)\mathbf{Adv}_{\mathcal{B}}^{HIBE}(\lambda).$$

This completes our proof. □

**Lemma 5.2.** *For the Type-$\tau$ adversary such that $\tau \in \{1, \dots, \ell\}$, the generic RHIBE-CS scheme with shorter ciphertexts is IND-CPA secure if the HIBE scheme is IND-CPA secure.*

*Proof.* Suppose there exists a Type-$\tau$ adversary $\mathcal{A}$ that attacks the RHIBE scheme with a non-negligible advantage. An algorithm $\mathcal{B}$ that attacks the HIBE scheme is initially given public parameters $PP_{HIBE_2}$ by a challenger $\mathcal{C}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

**Setup:** $\mathcal{B}$ generates $MK_{HIBE_1}, PP_{HIBE_1}$ by running **HIBE.Setup**$(1^\lambda, L+1)$. It initializes $RL_\varepsilon = \emptyset$ and gives $PP = (PP_{HIBE_1}, PP_{HIBE_2}, \mathcal{BT})$ to $\mathcal{A}$.

**Phase 1:** $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, decryption key, and revocation queries.

- For a private key query with $ID|_k$, $\mathcal{B}$ proceeds as follows: It generates $SK_{HIBE_1,ID|_k}$ by running **HIBE.GenKey**$(E_1(ID|_k), MK_{HIBE_1}, PP_{HIBE_1})$. It receives $SK_{HIBE_2,ID|_k}$ from $\mathcal{C}$ by querying a private key for $E_1(ID|_k)$. It gives $SK_{ID|_k} = (SK_{HIBE_1,ID|_k}, SK_{HIBE_2,ID|_k})$ to $\mathcal{A}$.

- For an update key query with $ID|_{k-1}$ and time $T$, $\mathcal{B}$ proceeds as follows:

  1. It initializes $RV = \emptyset$. For each $(ID_j, T_j) \in RL_{ID|_{k-1}}$, it adds a leaf node $v_j \in \mathcal{BT}$ into $RV$ if $T_j \leq T$. It obtains $CV_{k-1}$ by running **CS.Cover**$(\mathcal{BT}, RV)$.
  2. For each $S_{i_u} \in CV_{k-1}$, it proceeds as follows:
     (a) It obtains path nodes $PN = \{v_{i_0}, \dots, v_{i_u}\}$ of the node $v_{i_u}$ related to the subset $S_{i_u}$.
     (b) It encodes a node identity vector $NID_u = (L_0 \| T, \dots, L_u \| T)$ where $L_i = \text{Label}(v_{i_u})$.
     (c) It receives $SK_{HIBE_2,S_{i_u}}$ from $\mathcal{C}$ by querying a private key for $(E_1(ID|_{k-1}), E_2(NID_u))$.
  3. It creates $UK_{ID|_{k-1},T} = \big(CV_{k-1}, \{SK_{HIBE_2,S_i}\}_{S_i \in CV_{k-1}}\big)$ and gives $UK_{ID|_{k-1},T}$ to $\mathcal{A}$.

- For a decryption key query with $ID|_k$ and time $T$, $\mathcal{B}$ proceeds as follows:

  1. It generates $SK_{HIBE_1,ID|_k,T}$ by running **HIBE.GenKey**$((E_1(ID|_k), E_3(T)), MK_{HIBE_1}, PP_{HIBE_1})$.
  2. Next, it obtains update keys $UK_{ID|_0,T}, \dots, UK_{ID|_{k-1},T}$ by querying its own update key oracle for $(ID|_0, T), \dots, (ID|_{k-1}, T)$.
  3. For each $u \in [k]$, it obtains $SK_{HIBE_2,u}$ if $I_u$ is not revoked in $UK_{ID|_{u-1},T}$ by following the procedure in the **RHIBE-CS.DeriveKey** algorithm.
  4. If $I_u$ was revoked in $UK_{ID|_{u-1},T}$ for any $u \in [k]$, then it gives $\perp$ to $\mathcal{A}$. Otherwise, t gives $DK_{ID|_k,T} = (SK_{HIBE_1,ID|_k,T}, SK_{HIBE_2,1}, \dots, SK_{HIBE_2,k})$ to $\mathcal{A}$.

- For a revocation query with a hierarchical identity $ID|_k$ and time $T$, $\mathcal{B}$ adds $(ID|_k, T)$ to $RL_{ID|_{k-1}}$ if $ID|_k$ was not revoked before.

**Challenge**: $\mathcal{A}$ submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*, \dots, I_\ell^*)$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$. $\mathcal{B}$ proceeds as follows:

1. It first selects random $R_1, \{R_{2,i}\}_{1 \leq i \neq \tau \leq \ell}$ and computes $R_2 = \bigoplus_{1 \leq i \neq \tau \leq \ell} R_{2,i}$. It sets $R_{2,\tau,0} = M_0^* \oplus R_1 \oplus R_2$ and $R_{2,\tau,1} = M_1^* \oplus R_1 \oplus R_2$. Next, it generates $CT_{HIBE_1}^*$ by running **HIBE.Encrypt**$((E_1(ID^*|_\ell), E_3(T^*)), R_1, PP_{HIBE_1})$.

2. For each $k \in [\ell]$, it performs the following steps:

   (a) It obtains $PV_k = \{S_{j_0}, \dots, S_{j_n}\}$ by running **CS.Assign**$(\mathcal{BT}, v_k)$ where a leaf node $v_k$ is associated with $I_k^*$.
   (b) It encodes a path identity vector $PID_k^* = (L_{j_0} \| T^*, \dots, L_{j_n} \| T^*)$ where $L_{j_u} = \text{Label}(S_{j_u})$.
   (c) Case $k \neq \tau$: It generates $CT_{HIBE_2,k}^*$ by running **HIBE.Encrypt**$((E_1(ID^*|_{k-1}), E_2(PID_k^*)), R_{2,i}, PP_{HIBE_2})$.
   (d) Case $k = \tau$: It receives $CT_{HIBE_2,\tau}^*$ from $\mathcal{C}$ by submitting an hierarchical identity $(E_1(ID^*|_{\tau-1}), E_2(PID_\tau^*))$ and challenge messages $R_{2,\tau,0}, R_{2,\tau,1}$.

3. It gives a challenge ciphertext $CT^* = (CT_{HIBE_1}^*, CT_{HIBE_2,1}^*, \dots, CT_{HIBE_2,\ell}^*)$ to $\mathcal{A}$.

**Phase 2**: Same as Phase 1.
**Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$. $\mathcal{B}$ also outputs $\mu'$.

To finish the proof, we show that the simulator can correctly handle private key, update key, and decryption key queries of the adversary. At this time, the HIBE$_2$ private key queries and the HIBE$_2$ challenge ciphertext requested by the simulator to the challenger must satisfy the constraint of the HIBE security model. To simplify the analysis, we first analyze the HIBE$_2$ challenge ciphertext of the challenge ciphertext and check whether the HIBE$_2$ private keys of private keys, update keys, and decryption keys satisfy the constraint.

- Challenge ciphertext: The challenge ciphertext consists of multiple HIBE$_2$ ciphertexts, and the HIBE$_2$ challenge ciphertext is $CT_{HIBE_2,\tau}$. In this case, the HIBE$_2$ challenge ciphertext is associated with $(E_1(ID^*|_{\tau-1}), E_2(PID^*_\tau))$ where $PID^*_\tau = (L_{j_0}\|T^*, \ldots, L_{j_n}\|T^*)$.
- Private key: Since the adversary is Type-$\tau$, he cannot request a private key for $ID^*|_k$ and $k < \tau$, but must request a private key for $ID^*|_\tau$. To simplify the analysis, we divide the private key generation of the simulator into the following cases:
  - Case $ID|_k \notin \text{Prefix}(ID^*|_\ell)$: In this case, the constraint of the HIBE$_2$ security model is satisfied, so it can generate a private key by using the HIBE$_2$ private key query.
  - Case $ID|_k \in \text{Prefix}(ID^*|_\ell) \wedge k = \tau$: In this case, we have $ID|_k = ID^*|_\tau = (I^*_1, \ldots, I^*_\tau)$, and the simulator requests an HIBE$_2$ private key for $E_1(ID^*|_\tau) = (E_1(I^*_1), \ldots, E_1(I^*_\tau))$ to the challenger. Since the encoding functions satisfy $E_1(x) \neq E_2(y)$ for any $x$ and $y$, it is established that $E_1(I^*|_\tau) \neq E_2(PID^*_\tau)$. Thus, it can generate a private key by using the HIBE$_2$ private key query since the constraint of the HIBE$_2$ security model is satisfied.
  - Case $ID|_k \in \text{Prefix}(ID^*|_\ell) \wedge k > \tau$: This case also includes the index $i = \tau$, so the same logic as in the previous case is applied. Thus, it can generate a private key by using the HIBE$_2$ private key query.
- Update key: The simulator must be able to generate an update key for any $ID|_{k-1}$ and $T$. To simplify the analysis, we divide the update key generation of the simulator into the following cases:
  - Case $T \neq T^*$: In this case, the hierarchical identity of an HIBE$_2$ private key in an update key contains a string $E_2(NID_u)$ that contains $T$ and the challenge hierarchical identity of the HIBE$_2$ challenge ciphertext contains a string $E_2(PID^*_\tau)$ that contains $T^*$, so it is established that $E_2(NID_u) \neq E_2(PID^*_\tau)$ from $T \neq T^*$. Thus, it can generate an update key by using the HIBE$_2$ private key query.
  - Case $T = T^* \wedge ID|_{k-1} \notin \text{Prefix}(ID^*|_{\tau-1})$: In this case, it is established that $(E_1(ID|_{k-1}), E_2(NID_u)) \notin \text{Prefix}((E_1(ID^*|_{\tau-1}), E_2(PID^*_\tau)))$ in the HIBE$_2$ scheme. Thus, since this HIBE$_2$ private key query is allowed in the HIBE security model, it can generate an update key.
  - Case $T = T^* \wedge ID|_{k-1} \in \text{Prefix}(ID^*|_{\tau-1}) \wedge k \leq \tau - 1$: In this case, the HIBE$_2$ private key of an update key is associated with a string $(E_1(ID|_{k-1}), E_2(NID_u))$, and we have $E_2(NID_u) \neq E_1(I^*_k)$ by the encoding function. This means that $(E_1(ID|_{k-1}), E_2(NID_u)) \notin \text{Prefix}((E_1(ID^*|_{\tau-1}), E_2(PID^*_\tau)))$. Thus, it can generate an update key since this HIBE$_2$ private key query is allowed in the HIBE security model.
  - Case $T = T^* \wedge ID|_{k-1} \in \text{Prefix}(ID^*|_{\tau-1}) \wedge k = \tau$: In this case, by the constraints of the RHIBE security model, the identity $I^*_\tau$ must be revoked in this update key. From the second property of

the CS method in Lemma 3.1, we have that $PV_\tau \cap CV_{\tau-1}$ is an empty set. That is, a label $L_i$ of the cover $CV_{\tau-1}$ is different from the label $L_{j_\rho}$ of the path node $PV_\tau$, so we have $NID_u \neq PID^*_\tau$. This means that $(E_1(ID^*|_{\tau-1}), E_2(NID_u)) \neq (E_1(ID^*|_{\tau-1}), E_2(PID^*_\tau))$. Thus, it can generate an update key by using the HIBE$_2$ private key query.

- Decryption key: In the case of the decryption key, if the simulator can correctly generate update keys, then simulator can also correctly generate a decryption key.

This completes our proof. □

**Lemma 5.3.** *For the Type-$\ell+1$ adversary, the generic RHIBE scheme with shorter ciphertexts is IND-CPA secure if the HIBE scheme is IND-CPA secure.*

We omit the proof of this lemma since it is almost the same as that of Lemma 3.4.

## 6 Instantiations

In this section, we look at how to instantiate our generic RHIBE constructions in bilinear groups or lattices.

### 6.1 Construction from Bilinear Maps

We first instantiate an RHIBE-CS scheme in bilinear groups. To instantiate an RHIBE-CS scheme that provides the selective security, we can choose the HIBE scheme of Boneh and Boyen [4] that provides the selective security. To instantiate an RHIBE-CS scheme that provides the adaptive security, we can choose the HIBE scheme of Waters [40] that provides the adaptive security. In these HIBE schemes, a private key is composed of $O(\ell)$ group elements, and a ciphertext is composed of $O(\ell)$ group elements. Thus, the private key, update key, and ciphertext of the RHIBE-CS scheme are composed of $O(\ell), O(\ell rn)$, and $O(\ell^2 n)$ group elements, respectively.

Next, we instantiate an RHIBE-SD scheme in bilinear groups. To instantiate an RHIBE-SD scheme that provides the selective security, we choose the HIBE scheme of Boneh and Boyen that provides the selective security, and the HSRE scheme that provides the selective security in Section 6.3. Although there is still no HSRE scheme that provides the adaptive security, we expect that an adaptively secure HSRE scheme can be made without difficulty by combining the adaptively secure SRE scheme of Lee and Park [26] and the adaptively secure HIBE of Lewko and Waters [28]. The private key and ciphertext of the HSRE scheme in Section 6.3 consist of $O(\ell)$ and $O(\ell)$ group elements, respectively. Thus, the private key, update key, and ciphertext of the RHIBE-SD scheme are composed of $O(\ell), O(\ell r)$, and $O(\ell^2 n^2)$ group elements, respectively.

Finally, we instantiate an RHIBE-CS scheme with shorter ciphertexts in bilinear groups. We choose the HIBE scheme of Boneh, Boyen, and Goh [5], which provides the selective security with constant size ciphertext, or the HIBE scheme of Lewko and Waters [28], which provides the adaptive security with constant size ciphertext. In these HIBE schemes, a private key is composed of $O(L)$ group elements, and a ciphertext is composed of $O(1)$ group elements. Thus, the private key, update key, and ciphertext of the RHIBE-CS scheme are composed of $O(L+n), O((L+n)rn)$, and $O(\ell)$ group elements, respectively. In Table 1, we compare our RHIBE schemes with other RHIBE scheme in bilinear groups.

Table 1: Comparison of RHIBE schemes in bilinear groups

| Scheme | PP Size | SK Size | UK Size | CT Size | Model | Generic |
|---|---|---|---|---|---|---|
| SE [34] | $O(L)$ | $O(\ell^2 \log N)$ | $O(\ell r \log \frac{N}{r})$ | $O(\ell)$ | SE-IND | No |
| SE (CS) [36] | $O(L)$ | $O(L \log N)$ | $O(Lr \log \frac{N}{r})$ | $O(1)$ | SE-IND | No |
| SE (SD) [36] | $O(L)$ | $O(L \log^2 N)$ | $O(Lr)$ | $O(1)$ | SRL-IND | No |
| RLPL [33] | $O(1)$ | $O(\ell \log N)$ | $O(\ell r \log \frac{N}{r})$ | $O(\ell)$ | SE-IND | No |
| LP (CS) [27] | $O(1)$ | $O(\log N)$ | $O(\ell + r \log \frac{N}{r})$ | $O(\ell)$ | SE-IND | No |
| LP (SD) [27] | $O(1)$ | $O(\log^2 N)$ | $O(\ell + r)$ | $O(\ell)$ | SRL-IND | No |
| Lee (CS) [21] | $O(L)$ | $O(L \log N)$ | $O(L + r \log \frac{N}{r})$ | $O(1)$ | AD-IND | No |
| Lee (SD) [21] | $O(L)$ | $O(L \log^2 N)$ | $O(L + r)$ | $O(1)$ | AD-IND | No |
| ETW [15] | $O(L)$ | $O(L \log N)$ | $O(Lr \log \frac{N}{r})$ | $O(1)$ | AD-IND | No |
| Ours-1 (CS) | $O(L)$ | $O(\ell)$ | $O(\ell r n)$ | $O(\ell^2 n)$ | AD-IND | Yes |
| Ours-2 (SD) | $O(L)$ | $O(\ell)$ | $O(\ell r)$ | $O(\ell^2 n^2)$ | SE-IND | Yes |
| Ours-3 (CS) | $O(L+n)$ | $O(L+n)$ | $O((L+n)rn)$ | $O(\ell)$ | AD-IND | Yes |

Let $N$ be the number of maximum users in each level, $r$ be the number of revoked users, $L$ be the maximum level of a hierarchical identity, $\ell$ be the level of a hierarchical identity, and $n$ be the depth of a binary tree in generic constructions. We count the number of group elements to measure the size. We use symbols SE-IND for selective IND-CPA, SRL-IND for selective revocation list IND-CPA, and AD-IND for adaptive IND-CPA.

## 6.2 Construction from Lattices

We instantiate an RHIBE-CS scheme from lattices. First, in order to instantiate an RHIBE-CS scheme that provides the selective security, we select the HIBE scheme of Agrawal, Boneh, and Boyen [1] that is selectively secure under the LWE assumption. In order to instantiate an RHIBE-CS scheme with shorter ciphertexts from lattices, we can select the HIBE scheme of Agrawal, Boneh, and Boyen [2] with shorter ciphertext. Unfortunately, we cannot instantiate an RHIBE-SD scheme from lattices since a lattice-based HSRE scheme does not exist yet.

## 6.3 An HSRE Scheme from Bilinear Maps

An HSRE scheme that is designed by combining the HIBE scheme of Boneh and Boyen [4] and the SRE scheme of Lee and Park [26] is described as follows:

**HSRE.Setup($1^\lambda, L$):** It first generates a bilinear group $\mathbb{G}$ of prime order $p$ of bit size $\Theta(\lambda)$. Let $g$ be a generator of $\mathbb{G}$. It chooses a random exponent $\alpha \in \mathbb{Z}_p$ and random elements $u, h, w, v \in \mathbb{G}$. It also chooses a random hash function $H$ from $\mathcal{H}$. It outputs a master key $MK = \alpha$ and public parameters as

$$PP = \Big( (p, \mathbb{G}, \mathbb{G}_T, e),\ g,\ u, h_1, \ldots, h_L,\ w, v,\ \Omega = e(g,g)^\alpha \Big).$$

**HSRE.GenKey($ID|_k, (GL, ML), MK, PP$):** Let $SK_{ID|_k} = (K'_0, K'_{1,1}, \ldots, K'_{1,k})$. It selects random exponents

$r_1, \ldots, r_{k+1}, r_{k+2} \in \mathbb{Z}_p$ and outputs a private key $SK_{ID|_k,(GL,ML)}$ as

$$K_0 = g^{\alpha} \prod_{i=1}^{k} (u^{I_i} h_i)^{r_i} (u^{GL} h_{k+1})^{r_{k+1}} w^{r_{k+2}},\ \big\{K_{1,i} = g^{-r_i}\big\}_{i=1}^{k+2},\ K_2 = (w^{ML} v)^{r_{k+2}}.$$

**HSRE.Delegate(**$ID|_k, SK_{ID|_{k-1}}, PP$**):** Let $SK_{ID|_{k-1}} = (K'_0, K'_{1,1}, \ldots, K'_{1,k-1})$. It selects random exponents $r'_1, \ldots, r'_{k-1}, r_k \in \mathbb{Z}_p$ and outputs a delegated private key $SK_{ID|_k}$ as

$$K_0 = K'_0 \prod_{i=1}^{k-1} (u^{I_i} h_i)^{r'_i} (u^{I_k} h_k)^{r_k},\ \big\{K_{1,i} = K'_{1,i} g^{-r'_i}\big\}_{i=1}^{k-1},\ K_{1,k} = g^{-r_k}.$$

**HSRE.MakeKey(**$(GL, ML), SK_{ID|_k}, PP$**):** Let $SK_{ID|_k} = (K'_0, K'_{1,1}, \ldots, K'_{1,k})$. It selects random exponents $r'_1, \ldots, r'_k, r_{k+1}, r_{k+2} \in \mathbb{Z}_p$ and outputs a private key $SK_{ID|_k,(GL,ML)}$ as

$$K_0 = K'_0 \prod_{i=1}^{k} (u^{I_i} h_i)^{r'_i} (u^{GL} h_{k+1})^{r_{k+1}} w^{r_{k+2}},\ \big\{K_{1,i} = K'_{1,i} g^{-r'_i}\big\}_{i=1}^{k},\ \big\{K_{1,i} = g^{-r_i}\big\}_{i=k+1}^{k+2},$$
$$K_2 = (w^{ML} v)^{r_{k+2}}.$$

**HSRE.Encrypt(**$ID|_{\ell-1}, (GL, ML), M, PP$**):** It chooses a random exponent $t \in \mathbb{Z}_p$ and outputs a ciphertext $CT_{ID|_{\ell-1},(GL,ML)}$ as

$$C = \Omega^t \cdot M,\ C_0 = g^t,\ \big\{C_{1,i} = (u^{I_i} h_i)^t\big\}_{i=1}^{\ell-1},\ C_{1,\ell} = (u^{GL} h)^t,\ C_2 = (w^{ML} v)^t.$$

**HSRE.Decrypt(**$CT_{ID|_{\ell-1},(GL,ML)}, SK_{ID'|_{\ell-1},(GL',ML')}, PP$**):** If $(ID|_{\ell-1} = ID'|_{\ell-1}) \wedge (GL = GL') \wedge (ML \neq ML')$, then it outputs a message as

$$M = C \cdot \Big( e(C_0, K_0) \cdot \prod_{i=1}^{\ell} e(C_{1,i}, K_{1,i}) \cdot \big( e(C_0, K_2) \cdot e(C_2, K_{1,\ell+1}) \big)^{-1/(ML'-ML)} \Big)^{-1}.$$

Otherwise, it outputs $\perp$.

**Theorem 6.1.** *The HSRE scheme is selectively IND-CPA secure if the DBDH assumption holds.*

*Proof.* Suppose there exists an adversary $\mathcal{A}$ that breaks the security of HSRE with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the DBDH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \mathbb{G}_T, e), g, g^a, g^b, g^c)$ and $Z$ where $Z = e(g,g)^{abc}$ or $Z = e(g,g)^d$. Then $\mathcal{B}$ interacts with $\mathcal{A}$ as follows:

**Init**: $\mathcal{A}$ initially submits a challenge hierarchical identity $ID^*|_{\ell-1} = (I^*_1, \ldots, I^*_{\ell-1})$ and challenge labels $(GL^*, ML^*)$.

**Setup**: $\mathcal{B}$ selects random exponents $u', h'_1, \ldots, h'_\ell, w', v' \in \mathbb{Z}_p$ and creates public parameters implicitly setting $\alpha = ab$ as

$$g,\ u = g^a g^{u'},\ h_1 = (g^a)^{-I^*_1} g^{h'_1},\ \ldots,\ h_{\ell-1} = (g^a)^{-I^*_{\ell-1}} g^{h'_{\ell-1}},\ h_\ell = (g^a)^{-GL^*} g^{h'_\ell},$$
$$w = g^a g^{w'}, v = (g^a)^{-ML^*} g^{v'},\ \Omega = e(g^a, g^b).$$

**Query 1**: $\mathcal{A}$ adaptively request delegate keys and private keys. If this is a delegate key query for $ID|_k$, then it handles this query as follows:

- **Case** $ID|_k \notin \text{Prefix}(ID^*|_{\ell-1})$: In this case, there is an index $j \in [k]$ such that $I_j - I_j^* \neq 0$. It selects random exponents $\{r_i\}_{1 \le i \neq j \le k}, r'_j \in \mathbb{Z}_p$ and creates a delegate key by implicitly setting $r_j = -b/(I_j - I_j^*) + r'_j$ as

$$K_0 = \prod_{1 \le i \neq j \le k} (u^{I_i} h_i)^{r_i} (g^b)^{-(u' I_j + h'_j)/(I_j - I_j^*)} (u^{I_j} h_j)^{r'_j},$$
$$\{K_{1,i} = g^{-r_i}\}_{1 \le i \neq j \le k},\ K_{1,j} = (g^b)^{1/(I_j - I_j^*)} g^{-r'_j}.$$

If this is a private key query for $ID|_k$ and $(GL, ML)$, then it handles this query as follows:

- **Case** $ID|_k \notin \text{Prefix}(ID^*|_{\ell-1})$: In this case, there is an index $j \in [k]$ such that $I_j - I_j^* \neq 0$. It selects random exponents $\{r_i\}_{1 \le i \neq j \le k}, r'_j, r_{k+1}, r_{k+2} \in \mathbb{Z}_p$ and creates a private key by implicitly setting $r_j = -b/(I_j - I_j^*) + r'_j$ as

$$K_0 = \prod_{1 \le i \neq j \le k} (u^{I_i} h_i)^{r_i} (g^b)^{-(u' I_j + h'_j)/(I_j - I_j^*)} (u^{I_j} h_j)^{r'_j} (u^{GL} h_{k+1})^{r_{k+1}} w^{r_{k+2}},$$
$$\{K_{1,i} = g^{-r_i}\}_{1 \le i \neq j \le k+2},\ K_{1,j} = (g^b)^{1/(I_j - I_j^*)} g^{-r'_j},\ K_2 = (w^{ML} v)^{r_{k+2}}.$$

- **Case** $ID|_k = ID^*|_{\ell-1} \wedge GL \neq GL^*$: It selects random exponents $r_1, \ldots, r_{\ell-1}, r'_\ell, r_{\ell+1} \in \mathbb{Z}_p$ and creates a private key by implicitly setting $r_\ell = -b/(GL - GL^*) + r'_\ell$ as

$$K_0 = \prod_{i=1}^{\ell-1} (u^{I_i} h_i)^{r_i} (g^b)^{-(u' GL + h')/(GL - GL^*)} (u^{GL} h_\ell)^{r'_\ell} w^{r_{\ell+1}},$$
$$\{K_{1,i} = g^{-r_i}\}_{1 \le i \neq \ell \le \ell+1},\ K_{1,\ell} = (g^b)^{1/(GL - GL^*)} g^{-r'_\ell},\ K_2 = (w^{ML} v)^{r_{\ell+1}}.$$

- **Case** $ID|_k = ID^*|_{\ell-1} \wedge GL = GL^* \wedge ML = ML^*$: It selects random exponents $r_1, \ldots, r_\ell, r'_{\ell+1} \in \mathbb{Z}_p$ and creates a private key by implicitly setting $r_{\ell+1} = -b + r'_{\ell+1}$ as

$$K_0 = \prod_{i=1}^{\ell-1} (u^{I_i} h_i)^{r_i} (u^{GL} h_\ell)^{r_\ell} (g^b)^{-w'} w^{r'_{\ell+1}},$$
$$\{K_{1,i} = g^{-r_i}\}_{i=1}^{\ell},\ K_{1,\ell+1} = g^b g^{-r'_{\ell+1}},\ K_2 = (g^b)^{-(w' ML + v')} (w^{ML} v)^{r'_{\ell+1}}.$$

**Challenge**: $\mathcal{A}$ submits two messages $M_0^*, M_1^*$. $\mathcal{B}$ flips a random coin $\mu \in \{0,1\}$ internally. Next, it implicitly sets $t = c$ and creates a challenge ciphertext as

$$C = Z \cdot M_\mu^*,\ C_0 = g^c,\ \{C_{1,i} = (g^c)^{u' I_i^* + h'_i}\}_{i=1}^{\ell-1},\ C_{1,\ell} = (g^c)^{u' GL^* + h'_\ell},\ C_2 = (g^c)^{w' ML^* + v'}.$$

**Query 2:** Same as Query 1.
**Guess**: Finally, $\mathcal{A}$ outputs a guess $\mu'$. If $\mu = \mu'$, $\mathcal{B}$ outputs 0. Otherwise, it outputs 1. □

## 7 Conclusion

In this paper, we have shown for the first time that it is possible to construct RHIBE schemes that provide the key revocation function by generically combining underlying cryptographic primitives and tree-based revocation methods. The first result is the RHIBE-CS scheme which combines HIBE schemes and the CS method. In this scheme, a ciphertext consists of $O(\ell n)$ HIBE ciphertexts and an update key consists of $O(rn)$ HIBE private keys where $\ell$ is the level of a hierarchical identity, $n$ is the depth of a binary tree, and $r$ is the number of revoked users. The second result is the RHIBE-SD scheme which combines HIBE and HSRE schemes, and the SD method. In this scheme, a ciphertext consists of one HIBE ciphertext and $O(\ell n^2)$ HSRE ciphertexts, and an update key consists of $O(r)$ HSRE private keys. The third result is the RHIBE-CS scheme with shorter ciphertexts that uses an HIBE scheme with constant-size ciphertext and a better identity encoding method. In this scheme, a ciphertext consists of $O(\ell)$ HIBE ciphertexts and an update key consists of $O(rn)$ HIBE private keys.

From this study, we left some interesting problems. The first one is to construct a revocable attribute-based encryption (RABE) scheme by generically combining an ABE scheme and the tree-based revocation method. In HIBE, it is possible to associate an identity with a leaf node in a binary tree, but in ABE, it is not easy to do that. The second one is to improve the RHIBE-SD scheme to have shorter ciphertexts. In order to reduce the ciphertext size of the RHIBE-SD scheme, a method of aggregating multiple HSRE ciphertexts into a single ciphertext is needed. The third one is to improve the update key size of the RHIBE-CS scheme with shorter ciphertexts.

## References

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

[2] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter-ciphertext hierarchical IBE. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2010.

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

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

[5] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 440–456. Springer, 2005.

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

[7] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In Eli Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 255–271. Springer, 2003.

[8] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 207–222. Springer, 2004.

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

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

[11] Shantian Cheng and Juanyang Zhang. Adaptive-ID secure revocable identity-based encryption from lattices via subset difference method. In Javier López and Yongdong Wu, editors, *Information Security Practice and Experience - ISPEC 2015*, volume 9065 of *Lecture Notes in Computer Science*, pages 283–297. Springer, 2015.

[12] Clifford C. Cocks. An identity based encryption scheme based on quadratic residues. In Bahram Honary, editor, *Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer, 2001.

[13] Yevgeniy Dodis and Nelly Fazio. Public key broadcast encryption for stateless receivers. In Joan Feigenbaum, editor, *Security and Privacy in Digital Rights Management - DRM 2002*, volume 2696 of *Lecture Notes in Computer Science*, pages 61–80. Springer, 2002.

[14] Nico Döttling and Sanjam Garg. Identity-based encryption from the Diffie-Hellman assumption. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology - CRYPTO 2017*, volume 10401 of *Lecture Notes in Computer Science*, pages 537–569. Springer, 2017.

[15] Keita Emura, Atsushi Takayasu, and Yohei Watanabe. Adaptively secure revocable hierarchical IBE from k-linear assumption. Cryptology ePrint Archive, Report 2020/886, 2020. `http://eprint.iacr.org/2020/886`.

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

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

[18] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology - ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer, 2002.

[19] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer, 2002.

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

[21] Kwangsu Lee. Revocable hierarchical identity-based encryption with adaptive security. Cryptology ePrint Archive, Report 2016/749, 2016. `http://eprint.iacr.org/2016/749`.

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

[23] Kwangsu Lee. Delegate and verify the update keys of revocable identity-based encryption. Cryptology ePrint Archive, Report 2020/1475, 2020. `http://eprint.iacr.org/2020/1475`.

[24] Kwangsu Lee, Woo Kwon Koo, Dong Hoon Lee, and Jong Hwan Park. Public-key revocation and tracing schemes with subset difference methods revisited. In Miroslaw Kutylowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, volume 8713 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2014.

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

[26] Kwangsu Lee and Jong Hwan Park. Identity-based revocation from subset difference methods under simple assumptions. *IEEE Access*, 7:60333–60347, 2019.

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

[28] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In Daniele Micciancio, editor, *Theory of Cryptography - TCC 2010*, volume 5978 of *Lecture Notes in Computer Science*, pages 455–479. Springer, 2010.

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

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

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

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

[33] Geumsook Ryu, Kwangsu Lee, Seunghwan Park, and Dong Hoon Lee. Unbounded hierarchical identity-based encryption with efficient revocation. In Howon Kim and Dooho Choi, editors, *Information Security Applications - WISA 2015*, volume 9503 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2015.

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

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

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

[37] Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology - CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1984.

[38] Atsushi Takayasu and Yohei Watanabe. Lattice-based revocable identity-based encryption with bounded decryption key exposure resistance. In Josef Pieprzyk and Suriadi Suriadi, editors, *Information Security and Privacy - ACISP 2017*, volume 10342 of *Lecture Notes in Computer Science*, pages 184–204. Springer, 2017.

[39] Brent Waters. Efficient identity-based encryption without random oracles. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.

[40] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636. Springer, 2009.