Efficient Revocable Identity-Based Encryption via Subset Difference Methods

Kwangsu Lee[*] Dong Hoon Lee[†] Jong Hwan Park[‡]

Abstract

Providing an efficient revocation mechanism for identity-based encryption (IBE) is very important since a user's credential (or private key) can be expired or revealed. Revocable IBE (RIBE) is an extension of IBE that provides an efficient revocation mechanism. Previous RIBE schemes essentially use the complete subtree (CS) scheme for key revocation. In this paper, we present a new technique for RIBE that uses the efficient subset difference (SD) scheme or the layered subset difference (LSD) scheme instead of using the CS scheme to improve the size of update keys. Following our new technique, we first propose an efficient RIBE scheme in prime-order bilinear groups by combining the IBE scheme of Boneh and Boyen and the SD scheme and prove its selective security under the standard assumption. Our RIBE scheme is the first RIBE scheme in bilinear groups that has $O(r)$ number of group elements in update keys. Next, we also propose another RIBE scheme in composite-order bilinear groups and prove its full security under static assumptions. Our RIBE schemes also can be integrated with the LSD scheme to reduce the size of private keys.

Keywords: Identity-based encryption, Revocable identity-based encryption, Key revocation, Subset difference method, Bilinear maps.

[*]Korea University, Korea. Email: guspin@korea.ac.kr.
[†]Korea University, Korea. Email: donghlee@korea.ac.kr.
[‡]Sangmyung University, Korea. Email: jhpark@smu.ac.kr.

1 Introduction

Identity-based encryption (IBE) is a new paradigm of public-key encryption (PKE) that uses the identity string of a user for the public key of the user [5, 6, 38, 40]. IBE and its extensions like hierarchical IBE (HIBE) [5, 12, 16], attribute-based encryption (ABE) [3, 14, 35], predicate encryption (PE) [8, 17, 21], and functional encryption (FE) [7, 13] opened new applications of encryption systems such as the delegation of decryption capability, access control in encrypted data, searches on encrypted data, and functional evaluation on encrypted data. If an IBE scheme is used in real-world applications, an efficient *revocation mechanism* for IBE that can handle dynamic credentials of users is needed since a user's credential can be revealed or expired. Revocable IBE (RIBE) is an extension of IBE that can handle the dynamic credentials of users by providing an efficient revocation mechanism. An ideal revocation method in IBE is that a sender just creates a ciphertext without worrying about the revocation of a receiver and only the receiver needs to check the revocation of his credential to decrypt the ciphertext.

Boneh and Franklin [6] proposed the first RIBE scheme by representing an identity as $ID\|T$ where ID is the real identity and T is a current time, but it is inefficient and not scalable because of the requirement of secure channels between the center and all users. A scalable RIBE scheme was introduced by Boldyreva, Goyal, and Kumar [4]. They defined the first formal definition of RIBE and proposed a selectively secure RIBE scheme by combining the fuzzy IBE (FIBE) scheme of Sahai and Waters [35] and the complete subtree (CS) scheme of Naor et al. [28]. After that fully secure RIBE schemes were proposed [25, 37]. Recently, the design technique of RIBE was successfully applied to achieve a revocable HIBE (RHIBE) scheme, a revocable-storage ABE (RS-ABE) scheme, and a revocable-storage PE (RS-PE) scheme [18, 34, 36].

Although efficient RIBE schemes and their extended schemes were proposed, the main design principle of these constructions essentially follows that of Boldyreva et al. [4] that uses the CS scheme of Naor et al. [28] for key revocation. The CS scheme is one instance of the general subset cover framework of Naor et al. and there are other efficient subset cover schemes like the subset difference (SD) scheme [28] and the layered subset difference (LSD) scheme [15]. Therefore, we ask the following natural question in this paper.

> *"Can we build an efficient RIBE scheme by using the SD scheme (or the LSD scheme) instead of using the CS scheme?"*

If it is possible, then the size of update keys can be reduced from $O(r\log(N_{max}/r))$ to $O(r)$ group elements by slightly increasing the size of private keys where N_{max} is the maximum number of users and r is the number of revoked users.

1.1 Our Results

In this paper, we give the affirmative answer for the above question by presenting a new technique that combines an IBE scheme and the SD scheme. The following is our results:

New Technique for RIBE. We first present a new technique for RIBE that combines an IBE scheme and the SD scheme instead of using the CS scheme. The CS scheme was easily integrated with an IBE scheme since an assigned key for a subset in the CS scheme is independent of each other [28]. However, it is unclear how to integrate the SD scheme with an IBE scheme since an assigned key for a subset in the SD scheme is dependent on another [28]. We use a random polynomial of degree one to solve the complex key assignment problem in the SD scheme, and integrate the SD scheme with an IBE scheme by using the observation that the SD scheme is related with a single member revocation scheme which can be implemented by a random polynomial of degree one. However, this idea does not directly lead to a secure scheme because of collusion attacks. To provides the security against collusion attacks, we personalize the private key components with an identity ID and constrain the usage of update key components by a time T.

RIBE with Selective Security. Following our new technique for RIBE, we construct an RIBE scheme in prime-order bilinear groups by combining the IBE scheme of Boneh and Boyen [5] and the SD scheme of Naor et al. [28], and then we prove its security in the selective revocation list model under the standard assumption. In our RIBE scheme, the number of group elements in the public parameters, a private key, an update key, and a ciphertext is $O(1)$, $O(\log^2 N_{max})$, $O(r)$, and $O(1)$ respectively where N_{max} is the maximum number of users and r is the number of revoked users. Compared with the previous RIBE schemes [4,25,37] that have $O(r\log(N_{max}/r))$ group elements in an update key, our RIBE scheme just has $O(r)$ group elements in an update key. If the LSD scheme of Halevy and Shamir [15] is used instead of the SD scheme, then the number of group elements in a private key is reduced from $O(\log^2 N_{max})$ to $O(\log^{1.5} N_{max})$.

RIBE with Full Security. Next, we propose another RIBE scheme in composite-order bilinear groups by combining the IBE scheme of Lewko and Waters [24] and the SD scheme, and the we prove its security in the full model under static assumptions. To prove the security in the full model, we use the dual system encryption technique of Waters [24,41]. However, the original dual system encryption technique that was used to prove the security of IBE and HIBE is not directly applicable to an RIBE scheme since the adversary of RIBE can request a private key for a challenge identity ID^* and an update key for a challenge time T^* that were not allowed in the security model of IBE. Additionally, the complex key assignment of the SD scheme introduces another difficulty in the proof of using the dual system encryption technique. To solve these problems, we carefully redesign the semi-functional types of each key and hybrid games for the dual system encryption.

1.2 Our Techniques

The CS scheme can be easily integrated with an IBE scheme to construct an RIBE scheme since it assigns a random independent key for each subset in CS [4]. In contrast, the SD scheme cannot be easily integrated with an IBE scheme to construct an RIBE scheme since it assigns a dependent key for each subset in SD by using a pseudo-random generator [28]. To overcome the complex and dependent key assignment of the SD scheme, we use the observation that a subset $S_{i,j}$ in SD can be interpreted as *single member revocation*. In the SD scheme, a subset $S_{i,j}$ that is associated with two nodes v_i and v_j of a binary tree is defined as the set of leaf nodes that belong to $T_i \setminus T_j$ where T_i is a subtree rooted at v_i and T_j is a subtree rooted at v_j. If we define a group GL as the set of subsets $S_{i,j}$ such that v_i is the same and the depth d_j of v_j is also the same, then the subset $S_{i,j}$ can be interpreted as this subset $S_{i,j}$ is almost same as GL except that one member node v_j is excluded (or revoked). This observation was implicitly made by Lee et al. [20] and they used this observation to construct a public-key trace and revoke scheme by combining the SD scheme and a single-revocation encryption scheme in bilinear groups. To use this observation for RIBE, we use the polynomial-based revocation scheme of Naor and Pinkas [30]. That is, a revocation scheme that uses a random polynomial of degree one can be used to revoke a single user.

In the SD scheme, the collection S is defined as the set of subsets $S_{i,j}$ where v_i and v_j is a node in a tree and v_j is a descendant of v_i. As mentioned before, the subsets $S_{i,j}$ can be categorized as groups and a one group GL is defined as a set of subsets $S_{i,j}$ such that v_i is the same and the depth d_j of v_j is the same. To use the polynomial-based revocation scheme, we assign a random polynomial $f_{GL}(x) = a_{GL}x + \alpha$ once to each group where a_{GL} is a random value and α is a fixed value for all groups. In a group GL, each member L_i that is associated with a node v_j has a share $g^{f_{GL}(L_j)}$ where L_i is a identifier of the node v_j. If one member $L_{j'}$ is revoked, then his share $g^{f_{GL}(L_{j'})}$ is revealed to all members, then any member in the group GL except the revoked member can reconstruct the secret g^{α} by using the Lagrange interpolation method since two points of a degree one polynomial are enough for reconstruction. However, this simple method is insecure against collusion attacks since any two members can collude to reconstruct the secret α.

To provide the security against collusion attacks, the share of a member is personalized by using his identity ID and the share of the revoked member is constrained by using a revoked time T. That is the personalized private key for a member L_j is defined as $g^{f_{GL}(L_j)} H(ID)^{r_1}, g^{r_1}$ where H is a hash function and the time-constrained update key for a revoked member $L_{j'}$ is defined as $g^{f_{GL}(L'_j)} H(T)^{r_2}, g^{-r_2}$. Thus a non-revoked member in the group can derive a decryption key as $g^{\alpha} H(ID)^{r_1} H(T)^{r_2}, g^{-r_1}, g^{-r_2}$ from his personalized private key and the time-constrained update key. Note that if non-revoked two members collude, then they only can derive $g^{\alpha} H(ID)^{r_1} H(ID')^{r_2}, g^{-r_1}, g^{-r_2}$ that are not useful to decrypt a ciphertext. In the RIBE scheme, a private key for a user consists of many subsets and an update key for a time also consists of many subsets.

1.3 Related Work

Certificate Revocation in PKE. In PKE that uses public-key infrastructure (PKI), CRL and OCSP are the traditional methods to revoke certificates of users. However, these methods are inefficient in terms of transmission costs and computation costs since CRL includes all serial numbers of revoked certificates and OCSP requires the generation of digital signature for each queries. Furthermore, they also require for each client who uses a certificate to implement a path validation module to check the validity of a digital signature in CRL or OCSP. A better solution named certificate revocation system (CRS) was proposed by Micali [27] and it uses a hash-chain to check the validity of the certificate. This method was improved by Naor and Nissim [29] and Aiello et al. [1]. Although CRS improves the previous CRL and OCSP, these methods still require a sender to check the validity of a certificate through a heavy infrastructure and this problem is the serious point of these methods. Gentry [11] solved this problem by introducing certificate-based encryption (CBE) and proposed an efficient CBE scheme in bilinear groups.

Revocation in IBE. As mentioned, an ideal revocation method for IBE is that a sender can create a ciphertext as the same as that of IBE without worrying about the revocation of a receiver and only the receiver checks the revocation of his key to decrypt the ciphertext. Boneh and Franklin [6] proposed the first IBE scheme that support the revocation capability, but their scheme is inefficient and not scalable since each user should be connected to the center through a secure channel to receive an updated private key. A scalable and RIBE scheme was proposed by Boldyreva et al. [4]. They constructed an RIBE scheme by combining the FIBE scheme of Sahai and Waters [35] and the CS scheme and proved it selective security. After that, fully secure RIBE schemes were proposed by Libert and Vergnaud [25] and Seo and Emura [37], and Seo and Emura refined the security model of RIBE by considering the decryption key exposure attacks. Recently, Park et al. [33] proposed an RIBE scheme with shorter private key and update key by using multilinear maps, but the size of the public parameters is dependent to the number of users. The design technique of RIBE also can be applicable to the extensions of IBE, like HIBE, ABE, and PE. Boldyreva et al. [4] already proposed a revocable ABE (R-ABE) scheme. Seo and Emura [36] proposed an RHIBE scheme by using the HIBE scheme of Boneh and Boyen. For cloud storage, Sahai et al. [34] proposed RS-ABE schemes that provide the key revocation and ciphertext update functionalities, and Lee et al. [18] proposed improved RS-ABE and RS-PE schemes by introducing self-updatable encryption.

Revocation Encryption. Revocation encryption (RE) is a special type of broadcast encryption (BE) [10] such that a sender creates a ciphertext by specifying a set of revoked users R instead of a set of receivers S and a receiver can decrypt a ciphertext if he is not included in the set of revoked users [23,28,30]. However, there is a crucial difference between the security model of RE and that of RIBE. In RE, the collusion of non-revoked users is not allowed since an adversary cannot request private keys for non-revoked users [23], but the collusion of non-revoked users is allowed in RIBE since an adversary can request private keys for non-revoked users except the challenge user ID^* [4]. Although RE alone does not directly solve the revocation problem of IBE, RE can be combined with IBE or its extensions to directly revoke a set of revoked users by specifying a receiver and a revoked set R in a ciphertext [2, 19, 31, 39]. This approach requires a sender to take care of the revocation of users.

2 Preliminaries

In this section, we introduce the subset difference method and define the syntax and the security model of revocable IBE.

2.1 Full Binary Tree

A full binary tree $\mathcal{BT}$ is a tree data structure where each node except the leaf nodes has two child nodes. Let N_{max} be the number of leaf nodes in $\mathcal{BT}$. The number of all nodes in $\mathcal{BT}$ is $2N_{max}-1$ and for any $1 \leq i \leq 2N_{max}-1$ we denote by v_i a node in $\mathcal{BT}$. The depth d_i of a node v_i is the length of the path from the root node to the node. The root node is at depth zero. The depth of $\mathcal{BT}$ is the length of the path from the root node to a leaf node. A level of $\mathcal{BT}$ is a set of all nodes at given depth. For any node $v_i \in \mathcal{BT}$, T_i is defined as a subtree that is rooted at v_i. For any two nodes $v_i, v_j \in \mathcal{BT}$ such that v_j is a descendant of v_i, $T_{i,j}$ is defined as a subtree $T_i - T_j$, that is, all nodes that are descendants of v_i but not v_j. For any node $v_i \in \mathcal{BT}$, S_i is defined as the set of leaf nodes in T_i. Similarly, $S_{i,j}$ is defined as the set of leaf nodes in $T_{i,j}$, that is, $S_{i,j} = S_i \setminus S_j$.

For any node $v_i \in \mathcal{BT}$, L_i is defined as an identifier that is a fixed and unique string. The identifier of each node in the tree is assigned as follows: Each edge in the tree is assigned with 0 or 1 depending on whether the edge is connected to its left or right child node. The identifier L_i of a node v_i is defined as the bitstring obtained by reading all the labels of edges in the path from the root node to the node v_i. We define $L(v_i)$ be a mapping from a node v_i to an identifier L_i. We also define $L(T_i)$ be a mapping from a subtree T_i to the identifier L_i of the node v_i and $L(T_{i,j})$ be a mapping from a subtree $T_{i,j}$ to a tuple (L_i, L_j) of identifiers. Similarly, we can define $L(S_i) = L(T_i)$ and $L(S_{i,j}) = L(T_{i,j})$.

For a full binary tree $\mathcal{BT}$ and a subset R of leaf nodes, $ST(\mathcal{BT}, R)$ is defined as the Steiner Tree induced by the set R and the root node, that is, the minimal subtree of $\mathcal{BT}$ that connects all the leaf nodes in R and the root node. we simply denote $ST(\mathcal{BT}, R)$ by $ST(R)$.

2.2 Subset Difference Method

The subset difference (SD) method is a special instance of the subset cover framework of Naor, Naor, and Lotspiech [28] that is a general methodology for revocation schemes. The well-known complete subtree (CS) scheme is also one instance of the subset cover framework. The original subset cover framework consists of a subset assignment part and a key assignment part. In this paper, we define the subset cover framework by using the subset assignment part only. The formal definition is given as follows:

Definition 2.1 (Subset Cover)**.** *A subset cover scheme for the set $\mathcal{N} = \{1, \ldots, N_{max}\}$ of users consists of four PPT algorithms **Setup**, **Assign**, **Cover**, and **Match**, which are defined as follows:*

***Setup**(N_{max}). The setup algorithm takes as input the maximum number N_{max} of users and outputs a collection $\mathcal{S}$ of subsets $S_1, \dots, S_w$ where $S_i \subseteq \mathcal{N}$.*

***Assign**($\mathcal{S}, u$). The assigning algorithm takes as input the collection $\mathcal{S}$ and a user $u \in \mathcal{N}$, and outputs a private set $PV_u = \{S_{j_1}, \dots, S_{j_n}\}$ that is associated with the user u.*

***Cover**($\mathcal{S}, R$). The covering algorithm takes as the collection $\mathcal{S}$ and a revoked set $R \subset \mathcal{N}$ of users, and outputs a covering set $CV_R = \{S_{i_1} \dots, S_{i_m}\}$ that is a partition of the non-revoked users $\mathcal{N} \setminus R$ into disjoint subsets $S_{i_1}, \dots, S_{i_m}$ such that $\mathcal{S} \setminus R = \bigcup_{k=1}^{m} S_{i_k}$.*

***Match**(CV_R, PV_u). The matching algorithm takes as input a covering set $CV_R = \{S_{i_1}, \dots, S_{i_m}\}$ and a private set $PV_u = \{S_{j_1}, \dots, S_{j_n}\}$ of a user u. It outputs $(S_{i_k}, S_{j_{k'}})$ such that $S_{i_k} \in CV_R$, $u \in S_{i_k}$, and $S_{j_{k'}} \in PV_u$, or it outputs $\perp$.*

*The correctness of subset cover is defined as follows: For all $\mathcal{S}$ generated by **Setup**, all PV_u generated by **Assign**, and any R, it is required that:*

- *If $u \notin R$, then $\textbf{Match}(\textbf{Cover}(\mathcal{S}, R), PV_u) = (S_{i_k}, S_{j_{k'}})$ such that $S_{i_k} \in CV_R$ and $S_{j_{k'}} \in PV_u$.*
- *If $u \in R$, then $\textbf{Match}(\textbf{Cover}(\mathcal{S}, R), PV_u) = \perp$.*

Note that the exact conditions of the subsets outputted by the matching algorithm is defined by the specific instance of the SC scheme.

As mentioned, the SD scheme is one instance of the SC scheme and it was proposed by Naor et al. [28] as an improvement on the CS scheme. The SD scheme is described as follows:

SD.Setup(N_{max}): This algorithm takes as input the maximum number N_{max} of users. 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($\mathcal{BT}, u$): This algorithm takes as input the tree $\mathcal{BT}$ and a user $u \in \mathcal{N}$. Let v_u be the leaf node of $\mathcal{BT}$ that is assigned to the user u. Let $(v_{k_0}, v_{k_1}, \dots, v_{k_n})$ be the path from the root node v_{k_0} to the leaf node $v_{k_n} = v_u$. It first sets a private set PV_u as an empty one. For all $i, j \in \{k_0, k_1, \dots, k_n\}$ 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_u. It outputs the private set $PV_u = \{S_{i,j}\}$.

SD.Cover($\mathcal{BT}, R$): This algorithm takes as input the tree $\mathcal{BT}$ and a revoked set R of users. 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_u)**:** This algorithm takes input as a covering set $CV_R = \{S_{i,j}\}$ and a private set $PV_u = \{S'_{i,j}\}$. It finds two subsets $S_{i,j}$ and $S'_{i',j'}$ such that $S_{i,j} \in CV_R$, $S'_{i',j'} \in PV_u$, and $(i = i') \wedge (d_j = d_{j'}) \wedge (j \neq j')$ 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 $\perp$.

Lemma 2.2 ([28])**.** *Let N_{max} be the number of leaf nodes in a full binary tree and r be the size of a revoked set. In the SD scheme, the size of a private set is $O(\log^2 N_{max})$ and the size of a covering set is at most $2r-1$.*

Remark 2.3. *The covering algorithm of the SD scheme is only defined for $r \geq 1$. One simple way to handle the case $r = 0$ is to use a dummy user that is always revoked. In the SD scheme, the size of the covering set is at most $2r-1$, but it is rough worst-case analysis and the size is always smaller than that of the CS scheme since a subset in the CS scheme is defined by a subset in the SD scheme [28]. The better analysis of this covering set size is given by Martin et al. [26].*

The layered subset difference (LSD) scheme was proposed by Halevy and Shamir [15] to reduce the size of a private set in the SD scheme. The SD scheme in a cryptosystem generally can be replaced by the LSD scheme since the LSD scheme is a special case of the SD scheme.

Lemma 2.4 ([15])**.** *Let N_{max} be the number of leaf nodes in a full binary tree and r be the size of a revoked set. In the LSD scheme, the size of a private set is $O(\log^{1.5} N_{max})$ and the size of a covering set is at most $4r-2$.*

2.3 Revocable Identity-Based Encryption

Revocable IBE (RIBE) is an extension of IBE that can revoke a users if his credential is expired or revealed [4]. In RIBE, a sender creates a ciphertext for a receiver identity ID and a time T. A user first obtains a (long-term) private key SK_{ID} for his identity ID from a center, and the center periodically broadcasts an update key $UK_{T,R}$ for a time T and a revoked identity set R. If a user ID is not revoked in R of the update key, then he can derive a (short-term) decryption key $DK_{ID,T}$ for his identity ID and the time T from SK_{ID} and $UK_{T,R}$ and he can use this decryption key to decrypt the ciphertext. Note that the center does not encrypt an update key for broadcasting. The syntax of RIBE is formally defined as follows:

Definition 2.5 (Revocable IBE)**.** *A revocable IBE (RIBE) scheme that is associated with 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($1^\lambda, N_{max}$)*: The setup algorithm takes as input a security parameter 1^λ and the maximum number of users N_{max}. It outputs a master key MK, an (empty) revocation list RL, a state ST, and public parameters PP.*

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

UpdateKey(T, RL, MK, ST, PP)*: The update key generation algorithm takes as input an update time $T \in \mathcal{T}$, the revocation list RL, the master key MK, the state ST, and the public parameters PP. It outputs an update key $UK_{T,R}$ for T and R where R is a revoked identity set on the time T.*

DeriveKey($SK_{ID}, UK_{T,R}, PP$)*: The decryption key derivation algorithm takes as input a private key SK_{ID}, an update key $UK_{T,R}$, and the public parameters PP. It outputs a decryption key $DK_{ID,T}$ or $\perp$.*

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

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

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

*The correctness of RIBE is defined as follows: For all MK, RL, ST, and PP generated by **Setup**$(1^\lambda, N_{max})$, SK_{ID} generated by **GenKey**(ID, MK, ST, PP) for any ID, $UK_{T,R}$ generated by **UpdateKey**(T, RL, MK, ST, PP) for any T and RL, CT_{ID_c,T_c} generated by **Encrypt**(ID_c, T_c, M, PP) for any ID_c, T_c, and M, it is required that*

- *If $(ID \notin R)$, then **DeriveKey**$(SK_{ID}, UK_{T,R}, PP) = DK_{ID,T}$.*
- *If $(ID \in R)$, then **DeriveKey**$(SK_{ID}, UK_{T,R}, PP) = \perp$ with all but negligible probability.*
- *If $(ID_c = ID) \wedge (T_c = T)$, then **Decrypt**$(CT_{ID_c,T_c}, DK_{ID,T}, PP) = M$.*
- *If $(ID_c \neq ID) \vee (T_c \neq T)$, then **Decrypt**$(CT_{ID,T}, DK_{ID,T}, PP) = \perp$ with all but negligible probability.*

The security model of RIBE was introduced by Boldyreva et al. [4] and it was refined by Seo and Emura [37] by considering the decryption key exposure attacks. In this paper, we follow the refined security model of RIBE. In the security game of RIBE, an adversary adaptively request a private key for an identity *ID*, an update key for a time T and a current revocation list *RL*, and a decryption key for an identity *ID* and a time T. In the challenge step, the adversary submits a challenge identity ID^*, a challenge time T^*, and challenge messages M_0^*, M_1^* with additional restrictions and he receives a challenge ciphertext CT^* that is an encryption of a message M_μ^* for a random bit μ. After that, the adversary may request additional private key, update key, and decryption key queries, and finally he outputs a guess μ'. If his guess is correct, then he wins the game. The security of RIBE is formally defined as follows:

Definition 2.6 (Security)**.** *The security of RIBE under chosen plaintext attacks is defined in terms of the following experiment between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:*

1. ***Setup**: $\mathcal{C}$ generates a master key MK, a revocation list RL, a state ST, and public parameters PP by running **Setup**$(1^\lambda, N_{max})$. It keeps MK, RL, ST to itself and gives PP to $\mathcal{A}$.*
2. ***Phase 1**: $\mathcal{A}$ adaptively request a polynomial number of queries. These queries are processed as follows:*
    - *If this is a private key query for an identity ID, then it gives the corresponding private key SK_{ID} to $\mathcal{A}$ by running **GenKey**(ID, MK, ST, PP).*
    - *If this is an update key query for a time T, then it gives the corresponding update key $UK_{T,R}$ to $\mathcal{A}$ by running **UpdateKey**(T, RL, MK, ST, PP).*
    - *If this is a decryption key query for an identity ID and a time T, then it gives the corresponding decryption key $DK_{ID,T}$ to $\mathcal{A}$ by running **DeriveKey**$(SK_{ID}, UK_{T,R}, PP)$.*
    - *If this is a revocation query for an identity ID and a revocation time T, then it updates the revocation list RL by running **Revoke**(ID, T, RL, ST) with the restriction: The revocation query for a time T cannot be queried if the update key query for the time T was already requested.*

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

3. ***Challenge**: $\mathcal{A}$ submits a challenge identity ID^*, a challenge time T^*, and two challenge messages M_0^*, M_1^* with equal length with the following restrictions:*

   - *If a private key query for an identity ID such that $ID = ID^*$ was requested, then the identity ID^* should be revoked at some time T such that $T \leq T^*$.*
   - *The decryption key query for ID^* and T^* was not requested.*

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

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

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

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

We can also define the selective revocation list model that is weaker than the previous security model of RIBE. In the selective revocation list model, an adversary should submits a challenge identity ID^*, a challenge time T^*, and revocation identity set R^* at the time T^* before he receives the public parameters. This model was introduced by Boldyreva et al. [4] to prove their revocable ABE scheme.

Definition 2.7 (Selective Revocation List Security)**.** *The selective revocation list security of RIBE under chosen plaintext attacks is similar to the above security except that the adversary $\mathcal{A}$ submits a challenge identity ID^*, a challenge time T^*, and a revoked identity set R^* on the time T^* before receiving the public parameters. The advantage of $\mathcal{A}$ is defined as $\boldsymbol{Adv}_{RIBE,\mathcal{A}}^{IND\text{-}sRL\text{-}CPA}(\lambda) = \left|\Pr[\mu = \mu'] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the experiment. An RIBE scheme is secure in the selective revocation list model under chosen plaintext attacks if for all PPT adversary $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above experiment is negligible in the security parameter λ.*

3 Revocable IBE with Selective Security

In this section, we propose an RIBE scheme in prime-order bilinear groups and prove its security in the selective revocation list model under the standard assumption.

3.1 Bilinear Groups of Prime Order

Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of same 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:

1. Bilinearity: $\forall u, v \in \mathbb{G}$ and $\forall a, b \in \mathbb{Z}_p$, $e(u^a, v^b) = e(u,v)^{ab}$.

2. Non-degeneracy: $\exists g$ such that $e(g,g)$ has order p, that is, $e(g,g)$ is a generator of $\mathbb{G}_T$.

We say that $\mathbb{G}$ is a bilinear group if the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable. Furthermore, we assume that the description of $\mathbb{G}$ and $\mathbb{G}_T$ includes generators of $\mathbb{G}$ and $\mathbb{G}_T$ respectively.

3.2 Complexity Assumptions

Assumption 3.1 (Decisional Bilinear Diffie-Hellman, DBDH)**.** *Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of prime order p. Let g be generators of subgroups $\mathbb{G}$. The DBDH assumption is that if the challenge tuple*

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

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

3.3 Construction

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 RIBE scheme is described as follows:

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

1. It first generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p of bit size $\Theta(\lambda)$. Let g be a random generator of $\mathbb{G}$. It selects a random exponent $\alpha \in \mathbb{Z}_p$ and random elements $u_1, h_1, u_2, h_2 \in \mathbb{G}$. It sets a user list UL that contains a tuple (ID, u) as an empty one, and also sets a function list FL that contains a tuple $(GL, f_{GL}(x))$ for a group label GL as an empty one.
2. It obtains $\mathcal{BT}$ by running **SD.Setup**(N_{max}). Let $\mathcal{S}$ be the collection of all subsets $S_{i,j}$ of $\mathcal{BT}$. For each $S_{i,j} \in \mathcal{S}$, it sets $GL = L_i \| d_j$ and performs the following: If $(GL, *) \notin FL$, then it selects a random polynomial $f_{GL}(x)$ of degree 1 such that $f_{GL}(0) = \alpha$ and saves $(GL, f_{GL}(x))$ to FL.
3. It outputs a master key $MK = (\alpha, FL)$, an empty revocation list RL, a state $ST = (\mathcal{BT}, UL)$, and public parameters $PP = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, u_1, h_1, u_2, h_2, \Omega = e(g,g)^\alpha\big)$.

RIBE.GenKey(ID, MK, ST, PP): This algorithm takes as input an identity $ID \in \mathcal{I}$, the master key MK, the state $ST = (\mathcal{BT}, UL)$, and public parameters PP.

1. It assigns the identity ID to a leaf node v_u in $\mathcal{BT}$ that is not yet assigned where $u \in \mathcal{N}$ is an index that is assigned to ID. It saves (ID, u) to UL. Next, it obtains $PV_u = \{S_{i,j}\}$ by running **SD.Assign**$(\mathcal{BT}, u)$.
2. For each $S_{i,j} \in PV_u$, it performs the following steps: It sets $GL = L_i \| d_j$ and retrieves $(GL, f_{GL}(x))$ from FL. Next, it selects a random exponent $r_1 \in \mathbb{Z}_p$ and creates a personalized private key as

$$PSK_{ID,S_{i,j}} = \Big(K_0 = g^{f_{GL}(L_j)} (u_1^{ID} h_1)^{r_1},\ K_1 = g^{-r_1}\Big).$$

3. Finally, it outputs the updated state ST and a private key $SK_{ID} = \big(PV_u, \{PSK_{ID,S_{i,j}}\}_{S_{i,j} \in PV_u}\big)$.

RIBE.UpdateKey(T, RL, MK, ST, PP): This algorithm takes as input a time T, the revocation list RL, the master key MK, the state $ST = (\mathcal{BT}, UL)$, and public parameters PP.

1. It first defines the revoked set R of user identities on the time T from RL. That is, if there exists (ID', T') such that $(ID', T') \in RL$ for any $T' \leq T$, then $ID' \in R$. It also defines the revoked index set $RI \subseteq \mathcal{N}$ of the revoked identity set R by using UL. Next, it obtains $CV_{RI} = \{S_{i,j}\}$ by running **SD.Cover**$(\mathcal{BT}, RI)$.
2. For each $S_{i,j} \in CV_R$, it performs the following steps: It sets $GL = L_i \| d_j$ and retrieves $(GL, f_{GL}(x))$ from FL. Next, it selects a random exponent $r_2 \in \mathbb{Z}_p$ and creates a time-constrained update key as
$$TUK_{T,S_{i,j}} = \Big(U_0 = g^{f_{GL}(L_j)} (u_2^T h_2)^{r_2},\ U_1 = g^{-r_2} \Big).$$
3. Finally, it outputs the updated state ST and an update key $UK_{T,R} = \big(CV_{RI}, \{TUK_{T,S_{i,j}}\}_{S_{i,j} \in CV_{RI}} \big)$.

RIBE.DeriveKey$(SK_{ID}, UK_{T,R}, PP)$**:** This algorithm takes as input a private key $SK_{ID} = (PV_u, \{PSK_{ID,S_{i,j}}\})$, an update key $UK_{T,R} = (CV_{RI}, \{TUK_{T,S_{i,j}}\})$, and the public parameters PP.

1. If $ID \notin R$, then it obtains $(S_{i,j}, S_{i',j'})$ by running **SD.Match**(CV_{RI}, PV_u) such that $S_{i,j} \in CV_R$, $S_{i',j'} \in PV_u$, and $i = i' \wedge d_j = d_{j'} \wedge j \neq j'$. Otherwise, it outputs $\perp$.
2. It retrieves $TUK_{T,S_{i,j}} = (U_0, U_1)$ from $UK_{T,R}$ and $PSK_{ID,S_{i',j'}} = (K_0, K_1)$ from SK_{ID}. Note that $TUK_{T,S_{i,j}}$ and $PSK_{ID,S'_{i',j'}}$ share the same $f_{GL}(x)$ for $GL = L_i \| d_j$ since $i = i' \wedge d_j = d_{j'}$. Next, 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 chooses random exponents $r'_1, r'_2 \in \mathbb{Z}_p$ and creates decryption key components as
$$D_0 = (K_0)^{\Delta_{L_{j'},I}(0)} (U_0)^{\Delta_{L_j,I}(0)} \cdot (u_1^{ID} h_1)^{r'_1} (u_2^T h_2)^{r'_2},$$
$$D_1 = (K_1)^{\Delta_{L_{j'},I}(0)} \cdot g^{-r'_1},\ D_2 = (U_1)^{\Delta_{L_j,I}(0)} \cdot g^{-r'_2}.$$
3. Finally, it outputs a decryption key $DK_{ID,T} = \big(D_0, D_1, D_2 \big)$. Note that the components are formed as $D_0 = g^{\alpha} (u_1^{ID} h_1)^{r''_1} (u_2^T h_2)^{r''_2}, D_1 = g^{-r''_1}, D_2 = g^{-r''_2}$ since $f_{GL}(0) = \alpha$ where $r''_1 = r_1 \Delta_{L_{j'},I}(0) + r'_1$ and $r''_2 = r_2 \Delta_{L_j,I}(0) + r'_2$.

RIBE.Encrypt(ID, T, M, PP)**:** This algorithm takes as input an identity ID, a time T, a message M, and the public parameters PP. It first chooses a random exponent $s \in \mathbb{Z}_p$ and outputs a ciphertext by implicitly including ID and T as
$$CT_{ID,T} = \Big(C = \Omega^s \cdot M,\ C_0 = g^s,\ C_1 = (u_1^{ID} h_1)^s,\ C_2 = (u_2^T h_2)^s \Big).$$

RIBE.Decrypt$(CT_{ID,T}, DK_{ID',T'}, PP)$**:** This algorithm takes as input a ciphertext $CT_{ID,T} = (C, C_0, C_1, C_2)$, a decryption key $DK_{ID',T'} = (D_0, D_1, D_2)$, and the public parameters PP. If $(ID = ID') \wedge (T = T')$, then it outputs the encrypted message M as $M = C \cdot \big(\prod_{i=0}^{2} e(C_i, D_i) \big)^{-1}$. Otherwise, it outputs $\perp$.

RIBE.Revoke(ID, T, RL, ST)**:** This algorithm takes as input an identity ID, a revocation time T, the revocation list RL, and the state $ST = (\mathcal{BT}, UL, FL)$. If $(ID, *) \notin UL$, then it outputs $\perp$ since the private key of ID was not generated. Otherwise, it adds (ID, T) to RL. It outputs the updated revocation list RL.

3.4 Correctness

Let SK_{ID} be a private key for an identity ID, and $UK_{T,R}$ be an update key for a time T and a revoked identity set R. If $ID \notin R$, then two subsets $(S_{i,j}, S_{i',j'})$ such that $S_{i,j} \in CV_{RI}$, $S_{i',j'} \in PV_u$, and $i = i' \wedge d_j = d_{j'} \wedge j \neq j'$ can be obtained from the correctness of the SD scheme and a decryption key for ID and T can be derived from $PSK_{ID,S_{i',j'}} = (K_0, K_1)$ and $TUK_{T,S_{i,j}} = (U_0, U_1)$ as

$$\begin{aligned}
D_0 &= (K_0)^{\Delta_{L_{j'},J}(0)} (U_0)^{\Delta_{L_j,J}(0)} (u_1^{ID} h_1)^{r_1'} (u_2^T h_2)^{r_2'} \\
&= \left(g^{f_{GL}(L_{j'})} (u_1^{ID} h_1)^{r_1}\right)^{\Delta_{L_{j'},J}(0)} \left(g^{f_{GL}(L_j)} (u_2^T h_2)^{r_2}\right)^{\Delta_{L_j,J}(0)} (u_1^{ID} h_1)^{r_1'} (u_2^T h_2)^{r_2'} \\
&= g^{f_{GL}(0)} (u_1^{ID} h_1)^{r_1 \Delta_{L_{j'},J}(0) + r_1'} (u_2^T h_2)^{r_2 \Delta_{L_j,J}(0) + r_2'} = g^{\alpha} (u_1^{ID} h_1)^{r_1''} (u_2^T h_2)^{r_2''}, \\
D_1 &= (K_1)^{\Delta_{L_{j'},J}(0)} g^{-r_1'} = g^{-r_1 \Delta_{L_{j'},J}(0) - r_1'} = g^{-r_1''}, \\
D_2 &= (U_1)^{\Delta_{L_j,J}(0)} g^{-r_2'} = g^{-r_2 \Delta_{L_j,J}(0) - r_2'} = g^{-r_2''}
\end{aligned}$$

since $PSK_{ID,S_{i',j'}}$ and $TUK_{T,S_{i,j}}$ share the same polynomial $f_{GL}(x)$ and $L_j \neq L_{j'}$ where $r_1'' = r_1 \Delta_{L_{j'},J}(0) + r_1'$ and $r_2'' = r_2 \Delta_{L_j,J}(0) + r_2'$. If $ID \in R$, then there are no subsets $(S_{i,j}, S_{i',j'})$ such that $S_{i,j} \in CV_{RI}$, $S_{i',j'} \in PV_u$, and $i = i' \wedge d_j = d_{j'} \wedge j \neq j'$ from the correctness of the SD scheme.

Let $CT_{ID,T}$ be a ciphertext for an identity ID and a time T, and $DK_{ID',T'}$ be a decryption key for an identity ID' and a time T'. If $(ID = ID') \wedge (T = T')$, then the decryption algorithm correctly computes a session key by the following equation as

$$\begin{aligned}
\prod_{i=0}^{2} e_{1,2}(C_i, D_i) &= e(g^s, g^{\alpha} (u_1^{ID} h_1)^{r_1''} (u_2^T h_2)^{r_2''}) \cdot e((u_1^{ID} h_1)^s, g^{-r_1''}) \cdot e((u_2^T h_2)^s, g^{-r_2''}) \\
&= e(g,g)^{\alpha s} = \Omega^s.
\end{aligned}$$

3.5 Security Analysis

To prove the security of our RIBE scheme in the selective revocation list model, we use the partitioning method that was widely used for the security proof of other IBE schemes [5, 6, 40]. However, the direct use of the partitioning method does not work in RIBE since an adversary can request a private key query for a challenge identity ID^* and an update key query for a challenge time T^* that were not allowed in the security model of IBE. That is, the simulator that uses the partitioning method of IBE cannot handle the private key query for ID^* and the update key query for T^*.

To overcome this difficulty of using the partitioning method, we use the fact that a random polynomial $f(x)$ of degree one such that $f(0) = \alpha$ can be determined by one fixed point $(0, \alpha)$ and another random point $(\hat{x}, \hat{y})$. That is, if the adversary requests a private key for ID^* or an update key for T^*, then the simulator directly uses the values $\hat{y}$ of the random point $(\hat{x}, \hat{y})$ by implicitly defining $f(\hat{x}) = \hat{y}$ instead of using the Lagrange interpolation method to calculate $f(x')$ for some x' since the simulator cannot obtain an element for $f(0) = \alpha$ by using the partitioning method. To generate a private key for ID^*, the simulator assigns a random leaf node v_{u^*} to the identity ID^* and creates each personalized private key for a subset $S_{i,j}$ in PV_{u^*} by using a random point $(\hat{x}, \hat{y})$ that implicitly defines a random polynomial $f_{GL}(x)$ for the group GL. To generate an update key for T^*, the simulator obtains CV_{RI^*} from the given revocation identity set R^* and creates each time-constrained update key for a subset $S_{i,j}$ in CV_{RI^*} by using a random point $(\hat{x}, \hat{y})$ that implicitly defines a random polynomial $f_{GL}(x)$ for the group GL. Note that the simulation is only possible in the selective revocation list model since R^* is needed to generate the update key for T^*.

However, the above proof idea is not enough to assure us the soundness of the proof. For the assurance, we should show that a subset $S_{i,j}$ of the private key query for ID^* and a subset $S_{i,j'}$ of the update key query for T^* should be the same member ML in a group GL if they belong to the same group GL to use the above simulation technique that uses a random point $(\hat{x},\hat{y})$. At first, we have that each subset in the private set PV_{u^*} for ID^* belongs to different groups since PV_{u^*} is associated with a path, and each subset in the covering set CV_{RI^*} for T^* also belongs to different groups since CV_{RI^*} is a partition. If a subset $S_{i,j}$ of PV_{u^*} and a subset $S_{i,j'}$ of CV_{RI^*} belong to the same group GL, then $j = j'$ by the correctness of the SD scheme since $ID^* \in R^*$. Thus, two subsets $S_{i,j}$ and $S_{i,j'}$ should be the same member ML in the group GL.

Theorem 3.2. *The above RIBE scheme is secure in the selective revocation list model under chosen plaintext attacks if the DBDH assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}^{IND\text{-}sRL\text{-}CPA}_{RIBE,\mathcal{A}}(\lambda) \leq \boldsymbol{Adv}^{DBDH}_{\mathcal{B}}(\lambda)$.*

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

Init: $\mathcal{A}$ initially submits a challenge identity ID^*, a challenge time T^*, and a revoked identity set R^* on the time T^*.

Setup: $\mathcal{B}$ implicitly sets $\alpha = ab$ and proceeds as follows:

1. It first obtains $\mathcal{BT}$ by running **SD.Setup**(N_{max}). It sets UL and FL as an empty one respectively. It assigns ID^* to a random index u^* and saves (ID^*,u^*) to UL. For each $ID \in R^* \setminus \{ID^*\}$, it assigns ID to a random index u such that $(*,u) \notin UL$ and saves (ID,u) to UL. From R^*, it also defines the revoked index set RI^* by using UL.

2. It obtains PV_{u^*} and CV_{RI^*} by running **SD.Assign**$(\mathcal{BT},u^*)$ and **SD.Cover**$(\mathcal{BT},RI^*)$ respectively. If $ID^* \in R^*$, then it sets **FixedSubset**$(ID^*,R^*) = PV_{u^*} \cup CV_{RI^*}$. Otherwise, it sets **FixedSubset**$(ID^*,R^*) = CV_{RI^*}$. It sets the function list FL as follows:

   (a) For each $S_{i,j} \in$ **FixedSubset**(ID^*,R^*), it selects a random value $\hat{y} \in \mathbb{Z}_p$ and saves $(GL = L_i \| d_j, (\hat{x} = L_j, \hat{y}))$ to FL.

   (b) For each $S_{i,j} \in \mathcal{S} \setminus$ **FixedSubset**(ID^*,R^*), it selects random values $\hat{x},\hat{y} \in \mathbb{Z}_p$ and saves $(GL = L_i \| d_j, (\hat{x},\hat{y}))$ to FL if $(GL = L_i \| d_j, *) \notin FL$.

   Note that $f_{GL}(x)$ is implicitly defined by two points $(0,\alpha)$ and $(\hat{x},\hat{y})$ by using the Lagrange interpolation method.

3. It sets RL as an empty one and sets $ST = (\mathcal{BT},UL)$. It selects random exponents $h'_0,h'_1 \in \mathbb{Z}_p$ and publishes public parameters PP as

$$g,\ u_1 = g^a g^{u'_1}, h_1 = (g^a)^{-ID^*} g^{h'_1},\ u_2 = g^a g^{u'_2}, h_2 = (g^a)^{-T^*} g^{h'_2},\ \Omega = e(g^a,g^b).$$

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

- **Case** $ID \neq ID^*$: It performs the following steps:

1. It first selects a random exponent $r_1' \in \mathbb{Z}_p$ and builds temporal private key components for the point $(0, \alpha)$ as $A_0 = (g^b)^{-(u_1' ID + h_1')/(ID - ID^*)}(u_1^{ID} h_1)^{r_1'}$, $A_1 = (g^b)^{1/(ID-ID^*)} g^{-r_1'}$.
2. If $(ID, *) \in UL$, then it loads (ID, u) from UL. Otherwise, it selects a random index u such that $(u \neq u^*) \wedge ((*, u) \notin UL)$ and saves (ID, u) to UL. Next, it obtains $PV_u = \{S_{i,j}\}$ by running **SD.Assign**$(\mathcal{BT}, u)$.
3. For each $S_{i,j} \in PV_u$, it retrieves $(GL = L_i \| d_j, (\hat{x}, \hat{y}))$ from FL and performs the following:
   - Case $S_{i,j} \in$ **FixedSubset**(ID^*, R^*): Recall that $\hat{x} = L_j$ from the setup phase. It selects a random exponent $r_1 \in \mathbb{Z}_p$ and builds a personalized private key by implicitly setting $f_{GL}(L_j) = \hat{y}$ as
   $$PSK_{ID, S_{i,j}} = \Big(K_0 = g^{\hat{y}} (u_1^{ID} h_1)^{r_1},\ K_1 = g^{-r_1} \Big).$$
   - Case $S_{i,j} \notin$ **FixedSubset**(ID^*, R^*): Recall that $\hat{x} \neq L_j$ since $\hat{x}$ is a random value from the setup phase. 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 selects a random exponents $r_1'' \in \mathbb{Z}_p$ and builds a personalized private key as
   $$PSK_{ID, S_{i,j}} = \Big(K_0 = (A_0)^{\Delta_{0,I}(L_j)} (g^{\hat{y}})^{\Delta_{\hat{x},I}(L_j)} \cdot (u_1^{ID} h_1)^{r_1''},\ K_1 = (A_1)^{\Delta_{0,I}(L_j)} \cdot g^{-r_1''} \Big).$$
4. Finally, it creates a private key $SK_{ID} = \big(PV_u, \{ PSK_{ID, S_{i,j}} \}_{S_{i,j} \in PV_u} \big)$.

- **Case** $ID = ID^*$: In this case, we have $ID^* \in R^*$. It performs the following steps:

  1. It loads (ID^*, u^*) from UL where u^* is the pre-assigned index for ID^*. Next, it obtains $PV_{u^*} = \{S_{i,j}\}$ by running **SD.Assign**$(\mathcal{BT}, u^*)$.
  2. For each $S_{i,j} \in PV_{u^*}$, it retrieves $(GL = L_i \| d_j, (\hat{x}, \hat{y}))$ from FL and performs the following steps: Recall that $\hat{x} = L_j$ from the setup phase. It selects a random exponent $r_1 \in \mathbb{Z}_p$ and builds a personalized private key by implicitly setting $f_{GL}(L_j) = \hat{y}$ as
  $$PSK_{ID^*, S_{i,j}} = \Big(K_0 = g^{\hat{y}} (u_1^{ID} h_1)^{r_1},\ K_1 = g^{-r_1} \Big).$$
  3. Finally, it creates a private key $SK_{ID^*} = \big(PV_{u^*}, \{ PSK_{ID^*, S_{i,j}} \}_{S_{i,j} \in PV_{u^*}} \big)$.

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

- **Case** $T \neq T^*$: It performs the following steps.

  1. It first selects a random exponent $r_2' \in \mathbb{Z}_p$ and builds temporal update key components for the point $(0, \alpha)$ as $B_0 = (g^b)^{-(u_2' T + h_2')/(T - T^*)}(u_2^T h_2)^{r_2'}$, $B_1 = (g^b)^{1/(T-T^*)} g^{-r_2'}$.
  2. It defines the revoked identity set R on the time T and the revoked index set RI of R. Next, it obtains $CV_{RI} = \{S_{i,j}\}$ by running **SD.Cover**$(\mathcal{BT}, RI)$.
  3. For each $S_{i,j} \in CV_{RI}$, it retrieves $(GL = L_i \| d_j, (\hat{x}, \hat{y}))$ from FL and performs the following:
     - Case $S_{i,j} \in$ **FixedSubset**(ID^*, R^*): It selects a random exponent $r_2 \in \mathbb{Z}_p$ and creates a time-constrained update key by implicitly setting $f_{GL}(L_j) = \hat{y}$ as
     $$TUK_{T, S_{i,j}} = \Big(U_0 = g^{y_j} (u_2^T h_2)^{r_2},\ U_1 = g^{-r_2} \Big).$$

– Case $S_{i,j} \in$ **FixedSubset**(ID^*, R^*): It sets $I = \{0, \hat{x}\}$ and calculates two Lagrange coefficients $\Delta_{0,I}(L_j)$ and $\Delta_{\hat{z},I}(L_j)$. Next, it selects a random exponent $r_2'' \in \mathbb{Z}_p$ and builds a time-constrained update key as

$$TUK_{T,S_{i,j}} = \Big(U_0 = (B_0)^{\Delta_{0,I}(L_j)} (g^{\hat{y}})^{\Delta_{\hat{x},I}(L_j)} (u_2^T h_2)^{r_2''},\ U_1 = (B_1)^{\Delta_{0,I}(L_j)} g^{-r_2''} \Big).$$

4. Finally, it creates an update key $UK_{T,R} = \big(CV_{RI}, \{ TUK_{T,S_{i,j}} \}_{S_{i,j} \in CV_{RI}} \big)$.

- **Case** $T = T^*$: In this case, we have $R = R^*$. It performs the following steps:

1. It first defines the revoked identity set R on the time T and the revoked index set RI of R. Next, it obtains $CV_{RI^*} = \{S_{i,j}\}$ by running **SD.Cover**$(\mathcal{BT}, RI^*)$.
2. For each $S_{i,j} \in CV_{RI^*}$, it performs the following steps: It sets $GL = L_i \| d_j$ and retrieves $(GL, (\hat{x}, \hat{y}))$ from FL. Next, it selects a random exponent $r_2 \in \mathbb{Z}_p$ and builds a time-constrained update key by implicitly setting $f_{GL}(L_j) = \hat{y}$ as

$$TUK_{T^*,S_{i,j}} = \Big(U_0 = g^{\hat{y}} (u_2^{T^*} h_2)^{r_2},\ U_1 = g^{-r_2} \Big).$$

3. Finally, it creates an update key $UK_{T^*,R^*} = \big(CV_{RI^*}, \{ TUK_{T^*,S_{i,j}}) \}_{S_{i,j} \in CV_{RI^*}} \big)$.

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

- **Case** $ID \neq ID^*$: If $(ID, *) \notin UL$, then it selects a random index u such that $(*, u) \notin UL$ and saves (ID, u) to UL. It selects random exponents $r_1', r_2 \in \mathbb{Z}_p$ and creates a decryption key $DK_{ID,T}$ by implicitly setting $r_1 = -b/(ID - ID^*) + r_1'$ as

$$D_0 = (g^b)^{-(u_1' ID + h_1')/(ID - ID^*)} (u_1^{ID} h_1)^{r_1'} (u_2^T h_2)^{r_2},\ D_1 = (g^b)^{1/(ID - ID^*)} g^{-r_1'},\ D_2 = g^{-r_2}.$$

- **Case** $ID = ID^*$: In this case, we have $T \neq T^*$ from the restriction of Definition 2.7. It selects random exponents $r_1, r_2' \in \mathbb{Z}_p$ and creates a decryption key $DK_{ID,T}$ implicitly setting $r_2 = -b/(T - T^*) + r_2'$ as

$$D_0 = (u_1^{ID} h_1)^{r_1} (g^b)^{-(u_2' T + h_2')/(T - T^*)} (u_2^T h_2)^{r_2'},\ D_1 = g^{-r_1},\ D_2 = (g^b)^{1/(T - T^*)} g^{-r_2'}.$$

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

Challenge: $\mathcal{A}$ submits two challenge messages M_0^*, M_1^*. $\mathcal{B}$ chooses a random bit $\mu \in \{0,1\}$ and creates the challenge ciphertext CT^* by implicitly setting $s = c$ as

$$C = Z \cdot M_\mu^*,\ C_0 = g^c,\ C_1 = (g^c)^{u_1' ID^* + h_1'},\ C_2 = (g^c)^{u_2' T^* + h_2'}.$$

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

To finish the proof, we first show that the simulation is correct. The public parameters is correct since random exponents $u_1', h_1', u_2', h_2' \in \mathbb{Z}_p$ are chosen. We show that the private keys are correct. In case of

$ID \neq ID^*$, we have that a personalized private key for $S_{i,j}$ such that $S_{i,j} \notin \textbf{FixedSubset}(ID^*, R^*)$ is correctly distributed from the setting $r_1 = (-b/(ID - ID^*) + r_1')\Delta_{0,I}(L_j) + r_1''$ as

$$\begin{aligned}
K_0 &= g^{f_{GL}(L_j)}(u_1^{ID}h_1)^{r_1} = g^{\alpha\Delta_{0,I}(L_j)+\hat{y}\Delta_{\hat{x},I}(L_j)}(u_1^{ID}h_1)^{\tilde{r}_1\Delta_{0,I}(L_j)+r_1''} \\
&= \big(g^{\alpha}(u_1^{ID}h_1)^{\tilde{r}_1}\big)^{\Delta_{0,I}(L_j)} g^{\hat{y}\Delta_{\hat{x},I}(L_j)}(u_1^{ID}h_1)^{r_1''} \\
&= \big(g^{ab}((g^a)^{ID-ID^*}g^{u_1'ID+h_1'})^{-b/(ID-ID^*)+r_1'}\big)^{\Delta_{0,I}(L_j)} g^{\hat{y}\Delta_{\hat{x},I}(x_j)}(u_1^{ID}h_1)^{r_1''} \\
&= \big((g^b)^{-(u_1'ID+h_1')/(ID-ID^*)}(u_1^{ID}h_1)^{r_1'}\big)^{\Delta_{0,I}(L_j)} g^{\hat{y}\Delta_{\hat{x},I}(L_j)}(u_1^{ID}h_1)^{r_1''}, \\
K_1 &= g^{-r_1} = g^{(b/(ID-ID^*)-r_1')\Delta_{0,I}(L_j)-r_1''} = \big((g^b)^{1/(ID-ID^*)}g^{-r_1'}\big)^{\Delta_{0,I}(L_j)} g^{-r_1''}.
\end{aligned}$$

In case of $ID = ID^*$, we have that a personalized private key for $S_{i,j}$ is correctly distributed from the setting $L_j = \hat{x}$ and $f_{GL}(\hat{x}) = \hat{y}$ as

$$K_0 = g^{f_{GL}(L_j)}(u_1^{ID}h_1)^{r_1} = g^{\hat{y}}(u_1^{ID}h_1)^{r_1},\ K_1 = g^{-r_1}.$$

Next, we show that the update keys are correct. In case of $T \neq T^*$, we have that a time-constrained update key for $S_{i,j}$ such that $S_{i,j} \notin \textbf{FixedSubset}(ID^*, R^*)$ is correctly distributed from the setting $r_2 = (-b/(T - T^*) + r_2')\Delta_{0,I}(L_j) + r_2''$ as

$$\begin{aligned}
U_0 &= g^{f_{GL}(L_j)}(u_2^{T}h_2)^{r_2} = g^{\alpha\Delta_{0,I}(L_j)+\hat{y}\Delta_{\hat{x},I}(L_j)}(u_2^{T}h_2)^{\tilde{r}_2\Delta_{0,I}(L_j)+r_2''} \\
&= \big(g^{\alpha}(u_2^{T}h_2)^{\tilde{r}_2}\big)^{\Delta_{0,I}(L_j)} g^{\hat{y}\Delta_{\hat{x},I}(L_j)}(u_2^{T}h_2)^{r_2''} \\
&= \big(g^{ab}((g^a)^{T-T^*}g^{u_2'T+h_2'})^{-b/(T-T^*)+r_2'}\big)^{\Delta_{0,I}(L_j)} g^{\hat{y}\Delta_{\hat{x},I}(L_j)}(u_2^{T}h_2)^{r_2''} \\
&= \big((g^b)^{-(u_2'T+h_2')/(T-T^*)}(u_2^{T}h_2)^{r_2'}\big)^{\Delta_{0,I}(L_j)} g^{\hat{y}\Delta_{\hat{x},I}(L_j)}(u_2^{T}h_2)^{r_2''}, \\
U_1 &= g^{-r_2} = g^{(b/(T-T^*)-r_2')\Delta_{0,I}(L_j)-r_2''} = \big((g^b)^{1/(T-T^*)}g^{-r_2'}\big)^{\Delta_{0,I}(L_j)} g^{-r_2''}.
\end{aligned}$$

In case of $T = T^*$, we have that a time-constrained update key for $S_{i,j}$ is correctly distributed from the setting $L_j = \hat{x}$ and $f_{GL}(\hat{x}) = \hat{y}$ as

$$U_0 = g^{f_{GL}(L_j)}(u_2^{T}h_2)^{r_2} = g^{\hat{y}}(u_2^{T}h_2)^{r_2},\ U_1 = g^{-r_2}.$$

We show that the decryption keys are correct. In case of $ID \neq ID^*$, the decryption key is correctly distributed by setting $r_1 = -b/(ID - ID^*) + r_1'$ as

$$\begin{aligned}
D_0 &= g^{\alpha}(u_1^{ID}h_1)^{r_1}(u_2^{T}h_2)^{r_2} = g^{ab}((g^a)^{ID-ID^*}g^{u_1'ID+h_1'})^{-b/(ID-ID^*)+r_1'}(u_2^{T}h_2)^{r_2} \\
&= (g^b)^{-(u_1'ID+h_1')/(ID-ID^*)}(u_1^{ID}h_1)^{r_1'}(u_2^{T}h_2)^{r_2}, \\
D_1 &= g^{-r_1} = g^{b/(ID-ID^*)-r_1'} = (g^b)^{1/(ID-ID^*)}g^{-r_1'}.
\end{aligned}$$

In case of $ID = ID^*$, the decryption key is correctly distributed by setting $r_2 = -b/(T - T^*) + r_2'$ as

$$\begin{aligned}
D_0 &= g^{\alpha}(u_1^{ID}h_1)^{r_1}(u_2^{T}h_2)^{r_2} = g^{ab}(u_1^{ID}h_1)^{r_1}((g^a)^{T-T^*}g^{u_2'T+h_2'})^{-b/(T-T^*)+r_2'} \\
&= (u_1^{ID}h_1)^{r_1}(g^b)^{-(u_2'T+h_2')/(T-T^*)}(u_2^{T}h_2)^{r_2'}, \\
D_2 &= g^{-r_2} = g^{b/(T-T^*)-r_2'} = (g^b)^{1/(T-T^*)}g^{-r_2'}.
\end{aligned}$$

Finally, we show that the challenge ciphertext is correct. If $Z = Z_0 = e(g,g)^{abc}$ is given, then the challenge ciphertext is correctly distributed as

$$\begin{aligned} C &= \Omega^s = e(g,g)^{\alpha s} = e(g,g)^{abc},\ C_0 = g^s = g^c, \\ C_1 &= (u_1^{ID^*} h_1)^s = ((g^a)^{ID^* - ID^*} g^{u_1' ID^* + h_1'})^s = (g^c)^{u_1' ID^* + h_1'}, \\ C_2 &= (u_2^{T^*} h_2)^s = ((g^a)^{T^* - T^*} g^{u_2' T^* + h_2'})^s = (g^c)^{u_2' T^* + h_2'}. \end{aligned}$$

Otherwise, the component C of the challenge ciphertext is independent of δ in the $\mathcal{A}$'s view since Z_1 is a random element in $\mathbb{G}_T$.

Let η be a random bit for Z_η. From the above simulation, we have $\Pr[\mu = \mu' | \eta = 0] = \frac{1}{2} + \mathbf{Adv}_{RIBE,\mathcal{A}}^{IND\text{-}sRL\text{-}CPA}(\lambda)$ since the distribution of the simulation is correct, and we also have $\Pr[\mu = \mu' | \eta = 1] = \frac{1}{2}$ since μ is completely hidden to $\mathcal{A}$. Therefore we can obtain the following equation

$$\begin{aligned} \mathbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda) &= \big| \Pr[\mathcal{B}(D,Z_0) = 0] - \Pr[\mathcal{B}(D,Z_1) = 0] \big| \big| \Pr[\mu = \mu' | \eta = 0] \big| - \big| \Pr[\mu = \mu' | \eta = 1] \big| \\ &= \frac{1}{2} + \mathbf{Adv}_{RIBE,\mathcal{A}}^{IND\text{-}sRL\text{-}CPA}(\lambda) - \frac{1}{2} = \mathbf{Adv}_{RIBE,\mathcal{A}}^{IND\text{-}sRL\text{-}CPA}(\lambda). \end{aligned}$$

This completes our proof. □

3.6 Discussions

Efficiency. In our RIBE scheme that employs the SD scheme, the public parameters, a private key, an update key, and a ciphertext consist of $O(1), O(\log^2 N_{max}), O(r)$, and $O(1)$ number of group elements respectively where N_{max} is the maximum number of users and r is the number of revoked users in an update key. Compared with the previous RIBE scheme that employs the CS scheme that has $O(\log N_{max})$ number of group elements in a private key and $O(r \log(N_{max}/r))$ number of group elements in an update key, our RIBE scheme can reduce the number of group elements in an update key by increasing the number of group elements in a private key. If we use the LSD scheme of Halevy and Shamir [15] instead of the SD scheme, then we can reduce the number of group element in a private key to $O(\log^{1.5} N_{max})$.

Removing Stored Polynomials. The setup algorithm of our RIBE scheme should maintain the function list FL that stores a random polynomial $f(x)$ of degree one for each group in a tree. If N_{max} is the maximum number of users in the system, then the maximum number of groups is $N_{max} \log N_{max}$ since a group is defined by a node v_i that is not a leaf node in the tree and a depth d_j in the tree. A pseudo-random function *PRF* can be used to remove FL. That is, the master key MK consists of a random exponent α and a seed z for PRF, and a random polynomial $f_{GL}(x)$ for a group can be defined as $f_{GL}(x) = a_{GL} x + \alpha$ where GL is a group label and $a_{GL} = PRF_z(GL)$. The security of this modified scheme also holds from the security of the pseudo-random function.

Supporting an Exponential Number of Users. Our RIBE scheme takes the maximum number of users N_{max} as an input and assigns each user to a leaf node of a binary tree with depth $\log N_{max}$. To support an exponential number of users, a binary tree with depth 2λ can be used where λ is a security parameter and the bit size of an identity is 2λ. Additionally, a random function $f_{GL}(x)$ can be deterministically generated by using a pseudo-random function *PRF* instead of keeping a function list FL. Furthermore, if a user is assigned to a leaf node of a tree such that the label L of the leaf node is equal to the identity string *ID*, then the user list UL is not needed.

Layered Subset Difference. Compared with the previous RIBE scheme that employs the CS scheme, our RIBE scheme that uses the SD scheme reduce the number of group elements in update keys from

$O(r\log(N_{max}/r))$ to $O(r)$, but it increases the number of group elements in private keys from $O(\log N_{max})$ to $O(\log^2 N_{max})$. To reduce the size of private keys in RIBE, we can use the layered subset difference (LSD) scheme of Halevy and Shamir [15]. In the LSD scheme, the number of subsets in a private set PV_u is $O(\log^{1.5} N_{max})$ and the number of subsets in a covering set CV_R is still $O(r)$. Our RIBE scheme also can employ the LSD scheme since the LSD scheme is a special case of the SD scheme. The security proof of this RIBE scheme that uses the LSD scheme also holds.

Comparison with the RIBE scheme of Boldyreva et al. Compared with the previous RIBE schemes, our RIBE scheme and the RIBE scheme of Boldyreva et al. [4] has the similarity of using a random polynomial of degree one. However, the purpose of using a random polynomial is quite different between two schemes since we use a random polynomial of degree one for single member revocation to integrate with the SD scheme whereas they use a degree one polynomial for the FIBE scheme of Sahai and Waters [35].

Chosen-Ciphertext Security. The proposed RIBE scheme only provides the indistinguishability under chosen-plaintext attacks (IND-CPA). To provide the stronger indistinguishability under chosen-ciphertext attacks (IND-CCA) where an adversary can request decryption queries, we can use the general transformation of Canetti, Halevi, and Katz [9] since our RIBE scheme can be easily modified to support the HIBE scheme with 3-level of Boneh and Boyen [5]. That is, we can use the additional level of HIBE to provide the integrity of ciphertexts by using an one-time signature scheme. The proof of IND-CCA security easily follows since the decryption queries can be easily simulated by the private key delegation capability of the HIBE scheme.

Achieving Full Security. The security of our RIBE scheme is only proven in the selective revocation list model that is weaker than the well-known selective model since the revocation identity set R^* should be additionally given. The previous RIBE schemes that employ the CS scheme were already proven in the full model by using a fully secure IBE scheme since the assigned key of a subset in the CS scheme is independent with each other [25, 37]. However, it is not easy to prove the full security of our RIBE scheme by using a fully secure IBE scheme and a partitioning method since the assigned key of a subset in the SD scheme is dependent of each member in a group. In Section 4, we show that our RIBE scheme in composite-order bilinear groups can be proven in the full model if we use the dual system encryption technique of Waters [24, 41] instead of using the partitioning method.

4 Revocable IBE with Full Security

In this section, we propose an RIBE scheme in composite-order bilinear groups and prove its full security under static assumptions.

4.1 Bilinear Groups of Composite Order

Let $N = p_1 p_2 p_3$ where p_1, p_2, and p_3 are distinct prime numbers. Let $\mathbb{G}$ and $\mathbb{G}_T$ be two multiplicative cyclic groups of same composite order N and g be a generator of $\mathbb{G}$. The bilinear map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ has the following properties:

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

We say that $\mathbb{G}$ is a bilinear group if the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map e are all efficiently computable. Furthermore, we assume that the description of $\mathbb{G}$ and $\mathbb{G}_T$ includes generators of $\mathbb{G}$ and $\mathbb{G}_T$ respectively. We use the notation $\mathbb{G}_{p_i}$ to denote the subgroups of order p_i of $\mathbb{G}$ respectively. Similarly, we use the notation $\mathbb{G}_{T,p_i}$ to denote the subgroups of order p_i of $\mathbb{G}_T$ respectively.

4.2 Complexity Assumptions

We present tree static assumptions that were introduced by Lewko and Waters [24].

Assumption 4.1 (Subgroup Decision)**.** *Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The Assumption is that if the challenge tuple*

$$D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3}) \text{ and } Z,$$

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = X_1$ from $Z = Z_1 = X_1 R_1$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\boldsymbol{Adv}_{\mathcal{A}}^{SD}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $X_1 \in \mathbb{G}_{p_1}$ and $R_1 \in \mathbb{G}_{p_2}$.

Assumption 4.2 (General Subgroup Decision)**.** *Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The Assumption is that if the challenge tuple*

$$D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3}, X_1 R_1, R_2 Y_1) \text{ and } Z,$$

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = X_2 Y_2$ from $Z = Z_1 = X_2 R_3 Y_2$ with more than a negligible advantage. The advantage of $\mathcal{B}$ is defined as $\boldsymbol{Adv}_{\mathcal{A}}^{GSD}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $X_1, X_2 \in \mathbb{G}_{p_1}$, $R_1, R_2, R_3 \in \mathbb{G}_{p_2}$, and $Y_1, Y_2 \in \mathbb{G}_{p_3}$.

Assumption 4.3 (Composite Diffie-Hellman)**.** *Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The Assumption is that if the challenge tuple*

$$D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_2}, g_{p_3}, g_{p_1}^a R_1, g_{p_1}^b R_2) \text{ and } Z,$$

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = e(g_{p_1}, g_{p_1})^{ab}$ from $Z = Z_1 = e(g_{p_1}, g_{p_1})^c$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\boldsymbol{Adv}_{\mathcal{A}}^{ComDH}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b, c \in \mathbb{Z}_N$, and $R_1, R_2 \in \mathbb{G}_{p_2}$.

4.3 Construction

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 $i \in \mathbb{Z}_N$ and a set of indexes I in $\mathbb{Z}_N$. Our RIBE scheme is described as follows:

RIBE.Setup(1^λ, N_{max}): This algorithm takes as input a security parameter 1^λ and the maximum number N_{max} of users.

1. It first generates a bilinear group $\mathbb{G}$ of composite order $N = p_1 p_2 p_3$ where p_1, p_2, and p_3 are random primes. Let g_1 be a random generator of $\mathbb{G}_{p_1}$. It selects a random exponent $\alpha \in \mathbb{Z}_N$ and random elements $u_1, h_1, u_2, h_2 \in \mathbb{G}_{p_1}$, $Y \in \mathbb{G}_{p_3}$. It sets a user list UL that contains a tuple (ID, u) as an empty one, and also sets a function list FL that contains a tuple $(GL, f_{GL}(x))$ for a group label GL as an empty one.

2. It obtains $\mathcal{BT}$ by running **SD.Setup**(N_{max}). Let $\mathcal{S}$ be the collection of all subsets $S_{i,j}$ of $\mathcal{BT}$. For each $S_{i,j} \in \mathcal{S}$, it sets $GL = L_i \| d_j$ and performs the following steps: If $(GL, *) \notin FL$, then it selects a random polynomial $f_{GL}(x)$ of degree 1 such that $f_{GL}(0) = \alpha$ and saves $(GL, f_{GL}(x))$ to FL.
3. It outputs a master key $MK = (\alpha, FL)$, an empty revocation list RL, a state $ST = (\mathcal{BT}, UL)$, and public parameters $PP = \big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_1, Y, u_1, h_1, u_2, h_2, \Omega = e(g,g)^{\alpha}\big)$.

RIBE.GenKey(ID, MK, ST, PP**):** This algorithm takes as input an identity $ID \in \mathcal{I}$, the master key MK, the state $ST = (\mathcal{BT}, UL)$, and public parameters PP.

1. It first randomly assigns the identity ID to a leaf node v_u in $\mathcal{BT}$ that is not yet assigned where $u \in \mathcal{N}$ is an index assigned to ID. It saves (ID, u) to UL. Next, it obtains $PV_u = \{S_{i,j}\}$ by running **SD.Assign**$(\mathcal{BT}, u)$.
2. For each $S_{i,j} \in PV_u$, it performs the following steps: It sets $GL = L_i \| d_j$ and retrieves $(GL, f_{GL}(x))$ from FL. Next, it selects random $r_1 \in \mathbb{Z}_N$, $Y_0, Y_1 \in \mathbb{G}_{p_3}$ and creates a personalized private key as
$$PSK_{ID,S_{i,j}} = \Big(K_0 = g^{f_{GL}(L_j)} (u_1^{ID} h_1)^{r_1} Y_0,\ K_1 = g^{-r_1} Y_1\Big).$$
3. Finally, it outputs the state ST and a private key $SK_{ID} = \big(PV_u, \{PSK_{ID,S_{i,j}}\}_{S_{i,j} \in PV_u}\big)$.

RIBE.UpdateKey(T, RL, MK, ST, PP**):** This algorithm takes as input a time T, the revocation list RL, the master key MK, the state $ST = (\mathcal{BT}, UL, FL)$, and public parameters PP.

1. It first defines the revoked set R of user identities on the time T from RL. That is, if there exists (ID', T') such that $(ID', T') \in RL$ for any $T' \leq T$, then $ID' \in R$. It also defines the revoked index set $RI \subseteq \mathcal{N}$ of the revoked identity set R by using UL. Next, it obtains $CV_{RI} = \{S_{i,j}\}$ by running **SD.Cover**$(\mathcal{BT}, RI)$.
2. For each $S_{i,j} \in CV_R$, it performs the following steps: It sets $GL = L_i \| d_j$ and retrieves $(GL, f_{GL}(x))$ from FL. Next, it selects random $r_2 \in \mathbb{Z}_N$, $Y_0, Y_1 \in \mathbb{G}_{p_3}$ and creates a time-constrained update key as
$$TUK_{T,S_{i,j}} = \Big(U_0 = g^{f_{GL}(L_j)} (u_2^{T} h_2)^{r_2} Y_0,\ U_1 = g^{-r_2} Y_1\Big).$$
3. Finally, it outputs the state ST and an update key $UK_{T,R} = \big(CV_{RI}, \{TUK_{T,S_{i,j}}\}_{S_{i,j} \in CV_{RI}}\big)$.

RIBE.DeriveKey($SK_{ID}, UK_{T,R}, PP$**):** This algorithm takes as input a private key $SK_{ID} = (PV_u, \{PSK_{ID,S_{i,j}}\})$ for an identity ID, an update key $UK_{T,R} = (CV_{RI}, \{TUK_{T,S_{i,j}}\})$ for a time T and a revoked set R of identities, and the public parameters PP.

1. If $ID \notin R$, then it obtains $(S_{i,j}, S_{i',j'})$ by running **SD.Match**(CV_{RI}, PV_u) such that $S_{i,j} \in CV_R$, $S_{i',j'} \in PV_u$, and $i = i' \wedge d_j = d_{j'} \wedge j \neq j'$. Otherwise, it outputs $\perp$.
2. It retrieves $TUK_{T,S_{i,j}} = (U_0, U_1)$ from $UK_{T,R}$, and $PSK_{ID,S_{i',j'}} = (K_0, K_1)$ from SK_{ID}. Next, 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 chooses random $r_1', r_2' \in \mathbb{Z}_N$, $Y_0', Y_1', Y_2' \in \mathbb{G}_{p_3}$ and creates decryption key components as
$$\begin{aligned} D_0 &= (K_0)^{\Delta_{L_{j'},I}(0)} (U_0)^{\Delta_{L_j,I}(0)} \cdot (u_1^{ID} h_1)^{r_1'} (u_2^{T} h_2)^{r_2'} Y_0', \\ D_1 &= (K_1)^{\Delta_{L_{j'},I}(0)} \cdot g^{-r_1'} Y_1',\ D_2 = (U_1)^{\Delta_{L_j,I}(0)} \cdot g^{-r_2'} Y_2'. \end{aligned}$$

3. Finally, it outputs a decryption key $DK_{ID,T} = \big(D_0, D_1, D_2\big)$.

RIBE.Encrypt(ID, T, M, PP): This algorithm takes as input an identity ID, a time T, a message M, and the public parameters PP. It first chooses a random exponent $s \in \mathbb{Z}_N$ and outputs a ciphertext by implicitly including ID and T as

$$CT_{ID,T} = \Big(C = \Omega^s \cdot M,\ C_0 = g^s,\ C_1 = (u_1^{ID} h_1)^s,\ C_2 = (u_2^T h_2)^s\Big).$$

RIBE.Decrypt($CT_{ID,T}, DK_{ID',T'}, PP$): This algorithm takes as input a ciphertext $CT_{ID,T} = (C, C_0, C_1, C_2)$, a decryption key $DK_{ID',T'} = (D_0, D_1, D_2)$, and the public parameters PP. If $(ID = ID') \wedge (T = T')$, then it outputs the encrypted message M as $M = C \cdot \big(\prod_{i=0}^{2} e(C_i, D_i)\big)^{-1}$. Otherwise, it outputs $\perp$.

RIBE.Revoke(ID, T, RL, ST): This algorithm is the same as that of Section 3.3.

4.4 Security Analysis

To prove the security of our RIBE scheme in composite-order bilinear groups, we use the dual system encryption technique of Waters [24, 41]. The dual system encryption technique was successfully used to prove the security of HIBE, ABE, and PE schemes [22,24,32,41]. However, the the dual system encryption does not directly applicable to the RIBE scheme since the adversary of RIBE can request a private key query for a challenge identity ID^* and an update key query for a challenge time T^* that were not allowed in IBE, HIBE, and ABE. Note that the dual system encryption technique essentially uses those restrictions of adversary in IBE, HIBE, and ABE when it changes normal private keys to semi-functional private keys to solve the paradox of dual system encryption. To handle the private key query for ID^* and the update key query for T^* in RIBE, we need different techniques for dual system encryption.

We organize personalized private keys in a private key and time-constrained update keys in an update key in the order of groups, and change those keys in the same group from normal to semi-functional through hybrid games. Note that this strategy that change keys in the same group from normal to semi-functional was used in the security proof of RS-ABE [18, 34]. In contrast to the RS-ABE scheme that uses the CS scheme, our RIBE scheme uses the SD scheme that has a complex key assignment part and this makes it difficult for us to prove the security. To overcome this difficulty, we carefully redesign semi-functional types and hybrid games by using the fact that there are only one private key query for ID^* and one update key query for T^* in RIBE that match to the challenge ciphertext.

Theorem 4.4. *The above RIBE scheme is fully secure under chosen plaintext attacks if the SD, GSD, and DBDH assumptions hold. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{RIBE,\mathcal{A}}^{IND\text{-}CPA}(\lambda) \leq \boldsymbol{Adv}_{\mathcal{B}}^{SD}(\lambda) + O(q^2 \log^2 N_{max} + q^2 r_{max}) \boldsymbol{Adv}_{\mathcal{B}}^{GSD}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}}^{DBDH}(\lambda)$ where q is the maximum number of private key, update key, and decryption key queries of $\mathcal{A}$.*

Proof. We first define the semi-functional type of private keys, update keys, decryption keys, and ciphertexts. For the semi-functional type, we let g_2 denote a fixed generator of the subgroup $\mathbb{G}_{p_2}$.

RIBE.GenKeySF. This algorithm first creates a normal private key $SK'_{ID} = (PV_u, \{PSK'_{ID,S_{i,j}}\}_{S_{i,j} \in PV_u})$ by using MK where $PV_u = \{S_{i,j}\}$ and $PSK'_{ID,S_{i,j}} = (K'_0, K'_1)$. For each $S_{i,j} \in PV_u$, it chooses a random exponent $\delta_{i,j} \in \mathbb{Z}_N$ once for $S_{i,j}$ and builds a semi-functional personalized private key $PSK_{ID,S_{i,j}} = \big(K_0 = K'_0 g_2^{\delta_{i,j}}, K_1 = K'_1\big)$. It outputs a semi-functional private key $SK_{ID} = (PV_u, \{PSK_{ID,S_{i,j}}\}_{S_{i,j} \in PV_u})$.

RIBE.UpdateKeySF. This algorithm first creates a normal update key $UK'_{T,R} = (CV_{RI}, \{TUK'_{T,S_{i,j}}\}_{S_{i,j} \in CV_{RI}})$ by using MK. For each $S_{i,j} \in CV_{RI}$, it chooses a random exponent $\delta_{i,j} \in \mathbb{Z}_N$ once for $S_{i,j}$ and builds a semi-functional time-constrained private key $TUK_{T,S_{i,j}} = \big(U_0 = U'_0 g_2^{\delta_{i,j}}, U_1 = U'_1\big)$. It outputs a semi-functional update key $UK_{T,R} = (CV_{RI}, \{PUK_{T,S_{i,j}}\}_{S_{i,j} \in CV_{RI}})$.

RIBE.DeriveKeySF. This algorithm first creates a normal decryption key $DK'_{ID,T} = (D'_0, D'_1, D'_2)$ by using MK. It chooses a random exponent $a \in \mathbb{Z}_N$ and outputs a semi-functional decryption key $DK_{ID,T} = \big(D_0 = D'_0 g_2^a, D_1 = D'_1, D_2 = D'_2\big)$.

RIBE.EncryptSF. This algorithm first creates a normal ciphertext $CT'_{ID,T} = (C', C'_0, C'_1, C'_2)$. It chooses random exponents $c, d_1, d_2 \in \mathbb{Z}_N$ and outputs semi-functional ciphertext $CT_{ID,T} = \big(C_0 = C'_0 g_2^c, C_1 = C'_1 g_2^{cd_1}, C_2 = C'_2 g_2^{cd_2}\big)$.

Note that if a semi-functional decryption key is used to decrypt a semi-functional ciphertext, then the decryption fails since an additional random element $e(g_2, g_2)^{ac}$ is left.

To prove the security, we use a sequence of hybrid games. For the hybrid games that change personalized private keys (or time-constrained update keys) that are related with a subset $S_{i,j}$ from normal ones to semi-functional ones, we need to state additional information of a subset $S_{i,j}$ in $\mathcal{BT}$. Note that a personalized private key for $S_{i,j}$ and a time-constrained update key for $S_{i',j'}$ share the same polynomial $f(x)$ if $i = i' \wedge d_j = d_{j'}$ since they belong to the same group $GL = L_i \| d_j$ where $L_i = L(v_i)$ and d_j is the depth of v_j. Thus we associate a personalized private key (or a time-constrained update key) with a tuple of indexes (i_g, i_m, i_c) to state additional information about the group GL where i_g is a group index, i_m is a member index, and i_c is a counter index.

Suppose that a personalized private key (or a time-constrained update key) is related with a subset $S_{i,j}$, Then this key has a group label $GL = L_i \| d_j$ and a member label $ML = L_j$. The group index i_g for personalized private keys (or time-constrained update keys) is assigned as follows: If the group GL appears first time in queries, then we set i_g as the number of distinct group GL' in previous queries plus one. If the group GL already appeared before in queries, then we set i_g as the value i'_g of previous personalized private key (or time-constrained update key) with the same group GL. The member index i_m for the group i_g is assigned as follows: If the member ML for this group GL appears first time in queries, then we set i_m as the number of distinct members for this group GL in previous queries plus one. If the member ML for this group already appeared before in queries, then we set i_m as the value i'_m of previous one. The counter index i_c is assigned as follows: If the group and member (GL, ML) appears first time in queries, then we set i_c as one. If the group and member (GL, ML) appeared before in queries, then we set i_c as the number of queries with the group and member (GL, ML) that appeared before plus one.

The security proof consists of the sequence of hybrid games: The first game will be the original security game and the last one will be a game such that the adversary has no advantage. We define the games as follows:

Game $\mathbf{G}_0$. This game is the original security game. In this game, all personalized private keys, time-constrained update keys, decryption keys, and the challenge ciphertext are normal.

Game $\mathbf{G}_1$. In the next game, all personalized private keys, time-constrained update keys, and decryption keys are normal, but the challenge ciphertext is semi-functional.

Game $\mathbf{G}_2$. Next, we define a new game $\mathbf{G}_2$. In this game, all personalized private keys, time-constrained update keys, and the challenge ciphertext are semi-functional, but decryption keys are normal. For the security proof, we additionally define a sequence of games $\mathbf{G}_{1,1},\ldots,\mathbf{G}_{1,h},\ldots,\mathbf{G}_{1,q_g}$ where $\mathbf{G}_1=\mathbf{G}_{1,0}$ and q_g is the maximum number of groups that are used in private keys and update keys. In the game $\mathbf{G}_{1,h}$ for $1\le h\le q_g$, the challenge ciphertext is semi-functional, personalized private keys and time-constrained update keys with a group index i_g such that $i_g\le h$ are semi-functional, and the remaining personalized private keys and time-constrained update keys with an index i_g such that $h<i_g$ are normal. It is obvious that $\mathbf{G}_{1,q_g}=\mathbf{G}_2$.

Game $\mathbf{G}_3$. In this game $\mathbf{G}_3$, all personalized private keys, time-constrained update keys, decryption keys, and the the challenge ciphertext are semi-functional.

Game $\mathbf{G}_4$. In the final game $\mathbf{G}_4$, all personalized private keys, time-constrained update keys, decryption keys, and the challenge ciphertext are semi-functional, but the challenge ciphertext component C is random.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_j}$ be the advantage of $\mathcal{A}$ in the game $\mathbf{G}_j$. We easily obtain that $\mathbf{Adv}_{RIBE,\mathcal{A}}^{\mathit{IND\text{-}CPA}}(\lambda)=\mathbf{Adv}_{\mathcal{A}}^{G_0}$, $\mathbf{Adv}_{\mathcal{A}}^{G_1}=\mathbf{Adv}_{\mathcal{A}}^{G_{1,0,2}}$, $\mathbf{Adv}_{\mathcal{A}}^{G_2}=\mathbf{Adv}_{\mathcal{A}}^{G_{1,q,2}}$, and $\mathbf{Adv}_{\mathcal{A}}^{G_4}=0$. Through the following Lemmas 4.5, 4.6, 4.7, and 4.10, we can obtain the following equation

$$\begin{aligned}
&\mathbf{Adv}_{RIBE,\mathcal{A}}^{\mathit{IND\text{-}CPA}}(\lambda)\\
&\le \left|\mathbf{Adv}_{\mathcal{A}}^{G_0}-\mathbf{Adv}_{\mathcal{A}}^{G_1}\right|+\sum_{h=1}^{q_g}\left|\mathbf{Adv}_{\mathcal{A}}^{G_{1,h-1}}-\mathbf{Adv}_{\mathcal{A}}^{G_{1,h}}\right|+\left|\mathbf{Adv}_{\mathcal{A}}^{G_2}-\mathbf{Adv}_{\mathcal{A}}^{G_3}\right|+\left|\mathbf{Adv}_{\mathcal{A}}^{G_3}-\mathbf{Adv}_{\mathcal{A}}^{G_4}\right|\\
&\le \mathbf{Adv}_{\mathcal{B}}^{SD}(\lambda)+5(q_{sk}+q_{uk})\sum_{h=1}^{q_g}\sum_{h_m=1}^{q_m}\sum_{h_c=1}^{q_c}\mathbf{Adv}_{\mathcal{B}}^{GSD}(\lambda)+2q_{dk}\mathbf{Adv}_{\mathcal{B}}^{GSD}(\lambda)+\mathbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda)\\
&\le \mathbf{Adv}_{\mathcal{B}}^{SD}(\lambda)+\big(\frac{5}{2}q^2\log^2 N_{max}+10q^2 r_{max}+2q\big)\mathbf{Adv}_{\mathcal{B}}^{GSD}(\lambda)+\mathbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda).
\end{aligned}$$

where $q=q_{sk}+q_{uk}+q_{dk}$. This completes our proof. □

Lemma 4.5. *If the SD assumption holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the SD assumption using $\mathcal{A}$ is given: a challenge tuple $D=((N,\mathbb{G},\mathbb{G}_T,e), g_{p_1},g_{p_3})$ and Z where $Z=Z_0=X_1\in\mathbb{G}_{p_1}$ or $Z=Z_1=X_1R_1\in\mathbb{G}_{p_1p_2}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ first chooses random exponents $u_1',h_1',u_2',h_2',\alpha\in\mathbb{Z}_N$. It sets $\mathcal{BT}$ by running **SD.Setup** and FL by selecting $f_{GL}(x)$ for each GL in $\mathcal{BT}$. It sets $MK=(\alpha,FL)$, $RL=\emptyset$, $ST=(\mathcal{BT},UL=\emptyset)$, and publishes $PP=\big((N,\mathbb{G},\mathbb{G}_T,e),g=g_{p_1},Y=g_{p_3},u_1=g_{p_1}^{u_1'},h_1=g_{p_1}^{h_1'},u_2=g_{p_1}^{u_2'},h_2=g_{p_1}^{h_2'},\Omega=e(g,g)^{\alpha}\big)$.
Phase 1: To response queries, $\mathcal{B}$ creates normal keys by running the normal algorithms since it knows MK. Note that it cannot create semi-functional keys since it does not know g_{p_2}.
Challenge: $\mathcal{A}$ submits a challenge identity ID^*, a challenge time T^*, and challenge messages M_0^*,M_1^*. $\mathcal{B}$ flips a random coin $\mu\in\{0,1\}$ and creates a challenge ciphertext CT^* by implicitly setting g^s to be the $\mathbb{G}_{p_1}$ part of Z as $CT^*=\big(C=e(Z,g)^{\alpha}\cdot M_{\mu}^*,\ C_0=Z,\ C_1=(Z)^{u_1'ID^*+h_1'},\ C_2=(Z)^{u_2'T^*+h_2'}\big)$.
Phase 2: Same as Phase 1.
Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu=\mu'$, then $\mathcal{B}$ outputs 1. Otherwise, it outputs 0.

If $Z = Z_0 = X_1$, then the simulation is the same as $\mathbf{G}_0$ since the challenge ciphertext is correctly distributed. If $Z = Z_1 = X_1R_1$, then the simulation is the same as $\mathbf{G}_1$ since the challenge ciphertext is generated as semi-functional by implicitly setting $d_1 \equiv u'_1 ID^* + h'_1 \mod p_2, d_2 \equiv u'_2 T^* + h'_2 \mod p_2$. The values u'_1, h'_1, u'_2, h'_2 modulo p_2 are not correlated with their values modulo p_1 by the Chinese Remainder Theorem (CRT). This completes our proof. □

Lemma 4.6. *If the GSD assumption holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_{1,h-1}$ *and* $\mathbf{G}_{1,h}$ *with a non-negligible advantage.*

Proof. We first divide the behavior of an adversary as two types: Type-I and Type-II. We next show that this lemma holds for two types of the adversary. Let ID^* and T^* be the challenge identity and the challenge time respectively. The two types of adversaries are formally defined as follows:

Type-I. An adversary is Type-I if it queries on an identity ID such that $ID = ID^*$ for at least one personalized private key with the group index h, or it queries on a time T such that $T = T^*$ for at least one time-constrained update key with the group index h. More specifically, this adversary can be divided as follows:

- Type-I-A. This adversary queries on an identity ID such that $ID \neq ID^*$ for all personalized private keys with the group index h, and it queries on a time T such that $T = T^*$ for at least one time-constrained update key with the group index h.
- Type-I-B. This adversary queries on a time T such that $T \neq T^*$ for all time-constrained update keys with h, and it queries on an identity ID such that $ID = ID^*$ for at least one personalized private key with h.
- Type-I-C. This adversary queries on an identity ID such that $ID = ID^*$ for at least one personalized private key with h, and it queries on a time T such that $T = T^*$ for at least one time-constrained update key with h.

Type-II. An adversary is Type-II if it queries on an identity ID such that $ID \neq ID^*$ for all personalized private keys with the group index h, and it queries on a time T such that $T \neq T^*$ for all time-constrained update keys with the group index h.

Let CV_{RI^*} be the covering set of the update key for the time T^* and revoked set R^*, and PV_{u^*} be the private set of the private key for the identity ID^*. Let h^*_m be a member index of the group index h such that the personalized private key for ID^* or the time-constrained update key for T^* belong to the member index h^*_m. If the adversary is Type-I-A, then there is only one member index h^*_m since CV_{RI^*} is a partition. If the adversary is In Type-I-B, then there is only one member index h^*_m since PV_{u^*} is related with a path. If the adversary is Type-I-C, the member index h^*_m of CV_{RI^*} with the group index h should be the same as that of PV_{u^*} with the same group index h in the SD scheme if $ID^* \in R^*$. If the adversary is Type-II, then there is no member index h^*_m since the adversary does not request a key query for ID^* or T^*.

For the Type-I adversary $\mathcal{A}_I$, we define hybrid games $\mathbf{H}_{(1,1),1}, \mathbf{H}_{(1,1),2}, \ldots, \mathbf{H}_{(q_m,q_c),1}, \mathbf{H}_{(q_m,q_c),2} = \mathbf{H}'_{(q_m,q_c),2}$, $\mathbf{H}'_{(q_m,q_c),1}, \ldots, \mathbf{H}'_{(1,1),2}, \mathbf{H}'_{(1,1),1}, \mathbf{H}'_{(1,0),2}, \mathbf{H}''$ where $\mathbf{G}_{1,h-1} = \mathbf{H}_{(1,0),2}$, $\mathbf{H}'' = \mathbf{G}_{1,h}$, q_m is the maximum number of distinct member subsets of the group index h, and q_c is the maximum number of queries for one member subset. The games are formally defined as follows:

Game $\mathbf{H}_{(h_m,h_c),1}$. This game $\mathbf{H}_{(h_m,h_c),1}$ for $1 \leq h_m \leq q_m$ and $1 \leq h_c \leq q_c$ is almost the same as $\mathbf{G}_{1,h-1}$ except the generation of personalized private keys and time-constrained update keys with the group index

h. These personalized private keys and time-constrained update keys with indexes $(i_g = h, i_m, i_c)$ are generated as follows:

- **Case** $i_g < h$: The keys (personalized private keys and time-constrained update keys) are generated as semi-functional.
- **Case** $i_g = h$: The keys are generated as follows:
  - $(i_m \neq h_m^*) \wedge (i_m < h_m)$ or $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c < h_c)$:
    If this is a personalized private key query, then it generates a normal $PSK' = (K_0', K_1')$ and creates the semi-functional personalized private key of type 2 as $PSK_{ID,S_{i,j}} = \big(K_0 = K_0' g_2^a, K_1 = K_1'\big)$ by selecting a new random exponent $a \in \mathbb{Z}_N$.
    If this is a time-constrained update key query, then it generates a normal $TUK' = (U_0', U_1')$ and creates the semi-functional time-constrained update key of type 2 as $TUK_{T,S_{i,j}} = \big(U_0 = U_0' g_2^a, U_1 = U_1'\big)$ by selecting a new random exponent $a \in \mathbb{Z}_N$.
  - $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$:
    If this is a personalized private key query, then it generates a normal $PSK' = (K_0', K_1')$ and creates the semi-functional personalized private key of type 1 as $PSK_{ID,S_{i,j}} = \big(K_0 = K_0' g_2^a, K_1 = K_1' g_2^b\big)$ by selecting new random exponents $a, b \in \mathbb{Z}_N$.
    If this is a time-constrained update key query, then it generates a normal $TUK' = (U_0', U_1')$ and creates the semi-functional time-constrained update key of type 1 as $TUK_{T,S_{i,j}} = \big(U_0 = U_0' g_2^a, U_1 = U_1' g_2^b\big)$ by selecting new random exponents $a, b \in \mathbb{Z}_N$.
  - $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (h_c < i_c)$ or $(i_m \neq h_m^*) \wedge (h_m < i_m)$: It simply creates a normal type key.
  - $(i_m = h_m^*)$: It simply creates a normal type key.
- **Case** $i_g > h$: The keys are generated as normal.

Note that if a semi-functional personalized private key of type 1 and a normal time-constrained update key are used to decrypt a semi-functional ciphertext, then the decryption fails since an additional random element $e(g_2, g_2)^{c(a-bd_1)}$ is left. If $a = bd_1$, then the the decryption succeeds and this personalized private key is *nominally* semi-functional of type 1. Similarly, if a semi-functional time-constrained update key of type 1 and a normal personalized private key are used to decrypt a semi-functional ciphertext, then the decryption fails since an additional random element $e(g_2, g_2)^{c(a-bd_1)}$ is left. If $a = bd_1$, then the the decryption succeeds and this time-constrained update key is *nominally* semi-functional of type 1.

Game $\mathbf{H}_{(h_m,h_c),2}$. This game $\mathbf{H}_{(h_m,h_c),2}$ is almost the same as $\mathbf{H}_{(h_m,h_c),1}$ except that the personalized private key (or the time-constrained update key) with the indexes $(i_g = h, i_m, i_c)$ such that $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$ is generated with $b = 0$. In the game $\mathbf{H}_{(q_m,q_c),2}$, all personalized private keys and time-constrained update keys with the group index h are semi-functional of type 2 except that personalized private keys and time-constrained update keys with the member index h_m^* are normal.

Game $\mathbf{H}'_{(h_m,h_c),1}$. This game $\mathbf{H}'_{(h_m,h_c),1}$ is almost the same as $\mathbf{H}_{(h_m,h_c),1}$ except the generation of a personalized private key (or a time-constrained update key) with the indexes $(i_g = h, i_m, i_c)$ such that $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (h_c \leq i_c)$ or $(i_m \neq h_m) \wedge (h_m < i_m)$. These personalized private keys (or time-constrained update keys) are generated as follows:

- $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$: Let $\delta_{i,j}$ be a random exponent in $\mathbb{Z}_N$ that is fixed for this member subset $S_{i,j}$.

  If this is a personalized private key query, then it generates $PSK'' = (K_0'', K_1'')$ as the same as $\mathbf{H}_{(h_m,h_c),1}$ and creates the semi-functional personalized private key as $PSK_{ID,S_{i,j}} = \big(K_0 = K_0'' g_2^{\delta_{i,j}}, K_1 = K_1''\big)$.

  If this is a time-constrained update key query, then it generates $TUK'' = (U_0'', U_1'')$ as the same as $\mathbf{H}_{(h_m,h_c),1}$ and creates the semi-functional time-constrained update key as $TUK_{T,S_{i,j}} = \big(U_0 = U_0'' g_2^{\delta_{i,j}}, U_1 = U_1''\big)$.
- $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (h_c < i_c)$ or $(i_m \neq h_m^*) \wedge (h_m < i_m)$: It creates a semi-functional key by using the fixed $\delta_{i,j}$ for this member subset $S_{i,j}$.

Game $\mathbf{H}'_{(h_m,h_c),2}$. This game $\mathbf{H}'_{(h_m,h_c),2}$ is almost the same as $\mathbf{H}'_{(h_m,h_c),1}$ except that the personalized private key or time-constrained update key with the indexes $(i_g = h, i_m, i_c)$ such that $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$ is generated with $b = 0$. The modification is similar to the game $\mathbf{H}'_{(h_m,h_c),1}$. In the game $\mathbf{H}'_{(1,0),2}$, all personalized private keys and all time-constrained update keys with the group index h except the keys with the member index h_m^* are semi-functional where a fixed $\delta_{i,j}$ is used for each member.

Game $\mathbf{H}''$. This game $\mathbf{H}''$ is the same as $\mathbf{G}_{1,h}$. Compared to the game $\mathbf{H}'_{(1,0),2}$, the remaining personalized private keys and time-constrained update keys with the member index h_m^* are changed to be semi-functional by using a fixed $\delta_{i,j}$ for this member subset $S_{i,j}$.

Let $\mathbf{Adv}_{\mathcal{A}_I}^{H_i}$ be the advantage of $\mathcal{A}_I$ in a game $\mathbf{H}_i$. From the following Claims 4.11, 4.12, 4.13, 4.14, and 4.15, we can obtain the following equation

$$
\begin{aligned}
&\mathbf{Adv}_{\mathcal{A}_I}^{H_{(1,0),2}} - \mathbf{Adv}_{\mathcal{A}_I}^{H''} \\
&\leq \sum_{h_m=1}^{q_m} \sum_{h_c=1}^{q_c} \left|\mathbf{Adv}_{\mathcal{A}_I}^{H_{(h_m,h_c-1),2}} - \mathbf{Adv}_{\mathcal{A}_I}^{H_{(h_m,h_c),1}}\right| + \sum_{h_m=1}^{q_m} \sum_{h_c=1}^{q_c} \left|\mathbf{Adv}_{\mathcal{A}_I}^{H_{(h_m,h_c),1}} - \mathbf{Adv}_{\mathcal{A}_I}^{H_{(h_m,h_c),2}}\right| + \\
&\quad \sum_{h_m=1}^{q_m} \sum_{h_c=1}^{q_c} \left|\mathbf{Adv}_{\mathcal{A}_I}^{H'_{(h_m,h_c),2}} - \mathbf{Adv}_{\mathcal{A}_I}^{H'_{(h_m,h_c),1}}\right| + \sum_{h_m=1}^{q_m} \sum_{h_c=1}^{q_c} \left|\mathbf{Adv}_{\mathcal{A}_I}^{H'_{(h_m,h_c),1}} - \mathbf{Adv}_{\mathcal{A}_I}^{H'_{(h_m,h_c-1),2}}\right| + \\
&\quad \left|\mathbf{Adv}_{\mathcal{A}_I}^{H'_{(1,0),2}} - \mathbf{Adv}_{\mathcal{A}_I}^{H''}\right| \\
&\leq 5(q_{sk} + q_{uk}) \sum_{h_m=1}^{q_m} \sum_{h_c=1}^{q_c} \mathbf{Adv}_{\mathcal{B}}^{GSD}(\lambda).
\end{aligned}
$$

For the Type-II adversary $\mathcal{A}_{II}$, we define hybrid games $\mathbf{I}_{(1,1),1}, \mathbf{I}_{(1,1),2}, \ldots, \mathbf{I}_{(q_m,q_c),1}, \mathbf{I}_{(q_m,q_c),2} = \mathbf{I}'_{(q_m,q_c),2}, \mathbf{I}'_{(q_m,q_c),1}, \ldots, \mathbf{I}'_{(1,1),2}, \mathbf{I}'_{(1,1),1}, \mathbf{I}'_{(1,0),2}, \mathbf{I}''$ where $\mathbf{G}_{1,h-1} = \mathbf{I}_{(1,0),2}$ and $\mathbf{I}'' = \mathbf{G}_{1,h}$. The games are formally defined as follows:

Game $\mathbf{I}_{(h_m,h_c),1}$. This game $\mathbf{I}_{(h_m,h_c),1}$ is almost the same as $\mathbf{H}_{(h_m,h_c),1}$ except that there is no case $i_m = h_m^*$ since the adversary is Type-II.

Game $\mathbf{I}_{(h_m,h_c),2}$. This game $\mathbf{I}_{(h_m,h_c),2}$ is almost the same as $\mathbf{H}_{(h_m,h_c),2}$ except that there is no case $i_m = h_m^*$ since the adversary is Type-II.

Game $\mathbf{I}'_{(h_m,h_c),1}$. This game $\mathbf{I}'_{(h_m,h_c),1}$ is almost the same as $\mathbf{H}'_{(h_m,h_c),1}$ except that there is no case $i_m = h^*_m$ since the adversary is Type-II.

Game $\mathbf{I}'_{(h_m,h_c),2}$. This game $\mathbf{I}'_{(h_m,h_c),1}$ is almost the same as $\mathbf{H}'_{(h_m,h_c),1}$ except that there is no case $i_m = h^*_m$ since the adversary is Type-II. In the game $\mathbf{I}'_{(1,0),2}$, all personalized private keys and all time-constrained update keys with the group index h are semi-functional where a fixed $\delta_{i,j}$ is used for each member.

Let $\mathbf{Adv}^{I_i}_{\mathcal{A}_{II}}$ be the advantage of $\mathcal{A}_{II}$ in a game $\mathbf{I}_i$. From the following Claims 4.16, 4.17, 4.18, and 4.19, we can easily obtain the equation $\mathbf{Adv}^{I_{0,2}}_{\mathcal{A}_{II}} - \mathbf{Adv}^{I''}_{\mathcal{A}_{II}} \leq 5(q_{sk}+q_{uk})\mathbf{Adv}^{GSD}_{\mathcal{B}}(\lambda)$.

Let E_I, E_{II} be the event such that an adversary behave like the Type-I, Type-II adversary respectively. From the above three inequalities for three types, we have the following inequality as

$$\begin{aligned}
\mathbf{Adv}^{G_{1,h-1}}_{\mathcal{A}} - \mathbf{Adv}^{G_{1,h}}_{\mathcal{A}} &\leq \Pr[E_I](\mathbf{Adv}^{G_{1,h-1}}_{\mathcal{A}_I} - \mathbf{Adv}^{G_{1,h}}_{\mathcal{A}_I}) + \Pr[E_{II}](\mathbf{Adv}^{G_{1,h-1}}_{\mathcal{A}_{II}} - \mathbf{Adv}^{G_{1,h}}_{\mathcal{A}_{II}}) \\
&\leq \Pr[E_I](\mathbf{Adv}^{H_{(1,0),2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H''}_{\mathcal{A}_I}) + \Pr[E_{II}](\mathbf{Adv}^{I_{(1,0),2}}_{\mathcal{A}_{II}} - \mathbf{Adv}^{I''}_{\mathcal{A}_{II}}) \\
&\leq 5(q_{sk}+q_{uk}) \sum_{h_m=1}^{q_m} \sum_{h_c=1}^{q_c} \left(\Pr[E_I]\mathbf{Adv}^{GSD}_{\mathcal{B}}(\lambda) + \Pr[E_{II}]\mathbf{Adv}^{GSD}_{\mathcal{B}}(\lambda)\right) \\
&\leq 5(q_{sk}+q_{uk}) \sum_{h_m=1}^{q_m} \sum_{h_c=1}^{q_c} \mathbf{Adv}^{GSD}_{\mathcal{B}}(\lambda).
\end{aligned}$$

This completes our proof. □

Lemma 4.7. *If the GSD assumption holds, then no polynomial-time adversary can distinguish between $\mathbf{G}_2$ and $\mathbf{G}_3$ with a non-negligible advantage.*

Proof. Let q_{dk} be the number of decryption key queries of an adversary. For the security proof, we additionally define a sequence of games $\mathbf{G}_{2,1,1}, \mathbf{G}_{2,1,2}, \ldots, \mathbf{G}_{2,k,1}, \mathbf{G}_{2,k,2}, \ldots, \mathbf{G}_{2,q_{dk},1}, \mathbf{G}_{2,q_{dk},2}$ where $\mathbf{G}_2 = \mathbf{G}_{2,0,2}$. The games are defined as follows:

Game $\mathbf{G}_{2,k,1}$. In this game, all personalized private keys, all time-constrained update keys, and the challenge ciphertext are semi-functional. The first $k-1$ decryption keys are semi-functional. The kth decryption key is semi-functional of type 1 and it is generated as $DK_{ID,T} = (D_0 = D'_0 g_2^a, D_1 = D'_1 g_2^{-b_1}, D_2 = D'_2 g_2^{-b_2})$ where $DK'_{ID,T} = (D'_0, D'_1, D'_2)$ is a normal decryption key and a, b_1, b_2 are random exponents in $\mathbb{Z}_N$. The remaining decryption keys are normal.

Game $\mathbf{G}_{2,k,2}$. In this game, all personalized private keys, all time-constrained update keys, the challenge ciphertext header, and the first k decryption keys are semi-functional. But the remaining decryption keys are normal. It is obvious that $\mathbf{G}_{2,q_{dk},2} = \mathbf{G}_3$.

Note that if a semi-functional decryption key of type 1 is used to decrypt a semi-functional ciphertext, then the decryption fails since an additional random element $e(g_2,g_2)^{c(a-b_1d_1-b_2d_2)}$ is left. If $a = b_1d_1 + b_2d_2$, then the the decryption succeeds. In this case, we say that the decryption key is *nominally* semi-functional of type 1.

Let $\mathbf{Adv}^{G_{2,i,j}}_{\mathcal{A}}$ be the advantage of $\mathcal{A}$ in the game $\mathbf{G}_{2,i,j}$. We easily obtain that $\mathbf{Adv}^{G_2}_{\mathcal{A}} = \mathbf{Adv}^{G_{2,0,2}}_{\mathcal{A}}$ and $\mathbf{Adv}^{G_3}_{\mathcal{A}} = \mathbf{Adv}^{G_{2,q_{dk},2}}_{\mathcal{A}}$. From the following Claims 4.8 and 4.9, we can obtain the following equation

$$\begin{aligned}
\mathbf{Adv}^{G_2}_{\mathcal{A}} - \mathbf{Adv}^{G_3}_{\mathcal{A}} &\leq \sum_{k=1}^{q_{dk}} \left|\mathbf{Adv}^{G_{2,k-1,2}}_{\mathcal{A}} - \mathbf{Adv}^{G_{2,k,1}}_{\mathcal{A}}\right| + \sum_{k=1}^{q_{dk}} \left|\mathbf{Adv}^{G_{2,k,1}}_{\mathcal{A}} - \mathbf{Adv}^{G_{2,k,2}}_{\mathcal{A}}\right| \\
&\leq 2q_{dk}\mathbf{Adv}^{GSD}_{\mathcal{B}}(\lambda).
\end{aligned}$$

This completes our proof. $\square$

Claim 4.8. *If the GSD assumption holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_{2,k-1,2}$ *and* $\mathbf{G}_{2,k,1}$ *with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}_I$ that distinguishes between $\mathbf{G}_{2,k-1,2}$ and $\mathbf{G}_{2,k,1}$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the GSD assumption using $\mathcal{A}_I$ is given: a challenge tuple $D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3}, X_1R_1, R_2Y_1)$ and Z where $Z = Z_0 = X_2Y_2$ or $Z = Z_1 = X_2R_3Y_2$. Then $\mathcal{B}$ that interacts with $\mathcal{A}_I$ is described as follows:

Setup: $\mathcal{B}$ first chooses random exponents $u'_1, h'_1, u'_2, h'_2, \alpha \in \mathbb{Z}_N$. It sets $\mathcal{BT}$ by running **SD.Setup** and FL by selecting $f_{GL}(x)$ for each GL in $\mathcal{BT}$. It sets $MK = (\alpha, FL)$, $RL = \emptyset$, $ST = (\mathcal{BT}, UL = \emptyset)$, and publishes $PP = \big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_{p_1}, Y = g_{p_3}, u_1 = g_{p_1}^{u'_1}, h_1 = g_{p_1}^{h'_1}, u_2 = g_{p_1}^{u'_2}, h_2 = g_{p_1}^{h'_2}, \Omega = e(g,g)^{\alpha}\big)$.
Phase 1: For each query, $\mathcal{B}$ proceeds as follows: If this is a personalized private key (or time-constrained update key) query, then it creates a semi-functional one by using MK and R_2Y_1 given in the assumption. If this is a jth decryption key query for ID and T, then it handles this query as follows:

- If $j < k$, then it creates a semi-functional decryption key since it knows MK and R_2Y_1 is given in the assumption.
- If $j = k$, then it selects random $r'_1, r'_2 \in \mathbb{Z}_N$, $Y'_0, Y'_1, Y'_2 \in \mathbb{G}_{p_3}$ and creates a decryption key $DK_{ID,T} = \big(D_0 = g^{\alpha}(Z)^{(u'_1ID+h'_1)r'_1}(Z)^{(u'_2T+h'_2)r'_2}Y'_0,\ D_1 = (Z)^{-r'_1}Y'_1,\ D_2 = (Z)^{-r'_2}Y'_2\big)$.
- If $j > k$, then it creates a normal decryption key since it knows MK.

Challenge: $\mathcal{B}$ flips a random coin $\mu \in \{0,1\}$ and creates a semi-functional ciphertext by implicitly setting $g^s = X_1$ and $g_2^c = R_1$ as $CT^* = \big(C = e(X_1R_1, g)^{\alpha} \cdot M^*_{\mu},\ C_0 = X_1R_1,\ C_1 = (X_1R_1)^{u'_1ID^*+h'_1},\ C_2 = (X_1R_1)^{u'_2T^*+h'_2}\big)$.
Phase 2: Same as Phase 1.
Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu = \mu'$, then $\mathcal{B}$ outputs 1. Otherwise, it outputs 0.

If $Z = Z_0 = X_2Y_2$, then the simulation is the same as $\mathbf{G}_{2,k-1,2}$ since the kth decryption key and the semi-functional challenge ciphertext are correctly distributed by implicitly setting $r_1 \equiv \log_g(X_2)r'_1 \mod p_1$, $r_2 \equiv \log_g(X_2)r'_2 \mod p_1$, and $s \equiv \log_g(X_1) \mod p_1$. If $Z = Z_1 = X_2R_3Y_2$, then the simulation is almost the same as $\mathbf{G}_{2,k,1}$ except that the kth decryption key is generated as a nominally semi-functional one of type 1 by implicitly setting $b_1 \equiv \log_{g_2}(R_3)r'_1 \mod p_2$, $b_2 \equiv \log_{g_2}(R_3)r'_2 \mod p_2$, and $a \equiv b_1(u'_1ID + h'_1) + b_2(u'_2T + h'_2) \mod p_2$. To finish the proof, we should argue that the adversary cannot distinguish the nominally semi-functional decryption key from the semi-functional decryption key of type 1. To argue this, we use the restriction of the security model such that a decryption key query for an identity ID and a time T such that $(ID = ID^*) \wedge (T = T^*)$ is not allowed. Suppose there exists an unbounded adversary. Then the adversary can gather the values $a \equiv b_1(u'_1ID + h'_1) + b_2(u'_2T + h'_2), b_1, b_2 \mod p_2$ from the kth decryption key and $d_1 \equiv u'_1ID^* + h'_1, d_2 \equiv u'_2T^* + h'_2 \mod p_2$ from the challenge ciphertext. If $(ID \neq ID^*) \vee (T \neq T^*)$, then $b_1(u'_1ID + h'_1) + b_2(u'_2T + h'_2) \mod p_2$, $u'_1ID^* + h'_1 \mod p_2$ and $u'_2T^* + h'_2 \mod p_2$ look random to the adversary since $u'_ix + h'_i$ is a pair-wise independent function, $(ID \neq ID^*) \vee (T \neq T^*)$ by the restriction of the security model, and $u'_1, h'_1, u'_2, h'_2 \mod p_2$ are information theoretically hidden to the adversary. This completes our proof. $\square$

Claim 4.9. *If the GSD assumption holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_{2,k,1}$ *and* $\mathbf{G}_{2,k,2}$ *with a non-negligible advantage.*

Proof. The proof of this claim is almost the same as that of Claim 4.8, except the generation of the kth decryption key. The kth decryption key for ID and T is generated as follows:

- If $j = k$, then it selects random $r'_1, r'_2, a' \in \mathbb{Z}_N$, $Y'_0, Y'_1, Y'_2 \in \mathbb{G}_{p_3}$ and creates a decryption key $DK_{ID,T} = \big(D_0 = g^{\alpha}(Z)^{(u'_1 ID + h'_1)r'_1}(Z)^{(u'_2 T + h'_2)r'_2}(R_2Y_1)^{a'}Y'_0,\ D_1 = (Z)^{-r'_1}Y'_1,\ D_2 = (Z)^{-r'_2}Y'_2\big)$.

Note that the kth decryption key is no longer correlated with CT^* since the element D_0 is re-randomized by $(R_2Y_1)^{a'}$. If $Z = Z_0 = X_2Y_2$, then the simulation is the same as $\mathbf{G}_{2,k,2}$. If $Z = Z_1 = X_2R_3Y_2$, then the simulation is the same as $\mathbf{G}_{2,k,1}$. □

Lemma 4.10. *If the CompDH assumption holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_3$ *and* $\mathbf{G}_4$ *with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguish $\mathbf{G}_3$ from $\mathbf{G}_4$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the CompDH assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_2}, g_{p_3}, g_{p_1}^a R_1, g_{p_1}^b R_2)$ and Z where $Z = Z_0 = e(g_{p_1}, g_{p_1})^{ab}$ or $Z = Z_1 = e(g_{p_1}, g_{p_1})^c$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ chooses random exponents $u'_1, h'_1, u'_2, h'_2 \in \mathbb{Z}_N$ and implicitly sets $\alpha = a$ from the term $g_{p_1}^a R_1$. It sets $\mathcal{BT}$ by running **SD.Setup** and FL by selecting a random point (x, y) for each GL in BT. Note that a random $f_{GL}(x)$ for GL is implicitly defined by two points $(0, a)$ and (x, y) and $g_{p_1}^{f_{GL}(L_j)}R$ for any L_j can be computable from the Lagrange interpolation method where $R \in \mathbb{G}_{p_2}$. It sets $RL = \emptyset$, $ST = (\mathcal{BT}, UL = \emptyset)$ and publishes the public parameters $PP = \big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_{p_1}, Y = g_{p_3}, u_1 = g^{u'_1}, h_1 = g^{h'_1}, u_2 = g^{u'_2}, h_2 = g^{h'_2}, \Omega = e(g, g_{p_1}^a R_1)\big)$.

Phase 1: For each query, $\mathcal{B}$ creates a semi-functional key since $g_2^a R_1$ and g_2 are given from the assumption. Note that it cannot create a normal update key since it does not know $g_{p_1}^a \in \mathbb{G}_{p_1}$.

Challenge: $\mathcal{B}$ first flips a random coin $\mu \in \{0,1\}$ and creates a challenge ciphertext $CT^* = \big(C = Z \cdot M_\mu^*,\ C_0 = g_{p_1}^b R_2,\ C_1 = (g_{p_1}^b R_2)^{u'_1 ID^* + h'_1},\ C_2 = (g_{p_1}^b R_2)^{u'_2 T^* + h'_2}\big)$.

Phase 2: Same as Phase 1.

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

If $Z = Z_0$, then the simulation is the same as $\mathbf{G}_3$ since the challenge ciphertext is correctly distributed by implicitly setting $s = b$. If $Z = Z_1$, then the simulation is the same as $\mathbf{G}_4$ since the element C is random. □

4.4.1 Type-I Adversary

Claim 4.11. *If the GSD assumption holds, then no polynomial-time Type-I adversary can distinguish between* $\mathbf{H}_{(h_m, h_c - 1),2}$ *and* $\mathbf{H}_{(h_m, h_c),1}$ *with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}_I$ that distinguishes between $\mathbf{H}_{(h_m, h_c-1),2}$ and $\mathbf{H}_{(h_m,h_c),1}$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the GSD assumption using $\mathcal{A}_I$ is given: a challenge tuple $D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3}, X_1R_1, R_2Y_1)$ and Z where $Z = Z_0 = X_2Y_2$ or $Z = Z_1 = X_2R_3Y_2$. Then $\mathcal{B}$ that interacts with $\mathcal{A}_I$ is described as follows:

Setup: $\mathcal{B}$ selects random exponents $u'_1, h'_1, u'_2, h'_2, \alpha \in \mathbb{Z}_N$. It sets $\mathcal{BT}$ by running **SD.Setup** and FL by selecting $f_{GL}(x)$ for each GL in $\mathcal{BT}$. It sets $MK = (\alpha, FL)$, $RL = \emptyset$, $ST = (\mathcal{BT}, UL = \emptyset)$, and $PP = \big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_{p_1}, Y = g_{p_3}, u_1 = g^{u'_1}, h_1 = g^{h'_1}, u_2 = g^{u'_2}, h_2 = g^{h'_2}, \Omega = e(g,g)^{\alpha}\big)$.

Phase 1: Let h_m^* be a member index of the group index h such that the personalized private key for ID^* or the time-constrained update key for T^* belong to the member index h_m^* such that $1 \leq h_m^* \leq q_m$ where q_m is the maximum number of members in the group index h. As mentioned before, there is only one index h_m^* in the Type-I adversary. $\mathcal{B}$ selects a random index k such that $1 \le k \le q_m$ to guess h_m^*, and it can correctly guess h_m^* with the probability of $1/q_m$. Note that $q_m \le q_{sk} + q_{uk}$ since the private set of a private key is related with a path and the covering set of an update key is a partition where q_{sk} is the number of private key queries and q_{uk} is the number of update key queries of the adversary.

For each query, $\mathcal{B}$ proceeds as follows: If this is a decryption key query, then it creates a normal one since it knows MK. If this is a personalized private key or a time-constrained update key query with indexes (i_g, i_m, i_c), then it handles this query as follows:

- **Case** $i_g < h$: It first builds a normal key since it knows MK and converts it to a semi-functional one by using R_2Y_1 that is given in the assumption and selecting a random exponent $\delta'_{i,j}$ once for the subset $S_{i,j}$.
- **Case** $i_g = h$: It generates the key as follows:
  - $(i_m \ne h_m^*) \wedge (i_m < h_m)$ or $(i_m \ne h_m^*) \wedge (i_m = h_m) \wedge (i_c < h_c)$:
    If this is a personalized private key query, then it first builds a normal one and converts it to a semi-functional one of type 2 by selecting a new random exponent $a' \in \mathbb{Z}_N$ as $PSK_{ID,S_{i,j}} = \big(K_0 = g^{f_{GL}(L_j)}(u_1^{ID}h_1)^{r_1}Y'_0 \cdot (R_2Y_1)^{a'},\ K_1 = g^{-r_1}Y'_1\big)$.
    If this is a time-constrained update key query, then it first builds a normal one and converts it to a semi-functional one of type 2 by selecting a new random exponent $a' \in \mathbb{Z}_N$ as $TUK_{T,S_{i,j}} = \big(U_0 = g^{f_{GL}(L_j)}(u_2^{T}h_2)^{r_2}Y'_0 \cdot (R_2Y_1)^{a'},\ U_1 = g^{-r_2}Y'_1\big)$.
  - $(i_m \ne h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$:
    If this is a personalized private key query, then it chooses random elements $Y'_0, Y'_1 \in \mathbb{G}_{p_3}$ and creates a key as $PSK_{ID,S_{i,j}} = \big(K_0 = g^{f_{GL}(L_j)}(Z)^{u'_1 ID + h'_1}Y'_0,\ K_1 = Z^{-1}Y'_1\big)$.
    If this is a time-constrained update key query, then it chooses random elements $Y'_0, Y'_1 \in \mathbb{G}_{p_3}$ and creates a key as $TUK_{T,S_{i,j}} = \big(U_0 = g^{f_{GL}(L_j)}(Z)^{u'_2 T + h'_2}Y'_0,\ U_2 = Z^{-1}Y'_1\big)$.
  - $(i_m \ne h_m^*) \wedge (i_m = h_m) \wedge (h_c < i_c)$ or $(i_m \ne h_m^*) \wedge (h_m < i_m)$: It creates a normal key since it knows MK.
  - $(i_m = h_m^*)$: It creates a normal key since it knows MK.
- **Case** $i_g > h$: It creates a normal key since it knows MK.

Challenge: $\mathcal{B}$ flips a random coin $\mu \in \{0,1\}$ and creates a semi-functional ciphertext by implicitly setting $g^s = X_1$ and $g_2^c = R_1$ as $CT^* = \big(C = e(X_1R_1, g)^{\alpha} \cdot M_\mu^*,\ C_0 = X_1R_1,\ C_1 = (X_1R_1)^{u'_1 ID^* + h'_1},\ C_2 = (X_1R_1)^{u'_2 T^* + h'_2}\big)$.
Phase 2: Same as Phase 1.
Guess: $\mathcal{A}$ outputs a guess μ'. If $\mu = \mu'$, then $\mathcal{B}$ outputs 1. Otherwise, it outputs 0.

If $Z = Z_0 = X_2Y_2$, then the simulation is the same as $\mathbf{H}_{(h_m,h_c-1),2}$ since the personalized private key (or the time-constrained update key) with $(i_m \ne h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$ and the semi-functional challenge ciphertext are correctly distributed by implicitly setting $r_1 \equiv \log_g(X_2) \mod p_1$ (or $r_2 \equiv \log_g(X_2) \mod p_1$), and $s \equiv \log_g(X_1) \mod p_1$. If $Z = Z_1 = X_2R_3Y_2$, then the simulation is almost the same as $\mathbf{H}_{(h_m,h_c),1}$ except that the personalized private key (or the time-constrained update key) with $(i_m \ne h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$ is generated as a nominally semi-functional one of type 1 by implicitly setting $a \equiv \log_{g_2}(R_3)(u'_1 ID + h'_1) \mod p_2$ (or $a \equiv \log_{g_2}(R_3)(u'_2 T + h'_2) \mod p_2$) and $b \equiv \log_{g_2}(R_3) \mod p_2$.

To finish the proof, we should argue that the Type-I adversary cannot distinguish the nominally semi-functional one of type 1 from the semi-functional one of type 1. To argue this, we use the fact that we have

$ID \neq ID^*$ for all personalized private key queries for ID with indexes $(i_g = h, i_m, i_c)$ such that $i_m \neq h_m^*$, and $T \neq T^*$ for all time-constrained update key queries for T with indexes $(i_g = h, i_m, i_c)$ such that $i_m \neq h_m^*$. Suppose there exists an unbounded Type-I adversary. If the query with $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$ is a personalized private key, then the adversary can gather the values $a \equiv b(u_1' ID + h_1'), b \mod p_2$ from the personalized private key with $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$ and $d_1 \equiv u_1' ID^* + h_1' \mod p_2$, $d_2 \equiv u_2' T^* + h_2' \mod p_2$ from the challenge ciphertext. We obtain that $u_1' ID + h_1' \mod p_2$ and $u_1' ID^* + h_1' \mod p_2$ look random to the adversary since $u_1' x + h_1'$ is a pair-wise independent function, $ID \neq ID^*$ if $i_m \neq h_m^*$, and $u_1', h_1' \mod p_2$ are information theoretically hidden to the adversary. If the query with $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$ is a time-constrained update key, then we also obtain that $u_2' T + h_2' \mod p_2$ and $u_2' T^* + h_2'$ are random to the adversary since $u_2' x + h_2'$ pair-wise independent function, $T \neq T^*$ if $i_m \neq h_m^*$, and $u_2', h_2' \mod p_2$ are information theoretically hidden to the adversary. This completes our proof. □

Claim 4.12. *If the GSD assumption holds, then no polynomial-time Type-I adversary can distinguish between* $\boldsymbol{H}_{(h_m,h_c),1}$ *and* $\boldsymbol{H}_{(h_m,h_c),2}$ *with a non-negligible advantage.*

Proof. The proof of this claim is almost the same as that of Claim 4.11 except the generation of the key with indexes $i_g = h$ and $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$. This key with the group index h is generated as follows:

- $(i_m \neq h_m^*) \wedge (i_m = h_m) \wedge (i_c = h_c)$:

  If this is a personalized private key query, then it chooses random $a' \in \mathbb{Z}_N$, $Y_0', Y_1' \in \mathbb{G}_{p_3}$ and creates a key as $PSK_{ID,S_{i,j}} = \big(K_0 = g^{f_{GL}(L_j)} (Z)^{u_1' ID + h_1'} Y_0' (R_2 Y_1)^{a'},\ K_1 = Z^{-1} Y_1'\big)$.

  If this is a time-constrained update key query, then it chooses random $a' \in \mathbb{Z}_N$, $Y_0', Y_1' \in \mathbb{G}_{p_3}$ and creates a key as $TUK_{T,S_{i,j}} = \big(U_0 = g^{f_{GL}(L_j)} (Z)^{u_2' T + h_2'} Y_0' (R_2 Y_1)^{a'},\ K_1 = Z^{-1} Y_1'\big)$.

Note that this personalized private key or time-constrained update key is no longer correlated with CT^* since the element K_0 is re-randomized by $(R_2 Y_1)^{a'}$. □

Claim 4.13. *If the GSD assumption holds, then no polynomial-time Type-I adversary can distinguish between* $\boldsymbol{H}'_{(h_m,h_c-1),2}$ *and* $\boldsymbol{H}'_{(h_m,h_c),1}$ *with a non-negligible advantage.*

Claim 4.14. *If the GSD assumption holds, then no polynomial-time Type-I adversary can distinguish between* $\boldsymbol{H}'_{(h_m,h_c),1}$ *and* $\boldsymbol{H}'_{(h_m,h_c),2}$ *with a non-negligible advantage.*

The proofs of Claim 4.13 and Claim 4.14 are almost the same as that of Claim 4.11 and Claim 4.12 respectively. The only difference is that each element K_0 of personalized private keys and each element U_0 of time-constrained update keys with indexes $(i_g = h, i_m, i_c)$ such that $i_m \neq h_m^*$ that are generated in Claim 4.11 and Claim 4.12 respectively are additionally multiplied by $(R_2 Y_1)^{\delta'_{i,j}}$ where $\delta'_{i,j}$ is a fixed exponent that is related with the member subset $S_{i,j}$. This modification is possible since $R_2 Y_1$ is given in the assumption. We omit the detailed proofs of these claims.

Claim 4.15. *If the GSD assumption holds, then no polynomial-time Type-I adversary can distinguish between* $\boldsymbol{H}'_{(1,0),2}$ *and* $\boldsymbol{H}''$ *with a non-negligible advantage.*

Proof. The proof of this claim is almost the same as that of Claim 4.11 except that the generation of personalized private keys and time-constrained update keys with the group index $i_g = h$. These keys with the group index $i_g = h$ are generated as follows:

- **Case** $i_g = h$: Let $\delta'_{i,j}$ be a fixed exponent in $\mathbb{Z}_N$ for each member $S_{i,j}$ in this group index h.

– $(i_m \neq h_m^*)$:
If this is a personalized private key query, then it selects random $r_1 \in \mathbb{Z}_N$, $Y_0', Y_1' \in \mathbb{G}_{p_3}$ and creates a semi-functional key as $PSK_{ID,S_{i,j}} = \big(K_0 = (Z)^{L_j} g^{\alpha} (u_1^{ID} h_1)^{r_1} Y_0' \cdot (R_2 Y_1)^{\delta_{i,j}'},\ K_1 = g^{-r_1} Y_1'\big)$.
If this is a time-constrained update key query , then it selects random $r_2 \in \mathbb{Z}_N$, $Y_0', Y_1' \in \mathbb{G}_{p_3}$ and creates a semi-functional key as $TUK_{T,S_{i,j}} = \big(U_0 = (Z)^{L_j} g^{\alpha} (u_2^{T} h_2)^{r_2} Y_0' \cdot (R_2 Y_1)^{\delta_{i,j}'},\ U_1 = g^{-r_2} Y_1'\big)$.

– $(i_m = h_m^*)$:
If this is a personalized private key query, then it selects random $r_1 \in \mathbb{Z}_N$, $Y_0', Y_1' \in \mathbb{G}_{p_3}$ and creates a key as $PSK_{ID,S_{i,j}} = \big(K_0 = (Z)^{L_j} g^{\alpha} (u_1^{ID} h_1)^{r_1} Y_0',\ K_1 = g^{-r_1} Y_1'\big)$.
If this is a time-constrained update key query, then it selects random $r_2 \in \mathbb{Z}_N$, $Y_0', Y_1' \in \mathbb{G}_{p_3}$ and creates a key as $TUK_{T,S_{i,j}} = \big(U_0 = (Z)^{L_j} g^{\alpha} (u_2^{T} h_2)^{r_2} Y_0',\ U_1 = g^{-r_2} Y_1'\big)$.

If $Z = Z_0 = X_2 Y_2$, then the simulation is the same as $\mathbf{H}'_{(1,0),2}$ since all personalized private keys and time-constrained update keys with the group index h implicitly uses a random polynomial $f_{GL}(x) \equiv \log_g(X_2) \cdot x + \alpha \mod p_1$ and it implicitly sets $\delta_{i,j} \equiv \log_{g_{p_2}}(R_2)\delta_{i,j}' \mod p_2$ for each member index $i_m \neq h_m^*$. If $Z = Z_1 = X_2 R_3 Y_2$, then the simulation is the same as $\mathbf{H}''$ since it implicitly sets $\delta_{i,j} = \log_{g_{p_2}}(R_3) L_j \mod p_2$ for the member index h_m^*. As mentioned, the personalized private key query for ID^* and the time-constrained update key query for T^* should belong to the member index h_m^*. This completes our proof. $\square$

4.4.2 Type-II Adversary

Claim 4.16. *If the GSD assumption holds, then no polynomial-time Type-II adversary can distinguish between $\boldsymbol{I}_{(h_m,h_c-1),2}$ and $\boldsymbol{I}_{(h_m,h_c),1}$ with a non-negligible advantage.*

Claim 4.17. *If the GSD assumption holds, then no polynomial-time Type-II adversary can distinguish between $\boldsymbol{I}_{(h_m,h_c),1}$ and $\boldsymbol{I}_{(h_m,h_c),2}$ with a non-negligible advantage.*

Claim 4.18. *If the GSD assumption holds, then no polynomial-time Type-II adversary can distinguish between $\boldsymbol{I}'_{h_c-1,2}$ and $\boldsymbol{I}'_{h_c,1}$ with a non-negligible advantage.*

Claim 4.19. *If the GSD assumption holds, then no polynomial-time Type-II adversary can distinguish between $\boldsymbol{I}'_{h_c,1}$ and $\boldsymbol{I}'_{h_c,2}$ with a non-negligible advantage.*

The proofs of Claim 4.16, 4.17, 4.18, and 4.19 are almost the same as those of Claim 4.11, 4.12, 4.13, and 4.14 respectively except that there is no case $i_m = h_m^*$ since the Type-II adversary does not request a personalized private key for ID^* and a time-constrained update key for T^*. We omit the detailed proofs of these claims.

5 Conclusion

In this work, we presented a new technique for RIBE that can use the efficient SD scheme (or the LSD scheme) instead of the CS scheme for key revocation. By following our technique, we first proposed a new RIBE scheme in bilinear groups by combining the IBE scheme of Boneh and Boyen [5] and the SD scheme, and then we proved it security in the selective revocation list model. We also proposed another RIBE scheme in bilinear groups by combining the IBE scheme in composite-order bilinear groups of Lewko and Waters [24] and the SD scheme, and proved its full security under static assumptions. Our constructions also can be integrated with the LSD scheme to reduce the size of private keys.

An interesting open problem is to build efficient R-ABE, RS-ABE, and RS-PE schemes that provide the revocation functionality for ABE and PE by using the SD scheme. One may expect that our technique in this work can be used to achieve these schemes, but there is one crucial difficulty to prove the security of the schemes since our proof techniques in the selective revocation list model (or the full model) only work when there is only one private key (or one update key) that matches to a challenge ciphertext. However, there are many private key queries in ABE that can decrypt a challenge ciphertext. Thus, we expect that a new different technique will be needed to build R-ABE, RS-ABE, and RS-PE schemes that use the SD scheme.

References

[1] William Aiello, Sachin Lodha, and Rafail Ostrovsky. Fast digital identity revocation (extended abstract). In Hugo Krawczyk, editor, *CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 137–152. Springer, 1998.

[2] Nuttapong Attrapadung and Hideki Imai. Conjunctive broadcast and attribute-based encryption. In Hovav Shacham and Brent Waters, editors, *Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*, pages 248–265. Springer, 2009.

[3] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[4] 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.

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

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

[7] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *Lecture Notes in Computer Science*, pages 253–273. Springer, 2011.

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

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

[10] Amos Fiat and Moni Naor. Broadcast encryption. In Douglas R. Stinson, editor, *CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 480–491. Springer, 1993.

[11] Craig Gentry. Certificate-based encryption and the certificate revocation problem. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 272–293. Springer, 2003.

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

[13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 2012.

[14] 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.

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

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

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

[18] 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, *ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 235–254. Springer, 2013.

[19] Kwangsu Lee, Intae Kim, and Seong Oun Hwang. Privacy preserving revocable predicate encryption revisited. Cryptology ePrint Archive, Report 2012/655, 2012. `http://eprint.iacr.org/2012/655`.

[20] Kwangsu Lee, Woo Kwon Koo, Dong Hoon Lee, and Jong Hwan Park. Public-key revocation and tracing schemes with subset difference methods revisited. Cryptology ePrint Archive, Report 2013/228, 2013. `http://eprint.iacr.org/2013/228`.

[21] Kwangsu Lee and Dong Hoon Lee. Improved hidden vector encryption with short ciphertexts and tokens. *Designs Codes Cryptogr.*, 58(3):297–319, 2011.

[22] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

[23] Allison B. Lewko, Amit Sahai, and Brent Waters. Revocation systems with very small private keys. In *IEEE Symposium on Security and Privacy*, pages 273–285. IEEE Computer Society, 2010.

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

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

[26] Thomas Martin, Keith M. Martin, and Peter R. Wild. Establishing the broadcast efficiency of the subset difference revocation scheme. *Designs Codes Cryptogr.*, 51(3):315–334, 2009.

[27] Silvio Micali. Efficient certificate revocation. Technical Report TM-542b, MIT Laboratory for Computer Science, 1996.

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

[29] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000.

[30] Moni Naor and Benny Pinkas. Efficient trace and revoke schemes. In Yair Frankel, editor, *FC 2000*, volume 1962 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2000.

[31] Juan Manuel González Nieto, Mark Manulis, and Dongdong Sun. Fully private revocable predicate encryption. In Willy Susilo, Yi Mu, and Jennifer Seberry, editors, *ACISP 2012*, volume 7372 of *Lecture Notes in Computer Science*, pages 350–363. Springer, 2012.

[32] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 591–608. Springer, 2012.

[33] Seunghwan Park, Kwangsu Lee, and Dong Hoon Lee. New constructions of revocable identity-based encryption from multilinear maps. Cryptology ePrint Archive, Report 2013/880, 2013. `http://eprint.iacr.org/2013/880`.

[34] Amit Sahai, Hakan Seyalioglu, and Brent Waters. Dynamic credentials and ciphertext delegation for attribute-based encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 199–217. Springer, 2012.

[35] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 457–473. Springer, 2005.

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

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

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

[39] Le Su, Hoon Wei Lim, San Ling, and Huaxiong Wang. Revocable ibe systems with almost constant-size key update. In Zhenfu Cao and Fangguo Zhang, editors, *Pairing 2013*, volume 8365 of *Lecture Notes in Computer Science*, pages 168–185. Springer, 2013.

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

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