Delegate and Verify the Update Keys of Revocable Identity-Based Encryption

Kwangsu Lee*

Abstract

Revocable identity-based encryption (RIBE) is an extension of identity-based encryption (IBE) and it supports efficient revocation of private keys. In the past, many efficient RIBE schemes have been proposed, but research on efficiently delegating the generation of update keys to a cloud server is somewhat insufficient. In this paper, we newly introduce the concept of delegated RIBE (DRIBE) that can delegate the generation of update keys to the cloud server and define the security models of DRIBE. Next, we propose a DRIBE scheme by generically combining a hierarchical IBE (HIBE) scheme, an identity-based broadcast encryption (IBBE) scheme, and a collision-resistant hash function. In addition, we propose a DRIBE-INC scheme that generates an occasional base update key and a periodic incremental update key to reduce the size of the update key in our DRIBE scheme.

Keywords: Identity-based encryption, Key revocation, Update key delegation, Public Verifiability.

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

Revocable identity-based encryption (RIBE) is an extension of identity-based encryption (IBE) in which an identity string plays the role of a public key and additionally it supports the key revocation functionality [3][6]. In a public key encryption (PKE) scheme, a credential of a user can be revoked by using a certificate revocation mechanism since there is a certificate that binds the public key of a user with the identity of the user. However, it is difficult to provide the key revocation for an IBE scheme because the IBE scheme does not have a certificate. In the IBE scheme, there are two methods of revoking the credential of a user: a direct revocation method in which a sender specifies a receiver set in a ciphertext, and an indirect revocation method in which a trusted center periodically issues new (updated) keys for non-revoked users. In this paper, we consider IBE schemes that support the indirect revocation method because the sender does not need to care about revoked users when generating ciphertexts.

The first revocation method for IBE was presented by Boneh and Franklin [6], but their method has the disadvantage of requiring a secure channel between a user and the trusted center. An RIBE scheme that efficiently processes the key revocation of a user’s private key by using a binary tree was first proposed by Boldyreva et al. [3]. The main idea of their RIBE scheme is that a private key is associated with the path nodes of the binary tree and an update key is associated with the cover nodes of the binary tree that excludes leaf nodes that are related to revoked private keys. In this case, a common node exists between the path nodes and the cover nodes unless the private key is revoked in the update key. If the master key of the RIBE scheme is separated for each tree node by using a secret sharing scheme, then a decryption key can be derived by recovering the shared secret for the corresponding common node and this decryption key can be used to decrypt a ciphertext. After the first construction of an efficient RIBE scheme, various RIBE schemes that have improved the security or efficiency of the previous RIBE scheme have been proposed [21,24,27,31,34]. The key revocation method of RIBE also can be applied to an HIBE scheme that supports private key delegation, so various revocable HIBE (RHIBE) schemes have been proposed [19,22,29,30,32].

As described above, most of the RIBE schemes require a trusted center to generate a user’s private key and periodically issue an update key for non-revoked users. In order to reduce the load on the trusted center, we may consider to delegate the generation of update keys to a cloud server. However, the previous RIBE schemes using a binary tree require the same master key for the generation of private keys and update keys because the private key generation and the update key generation use the same state information of the binary tree. For this reason, if the generation of update keys is simply delegated to the cloud server, then it is possible for the cloud server to easily derive the user’s private key from the state information. In order to support secure delegation of update keys, an RIBE scheme that does not use a binary tree has been proposed, but this scheme has the disadvantage such that the update key size is linearly dependent on the number of non-revoked users in the system [23]. In this paper, we ask whether it is possible to design an RIBE scheme that can delegate the generation of update keys to a cloud server while supporting efficient key revocation by using a binary tree.

1.1 Out Contributions

Definition of DRIBE. We first define delegated RIBE (DRIBE) that can delegate the generation of an update key in RIBE to a cloud server. In DRIBE, a trusted center only generates private keys by using a master secret key and manages a revocation list RL. For the generation of update keys, a cloud server periodically issues an update key for the revocation list by using a master update key received from the trusted center, and then it broadcasts the update key to non-revoked users. At this time, anyone can publicly verify that the update key generated by the cloud server is valid on the revocation list RL. Similar to the security of RIBE, the DRIBE

2
scheme also provides indistinguishability under chosen-plaintext attack (IND-CPA) security in which an external attacker cannot distinguish between challenge messages. We also define indistinguishability under update-key attack (IND-UKA) security in which the cloud server also cannot distinguish between challenge messages if the private key for a challenge identity is not given. In addition, we define the update-key verifiability (UKV) security in order to prevent the cloud server which has the master update key from issuing a malicious update key that is helpful to external attackers. That is, the UKV security guarantees that the final update key generated by the cloud server is difficult to pass the verification algorithm for two different revocation lists $RL_0$ and $RL_1$.

Delegated RIBE. Next, we construct a DRIBE scheme that supports the delegation of update keys by generically combining an HIBE scheme, an IBBE scheme, and a hash function, and then we prove the security of our construction. The reason why it is difficult to support the delegation of update keys in the previous RIBE schemes that use a binary tree is that the same state information must be shared for the generation of private keys and update keys. To support the delegation of update keys, we pay attention to the RIBE scheme of Ma and Lin (ML-RIBE) [25] that additionally provides the decryption key exposure resistance (DKER) security. The ML-RIBE scheme combines an IBE scheme and an HIBE scheme, and the generation of private keys and the generation of update keys are separated from each other because the HIBE master key is used to generate private keys and the IBE master key is used to generate update keys. However, the ML-RIBE scheme has a limitation in that it cannot provide the UKV security. In order to provide the UKV security, we additionally employ a collision-resistant hash function to prevent malicious generation of update keys by relying on the hardness of finding collisions in the hash function. The DRIBE scheme proposed in this paper combines an HIBE scheme, an IBBE scheme, and a hash function, so that the ciphertext size and the secret key size are compact, and the update key size is proportional to the product of the number of revoked users and the security parameter.

DRIBE with Incremental Updates. The drawback of our DRIBE scheme is that the update key size is somewhat large because the depth of a binary tree is set to be proportional to the length of an identity string. To overcome this shortcoming, we propose a DRIBE-INC scheme that supports the generation of incremental update keys. The concept of incremental revocation was already used in the form of delta certificate revocation lists (delta-CRLs) when issuing CRLs for certificate revocation in the public-key infrastructure [12]. The DRIBE-INC scheme separates the update key into two types, a base update key and an incremental update key. For this, a time step $T_i$ to generate the base update key is fixed and a time period $T$ is expressed as $T = T_b + T_c$ where $T ≜ T_c \mod T_b$. At this time, if $T$ is a multiple of $T_b$ ($T = T_b$), a base update key that considers non-revoked users before $T_b$ is issued. Otherwise, an incremental update key that considers non-revoked users from $T_b$ to $T$ is issued. When a ciphertext with an identity $ID$ and time $T$ is given, we can decrypt the ciphertext by combining a private key with base and incremental update keys if $ID$ is not revoked both the base update key at time $T_b$ and the incremental update keys at time $T$. In general, the size of an incremental update key is relatively small since the number of revoked users during the short period of time is rather small compared to the number of revoked users in the base update key. To design the DRIBE-INC scheme, we use the DRIBE scheme to process base update keys, and an HIBE scheme, an IBBE scheme, and hash function to process incremental update keys.

1.2 Related Work

IBE is an extension of public-key encryption (PKE) that can use the user’s identity string as a public key [33]. The first IBE scheme was proposed by Boneh and Franklin [6], and they constructed their IBE scheme by using a bilinear map and proved its security in the random oracle model. Since then, various IBE schemes
have been proposed in bilinear maps, quadratic residues, and lattices [4, 11, 14]. In order to use an IBE scheme in a real application environment, an RIBE scheme that provides the functionality of effectively revoking a user’s private key is required. An RIBE scheme that provides efficient key revocation using a binary tree was first proposed by Boldyreva et al. [3]. Since then, various RIBE schemes have been proposed to enhance the security or improve the performance [10, 21, 24, 31]. Recently, generic methods for designing RIBE schemes using binary trees has been proposed [20, 25]. A key principle that enables the generic RIBE design is to make the path of a binary tree associated with a ciphertext instead of associated with a private key. However, since the generic RIBE design requires a larger binary tree compared to the existing direct RIBE design method, there is a problem of inefficiency in terms of update key size. In order to reduce the computational load of a key generation center in RIBE, an RIBE scheme that delegates the generation of update keys to a cloud server was proposed, but the proposed RIBE scheme has a problem that the size of an update key increases in proportion to the number of users [23].

HIBE is an extension of IBE that expresses the identity of a user as a hierarchical identity vector string and provides the functionality of delegating the private key generation to reduce the computational burden of a key generation center [15, 17]. After the first constructions of HIBE, various HIBE schemes have been proposed in bilinear maps and lattices [2, 5, 9, 35]. The first RHIBE scheme that provides the revocation of private keys was proposed by Seo and Emura [30]. In order to design the first RHIBE scheme, they devised a method similar to the RIBE scheme to support the revocation of private keys by using a binary tree for each individual user. After that, Seo and Emura also proposed a more efficient RHIBE scheme in which the private key of an individual user do not need to remember the history of private key delegation [32]. Lee and Park proposed a new RHIBE scheme that surprisingly reduced the private key size and update key size of the existing RHIBE scheme [22]. They devised a new type of HIBE scheme to improve the performance so that the private key and update key size of their RHIBE scheme are not significantly affected by the depth of an identity string. In addition, other RHIBE schemes with improved security and new functionalities have been proposed [18, 19, 29].

2 Preliminaries

In this section, we review the definitions of an HIBE scheme, an IBBE scheme, a hash function, and a complete subtree (CS) scheme in a binary tree, which are the building blocks of our DRIBE scheme.

2.1 Hierarchical Identity-Based Encryption

Hierarchical IBE (HIBE) is an extension of IBE, which expresses the user identity as a hierarchical identity vector and supports the delegation of the generation of private keys [15, 17]. In HIBE, a trusted center generates the private key of a user by using a master key, and an upper-level user who owns a private key can issue the private key of a lower-level user. A sender creates a ciphertext by specifying the hierarchical identity of a receiver. If the hierarchical identity of the receiver’s private key belongs to the prefix set of the hierarchical identity in the ciphertext, the receiver can decrypt the ciphertext. A more detailed syntax of HIBE is given as follows.

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

- **Setup**(1^λ, Lmax). The setup algorithm takes as input a security parameter 1^λ and maximum hierarchical depth Lmax. It outputs a master key MK and public parameters PP.
The correctness of HIBE is defined as follows: For all $\text{Setup}(\lambda, k)$, the master key $MK$, and the public parameters $PP$. It outputs a private key $SK_{ID_k}$ for $ID_k$.

$\text{DelegationKey}(ID_k, SK_{ID_{k-1}}, PP)$. The delegation algorithm takes as input a hierarchical identity $ID_k$, a private key $SK_{ID_{k-1}}$ for $ID_{k-1}$, and the public parameters $PP$. It outputs a delegated private key $SK_{ID_k}$ for $ID_k$.

$\text{Encrypt}(ID_\ell, M, PP)$. The encryption algorithm takes as input a hierarchical identity $ID_\ell = (I_1, \ldots, I_\ell) \in \mathcal{I}_\ell$, a message $M$, and public parameters $PP$. It outputs a ciphertext $CT_{ID_\ell}$.

$\text{Decrypt}(CT_{ID_\ell}, SK_{ID_k}, PP)$. The decryption algorithm takes as input a ciphertext $CT_{ID_\ell}$, a private key $SK_{ID_k}$, and public parameters $PP$. It outputs a message $M$ or $\perp$.

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

- If $ID'_k \in \text{Prefix}(ID_\ell)$, then $\text{Decrypt}(\text{Encrypt}(ID_\ell, M, PP), SK_{ID'_k}, PP) = M$.

The security model of HIBE is similar to that of IBE except that it additionally considers the delegation of private keys [15]. In the security model, public parameters are given to an attacker, and the attacker can query a private key for an hierarchical identity. In the challenge phase, the attacker submits challenge $ID^*, M^*_0$, and $M^*_1$ and receives a challenge ciphertext. At this time, there is a constraint that the hierarchical identities of private keys requested by the attacker do not belong to the prefix set of the challenge $ID^*$. After that, the attacker can query additional private keys and finally submits a guess for the challenge ciphertext. The attacker’s advantage is the value subtracting 1/2 from the probability of guessing the message. An HIBE scheme is secure if the advantage of all efficient attackers is negligible.

### 2.2 Identity-Based Broadcast Encryption

Identity-based broadcast encryption (IBBE) is an extension of public-key broadcast encryption (PKBE) except that individual users are specified as identity strings rather than indexes and it supports exponential numbers of system users [13]. In IBBE, a trusted center generates a private key for a user’s identity. A sender creates a ciphertext by specifying a set of receiver’s identities. A receiver can decrypt the ciphertext if his/her identity of the private key belongs to the receiver’s set in the ciphertext. A more detailed syntax of IBBE is given as follows.

**Definition 2.2 (Identity-Based Broadcast Encryption, IBBE).** An IBBE scheme consists of four algorithms $\text{Setup}$, $\text{GenKey}$, $\text{Encrypt}$, and $\text{Decrypt}$, which are defined as follows:

- $\text{Setup}(\lambda, k)$: The setup algorithm takes as input a security parameter $\lambda$. It outputs a master key $MK$ and public parameters $PP$.
- $\text{GenKey}(ID, MK, PP)$: The private key generation algorithm takes as input an identity $ID \in \mathcal{I}$, the master key $MK$, and public parameters $PP$. It outputs a private key $SK_{ID}$.
- $\text{Encrypt}(S, M, PP)$: The encryption algorithm takes as input a set $S \subseteq \mathcal{I}$ of receivers, a message $M \in \mathcal{M}$, and public parameters $PP$. It outputs a ciphertext $CT_S$.
- $\text{Decrypt}(CT_S, SK_{ID}, PP)$: The decryption algorithm takes as input a ciphertext $CT_S$, a private key $SK_{ID}$, and public parameters $PP$. It outputs a message $M$ or $\perp$. 

5
The correctness of IBBE is defined as follows: For all MK and PP generated by Setup(1^λ), SK_ID generated by GenKey(ID, MK, PP) for any ID, and any S and M, it is required that

- If ID ∈ S, then Decrypt(Encrypt(S, M, PP), SK_ID, PP) = M.

The security model of IBBE is similar to that of PKBE except that it additionally supports the private key query of an attacker for an identity string [13]. In the security model, an attacker is given public parameters, and the attacker can query a private key for an arbitrary identity string. In the challenge phase, the attacker submits a challenge set S′, challenge messages M0, M1, and receives a challenge ciphertext. At this time, there is a constraint that the private keys requested by the attacker are not belong to S′. After that, the attacker can query additional private keys with the same constraints and finally outputs a guess for the challenge ciphertext. At this time, the advantage of the attacker is defined as the value subtracting 1 from the probability of guessing the message. An IBBE scheme is secure if the advantage of all efficient attackers is negligible.

### 2.3 Collision Resistant Hash Function

A hash function family (HF) \( \mathcal{H} \) is a set of hash functions \( H : \mathcal{X} \rightarrow \mathcal{Y} \) where \( \mathcal{X} \) is an input domain and \( \mathcal{Y} \) is an output range. We say that the hash function family \( \mathcal{H} \) is collision resistant if for all efficient adversaries \( \mathcal{A} \), the advantage of \( \mathcal{A} \) defined as \( \text{Adv}_{HF, \mathcal{A}}(\lambda) = \Pr[H(x) = H(x') : H \leftarrow \mathcal{H}, (x, x') \leftarrow \mathcal{A}(H)] \) is negligible in the security parameter \( \lambda \).

### 2.4 Binary Tree

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

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

A subtree \( \mathcal{T}_i \) in \( \mathcal{B}T \) is defined as a tree that is rooted at a node \( v_i \) ∈ \( \mathcal{B}T \). A subset \( S_i \) is defined as a set of all leaf nodes in \( \mathcal{T}_i \). For any two nodes \( v_i, v_j \) ∈ \( \mathcal{B}T \) where \( v_j \) is a descendant of \( v_i \), \( \mathcal{T}_{i,j} \) is defined as a subtree \( \mathcal{T}_i - \mathcal{T}_j \), that is, all nodes that are descendants of \( v_i \) but not \( v_j \). A subset \( S_{i,j} \) is defined as a set of leaf nodes in \( \mathcal{T}_{i,j} \), that is, \( S_{i,j} = S_i \setminus S_j \).

For a perfect binary tree \( \mathcal{B}T \) and a subset \( \mathcal{R}V \) of leaf nodes, \( ST(\mathcal{B}T, \mathcal{R}V) \) is defined as the Steiner Tree induced by the set \( \mathcal{R}V \) and the root node, that is, the minimal subtree of \( \mathcal{B}T \) that connects all the leaf nodes in \( \mathcal{R}V \) and the root node.

### 2.5 Complete Subtree Method

The complete subtree method (CS) is one of the subset cover methods which are used to design symmetric-key broadcast encryption schemes using binary trees [26]. In CS, the assignment algorithm outputs the path nodes of a leaf node when the leaf node is given as input, and the cover algorithm outputs a set of cover
nodes that covers all non-revoked leaf nodes when a set of revoked leaf nodes are given as input. The final
matching algorithm outputs one node that is common to each other in the path nodes and the cover nodes.
A more detailed syntax of CS is given as follows.

\[
\text{CS.Setup}(N_{\text{max}}): \; \text{Let } N_{\text{max}} = 2^n \text{ for simplicity. It first sets a perfect binary tree } BT \text{ of depth } n. \text{ Each user is assigned to a different leaf node in } BT. \text{ The collection } S \text{ is defined as } \{S_i\} \text{ where } S_i \text{ is a set of all leaves in a subtree } T_i \text{ with a subroot } v_i \in BT. \text{ It outputs the binary tree } BT.
\]

\[
\text{CS.Assign}(BT, vID): \; \text{Let } vID \text{ be a leaf node of } BT \text{ that is assigned to the user } ID. \text{ Let } (v_{k_0}, v_{k_1}, \ldots, v_{k_n}) \text{ be the path from the root node } v_{k_0} = v_0 \text{ to the leaf node } v_{k_n} = v_{ID}. \text{ For all } j \in \{k_0, \ldots, k_n\}, \text{ it adds } S_j \text{ into } PV. \text{ It outputs the private set } PV = \{S_j\}.
\]

\[
\text{CS.Cover}(BT, RV): \; \text{It first computes the Steiner tree } ST(RV). \text{ Let } T_{k_1}, \ldots, T_{k_m} \text{ be all the subtrees of } BT \text{ that hang off } ST(RV), \text{ that is all subtrees whose roots } v_{k_1}, \ldots, v_{k_m} \text{ are not in } ST(RV) \text{ but adjacent to nodes of outdegree 1 in } ST(RV). \text{ For all } i \in \{k_1, \ldots, k_m\}, \text{ it adds } S_i \text{ into } CV. \text{ It outputs a covering set } CV = \{S_i\}.
\]

\[
\text{CS.Match}(CV, PV): \; \text{It finds a subset } S_k \text{ with } S_k \subseteq CV \text{ and } S_k \subseteq PV. \text{ If there is such a subset, it outputs } (S_k, S_k). \text{ Otherwise, it outputs } \bot.
\]

The correctness of the CS method is that if a leaf node is not revoked in the cover nodes, the matching algorithm outputs one common node. In the CS method, the size of the cover nodes is approximately \( r \log(N/r) \) when the number of revoked users is \( r \) and the total number of binary tree leaf nodes is \( N \).

3 Revocable IBE with Delegated Update Keys

In this section, we define the syntax and security model of DRIBE and show that a DRIBE scheme can be generically constructed by combining existing HIBE and IBBE schemes and a hash function.

3.1 Definition

DRIBE is an extension of RIBE that supports user revocation and it can additionally delegate issuance of update keys to a cloud server. In addition, anyone can publically verify the update key of DRIBE.

In a DRIBE scheme, a trusted center generates two master keys \( MSK, MUK \), and public parameters \( PP \), and sends the master update key \( MUK \) to a cloud server to delegate the generation of update keys. The trusted center generates the private key of each user by using the master secret key \( MSK \) and maintains a revocation list \( RL \) that records the information of revoked users. The cloud server periodically broadcasts an update key for non-revoked users by using the master update key \( MUK \) and the revocation list \( RL \). If a user wants to deliver a message securely to another user, the sender creates a ciphertext by specifying a receiver identity \( ID \) and time \( T \). If the private key of the receiver is not revoked in the update key at the corresponding time, the receiver can decrypt the ciphertext by combining his private key and the update key. In addition, any user can publically verify the validity of the update key generated by the cloud server.

**Definition 3.1** (Delegated Revocable IBE). A delegated revocable IBE (DRIBE) scheme that is associated with identity space \( \mathcal{I} \), time space \( \mathcal{T} \), and message space \( \mathcal{M} \), consists of eight algorithms \( \text{Setup}, \text{GenKey}, \text{Revoke}, \text{UpdateKey}, \text{VerifyUK}, \text{DeriveKey}, \text{Encrypt}, \text{and Decrypt} \), which are defined as follows:
The correctness of DRIBE is defined as follows: For all $M$, $MUK$, and public parameters $PP$. It outputs a master secret key $MSK$. A more detailed definition of the IND-CPA security model is described as follows.

**Setup**($1^λ$): The setup algorithm takes as input a security parameter $1^λ$. It outputs a master secret key $MSK$, a master update key $MUK$, and public parameters $PP$. 

**GenKey**(ID, $MSK$, $PP$): The private key generation algorithm takes as input an identity ID, the master secret key $MSK$, and public parameters $PP$. It outputs a private key $SK_ID$.

**Revoke**(ID, $T$, RL): The revocation algorithm takes as input an identity ID, revocation time $T$, a current revocation list RL. It outputs an updated revocation list RL.

**UpdateKey**(T, RL, MUK, PP): The update key generation algorithm takes as input update time $T$, a revocation list RL, the master update key $MUK$, and the public parameters $PP$. It outputs an update key $UK_T$.

**VerifyUK**(UK, $T$, RL, PP): The update key verification generation algorithm takes as input an update key $UK$, update time $T$ is in $T$, a revocation list RL, and the public parameters $PP$. It outputs 1 or 0 depending on the validity of the update key.

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

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

**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 a message $M$ or ⊥.

The correctness of DRIBE is defined as follows: For all $MSK$, $MUK$, and $PP$ generated by Setup($1^λ$), $SK_ID$ generated by GenKey(ID, $MSK$, $PP$) for any ID, $UK_T$ generated by UpdateKey(T, RL, MUK, PP) for any $T$ and RL, $CT_{ID,T}$ generated by Encrypt([ID, $T$, $M$, PP]) for any ID, $T$, and $M$, it is required that

- If (ID, $T''$) ∉ RL for all $T'' ≤ T$, then DeriveKey(SK_ID, UK_T, PP) = DK_{ID,T}.
- If (ID = ID') ∧ (T = T'), then Decrypt(CD_{ID,T}, DK_{ID',T'}, PP) = M.
- VerifyUK(UK_T, $T$, RL, PP) = 1.

Since DRIBE is an extension of RIBE, the IND-CPA security of RIBE must be satisfied to guarantee the message hiding security against external attackers. In addition, the indistinguishability under update-key attack (IND-UKA) security should be satisfied to guarantee the message hiding security when the cloud server that owns the master update key $MUK$ became an attacker. Lastly, we must ensure the update-key verifiability (UKV) security that guarantee the validity of update keys in order to prevent attacks that helps other external attackers by malicious generation of the update key by the cloud server.

The IND-CPA security model is the same as the IND-CPA security model of RIBE [3, 33].
Definition 3.2 (IND-CPA Security). The indistinguishability under chosen-plaintext attack (IND-CPA) security of DRIBE is defined as the following experiment between a challenger $C$ and an adversary $A$:

1. **Setup**: $C$ generates a master secret key $MSK$, a master update key $MUK$, and public parameters $PP$ by running $\text{Setup}(1^\lambda)$. It initializes a revocation list $RL = \emptyset$. It keeps $MSK, MUK$ to itself and gives $PP$ to $A$.

2. **Phase 1**: $A$ adaptively request a polynomial number of queries. $C$ handles these queries as follows:
   - For a private key query on an identity $ID$, it gives a private key $SK_{ID}$ to $A$ by running $\text{GenKey}(ID, MSK, PP)$.
   - For a revocation query on an identity $ID$ and time $T$, it updates $RL$ by running $\text{Revoke}(ID, T, RL)$.
   - For an update key query on time $T$, it gives an update key $UK_T$ to $A$ by running $\text{UpdateKey}(T, RL, MUK, PP)$.
   - For a decryption key query on an identity $ID$ and time $T$, it gives a decryption key $DK_{ID, T}$ to $A$ by running $\text{DeriveKey}(SK_{ID}, UK_T, PP)$.

3. **Challenge**: $A$ submits a challenge identity $ID^*$, challenge time $T^*$, and two challenge messages $M^*_0, M^*_1$ with equal length. Next, $C$ flips a random coin $\mu \in \{0, 1\}$ and gives the challenge ciphertext $CT^*$ to $A$ by running $\text{Encrypt}(ID^*, T^*, M^*_\mu, PP)$.

4. **Phase 2**: $A$ may continue to request additional queries subject to the same restrictions as before.

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

The queries of the adversary in the above experiment should satisfy the following conditions:

1. If a private key query on $ID^*$ was requested, then a revocation query on $ID^*$ and $T$ such that $T \leq T^*$ should be requested.

2. The update key and revocation queries are requested on time which is greater than or equal to the time of all previous queries, and these queries are requested only in non-decreasing order of time.

3. A revocation query on $T$ cannot be queried if an update key query on $T$ was already requested.

4. A decryption key query on $T$ cannot be requested before an update key query on $T$ was requested.

5. A decryption key query on $ID^*$ and $T^*$ was not requested.

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

The IND-UKA is a security model that considers the case that an cloud server that can access to the master update key $MUK$ is an attacker. In this security model, an attacker corresponding to the cloud server can request private key, revocation, and decryption key queries. Since the attacker can issue an arbitrary update key by using $MUK$, there is a restriction that the attacker cannot query a private key corresponding to the challenge identity $ID^*$ in order to prevent a simple attack. The goal of the attacker is to distinguish challenge messages when the challenge ciphertext of $ID^*$ and $T^*$ is given. A more detailed definition of the IND-UKA security is described as follows.
Definition 3.3 (IND-UKA Security). The indistinguishability under update-key attack (IND-UKA) security of DRIBE is defined in terms of the following experiment between a challenger $C$ and an adversary $A$:

1. **Setup**: $C$ generates a master secret key $MSK$, a master update key $MUK$, and public parameters $PP$ by running $\text{Setup}(1^\lambda)$. It initialize a revocation list $RL = \emptyset$. It keeps $MSK$ to itself and gives $MUK, PP$ to $A$.

2. **Phase 1**: $A$ adaptively request a polynomial number of queries. $C$ handles these queries as follows:
   - For a private key query on an identity $ID$, it gives a private key $SK_{ID}$ to $A$ by running $\text{GenKey}(ID, MSK, PP)$.
   - For a revocation query on an identity $ID$ and time $T$, it updates $RL$ by running $\text{Revoke}(ID, T, RL)$.
   - For a decryption key query on an identity $ID$ and time $T$, it takes an input $UK_T$ for $RL$ and gives a decryption key $DK_{ID,T}$ to $A$ by running $\text{DeriveKey}(SK_{ID}, UK_T, PP)$.

3. **Challenge**: $A$ submits a challenge identity $ID^*$, challenge time $T^*$, and two challenge messages $M_{0}^*$, $M_{1}^*$ with equal length. Next, $C$ flips a random coin $\mu \in \{0, 1\}$ and gives the challenge ciphertext $CT^*$ to $A$ by running $\text{Encrypt}(ID^*, T^*, M_{\mu}^*, PP)$.

4. **Phase 2**: $A$ may continue to request additional queries subject to the same restrictions as before.

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

The queries of the adversary in the above experiment should satisfy the following conditions:

1. A private key query on $ID^*$ was not requested.

2. The revocation queries are requested on time which is greater than or equal to the time of all previous queries, and these queries are requested only in non-decreasing order of time.

3. A decryption key query on $ID^*$ and $T^*$ was not requested.

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

The UKV is a security model that considers the case that a cloud server can help other external attackers by maliciously generating an update key. In this model, an attacker corresponding to the cloud server can request private key, revocation, and decryption key queries. The attacker finally submits a malicious update key $UK^*$ of time $T^*$ and two revocation lists of different $RL_0^*$ and $RL_1^*$. At this time, if the update key $UK^*$ is correctly verified for not only $RL_0^*$ but also $RL_1^*$ revocation list, then the attacker succeeds. A more detailed definition of the UKA security model is described as follows.

Definition 3.4 (Update-Key Verifiability). The update-key verifiability (UKV) of DRIBE is defined in terms of the following experiment between a challenger $C$ and an adversary $A$:

1. **Setup**: $C$ generates a master secret key $MSK$, a master update key $MUK$, and public parameters $PP$ by running $\text{Setup}(1^\lambda)$. It initialize a revocation list $RL = \emptyset$. It keeps $MSK$ to itself and gives $MUK, PP$ to $A$.

2. **Query**: $A$ adaptively request a polynomial number of queries. $C$ handles these queries as follows:
• For a private key query on an identity $ID$, it gives a private key $SK_{ID}$ to $A$ by running $\text{GenKey}(ID, MSK, PP)$.

• For a revocation query on an identity $ID$ and time $T$, it updates $RL$ by running $\text{Revoke}(ID, T, RL)$.

• For a decryption key query on an identity $ID$ and time $T$, it gives a decryption key $DK_{ID,T}$ to $A$ by running $\text{DeriveKey}(SK_{ID}, UK_{T}, PP)$.

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

3. Output: Finally, $A$ outputs an update key $UK^*$, a time period $T^*$, two revocation sets $RL^*_0, RL^*_1$. $C$ outputs 1 if the update key $UK^*$ satisfies the following two conditions, or outputs 0 otherwise: 1) $\text{VerifyUK}(UK^*, T^*, RL^*_0, PP) = 1$ and $\text{VerifyUK}(UK^*, T^*, RL^*_1, PP) = 1$, 2) $R^*_0 \neq R^*_1$ where $R^*_b$ is a set of identities $ID_i$ such that $(ID_i, T_i) \in RL^*_b$ and $T_i \leq T^*$.

The advantage of $A$ is defined as $\text{Adv}^{UKV}_{\text{DRIBE}, A}(\lambda) = \mathbb{Pr}[C = 1]$ where the probability is taken over all the randomness of the experiment. A DRIBE scheme is update-key verifiable if for all PPT adversary $A$, the advantage of $A$ in the above experiment is negligible in the security parameter $\lambda$.

3.2 Design Principle

Most of the previous RIBE schemes using a binary tree can efficiently process user revocation because the path of the binary tree is associated with a private key and the cover of the binary tree is associated with an update key [3, 21, 24, 31]. However, since these RIBE schemes use a secret sharing scheme to share the master secret key for each node of the binary tree, the private key generation algorithm and the update key generation algorithm require the same state information. For this reason, these RIBE schemes have a problem in that it is difficult to separate the private key generation process and the update key generation process independently, and thus it is difficult to delegate the update key generation to an external cloud server. In addition, in the previous RIBE schemes, it is impossible for a general user to publicly verify that the update key has been properly generated without the status information used to generate the update key.

In order to design a DRIBE scheme that can delegate the update key generation, we pay attention to the RIBE scheme proposed by Ma and Lin [25]. In terms of using a binary tree, the ML-RIBE scheme is also similar to the previous RIBE schemes using a binary tree. However, in the previous RIBE schemes, the path of the binary tree is related to a private key, but the ML-RIBE scheme has a big difference in that the path of the binary tree is related to a ciphertext. In order to clearly separate the underlying cryptographic primitives used for private key generation and update key generation, we use the ML-RIBE scheme that provides the decryption key exposure resistance (DKER) property by combining an HIBE scheme and an IBE scheme. In this case, the private key generation is processed by the master key of the HIBE scheme, and the update key generation is processed by generating key elements for each cover nodes of the binary tree by using the master key of the IBE scheme.

To provide the public verification of update keys, we perform verification using the IBE encryption process using the fact that the update key elements of the ML-RIBE scheme are all composed of the private keys of the IBE scheme. However, this simple method is not enough to provide the update key verification (UKV) security we defined earlier. The reason is that the attacker can access the master update key $MUK$ which can be used to generate an arbitrary update key. To overcome this problem, we additionally apply a collision-resistant hash function to prevent the update key generated by the attacker from being valid for not only a revocation list $RL$ but also other revocation list $RL'$. In this case, the attacker owning $MUK$ can
DRIBE.Setup: A generic DRIBE scheme that uses HIBE, IBBE, and CRHF schemes is described as follows:

1. It first obtains \( MK_{HIBE}, PP_{HIBE} \) by running HIBE.Setup\((1^\lambda, 2)\) and obtains \( MK_{IBBE}, PP_{IBBE} \) by running IBBE.Setup\((1^\lambda, \ell)\). It also selects a collision resistant hash function \( H \in \mathcal{H} \).
2. It defines a binary tree \( BT \) by running CS.Setup\((2^\ell)\) where an identity \( ID \) is uniquely assigned to a leaf node \( v \) such that \( \text{Label}(v) = ID \).
3. It outputs a master secret key \( MSK = MK_{HIBE} \), a master update key \( MUK = MK_{IBBE} \), and public parameters \( PP = (PP_{HIBE}, PP_{IBBE}, H, BT) \).

DRIBE.GenKey\((ID, MSK, PP)\): It obtains \( SK_{HIBE} \) by running HIBE.GenKey\((ID, MK_{HIBE}, PP_{HIBE})\). It outputs a private key \( SK_{ID} = SK_{HIBE} \).

DRIBE.Revoke\((ID, T, RL)\): If \( (ID, *) \) already exists in RL, it outputs RL. Otherwise, it adds \( (ID, T) \) to RL and outputs the updated RL.

DRIBE.UpdateKey\((T, RL, MUK, PP)\): To generate an update key for \( T \), it proceeds as follows:

1. It initializes \( RV = \emptyset \). For each \((ID_j, T_j) \in RL \) it adds a leaf node \( v_j \in BT \) which is associated with \( ID_j \) into \( RV \) if \( T_j \leq T \). It obtains \( CV_T \) by running CS.Cover\((BT, RV)\).
2. For each \( S_i \in CV_T \), it obtains \( SK_{IBBE,S_i} \) by running IBBE.GenKey\((H(\text{Label}(S_i)||T), MK_{IBBE}, PP_{IBBE})\).
3. It outputs an update key \( UK_T = (CV_T, \{SK_{IBBE,S_i}\}_{S_i \in CV_T}) \).

DRIBE.VerifyUK\((UK, T, RL, PP)\): Let \( UK_T = (CV_T, \{SK_{IBBE,S_i}\}) \). To verify an update key for \( T \) and RL, it proceeds as follows:

1. It initializes \( RV = \emptyset \). For each \((ID_j, T_j) \in RL \) it adds a leaf node \( v_j \in BT \) which is associated with \( ID_j \) into \( RV \) if \( T_j \leq T \). It obtains \( CV' \) by running CS.Cover\((BT, RV)\) and checks that \( CV' = CV_T \).
2. It sets \( S = \emptyset \). For each \( S_i \in CV_T \), it adds a string \( H(\text{Label}(S_i)||T) \) to \( S \). It chooses a random \( M \in M \) and obtains \( CT_{IBBE} \) by running IBBE.Encrypt\((S, M, PP_{IBBE})\).
3. For each \( S_i \in CV_T \), it performs the followings:
   (a) It retrieves \( SK_{IBBE,S_i} \) from \( UK_T \) and obtains \( M' \) by running IBBE.Decrypt\((CT_{IBBE}, SK_{IBBE,S_i}, PP_{IBBE})\).
   (b) If \( M' \neq M \), then it outputs 0 since \( UK \) is invalid.
4. Finally, it outputs 1 since \( UK \) is valid.

DRIBE.DeriveKey\((SK_{ID}, UK_T, PP)\): Let \( SK_{ID} = SK_{HIBE} \). To derive a decryption key for \( ID \) and \( T \), it proceeds as follows:
1. It obtains $PV_{ID}$ by running $\text{CS.Assign}(BT, v_{ID})$ where $v_{ID}$ is a leaf node such that $ID = Label(v_{ID})$. If $PV_{ID} \cap CV_T = \emptyset$, then it outputs $\bot$ since $ID$ was revoked in $UK_T$.

2. It obtains $DK_{HIBE}$ by running $\text{HIBE.DelegateKey}(SK_{HIBE}, T, PP_{HIBE})$.

3. Finally, it outputs a decryption key $DK_{ID, T} = (DK_{HIBE}, UK_T)$.

**DRIBE.Encrypt($ID, T, M, PP$):** To generate a ciphertext for $ID$ and $T$, it proceeds as follows:

1. It selects random $M_1 \in M$ and sets $M_2 = M \oplus M_1$. It obtains $CT_{HIBE}$ by running $\text{HIBE.Encrypt}((ID, T), M_1, PP_{HIBE})$.

2. Let $v_{ID}$ be a leaf node associated with $ID$ such that $ID = Label(v_{ID})$. It obtains $PV_{ID}$ by running $\text{CS.Assign}(BT, v_{ID})$.

3. It sets $S = \emptyset$. For each $S_i \in PV_{ID}$, it adds a string $H(Label(S_i) \parallel T)$ to $S$. It obtains $CT_{IBBE}$ by running $\text{IBBE.Encrypt}(S, M_2, PP_{IBBE})$.

4. Finally, it outputs a ciphertext $CT_{ID, T} = (CT_{HIBE}, CT_{IBBE})$ by implicitly including $ID$ and $T$.

**DRIBE.Decrypt($CT_{ID, T}, DK_{ID, T}, PP$):** Let $CT_{ID, T} = (CT_{HIBE}, CT_{IBBE})$ and $DK_{ID, T} = (DK_{HIBE}, UK_T = (CV_T, \{SK_{IBBE, S_i}\}))$. It proceeds as follows:

1. It first obtains $M_1$ by running $\text{HIBE.Decrypt}(CT_{HIBE}, DK_{HIBE}, PP_{HIBE})$.

2. Let $v_{ID}$ be a leaf node associated with $ID$ such that $ID = Label(v_{ID})$. It obtains $PV_{ID}$ by running $\text{CS.Assign}(BT, v_{ID})$.

3. It finds $(S_i, S_i) = \text{CS.Match}(PV_{ID}, CV_T)$. It retrieves $SK_{IBBE, S_i}$ from $UK_T$ and obtains $M_2$ by running $\text{IBBE.Decrypt}(CT_{IBBE}, SK_{IBBE, S_i}, PP_{IBBE})$.

4. Finally, it outputs a message $M = M_1 \oplus M_2$.

### 3.4 Correctness

In this section we show the correctness of the DRIBE scheme. The ciphertext of our DRIBE scheme consists of two elements: an HIBE ciphertext and an IBBE ciphertext. First, if the identity $ID$ and time $T$ of the HIBE ciphertext and the identity $ID$ and time $T$ of the HIBE private key are the same, we can decrypt $M_1$ by the correctness of the HIBE scheme. From the correctness of the CS scheme, a common node $v$ in a binary tree exists if the path of the binary tree associated with $ID$ is not revoked in the cover of the binary tree. The IBBE ciphertext sets all nodes in the path as a set of recipients, and the update key consists of the IBBE private keys associated with the cover nodes of the binary tree. Because of this, the common node $v$ becomes an element of the path node set, and $M_2$ is decrypted by the correctness of the IBBE scheme. Therefore, we can derive the correct message with the operation $M = M_1 \oplus M_2$.

The update key of our DRIBE scheme is composed of IBBE private keys for cover nodes related to leaf nodes excluding all revoked identities obtained from the revocation list $RL$. Therefore, if an IBBE ciphertext for these cover nodes as recipients are generated and this IBBE ciphertext is decrypted for each IBBE private key to verify that the correct message is decrypted, then the update key is verified correctly by the correctness of the IBBE scheme.
3.5 Security Analysis

In this section, we show that our DRIBE scheme provides the IND-CPA security, the IND-UKA security, and the UKV security by relying on the security of underlying cryptographic primitives.

Theorem 3.1. The above DRIBE scheme is IND-CPA secure if the underlying HIBE and IBBE schemes are IND-CPA secure.

Proof. Let ID* be the challenge identity and T* be the challenge time. We divide the behavior of an adversary as two types: Type-A and Type-B, which are defined as follows:

Type-A. An adversary is type-A if it requests a private key on an identity ID such that ID ≠ ID* for all private key queries.

Type-B. An adversary is type-B if it requests a private key on an identity ID such that ID = ID* for some private key query. In this case, the identity ID* should be revoked on time T such that T ≤ T* by the restriction of the security model.

Let Ei be the event that A behaves like type-i adversary. From Lemmas 3.2 and 3.3, we obtain the following result

\[
\text{Adv}_{\text{DRIBE, A}}^{\text{IND-CPA}}(\lambda) \leq \Pr[E_A] \cdot \text{Adv}_{\text{DRIBE, A}}^{\text{IND-CPA}}(\lambda) + \Pr[E_B] \cdot \text{Adv}_{\text{DRIBE, A}}^{\text{IND-CPA}}(\lambda) \\
\leq \text{Adv}_{\text{HIBE, B}}^{\text{IND-CPA}}(\lambda) + \text{Adv}_{\text{IBBE, B}}^{\text{IND-CPA}}(\lambda).
\]

This completes our proof.

Lemma 3.2. For the type-A adversary, the DRIBE scheme is IND-CPA secure if the HIBE scheme is IND-CPA secure.

Proof. Suppose there exists a type-A adversary A that attacks the DRIBE scheme with a non-negligible advantage. An algorithm B that attacks the HIBE scheme is initially given public parameters PP_{HIBE} by a challenger C. Then B that interacts with A is described as follows:

Setup: B generates MK_{IBBE}, PP_{IBBE} by running the setup algorithm of IBBE. It initializes RL = ∅ and gives PP = (PP_{HIBE}, PP_{IBBE}, H, BT) to A.

Phase 1: A adaptively requests a polynomial number of queries. B handles these queries as follows:

- For a private key query on ID, B proceeds as follows: It receives SK_{HIBE} from C by querying a private key for ID. It gives SK_{ID} = SK_{HIBE} to A.
- For a revocation query on ID and T, B updates RL by running the revocation algorithm of DRIBE.
- For an update key query on T, B generates UK_T by using RL and MK_{IBBE} and gives UK_T to A.
- For a decryption key query on ID and T, B proceeds as follows: It receives DK_{HIBE} from C by querying a private key for ID and T. It generates UK_T by using MK_{IBBE}. It gives DK_{ID,T} = (DK_{HIBE}, UK_T) to A.

Challenge: A submits a challenge identity ID*, challenge time T*, and two challenge messages M_0^*, M_1^*. B proceeds as follows:

1. It first select a random M_2 ∈ M and sets M_{1,0} = M_0^* ⊕ M_2, M_{1,1} = M_1^* ⊕ M_2.

14
2. Next, it receives $CT_{HIBE}^*$ from C by submitting $ID^*$, $T^*$, and two challenge messages $M_{1,0}, M_{1,1}$.

3. It obtains $PV^*$ by running $\text{CS.Assign}(BT, v^*)$ where a leaf node $v^*$ is associated with $ID^*$. It sets $S = \emptyset$. For each $S_i \in PV^*$, it adds $H(\text{Label}(S_i)||T^*)$ to $S$. It generates $CT_{IBBE}^*$ by running $\text{DRIBE.Encrypt}(S, M_2, PP_{IBBE})$.

4. It gives a challenge ciphertext $CT^* = (CT_{HIBE}^*, CT_{IBBE}^*)$ to A.

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

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

In order to analyze the correctness of the simulation described above, we show that it is possible to process the private key and decryption key query of the DRIBE scheme by using the private key query of the HIBE scheme. The type-A attacker requests the DRIBE private key query with the restriction that $ID \neq ID^*$, so the DRIBE private key query can be handled by using the HIBE private key query. In the constraints of the HIBE scheme, the type-A attacker requests the DRIBE private key query with the restriction that $ID \neq ID^*$ or $ID = ID^* \wedge T \neq T^*$, so the DRIBE decryption key query also can be handled by using the HIBE private key query. The challenge message selected by the attacker in the challenge phase is linked to the HIBE challenge message, so the DRIBE challenge ciphertext is linked to the HIBE challenge ciphertext. Thus the probability of distinguishing the HIBE challenge ciphertext becomes the same as the probability of distinguishing the DRIBE challenge ciphertext.

**Lemma 3.3.** For the type-B adversary, the DRIBE scheme is IND-CPA secure if the IBBE scheme is IND-CPA secure.

**Proof.** Suppose there exists a type-B adversary $A$ that attacks the DRIBE scheme with a non-negligible advantage. An algorithm $B$ that attacks the IBBE scheme is initially given public parameters $PP_{IBBE}$ by a challenger $C$. Then $B$ that interacts with $A$ is described as follows:

**Setup:** $B$ generates $MK_{HIBE}, PP_{HIBE}$ by running the setup algorithm of HIBE. It initializes $RL = \emptyset$ and gives $PP = (PP_{HIBE}, PP_{IBBE}, H, BT)$ to $A$.

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

- For a private key query on $ID$, $B$ generates $SK_{ID}$ by using $MK_{HIBE}$ and gives $SK_{ID}$ to $A$.
- For a revocation query on $ID$ and $T$, $B$ updates $RL$ by running the revocation algorithm of DRIBE.
- For an update key query on $T$, $B$ proceeds as follows:
  1. It initializes $RV = \emptyset$. For each $(ID_j, T_j) \in RL$, it adds a leaf node $v_j \in BT$ into $RV$ if $T_j \leq T$. It obtains $CV_T$ by running $\text{CS.Cover}(BT, RV)$.
  2. For each $S_i \in CV_T$, it receives $SK_{IBBE,S_i}$ from $C$ by querying a private key for $H(\text{Label}(S_i)||T)$.
  3. It creates $UK_T = \{CV_T, \{SK_{IBBE,S_i}\}_{S_i \in CV_T}\}$ and gives $UK_T$ to $A$.
- For a decryption key query on $ID$ and $T$, $B$ proceeds as follows: It generates $DK_{HIBE}$ for $ID$ and $T$ by using $MK_{HIBE}$. It receives $UK_T$ by querying an update key to its own oracle. It gives $DK_{ID,T} = (DK_{HIBE}, UK_T)$ to $A$.

**Challenge:** $A$ submits a challenge identity $ID^*$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$. $B$ proceeds as follows:
1. It first selects a random $M_1 \in \mathcal{M}$ and sets $M_{2,0} = M_0^* \oplus M_1, M_{2,1} = M_1^* \oplus M_1$. It generates $CT^*_HIBE$ by running HIBE.Encrypt$((ID^*, T^*), M_1, PP_HIBE)$.

2. It obtains $PV^*$ by running CS.Assign$((BT, v^*))$ where a leaf node $v^*$ is associated with $ID^*$. It sets $S = \emptyset$. For each $S_i \in PV^*$, it adds $H(Label(S_i) \| T^*)$ to $S$.

3. It receives $CT^*_IBBE$ from $C$ by submitting a challenge set $S$ and challenge messages $M_{2,0}, M_{2,1}$.

4. It gives a challenge ciphertext $CT^* = (CT^*_HIBE, CT^*_IBBE)$ to $A$.

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

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

In order to analyze whether the simulation described above is correct, we show that it is possible to process the update key and decryption key queries of the DRIBE scheme by using the private key queries of the IBBE scheme. From the restriction of a type-B attacker, we have that a private key for $ID^*$ is revoked in an update key at time $T^*$ if the attacker requested a private key for $ID^*$. When generating the update key of the DRIBE scheme, the string $H(S_i \| T)$ is used as the identity of the IBBE scheme. Thus, the update key at time $T \neq T^*$ can be created by generating IBBE private keys using the fact that $H(S_i \| T) \neq H(S_i \| T^*)$ since $T \neq T^*$. The update key at time $T = T^*$ also can be created by generating IBBE private keys using the fact that $H(S_i \| T^*) \neq H(S_i \| T^*)$ because the path nodes of $ID^*$ are all removed from the cover nodes at time $T^*$. The decryption key query can also be easily handled by using the update key query in the simulation. Finally, in the challenge phase, the attacker’s challenge message is linked to the IBBE challenge message, so the DRIBE challenge ciphertext is linked to the IBBE challenge ciphertext.

**Theorem 3.4.** The above DRIBE scheme is IND-UKA secure if the underlying HIBE scheme is IND-CPA secure.

**Proof.** Suppose there exists an adversary $A$ that attacks the DRIBE scheme with a non-negligible advantage. An algorithm $B$ that attacks the HIBE scheme is initially given public parameters $PP_HIBE$ by a challenger $C$. Then $B$ that interacts with $A$ is described as follows:

**Setup**: $B$ generates $MK_{IBBE}, PP_{IBBE}$ by running the setup algorithm of IBBE. It initializes $RL = \emptyset$ and gives $MUK = MK_{IBBE}, PP = (PP_HIBE, PP_{IBBE}, H, BT)$ to $A$.

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

- For a private key query on $ID$, $B$ proceeds as follows: It receives $SK_{HIBE}$ from $C$ by querying a private key for $ID$. It gives $SK_{ID} = SK_{HIBE}$ to $A$.
- For a revocation query on $ID$ and $T$, $B$ updates $RL$ by running the revocation algorithm of DRIBE.
- For a decryption key query on $ID$ and $T$, $B$ proceeds as follows: It receives $DK_{HIBE}$ from $C$ by querying a private key for $ID$ and $T$. Next, it generates $UK_T$ by using $MK_{IBBE}$. It gives $DK_{ID,T} = (DK_{HIBE}, UK_T)$ to $A$.

**Challenge**: $A$ submits a challenge identity $ID^*$, challenge time $T^*$, and two challenge messages $M^*_0, M^*_1$. $B$ proceeds as follows:

1. It first select a random $M_2 \in \mathcal{M}$ and sets $M_{1,0} = M_0^* \oplus M_2, M_{1,1} = M_1^* \oplus M_2$.

2. Next, it receives $CT^*_HIBE$ from $C$ by submitting $ID^*, T^*$, and two challenge messages $M_{1,0}, M_{1,1}$.
Lemma 3.2. Suppose there exists an adversary $A$ that attacks the DRIBE scheme with a non-negligible advantage. An algorithm $B$ that attacks the CRHF scheme is initially given $H$ by a challenger $C$. Then $B$ that interacts with $A$ is described as follows:

**Setup:** $B$ generates $MK_{HIBE}, PP_{HIBE}$ by running the setup algorithm of HIBE. It also generates $MK_{IBBE}, PP_{IBBE}$ by running the setup algorithm of IBBE. It initializes $RL = \emptyset$. It sets $MSK = MK_{HIBE}$ and gives $MUK = MK_{IBBE}, PP = (PP_{HIBE}, PP_{IBBE}, H, BT)$ to $A$.

**Query:** $A$ adaptively requests a polynomial number of private key, revocation, and decryption key queries. $B$ can handle these queries since it knows $MSK$.

**Output:** Finally, $A$ outputs an update key $UK^*$, a time period $T^*$, two revocation lists $RL_0^*, RL_1^*$. $B$ proceeds as follows:

1. It first checks the validity of the update key $UK^*$ by calling $\text{DRIBE.VerifyUK}(UK^*, T^*, RL_0^*, PP) = 1$ and $\text{DRIBE.VerifyUK}(UK^*, T^*, RL_1^*, PP) = 1$.

2. It initializes $RV_0^* = \emptyset$ and $RV_1^* = \emptyset$. For each $(ID_i, T_i) \in RL_0^*$, it adds a leaf node $v_i \in BT$ which is associated with $ID_i$ into $RV_0^*$ if $T_i \leq T^*$. For each $(ID_i, T_i) \in RL_1^*$, it adds a leaf node $v_i \in BT$ which is associated with $ID_i$ into $RV_1^*$ if $T_i \leq T^*$. It checks that $RV_0^* \neq RV_1^*$.

3. It obtains $CV_0^*$ and $CV_1^*$ by calling $\text{CS.Cover}(BT, RV_0^*)$ and $\text{CS.Cover}(BT, RV_1^*)$ respectively. Next, it derives a common node set $A = CV_0^* \cap CV_1^*$. It computes two (non-empty) disjoint sets $B_0 = CV_0^* \setminus A$ and $B_1 = CV_1^* \setminus A$ since $CV_0^* \neq CV_1^*$.

4. Let $UK^* = (CV^*, \{SK_{IBBE, S_i}\}_{S_i \in CV^*})$. It chooses a node $v_k \in B_0$ and retrieves a private key $SK_{IBBE, S_k} \in UK^*$ which is associated with the node $v_k$. Note that the private key should exist since the update key is verified under $RL_0^*$.

5. For each $v_i \in B_1$, it proceeds as follows:
   
   (a) It chooses a random $M \in M$ and obtains $CT_i$ by running $\text{IBBE.Encrypt}(H(\text{Label}(v_i))|T^*), M, PP_{IBBE})$.

   (b) It obtains $M'$ by running $\text{IBBE.Decrypt}(CT_i, SK_{IBBE, S_k}, PP_{IBBE})$. If $M = M'$, then it sets $v_j = v_i$ and breaks the loop. Note that the matching node $v_j$ should exist since the update key is verified under $RL_1^*$.

The restrictions of the attacker in the IND-UKA security model are the same as those of the type-A attacker in the Lemma 3.2. Thus the correctness of this simulation is the same as that of the simulation in Lemma 3.2. $\square$

**Theorem 3.5.** The above DRIBE scheme is update-key verifiable if the underlying hash function is collision resistant.

**Proof:** Suppose there exists an adversary $A$ that attacks the DRIBE scheme with a non-negligible advantage. An algorithm $B$ that attacks the CRHF scheme is initially given $H$ by a challenger $C$. Then $B$ that interacts with $A$ is described as follows:

**Setup:** $B$ generates $MK_{HIBE}, PP_{HIBE}$ by running the setup algorithm of HIBE. It also generates $MK_{IBBE}, PP_{IBBE}$ by running the setup algorithm of IBBE. It initializes $RL = \emptyset$. It sets $MSK = MK_{HIBE}$ and gives $MUK = MK_{IBBE}, PP = (PP_{HIBE}, PP_{IBBE}, H, BT)$ to $A$.

**Query:** $A$ adaptively requests a polynomial number of private key, revocation, and decryption key queries. $B$ can handle these queries since it knows $MSK$.

**Output:** Finally, $A$ outputs an update key $UK^*$, a time period $T^*$, two revocation lists $RL_0^*, RL_1^*$. $B$ proceeds as follows:

1. It first checks the validity of the update key $UK^*$ by calling $\text{DRIBE.VerifyUK}(UK^*, T^*, RL_0^*, PP) = 1$ and $\text{DRIBE.VerifyUK}(UK^*, T^*, RL_1^*, PP) = 1$.

2. It initializes $RV_0^* = \emptyset$ and $RV_1^* = \emptyset$. For each $(ID_i, T_i) \in RL_0^*$, it adds a leaf node $v_i \in BT$ which is associated with $ID_i$ into $RV_0^*$ if $T_i \leq T^*$. For each $(ID_i, T_i) \in RL_1^*$, it adds a leaf node $v_i \in BT$ which is associated with $ID_i$ into $RV_1^*$ if $T_i \leq T^*$. It checks that $RV_0^* \neq RV_1^*$.

3. It obtains $CV_0^*$ and $CV_1^*$ by calling $\text{CS.Cover}(BT, RV_0^*)$ and $\text{CS.Cover}(BT, RV_1^*)$ respectively. Next, it derives a common node set $A = CV_0^* \cap CV_1^*$. It computes two (non-empty) disjoint sets $B_0 = CV_0^* \setminus A$ and $B_1 = CV_1^* \setminus A$ since $CV_0^* \neq CV_1^*$.

4. Let $UK^* = (CV^*, \{SK_{IBBE, S_i}\}_{S_i \in CV^*})$. It chooses a node $v_k \in B_0$ and retrieves a private key $SK_{IBBE, S_k} \in UK^*$ which is associated with the node $v_k$. Note that the private key should exist since the update key is verified under $RL_0^*$.

5. For each $v_i \in B_1$, it proceeds as follows:
   
   (a) It chooses a random $M \in M$ and obtains $CT_i$ by running $\text{IBBE.Encrypt}(H(\text{Label}(v_i))|T^*), M, PP_{IBBE})$.

   (b) It obtains $M'$ by running $\text{IBBE.Decrypt}(CT_i, SK_{IBBE, S_k}, PP_{IBBE})$. If $M = M'$, then it sets $v_j = v_i$ and breaks the loop. Note that the matching node $v_j$ should exist since the update key is verified under $RL_1^*$. $\square$
6. Finally, it outputs $\text{Label}(v_k)||T^*$ and $\text{Label}(v_j)||T^*$ as the collision of the hash function since $v_k \neq v_j$.

Let’s analyze the correctness of the simulation. Since the simulator has the master secret key $\text{MSK}$, it can correctly process all queries of an attacker. Now, let’s show that the hash collision outputted by the simulator is correct. We have that $\text{RV}^*_0 \neq \text{RV}^*_1$ by the definition of the security model. We also obtains that $\text{CV}^*_0 \neq \text{CV}^*_1$ and $|\text{CV}^*_0| = |\text{CV}^*_1|$ because the cover $\text{CV}^*_b$ is derived from the revoked nodes $\text{RV}^*_b$ and the IBBE private key in an update key is associated with each node in the cover. Therefore, two node sets $B_0$ and $B_1$ in which the common nodes are removed cannot be empty. Because the IBBE private key corresponding to the node $v_k \in B_0$ must be able to decrypt the IBBE ciphertext of the node $v_j \in B_1$ due to the correctness of the verification algorithm, two hash strings $\text{H}(\text{Label}(v_k)||T^*)$ and $\text{H}(\text{Label}(v_j)||T^*)$ are associated with the same identity of the IBBE scheme. Therefore, two strings $\text{Label}(v_k)||T^*$ and $\text{Label}(v_j)||T^*$ become the collision of the hash function since $v_k \neq v_j$.

3.6 Discussions

Efficiency Analysis. The public parameters of the DRIBE scheme consist of HIBE and IBE public parameters. In addition, the public parameters require binary tree information to be used in the DRIBE scheme, and the binary tree has a depth of $2\lambda$ where $\lambda$ is the security parameter. The DRIBE private key is compact since it composed of one HIBE private key. The DRIBE update key is related to the number of cover nodes in the binary tree. If the number of revoked users is $r$, then the size of the cover nodes is $r \log(2^{2\lambda}/r)$. Thus the update key consists of approximately $2r\lambda$ IBEB private keys. The DRIBE ciphertext is compact because it consists of one HIBE ciphertext and one IBBE ciphertext. The DRIBE encryption algorithm performs one HIBE encryption and one IBBE encryption, and the decryption algorithm requires one HIBE decryption and one IBBE decryption. Since the DRIBE update key verification algorithm needs to check all IBBE private keys in the update key, it requires one IBBE encryption and $2r\lambda$ IBBE decryption operations.

CCA Security. The IND-CPA security of our DRIBE scheme was already shown. The stronger security model for encryption schemes is the IND-CCA security model, which allows an attacker to additionally query the decryption of chosen ciphertexts. In order to design an IND-CCA secure DRIBE scheme, we can apply the CHK transformation of Canetti et al. [8] that uses the key delegation function of HIBE. That is, the modified encryption algorithm derived from the CHK transformation generates a public key and a private key of a one-time signature (OTS) scheme, and binds the OTS public key with the identity of the HIBE scheme to generate a CPA ciphertext. Next, an OTS signature is generated for the CPA ciphertext to ensure the integrity of the CPA ciphertext. This modified ciphertext to ensure the IND-CCA security consists of an OTS public key, a CPA ciphertext, and an OTS signature. Note that a simulator in the security proof can process the decryption query in the IND-CCA model since it can derive an HIBE private key for the OTS public key of the queried ciphertext by using the key delegation function of the underlying HIBE scheme.

Server-Aided Decryption. In order to improve the decryption performance of an RIBE scheme, a server-aided RIBE scheme, in which a cloud server processes part of the decryption operation, was proposed [28]. Our DRIBE scheme naturally supports the server-aided decryption functionality. That is, a decrypter first sends the IBBE ciphertext of a DRIBE ciphertext and the DRIBE update key of a DRIBE decryption key to a cloud server. Then, the cloud server obtains a partial message $M_2$ by decrypting the IBBE ciphertext using an IBEB private key which is associated to the matching node of a binary tree, and returns it to the decrypter. Note that the cloud server cannot obtain any information about the original message since $M_2$ was selected as a random value. After that, the decrypter decrypts the HIBE ciphertext to obtain $M_1$, and combines $M_1$ with $M_2$ to derive the original message. The decryption process of the decrypter is efficient.
since the decrypter needs only one HIBE decryption operation.

**Reducing the Update Key Size.** In our DRIBE scheme, the path of a binary tree is associated with a receiver’s identity. Thus the depth of the binary tree should be set to be proportional to the length of an identity, and the depth should be the output length of a hash function if an arbitrary long string is used for the identity. That is, when the security parameter of the system is $\lambda$, the depth of the binary tree should be $2\lambda$. Since the binary tree is also involved in the generation of an update key, the size of the update key is $r \log(2^{2\lambda}/r)$ where $r$ is the number of revoked users. The problem of our DRIBE scheme is that the depth of the binary tree should be $2\lambda$, whereas the depth of the previous RIBE schemes is just $\log N$ where $N$ is the number of system users. Specifically, if we set $\lambda = 80$ and $N = 2^{30}$, then the update key size of the previous RIBE schemes is approximately $r \times 30$, but the update key size of our DRIBE scheme is $r \times 160$. Thus the update key of the DRIBE scheme is 5 times larger than that of the previous RIBE schemes. To reduce the update key size, we propose a DRIBE-INC scheme in the next section that generates an incremental update key for revoked users in a short period of time.

### 4 Delegated RIBE with Incremental Update Keys

In this section, we define the syntax and security model of DRIBE-INC that supports the generation of incremental update keys. We next propose a DRIBE-INC scheme by generically combining DRIBE, HIBE, IBBE schemes and a hash function, and prove the security of our construction.

#### 4.1 Definition

A DRIBE-INC scheme is similar to a DRIBE scheme that delegate the generation of update keys to an external cloud server except that it additionally supports the generation of incremental update keys. To support incremental update, the DRIBE-INC scheme separates an update key into a base update key and an incremental update key. If we let $T_b$ be an interval period for generating the base update key, a time period $T$ is expressed as $T = T_b + T_c$ where $T_c = T \mod T_b$. The cloud server generates a base update key that includes all users who were not revoked before the time $T_b$ if $T = T_b$, and generates an incremental update key that includes all users that were not revoked between time $T_b$ and time $T$ if $T \neq T_b$. Similar to the DRIBE scheme, a sender creates a ciphertext for an identity $ID$ and time $T$ and delivers the ciphertext to a receiver. The receiver can decrypt the ciphertext if $ID$ is not revoked in both the base update key of $T_b$ and the incremental update key of $T$. The detailed syntax of DRIBE-INC is defined as follows.

**Definition 4.1** (Incremental Delegated RIBE). An incremental DRIBE (DRIBE-INC) scheme that is associated with identity space $\mathcal{I}$, time space $\mathcal{T}$, and message space $\mathcal{M}$, consists of eight algorithms Setup, GenKey, UpdateKey, IncUpdateKey, DeriveKey, Encrypt, Decrypt, and Revoke, which are defined as follows:

- **Setup($1^k$):** The setup algorithm takes as input a security parameter $1^k$. It outputs a master key $MSK$, a master update key $MUK$, and public parameters $PP$.

- **GenKey($ID, MSK, PP$):** The private key generation algorithm takes as input an identity $ID$, the master key $MK$ and public parameters $PP$. It outputs a private key $SK_{ID}$.

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


The correctness of DRIBE-INC is defined as follows: For all $\text{MSK}$, $\text{MUK}$, and the public parameters $\text{PP}$. It outputs a base update key $\text{BUK}_{T_b}$ for a base time $T_b$.

**IncUpdateKey($T, RL, MUK, PP$):** The incremental update key generation algorithm takes as input update time $T \in \mathcal{T}$, a revocation list $RL$, the master update key $MUK$, and the public parameters $PP$. It outputs an incremental update key $IUK_T$.

**VerifyBUK($BUK_{T_b}, RL, PP$):** The base update key verification algorithm takes as input a base update key $BUK$, base time $T_b \in \mathcal{T}$, a revocation list $RL$, and the public parameters $PP$. It outputs 1 or 0 depending on the validity of the base update key.

**VerifyIUK($IUK_T, RL, PP$):** The incremental update key verification algorithm takes as input an incremental update key $IUK$, update time $T \in \mathcal{T}$, a revocation list $RL$, and the public parameters $PP$. It outputs 1 or 0 depending on the validity of the incremental update key.

**DeriveKey($SK_{ID}, BUK_{T_b}, IUK_T, PP$):** The decryption key derivation algorithm takes as input a private key $SK_{ID}$, a base update key $BUK_{T_b}$, an incremental update key $IUK_T$, and the public parameters $PP$. It outputs a decryption key $DK_{ID,T}$ or $\bot$.

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

**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 a message $M$ or $\bot$.

The correctness of DRIBE-INC is defined as follows: For all $\text{MSK}$, $\text{MUK}$, and $\text{PP}$ generated by $\text{Setup}(1^\lambda)$, $SK_{ID}$ generated by $\text{GenKey}(ID, \text{MSK}, \text{PP})$ for any $ID$, $BUK_{T_b}$ and $IUK_T$ generated by $\text{UpdateKey}(T_b, RL, \text{MUK}, \text{PP})$ and $\text{IncUpdateKey}(T, RL, \text{MUK}, \text{PP})$ respectively for any $T = T_b + T_b$ and $RL$, $CT_{ID,T}$ generated by $\text{Encrypt}(ID, T, M, PP)$ for any $ID$, $T$, and $M$, it is required that

- If $(ID, T') \notin RL$ for all $T' \leq T$, then $\text{DeriveKey}(SK_{ID}, BUK_{T_b}, IUK_T, PP) = DK_{ID,T}$.
- If $(ID = ID') \land (T = T')$, then $\text{Decrypt}(CT_{ID,T}, DK_{ID',T'}, PP) = M$.
- $\text{VerifyBUK}(BUK_{T_b}, T_b, RL, PP) = 1$.
- $\text{VerifyIUK}(IUK_T, T, RL, PP) = 1$.

The security models of DRIBE-INC are similar to those of DRIBE, which are the IND-CPA security model that defines the message hiding against external attackers, the IND-UKA security model that defines the message hiding against external cloud servers, and the UKA security model that defines the validity of update keys. The only change in the security models is that an update key is divided into a base update key and an incremental update key. For definitions of security models, please refer to the security definitions of DRIBE in Section 5.1.
4.2 Construction

To design a DRIBE-INC scheme, we express time $T$ as $T = T_b + T_c$ where $T_c \equiv T \mod T_s$. If $T \equiv 0 \mod T_s$, we use the DRIBE scheme to generate a base update key for non-revoked users before time $T$. If $T \not\equiv 0 \mod T_s$, we use the HIBE and IBBE schemes to generate an incremental update key for non-revoked users between time $T_b$ and time $T$. The method of creating the incremental update key is similar to that in the DRIBE scheme.

A generic DRIBE-INC scheme that uses DRIBE, HIBE, IBBE, and CRHF schemes is described as follows:

**DRIBE-INC.Setup**($1^k$): Let $I = \{0, 1\}^k$ be the identity space.

1. It first obtains $MSK_{DRIBE}, MUK_{DRIBE}, RL_{DRIBE}, PP_{DRIBE}$ by running **DRIBE.Setup**($1^k$). It obtains $MK_{HIBE}, PP_{HIBE}$ by running **HIBE.Setup**($1^k, 2$) and obtains $MK_{IBBE}, PP_{IBBE}$ by running **IBBE.Setup**($1^k, \ell$). It also selects a collision resistant hash function $H \in \mathcal{H}$.
2. It sets a binary tree $BT$ with depth $\ell$ where an identity $ID \in I$ is uniquely assigned to a leaf node such that $Label(v) = ID$.
3. It outputs a master secret key $MSK = (MSK_{DRIBE}, MK_{HIBE})$, a master update key $MUK = (MUK_{DRIBE}, MK_{IBBE})$, and public parameters $PP = (PP_{DRIBE}, PP_{HIBE}, PP_{IBBE}, H, BT)$.

**DRIBE-INC.GenKey**(ID, MSK, PP): To generate a private key of ID, it proceeds as follows:

1. It obtains $SK_{DRIBE}$ by running **DRIBE.GenKey**(ID, MSK, PP). It obtains $SK_{HIBE}$ by running **HIBE.GenKey**(ID, MK, PP).
2. It outputs a private key $SK_{ID} = (SK_{DRIBE}, SK_{HIBE})$.

**DRIBE-INC.Revoke**(ID, T, RL): If (ID, *) already exists in RL, it outputs RL. Otherwise, it adds (ID, T) to RL and outputs the updated RL.

**DRIBE-INC.UpdateKey**(T, RL, MUK, PP): Let $T_b$ be base time such that $T_b \equiv 0 \mod T_s$. If $T_b \not\equiv 0 \mod T_s$, then it returns $\bot$ since a base update key will not be generated.

1. It initializes $BRL = \emptyset$. For each (ID, $T_i$) $\in RL$, it adds (ID, $T_j$) to BRL if $T_j \leq T_b$. It obtains $UK_{DRIBE}$ by running **DRIBE.UpdateKey**(T, BRL, MUK, PP).
2. Finally, it outputs a base update key $BUK_{T_b} = UK_{DRIBE}$.

**DRIBE-INC.IncUpdateKey**(T, RL, MUK, PP): Let $T = T_b + T_c$ where $T_b \equiv 0 \mod T_s$ and $0 \leq T_c < T_s$.

1. It initializes $IRV = \emptyset$. For each (ID, $T_i$) $\in RL$, it adds $v_i$ which is associated with ID into IRV if $T_b < T_i \leq T$. It obtains $ICV_T$ by running **CS.Cover**(BT, IRV).
2. For each $S_i \in ICV_T$, it obtains $SK_{IBBE,S_i}$ by running **IBBE.GenKey**(H(Label(S_i)||T), MK, PP).
3. Finally, it outputs an incremental update key $IU_K_T = (ICV_T, \{SK_{IBBE,S_i}\}_{S_i \in ICV_T})$.

**DRIBE-INC.VerifyBUK**(BUK, T_b, RL, PP): To verify the validity of BUK, it returns **DRIBE.VerifyUK**(BUK, T_b, RL, PP).

**DRIBE-INC.VerifyUK**(IU_K, T, RL, PP): Let $IU_K = (ICV_T, \{SK_{IBBE,S_i}\})$. It proceeds as follows:
1. It initializes $RV = \emptyset$. For each $(ID_j, T_j) \in RL$, it adds a leaf node $v_j \in BT$ which is associated with $ID_j$ into $RV$ if $T_j \leq T$. It obtains $CV'$ by running $\text{CS.Cover}(BT, RV)$ and checks that $CV' = CV_T$.

2. It sets $S = \emptyset$. For each $S_i \in CV_T$, it adds a string $H(Label(S_i)||T)$ to $S$. It chooses a random $M \in \mathcal{M}$ and obtains $CT_{IBBE}$ by running $\text{IBBE.Encrypt}(S, M, PP_{IBBE})$.

3. For each $S_i \in CV_T$, it performs the followings:
   
   (a) It retrieves $SK_{IBBE, S_i}$ from $UK_T$ and obtains $M'$ by running $\text{IBBE.Decrypt}(CT_{IBBE}, SK_{IBBE, S_i}, PP_{IBBE})$.
   
   (b) If $M' \neq M$, then it outputs 0 since $UK$ is invalid.

4. Finally, it outputs 1 since $UK$ is valid.

**DRIBE-INC.DeriveKey(SK_{ID}, BU_{K_T}, IU_{K_T}, PP):** Let $SK_{ID} = SK_{DRIBE}$, $BU_{K_T} = UK_{DRIBE}$, and $IU_{K_T} = (ICV_T, \{SK_{IBBE, S_i}\})$. If $T \neq T_b + T_c$ for some $T_c$ such that $0 \leq T_c < T$, then it returns ⊥ since $BU_{K_T}$ and $IU_{K_T}$ are not a valid update key pair.

1. It obtains $DK_{DRIBE}$ by running $\text{DRIBE.DeriveKey}(SK_{DRIBE}, IU_{K_T}, PP_{DRIBE})$. If $DK_{DRIBE} = \bot$, it returns ⊥ since ID was revoked.

2. It obtains $PV_{ID}$ by running $\text{CS.Assign}(BT, v_{ID})$ where $v_{ID}$ is a leaf node such that $ID = Label(v_{ID})$. If $PV_{ID} \cap ICV_T = \emptyset$, then it outputs ⊥ since ID was revoked in $IU_{K_T}$.

3. It obtains $DK_{HIBE}$ by running $\text{HIBE.DelegateKey}(SK_{HIBE}, T, PP_{HIBE})$.

4. Finally, it outputs a decryption key $DK_{ID,T} = (DK_{DRIBE}, DK_{HIBE}, IU_{K_T})$.

**DRIBE-INC.Encrypt(ID, T, M, PP):** Let $T = T_b + T_c$ where $T_b \equiv 0 \mod T_c$ and $0 \leq T_c < T$.

1. It selects random $M_1, M_2 \in \mathcal{M}$ and sets $M_3 = M \oplus M_1 \oplus M_2$. It obtains $CT_{DRIBE}$ by running $\text{DRIBE.Encrypt}(ID, T_b, M_1, PP_{DRIBE})$. It also obtains $CT_{HIBE}$ by running $\text{HIBE.Encrypt}(ID, T, M_2, PP_{HIBE})$.

2. Let $v_{ID}$ be a leaf node associated with $ID$ such that $ID = Label(v_{ID})$. It obtains $PV_{ID}$ by running $\text{CS.Assign}(BT, v_{ID})$.

3. It sets $S = \emptyset$. For each $S_i \in PV_{ID}$, it adds a string $H(Label(S_i)||T)$ to $S$. Next, it obtains $CT_{IBBE}$ by running $\text{IBBE.Encrypt}(S, M_3, PP_{IBBE})$.

4. Finally, it outputs a ciphertext $CT_{ID,T} = (CT_{DRIBE}, CT_{HIBE}, CT_{IBBE})$.

**DRIBE-INC.Decrypt(CT_{ID,T}, DK_{ID,T}, PP):** Let $CT_{ID,T} = (CT_{DRIBE}, CT_{HIBE}, CT_{IBBE})$ and $DK_{ID,T} = (DK_{DRIBE}, DK_{HIBE}, IU_{K_T})$ where $IU_{K_T} = (ICV_T, \{SK_{IBBE, S_i}\})$. If $(ID \neq ID') \lor (T \neq T')$, then it returns ⊥. Otherwise, it proceeds as follows:

1. It obtains $M_1$ by running $\text{HIBE.Decrypt}(CT_{HIBE}, DK_{HIBE}, PP_{HIBE})$. It obtains $M_2$ by running $\text{DRIBE.Decrypt}(CT_{DRIBE}, DK_{DRIBE}, PP_{DRIBE})$.

2. Let $v_{ID}$ be a leaf node associated with $ID$ such that $ID = Label(v_{ID})$. It obtains $PV_{ID}$ by running $\text{CS.Assign}(BT, v_{ID})$.

3. It finds $(S_i, S_i) = \text{CS.Match}(PV_{ID}, CV_T)$. It retrieves $SK_{IBBE, S_i}$ from $IU_{K_T}$ and obtains $M_3$ by running $\text{IBBE.Decrypt}(CT_{IBBE}, SK_{IBBE, S_i}, PP_{IBBE})$.

4. Finally, it outputs a message $M = M_1 \oplus M_2 \oplus M_3$. 

22
4.3 Correctness

In this section we show the correctness of the DRIBE-INC scheme. The ciphertext of our DRIBE-INC scheme consists of three elements: a DRIBE ciphertext, an HIBE ciphertext, and an IBBE ciphertext. First, if the identity ID of the DRIBE ciphertext is not revoked in a base update key, we can decrypt $M_1$ by the correctness of the DRIBE scheme. Then, if $ID$ and $T$ of the HIBE ciphertext and $ID$ and $T$ of an HIBE decryption key are the same, we can decrypt $M_2$ by the correctness of the HIBE scheme. Now, when the leaf node of a binary tree associated with $ID$ is not revoked in an incremental update key, a common node $v$ exists by the correctness of the CS scheme. At this time, the IBBE ciphertext sets path nodes as a set of recipients, and an update key consists of IBBE private keys associated with cover nodes. Because of this, the common node becomes an element in path nodes, so we can decrypt $M_3$ by the correctness of the IBBE scheme. Therefore, we can derive the original message by computing $M = M_1 \oplus M_2 \oplus M_3$.

4.4 Security Analysis

In this section, we show that our DRIBE-INC scheme provides the IND-CPA security, the IND-UKA security, and the UKV security by relying on the security of underlying cryptographic primitives.

Theorem 4.1. The above DRIBE-INC scheme is IND-CPA secure if the underlying DRIBE, HIBE, and IBBE schemes are IND-CPA secure.

Proof. Let $ID^*$ be the challenge identity and $T^*$ be the challenge time such that $T^* = T_b^* + T_c^*$. We divide the behavior of an adversary as three types: Type-A, Type-B, and Type-C. These types are defined as follows:

Type-A. An adversary is type-A if it queries a private key corresponding to $ID \neq ID^*$ for all private keys.

Type-B. An adversary is type-B if it queries a private key corresponding to $ID = ID^*$ and the private key of $ID^*$ is revoked at some time $T$ such that $T \leq T_b^*$.

Type-C. An adversary is type-C if it queries a private key corresponding to $ID = ID^*$ and the private key of $ID^*$ is revoked at some time $T$ such that $T_b^* < T \leq T^*$.

Let $E_i$ be the event that $A$ behaves like type-$i$ adversary. From Lemmas 4.2, 4.3, and 4.4 we obtain the following result

$$Adv_{\text{DRIBE-INC}}^{\text{IND-CPA}}(\lambda) \leq Pr[E_A] \cdot Adv_{\text{DRIBE-INC}}^{\text{IND-CPA}}(\lambda) + Pr[E_B] \cdot Adv_{\text{IBBE}}^{\text{IND-CPA}}(\lambda) + Pr[E_C] \cdot Adv_{\text{HIBE}}^{\text{IND-CPA}}(\lambda)$$

This completes our proof.

Lemma 4.2. For the type-A adversary, the DRIBE-INC scheme is IND-CPA secure if the HIBE scheme is IND-CPA secure.

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

Setup: $B$ obtains $\text{MSK}_{\text{DRIBE}}, \text{MU}_{\text{DRIBE}}, \text{PP}_{\text{DRIBE}}$ by running the DRIBE.Setup algorithm. It obtains $MK_{\text{IBBE}}, PP_{\text{IBBE}}$ by running the IBBE.Setup algorithm. It initializes $RL = \emptyset$ and gives $PP = (PP_{\text{DRIBE}}, PP_{\text{HIBE}}, PP_{\text{IBBE}}, H, BST)$ to $A$.  

23
Phase 1: \( \mathcal{A} \) adaptively requests a polynomial number of private key, revocation, base update key, incremental update key, and decryption key queries.

- For a private key query on \( ID, B \) proceeds as follows: It generates \( SK_{DRIBE} \) by using \( MSK_{DRIBE} \). Next, it receives \( SK_{HIBE} \) from \( C \) by querying a private key for \( ID \). It gives \( SK_{ID} = (SK_{DRIBE}, SK_{HIBE}) \) to \( \mathcal{A} \).

- For a revocation query on \( ID \) and \( T \), \( B \) proceeds as follows: It adds \( (ID, T) \) to \( RL \) if \( ID \) was not revoked before.

- For a base update key query on \( T_b, B \) proceeds as follows: It generates \( BUK_{T_b} \) by using \( MK_{DRIBE} \). It gives \( BUK_{T_b} \) to \( \mathcal{A} \).

- For an incremental update key query on \( T = T_b + T_c, B \) proceeds as follows: It generates \( IUK_T \) by using \( MK_{IBBE} \). It gives \( IUK_T \) to \( \mathcal{A} \).

- For a decryption key query on \( ID \) and \( T = T_b + T_c, B \) proceeds as follows: It generates \( DK_{DRIBE} \) for \( T_b \) by using \( MSK_{DRIBE} \) and \( MUK_{DRIBE} \). It receives \( DK_{HIBE} \) from \( C \) by querying a private key for \( ID \) and \( T \). Next, it generates \( IUK_T \) by using \( MK_{IBBE} \). It gives \( DK_{ID,T} = (DK_{DRIBE}, DK_{HIBE}, IUK_T) \) to \( \mathcal{A} \).

Challenge: \( \mathcal{A} \) submits a challenge identity \( ID^* \), challenge time \( T^* \), and two challenge messages \( M_0^*, M_1^* \) where \( T^* = T_b^* + T_c^* \). \( B \) proceeds as follows:

1. It selects random \( M_1, M_3 \) and sets \( M_{2,0} = M'_0 \oplus M_1 \oplus M_3, M_{2,1} = M'_1 \oplus M_1 \oplus M_3 \).

2. Next, it generates \( CT_{DRIBE}^* \) by running \( DRIBE.Encrypt(ID^*, T_b^*, M_1, PP_{DRIBE}) \). It receives \( CT_{HIBE}^* \) from \( C \) by submitting challenge \( ID^*, T^* \), and challenge messages \( M_{2,0}, M_{2,1}^* \).

3. It obtains \( PV^* \) by running \( CS.Assign(v^*) \) where \( v^* \) is associated with \( ID^* \). It sets \( S = \emptyset \). For each \( S_i \in PV_{ID} \), it adds a string \( H(Label(S_i) || T) \) to \( S \). It creates \( CT_{IBBE}^* \) by running \( BIBE.Encrypt(S, M_3, PP_{IBBE}) \).

4. It gives a challenge ciphertext \( CT^* = (CT_{DRIBE}^*, CT_{HIBE}^*, CT_{IBBE}^*) \) to \( \mathcal{A} \).

Phase 2: Same as Phase 1.

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

To complete the proof, we analyze the correctness of the simulation. For this, we show that it is possible to process the private key and decryption key query for the DRIBE-INC scheme by using the private key query for the HIBE scheme. Similar to analysis of Lemma 3.2 all private key and decryption key queries can be handled using the HIBE private key query by the restrictions of the type-A adversary and the restrictions of the decryption key queries in the DRIBE-INC security model. The challenge ciphertext is set so that the distinction of the DRIBE-INC challenge message is related with that of the HIBE challenge message.

Lemma 4.3. For the type-B adversary, the DRIBE-INC scheme is IND-CPA secure if the DRIBE scheme is IND-CPA secure.

Proof. Suppose there exists a type-B adversary \( \mathcal{A} \) that attacks the DRIBE-INC scheme with a non-negligible advantage. An algorithm \( \mathcal{B} \) that attacks the DRIBE scheme is initially given public parameters \( PP_{DRIBE} \) by a challenger \( C \). Then \( \mathcal{B} \) that interacts with \( \mathcal{A} \) is described as follows:

Setup: \( \mathcal{B} \) generates \( MK_{HIBE}, PP_{HIBE} \) by running the HIBE.Setup algorithm and generates \( MK_{IBBE}, PP_{IBBE} \) by running the IBBE.Setup algorithm. It initializes \( RL = \emptyset \) and gives \( PP = (PP_{DRIBE}, PP_{HIBE}, PP_{IBBE}, H, \mathcal{B}T) \) to \( \mathcal{A} \).

24
**Phase 1:** $\mathcal{A}$ adaptively requests a polynomial number of private key, revocation, base update key, incremental update key, and decryption key queries.

- For a private key query on $ID$, $\mathcal{B}$ proceeds as follows: It receives $SK_{DRIBE}$ from $\mathcal{C}$ by querying a private key with $ID$. It generates $SK_{HIBE}$ by using $MK_{HIBE}$. It gives $SK_{ID} = (SK_{DRIBE}, SK_{HIBE})$ to $\mathcal{A}$.
- For a revocation query on $ID$ and $T$, $\mathcal{B}$ proceeds as follows: It queries the revocation of $ID$ and $T$ to $\mathcal{C}$. Next, it adds $(ID, T)$ to $RL$ if $ID$ was not revoked before.
- For a base update key query on $T_b$, $\mathcal{B}$ proceeds as follows: It receives $UK_{DRIBE}$ from $\mathcal{C}$ by querying an update key with $T_b$. It gives $BUK_{T_b} = UK_{DRIBE}$ to $\mathcal{A}$.
- For an incremental update key query on $T$, $\mathcal{B}$ proceeds as follows: It generates $IUK_T$ by using $MK_{IBBE}$. It gives $IUK_T$ to $\mathcal{A}$.
- For a decryption key query on $ID$ and $T$, $\mathcal{B}$ proceeds as follows: It first receives $DK_{DRIBE}$ from $\mathcal{C}$ by querying a decryption key for $ID$ and $T_b$. It derives $DK_{HIBE}$ of $SK_{HIBE}$ by using $MK_{HIBE}$. Next, it generates $IUK_T$ by using $MK_{IBBE}$. It gives $DK_{ID,T} = (DK_{DRIBE}, DK_{HIBE}, IUK_T)$ to $\mathcal{A}$.

**Challenge:** $\mathcal{A}$ submits a challenge identity $ID^*$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$ where $T^* = T_b^* + T_c^*$. $\mathcal{B}$ proceeds as follows:

1. It select random $M_2, M_3$ and sets $M_{1,0} = M_0^* \oplus M_2 \oplus M_3, M_{1,1} = M_1^* \oplus M_2 \oplus M_3$.
2. Next, it receives $CT^*_{DRIBE}$ from $\mathcal{C}$ by submitting $ID^*, T_b^*$, and two challenge messages $M_{1,0}, M_{1,1}$. It obtains $CT^*_{HIBE}$ by running $HIBE.Encrypt(ID^*, T^*, M_2, PP_{HIBE})$.
3. It obtains $PV^*$ by running $CS.Assign(v^*)$ where $v^*$ is associated with $ID^*$. It sets $S = \emptyset$. For each $S_i \in PV^*$, it adds a string $H(Label(S_i)||T^*)$ to $S$. It creates $CT^*_{IBBE}$ by running $IBBE.Encrypt(S, M_3, PP_{IBBE})$.
4. It gives a challenge ciphertext $CT^* = (CT^*_{DRIBE}, CT^*_{HIBE}, CT^*_{IBBE})$ to $\mathcal{A}$.

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

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

To analyze the correctness of the simulation, we show that it is possible to process all DRIBE-INC queries of the adversary by using DRIBE queries. The private key and base update key queries are handled correctly using the private key and the update key of DRIBE, respectively. In the case of a type-B adversary, the decryption key query can be processed under the constraint that if the private key for $ID^*$ is queried, then the corresponding $ID^*$ is revoked before the time $T_b^*$. For reference, if you query the decryption key for $ID^*$ and $T \geq T_b^*$, then the simulator simply outputs $\perp$ as a decryption key since the decryption key for $ID^*$ and $T \geq T_b^*$ does not exist when $ID^*$ has already been revoked. The challenge ciphertext is set so that the DRIBE-INC challenge message is related to the DRIBE challenge message.

**Lemma 4.4.** For the type-C adversary, the DRIBE-INC scheme is IND-CPA secure if the IBBE scheme is IND-CPA secure.

**Proof.** Suppose there exists a type-C adversary $\mathcal{A}$ that attacks the DRIBE-INC scheme with a non-negligible advantage. An algorithm $\mathcal{B}$ that attacks the IBBE scheme is initially given public parameters $PP_{IBBE}$ by a challenger $\mathcal{C}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:
Setup: $B$ obtains $MSK_{DRIBE}, MUK_{DRIBE}, PP_{DRIBE}$ by running the DRIBE.Setup algorithm. It also obtains $MK_{HIBE}, PP_{HIBE}$ by running HIBE.Setup algorithm. It initializes $RL = \emptyset$ and gives $PP = (PP_{DRIBE}, PP_{HIBE}, PP_{IBBE}, H, BT)$ to $A$.

**Phase 1:** $A$ adaptively requests a polynomial number of private key, revocation, base update key, incremental update key, and decryption key queries.

- For a private key query on $ID$, $B$ proceeds as follows: It generates $SK_{DRIBE}$ by using $MSK_{DRIBE}$. Next, it generates $SK_{HIBE}$ by using $MK_{HIBE}$. It gives $SK_{ID} = (SK_{DRIBE}, SK_{HIBE})$ to $A$.
- For a revocation query on $ID$ and $T$, $B$ proceeds as follows: It adds $(ID, T)$ to $RL$ if $ID$ was not revoked before.
- For a base update key query on $T_b$, $B$ proceeds as follows: It generates $UK_{DRIBE}$ for $T_b$ by using $MUK_{DRIBE}$. It gives $BUK_{T_b} = UK_{DRIBE}$ to $A$.
- For an incremental update key query on $T = T_b + T_c$, $B$ proceeds as follows:
  1. It initializes $IRV = \emptyset$. For each $(ID_i, T_i) \in RL$, it adds $v_i$ of $ID_i$ into $IRV$ if $T_b < T_i \leq T$. It obtains $ICV_T$ by running CS.Cover$(BT, IRV)$.
  2. For each $S_i \in ICV_T$, it receives $SK_{IBBE, S_i}$ from $C$ by querying a private key for $H(Label(S_i)||T)$.
  3. It creates $IUK_T = \{ICV_T, \{SK_{IBBE, S_i}\}_{S_i \in ICV_T}\}$ and gives $IUK_T$ to $A$.
- For a decryption key query on $ID$ and $T = T_b + T_c$, $B$ proceeds as follows: It generates $DK_{DRIBE}$ by using $MSK_{DRIBE}$ and $MUK_{DRIBE}$. Next, it retrieves $IUK_T$ by querying an incremental update key to its own oracle. It generates a delegated key $DK_{HIBE}$ of $SK_{HIBE}$ by using $MK_{HIBE}$. It gives $DK_{ID, T} = (DK_{DRIBE}, DK_{HIBE}, IUK_T)$ to $A$.

**Challenge:** $A$ submits a challenge identity $ID^*$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$ where $T^* = T_b^* + T_c^*$. $B$ proceeds as follows:

1. It selects random $M_1, M_2$ and sets $M_{3,0} = M_0^* \oplus M_1 \oplus M_2, M_{3,1} = M_1^* \oplus M_1 \oplus M_2$.
2. Next, it generates $CT^*_{DRIBE}$ by running DRIBE.Encrypt$(ID^*, T_b^*, M_1, PP_{DRIBE})$. It generates $CT^*_{HIBE}$ by running HIBE.Encrypt$(\{ID^*, T^*\}, M_2, PP_{HIBE})$.
3. It obtains $PV^*$ by running CS.Assign$(BT, v^*)$ where a leaf node $v^*$ is associated with $ID^*$. It sets $S = \emptyset$. For each $S_i \in PV^*$, it adds $H(Label(S_i)||T^*)$ to $S$. It receives $CT_{IBBE}^*$ from $C$ by submitting a challenge set $S$ and challenge messages $M_{3,0}, M_{3,1}$.
4. It gives a challenge ciphertext $CT^* = (CT^*_{DRIBE}, CT^*_{HIBE}, CT^*_{IBBE})$ to $A$.

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

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

To analyze the correctness of the simulation, we show that it is possible to process all DRIBE-INC queries of the adversary by using IBBE queries. From the restriction of a type-C adversary, we have that the challenge identity $ID^*$ is revoked from an incremental update key at time $T^*$ if the private key for $ID^*$ is queried. The incremental update key at time $T^*$ is composed of the IBBE private key corresponding to the cover nodes of a binary tree, and the ciphertext for $ID^*$ and $T^*$ is an IBBE ciphertext that uses the path nodes for $ID^*$ as the receiver set. In this case, since the path nodes for $ID^*$ are removed from the
incremental update key at time $T^*$, the IBBE private key belonging to the recipient set is not created. Thus the incremental update key can be generated by using the IBBE private key queries. The IBBE decryption key query can also be generated since the incremental update key is correctly processed.

**Theorem 4.5.** The above DRIBE-INC scheme is IND-UKA secure if the HIBE scheme is IND-CPA secure.

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

**Setup:** $B$ generates $MSK_{DRIBE}, MU_{KDRIBE}, PP_{DRIBE}$ by running DRIBE.Setup algorithm. It also generates $MK_{IBBE}, PP_{IBBE}$ by running the IBBE.Setup algorithm. It initializes $RL = \emptyset$ and gives $MU_{K} = (MU_{KDRIBE}, MK_{IBBE}), PP = (PP_{DRIBE}, PP_{HIBE}, PP_{IBBE}, H, B_T)$ to $A$.

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

- For a private key query on $ID$, $B$ proceeds as follows: It generates $SK_{DRIBE}$ by using $MSK_{DRIBE}$. It receives $SK_{HIBE}$ from $C$ by querying a private key for $ID$. It gives $SK_{ID} = (SK_{DRIBE}, SK_{HIBE})$ to $A$.

- For a revocation query on $ID$ and $T$, $B$ updates $RL$ by running DRIBE.Revoke($ID, T, RL$).

- For a decryption key query on $ID$ and $T$, $B$ proceeds as follows: It receives $DK_{HIBE}$ from $C$ by querying a private key for $ID$ and $T$. It generates $IUK_T$ by using $MK_{IBBE}$. It gives $DK_{ID,T} = (DK_{DRIBE}, DK_{HIBE}, IUK_T)$ to $A$.

**Challenge:** $A$ submits a challenge identity $ID^*$, challenge time $T^*$, and two challenge messages $M_0^*, M_1^*$ where $T^* = T_b^* + T^*$. $B$ proceeds as follows:

1. It first select random $M_1, M_3 \in M$ and sets $M_{2,0} = M_0^* \oplus M_1 \oplus M_3, M_{2,1} = M_1^* \oplus M_1 \oplus M_3$.

2. Next, it generates $CT_{DRIBE}^*$ by running DRIBE.Encrypt($ID^*, T_b^*, M_1, PP_{DRIBE}$). It receives $CT_{HIBE}^*$ from $C$ by submitting challenge $ID^*, T^*$, and challenge $M_{2,0}, M_{2,1}$.

3. It obtains $PV^*$ by running CS.Assign($BT, v^*$) where a leaf node $v^*$ is associated with $ID^*$. It sets $S = \emptyset$. For each $S_i \in PV^*$, it adds $H(\text{Label}(S_i) || T^*)$ to $S$. It generates $CT_{IBBE}^*$ by running DRIBE.Encrypt($S, M_3, PP_{IBBE}$).

4. It gives a challenge ciphertext $CT^* = (CT_{DRIBE}^*, CT_{HIBE}^*, CT_{IBBE}^*)$ to $A$.

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

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

**Theorem 4.6.** The above DRIBE-INC scheme is update-key verifiable if the DRIBE scheme is update-key verifiable and the hash function is collision resistant.

**Proof.** Let $T^*$ be the time period where $T^* = T_b^* + T_1^*$ and $RL_b^*, RL_1^*$ be revocation lists. We define $BR_b^*$ as a set of identities $ID$ such that $(ID, T_i) \in RL_b^*$ and $T_i \leq T_b^*$ for each $b \in \{0, 1\}$. We also define $IR_b^*$ as a set of identities $ID$ such that $(ID, T_i) \in RL_b^*$ and $T_b^* < T_i \leq T^*$ for each $b \in \{0, 1\}$. We divide the behavior of an adversary as two types: Type-A and Type-B. These types are defined as follows:

**Type-A.** An adversary is type-A if $BR_0^* \neq BR_1^*$.

**Type-B.** An adversary is type-B if $BR_0^* = BR_1^*$ and $IR_0^* \neq IR_1^*$.
Suppose there exists a type-A adversary $A$ that attacks the DRIBE-INC scheme with a non-negligible advantage. An algorithm $B$ that attacks the DRIBE scheme handles all queries of $A$ by using the oracles of DRIBE and the master keys of HIBE and IBBE. If $A$ outputs update keys $BUK^*, IU^K*$, a time period $T^* = T_b + T_c$, and two revocation lists $RL_0^*, RL_1^*$ in the final step, $B$ simply outputs $BUK^*, T_b^*$, and $RL_0^*, RL_1^*$. Because $BR_0^* \neq BR_1^*$, $B$ succeeds to break the update-key verifiability of DRIBE.

Suppose there exists a type-B adversary $A$ that attacks the DRIBE-INC scheme with a non-negligible advantage. An algorithm $B$ that attacks the CRHF scheme handles all queries of $A$ by using the master keys of DRIBE, HIBE, and IBBE. Finally, $A$ outputs update keys $BUK^*, IU^K*$, a time period $T^*$, two revocation lists $RL_0^*, RL_1^*$ where $T^* = T_b^* + T_c^*$. The process of finding collisions is very similar to that of Theorem 3.5 $B$ proceeds as follows:

1. It first checks the validity of $BUK^*$ for $RL_0^*, RL_1^*$. It also checks the validity of $IU^K*$ for $RL_0^*, RL_1^*$.

2. It initializes $IRV_0^* = \emptyset$ and $IRV_1^* = \emptyset$. For each $(ID_i, T_i) \in RL_0^*$, it adds a leaf node $v_i$ of $ID_i$ into $IRV_0^*$ if $T_b^* < T_i \leq T^*$. For each $(ID_i, T_i) \in RL_1^*$, it adds a leaf node $v_i$ of $ID_i$ into $IRV_1^*$ if $T_b^* < T_i \leq T^*$. It checks that $IRV_0^* \neq IRV_1^*$.

3. It obtains $ICV_0^*$ and $ICV_1^*$ by calling $CS.Cover(BT, IRV_0^*)$ and $CS.Cover(BT, IRV_1^*)$ respectively. Next, it derives a common node set $A = ICV_0^* \cap ICV_1^*$. It computes two (non-empty) disjoint sets $B_0 = ICV_0^* \setminus A$ and $B_1 = ICV_1^* \setminus A$ since $ICV_0^* \neq ICV_1^*$.

4. Let $IU^K* = (ICV^*, \{SK_{IBBE, S_i}^*\}_{S_i \in ICV^*})$. It chooses a node $v_k \in B_0$ and retrieves a private key $SK_{IBBE, S_k}^*$ in $IU^K*$ which is associated with the node $v_k$. Note that the private key should exist since the update key is verified under $RL_0^*$.

5. For each $v_i \in B_1$, it proceeds as follows:

   (a) It chooses a random $M \in \mathcal{M}$ and obtains $CT_i$ by running $IBBE.Encrypt(H(\text{Label}(v_i)||T^*), M, PP_{IBBE})$.

   (b) It obtains $M'$ by running $IBBE.Decrypt(CT_i, SK_{IBBE, S_i}^*, PP_{IBBE})$. If $M = M'$, then it sets $v_j = v_i$ and breaks the loop. Note that the matching node $v_j$ should exist since the update key is verified under $RL_1^*$.

6. Finally, it outputs $\text{Label}(v_k)||T^*$ and $\text{Label}(v_j)||T^*$ as the collision of the hash function since $v_k \neq v_j$.

This completes our proof.

4.5 Discussions

**Efficiency Analysis.** The private key of our DRIBE-INC scheme is compact because it is composed of a DRIBE private key which is an HIBE private key and an additional HIBE private key. The ciphertext is also compact because it consists of a DRIBE ciphertext, an HIBE ciphertext, and an IBBE ciphertext. The base update key consists of a DRIBE update key and an incremental update key consists of many IBBE private keys. For concrete analysis of the update key size, the number of revoked users $r$ in the base update key and the number of revoked users $s$ in the incremental update key must be determined. We set the maximum number of users in the system to $2^{30}$, the security parameter to 80 bits, and the hash length to 160 bits. For comparison, consider a scenario in which an RIBE scheme issues a daily update key, and the DRIBE-INC scheme issues a base update key once a month and a daily incremental update key. In this case, we compare
the update key size of the RIBE scheme and the DRIBE-INC scheme. The update key of the RIBE scheme which is generated every day consists of $r \log_2 30 = 30r$ binary tree nodes. The base update key of the DRIBE-INC scheme which is generated once a month consists of $r \log_2 160 = 160r$ binary tree nodes. The incremental update key of the DRIBE-INC scheme is $s \log_2 160 = 160s = 160(r/10) = 16r$ binary tree nodes if we simply set $s = r/10$. Therefore, the amortized size of the update key of the DRIBE-INC scheme is $160r/30 + 16r = 21.3r$ binary tree nodes, and it is similar to that of the RIBE scheme.

5 Instantiations

In this section, we instantiate our DRIBE scheme by using bilinear groups or lattices.

5.1 DRIBE from Bilinear Maps

We instantiate our DRIBE scheme using pairing-based encryption schemes. First, we consider realizing a DRIBE scheme that provides the full-model security. For this, we select the HIBE scheme of Waters [35] that provides full-model security under the standard assumption. We select the IBBE scheme of Gentry and Waters [16] which has a constant-size ciphertext and a short private key because it provides the full-model security in the random oracle model. For a cryptographic hash function, we select the SHA256 scheme and truncate the hash output if necessary. Next, we consider realizing a DRIBE scheme that provides the selective security. First, we select the efficient HIBE scheme of Boneh and Boyen [4] which provides the selective security under the standard assumption. Alternatively, if we want to reduce the ciphertext size, we can select the HIBE scheme of Boneh et al. [5] with constant size ciphertext. We select the IBBE scheme of Delerablee [13] which has a constant size ciphertext and a short private key. We also select the SHA256 scheme as a hash function.

5.2 DRIBE from Lattices

We instantiate our DRIBE scheme using lattice-based encryption schemes. For a lattice HIBE scheme, we select the efficient HIBE scheme of Agrawal et al. [1] that provides the selective security under the LWE assumption. For a lattice IBBE scheme, we try to use the IBBE scheme of Brakerski and Vaikuntanathan [7] which is derived from their ciphertext-policy attribute-based encryption scheme. One drawback of this IBBE scheme is that it lacks the formal security analysis. Another alternative for the IBBE scheme is to use the fixed-dimension HIBE scheme of Agrawal et al. [2] with a short ciphertext to replace the IBBE scheme. In this case, the IBBE encryption is replaced by the HIBE encryption on the path string of a leaf node, and the IBBE key generation is replaced by the HIBE key generation for cover nodes in a binary tree. The IBBE decryption is possible by using the private key delegation of the HIBE scheme. Finally, we select the SHA256 scheme and truncate the hash output if necessary.

6 Conclusion

In this paper, we introduced the concept of DRIBE that delegates the generation of update keys to a cloud server, and proposed an efficient DRIBE scheme by generically combining an HIBE scheme, an IBBE scheme, and a hash function. Our proposed DRIBE scheme satisfies not only the IND-CPA and IND-UKA security but also our newly defined UKV security that guarantees the unambiguity of update keys. In addition, we proposed an DRIBE-INC scheme that supports incremental update keys to reduce the update
key size of our DRIBE scheme. Our DRIBE-INC scheme has the effect of reducing the overall update key size by issuing a large-sized base update key occasionally and a small-sized incremental update key periodically.

References


