Revocable Hierarchical Identity-Based Encryption with Shorter Private Keys and Update Keys

Kwangsu Lee* Seunghwan Park†

Abstract

Revocable hierarchical identity-based encryption (RHIBE) is an extension of HIBE that supports the revocation of user's private keys to manage the dynamic credentials of users in a system. Many different RHIBE schemes were proposed previously, but they are not efficient in terms of the private key size and the update key size since the depth of a hierarchical identity is included as a multiplicative factor. In this paper, we propose efficient RHIBE schemes with shorter private keys and update keys and small public parameters by removing this multiplicative factor. To achieve our goals, we first present a new HIBE scheme with the different generation of private keys such that a private key can be simply derived from a short intermediate private key. Next, we show that two efficient RHIBE schemes can be built by combining our HIBE scheme, an IBE scheme, and a tree based broadcast encryption scheme in a modular way.

Keywords: Identity-based encryption, Hierarchical identity-based encryption, Key revocation, Modular design, Bilinear maps.

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

†Korea University, Seoul, Korea. Email: sgusa@korea.ac.kr.

1 Introduction

The secure digital communication between different entities through untrusted channel is one of the most important problems in computer network. Public-key encryption (PKE) was invented to solve this fundamental problem [10]. To handle dynamic credentials of each user in PKE, the public-key infrastructure (PKI) that can issue and revoke the certificates of users was successfully deployed. Although the PKI was effective in the traditional communication environment, it is not effective in the Internet of Things (IoT) environment since the certificate management of devices in IoT can be a big burden to the PKI.

Identity-based encryption (IBE), introduced by Shamir [35] and proposed by Boneh and Franklin [5], can solve this certificate management problem since the (already-known) identity of an entity can be served as a public key. To reduce the burden of a trusted center in IBE, hierarchical IBE (HIBE) that organizes each entity in a hierarchy was introduced by Horwitz and Lynn [17]. Similar to PKE, IBE and HIBE also should incorporate the revocation mechanism to handle the dynamic credential (or private key) of each identity.

1.1 Previous Work

A revocable IBE (RIBE) scheme that can revoke the private key of a user was presented by Boneh and Franklin [5], but their solution was not scalable since each user should retrieve a new private key from a trusted center per each time period. A scalable RIBE scheme was proposed by Boldyreva, Goyal, and Kumar [1]. In this RIBE scheme, a user is assigned to a leaf node in a full binary tree and a trusted center generates the private key of the user where the private key is associated with the path nodes. After that, the center periodically broadcasts an update key for non-revoked users per each time period where the update key is associated with the covering nodes that can cover the set of all non-revoked leaf nodes. If a user is not revoked in the update key, then he can decrypt a ciphertext by deriving a decryption key from his private key and the update key. This approach of RIBE was successfully used in the construction of other RIBE schemes [20, 25, 27, 31, 37].

The design approach of RIBE schemes that use a binary tree also can be applied to build revocable HIBE (RHIBE) schemes. The previous RHIBE schemes can be divided into two types depending on the generation of a decryption key from a private key and an update key. The first RHIBE scheme was proposed by Seo and Emura [30] by combining the HIBE scheme of Boneh and Boyen (BB-HIBE) [2] and the complete subtree (CS) scheme [26]. This RHIBE scheme is the type of history-preserving updates such that all update keys of ancestor identities are needed to derived a decryption key. In this RHIBE scheme, each identity keeps his own binary tree to handle the key revocation of child identities by following the design principle of the BGK-RIBE scheme. The main hurdle of the RHIBE design is to handle random values in each binary tree independently chosen by each identity. By carefully organizing a private key and an update key, they correctly derived a decryption key by removing the randomness. However, a private key consists of $O(\ell^2 \log N)$ group elements since a private key contains all private keys of ancestor's identities and an update key consists of $O(\ell r \log \frac{N}{r})$ group elements.

The second RHIBE scheme also proposed by Seo and Emura [33] by combining the HIBE scheme of Boneh et al. (BBG-HIBE) [4] and the CS scheme. This RHIBE scheme is the type of history-free updates such that a decryption key is derived from a private key and a parent's update key only. Compared to the previous RHIBE scheme via history-preserving updates, this RHIBE scheme via history-free updates can reduce the size of private keys since a private key does not include all private keys of ancestor's identities. To achieve the history-free update method, they observe that if an update key is derived from a decryption key instead of a parent's update key, then the random values chosen by ancestor's identities can be completely removed. In this RHIBE scheme, a private key consists of $O(\ell \log N)$ group elements and an update key

Table 1: Comparison of revocable hierarchical identity-based encryption schemes

Scheme	PP Size	SK Size	UK Size	CT Size	Model	Assumption
SE [30]	$O(\ell)$	$O(\ell^2 \log N)$	$O(\ell r \log \frac{N}{r})$	$O(\ell)$	SE-IND	DBDH
SE (CS) [33]	$O(\ell)$	$O(\ell \log N)$	$O(\ell r \log \frac{N}{r})$	$O(1)$	SE-IND	q-Type
SE (SD) [33]	$O(\ell)$	$O(\ell \log^2 N)$	$O(\ell r)$	$O(1)$	SRL-IND	q-Type
ESY [11]	$O(\ell)$	$O(\ell \log N)$	$O(\ell r \log \frac{N}{r})$	$O(\ell)$	SE-IND	DBDH
RLPL [29]	$O(1)$	$O(\ell \log N)$	$O(\ell r \log \frac{N}{r})$	$O(\ell)$	SE-IND	q-Type
Ours (CS)	$O(1)$	$O(\log N)$	$O(\ell + r \log \frac{N}{r})$	$O(\ell)$	SE-IND	q-Type
Ours (SD)	$O(1)$	$O(\log^2 N)$	$O(\ell + r)$	$O(\ell)$	SRL-IND	q-Type

We let N be the number of maximum users in each level, r be the number of revoked users, and ℓ be the depth of a hierarchical identity. For security model, we use symbols SE-IND for selective IND-CPA model, SRL-IND for selective revocation list IND-CPA model.

consists of $O(\ell r \log \frac{N}{r})$ group elements. They also showed that the subset difference (SD) scheme can be used to reduced the size of an update key by following the method of Lee et al. [20]. By following this approach, various RHIBE schemes with different properties were proposed [11, 29].

Although the size of a private key in RHIBE is somewhat reduced by using the history-free update method, the current RHIBE scheme is still inefficient in terms of the private key size and the update key size since the depth ℓ of a hierarchical identity is contained as a multiplicative factor. Thus, it is an important problem to reduce the size of private keys and update keys in RHIBE.

1.2 Our Results

In this paper, we show that efficient RHIBE schemes with shorter private keys and shorter update keys can be built by following the modular design approach. That is, we remove the multiplicative factor ℓ from the size of a private key and the size of an update key.

We first propose an HIBE scheme by modifying the HIBE scheme of Rouselakis and Waters (RW-HIBE) [28] that is derived from their key-policy attribute-based encryption (KP-ABE) scheme. The interesting feature of our HIBE scheme is that it allows the generation of a short intermediate private key. Furthermore, this intermediate private key can be easily converted into a normal private key by using public parameters. Note that the size of a private key in HIBE usually depends on the depth of a hierarchical identity [2, 4]. Although this intermediate private key does not play an important role in HIBE, we observe that this short intermediate private key enable to reduce the size of private key in RHIBE. That is, a private key in RHIBE can be reduced from $O(\ell \log N)$ group elements to $O(\log N)$ group elements if we use an intermediate private key instead of a normal private key. We also define additional algorithms of HIBE that express the special properties of our HIBE scheme. By using these additional algorithms, we can design our RHIBE schemes in a modular way.

Next, we propose an RHIBE-CS scheme by combining our HIBE scheme, the IBE scheme of Boneh and Boyen (BB-IBE) [2], and the CS scheme of Naor et al. [26]. Basically, we use the underlying HIBE, IBE, and CS schemes as modules when we design and prove the security of our RHIBE scheme. Although this modular design is different to the well-known black-box design, this modular approach can simplify the design and the security analysis of our RHIBE scheme. Because of the modular design, our RHIBE-CS

scheme can provide shorter private keys and shorter update keys by removing the multiplicative factor ℓ where ℓ is the depth of a hierarchical identity. That is, if we compare our RHIBE-CS scheme to the RHIBE scheme of Seo and Emura [33], the size of a private key is reduced from $O(\ell \log N)$ to $O(\log N)$ and the size of an update key is reduced from $O(\ell r \log \frac{N}{r})$ to $O(\ell + r \log \frac{N}{r})$. The detailed comparison of RHIBE schemes is given in Table 1.

Finally, we propose an RHIBE-SD scheme by combining our HIBE scheme, the BB-IBE scheme, and the subset difference (SD) scheme of Naor et al. [26]. As mentioned before, we can simplify the design and security analysis of our scheme by following the modular approach. By replacing the CS scheme with the SD scheme in our RHIBE-SD scheme, the size of an update key is reduced from $O(\ell + r \log \frac{N}{r})$ to $O(\ell + r)$, but the size of a private key is increased from $O(\log N)$ to $O(\log^2 N)$. To reduce the size of private keys, we can use the layered SD (LSD) scheme [16] instead of the SD scheme. In this case, the size of a private key is reduced to $O(\log^{1.5} N)$, but the size of an update key is slightly increased.

Note that we can use other HIBE scheme as the underlying HIBE scheme for our RHIBE scheme since our modular approach is a semi-generic method. For instance, if we use the BB-HIBE scheme [2], then an RHIBE scheme secure under the DBDH assumption can be obtained with larger public parameters. Furthermore, if we use the BBG-HIBE scheme [4], then we have an RHIBE scheme with short ciphertexts and longer private keys since the BBG-HIBE scheme cannot have intermediate private keys.

1.3 Our Techniques

To reduce the size of a private key in RHIBE, we first build an HIBE scheme modified from the RW-HIBE scheme [28] that supports a short intermediate private key ISK_{HIBE}. The size of a private key SK_{HIBE} in HIBE usually cannot be short since it depends on the depth ℓ of a hierarchical identity. However, we observe that a trusted center can generate a short intermediate private key ISK_{HIBE} if an independent random value r_i is used for each level-i and a special identity encoding is used. Let $ID = (I_1, I_2, I_3)$ be a hierarchical identity. We consider a special identity encoding $CID = (CI_1, CI_2, CI_3) = (I_1, I_1 \| I_2, I_1 \| I_2 \| I_3)$ where $\|$ is the concatenation of two strings. An interesting feature of this encoding is that the last identity values CI_3 and CI'_3 are different if CID and CID' are different. By using this feature, a simulator in the security proof can generate a short intermediate private key ISK_{HIBE} for $CID = (CI_1, CI_2, CI_3)$ by carefully controlling the random value r_3 only for the last identity CI_3 since CI_3 is different with CI^*_3 in the challenge ciphertext. Additionally, we can easily convert ISK_{HIBE} to SK_{HIBE} by selecting additional random values r_1, r_2 for other levels.

In HIBE, the intermediate private key ISK_{HIBE} with the constant size is not useful since a normal private key is needed for the key delegation. However, this short intermediate private key of HIBE can be very useful for the private key of RHIBE. In an RHIBE scheme, each user with a hierarchical identity ID keeps his own binary tree $\mathcal{BT}_{ID}$ to manage the revocation of child users. The private key SK of a child user with ID' consists of HIBE private keys $\{SK_{HIBE,v_i}\}$ that are associated with the path nodes $\{v_i\}$ in $\mathcal{BT}_{ID}$. We can replace the HIBE private key SK_{HIBE,v_i} to an HIBE intermediate private key ISK_{HIBE,v_i} to reduce the size of SK since the private key SK is not directly used for the key delegation in RHIBE. Note that the original HIBE private key SK_{HIBE} can be derived in the derivation process of a decryption key DK by using a private key SK that contains ISK_{HIBE,v_i} and an update key UK. Thus, the size of SK can be reduced since it consists of ISK_{HIBE,v_i} instead of SK_{HIBE,v_i}.

Furthermore, we reduce the size of an update key UK in RHIBE by separating a randomized decryption key RDK and an IBE private key for time in a modular way. In an RHIBE scheme, an update key UK for ID and time T consists of partial private keys $\{PSK_{v_i}\}$ that are associated with the cover nodes $\{v_i\}$ that cover all non-revoked leaf nodes. We observe that this partial private key PSK_{v_i} can be separated as a (randomized)

decryption key $RDK_{ID,T}$ and an IBE private key SK_{IBE,v_i} for time T. That is, the update key UK can consist of just one $RDK_{ID,T}$ and multiple $\{SK_{IBE,v_i}\}$ associated with the cover nodes. Therefore, we removed the multiplicative factor ℓ in a private key and an update key since ISK and SK_{IBE} consist of the constant number of group elements.

1.4 Related Work

As mentioned before, Horwitz and Lynn introduced the concept of HIBE and they presented a two-level HIBE scheme in bilinear maps [17]. An HIBE scheme that supports many-levels was proposed by Gentry and Silverberg [13] and the full model security was also given in the random oracle model. The HIBE scheme of Gentry and Silverberg also can be converted into a hierarchical identity-based signature (HIBS) scheme since the private key of HIBE can be a signature by the observation of Naor [5]. An HIBE scheme without random oracles was presented by Canetti et al. [9] and they showed that a forward-secure encryption can be built from any HIBE scheme. After that, Boneh and Boyen proposed an efficient HIBE scheme that is secure in the selective model under the standard assumption [2, 3]. An HIBE scheme with the constant size ciphertexts was given by Boneh, Boyen, and Goh [4] by using the power of a q-type assumption. There are many other HIBE schemes with different properties in bilinear maps [6, 7, 34].

Many HIBE schemes without random oracles were proven in the selective model where an adversary should submits the challenge hierarchical identity ID^* before he receives the public parameters [9]. An HIBE scheme that is secure in the full model without the security degradation was proposed by Gentry and Halevi [12] by using a q-type assumption. To achieve a fully secure HIBE scheme, Waters proposed the dual system framework where private keys and ciphertexts can have two types [38]. By using the dual system framework, Waters proposed a fully-secure HIBE scheme in the standard assumption. Lewko and Waters also presented another fully secure HIBE scheme with the constant size ciphertexts by using the dual system framework [23]. This dual system framework was wildly used in other HIBE schemes [21,22,24]. Recently, the dual system framework was also used to prove the full security of some RHIBE schemes [18, 32].

2 Preliminaries

In this section, we first briefly review bilinear groups and the complexity assumption in bilinear groups. Next, we define the syntax and the security model of revocable HIBE.

2.1 Notation

Let λ be a security parameter and $[n]$ be the set $\{1,\ldots,n\}$ for $n \in \mathbb{Z}$. Let $\mathcal{I}$ be the identity space. A hierarchical identity ID with a depth k is defined as an identity vector $ID = (I_1,\ldots,I_k) \in \mathcal{I}^k$. We let $ID|_j$ be a vector $(I_1,\ldots,I_j)$ of size j derived from ID. If $ID = (I_1,\ldots,I_k)$, then we have $ID = ID|_k$. We define $ID|_0 = \varepsilon$ for simplicity. We define a useful function for the hierarchical identity. $\mathbf{Prefix}(ID|_k)$ is a function that returns a set of prefix vectors $\{ID|_j\}$ where $1 \le j \le k$ where $ID|_k = (I_1,\ldots,I_k) \in \mathcal{I}^k$ for some k. For two hierarchical identities $ID|_i$ and $ID|_j$ with $i < j$, $ID|_i$ is an ancestor of $ID|_j$ and $ID|_j$ is a descendant of $ID|_i$ if $ID|_i \in \mathbf{Prefix}(ID|_j)$.

2.2 Bilinear Groups

Let $\mathbb{G}$ and $\mathbb{G}_T$ be multiplicative cyclic groups of prime order p and g be a generator of $\mathbb{G}$. The bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ has the following properties:

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

We say that $\mathbb{G}, \mathbb{G}_T$ are bilinear groups if the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable.

2.3 Complexity Assumptions

We introduce the q-RW2 assumption of Rouselakis and Waters [28] that was used to prove the security of their attribute-based encryption schemes.

Assumption 1 (q-RW2, [28])**.** Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ be the description of a bilinear groups of prime order p. Let g be a random generator of $\mathbb{G}$. The q-RW2 assumption is that if a challenge tuple

$$D = \Big((p, \mathbb{G}, \mathbb{G}_T, e), g, g^x, g^y, g^z, g^{(xz)^2}, \big\{ g^{b_i}, g^{xzb_i}, g^{xz/b_i}, g^{x^2 z b_i}, g^{y/b_i^2}, g^{y^2/b_i^2} \big\}_{\forall\, i \in [q]}, \\ \big\{ g^{xzb_i/b_j}, g^{yb_i/b_j^2}, g^{xyzb_i/b_j^2}, g^{(xz)^2 b_i/b_j} \big\}_{\forall\, i,j \in [q], i \neq j} \Big) \text{ and } Z$$

are given, no probabilistic polynomial time (PPT) algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = e(g, g)^{xyz}$ from $Z = Z_1 = e(g, g)^f$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{q\text{-}RW2}(\lambda) = \big| \Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0] \big|$ where the probability is taken over random choices of $x, y, z, \{b_i\}_{i \in [q]}$, $f \in \mathbb{Z}_p$.

2.4 Pseudo-Random Functions

A pseudo-random function (PRF) [14] is an efficiently computable function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ where $\mathcal{K}$ is the key space, $\mathcal{X}$ is the domain, and $\mathcal{Y}$ is the range. Let $F(k, \cdot)$ be an oracle for a uniformly chosen $k \in \mathcal{K}$ and $f(\cdot)$ be an oracle for a uniformly chosen function $f : \mathcal{X} \to \mathcal{Y}$. We say that a PRF is secure if for all efficient adversaries $\mathcal{A}$ the advantage $\mathbf{Adv}_{PRF,\mathcal{A}}(\lambda) = \big| \Pr[\mathcal{A}^{F(k,\cdot)} = 1] - \Pr[\mathcal{A}^{f(\cdot)} = 1] \big|$ is negligible.

2.5 Revocable HIBE

Revocable HIBE (RHIBE) is an extension of HIBE that allows a user to revoke the private key of a next level user if the private key is revealed or expired when the next level user's private key is delegated from his private key [30, 33]. In RHIBE, a user with $ID|_{k-1}$ can delegate his private key $SK_{ID|_{k-1}}$ to a next level user with $ID|_k$ by generating a private key $SK_{ID|_k}$. After that, the user periodically broadcasts an update key $UK_{T, R_{ID|_{k-1}}}$ for non-revoked users on time period T where $R_{ID|_{k-1}}$ is the set of revoked users. The next level user who has a private key $SK_{ID|_k}$ can derive a decryption key $DK_{ID|_k, T}$ from $SK_{ID|_k}$ and $UK_{T, R_{ID|_{k-1}}}$ if his private key is not revoked in the update key (i.e. $ID|_k \notin R_{ID|_{k-1}}$). Now, the user can decrypt a ciphertext for $ID'|_\ell$ and T' by using the derived $DK_{ID|_k, T}$ if $ID|_k = ID'|_\ell$ and $T = T'$.

Currently, there are two approaches that handle update keys in RHIBE. The first one is the history-preserving update approach where a user simply creates an update key without checking whether his private key is revoked or not [30]. In this case, a next level user should retrieves all update keys generated by his ancestors to derive a decryption key. The second one is the history-free update approach where a user can create an update key only when his private key is not revoked [33]. That is, an update key can be created if a user can derive a decryption key first. In this case, a next level user only needs to retrieve an update key generated by his parent to derive a decryption key. Note that the syntax and the security model of one

approach are slightly different from those of another approach. We follow the definition of RHIBE with history-free updates. The syntax of RHIBE with history-free updates is defined as follows:

Definition 2.1 (Revocable HIBE)**.** An RHIBE scheme with history-free 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$(1^\lambda, N_{max})$: This algorithm takes as input a security parameter 1^λ and the maximum number N_{max} of users in each level. It outputs a master key MK, an (empty) revocation list RL_ε, a state ST_ε, and public parameters PP.

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

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

DeriveKey$(SK_{ID|_k}, UK_{T,R_{ID|_{k-1}}}, PP)$: This algorithm takes as input a private key $SK_{ID|_k}$ for a hierarchical identity $ID|_k$, an update key $UK_{T,R_{ID|_{k-1}}}$ for time T and a revoked set $R_{ID|_{k-1}}$, 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$(ID|_k, T, RL_{ID|_{k-1}}, ST_{ID|_{k-1}})$: This algorithm takes as input a hierarchical identity $ID|_k$, revocation time T, a revocation list $RL_{ID|_{k-1}}$, and a state $ST_{ID|_{k-1}}$. 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**$(1^\lambda, N_{max})$, $SK_{ID|_k}$ generated by **GenKey**$(ID|_k, ST|_{k-1}, PP)$ for any $ID|_k$, $UK_{T,R_{ID|_{k-1}}}$ generated by **UpdateKey**$(T, RL_{ID|_{k-1}}, DK_{ID|_{k-1},T}$, $ST_{ID|_{k-1}}, PP)$ for any T and RL, $CT_{ID|_\ell,T}$ generated by **Encrypt**$(ID|_\ell, T, M, PP)$ for any $ID|_\ell$, T, and M, it is required that

- If $ID|_k \notin R_{ID|_{k-1}}$, then **DeriveKey**$(SK_{ID|_k}, UK_{T,R_{ID|_{k-1}}}, PP) = DK_{ID|_k,T}$.
- If $ID|_k \in R_{ID|_{k-1}}$, then **DeriveKey**$(SK_{ID|_k}, UK_{T,R_{ID|_{k-1}}}, PP) = \perp$.
- If $(ID|_\ell = ID'|_k) \wedge (T = T')$, then **Decrypt**$(CT_{ID|_\ell,T}, DK_{ID'|_k,T'}, PP) = M$.
- If $(ID|_\ell \neq ID'|_k) \vee (T \neq T')$, then **Decrypt**$(CT_{ID|_\ell,T}, DK_{ID'|_k,T'}, PP) = \perp$.

The selective security model of RHIBE with history-free updates that considers the decryption key exposure attack and the insider security was defined by Seo and Emura [33]. In this model, an adversary initially submits a challenge hierarchical identity $ID^*|_\ell$ and challenge time T^*. After receiving public parameters, the adversary can request private key, update key, decryption key, and revocation queries with some restrictions to prevent obvious attacks. In the challenge step, the adversary submits two challenge messages M_0^*, M_1^* and

receives a challenge ciphertext CT^* that is an encryption of one challenge message. The adversary wins the game if he correctly guesses the encrypted message.

By carefully examining the security model of Seo and Emura [33], we found that their definition of the security model is not complete since they missed one important restriction in update key queries. That is, an adversary cannot query an update key for $ID|_{k-1}$ on time T if one ancestor of $ID|_{k-1}$ is already revoked on time T since the decryption key $DK_{ID|_{k-1},T}$ cannot be created by the syntax of RHIBE with history-free updates. Recall that an adversary easily distinguishes whether it is a simulation or not by simply querying the update key if this restriction is not enforced. Note that this restriction in update key queries is not needed in the security model of RHIBE with history-preserving updates [30]. The corrected security model of RHIBE with history-free updates is defined as follows:

Definition 2.2 (Selective IND-CPA Security (SE-IND-CPA))**.** The selective 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. **Init**: $\mathcal{A}$ initially submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$ and challenge time T^*.

2. **Setup**: $\mathcal{C}$ obtains a master key MK, a revocation list RL_ε, a state ST_ε, and public parameters PP by running $\mathbf{Setup}(1^\lambda, N_{max})$. It keeps $MK, RL_\varepsilon, ST_\varepsilon$ to itself and gives PP to $\mathcal{A}$.

3. **Phase 1**: $\mathcal{A}$ adaptively requests a polynomial number of queries. These queries are processed as follows:

   - **Private key**. If it is a private key query for a hierarchical identity $ID|_k$, then $\mathcal{C}$ gives a private key $SK_{ID|_k}$ and a state $ST_{ID|_k}$ by running $\mathbf{GenKey}(ID|_k, ST_{ID|_{k-1}}, PP)$ with the restriction: If $ID|_k \in \mathbf{Prefix}(ID^*|_\ell)$ where $k \leq \ell$, then $ID|_k$ or one of its ancestors must be revoked at some time T where $T \leq T^*$.
   - **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 $UK_{T,R_{ID|_{k-1}}}$ by running $\mathbf{UpdateKey}(T, RL_{ID|_{k-1}}, DK_{ID|_{k-1},T}, ST_{ID|_{k-1}}, PP)$ with the restriction: If $ID|_{k-1}$ or one of its ancestors is revoked on time T, then this update key query cannot be requested since $DK_{ID|_{k-1},T}$ cannot be derived.
   - **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 $\mathbf{DeriveKey}(SK_{ID|_k}, UK_{T,R_{ID|_{k-1}}}, PP)$ with the restriction: A decryption key query for the challenge identity $ID^*|_k$ or its ancestors on the challenge time T^* cannot be requested.
   - **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 $\mathbf{Revoke}(ID|_k, T, RL_{ID|_{k-1}}, ST_{ID|_{k-1}})$ 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.

4. **Challenge**: $\mathcal{A}$ submits two challenge messages M_0^*, M_1^* with the same length. $\mathcal{C}$ flips a random coin $\mu \in \{0,1\}$ and gives the challenge ciphertext $CT^*_{ID^*|_\ell, T^*}$ to $\mathcal{A}$ by running $\mathbf{Encrypt}(ID^*|_\ell, T^*, M_\mu^*, PP)$.

5. **Phase 2**: $\mathcal{A}$ may continue to request a polynomial number of queries subject to the same restrictions as before.

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

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

Definition 2.3 (Selective Revocation List IND-CPA Security (SRL-IND-CPA))**.** The selective revocation list IND-CPA security of RHIBE is weaker than the previous SE-IND-CPA security of RHIBE. In this model, an adversary initially submits a challenge hierarchical identity $ID^*|_\ell$, challenge time T^*, and a revocation list RL^* on the time T^*. An RHIBE scheme is SRL-IND-CPA secure if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ is negligible in the security parameter λ.

3 Hierarchical Identity-Based Encryption

In this section, we first propose an HIBE scheme that supports the generation of short intermediate private keys. We also present the IBE scheme of Boneh and Boyen [2] with additional algorithms. Note that we present the KEM version of HIBE and IBE for the modular approach.

3.1 Definitions

HIBE is an extension of IBE that can reduce the workload of a trusted center by delegating the generation of private keys to other entities [13, 17]. In HIBE, all users are organized in a hierarchy and the hierarchical identity of a user is represented as an identity vector. A user with a hierarchical identity $ID|_k = (I_1, \ldots, I_k)$ can receive his private key $SK_{ID|_k}$ from a trusted center and later he can delegate $SK_{ID|_k}$ to another user with a hierarchical identity $ID|_{k+1} = (I_1, \ldots, I_k, I_{k+1})$ if $ID|_k$ is a prefix of $ID|_{k+1}$. A sender can create a ciphertext header $CH_{ID|_\ell}$ and a session key EK for a user with $ID|_\ell$ by using public parameters. A receiver who has a private key $SK_{ID|_k}$ can derive the session key from the ciphertext header if $ID|_k$ is a prefix of $ID|_\ell$. The syntax of HIBE is defined as follows:

Definition 3.1 (HIBE)**.** An HIBE scheme consists of five algorithms **Setup**, **GenKey**, **Delegate**, **Encaps**, and **Decaps**, which are defined as follows:

Setup(1^λ). The setup algorithm takes as input a security parameter 1^λ. 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 $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$, the master key MK, and the public parameters PP. It outputs a private key $SK_{ID|_k}$ for $ID|_k$.

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 $SK_{ID|_k}$ for $ID|_k$.

Encaps$(ID|_k, PP)$. The key encapsulation algorithm takes as input a hierarchical identity $ID|_k$ and the public parameters PP. It outputs a ciphertext header $CH_{ID|_k}$ for $ID|_k$ and a session key EK.

Decaps$(CH_{ID|_k}, SK_{ID'|_\ell}, PP)$. The key decapsulation algorithm takes as input a ciphertext header $CH_{ID|_k}$ for $ID|_k$, a private key $SK_{ID'_\ell}$ for ID'_ℓ, and the public parameters PP. It outputs a session key EK or $\perp$.

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

- If $ID|_k \in \mathbf{Prefix}(ID'|_\ell)$, then $\mathbf{Decaps}(\mathbf{Encaps}(ID'|_\ell, PP), SK_{ID|_k}, PP) = EK$.
- If $ID|_k \notin \mathbf{Prefix}(ID'|_\ell)$, then $\mathbf{Decaps}(\mathbf{Encaps}(ID'|_\ell, PP), SK_{ID}, PP) = \perp$.

The security model of HIBE was defined by Gentry and Silverberg [13] and the selective security model was introduced by Canetti et al. [8]. In this paper, we define the KEM version of the selective security model. In this model, an adversary initially submits a challenge hierarchical identity $ID^*|_\ell$ before he receives the public parameters. After that, the adversary can request private key queries for some hierarchical identities that are not prefixes of $ID^*|_\ell$. In the challenge step, the adversary receives a ciphertext header CH^* and a challenge session key EK^* that is real or random. Finally, the adversary guesses whether the challenge session key is real or random and outputs his guess. If the adversary correctly guesses the session key, then he wins the game. The detailed definition of the security model is defined as follows:

Definition 3.2 (Selective IND-CPA Security (SE-IND-CPA))**.** The selective IND-CPA security of HIBE is defined in terms of the following experiment between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:

1. **Init**: $\mathcal{A}$ initially submits a challenge identity $ID^*|_\ell$.
2. **Setup**: $\mathcal{C}$ generates a master key MK and public parameters PP by running $\mathbf{Setup}(1^\lambda)$. It keeps MK to itself and gives PP to $\mathcal{A}$.
3. **Phase 1**: $\mathcal{A}$ may adaptively request a polynomial number of private key queries. If this is a private key query for a hierarchical identity $ID|_k$ with the restriction $ID|_k \notin \mathbf{Prefix}(ID^*|_\ell)$, then it creates a private key $SK_{ID|_k}$ by calling $\mathbf{GenKey}(ID|_k, MK, PP)$.
4. **Challenge**: In the challenge step, $\mathcal{C}$ creates a ciphertext header CH^* and a real session key EK^* by running $\mathbf{Encaps}(ID^*|_\ell, PP)$. Next, it flips a random coin $\mu \in \{0,1\}$ and gives CH^*, EK^* to $\mathcal{A}$ if $\mu = 0$. Otherwise, it gives CH^* and a random session key to $\mathcal{A}$.
5. **Phase 2**: $\mathcal{A}$ continues to request a polynomial number of private key queries subject to the same restriction as before.
6. **Guess**: $\mathcal{A}$ outputs a guess $\mu' \in \{0,1\}$ of μ, and wins the game if $\mu = \mu'$.

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

3.2 Concatenated Identity Encoding Function

We define a concatenated identity encoding function that converts a hierarchical identity $ID|_k$ to a concatenated hierarchical identity $CID|_k$. This encoding was informally introduced by Waters [38] and also used by Lewko and Waters [24]. Let H be a collision-resistant hash function that takes as input a bit string $\{0,1\}^*$ and outputs an element in $\mathbb{Z}_p$. $\mathbf{EncodeCID}(ID|_k)$ is a function that takes as input a hierarchical identity $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ and returns a concatenated hierarchical identity $CID|_k = (CI_1, \ldots, CI_k) \in \mathbb{Z}_p^k$ where $CI_j = H(ID|_j) = H(I_1 \| \cdots \| I_j)$ where $\|$ denotes the concatenation of two strings. The following lemma shows an interesting property of this function.

Lemma 3.1. *Let $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ and $ID'|_\ell = (I'_1, \ldots, I_\ell) \in \mathcal{I}^\ell$ for some $k, \ell \in \mathbb{Z}$. We let $CID|_k = (CI_1, \ldots, CI_k)$ and $CID'|_\ell = (CI'_1, \ldots, CI'_\ell)$ that are returned by **EncodeCID**$(ID|_k)$ and **EncodeCID**$(ID'|_\ell)$ respectively. The function **EncodeCID**$(\cdot)$ satisfies the following properties:*

- *Property 1. If $ID|_k \in$ **Prefix**$(ID'|_\ell)$, then $CI_j = CI'_j$ for all $j \in [k]$.*
- *Property 2. If $ID|_k \notin$ **Prefix**$(ID'|_\ell)$, then $CI_k \neq CI'_i$ for all $i \in [\ell]$ except with negligible probability.*

Proof. The property 1 is straightforward from the fact that $ID|_j = ID'|_j$ since $ID|_k \in$ **Prefix**$(ID'|_\ell)$. To show the property 2, we consider two cases: $k \leq \ell$ and $\ell < k$. In case of $k \leq \ell$, there exists $j^* \in [k]$ such that $I_{j^*} \neq I'_{j^*}$ since $ID|_k \notin$ **Prefix**$(ID'|_\ell)$. If we suppose that $CI_k = CI'_i$ for some $i \in [\ell]$, then we have a collision $H(ID|_k) = H(ID'|_i)$ since $ID|_k = (I_1, \ldots, I_{j^*}, \ldots, I_k)$ is not equal to $ID'|_i = (I'_1, \ldots, I'_i)$. However, it is a contradiction to the collision-resistance of a hash function. In case of $\ell < k$, we also have $ID|_k \neq ID'|_i$ since $k > \ell$. Thus $CI_k \neq CI'_i$ is satisfied by the collision resistance of a hash function. □

3.3 HIBE Construction

A generic construction of an HIBE scheme from a KP-ABE scheme with the delegation of private keys was shown by Goyal et el. [15]. That is, a KP-ABE scheme can be converted to an HIBE scheme if an access structure in a private key is represented as one AND gate and an attribute set in a ciphertext is specially encoded from a hierarchical identity. For example, we can encode a hierarchical identity $ID =$ ("*com*","*dev*","*john*") as an attribute set $S =$ ("$1:$ *com*","$2:$ *dev*","$3:$ *john*"). Rouselakis and Waters also pointed out that their large-universe KP-ABE scheme with short public parameters can be easily converted into an HIBE scheme (RW-HIBE) [28].

We slightly modify the RW-HIBE scheme to reduce the size of private keys. The first modification is to use a simple secret sharing method instead of an LSSS method in a private keys since one AND gate is enough for HIBE. If we use a simple secret sharing method, then we can compress some group elements in private keys. The second modification is to use the concatenated identity encoding function in the previous section. If we can use the concatenated identity encoding function, then it is possible to generate a private key in a different way. That is, a trusted center first generates an intermediate private key with the constant number of group elements, and then anyone who has the intermediate private key can derive an original private key by using public parameters. This new method enables to reduce the size of private keys in RHIBE. The third modification is to take a random exponent for a session key from outside and to define additional algorithms. These changes are helpful to use an HIBE scheme as a module.

Our HIBE scheme with the generation of short intermediate private keys is described as follows:

HIBE.Setup(GDS)**:** Let $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$ be the description of a bilinear group with a generator $g \in \mathbb{G}$. It selects random elements $u, h, w \in \mathbb{G}$ and a random exponent $\gamma \in \mathbb{Z}_p$. It outputs a master key $MK = \gamma$ and public parameters $PP = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, u, h, w, \Lambda = e(g,g)^\gamma\big)$.

HIBE.GenKey$(ID|_k, MK, PP)$**:** Let $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ and $MK = \gamma$. It obtains $CID|_k = (CI_1, \ldots, CI_k)$ by calling **EncodeCID**$(ID|_k)$. It chooses random exponents $r_1, \ldots, r_k \in \mathbb{Z}_p$ and outputs a private key $SK_{ID|_k} = \big(K_0 = g^\gamma \prod_{i=1}^k w^{r_i}, \big\{K_{i,1} = (u^{CI_i} h)^{-r_i}, K_{i,2} = g^{r_i}\big\}_{i=1}^k\big)$. For notational simplicity, we define $SK_{ID|_0} = (K_0 = g^\gamma)$.

HIBE.RandKey$(SK_{ID|_k}, PP)$**:** Let $SK_{ID|_k} = (K'_0, \{K'_{i,1}, K'_{i,2}\}_{i=1}^k)$. It obtains $CID|_k = (CI_1, \ldots, CI_k)$ from $ID|_k$. It chooses random exponents $r_1, \ldots, r_k \in \mathbb{Z}_p$ and outputs a randomized private key $SK_{ID|_k} = \big(K_0 = K'_0 \cdot \prod_{i=1}^k w^{r_i}, \big\{K_{i,1} = K'_{i,1} \cdot (u^{CI_i} h)^{-r_i}, K_{i,2} = K'_{i,2} \cdot g^{r_i}\big\}_{i=1}^k\big)$.

HIBE.Delegate$(ID|_k, SK_{ID|_{k-1}}, PP)$**:** Let $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ and $SK_{ID|_{k-1}} = \left(K'_0, \{K'_{i,1}, K'_{i,2}\}_{i=1}^{k-1}\right)$ where $ID|_{k-1}$ is a prefix of $ID|_k$. It obtains $CID|_k = (CI_1, \ldots, CI_k)$ from $ID|_k$. It chooses a random exponent $r_k \in \mathbb{Z}_p$ and creates a temporal private key $TSK = \left(K_0 = K'_0 \cdot w^{r_k}, \left\{K_{i,1} = K'_{i,1}, K_{i,2} = K'_{i,2}\right\}_{i=1}^{k-1}, \left\{K_{k,1} = (u^{CI_k} h)^{-r_k}, K_{k,2} = g^{r_k}\right\}\right)$. Next, it outputs a delegated private key $SK_{ID|_k}$ by running **HIBE.RandKey** (TSK, PP).

HIBE.Encaps$(ID|_\ell, t, PP)$**:** Let $ID|_\ell = (I_1, \ldots, I_\ell) \in \mathcal{I}^\ell$. It obtains $CID|_k = (CI_1, \ldots, CI_k)$ from $ID|_k$. It chooses random exponents $s_1, \ldots, s_k \in \mathbb{Z}_p$ and outputs a ciphertext header $CH_{ID|_\ell} = \left(C_0 = g^t, \left\{C_{i,1} = g^{s_i}, C_{i,2} = (u^{CI_i} h)^{s_i} w^{-t}\right\}_{i=1}^{\ell}\right)$ and a session key $EK = \Lambda^t$.

HIBE.Decaps$(CT_{ID|_\ell}, SK_{ID'|_k}, PP)$**:** Let $CH_{ID|_\ell} = (C_0, \{C_1, C_2\}_{i=1}^{\ell})$ and $SK_{ID'|_k} = (K_0, \{K_{i,1}, K_{i,2}\}_{i=1}^{k})$. If $ID'|_k \in$ **Prefix**$(ID|_\ell)$, then it outputs a session key EK by calculating $e(C_0, K_0) \cdot \prod_{i=1}^{k} \left(e(C_{i,1}, K_{i,1}) \cdot e(C_{i,2}, K_{i,2})\right)$. Otherwise, it outputs $\perp$.

Remark 1. The above HIBE scheme is a slightly different with the definition of an HIBE scheme. That is, we added **RandKey** algorithm and modified the **Encrypt** algorithm to takes as input an exponent for a session key. The reason of this modification is for the modular construction of an RHIBE scheme.

We define additional algorithms **ChangeKey**, **MergeKey**, **GenIKey**, **RandIKey**, **ChangeIKey**, and **DeriveKey**. The **ChangeKey** algorithm changes the master key part of a private key by performing a constant addition or a constant multiplication. The **MergeKey** algorithm derives a new private key where the new master key part is the additive homomorphism of two master key parts of inputs. The **GenIKey** algorithm creates an intermediate private key that consists of just three group elements. The **DeriveKey** algorithm derives a private key from the intermediate private key. These algorithms are very helpful to build our RHIBE schemes in a modular way.

HIBE.ChangeKey$(SK_{ID|_k}, \{(op_i, \delta_i)\}_{i=1}^{n}, PP)$**:** Let $SK_{ID|_k} = (K'_0, \{K'_{i,1}, K'_{i,2}\}_{i=1}^{k})$ and $op_i \in \{+, \times\}$. It sets $TSK^{(0)} = SK_{ID|_k}$. For each (op_i, δ_i), it performs: If $op_i = +$, then it sets $TSK^{(i)} = (K_0^{(i)} = K_0^{(i-1)} \cdot g^{\delta_i}, \{K_{j,1}^{(i)} = K_{j,1}^{(i-1)}, K_{j,2}^{(i)} = K_{j,2}^{(i-1)}\}_{j=1}^{k})$. If $op_i = \times$, then it sets $TSK^{(i)} = (K_0^{(i)} = (K_0^{(i-1)})^{\delta_i}, \{K_{j,1}^{(i)} = (K_{j,1}^{(i-1)})^{\delta_i}, K_{j,2}^{(i)} = (K_{j,2}^{(i-1)})^{\delta_i}\}_{j=1}^{k})$. It outputs a new private key $SK_{ID|_k}$ by running **HIBE.RandKey** $(TSK^{(n)}, PP)$.

HIBE.MergeKey$(SK_{ID|_k}^{(1)}, SK_{ID|_k}^{(2)}, \eta, PP)$**:** Let $SK_{ID|_k}^{(1)} = (K'_0, \{K'_{i,1}, K'_{i,2}\}_{i=1}^{k})$ and $SK_{ID|_k}^{(2)} = (K''_0, \{K''_{i,1}, K''_{i,2}\}_{i=1}^{k})$ be two private keys for the same identity $ID|_k$. It computes a temporal merged private key $TSK = \left(K_0 = K'_0 \cdot K''_0, \left\{K_{i,1} = K'_{i,1} \cdot K''_{i,1}, K_{i,2} = K'_{i,2} \cdot K''_{i,1}\right\}_{i=1}^{k}\right)$. Next, it outputs a merged private key $SK_{ID|_k}$ by running **HIBE.ChangeKey**$(TSK, (+, \eta), PP)$. Note that the master key part is $\gamma_1 + \gamma_2 + \eta$ if the master key parts of $SK_{ID|_k}^{(1)}$ and $SK_{ID|_k}^{(2)}$ are γ_1 and γ_2 respectively.

HIBE.GenIKey$(ID|_k, MK, PP)$**:** Let $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ and $MK = \gamma$. It obtains $CID|_k = (CI_1, \ldots, CI_k)$ from $ID|_k$. It chooses a random exponent $r_k \in \mathbb{Z}_p$ and outputs an intermediate private key $ISK_{ID|_k} = \left(K_0 = g^{\gamma} w^{r_k}, \left\{K_{k,1} = (u^{CI_k} h)^{-r_k}, K_{k,2} = g^{r_k}\right\}\right)$.

HIBE.RandIKey, HIBE.ChangeIKey These algorithms are very similar to the algorithms **HIBE.RandKey** and **HIBE.ChangeKey** except that they take $ISK_{ID|_k}$ and return $ISK_{ID|_k}$. We omit the description.

HIBE.DeriveKey$(ISK_{ID|_k}, PP)$**:** Let $ISK_{ID|_k} = (K'_0, \{K'_{k,1}, K'_{k,2}\})$ for $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$. It obtains $CID|_k = (CI_1, \ldots, CI_k)$ from $ID|_k$. It chooses random exponents $r_1, \ldots, r_k \in \mathbb{Z}_p$ and outputs a private key

$SK_{ID|_k} = \big(K_0 = K'_0 \cdot \prod_{i=1}^{k} w^{r_i}, \{K_{i,1} = (u^{CI_i}h)^{-r_i}, K_{i,2} = g^{r_i}\}_{i=1}^{k-1}, \{K_{k,1} = K'_{k,1} \cdot (u^{CI_k}h)^{-r_k}, K_{k,2} = K'_{k,2} \cdot g^{r_k}\}\big)$. Note that the private key is correctly distributed because of newly chosen random exponents.

The correctness of the above HIBE scheme is relatively straightforward from that of the RW-HIBE scheme. We omit the description of the correctness.

3.4 IBE Construction

The IBE scheme of Boneh and Boyen [2] with additional algorithms is given as follows:

IBE.Setup(GDS): Let $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$ be the group description string of a bilinear group with a generator $g \in \mathbb{G}$. It selects random elements $u_I, h_I \in \mathbb{G}$ and a random exponent $\beta \in \mathbb{Z}_p$. It outputs a master key $MK = \beta$ and public parameters $PP = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, u_I, h_I, \Lambda = e(g,g)^{\beta}\big)$.

IBE.GenKey(ID, MK, PP): It chooses a random exponent $r \in \mathbb{Z}_p$ and outputs a private key $SK_{ID} = \big(K_0 = g^{\beta}(u_I^{ID}h_I)^r, K_1 = g^{-r}\big)$.

IBE.RandKey(SK_{ID}, PP): Let $SK_{ID} = (K'_0, K'_1)$. It chooses a random exponent $r \in \mathbb{Z}_p$ and outputs a randomized private key $SK_{ID} = \big(K_0 = K'_0 \cdot (u_I^{ID}h_I)^r, K_1 = K'_1 \cdot g^{-r}\big)$.

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

IBE.Decaps($CT_{ID}, SK_{ID'}, PP$): Let $CH_{ID} = (C_0, C_1)$ and $SK_{ID'} = (K_0, K_1)$. If $ID = ID'$, then it outputs a session key by computing $EK = e(C_0, K_0) \cdot e(C_1, K_1)$. Otherwise, it outputs $\perp$.

We additionally define two algorithms **ChangeKey** and **MergeKey**. Although these algorithms are not needed for IBE, those are very helpful to build our RHIBE scheme.

IBE.ChangeKey($SK_{ID}, \{(op_i, \delta_i)\}_{i=1}^{n}, PP$): Let $SK_{ID} = (K'_0, K'_1)$, $op_i \in \{+, \times\}$, and δ_i be a random exponent in $\mathbb{Z}_p$. It sets $TSK^{(0)} = SK_{ID}$. For each (op_i, δ_i), it performs: If $op_i = +$, then it sets $TSK^{(i)} = (K_0^{(i)} = K_0^{(i-1)} \cdot g^{\delta_i}, K_1^{(i)} = K_1^{(i-1)})$. If $op_i = \times$, then it sets $TSK^{(i)} = (K_0^{(i)} = (K_0^{(i-1)})^{\delta_i}, K_1^{(i)} = (K_1^{(i-1)})^{\delta_i})$. It outputs a new private key SK_{ID} by running **IBE.RandKey**$(TSK^{(n)}, PP)$.

IBE.MergeKey($SK_{ID}^{(1)}, SK_{ID}^{(2)}, \eta, PP$): Let $SK_{ID}^{(1)} = (K'_0, K'_1)$ and $SK_{ID}^{(2)} = (K''_0, K''_1)$ be two private keys for the same identity ID. It computes a temporal merged private key $TSK_{ID} = \big(K_0 = K'_0 \cdot K''_0, K_1 = K'_1 \cdot K''_1,\big)$. Next, it outputs a merged private key SK_{ID} by running **IBE.ChangeKey**$(TSK_{ID}, (+, \eta), PP)$.

Remark 2. The above IBE scheme is slightly different with the original BB-IBE scheme. That is, we added the **RandKey** algorithm that randomizes a private key, the **ChangeKey** algorithm that changes the master key part of a private key, and the **MergeKey** algorithm that merges two private keys for the same identity. Additionally, we modified the **Encrypt** algorithm to takes as input an exponent for a session key. The reason of this modification is for the modular construction of an RHIBE scheme.

3.5 Security Analysis

Theorem 3.2. *The above HIBE scheme is SE-IND-CPA secure if the q-RW2 assumption holds.*

Proof. Suppose that there exists an adversary $\mathcal{A}$ that attacks the above HIBE scheme with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the q-RW2 assumption using $\mathcal{A}$ is given: a challenge tuple $D = \big((p,\mathbb{G},\mathbb{G}_T,e),g,g^x,g^y,g^z,g^{(xz)^2},\{g^{b_i},g^{xzb_i},g^{xz/b_i},g^{x^2zb_i},g^{y/b_i^2},g^{y^2/b_i^2}\},\{g^{xzb_i/b_j},g^{yb_i/b_j^2},g^{xyzb_i/b_j^2},g^{(xz)^2b_i/b_j}\}\big)$ and Z where $Z = Z_0 = e(g,g)^{xyz}$ or $Z = Z_1 \in_R \mathbb{G}_T$. $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*,\ldots,I_\ell^*)$ where $\ell \le q$. It obtains $CID^*|_\ell = (CI_1^*,\ldots,CI_\ell^*)$ by calling **EncodeCID**$(ID^*|_\ell)$.

Setup: $\mathcal{B}$ chooses random exponents $u',h' \in \mathbb{Z}_p$ and creates public parameters PP by implicitly setting $\alpha = xy$ as

$$g,\ u = g^{u'}\prod_{i=1}^{\ell} g^{y/b_i^2},\ h = g^{h'}\prod_{i=1}^{\ell}\big(g^{xz/b_i}(g^{y/b_i^2})^{-CI_i^*}\big),\ w = g^x,\ \Lambda = e(g^x,g^y).$$

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

1. It first obtains $CID|_k = (CI_1,\ldots,CI_k)$ by calling **EncodeCID**$(ID|_k)$. If $ID|_k \notin$ **Prefix**$(ID^*|_\ell)$ by the restriction of the security model, then we have $CI_k \neq CI_i^*$ for all $i \in [\ell]$ from Lemma 3.1.

2. It chooses a random exponent $r_k' \in \mathbb{Z}_p$ and implicitly sets $r_k = -y + \Sigma_{i=1}^{\ell}\frac{xzb_i}{CI_k - CI_i^*} + r_k'$. It creates an intermediate private key $ISK_{ID|_k}$

$$\begin{aligned}
K_0 &= g^{\alpha}w^{r_k} = g^{xy}\cdot\big(g^x\big)^{-y+\Sigma_{i=1}^{\ell}\frac{xzb_i}{CI_k-CI_i^*}+r_k'} = g^{xy}\cdot g^{-xy}\cdot\prod_{i=1}^{\ell}\big(g^{x^2zb_i}\big)^{\frac{1}{CI_k-CI_i^*}}\cdot w^{r_k'}\\
&= \prod_{i=1}^{\ell}\big(g^{x^2zb_i}\big)^{\frac{1}{CI_k-CI_i^*}}\cdot w^{r_k'},\\
K_{k,1} &= \big(u^{CI_k}h\big)^{-r_k} = \big(u^{CI_k}h\big)^{y-\Sigma_{i=1}^{\ell}\frac{xzb_i}{CI_k-CI_i^*}-r_k'}\\
&= \Big(g^{u'CI_k+h'}\cdot\prod_{i=1}^{\ell}(g^{y/b_i^2})^{CI_k-CI_i^*}\cdot\prod_{i=1}^{\ell}g^{xz/b_i}\Big)^{y-\Sigma_{i=1}^{\ell}\frac{xzb_i}{CI_k-CI_i^*}}\cdot\big(u^{CI_k}h\big)^{-r_k'}\\
&= \big(g^y\big)^{u'CI_k+h'}\cdot\prod_{i=1}^{\ell}\big(g^{xzb_i}\big)^{-\frac{u'CI_k+h'}{CI_k-CI_i^*}}\cdot\prod_{i=1}^{\ell}\big(g^{y^2/b_i^2}\big)^{CI_k-CI_i^*}\cdot\prod_{i=1}^{\ell}\prod_{j=1,j\neq i}^{\ell}\big(g^{xyzb_j/b_i^2}\big)^{-\frac{CI_k-CI_i^*}{CI_k-CI_j^*}}\cdot\\
&\quad\prod_{i=1}^{\ell}\prod_{j=1}^{\ell}\big(g^{(xz)^2b_j/b_i}\big)^{-\frac{1}{CI_k-CI_j^*}}\cdot\big(u^{CI_k}h\big)^{-r_k'},\\
K_{k,2} &= g^{r_k} = g^{-y+\Sigma_{i=1}^{\ell}\frac{xzb_i}{CI_k-CI_i^*}+r_k'} = \big(g^y\big)^{-1}\cdot\prod_{i=1}^{\ell}\big(g^{xzb_i}\big)^{\frac{1}{CI_k-CI_i^*}}\cdot g^{r_k'}.
\end{aligned}$$

   Note that it cannot create a private key for $ID|_k \in$ **Prefix**$(ID^*|_\ell)$ since $CI_k = CI_k'$ from Lemma 3.1.

3. It derives $SK_{ID|_k}$ by running **HIBE.DeriveKey**$(ISK_{ID|_k},PP)$ and gives $SK_{ID|_k}$ to $\mathcal{A}$.

Challenge: For all $i \in [\ell]$, $\mathcal{B}$ computes the following ciphertext components by implicitly setting $t = z$ and $\{s_i = b_i\}$ as

$$C_{i,1} = g^{b_i},$$
$$C_{i,2} = \left(u^{CI_i^*} h\right)^{b_i} \cdot w^{-z} = \left(g^{u'CI_i^* + h'} \cdot \prod_{j=1}^{\ell} \left(\left(g^{y/b_j^2}\right)^{CI_i^* - CI_j^*} \cdot g^{xz/b_j}\right)\right)^{b_i} \cdot g^{-xz}$$
$$= \left(g^{b_i}\right)^{u'CI_i^* + h'} \cdot \prod_{j=1, j\neq i}^{\ell} \left(\left(g^{yb_i/b_j^2}\right)^{CI_i^* - CI_j^*} \cdot g^{xzb_i/b_j}\right).$$

It gives the challenge ciphertext header $CH_{ID^*|_\ell} = \left(C_0 = g^z, \{C_{i,1}, C_{i,2}\}_{i=1}^{\ell}\right)$ and the challenge session key $EK^* = Z$ to $\mathcal{A}$.

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

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

Remark 3. In the security proof, the simulator first creates an intermediate private key $ISK_{ID|_k}$ and then derives a private key $SK_{ID|_k}$ by simply running the **DeriveKey** algorithm instead of directly creating the private key. As mentioned before, this different key generation is possible because of the second property of the **EncodeCID** function. This feature of the simulator plays an important role in the proof of our RHIBE schemes.

Theorem 3.3 ([2]). *The above IBE scheme is SE-IND-CPA secure if the DBDH assumption holds.*

3.6 Discussions

Delegation History Dependence. The security models of HIBE are divided into two types depending on the process of delegation. In a security model with delegation history independence [2,3,13], the distribution of private keys is independent of the history of private key queries from an adversary. That is, the distribution of private keys generated from the root is the same as that of private keys delegated by a parent. In a security model with delegation history dependence [36,38], the distribution of private keys in dependent of the history of adversary's private key queries. In this model, an adversary can queries create, delegate, and reveal queries. If we use the security model with delegation history dependence, then we can reduce the size of private keys and simplify the process of re-randomization. That is, a challenger can use the **GenIKey** algorithm instead of the **GenKey** algorithm for the create query without using the **DeriveKey** algorithm.

Different Constructions. We observe that some of previous HIBE schemes can be modified to have additional algorithms and to support the generation of short intermediate private keys. We first consider the HIBE scheme of Boneh and Boyen (BB-HIBE) [2,3]. This scheme can be easily modified to have the **ChangeKey** and **MergeKey** algorithms since it belongs to the commutative blinding method [6], and it also can have the **GenIKey** and **DeriveKey** algorithms if the concatenated identity encoding function is used since the private key of the scheme uses different random values for each level. Next, we consider the HIBE scheme of Boneh, Boyen, and Goh (BBG-HIBE) [4]. This scheme also supports the **ChangeKey** and **MergeKey** algorithms, but it cannot have the **GenIKey** and **DeriveKey** algorithms since only one random value is used in a private key.

4 Revocable HIBE from Complete Subtree

In this section, we propose an RHIBE scheme with shorter keys by combining our HIBE scheme, the IBE scheme, and the complete subtree scheme.

4.1 The CS Scheme

The complete subtree (CS) scheme is one instance of the subset cover framework of Naor et al. [26]. We follow the definition of Lee et al. [19]. The CS scheme is given as follows:

CS.Setup(N_{max}): Let $N_{max} = 2^n$ for simplicity. It first sets a full binary tree $\mathcal{BT}$ of depth n. Each user is assigned to a different leaf node in $\mathcal{BT}$. 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 $v_i \in \mathcal{BT}$. It outputs the full binary tree $\mathcal{BT}$.

CS.Assign($\mathcal{BT}, ID$): Let v_{ID} be a leaf node of $\mathcal{BT}$ that is assigned to the user ID. 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 $v_{k_n} = v_{ID}$. For all $j \in \{k_0, \ldots, k_n\}$, it adds S_j into PV_{ID}. It outputs the private set $PV_{ID} = \{S_j\}$.

CS.Cover($\mathcal{BT}, R$): It first computes the Steiner tree $ST(R)$. 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 $ST(R)$. For all $i \in \{k_1, \ldots, k_m\}$, it adds S_i into CV_R. It outputs a covering set $CV_R = \{S_i\}$.

CS.Match(CV_R, PV_{ID}): It finds a subset S_k with $S_k \in CV_R$ and $S_k \in PV_{ID}$. If there is such a subset, it outputs (S_k, S_k). Otherwise, it outputs $\perp$.

We define **Label**(S_i) as a function that uniquely maps a subset $S_i \in \mathcal{S}$ to a label string L_i. We also define **Path**(ID) as the set of subsets $\{S_j\}$ associated with path nodes from the root node to a leaf node for ID in $\mathcal{BT}$.

4.2 Construction

The design approach of RHIBE schemes can be divided into the history-preserving update method and the history-free update method depending on the generation of update keys [30, 33]. In the history-preserving update method [30], an update key should include all update keys generated by his ancestor identities and a private key also includes all private keys of his ancestor identities because of the update key. In the history-free update method [33], an update key is generated from a decryption key that can be derived from a private key and a parent's update key if the private key is not revoked in the parent's update key. One nice property of the history-free update method is that a private key does not need to include all private keys of ancestor's identities. Thus, we follow the history-free update approach of Seo and Emura [33]. Note that a private key and an update key of the RHIBE scheme of Seo and Emura consists of $O(\ell \log N)$ group elements and $O(\ell r \log \frac{N}{r})$ group elements respectively.

In contrast to the previous ad-hoc design approach that builds an RHIBE scheme by combining an HIBE scheme and a CS scheme, we propose a modular design approach that builds an RHIBE scheme by combining an HIBE scheme, an IBE scheme, and a CS scheme. That is, we define each interface of HIBE, IBE, and CS schemes and we build an RHIBE scheme by just calling the interfaces of each underlying schemes. The main advantages of this modular approach is the simplicity and the reusability. In particular, we show that a private key of RHIBE can use an intermediate private key of HIBE instead of using a private key of HIBE. Recall that the intermediate private key of HIBE is shorter than the private key of

HIBE. Thus, we can reduce the size of an RHIBE private key from $O(\ell \log N)$ group elements to $O(\log N)$ group elements. Furthermore, we reduce the size of an update key from $O(\ell r \log \frac{N}{r})$ to $O(\ell + r \log \frac{N}{r})$ group elements by taking advantage of the modular design approach.

Let **HIBE** be the scheme in Section 3.3 and **IBE** be the scheme in Section 3.4. Our RHIBE scheme from CS is described as follows:

RHIBE.Setup$(1^\lambda, N_{max})$**:** This algorithm takes as input a security parameter 1^λ and the maximum number of users N_{max} for each level.

1. It first generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p. Let g be a generator of $\mathbb{G}$. It sets $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$. It obtains MK_{HIBE} and PP_{HIBE} by running **HIBE.Setup**(GDS). It also obtains MK_{IBE} and PP_{IBE} by running **IBE.Setup**(GDS).
2. It selects a random exponent $\alpha \in \mathbb{Z}_p$ and outputs a master key $MK = \alpha$ and public parameters $PP = \left(PP_{HIBE}, PP_{IBE}, \Omega = e(g,g)^\alpha, N_{max}\right)$. For notational simplicity, we define $SK_{ID|_0} = MK$.

Note that MK_{HIBE} and MK_{IBE} are discarded in RHIBE since secret shares of α are used for these master keys.

RHIBE.GenKey$(ID|_k, ST_{ID|_{k-1}}, PP)$**:** This algorithm takes as input a hierarchical identity $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ with $k \geq 1$, the state $ST_{ID|_{k-1}}$, and public parameters PP.

1. If $ST_{ID|_{k-1}}$ is empty (since it is first called), then it obtains $\mathcal{BT}_{ID|_{k-1}}$ by running **CS.Setup**(N_{max}) and generates a false master key $\beta_{ID|_{k-1}}$ and a PRF key $z_{ID|_{k-1}}$. Next, it sets $ST_{ID|_{k-1}} = (\mathcal{BT}_{ID|_{k-1}}, \beta_{ID|_{k-1}}, z_{ID|_{k-1}})$.
2. It assigns $ID|_k$ to a random leaf node $v \in \mathcal{BT}_{ID|_{k-1}}$ and obtains a private set $PV_{ID|_k} = \{S_j\}$ by running **CS.Assign**$(\mathcal{BT}_{ID|_{k-1}}, ID|_k)$.
3. For each $S_j \in PV_{ID|_k}$, it computes $\gamma_j = \mathbf{PRF}(z_{ID|_{k-1}}, L_j)$ where $L_j = \mathbf{Label}(S_j)$ and obtains ISK_{HIBE,S_j} by running **HIBE.GenIKey**$(ID|_k, \gamma_j, PP_{HIBE})$.
4. Finally, it outputs a private key $SK_{ID|_k} = \left(PV_{ID|_k}, \{ISK_{HIBE,S_j}\}_{S_j \in PV_{ID|_k}}\right)$. Note that the master key part of ISK_{HIBE,S_j} is γ_j.

RHIBE.UpdateKey$(T, RL_{ID|_{k-1}}, DK_{ID|_{k-1},T}, ST_{ID|_{k-1}}, PP)$**:** This algorithm takes as input time $T \in \mathcal{T}$, the revocation list $RL_{ID|_{k-1}}$, the decryption key $DK_{ID|_{k-1},T} = (RSK_{HIBE,ID|_{k-1}}, RSK_{IBE,T})$, the state $ST_{ID|_{k-1}} = (\mathcal{BT}_{ID|_{k-1}}, \beta_{ID|_{k-1}}, z_{ID|_{k-1}})$ with $k \geq 1$, and public parameters PP.

1. It obtains $RDK_{ID|_{k-1},T} = (RSK_{HIBE}, RSK_{IBE})$ by running **RHIBE.RandDK**$(DK_{ID|_{k-1},T}, -\beta_{ID|_{k-1}}, PP)$.
2. It derives the set $R_{ID|_{k-1}}$ of revoked identities at time T from $RL_{ID|_{k-1}}$. Next, it obtains a covering set $CV_{R_{ID|_{k-1}}} = \{S_i\}$ by running **CS.Cover**$(\mathcal{BT}_{ID|_{k-1}}, R_{ID|_{k-1}})$.
3. For each $S_i \in CV_{R_{ID|_{k-1}}}$, it computes $\gamma_i = \mathbf{PRF}(z_{ID|_{k-1}}, L_i)$ where $L_i = \mathbf{Label}(S_i)$ and obtains SK_{IBE,S_i} by running **IBE.GenKey**$(T, \beta_{ID|_{k-1}} - \gamma_i, PP_{IBE})$.
4. Finally, it outputs an update key $UK_{T,R_{ID|_{k-1}}} = \left(RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{IBE,S_i}\}_{S_i \in CV_{R_{ID|_{k-1}}}}\right)$. Note that the master key parts of RSK_{HIBE}, RSK_{IBE}, and SK_{IBE,S_i} are η', $\alpha - \eta' - \beta_{ID|_{k-1}}$, and $\beta_{ID|_{k-1}} - \gamma_i$ for some random η' respectively.

RHIBE.DeriveKey$(ID|_k, T, SK_{ID|_k}, UK_{T,R_{ID|_{k-1}}}, PP)$**:** This algorithm takes as input a hierarchical identity $ID|_k$ with $k \geq 0$, time T, a private key $SK_{ID|_k} = (PV_{ID|_k}, \{ISK_{HIBE,S_j}\}_{S_j \in PV_{ID|_k}})$, an update key $UK_{T,R_{ID|_{k-1}}} = (RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{IBE,S_i}\}_{S_i \in CV_{R_{ID|_{k-1}}}})$ where $RDK_{ID|_{k-1},T} = (RSK'_{HIBE}, RSK'_{IBE})$, and the public parameters PP.

If $k = 0$, then $SK_{ID|_0} = MK = \alpha$ and UK is empty. It proceeds as follows:

1. It selects a random exponent $\eta \in \mathbb{Z}_p$. It then obtains $RSK_{HIBE,ID|_0}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_0, \eta, PP_{HIBE})$ and **IBE.GenKey**$(T, \alpha - \eta, PP_{IBE})$ respectively.
2. It outputs a decryption key $DK_{ID|_0,T} = (RSK_{HIBE,ID|_0}, RSK_{IBE,T})$.

If $k \geq 1$, then it proceeds as follows:

1. If $ID|_k \notin R_{ID|_{k-1}}$, then it obtains (S_i, S_i) by running **CS.Match**$(CV_{R_{ID|_{k-1}}}, PV_{ID|_k})$. Otherwise, it outputs $\perp$. Next, it retrieves ISK_{HIBE,S_i} from $SK_{ID|_k}$ and SK_{IBE,S_i} from $UK_{T,R_{ID|_{k-1}}}$. It derives SK_{HIBE,S_i} by running **HIBE.DeriveKey**$(ISK_{HIBE,S_i}, PP_{HIBE})$.
2. It obtains RSK''_{HIBE} by running **HIBE.Delegate**$(ID|_k, RSK'_{HIBE}, PP_{HIBE})$ since RSK'_{HIBE} is for $ID|_{k-1}$. It selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ by running **HIBE.MergeKey**$(RSK''_{HIBE}, SK_{HIBE,S_i}, \eta, PP_{HIBE})$ and **IBE.MergeKey**$(RSK'_{IBE}, SK_{IBE,S_i}, -\eta, PP_{IBE})$ respectively.
3. Finally, it outputs a decryption key $DK_{ID|_k,T} = \left(RSK_{HIBE,ID|_k}, RSK_{IBE,T}\right)$.

Note that the master key parts of $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ are η' and $\alpha - \eta'$ for some random η' respectively.

RHIBE.RandDK$(DK_{ID|_k,T}, \beta, PP)$**:** This algorithm takes as input a decryption key $DK_{ID|_k,T} = \left(RSK'_{HIBE,ID|_k}, RSK'_{IBE,T}\right)$, an exponent $\beta \in \mathbb{Z}_p$, and the public parameters PP. It first selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ by running **HIBE.ChangeKey**$(RSK'_{HIBE,ID|_k}, (+, \eta), PP_{HIBE})$ and **IBE.ChangeKey**$(RSK'_{IBE,T}, (+, -\eta + \beta), PP_{IBE})$ respectively. It outputs a randomized decryption key $DK_{ID|_k,T} = \left(RSK_{HIBE,ID|_k}, RSK_{IBE,T}\right)$. Note that the master key parts of $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ are η' and $\alpha - \eta' + \beta$ respectively.

RHIBE.Encrypt$(ID|_\ell, T, M, PP)$**:** This algorithm takes as input a hierarchical identity $ID|_\ell = (I_1, \ldots, I_\ell) \in \mathcal{I}^\ell$ with $\ell \geq 1$, time T, a message M, and the public parameters PP. It first chooses a random exponent $t \in \mathbb{Z}_p$. Next, it obtains $CH_{HIBE,ID|_\ell}$ and EK_{HIBE} by running **HIBE.Encaps**$(ID|_\ell, t, PP_{HIBE})$. It also obtains $CH_{IBE,T}$ and EK_{IBE} by running **IBE.Encaps**(T, t, PP_{IBE}). It outputs a ciphertext $CT_{ID|_k,T} = \left(CH_{HIBE,ID|_\ell}, CH_{IBE,T}, C = \Omega^t \cdot M\right)$.

RHIBE.Decrypt$(CT_{ID|_\ell,T}, DK_{ID'|_k,T'}, PP)$**:** This algorithm takes as input a ciphertext $CT_{ID|_\ell,T} = (CH_{HIBE,ID|_\ell}, CH_{IBE,T}, C)$ with $\ell \geq 1$, a decryption key $DK_{ID|_k,T'} = (RSK_{HIBE,ID|_k}, RSK_{IBE,T'})$ with $k \geq 1$, and the public parameters PP. If $ID|_k \in$ **Prefix**$(ID|_\ell)$ and $T = T'$, then it obtains EK_{HIBE} and EK_{IBE} by running **HIBE.Decaps**$(CH_{HIBE,ID|_\ell}, RSK_{HIBE,ID|_k}, PP_{HIBE})$ and **IBE.Decaps**$(CH_{IBE,T}, RSK_{IBE,T'}, PP_{IBE})$ respectively. Otherwise, it outputs $\perp$. It outputs an encrypted message $M = C \cdot (EK_{HIBE} \cdot EK_{IBE})^{-1}$.

RHIBE.Revoke$(ID|_k, T, RL_{ID|_{k-1}}, ST_{ID|_{k-1}})$**:** This algorithm takes as input a hierarchical identity $ID|_k$ with $k \geq 1$, revocation time T, a revocation list $RL_{ID|_{k-1}}$, and a state $ST_{ID|_{k-1}} = (\mathcal{BT}_{ID|_{k-1}}, z)$. If $ID|_k$ is not assigned in $\mathcal{BT}_{ID|_{k-1}}$, then it outputs $\perp$. Otherwise, it updates $RL_{ID|_{k-1}}$ by adding $(ID|_k, T)$ to $RL_{ID|_{k-1}}$.

4.3 Correctness

To show the correctness of the above RHIBE scheme, we first show that a decryption key $DK_{ID|_k,T}$ is correctly derived from a private key $SK_{ID|_k}$ and an update key $UK_{T,R_{ID|_{k-1}}}$. Let $SK_{ID|_k} = (PV_{ID|_k}, \{ISK_{HIBE,S_j}\}_{S_j \in PV})$ and $UK_{T,R_{ID|_{k-1}}} = (RDK_{ID|_{k-1},T}, CV_R, \{SK_{IBE,S_i}\}_{S_i \in CV})$ where $RDK_{ID|_{k-1},T} = (RSK'_{HIBE}, RSK'_{IBE})$. From the **GenKey** and **UpdateKey** algorithms, the master key parts of ISK_{HIBE,S_j} and SK_{IBE,S_i} are associated with γ_j and $\beta_{ID|_{k-1}} - \gamma_i$ respectively. Additionally, the master key parts of RSK'_{HIBE} and RSK'_{IBE} are associated with η' and $\alpha - \eta' - \beta_{ID|_{k-1}}$ respectively.

If $ID|_k \notin R_{ID|_{k-1}}$, then the master key parts of ISK_{HIBE,S_i} and SK_{IBE,S_i} are associated with γ_i and $\beta_{ID|_{k-1}} - \gamma_i$ since there keys are related to the same tree node because of the correctness of the CS scheme. The master key part of SK'_{HIBE,S_i} derived from **HIBE.DeriveKey** still associated with γ_i. Thus, the master key part of RSK''_{HIBE} and RSK'_{IBE} that are returned by **HIBE.MergeKey** and **IBE.MergeKey** are associated with $\eta'' = (\eta') + \gamma_i + \eta$ and $(\alpha - \eta' - \beta_{ID|_{k-1}}) + \beta_{ID|_{k-1}} - \gamma_i - \eta = \alpha - \eta''$ respectively. Thus the **DeriveKey** algorithm is correct since we have α if we add two master key parts of the decryption key.

Next, we show that the message is correctly decrypted by the decryption algorithm. The correctness of the **Decrypt** algorithm can be shown by the correctness of the **HIBE.Decrypt** and **IBE.Decrypt**. That is, we have $e(g,g)^{\eta' \cdot t}$ from the correctness of HIBE and $e(g,g)^{(\alpha - \eta')t}$ from the correctness of IBE. By adding two partial session keys, we have $e(g,g)^{\alpha t}$. The message M can be easily obtained by using this session key.

4.4 Security Analysis

Theorem 4.1. *The above RHIBE scheme from CS is SE-IND-CPA secure if the PRF scheme is secure and the q-RW2 assumption holds.*

Proof. Let $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$ be the challenge hierarchical identity submitted by an adversary. To prove the selective IND-CPA security of our RHIBE scheme, we classify the type of adversaries into $\ell + 1$ types. The type of an adversary $\mathcal{A}$ is $\tau \in \{1, \ldots, \ell, \ell+1\}$ if $\mathcal{A}$ does not queries the private key of $ID^*|_j$ for all $j \in \{1, \ldots, \tau - 1\}$, but $\mathcal{A}$ should query the private key of $ID^*|_\tau$. That is, the oldest ancestor of $ID^*|_\ell$ which is queried by $\mathcal{A}$ is $ID^*|_\tau$. If the type of an adversary is $\ell + 1$, then $\mathcal{A}$ does not query the private key of $ID^*|_j$ for all $j \in \{1, \ldots, \ell\}$.

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

Game $\mathbf{G}_0$. This game is the original security game. That is, a simulator $\mathcal{B}$ uses a pseudo-random function for each hierarchical identity $ID|_{k-1}$ when it generates private keys for child identities and update keys for time periods.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is similar to the game $\mathbf{G}_0$ except that $\mathcal{B}$ uses a truly random function for the hierarchical identity $ID^*|_{k-1}$ for all $k \in \{1, \ldots, \tau\}$ when it generates private keys and update keys. That is, $\mathcal{B}$ selects a random element $\gamma_i \in \mathbb{Z}_p$ instead of computing $\gamma_i = \mathbf{PRF}(z_{ID^*|_{\tau-1}}, \mathbf{Label}(S_j))$ for each S_i in $\mathcal{BT}_{ID^*|_{k-1}}$.

Game $\mathbf{G}_2$. This final game $\mathbf{G}_2$ is almost the same with the previous game $\mathbf{G}_1$ except that a random session key is used to create the challenge ciphertext. Note that the advantage of $\mathcal{A}_\tau$ in this game is zero since the challenge ciphertext is not related to μ.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_i}$ be the advantage of $\mathcal{A}$ in a game $\mathbf{G}_i$. Let E_τ be the event that the adversary behaves like τ-type. From the Lemmas 4.2 and 4.3, we obtain the following equation

$$\left|\mathbf{Adv}_{\mathcal{A}}^{G_0} - \mathbf{Adv}_{\mathcal{A}}^{G_2}\right| \leq \sum_{\tau=1}^{\ell+1} \Pr[E_\tau] \cdot \left|\mathbf{Adv}_{\mathcal{A}_\tau}^{G_0} - \mathbf{Adv}_{\mathcal{A}_\tau}^{G_2}\right| \leq \ell \mathbf{Adv}_{\mathcal{B}}^{PRF}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{q\text{-}RW2}(\lambda).$$

This completes our proof. □

Lemma 4.2. *If the PRF scheme is secure, then no PPT τ-type adversary can distinguish between $\boldsymbol{G}_0$ and $\boldsymbol{G}_1$ with a non-negligible advantage.*

Proof. The proof of this lemma is relatively straightforward by using additional hybrid arguments if a simulator that distinguishes whether an oracle is PRF or not selects the master key of RHIBE by himself and uses the given oracle for the hierarchical identity $ID^*|_{k-1}$ when it generates private keys for child identities and update keys for time periods. We omit the details of this proof. □

Lemma 4.3. *If the q-RW2 assumption holds, then no PPT τ-type adversary can distinguish between $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ with a non-negligible advantage.*

Proof. Suppose there exists a τ-type adversary $\mathcal{A}_\tau$ that attacks the above RHIBE scheme with non-negligible advantage. A meta-simulator $\mathcal{B}$ that solves the q-RW2 assumption using $\mathcal{A}_\tau$ is given: a challenge tuple $D = \big((p,\mathbb{G},\mathbb{G}_T,e),g,g^x,g^y,g^z,g^{(xz)^2},\{g^{b_i},g^{xzb_i},g^{xz/b_i},g^{x^2zb_i},g^{y/b_i^2},g^{y^2/b_i^2}\},\{g^{xzb_i/b_j},g^{yb_i/b_j^2},g^{xyzb_i/b_j^2},g^{(xz)^2b_i/b_j}\}\big)$ and Z where $Z = Z_0 = e(g,g)^{xyz}$ or $Z = Z_1 \in_R \mathbb{G}_T$. Note that a challenge tuple $D_{DBDH} = ((p,\mathbb{G},\mathbb{G}_T,e),g,g^x,g^y,g^z)$ for the DBDH assumption can be derived from the challenge tuple D of the q-RW2 assumption. Let $\mathcal{B}_{HIBE}$ be the simulator of Theorem 3.2 and $\mathcal{B}_{IBE}$ be the simulator of Theorem 3.3. Then $\mathcal{B}$ that interacts with $\mathcal{A}_\tau$ is described as follows:

Init: $\mathcal{A}_\tau$ initially submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*,\ldots,I_\ell^*)$ and challenge time T^*. $\mathcal{B}$ runs $\mathcal{B}_{HIBE}$ by giving D and Z, and it also runs $\mathcal{B}_{IBE}$ by giving D_{DBDH} and Z.

Setup: $\mathcal{B}$ submits $ID^*|_\ell$ to $\mathcal{B}_{HIBE}$ and receives PP_{HIBE}. It also submits T^* to $\mathcal{B}_{IBE}$ and receives PP_{IBE}. It fixes a random leaf node $v^* \in \mathcal{BT}_{ID^*|_{\tau-1}}$ that will be assigned to the hierarchical identity $ID^*|_\tau$. It implicitly sets $\alpha = xy$ and gives the public parameters $PP = \big(PP_{HIBE}, PP_{IBE}, \Omega = e(g^x,g^y), N_{max}\big)$ to $\mathcal{A}_\tau$.

Phase 1: $\mathcal{A}_\tau$ adaptively requests a polynomial number of private key, update key, decryption key, and revocation queries. $\mathcal{B}$ handles these queries as follows:

If this is a private key query for $ID|_k = (I_1,\ldots,I_{k-1},I_k)$ with $k \geq 1$, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$: In this case, it normally generates a private key by using a normally generated $ST_{ID|_{k-1}}$ since it can obtains $DK_{ID|_{k-1},T}$ for any T.
  1. If the state $ST_{ID|_{k-1}}$ was not generated before, then it normally generates $ST_{ID|_{k-1}}$. Otherwise, it retrieves $ST_{ID|_{k-1}}$ that was previously generated.
  2. It obtains $SK_{ID|_k}$ by running **RHIBE.GenKey**$(ID|_k, ST_{ID|_{k-1}}, PP)$. Additionally, it normally generates $ST_{ID|_k}$ by himself. Finally, it gives $SK_{ID|_k}$ and $ST_{ID|_k}$ to $\mathcal{A}_\tau$.
- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k < \tau$: In this case, we have $ID|_k \neq ID^*|_k$ since $\mathcal{A}_\tau$ is τ-type.
  1. It queries an HIBE intermediate private key for $ID|_k$ to $\mathcal{B}_{HIBE}$ and receives ISK'_{HIBE}.
  2. It assigns $ID|_k$ to a random leaf node $v \in \mathcal{BT}_{ID^*|_{k-1}}$ and obtains $PV_{ID|_k} = \{S_j\}$ by running **CS.Assign**$(\mathcal{BT}_{ID^*|_{k-1}}, ID|_k)$.

3. For each $S_j \in PV_{ID|_k}$, it retrieves random γ_j associated to the node v_j and obtains ISK_{HIBE,S_j} by running **HIBE.ChangeIKey**$(ISK'_{HIBE}, (+, \gamma_j), PP_{HIBE})$.
4. It sets $SK_{ID|_k} = \left(PV_{ID|_k}, \{ISK_{HIBE,S_j}\}_{S_j \in PV_{ID|_k}}\right)$ and gives $SK_{ID|_k}$ to $\mathcal{A}_\tau$. Note that the master key part of ISK_{HIBE,S_j} is $\alpha + \gamma_j$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k = \tau$: In this case, it carefully divides the subsets in $\mathcal{BT}_{ID^*|_{\tau-1}}$ into two partitions. For one partition ($S_j \in$ **Path**$(ID^*|_\tau)$), it sets γ_j as the master key part. For another partition ($S_j \notin$ **Path**$(ID^*|_\tau)$), it sets $\alpha + \gamma_j$ as the master key part by using $\mathcal{B}_{HIBE}$.
  1. If $ID|_k \neq ID^*|_\tau$, it queries an HIBE intermediate private key for $ID|_k$ to $\mathcal{B}_{HIBE}$ and receives ISK'_{HIBE}.
  2. If $ID|_k \neq ID^*|_\tau$, it assigns $ID|_k$ to a random leaf node $v \in \mathcal{BT}_{ID^*|_{\tau-1}} (v \neq v^*)$. Otherwise ($ID|_k = ID^*|_\tau$), it assigns $ID^*|_\tau$ to the pre-fixed node v^*. Next, it obtains $PV_{ID|_k} = \{S_j\}$ by running **CS.Assign**$(\mathcal{BT}_{ID^*|_{\tau-1}}, ID|_k)$.
  3. For each $S_j \in PV_{ID|_k}$, it retrieves random γ_j that is associated to v_j and proceeds as follows:
     - Case $S_j \in$ **Path**$(ID^*|_\tau)$: It obtains ISK_{HIBE,S_j} by running **HIBE.GenIKey**$(ID|_k, \gamma_j, PP_{HIBE})$.
     - Case $S_j \notin$ **Path**$(ID^*|_\tau)$: It obtains ISK_{HIBE,S_j} by running **HIBE.ChangeIKey**$(ISK'_{HIBE}, (+, \gamma_j), PP_{HIBE})$.
  4. It sets $SK_{ID|_k} = \left(PV_{ID|_k}, \{ISK_{HIBE,S_j}\}_{S_j \in PV_{ID|_k}}\right)$ and gives $SK_{ID|_k}$ to $\mathcal{A}_\tau$. Note that the master key part of ISK_{HIBE,S_j} is γ_j if $S_j \in$ **Path**$(ID^*|_\tau)$ or it is $\alpha + \gamma_j$ otherwise.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k > \tau$: In this case, it is the same as the case $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$. That is, it follows the normal key generation algorithm. We omit the description of this case.

If this is an update key query for $ID|_{k-1} = (I_1, \ldots, I_{k-1})$ with $k \geq 1$ and T, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$: In this case, it can generate $UK_{T,R_{ID|_{k-1}}}$ by following the normal algorithm since it can obtain $DK_{ID|_{k-1},T}$ from the condition $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$.
  1. If the state $ST_{ID|_{k-1}}$ was generated before, then it retrieves $ST_{ID|_{k-1}}$. Otherwise, it normally generates $ST_{ID|_{k-1}}$ by himself. Next, it requests a decryption key query for $ID|_{k-1}$ and T to himself and receives $DK_{ID|_{k-1},T}$ since $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$.
  2. It obtains $UK_{T,R_{ID|_{k-1}}}$ by running **RHIBE.UpdateKey**$(T, RL_{ID|_{k-1}}, DK_{ID_{k-1},T}, ST_{ID|_{k-1}}, PP)$ and gives $UK_{T,R_{ID|_{k-1}}}$ to $\mathcal{A}_\tau$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k < \tau$: In this case, it can use the fact that $\mathcal{A}_\tau$ is τ-type.
  1. It selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_{k-1}}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_{k-1}, \eta, PP_{HIBE})$ and **IBE.GenKey**$(T, -\eta - \beta_{ID|_{k-1}}, PP_{IBE})$ respectively. It sets $RDK_{ID|_{k-1},T} = (RSK_{HIBE,ID|_{k-1}}, RSK_{IBE,T})$.
  2. It derives the set $R_{ID|_{k-1}}$ of revoked identities at time T from $RL_{ID|_{k-1}}$ and obtains $CV_{R_{ID|_{k-1}}} = \{S_i\}$ by running **CS.Cover**$(\mathcal{BT}_{ID|_{k-1}}, R_{ID|_{k-1}})$.
  3. For each $S_i \in CV_{R_{ID|_{k-1}}}$, it retrieves random γ_i associated to v_i and obtains SK_{IBE,S_i} by running **IBE.GenKey**$(ID|_{k-1}, \beta_{ID|_{k-1}} - \gamma_i, PP_{IBE})$.

4. It sets $UK_{T,R_{ID|_{k-1}}} = \left(RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{HIBE,S_i}\}_{S_i \in CV_{R_{ID|_{k-1}}}}\right)$ and gives $UK_{T,R_{ID|_{k-1}}}$ to $\mathcal{A}_\tau$. Note that the master key part of SK_{IBE,S_i} is $\beta_{ID|_{k-1}} - \gamma_i$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k = \tau$: In this case, $\mathcal{A}_\tau$ requested the private key of $ID^*|_\tau$, but the private key should be revoked in $\mathcal{BT}_{ID^*|_{\tau-1}}$ on the time T^*.

  1. It selects random exponents $\eta \in \mathbb{Z}_p$. Next it obtains $RSK_{HIBE,ID|_{k-1}}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_{k-1}, \eta, PP_{HIBE})$ and **IBE.GenKey**$(T, -\eta - \beta_{ID|_{k-1}}, PP_{IBE})$ respectively. It sets $RDK_{ID|_{k-1},T} = (RSK_{HIBE,ID|_{k-1}}, RSK_{IBE,T})$. If $T \neq T^*$, it additionally queries an IBE private key for T to $\mathcal{B}_{IBE}$ and receives $SK'_{IBE,T}$.
  2. It derives the set $R_{ID|_{k-1}}$ of revoked identities at time T from $RL_{ID|_{k-1}}$ and obtains $CV_{R_{ID|_{k-1}}} = \{S_i\}$ by running **CS.Cover**$(\mathcal{BT}_{ID|_{k-1}}, R_{ID|_{k-1}})$.
  3. For each $S_i \in CV_{R_{ID|_{k-1}}}$, it retrieves γ_i associated to v_i and proceeds as follows:
     - Case $S_i \in$ **Path**$(ID^*|_\tau)$: It obtains SK_{IBE,S_i} by running **IBE.ChangeKey**$(SK'_{IBE,T}, (+, \beta_{ID|_{k-1}} - \gamma_i), PP_{IBE})$ since $T \neq T^*$.
     - Case $S_i \notin$ **Path**$(ID^*|_\tau)$: It obtains SK_{IBE,S_i} by running **IBE.GenKey**$(T, \beta_{ID|_{k-1}} - \gamma_i, PP_{IBE})$.
  4. It sets $UK_{T,R_{ID|_{k-1}}} = \left(RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{IBE,S_i}\}_{S_i \in CV_{R_{ID|_{k-1}}}}\right)$ and gives $UK_{T,R_{ID|_{k-1}}}$ to $\mathcal{A}_\tau$. Note that the master key part of SK_{IBE,S_i} is $\alpha + \beta_{ID|_{k-1}} - \gamma_i$ if $S_i \in$ **Path**$(ID^*|_k)$ or it is $\beta_{ID|_{k-1}} - \gamma_i$ otherwise.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k > \tau$: In this case, it is almost the same as the first case $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$ except that it uses a decryption key by using the condition $T \neq T^*$. Note that $\mathcal{A}_\tau$ cannot query an update key on time T^* since $ID^*|_\tau$ was already revoked by the restriction of the security model. We omit the description of this case.

If this is a decryption key query for $ID|_k = (I_1, \ldots, I_k)$ with $k \geq 1$ and T, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID|_k \notin$ **Prefix**$(ID^*|_\ell)$: In this case, it can use $\mathcal{B}_{HIBE}$ to generate g^α since $ID|_k \notin$ **Prefix**$(ID^*|_\ell)$.

  1. It first queries an HIBE private key for $ID|_k$ to $\mathcal{B}_{HIBE}$ and receives $SK'_{HIBE,ID|_k}$.
  2. It selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ by running **HIBE.ChangeKey**$(SK'_{HIBE,ID|_k}, (+, \eta), PP_{HIBE})$ and **IBE.GenKey**$(T, -\eta, PP_{IBE})$ respectively.
  3. It gives $DK_{ID|_k,T} = \left(RSK_{HIBE,ID|_k}, RSK_{IBE,T}\right)$ to $\mathcal{A}_\tau$. Note that the master key part of $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ are $\alpha + \eta$ and $-\eta$ respectively.

- **Case** $ID|_k \in$ **Prefix**$(ID^*|_\ell)$ and $T \neq T^*$: In this case, it can use $\mathcal{B}_{IBE}$ to generates g^α since $T \neq T^*$.

  1. It first queries an IBE private key for T to $\mathcal{B}_{IBE}$ and receives $SK'_{IBE,T}$.
  2. It selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_k, \eta, PP_{HIBE})$ and **IBE.ChangeKey**$(SK'_{IBE,T}, (+, -\eta), PP_{IBE})$ respectively.
  3. It gives $DK_{ID|_k,T} = \left(RSK_{HIBE,ID|_k}, RSK_{IBE,T}\right)$ to $\mathcal{A}_\tau$. Note that the master key part of $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ are η and $\alpha - \eta$ respectively.

If this is a revocation query for $ID|_k$ and T, then $\mathcal{B}$ obtains an updated $RL_{ID|_{k-1}}$ by running **RHIBE.Revoke** $(ID|_k, T, RL_{ID|_{k-1}}, ST_{ID|_{k-1}})$. Note that $\mathcal{A}_\tau$ cannot query to revoke $ID|_k$ on time T if he already requested an update key query for $ID|_k$ on time T.

Challenge: $\mathcal{A}_\tau$ submits two challenge messages M_0^*, M_1^*. $\mathcal{B}$ flips a coin $\mu \in \{0,1\}$. Next, it requests the challenge ciphertext to $\mathcal{B}_{HIBE}$ and receives $CH_{HIBE,ID^*|_k}$ and EK_{HIBE}. It also requests the challenge ciphertext to $\mathcal{B}_{IBE}$ and receives CH_{IBE,T^*} and EK_{IBE}. It sets the challenge ciphertext $CT_{ID^*|_\ell,T^*} = \left(CH_{HIBE,ID^*|_\ell}, CH_{IBT,T^*}, Z \cdot M_\mu^*\right)$ and gives it to $\mathcal{A}_\tau$.

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

Guess: $\mathcal{A}_\tau$ outputs a guess $\mu' \in \{0,1\}$. If $\mu = \mu'$, then $\mathcal{B}$ outputs 0. Otherwise, $\mathcal{B}$ outputs 1.

To finish the proof, we should check that the public parameters, private keys, update keys, decryption keys, and the challenge ciphertext are all generated correctly. Recall that two sub-simulators $\mathcal{B}_{HIBE}$ and $\mathcal{B}_{IBE}$ correctly generate the public parameters, private keys, update keys, and challenge ciphertexts of HIBE and IBE respectively. The public parameters PP of RHIBE is correctly generated since two sub-simulators use the same g in the assumption to generate PP_{HIBE} and PP_{IBE} respectively. Note that $\mathcal{B}_{HIBE}$ implicitly sets $\gamma = \alpha = xy$ and $\mathcal{B}_{IBE}$ also implicitly sets $\beta = \alpha = xy$. It is also easy to show that decryption keys and the challenge ciphertext are correctly generated since $\mathcal{B}_{HIBE}$ and $\mathcal{B}_{IBE}$ correctly generate the private keys and the challenge ciphertexts of HIBE and IBE respectively.

We can check the generations of private keys and update keys by examining the consistency of the master key parts in private keys and update keys since the simulator uses HIBE and IBE schemes as modules. In cases of $ID|_{k-1} \notin \mathbf{Prefix}(ID^*|_\ell)$ and $ID|_{k-1} \in \mathbf{Prefix}(ID^*|_\ell)$ with $k > \tau$, private keys and update keys are correctly generated since they are normally generated. Next, we consider the case $ID|_{k-1} \in \mathbf{Prefix}(ID^*|_\ell)$ with $k < \tau$. The master key part of ISK_{HIBE,S_j} in a private key is $\alpha + \gamma_j$ and the master key part of SK_{IBE,S_j} in an update key is $\beta_{ID|_{k-1}} - \gamma_j$. Additionally, the master key parts of RSK_{HIBE}, RSK_{IBE} in an update key are η and $-\eta - \beta_{ID|_{k-1}}$ respectively. If we implicitly sets $\gamma_j' = \alpha + \gamma_j$ and $\beta'_{ID|_{k-1}} = \alpha + \beta_{ID|_{k-1}}$, then the master key parts of RSK_{HIBE}, RSK_{IBE} are η and $\alpha - \eta - \beta'$ respectively. The case of $ID|_{k-1} \in \mathbf{Prefix}(ID^*|_\ell)$ with $k = \tau$ is similar to the previous case except that the master key parts of ISK_{HIBE,S_j} and SK_{IBE,S_j} associated to a subset $S_j \in \mathbf{Path}(ID^*|_\tau)$ are differently set. That is, if $S_j \in \mathbf{Path}(ID^*|_\tau)$, then the master key parts of ISK_{HIBE,S_j} and SK_{IBE,S_j} are γ_j and $\alpha + \beta_{ID|_{k-1}} - \gamma_j$ respectively. The problem of this setting is that an update key on the time T^* cannot be generated if $CV \cap \mathbf{Path}(ID^*|_\tau) \neq \emptyset$. Fortunately, we have $CV \cap \mathbf{Path}(ID^*|_\tau) = \emptyset$ since the private key of $ID^*|_\tau$ should be revoked on the time T^* by the restriction of the security model. Thus the master key parts of private keys and update keys are consistent. This completes our proof. □

4.5 Discussions

Different Constructions. We can build different RHIBE schemes if we replace our HIBE scheme with other HIBE schemes if they can have the same interfaces of our HIBE scheme. At first, we can try to build an RHIBE scheme by using the BB-HIBE scheme [2]. As mentioned before, the BB-HIBE scheme can have the intermediate private keys. In this RHIBE scheme, the size of public parameters, a private key, an update key, and a ciphertext is $O(\ell)$, $O(\log N)$, $O(\ell + r\log\frac{N}{r})$, and $O(\ell)$ respectively. We can also try to build another RHIBE scheme by using the BBG-HIBE scheme [4]. Note that the BBG-HIBE scheme cannot have the intermediate private key. In this RHIBE scheme, the size of public parameters, a private key, an update key, and a ciphertext is $O(\ell)$, $O(\ell \log N)$, $O(\ell + r\log\frac{N}{r})$, and $O(1)$ respectively.

5 Revocable HIBE from Subset Difference

In this section, we propose another RHIBE scheme with shorter keys by combining our HIBE scheme, the IBE scheme, and the subset difference scheme.

5.1 The SD Scheme

The subset difference (SD) scheme is also another instance of the subset cover framework of Naor et al. [26]. We use the description of the SD scheme of Lee et al. [20]. The SD scheme is given as follows:

SD.Setup(N_{max}): Let $N_{max} = 2^n$ for simplicity. It first sets a full binary tree $\mathcal{BT}$ of depth n. Each user is assigned to a different leaf node in $\mathcal{BT}$. 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. It outputs the full binary tree $\mathcal{BT}$.

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

SD.Cover($\mathcal{BT}, R$): It first sets a subtree T as $ST(R)$, and then it builds a covering set CV_R iteratively by removing nodes from T until T consists of just a single node as follows:

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

It outputs the covering set $CV_R = \{S_{i,j}\}$.

SD.Match(CV_R, PV_{ID}): It finds two subsets $S_{i,j}$ and $S'_{i',j'}$ such that $S_{i,j} \in CV_R$, $S'_{i',j'} \in PV_{ID}$, $i = i'$, $d_j = d_{j'}$, and $j \neq j'$ where d_j is the depth of v_j. If it found two subsets, then it outputs $(S_{i,j}, S'_{i',j'})$. Otherwise, it outputs $\perp$.

We define $\mathbf{Label}(S_{i,j})$ as a function that uniquely maps a subset $S_{i,j} \in \mathcal{S}$ to label strings (L_i, L_j). We also define $\mathbf{Depth}(S_j)$ as a function that returns the depth d_j of the node v_j associated to S_j.

5.2 Construction

The construction of revocable IBE that uses the SD scheme instead of using the CS scheme was shown by Lee et al. [20]. By following the design principle of Lee et al., Seo and Emura proposed an RHIBE scheme [33]. First, we briefly review the design idea of Lee et al. In the SD scheme, a subset $S_{i,j}$ is defined as a set of leaf nodes in a subtree $\mathcal{T}_i$ associated with a subroot v_i by excluding a set of leaf nodes in another subtree $\mathcal{T}_j$ associated with a subroot v_j. Lee et al. interpreted the subset $S_{i,j}$ as a group with single member revocation. To implement a method for single member revocation that can be integrated with an IBE scheme, they applied a degree-one polynomial in exponents since a single point can be revoked in this polynomial and the Lagrange interpolation method works well in exponents.

We build an RHIBE-SD scheme by following the design principle of the RIBE-SD scheme of Lee et al. [20]. Similar to our RHIBE-CS scheme in the previous section, we construct an RHIBE-SD scheme by using the underlying HIBE and IBE schemes as modules. As mentioned before, we can simplify the construction and the security analysis of our RHIBE-SD scheme if we build it as a modular way. If our RHIBE-SD scheme is compared to our RHIBE-CS scheme, the number of group elements in an update key

is reduced from $O(\ell + r\log\frac{N}{r})$ to $O(\ell + r)$ but the number of group elements in a private key is increased from $O(\log N)$ to $O(\log^2 N)$. Note that we can reduce the number of group elements in a private key to $O(\log^{1.5} N)$ if the layered SD scheme is used.

Let $\Delta_{i,I}$ be a Lagrange coefficient which is defined as $\Delta_{i,I}(x) = \prod_{j\in I, j\neq i} \frac{x-j}{i-j}$ for an index $i \in \mathbb{Z}_p$ and a set of indexes I in $\mathbb{Z}_p$. Our RHIBE scheme from SD is described as follows:

RHIBE.Setup$(1^\lambda, N_{max})$**:** This algorithm takes as input a security parameter 1^λ and the maximum number of users N_{max} for each level.

1. It first generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p. Let g be a generator of $\mathbb{G}$. It sets $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$. It obtains MK_{HIBE} and PP_{HIBE} by running **HIBE.Setup**(GDS). It also obtains MK_{IBE} and PP_{IBE} by running **IBE.Setup**(GDS).
2. It selects a random exponent $\alpha \in \mathbb{Z}_p$ and outputs a master key $MK = \alpha$ and public parameters $PP = \big(PP_{HIBE}, PP_{IBE}, \Omega = e(g,g)^\alpha, N_{max}\big)$. For notational simplicity, we define $SK_{ID|_0} = MK$.

Note that MK_{HIBE} and MK_{IBE} are discarded in RHIBE since secret shares of α are used for these master keys.

RHIBE.GenKey$(ID|_k, ST_{ID|_{k-1}}, PP)$**:** This algorithm takes as input a hierarchical identity $ID|_k = (I_1, \ldots, I_k) \in \mathcal{I}^k$ with $k \geq 1$, the state $ST_{ID|_{k-1}}$, and public parameters PP.

1. If $ST_{ID|_{k-1}}$ is empty (since it is first called), then it obtains $\mathcal{BT}_{ID|_{k-1}}$ by running **SD.Setup**(N_{max}) and generates a false master key $\beta_{ID|_{k-1}} \in \mathbb{Z}_p$ and a PRF key $z_{ID|_{k-1}}$. Next, it sets $ST_{ID|_{k-1}} = (\mathcal{BT}_{ID|_{k-1}}, \beta_{ID|_{k-1}}, z_{ID|_{k-1}})$.
2. It assigns $ID|_k$ to a random leaf node $v \in \mathcal{BT}_{ID|_{k-1}}$ and obtains a private set $PV_{ID|_k} = \{S_{i,j}\}$ by running **SD.Assign**$(\mathcal{BT}_{ID|_{k-1}}, ID|_k)$.
3. For each $S_{i,j} \in PV_{ID|_k}$, it defines $f_{GL}(x) = a_{GL}x + \beta_{ID|_{k-1}}$ by computing $a_{GL} = \mathbf{PRF}(z_{ID|_{k-1}}, GL)$ where $(L_i, L_j) = \mathbf{Label}(S_{i,j})$, $d_j = \mathbf{Depth}(S_j)$, and $GL = L_i \| d_j$, and then it obtains ISK_{HIBE,S_j} by running **HIBE.GenIKey**$(ID|_k, f_{GL}(L_j), PP_{HIBE})$.
4. Finally, it outputs a private key $SK_{ID|_k} = \big(PV_{ID|_k}, \{ISK_{HIBE,S_{i,j}}\}_{S_{i,j}\in PV_{ID|_k}}\big)$. Note that the master key part of $SK_{HIBE,S_{i,j}}$ is $f_{GL}(L_j)$.

RHIBE.UpdateKey$(T, RL_{ID|_{k-1}}, DK_{ID|_{k-1},T}, ST_{ID|_{k-1}}, PP)$**:** This algorithm takes as input time $T \in \mathcal{T}$, the revocation list $RL_{ID|_{k-1}}$, the decryption key $DK_{ID|_{k-1},T} = (RSK_{HIBE,ID|_{k-1}}, RSK_{IBE,T})$, the state $ST_{ID|_{k-1}} = (\mathcal{BT}_{ID|_{k-1}}, \beta_{ID|_{k-1}}, z_{ID|_{k-1}})$ with $k \geq 1$, and public parameters PP.

1. It obtains $RDK_{ID|_{k-1},T} = (RSK_{HIBE}, RSK_{IBE})$ by running **RHIBE.RandDK**$(DK_{ID|_{k-1},T}, -\beta_{ID|_{k-1}}, PP)$.
2. It derives the set $R_{ID|_{k-1}}$ of revoked identities at time T from $RL_{ID|_{k-1}}$. Next, it obtains a covering set $CV_{R_{ID|_{k-1}}} = \{S_{i,j}\}$ by running **SD.Cover**$(\mathcal{BT}_{ID|_{k-1}}, R_{ID|_{k-1}})$.
3. For each $S_{i,j} \in CV_{R_{ID|_{k-1}}}$, it defines $f_{GL}(x) = a_{GL}x + \beta_{ID|_{k-1}}$ by computing $a_{GL} = \mathbf{PRF}(z_{ID|_{k-1}}, GL)$ where $(L_i, L_j) = \mathbf{Label}(S_{i,j})$, $d_j = \mathbf{Depth}(S_j)$, and $GL = L_i \| d_j$, and then it obtains $SK_{IBE,S_{i,j}}$ by running **IBE.GenKey**$(T, f_{GL}(L_j), PP_{IBE})$.
4. Finally, it outputs an update key $UK_{T,R_{ID|_{k-1}}} = \big(RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{IBE,S_{i,j}}\}_{S_{i,j}\in CV_{R_{ID|_{k-1}}}}\big)$. Note that the master key parts of RSK_{HIBE}, RSK_{IBE}, and SK_{IBE,S_i} are η', $\alpha - \eta' - \beta_{ID|_{k-1}}$, and $f_{GL}(L_j)$ for some random η' respectively.

RHIBE.DeriveKey$(ID|_k, T, SK_{ID|_k}, UK_{T,R_{ID|_{k-1}}}, PP)$**:** This algorithm takes as input an identity $ID|_k$ with $k \geq 0$, time T, a private key $SK_{ID|_k} = (PV_{ID|_k}, \{ISK_{HIBE,S_{i,j}}\}_{S_{i,j} \in PV_{ID|_k}})$, an update key $UK_{T,R_{ID|_{k-1}}} = (RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{IBE,S_{i,j}}\}_{S_{i,j} \in CV_{R_{ID|_{k-1}}}})$ where $RDK_{ID|_{k-1},T} = (RSK'_{HIBE}, RSK'_{IBE})$, and the public parameters PP.

If $k = 0$, then $SK_{ID|_0} = MK$ and UK is empty. It proceeds as follows:

1. It selects a random exponent $\eta \in \mathbb{Z}_p$. It then obtains $RSK_{HIBE,ID|_0}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_0, \eta, PP_{HIBE})$ and **IBE.GenKey**$(T, MK - \eta, PP_{IBE})$ respectively.
2. It outputs a decryption key $DK_{ID|_0,T} = (RSK_{HIBE,ID|_0}, RSK_{IBE,T})$.

If $k \geq 1$, then it proceeds as follows:

1. If $ID|_k \notin R_{ID|_{k-1}}$, then it obtains $(S_{i,j}, S_{i',j'})$ by running **SD.Match**$(CV_{R_{ID|_{k-1}}}, PV_{ID|_k})$. Otherwise, it outputs $\perp$. Next, it retrieves $ISK_{HIBE,S_{i',j'}}$ from $SK_{ID|_k}$ and $SK_{IBE,S_{i,j}}$ from $UK_{T,R_{ID|_{k-1}}}$.
2. It sets $I = \{L_j, L_{j'}\}$ and calculates two Lagrange coefficients $\Delta_{L_j,I}(0)$ and $\Delta_{L_{j'},I}(0)$ by using the fact $L_j \neq L_{j'}$. It obtains TSK_{HIBE} and TSK_{IBE} by running **HIBE.ChangeIKey**$(ISK_{HIBE,S_{i',j'}}, (\times, \Delta_{L_{j'},I}(0)), PP_{HIBE})$ and **IBE.ChangeKey**$(SK_{IBE,S_{i,j}}, (\times, \Delta_{L_j,I}(0)), PP_{IBE})$ respectively.
3. It obtains RSK''_{HIBE} by running **HIBE.Delegate**$(ID|_k, RSK'_{HIBE}, PP_{HIBE})$ since RSK'_{HIBE} is for $ID|_{k-1}$. It selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ by running **HIBE.MergeKey**$(RSK''_{HIBE}, TSK_{HIBE}, \eta, PP_{HIBE})$ and **IBE.MergeKey**$(RSK'_{IBE}, TSK_{IBE}, -\eta, PP_{IBE})$ respectively.
4. Finally, it outputs a decryption key $DK_{ID|_k,T} = \left(RSK_{HIBE,ID|_k}, RSK_{IBE,T}\right)$.

Note that the master key parts of $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ are η' and $\alpha - \eta'$ for some random η' respectively.

RHIBE.RandDK$(DK_{ID|_k,T}, \beta, PP)$**:** It is the same as the randomization algorithm in Section 4.2.

RHIBE.Encrypt$(ID|_\ell, T, M, PP)$**:** It is the same as the encryption algorithm in Section 4.2.

RHIBE.Decrypt$(CT_{ID|_\ell,T}, DK_{ID'|_k,T'}, PP)$**:** It is the same as the decryption algorithm in Section 4.2.

RHIBE.Revoke$(ID|_k, T, RL_{ID|_{k-1}}, ST_{ID|_{k-1}})$**:** It is the same as the revocation algorithm in Section 4.2.

5.3 Correctness

To show the correctness of the above RHIBE-SD scheme, we only show that a decryption key $DK_{ID|_k,T}$ is correctly derived from a private key $SK_{ID|_k}$ and an update key $UK_{T,R_{ID|_{k-1}}}$ since other parts are almost the same as those of the RHIBE-CS scheme. Let $SK_{ID|_k} = (PV_{ID|_k}, \{ISK_{HIBE,S_{i,j}}\}_{S_{i,j} \in PV})$ and $UK_{T,R_{ID|_{k-1}}} = (RDK_{ID|_{k-1},T}, CV_R, \{SK_{IBE,S_{i,j}}\}_{S_{i,j} \in CV})$ where $RDK_{ID|_{k-1},T} = (RSK'_{HIBE}, RSK'_{IBE})$. If $ID|_k \notin R_{ID|_{k-1}}$, then the master key parts of $ISK_{HIBE,S_{i',j'}}$ and $SK_{IBE,S_{i,j}}$ are associated with $f_{GL}(L_{j'})$ and $f_{GL}(L_j)$ where $f_{GL}(x) = a_{GL}x + \beta_{ID|_{k-1}}$ and $GL = L_i \| d_i$. From the correctness of the SD scheme, we have that the master key parts of TSK_{HIBE} and TSK_{IBE} are $\tilde{\eta}$ and $f_{GL}(0) - \tilde{\eta} = \beta_{ID|_{k-1}} - \tilde{\eta}$ respectively for some $\tilde{\eta}$. Thus, the master key parts of RSK_{HIBE} and RSK_{IBE} that are returned by **HIBE.MergeKey** and **IBE.MergeKey** are associated with $\eta'' = (\eta') + \tilde{\eta} + \eta$ and $(\alpha - \eta' - \beta_{ID|_{k-1}}) + \beta_{ID|_{k-1}} - \tilde{\eta} - \eta = \alpha - \eta''$ respectively. Thus the **DeriveKey** algorithm is correct since we have α if we add two master key parts of the decryption key.

5.4 Security Analysis

Theorem 5.1. *The above RHIBE scheme from SD is SRL-IND-CPA secure if the PRF scheme is secure and the q-RW2 assumption holds.*

Proof. Let $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$ be the challenge hierarchical identity submitted by an adversary. To prove the selective IND-CPA security of our RHIBE scheme, we also classify the type of adversaries into $\ell+1$ types as the same as in Theorem 4.1. That is, the type of an adversary $\mathcal{A}$ is $\tau \in \{1, \ldots, \ell, \ell+1\}$ if $\mathcal{A}$ does not queries the private key of $ID^*|_j$ for all $j \in \{1, \ldots, \tau-1\}$, but $\mathcal{A}$ should query the private key of $ID^*|_\tau$.

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

Game $\mathbf{G}_0$. This game is the original security game. That is, a simulator $\mathcal{B}$ uses a pseudo-random function for each hierarchical identity $ID|_{k-1}$ when it generates private keys for child identities and update keys for time periods.

Game $\mathbf{G}_1$. This game $\mathbf{G}_1$ is similar to the game $\mathbf{G}_0$ except that $\mathcal{B}$ uses a truly random function for the hierarchical identity $ID^*|_{k-1}$ for all $k \in \{1, \ldots, \tau\}$ when it generates private keys and update keys. That is, $\mathcal{B}$ sets $f_{GL}(x)$ as a degree-one polynomial that passes two points $(0, \beta_{ID|_{k-1}})$ and $(\hat{x}, \hat{y})$ where $\hat{y}$ is a random value in $\mathbb{Z}_p$ instead of setting $f_{GL}(x) = a_{GL}x + \beta_{ID|_{k-1}}$ where $a_{GL} = \mathbf{PRF}(z_{ID^*|_{k-1}}, GL)$ for each $S_{i,j}$ in $\mathcal{BT}_{ID^*|_{k-1}}$.

Game $\mathbf{G}_2$. This final game $\mathbf{G}_2$ is almost the same with the previous game $\mathbf{G}_1$ except that a random session key is used to create the challenge ciphertext. Note that the advantage of $\mathcal{A}_\tau$ in this game is zero since the challenge ciphertext is not related to μ.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_i}$ be the advantage of $\mathcal{A}$ in a game $\mathbf{G}_i$. Let E_τ be the event that the adversary behaves like τ-type. From the Lemmas 5.2 and 5.3, we obtain the following equation

$$\left|\mathbf{Adv}_{\mathcal{A}}^{G_0} - \mathbf{Adv}_{\mathcal{A}}^{G_2}\right| \leq \sum_{\tau=1}^{\ell+1} \Pr[E_\tau] \cdot \left|\mathbf{Adv}_{\mathcal{A}_\tau}^{G_0} - \mathbf{Adv}_{\mathcal{A}_\tau}^{G_2}\right| \leq \ell \mathbf{Adv}_{\mathcal{B}}^{PRF}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{q\text{-}RW2}(\lambda).$$

This completes our proof. □

Lemma 5.2. *If the PRF scheme is secure, then no PPT τ-type adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

We omit the proof of this lemma since it is the same as that of Lemma 4.2.

Lemma 5.3. *If the q-RW2 assumption holds, then no PPT τ-type adversary can distinguish between $\mathbf{G}_1$ and $\mathbf{G}_2$ with a non-negligible advantage.*

Proof. Suppose there exists a τ-type adversary $\mathcal{A}_\tau$ that attacks the above RHIBE scheme with non-negligible advantage. A meta-simulator $\mathcal{B}$ that solves the q-RW2 assumption using $\mathcal{A}_\tau$ is given: a challenge tuple $D = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, g^x, g^y, g^z, g^{(xz)^2}, \{g^{b_i}, g^{xzb_i}, g^{xz/b_i}, g^{x^2zb_i}, g^{y/b_i^2}, g^{y^2/b_i^2}\}, \{g^{xzb_i/b_j}, g^{yb_i/b_j^2}, g^{xyzb_i/b_j^2}, g^{(xz)^2b_i/b_j}\}\big)$ and Z where $Z = Z_0 = e(g,g)^{xyz}$ or $Z = Z_1 \in_R \mathbb{G}_T$. Note that a challenge tuple $D_{DBDH} = ((p, \mathbb{G}, \mathbb{G}_T, e), g, g^x, g^y, g^z)$ for the DBDH assumption can be derived from the challenge tuple D of the q-RW2 assumption. Let $\mathcal{B}_{HIBE}$ be the simulator of Theorem 3.2 and $\mathcal{B}_{IBE}$ be the simulator of Theorem 3.3. Then $\mathcal{B}$ that interacts with $\mathcal{A}_\tau$ is described as follows:

Init: $\mathcal{A}_\tau$ initially submits a challenge hierarchical identity $ID^*|_\ell = (I_1^*, \ldots, I_\ell^*)$, challenge time T^*, and revocation list RL^* on the time T^*. $\mathcal{B}$ runs $\mathcal{B}_{HIBE}$ by giving D and Z, and it also runs $\mathcal{B}_{IBE}$ by giving D_{DBDH} and Z.

Setup: $\mathcal{B}$ submits $ID^*|_\ell$ to $\mathcal{B}_{HIBE}$ and receives PP_{HIBE}. It also submits T^* to $\mathcal{B}_{IBE}$ and receives PP_{IBE}. For each level $k \in \{1, \ldots, \tau\}$, it performs the following steps:

1. It initializes a function list $\mathbf{FL}_{ID^*|_{k-1}}$ as an empty one. It derives $R^*_{ID^*|_{k-1}}$ from RL^* where $R^*_{ID^*|_{k-1}}$ is the set of revoked identities in $\mathcal{BT}_{ID^*|_{k-1}}$ on the time T^*.

2. For each $ID_i \in R^*_{ID^*|_{k-1}}$, it assigns ID_i to a random (unique) leaf node $v_i \in \mathcal{BT}_{ID^*|_{k-1}}$. If $k = \tau$ but $\tau \neq \ell + 1$, then it also assigns $ID^*|_\tau$ to a random leaf node $v^* \in \mathcal{BT}_{ID^*|_{k-1}}$. Recall that if $\tau = \ell + 1$, then $\mathcal{A}_\tau$ does not request the private key of $ID^*|_\tau$.

3. Next, it obtains $CV_{ID^*|_{k-1}}$ by running $\mathbf{SD.Cover}(\mathcal{BT}_{ID^*|_{k-1}}, R^*_{ID^*|_{k-1}})$. It also obtains $PV_{ID^*|_\tau}$ by running $\mathbf{SD.Assign}(\mathcal{BT}_{ID^*|_{k-1}}, ID^*|_\tau)$ if $k = \tau$ but $\tau \neq \ell + 1$. If $k = \tau$ but $\tau \neq \ell + 1$, then it defines $\mathbf{FixedSubset}_{ID^*|_{k-1}} = PV_{ID^*|_\tau} \cup CV_{ID^*|_{k-1}}$. Otherwise, it defines $\mathbf{FixedSubset}_{ID^*|_{k-1}} = CV_{ID^*|_{k-1}}$.

4. Let $\mathcal{S}$ be the collection of all subsets $S_{i,j}$ in $\mathcal{BT}_{ID^*|_{k-1}}$. For each $S_{i,j} \in \mathcal{S}$, it first sets $GL = L_i \| d_j$ where $(L_i, L_j) = \mathbf{Label}(S_{i,j})$ and $d_j = \mathbf{Depth}(S_j)$ and then it updates the function list as follows:

   - If $S_{i,j} \in \mathbf{FixedSubset}_{ID^*|_{k-1}}$, then it selects random $\hat{y} \in \mathbb{Z}_p$ and saves $(GL, (\hat{x} = L_j, \hat{y}))$ to $\mathbf{FL}_{ID^*|_{k-1}}$.
   - Otherwise, it selects random $\hat{x}, \hat{y} \in \mathbb{Z}_p$ and saves $(GL, (\hat{x}, \hat{y}))$ to $\mathbf{FL}_{ID^*|_{k-1}}$ if $(GL, *) \notin \mathbf{FL}_{ID^*|_{k-1}}$.

   Note that it implicitly defines $f_{GL}(x)$ as a degree-one polynomial defined by two points $(0, \beta_{ID^*|_{k-1}})$ and $(\hat{x}, \hat{y})$ by using the Lagrange interpolation method.

It implicitly sets $\alpha = xy$ and gives the public parameters $PP = \left(PP_{HIBE}, PP_{IBE}, \Omega = e(g^x, g^y), N_{max}\right)$ to $\mathcal{A}_\tau$.

Phase 1: $\mathcal{A}_\tau$ adaptively requests a polynomial number of private key, update key, decryption key, and revocation queries. $\mathcal{B}$ handles these queries as follows:

If this is a private key query for $ID|_k = (I_1, \ldots, I_{k-1}, I_k)$ with $k \geq 1$, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID|_{k-1} \notin \mathbf{Prefix}(ID^*|_\ell)$: In this case, it normally generates a private key by using a normally generated $ST_{ID|_{k-1}}$ since it can obtains $DK_{ID|_{k-1},T}$ for any T.

  1. If the state $ST_{ID|_{k-1}}$ was not generated before, then it normally generates $ST_{ID|_{k-1}}$. Otherwise, it retrieves $ST_{ID|_{k-1}}$ that was previously generated.
  2. It obtains $SK_{ID|_k}$ by running $\mathbf{RHIBE.GenKey}(ID|_k, ST_{ID|_{k-1}}, PP)$ with a false master key $\beta_{ID|_{k-1}} \in \mathbb{Z}_p$. Finally, it gives $SK_{ID|_k}$ and $ST_{ID|_k}$ to $\mathcal{A}_\tau$ by normally generating $ST_{ID|_k}$. .

- **Case** $ID|_{k-1} \in \mathbf{Prefix}(ID^*|_\ell)$ with $k \leq \tau$ and $ID|_k \neq ID^*|_k$: In this case, it can generate a private key by applying the Lagrange interpolation method since it can derive ISK' of HIBE that contains α from the condition $ID|_k \neq ID^*|_k$.

  1. It queries an HIBE intermediate private key for $ID|_k$ to $\mathcal{B}_{HIBE}$ and receives ISK'_{HIBE}. Note that the master key part of ISK'_{HIBE} is α.
  2. It assigns $ID|_k$ to a unique leaf node $v \in \mathcal{BT}_{ID^*|_{k-1}}$ and obtains $PV_{ID|_k} = \{S_{i,j}\}$ by running $\mathbf{SD.Assign}(\mathcal{BT}_{ID^*|_{k-1}}, ID|_k)$.
  3. For each $S_{i,j} \in PV_{ID|_k}$, it retrieves $(GL = L_i \| d_j, (\hat{x}, \hat{y}))$ from $\mathbf{FL}_{ID^*|_{k-1}}$ and proceeds as follows:

– It sets $I = \{0, \hat{x}\}$ and calculates two Lagrange coefficients $\Delta_{0,I}(L_j)$ and $\Delta_{\hat{x},I}(L_j)$. Next, it obtains $ISK_{HIBE,S_{i,j}}$ by running **HIBE.ChangeIKey**$(ISK'_{HIBE}, \{(+, \beta_{ID^*|_{k-1}}), (\times, \Delta_{0,I}(L_j)), (+, \hat{y}\Delta_{\hat{x},I}(L_j))\}, PP_{HIBE})$ where $f_{GL}(x) = (\alpha + \beta_{ID^*|_{k-1}})\Delta_{0,I}(x) + (\hat{y})\Delta_{\hat{x},I}(x)$.

4. It sets $SK_{ID|_k} = \left(PV_{ID|_k}, \{ISK_{HIBE,S_{i,j}}\}_{S_{i,j} \in PV_{ID|_k}}\right)$ and gives $SK_{ID|_k}$ to $\mathcal{A}_\tau$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k \leq \tau$ and $ID|_k = ID^*|_k$: In this case, it can generate a private key by simply using the pre-fixed points $\{(\hat{x}, \hat{y})\}$ in the function list since $k = \tau$ but $\tau \neq \ell + 1$.

1. It assigns $ID^*|_\tau$ to the pre-fixed node v^* and obtains $PV_{ID^*|_\tau} = \{S_{i,j}\}$ by running **SD.Assign** $(\mathcal{BT}_{ID^*|_{\tau-1}}, ID^*|_\tau)$.
2. For each $S_{i,j} \in PV_{ID^*|_\tau}$, it retrieves $(GL = L_i \| d_j, (\hat{x}, \hat{y}))$ from $\mathbf{FL}_{ID^*|_{\tau-1}}$ and proceeds as follows:
   – It obtains $ISK_{HIBE,S_{i,j}}$ by running **HIBE.GenIKey**$(ID^*|_\tau, \hat{y}, PP_{HIBE})$.
3. It sets $SK_{ID^*|_\tau} = \left(PV_{ID^*|_\tau}, \{ISK_{HIBE,S_{i,j}}\}_{S_{i,j} \in PV_{ID^*|_\tau}}\right)$ and gives $SK_{ID^*|_\tau}$ to $\mathcal{A}_\tau$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k > \tau$: In this case, it is the same as the case $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$. That is, it follows the normal key generation algorithm. We omit the description of this case.

If this is an update key query for $ID|_{k-1} = (I_1, \ldots, I_{k-1})$ with $k \geq 1$ and T, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$: In this case, it can generate $UK_{T,R_{ID|_{k-1}}}$ by following the normal algorithm since it can obtain $DK_{ID|_{k-1},T}$ from the condition $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$.

1. If the state $ST_{ID|_{k-1}}$ was generated before, then it retrieves $ST_{ID|_{k-1}}$. Otherwise, it normally generates $ST_{ID|_{k-1}}$ by himself. Next, it requests a decryption key query for $ID|_{k-1}$ and T to himself and receives $DK_{ID|_{k-1},T}$ since $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$.
2. It obtains $UK_{T,R_{ID|_{k-1}}}$ by running **RHIBE.UpdateKey**$(T, RL_{ID|_{k-1}}, DK_{ID_{k-1},T}, ST_{ID|_{k-1}}, PP)$ with a false master key $\beta_{ID|_{k-1}} \in \mathbb{Z}_p$ and gives $UK_{T,R_{ID|_{k-1}}}$ to $\mathcal{A}_\tau$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k \leq \tau$ and $T \neq T^*$: In this case, it can generate an update key by applying the Lagrange interpolation method since it can derive SK' of IBE that contains α from the condition $T \neq T^*$.

1. It selects random exponents $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_{k-1}}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_{k-1}, \eta, PP_{HIBE})$ and **IBE.GenKey**$(T, -\eta - \beta_{ID|_{k-1}}, PP_{IBE})$ respectively. It sets $RDK_{ID|_{k-1},T} = (RSK_{HIBE,ID|_{k-1}}, RSK_{IBE,T})$.
2. It queries an IBE private key for T to $\mathcal{B}_{IBE}$ and receives $SK'_{IBE,T}$ since $T \neq T^*$.
3. It derives the set $R_{ID|_{k-1}}$ of revoked identities at time T from $RL_{ID|_{k-1}}$ and obtains $CV_{R_{ID|_{k-1}}} = \{S_{i,j}\}$ by running **SD.Cover**$(\mathcal{BT}_{ID^*|_{k-1}}, R_{ID|_{k-1}})$.
4. For each $S_{i,j} \in CV_{R_{ID|_{k-1}}}$, it retrieves $(GL = L_i \| d_j, (\hat{x}, \hat{y}))$ from $\mathbf{FL}_{ID^*|_{k-1}}$ and proceeds as follows:
   – It sets $I = \{0, \hat{x}\}$ and calculates two Lagrange coefficients $\Delta_{0,I}(L_j)$ and $\Delta_{\hat{x},I}(L_j)$. Next, it obtains $SK_{IBE,S_{i,j}}$ by running **IBE.ChangeKey**$(SK'_{IBE,T}, \{(+, \beta_{ID^*|_{\tau-1}}), (\times, \Delta_{0,I}(L_j)), (+, \hat{y}\Delta_{\hat{x},I}(L_j))\}, PP_{IBE})$.
5. It sets $UK_{T,R_{ID|_{k-1}}} = \left(RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{HIBE,S_i}\}_{S_i \in CV_{R_{ID|_{k-1}}}}\right)$ and gives $UK_{T,R_{ID|_{k-1}}}$ to $\mathcal{A}_\tau$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k \leq \tau$ and $T = T^*$: In this case, it can generate an update key by using the fixed points $\{(\hat{x}, \hat{y})\}$ in the function list since $R_{ID|_{k-1}} = R^*_{ID^*|_{k-1}}$ on the challenge time T^*.

1. It selects random exponents $\eta \in \mathbb{Z}_p$. Next it obtains $RSK_{HIBE,ID|_{k-1}}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_{k-1}, \eta, PP_{HIBE})$ and **IBE.GenKey**$(T, -\eta - \beta_{ID|_{k-1}}, PP_{IBE})$ respectively. It sets $RDK_{ID|_{k-1},T} = (RSK_{HIBE,ID|_{k-1}}, RSK_{IBE,T})$.
2. It derives the set $R_{ID|_{k-1}}$ of revoked identities at time T from $RL_{ID|_{k-1}}$ and obtains $CV_{R_{ID|_{k-1}}} = \{S_{i,j}\}$ by running **SD.Cover**$(\mathcal{BT}_{ID|_{k-1}}, R_{ID|_{k-1}})$.
3. For each $S_{i,j} \in CV_{R_{ID|_{k-1}}}$, it retrieves $(GL = L_i \| d_j, (\hat{x}, \hat{y}))$ from $\mathbf{FL}_{ID^*|_{\tau-1}}$ and proceeds as follows:
   - It obtains $SK_{IBE,S_{i,j}}$ by running **IBE.GenKey**$(T, \hat{y}, PP_{IBE})$.
4. It sets $UK_{T,R_{ID|_{k-1}}} = \left(RDK_{ID|_{k-1},T}, CV_{R_{ID|_{k-1}}}, \{SK_{IBE,S_{i,j}}\}_{S_{i,j} \in CV_{R_{ID|_{k-1}}}}\right)$ and gives $UK_{T,R_{ID|_{k-1}}}$ to $\mathcal{A}_\tau$.

- **Case** $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k > \tau$: In this case, it is almost the same as the first case $ID|_{k-1} \notin$ **Prefix**$(ID^*|_\ell)$ except that it uses a decryption key by using the condition $T \neq T^*$. We omit the description of this case.

If this is a decryption key query for $ID|_k = (I_1, \ldots, I_k)$ with $k \geq 1$ and T, then $\mathcal{B}$ proceeds as follows:

- **Case** $ID|_k \notin$ **Prefix**$(ID^*|_\ell)$: In this case, it can use $\mathcal{B}_{HIBE}$ to generate g^α since $ID|_k \notin$ **Prefix**$(ID^*|_\ell)$.
  1. It first queries an HIBE private key for $ID|_k$ to $\mathcal{B}_{HIBE}$ and receives $SK'_{HIBE,ID|_k}$.
  2. It selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ by running **HIBE.ChangeKey**$(SK'_{HIBE,ID|_k}, (+, \eta), PP_{HIBE})$ and **IBE.GenKey**$(T, -\eta, PP_{IBE})$ respectively.
  3. It gives $DK_{ID|_k,T} = \left(RSK_{HIBE,ID|_k}, RSK_{IBE,T}\right)$ to $\mathcal{A}_\tau$. Note that the master key part of $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ are $\alpha + \eta$ and $-\eta$ respectively.

- **Case** $ID|_k \in$ **Prefix**$(ID^*|_\ell)$ and $T \neq T^*$: In this case, it can use $\mathcal{B}_{IBE}$ to generates g^α since $T \neq T^*$.
  1. It first queries an IBE private key for T to $\mathcal{B}_{IBE}$ and receives $SK'_{IBE,T}$.
  2. It selects a random exponent $\eta \in \mathbb{Z}_p$. Next, it obtains $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ by running **HIBE.GenKey**$(ID|_k, \eta, PP_{HIBE})$ and **IBE.ChangeKey**$(SK'_{IBE,T}, (+, -\eta), PP_{IBE})$ respectively.
  3. It gives $DK_{ID|_k,T} = \left(RSK_{HIBE,ID|_k}, RSK_{IBE,T}\right)$ to $\mathcal{A}_\tau$. Note that the master key part of $RSK_{HIBE,ID|_k}$ and $RSK_{IBE,T}$ are η and $\alpha - \eta$ respectively.

If this is a revocation query for $ID|_k$ and T, then $\mathcal{B}$ obtains an updated $RL_{ID|_{k-1}}$ by running **RHIBE.Revoke** $(ID|_k, T, RL_{ID|_{k-1}}, ST_{ID|_{k-1}})$. Note that $\mathcal{A}_\tau$ cannot query to revoke $ID|_k$ on time T if he already requested an update key query for $ID|_k$ on time T.

Challenge: $\mathcal{A}_\tau$ submits two challenge messages M_0^*, M_1^*. $\mathcal{B}$ flips a coin $\mu \in \{0,1\}$. Next, it requests the challenge ciphertext to $\mathcal{B}_{HIBE}$ and receives $CH_{HIBE,ID^*|_k}$ and EK_{HIBE}. It also requests the challenge ciphertext to $\mathcal{B}_{IBE}$ and receives CH_{IBE,T^*} and EK_{IBE}. It sets the challenge ciphertext $CT_{ID^*|_\ell,T^*} = \left(CH_{HIBE,ID^*|_\ell}, CH_{IBT,T^*}, Z \cdot M_\mu^*\right)$ and gives it to $\mathcal{A}_\tau$.

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

Guess: $\mathcal{A}_\tau$ outputs a guess $\mu' \in \{0,1\}$. If $\mu = \mu'$, then $\mathcal{B}$ outputs 0. Otherwise, $\mathcal{B}$ outputs 1.

To finish the proof, we should show that the public parameters, private keys, update keys, decryption keys, and the challenge ciphertext are all generated correctly. We omit the checking of the public parameters, decryption keys, and the challenge ciphertext since it is almost similar to that in Theorem 4.1.

Now, we should show that the master key parts of private keys and update keys are consistently generated in the cases of $ID|_{k-1} \in$ **Prefix**$(ID^*|_\ell)$ with $k \leq \tau$. As mentioned in the construction, a degree-one

polynomial $f_{GL}(x)$ is associated to the master key parts of ISK_{HIBE} in a private key and SK_{IBE} in an update key. If $ID|_{k-1} \in \mathbf{Prefix}(ID^*|_\ell)$, the simulator cannot create RDK that has α as a master key part. Thus, the simulator should set $f_{GL}(x)$ to have α for consistency. That is, $f_{GL}(x)$ is defined as a degree-one polynomial that passes two points $(0, \alpha + \beta_{ID|_{k-1}})$ and $(\hat{x}, \hat{y})$. A private key for $ID|_k \neq ID^*|_k$ and an update key for $T \neq T^*$ are consistently generated by the simulator since it can use the Lagrange interpolation method. To generate a private key for $ID|_\tau = ID^*|_\tau$ and and update key for $T = T^*$, the simulator simply use the fixed point $(\hat{x}, \hat{y})$ in the setup phase. Recall that the private key and the update key share the same fixed point since the private key of $ID^*|_\tau$ should be revoked on the time T^* by the restriction of the security model. This completes our proof. □

6 Conclusion

In this paper, we showed that RHIBE schemes with shorter private keys and update keys and small public parameters can be built by following the modular approach. To build our RHIBE schemes, we first proposed an HIBE scheme derived from the RW-HIBE scheme that supports the generation of short intermediate private keys. Next, we proposed efficient RHIBE schemes by combining our HIBE scheme, the BB-IBE scheme, and the CS (or SD) scheme in a modular way. We also proved the security of our RHIBE schemes in the selective model (or the selective revocation list model).

We mentioned that another RHIBE scheme also can be built by using another HIBE scheme (the BB-HIBE scheme [2] or the BBG-HIBE scheme [4]) instead of using our HIBE scheme. If the BBG-HIBE scheme is used, then the resulting RHIBE scheme cannot have shorter private keys since the BBG-HIBE scheme cannot provide shorter intermediate private keys. Thus, it is an interesting problem to build an RHIBE scheme with shorter private keys and update keys and constant size ciphertexts. Another interesting problem is to prove the full security against insiders of our RHIBE schemes.

References

[1] 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*, pages 417–426. ACM, 2008.

[2] 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.

[3] Dan Boneh and Xavier Boyen. Efficient selective identity-based encryption without random oracles. *J. Cryptology*, 24(4):659–693, 2011.

[4] 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.

[5] 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.

[6] Xavier Boyen. General *Ad Hoc* encryption from exponent inversion IBE. In Moni Naor, editor, *Advances in Cryptology - EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 394–411. Springer, 2007.

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

[8] 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.

[9] 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.

[10] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theory*, 22(6):644–654, 1976.

[11] Keita Emura, Jae Hong Seo, and Taek-Young Youn. Semi-generic transformation of revocable hierarchical identity-based encryption and its DBDH instantiation. *IEICE Transactions*, 99-A(1):83–91, 2016.

[12] Craig Gentry and Shai Halevi. Hierarchical identity based encryption with polynomially many levels. In Omer Reingold, editor, *Theory of Cryptography - TCC 2009*, volume 5444 of *Lecture Notes in Computer Science*, pages 437–456. Springer, 2009.

[13] 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.

[14] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[15] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.

[16] Dani Halevy and Adi Shamir. The LSD broadcast encryption scheme. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 47–60. Springer, 2002.

[17] 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.

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

[19] Kwangsu Lee, Seung Geol Choi, Dong Hoon Lee, Jong Hwan Park, and Moti Yung. Self-updatable encryption: Time constrained access control with hidden attributes and better efficiency. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 235–254. Springer, 2013.

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

[21] Kwangsu Lee, Jong Hwan Park, and Dong Hoon Lee. Anonymous HIBE with short ciphertexts: full security in prime order groups. *Designs Codes Cryptogr.*, 74(2):395–425, 2015.

[22] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology - EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 318–335. Springer, 2012.

[23] 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.

[24] Allison B. Lewko and Brent Waters. Unbounded HIBE and attribute-based encryption. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 547–567. Springer, 2011.

[25] 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.

[26] 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.

[27] 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.

[28] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM Conference on Computer and Communications Security - CCS 2013*, pages 463–474. ACM, 2013.

[29] 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, 2016.

[30] 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.

[31] 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.

[32] Jae Hong Seo and Keita Emura. Adaptive-id secure revocable hierarchical identity-based encryption. In Keisuke Tanaka and Yuji Suga, editors, *Advances in Information and Computer Security - IWSEC 2015*, volume 9241 of *Lecture Notes in Computer Science*, pages 21–38. Springer, 2015.

[33] 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.

[34] Jae Hong Seo, Tetsutaro Kobayashi, Miyako Ohkubo, and Koutarou Suzuki. Anonymous hierarchical identity-based encryption with constant size ciphertexts. In Stanislaw Jarecki and Gene Tsudik, editors, *Public-Key Cryptography - PKC 2009*, volume 5443 of *Lecture Notes in Computer Science*, pages 215–234. Springer, 2009.

[35] 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.

[36] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008*, volume 5126 of *Lecture Notes in Computer Science*, pages 560–578. Springer, 2008.

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

[38] 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.