Self-Updatable Encryption: Time Constrained Access Control with Hidden Attributes and Better Efficiency

Kwangsu Lee[*] Seung Geol Choi[†] Dong Hoon Lee[‡] Jong Hwan Park[§] Moti Yung[¶]

Abstract

Revocation and key evolving paradigms are central issues in cryptography, and in PKI in particular. A novel concern related to these areas was raised in the recent work of Sahai, Seyalioglu, and Waters (Crypto 2012) who noticed that revoking past keys should at times (e.g., the scenario of cloud storage) be accompanied by revocation of past ciphertexts (to prevent unread ciphertexts from being read by revoked users). They introduced revocable-storage attribute-based encryption (RS-ABE) as a good access control mechanism for cloud storage. RS-ABE protects against the revoked users not only the future data by supporting key-revocation but also the past data by supporting ciphertext-update, through which a ciphertext at time T can be updated to a new ciphertext at time $T+1$ *using only the public key*. Motivated by this pioneering work, we ask whether it is possible to have a modular approach, which includes a primitive for time managed ciphertext update as a primitive. We call encryption which supports this primitive a "self-updatable encryption" (SUE). We then suggest a modular cryptosystems design methodology based on three sub-components: a primary encryption scheme, a key-revocation mechanism, and a time-evolution mechanism which controls the ciphertext self-updating via an SUE method, coordinated with the revocation (when needed). Our goal in this is to allow the self-updating ciphertext component to take part in the design of new and improved cryptosystems and protocols in a flexible fashion. Specifically, we achieve the following results:

- We first introduce a new cryptographic primitive called *self-updatable encryption (SUE)*, realizing a time-evolution mechanism. In SUE, a ciphertext and a private key are associated with time. A user can decrypt a ciphertext if its time is earlier than that of his private key. Additionally, *anyone (e.g., a cloud server) can update the ciphertext* to a ciphertext with a newer time. We also construct an SUE scheme and prove its full security under static assumptions.
- Following our modular approach, we present a new RS-ABE scheme with shorter ciphertexts than that of Sahai et al. and prove its security. The length efficiency is mainly due to our SUE scheme and the underlying modularity.
- We apply our approach to predicate encryption (PE) supporting attribute-hiding property, and obtain a revocable-storage PE (RS-PE) scheme that is selectively-secure.
- We further demonstrate that SUE is of independent interest, by showing it can be used for timed-release encryption (and its applications), and for augmenting key-insulated encryption with forward-secure storage.

Keywords: Public-key encryption, Attribute-based encryption, Predicate encryption, Self-updatable encryption, Revocation, Key evolving systems, Cloud storage.

[*]Korea University, Korea. Email: guspin@korea.ac.kr. This work was partially done at Columbia University.
[†]US Naval Academy, USA. Email: choi@usna.edu. This work was partially done at Columbia University.
[‡]Korea University, Korea. Email: donghlee@korea.ac.kr.
[§]Korea University, Korea and Sangmyung University, Korea. Email: decartian@korea.ac.kr.
[¶]Google Inc. and Columbia University, USA. Email: moti@cs.columbia.edu.

Contents

1 Introduction

Cloud data storage has many advantages: A virtually unlimited amount of space can be flexibly allocated with very low costs, and storage management, including back-up and recovery, has never been easier. More importantly, it provides great accessibility: users in any geographic location can access their data through the Internet. However, when an organization is to store *privacy-sensitive data*, existing cloud services do not seem to provide a good security guarantee yet (since the area is in its infancy). In particular, access control is one of the greatest concerns, that is, the sensitive data items have to be protected from any illegal access, whether it comes from outsiders or even from insiders without proper access rights.

One possible approach for this problem is to use attribute-based encryption (ABE) that provides cryptographically enhanced access control functionality in encrypted data [19, 26, 41]. In ABE, each user in the system is issued a private key from an authority that reflects their attributes (or credentials), and each ciphertext specifies access to itself as a boolean formula over a set of attributes. A user will be able to decrypt a ciphertext if the attributes associated with their private key satisfy the boolean formula associated with the ciphertext. To deal with the change of user's credentials that takes place over time, revocable ABE (R-ABE) [3] has been suggested, in which a user's private key can be revoked. In R-ABE, a key generation authority uses broadcast encryption to allow legitimate users to update their keys. Therefore, a revoked user cannot learn any partial information about the messages encrypted when the ciphertext is created after the time of revocation (or after the user's credential has expired).

As pointed out by Sahai, Seyalioglu, and Waters [40], R-ABE alone does not suffice in managing dynamic credentials for cloud storage. In fact, R-ABE cannot prevent *a revoked user from accessing ciphertexts that were created before the revocation*, since the old private key of the revoked user is enough to decrypt these ciphertexts. To overcome this, they introduced a novel revocable-storage ABE (RS-ABE) which solves this issue by supporting not only the revocation functionality but also the ciphertext update functionality such that a ciphertext at any arbitrary time T can be updated to a new ciphertext at time $T+1$ by any party *just using the public key* (in particular, by the cloud servers).

Key-revocation and key evolution are general sub-area in cryptosystems design, and ciphertext-update is a new concern which may be useful elsewhere. So, in this paper, we ask natural questions:

> Can we achieve key-revocation and ciphertext-update in other encryption schemes? Can we use ciphertext-update as an underlying primitive by itself?

We note that, in contrast to our questions, the methodology that Sahai et al. [40] used to achieve ciphertext-update is customized to the context of ABE. In particular, they first added ciphertext-delegation to ABE, and then, they *represented time as a set of attributes*, and by doing so they reduced ciphertext-update to ciphertext-delegation.

1.1 Our Results

We address the questions by taking a modular approach, that is, by actually constructing a cryptographic component realizing each of the two functionalities: key revocation and ciphertext update. In particular, our design approach is as follows:

- The overall system has three components: a primary encryption scheme (i.e., ABE or some other encryption scheme), a key-revocation mechanism, and a time-evolution mechanism.

- We combine the components by putting the key-revocation mechanism in the center and connecting it with the other two. This is because the revoked users need to be taken into account both in the decryption of the primary scheme and in the time-evolution of ciphertexts.

There are a few potential benefits to this approach. First, we may be able to achieve key-revocation and time-evolution mechanisms, *independently of the primary encryption scheme*. Secondly, each mechanism may be of independent interest and be used in other interesting scenarios. Thirdly, looking at each mechanism alone may open the door to various optimizations and flexibilities of implementations.

Time-Evolution Mechanism: Self-Updatable Encryption. We first formulate a new cryptographic primitive called *self-updatable encryption (SUE)*, realizing a time-evolution mechanism. In SUE, a ciphertext and a private key are associated with time T_c and T_k respectively. A user who has a private key with time T_k can decrypt the ciphertext with time T_c if $T_c \leq T_k$. Additionally, *anyone can update the ciphertext* with time T_c to a new ciphertext with new time T_c' such that $T_c < T_c'$. We construct an SUE scheme in composite order bilinear groups. In our SUE scheme, a ciphertext consists of $O(\log T_{max})$ group elements, and a private key consists of $O(\log T_{max})$ group elements, where T_{max} is the maximum time period in the system. Our SUE scheme is fully secure under static assumptions by using the dual system encryption technique of Waters [27,45].

RS-ABE with Shorter Ciphertexts. Following the general approach above, we construct a new RS-ABE scheme and prove that it is fully secure under static assumptions. In particular, we take the ciphertext-policy ABE (CP-ABE) scheme of Lewko et al. [26] as the primary encryption scheme, and combine it with our SUE scheme and a revocation mechanism. The revocation mechanism follows the design principle of Boldyreva, Goyal, and Kumar [3] that uses the complete subtree method to securely update the keys of the non-revoked users. Compared with the scheme of Sahai et al. [40], our scheme has a shorter ciphertext length consisting of $O(l + \log T_{max})$ groups elements where l is the size of row in the ABE access structure; a ciphertext in their scheme consists of $O(l \log T_{max} + \log^2 T_{max})$ group elements (reflecting the fact that time is dealt with in a less modular fashion there, while we employ the more separated SUE component which is length efficient).

Revocable-Storage Predicate Encryption. We apply our approach to predicate encryption (PE) and give the first RS-PE scheme. In particular, taking the PE scheme of Park [36] as the primary encryption scheme, we combine it with the same revocation functionality and (a variant of) our SUE scheme. The scheme is in prime-order groups and is shown to be selectively secure (a previously used weaker notion than (full) security, where the adversary selects the target of attack at the start). Obviously, compared with the RS-ABE scheme, the RS-PE scheme is a PE system and, thus, additionally supports the attribute-hiding property: even a decryptor cannot obtain information about the attributes x of a ciphertext except $f(x)$, where f is the predicate of its private key.

Other Systems. These are discussed below in this section.

1.2 Our Technique

To devise our SUE scheme, we use a full binary tree structure to represent time. The idea of using the full binary tree for time was already used by Canetti et al. [10] to construct a forward-secure public-key encryption (FSE) scheme. However, our scheme greatly differs on a technical level from their approach; in our scheme, *a ciphertext is updated* from time T_i to time $T_j > T_i$, whereas in their scheme *a private key is updated* from time T_i to time $T_j > T_i$. We start from the HIBE scheme of Boneh and Boyen [4], and then construct a *ciphertext delegatable encryption (CDE)* scheme, by switching the structure of private keys with that of ciphertexts; our goal is to support ciphertext delegation instead of private key delegation. In CDE, each ciphertext is associated with a tree node, so is each private key. A ciphertext at a tree node v_c can be decrypted by any keys with a tree node v_k where v_k is a descendant (or self) of v_c. We note that the CDE scheme may be of independent interest. The ciphertext delegation property of CDE allows us to construct an SUE scheme. An SUE ciphertext at time T_i consists of multiple CDE ciphertexts in order to support ciphertext-update for every T_j such that $T_j > T_i$. We were able to reduce the number of group elements in the SUE ciphertext *by carefully reusing the randomness of CDE ciphertexts*.

Our key-revocation mechanism, as mentioned above, uses a symmetric-key broadcast encryption scheme to periodically broadcast update keys to non-revoked users. A set of non-revoked users is represented as a node (more exactly the leaves of the subtree rooted at the node) in a tree, following the complete subset (CS) scheme of Naor et al. [32]. So, we use two different trees in this paper, i.e., one for representing time in the ciphertext domain, and the other for managing non-revoked users in the key domain.

In the RS-ABE/RS-PE setting, a user u who has a private key with attributes x and an update key with a revoked set R at time T' can decrypt a ciphertext with a policy f and time T if the attribute satisfies the policy $(f(x) = 1)$ and the user is not revoked $(u \notin R)$, and $T \leq T'$. The main challenge in combining all the components is protecting the overall scheme against a collusion attack, e.g., a non-revoked user with a few attributes should not decrypt more ciphertexts than he is allowed to, given the help of a revoked user with many attributes. To achieve this, we use a secret sharing scheme as suggested in [3]. Roughly speaking, the overall scheme is associated with a secret key α. For each node v_i in the revocation tree, this secret key α is split into γ_i for ABE/PE, and $\alpha - \gamma_i$ for SUE, where γ_i is random. Initially, each user will have some tree nodes v_is according to the revocation mechanism, and get ABE/PE private keys subject to his attributes at each of v_is (associated with the ABE/PE master secret γ_i). In key-update at time T, only non-revoked users receive SUE private keys with time T at a tree node v_j representing a set of non-revoked users (associated with the SUE master secret $\alpha - \gamma_j$). Now consider the collusion scenario above: A ciphertext of message M at T of the overall scheme contains an element $e(g,g)^{\alpha s} \cdot M$ where g is a generator, and s is random. The non-revoked user will use a tree node v_j and obtain the SUE decryption part, i.e., $e(g,g)^{(\alpha-\gamma_j)s}$, but not the ABE/PE part due to lack of attributes. The revoked user doesn't have node v_j (recall v_j represents a set of non-revoked users), so he obtains nothing at v_j and, thus, provides no help to the non-revoked user.

1.3 Other Applications

Timed-Release Encryption. One application of SUE is timed-release encryption (TRE) and its variants [38, 39]. TRE is a specific type of PKE such that a ciphertext specified with time T can only be decrypted after time T. In TRE, a semi-trusted time server periodically broadcasts a time instant key (TIK) with time T' to all users. A sender creates a ciphertext by specifying time T, and a receiver can decrypt the ciphertext if he has a TIK with time T' such that $T' \geq T$. TRE can be used for electronic auctions, key escrow, on-line gaming, and press releases. TRE and its variants can be realizable by using IBE, certificateless encryption, or forward-secure PKE (FSE) [12, 38]. If we use a FSE scheme derived from the HIBE scheme of Boneh et al. [6] for a TRE scheme, then a ciphertext consists of $O(1)$ group element and a TIK consists of $O(\log^2 T_{max})$ group elements. An SUE scheme can be used for a TRE scheme with augmented properties, since a ciphertext with time T can be decrypted by a private key with time $T' \geq T$ from using the ciphertext update functionality, and, in addition, we have flexibility of having a public ciphertext server which can tune the ciphertext time forward before final public release. In this scheme, a ciphertext consists of $O(\log T_{max})$ and a TIK consists of $O(\log T_{max})$. TRE, in turn, can help in designing synchronized protocols, like fair exchanges in some mediated but protocol-oblivious server model.

Key-Insulated Encryption with Ciphertext Forward Security. SUE can be used to enhance the security of key-insulated encryption (KIE) [14]. KIE is a type of PKE that additionally provides tolerance against key exposures. For a component of KIE, a master secret key MK is stored on a physically secure device, and a temporal key SK_T for time T is stored on an insecure device. At a time period T, a sender encrypts a message with the time T and the public key PK, and then a receiver who obtains SK_T by interacting with the physically secure device can decrypt the ciphertext. KIE provides the security of all time periods except those in which the compromise of temporal keys occurred. KIE can be obtained from IBE. Though KIE provides strong level of security, it does not provide security of ciphertexts available in compromised time periods, even if these ciphertexts are to be read in a future time period. To enhance the security and prevent this premature disclosure, we can build a KIE scheme with forward-secure storage by combining KIE and SUE schemes. Having cryptosystems with key-insulated key and forward-secure storage is different from intrusion-resilient cryptosystems [13, 20].

1.4 Related Work

Attribute-Based Encryption. As mentioned, ABE extends IBE, such that a ciphertext is associated with an attribute x and a private key is associated with an access structure f. When a user has a private key with f, only then he can decrypt a ciphertext with x that satisfies $f(x) = 1$. Sahai and Waters [41] introduced fuzzy IBE (F-IBE) that is a special type of ABE. Goyal et al. [19] proposed a key-policy ABE (KP-ABE) scheme that supports flexible access structures in private keys. Bethencourt et al. [2] proposed a ciphertext-policy ABE (CP-ABE) scheme such that a ciphertext is associated with an access structure f and a private key is associated with an attribute x. After that, numerous ABE schemes with various properties were proposed [11, 26, 28–30, 35, 46]. Recently, an ABE scheme for general circuits was proposed [15].

Predicate Encryption. PE is also an extension of IBE that additionally provides an attribute-hiding property in ciphertexts: A ciphertext is associated with an attribute x and a private key is associated with a predicate f. A user who has a private key associated with f can decrypt a ciphertext with x if $f(x) = 1$. In this case, the user cannot obtain information about the attribute x except the information $f(x)$. Boneh and Waters [9] introduced the concept of PE and proposed a hidden vector encryption (HVE) scheme that supports conjunctive queries on encrypted data. Katz et al. [22, 23] proposed a PE scheme that supports inner-product queries on encrypted data. After that, many PE schemes with different properties were proposed [24, 33, 34, 36, 37, 44]. Boneh, Sahai, and Waters [8] formalized the concept of functional encryption (FE) by generalizing ABE and PE. Recently, FE schemes for general circuits were proposed [17, 18].

Revocation. Boneh and Franklin [7] proposed a revocation method for IBE that periodically re-issues the private key of users. That is, the identity ID of a user contains time information, and a user cannot obtain a valid private key for new time from a key generation center if he is revoked. However, this method requires for all users to establish secure channels to the server and prove their identities every time. To solve this problem, Boldyreva et al. [3] proposed an R-IBE scheme by combining an F-IBE scheme and a full binary tree structure. Libert and Vergnaud [31] proposed a fully secure R-IBE scheme. Recently, Seo and Emura [42, 43] proposed an R-IBE scheme that is fully secure when an intermediate key that is used for decryption is leaked and a revocable HIBE (R-HIBE) scheme that supports the functionalities of revocation, and private key delegation.

2 Preliminaries

In this section, we define the notation of this paper, and define full binary trees and the subset cover framework that are used for our schemes.

2.1 Notation

We let λ be a security parameter. Let $[n]$ denote the set $\{1,\dots,n\}$ for $n \in \mathbb{N}$. For a string $L \in \{0,1\}^n$, let $L[i]$ be the ith bit of L, and $L|_i$ be the prefix of L with i-bit length. For example, if $L = 010$, then $L[1] = 0, L[2] = 1, L[3] = 0$, and $L|_1 = 0, L|_2 = 01, L|_3 = 010$. Concatenation of two strings L and L' is denoted by $L \| L'$.

2.2 Full Binary Tree

A full binary tree $\mathcal{BT}$ is a tree data structure where each node except the leaf nodes has two child nodes. Let N be the number of leaf nodes in $\mathcal{BT}$. The number of all nodes in $\mathcal{BT}$ is $2N-1$. For any index $0 \leq i < 2N-1$, we denote by v_i a node in $\mathcal{BT}$. We assign the index 0 to the root node and assign other indices to other nodes by using breadth-first search. That is, if a node v has an index i, then the index of its left child node is $2i+1$ and the index of its right child node is $2i+2$, while the index of its parent node (if any) is $\lfloor \frac{i-1}{2} \rfloor$. The depth of a node v_i is the length of the path from the root node to the node. The root node is at depth zero. The depth of $\mathcal{BT}$ is the depth of a leaf node. A level of $\mathcal{BT}$ is a set of all nodes at given depth. Siblings are nodes that share the same parent node.

For any node $v_i \in \mathcal{BT}$, L is defined as a label that is a fixed and unique string. The label of each node in the tree is assigned as follows: Each edge in the tree is assigned with 0 or 1 depending on whether the edge is connected to its left or right child node. The label L of a node v_i is defined as the bitstring obtained by reading all the labels of edges in the path from the root node to the node v_i. Note that we assign a special empty string to the root node as a label. We define $ID(i)$ be a mapping from the index i of a node v_i to a label L. Note that there is a simple mapping between the index i and the label L of a node v_i such that $i = (2^d - 1) + \sum_{j=0}^{d-1} 2^j L[j]$ where d is the depth of v_i. We also use $ID(v_i)$ as $ID(i)$ if there is no ambiguity.

2.3 Subset Cover Framework

The subset cover (SC) framework introduced by Naor, Naor, and Lotspiech [32] is a general methodology for the construction of efficient revocation systems. Naor et al. proposed efficient revocation systems such that a center can send an encrypted message to non-revoked set of users by combining the SC framework and symmetric-key encryption schemes [32]. The SC framework consists of the subset-assigning part and key-assigning part for the subset. We define the SC scheme by including only the subset-assigning part.

In the SC scheme, a collection $\mathcal{S}$ is first defined as a set of subsets $S_1,\dots,S_w$ for some w such that $S_i \subseteq \mathcal{N}$ where $\mathcal{N}$ is the set of all users. Each user $u \in \mathcal{N}$ in the system is assigned a private set PV_u of subsets S_i that are associated with u. A sender finds a covering set CV_R that is the partition of non-revoked users $\mathcal{N} \setminus R$ into disjoint subsets where R is the set of revoked users. If a user u is not revoked, then he can find matching subsets from the covering set CV_R and the private set PV_u. The following is the syntax of the SC scheme.

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

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

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

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

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

*The correctness property of SC is defined as follows: For all $\mathcal{S}$ generated by **Setup**, all PV_u generated by **Assign**$(\mathcal{S},u)$ for any u, and all CV_R generated by **Cover**$(\mathcal{S},R)$ for any R, it is required that:*

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

We use the complete subset (CS) scheme proposed by Naor et al. [32] as a building block for our schemes. The CS scheme uses a full binary tree $\mathcal{BT}$ to define the subsets S_i. For any node $v_i \in \mathcal{BT}$, $\mathcal{T}_i$ is defined as a subtree that is rooted at v_i and S_i is defined as the set of leaf nodes in $\mathcal{T}_i$. For the tree $\mathcal{BT}$ and a subset R of leaf nodes, $ST(\mathcal{BT},R)$ is defined as the Steiner Tree induced by the set R and the root node, that is, the minimal subtree of $\mathcal{BT}$ that connects all the leaf nodes in R and the root node. we simply denote $ST(\mathcal{BT},R)$ by $ST(R)$. The CS scheme is described as follows:

CS.Setup(N_{max}): This algorithm takes as input the maximum number of users N_{max}. Let $N_{max} = 2^d$ for simplicity. It first sets a full binary tree $\mathcal{BT}$ of depth d. Each user is assigned to a different leaf node in $\mathcal{BT}$. The collection $\mathcal{S}$ of CS is $\{S_i : v_i \in \mathcal{BT}\}$. Recall that S_i is the set of all the leaves in the subtree $\mathcal{T}_i$. It outputs the full binary tree $\mathcal{BT}$.

CS.Assign($\mathcal{BT},u$): This algorithm takes as input the tree $\mathcal{BT}$ and a user $u \in \mathcal{N}$. Let v_u be the leaf node of $\mathcal{BT}$ that is assigned to the user u. Let $(v_{j_0},v_{j_1},\dots,v_{j_d})$ be the path from the root node $v_{j_0} = v_0$ to the leaf node $v_{j_n} = v_u$. It sets $PV_u = \{S_{j_0},\dots,S_{j_d}\}$, and outputs the private set PV_u.

CS.Cover($\mathcal{BT},R$): This algorithm takes as input the tree $\mathcal{BT}$ and a revoked set R of users. It first computes the Steiner tree $ST(R)$. Let $\mathcal{T}_{i_1},\dots\mathcal{T}_{i_m}$ be all the subtrees of $\mathcal{BT}$ that hang off $ST(R)$, that is all subtrees whose roots $v_{i_1},\dots v_{i_m}$ are not in $ST(R)$ but adjacent to nodes of outdegree 1 in $ST(R)$. It outputs a covering set $CV_R = \{S_{i_1},\dots,S_{i_m}\}$.

CS.Match(CV_R,PV_u): This algorithm takes input as a covering set $CV_R = \{S_{i_1},\dots,S_{i_m}\}$ and a private set $PV_u = \{S_{j_0},\dots,S_{j_d}\}$. It finds a subset S_k such that $S_k \in CV_R$ and $S_k \in PV_u$. If there is such a subset, it outputs (S_k,S_k). Otherwise, it outputs $\perp$.

Lemma 2.2 ([32]). *Let N_{max} be the number of leaf nodes in a full binary tree and r be the size of a revoked set. In the CS scheme, the size of a private set is $O(\log N_{max})$ and the size of a covering set is at most $r\log(N_{max}/r)$.*

3 Self-Updatable Encryption

In this section, we introduce the concept of SUE and propose an SUE scheme in bilinear groups.

3.1 Definitions

Ciphertext Delegatable Encryption (CDE). Before introducing self-updatable encryption, we first introduce ciphertext delegatable encryption. Ciphertext delegatable encryption (CDE) is a special type of public-key encryption (PKE) with the ciphertext delegation property such that a ciphertext can be easily converted to a new ciphertext under a more restricted label string by using public values. In CDE, the private key of a user is associated with a label string L' and a ciphertext is also associated with a label string L. If L is a prefix of L', then a user who has a private key with the label string L' can decrypt a ciphertext with the label string L. Additionally, the CDE scheme has the ciphertext delegation algorithm that convert a ciphertext with a label string L to a new ciphertext with a label string L'' such that L is a prefix of L'' by using public parameters. The following is the syntax of CDE.

Definition 3.1 (Ciphertext Delegatable Encryption)**.** *A ciphertext delegatable encryption (CDE) scheme for the set $\mathcal{L}$ of labels consists of seven PPT algorithms **Init**, **Setup**, **GenKey**, **Encrypt**, **DelegateCT**, **RandCT**, and **Decrypt**, which are defined as follows:*

***Init**(1^λ). The initialization algorithm takes as input a security parameter 1^λ, and it outputs a group description string GDS.*

***Setup**(GDS, d_{max}). The setup algorithm takes as input a group description string GDS and the maximum length d_{max} of the label strings, and it outputs public parameters PP and a master secret key MK.*

***GenKey**(L, MK, PP). The key generation algorithm takes as input a label string $L \in \{0,1\}^k$ with $k \leq d_{max}$, the master secret key MK, and the public parameters PP, and it outputs a private key SK_L.*

***Encrypt**(L, PP). The encryption algorithm takes as input a label string $L \in \{0,1\}^d$ with $d \leq d_{max}$ and the public parameters PP, and it outputs a ciphertext header CH_L and a session key EK.*

***DelegateCT**(CH_L, c, PP). The ciphertext delegation algorithm takes as input a ciphertext header CH_L for a label string $L \in \{0,1\}^d$ with $d < d_{max}$, a bit value $c \in \{0,1\}$, and the public parameters PP, and it outputs a delegated ciphertext header $CH_{L'}$ for the label string $L' = L \| c$.*

***RandCT**(CH_L, PP). The ciphertext randomization algorithm takes as input a ciphertext header CH_L for a label string $L \in \{0,1\}^d$ with $d < d_{max}$ and the public parameters PP, and it outputs a re-randomized ciphertext header CH'_L and a partial session key EK'.*

***Decrypt**$(CH_L, SK_{L'}, PP)$. The decryption algorithm takes as input a ciphertext header CH_L, a private key $SK_{L'}$, and the public parameters PP, and it outputs a session key EK or the distinguished symbol $\perp$.*

*The correctness property of CDE is defined as follows: For all PP,MK generated by **Setup**, all L, L', any $SK_{L'}$ generated by **GenKey**, any CH_L and EK generated by **Encrypt** or **DelegateCT**, it is required that:*

- *If L is a prefix of L', then **Decrypt**$(CH_L, SK_{L'}, PP) = EK$.*
- *If L is not a prefix of L', then **Decrypt**$(CH_L, SK_{L'}, PP) = \perp$ with all but negligible probability.*

*Additionally, it requires that the ciphertext distribution of **RandCT** is statistically equal to that of **Encrypt**.*

The security property of CDE can be similarly defined as the security property of PKE with an additional consideration for the ciphertext delegation property. In the security game of this security property, an adversary is first given public parameters, and then he can adaptively obtain many private keys for label strings $\{L_i\}$. In the challenge step, the adversary submits a challenge label string L^*, and then he receives a challenge ciphertext header CH^* and a challenge session key EK_b where EK_b is a correct session key or a random session key. Finally, the adversary outputs a guess for the random coin b that is used to create the session key. If the private key queries of the adversary satisfy the non-trivial conditions and the guess is correct, then he wins the game. The following is the formal definition of the security.

Definition 3.2 (Security)**.** *The security property for CDE schemes is defined in terms of the indistinguishability under a chosen plaintext attack (IND-CPA). The security game for this property is defined as the following game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:*

1. ***Setup**: $\mathcal{C}$ runs **Init** and **Setup** to generate the public parameters PP and the master secret key MK, and it gives PP to $\mathcal{A}$.*
2. ***Query 1**: $\mathcal{A}$ may adaptively request a polynomial number of private keys for label strings $L_1, \ldots, L_{q'}$, and $\mathcal{C}$ gives the corresponding private keys $SK_{L_1}, \ldots, SK_{L_{q'}}$ to $\mathcal{A}$ by running **GenKey**(L_i, MK, PP).*
3. ***Challenge**: $\mathcal{A}$ outputs a challenge label string L^* subject to the following restrictions: For all label strings L_i of private key queries, it is required that L^* is not a prefix of L_i. $\mathcal{C}$ chooses a random bit $b \in \{0,1\}$ and computes a ciphertext header CH^* and a session key EK^* by running **Encrypt**(L^*, PP). If $b = 0$, then it gives CH^* and EK^* to $\mathcal{A}$. Otherwise, it gives CH^* and a random session key to $\mathcal{A}$.*
4. ***Query 2**: $\mathcal{A}$ may continue to request private keys for additional label strings $L_{q'+1}, \ldots, L_q$ subject to the same restrictions as before, and $\mathcal{C}$ gives the corresponding private keys to $\mathcal{A}$.*
5. ***Guess**: Finally $\mathcal{A}$ outputs a bit b'.*

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{CDE}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the game. A CDE scheme is fully secure under a chosen plaintext attack if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.

Remark 3.3. *In the above security game, it is not needed to explicitly describe **DelegateCT** since the adversary can run **DelegateCT** to the challenge ciphertext header by just using PP. Note that the use of **DelegateCT** does not violate the security game since the adversary only can request a private key query for L_i such that L_i is not a prefix of L^*.*

We can also define the selective security property of CDE schemes by weakening the above security of CDE schemes. In the selective security game of the selective security property, an adversary should submit a target challenge bit string L^* before he receives public parameters.

Self-Updatable Encryption (SUE). Self-updatable encryption (SUE) is a new type of PKE with the ciphertext updating property such that a time is associated with private keys and ciphertexts and a ciphertext with a time can be easily updatable to a new ciphertext with a future time. In SUE, the private key of a user is associated with a time T' and a ciphertext is also associated with a time T. If $T \leq T'$, then a user who has a private key with a time T' can decrypt a ciphertext with a time T. That is, a user who has a private key for a time T' can decrypt any ciphertexts attached a past time T such that $T \leq T'$, but he cannot decrypt a ciphertext attached a future time T such that $T' < T$. Additionally, the SUE scheme has the ciphertext update algorithm that updates the time T of a ciphertext to a new time $T+1$ by using public parameters. The following is the syntax of SUE.

Definition 3.4 (Self-Updatable Encryption). *A self-updatable encryption (SUE) scheme consists of seven PPT algorithms **Init**, **Setup**, **GenKey**, **Encrypt**, **UpdateCT**, **RandCT**, and **Decrypt**, which are defined as follows:*

***Init**(1^λ). The initialization algorithm takes as input a security parameter 1^λ, and it outputs a group description string GDS.*

***Setup**(GDS, T_{max}). The setup algorithm takes as input a group description string GDS and the maximum time T_{max}, and it outputs public parameters PP and a master secret key MK.*

***GenKey**(T, MK, PP). The key generation algorithm takes as input a time T, the master secret key MK, and the public parameters PP, and it outputs a private key SK_T.*

***Encrypt**(T, PP). The encryption algorithm takes as input a time T and the public parameters PP, and it outputs a ciphertext header CH_T and a session key EK.*

***UpdateCT**($CH_T, T+1, PP$). The ciphertext update algorithm takes as input a ciphertext header CH_T for a time T, a next time $T+1$, and the public parameters PP, and it outputs an updated ciphertext header CH_{T+1}.*

***RandCT**(CH_T, PP). The ciphertext randomization algorithm takes as input a ciphertext header CH_T for a time T and the public parameters PP, and it outputs an re-randomized ciphertext header CH'_T and a partial session key EK'.*

***Decrypt**($CH_T, SK_{T'}, PP$). The decryption algorithm takes as input a ciphertext header CH_T, a private key $SK_{T'}$, and the public parameters PP, and it outputs a session key EK or the distinguished symbol $\perp$.*

*The correctness property of SUE is defined as follows: For all PP,MK generated by **Setup**, all T, T', any $SK_{T'}$ generated by **GenKey**, and any CH_T and EK generated by **Encrypt** or **UpdateCT**, it is required that:*

- *If $T \le T'$, then **Decrypt**$(CH_T, SK_{T'}, PP) = EK$.*
- *If $T > T'$, then **Decrypt**$(CH_T, SK_{T'}, PP) = \perp$ with all but negligible probability.*

*Additionally, it requires that the ciphertext distribution of **RandCT** is statistically equal to that of **Encrypt**.*

Remark 3.5. *For the definition of SUE, we follow the syntax of key encapsulation mechanisms instead of following that of standard encryption schemes since the session key of SUE serves as the partial share of a real session key in other schemes.*

The security property of SUE can be similarly defined as the security property of PKE with additional consideration for the ciphertext updating property. In the security game of this security property, an adversary is first given public parameters, and then he can adaptively obtain many private keys for times. In the challenge step, the adversary submits a challenge time T^*, and then he receives a challenge ciphertext header CH^* and a challenge session key EK_b where EK_b is a correct session key or a random session key. Finally, the adversary outputs a guess for the random coin b that is used to create the session key. If the private key queries of the adversary satisfy the non-trivial conditions and the guess is correct, then he wins the game. The following is the formal definition of the security.

Definition 3.6 (Security). *The security property for SUE schemes is defined in terms of the indistinguishability under a chosen plaintext attack (IND-CPA). The security game for this property is defined as the following game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:*

1. ***Setup***: *$\mathcal{C}$ runs* ***Init*** *and* ***Setup*** *to generate the public parameters PP and the master secret key MK, and it gives PP to $\mathcal{A}$.*

2. ***Query 1***: *$\mathcal{A}$ may adaptively request a polynomial number of private keys for times $T_1, \ldots, T_{q'}$, and $\mathcal{C}$ gives the corresponding private keys $SK_{T_1}, \ldots, SK_{T_{q'}}$ to $\mathcal{A}$ by running* ***GenKey***(T_i, MK, PP).

3. ***Challenge***: *$\mathcal{A}$ outputs a challenge time T^* subject to the following restriction: For all times $\{T_i\}$ of private key queries, it is required that $T_i < T^*$. $\mathcal{C}$ chooses a random bit $b \in \{0,1\}$ and computes a ciphertext header CH^* and a session key EK^* by running* ***Encrypt***(T^*, PP). *If $b = 0$, then it gives CH^* and EK^* to $\mathcal{A}$. Otherwise, it gives CH^* and a random session key to $\mathcal{A}$.*

4. ***Query 2***: *$\mathcal{A}$ may continue to request private keys for additional times $T_{q'+1}, \ldots, T_q$ subject to the same restriction as before, and $\mathcal{C}$ gives the corresponding private keys to $\mathcal{A}$.*

5. ***Guess***: *Finally $\mathcal{A}$ outputs a bit b'.*

The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{SUE}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the game. A SUE scheme is fully secure under a chosen plaintext attack if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.

Remark 3.7. *In the above security game, it is not needed to explicitly describe* ***UpdateCT*** *since the adversary can run* ***UpdateCT*** *to the challenge ciphertext header by just using PP. Note that the use of* ***UpdateCT*** *does not violate the security game since the adversary only can request a private key query for T_i such that $T_i < T^*$.*

We can also define the selective security property of SUE schemes by weakening the above security of SUE schemes. That is, an adversary in the selective security game should submit a target challenge time T^* before he receives public parameters.

3.2 Bilinear Groups of Composite Order

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

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

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

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

3.3 Complexity Assumptions

We introduce three static assumptions in bilinear groups of composite order. These assumptions were introduced by Lewko and Waters [27] to prove the security of their IBE and HIBE schemes by using the dual system encryption technique, and were also used to prove the security of ABE schemes of Lewko et al. [26].

Assumption 1 (Subgroup Decision) Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The Assumption is that if the challenge tuple

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

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = X_1$ from $Z = Z_1 = X_1 R_1$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{A1}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $X_1 \in \mathbb{G}_{p_1}$ and $R_1 \in \mathbb{G}_{p_2}$.

Assumption 2 (General Subgroup Decision) Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The Assumption is that if the challenge tuple

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

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = X_2 Y_2$ from $Z = Z_1 = X_2 R_3 Y_2$ with more than a negligible advantage. The advantage of $\mathcal{B}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{A2}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $X_1, X_2 \in \mathbb{G}_{p_1}$, $R_1, R_2, R_3 \in \mathbb{G}_{p_2}$, and $Y_1, Y_2 \in \mathbb{G}_{p_3}$.

Assumption 3 (Composite Diffie-Hellman) Let $(N, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of composite order $N = p_1 p_2 p_3$. Let $g_{p_1}, g_{p_2}, g_{p_3}$ be generators of subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}$ respectively. The Assumption is that if the challenge tuple

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

are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = e(g_{p_1}, g_{p_1})^{ab}$ from $Z = Z_1 = e(g_{p_1}, g_{p_1})^c$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\mathbf{Adv}_{\mathcal{A}}^{A3}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b, c \in \mathbb{Z}_N$, and $R_1, R_2 \in \mathbb{G}_{p_2}$.

3.4 Design Principle

We use a full binary tree to represent time in our SUE scheme by assigning time periods to all tree nodes instead of assigning time periods to leaf nodes only. The use of binary trees to construct key-evolving schemes dates back to the work of Bellare and Miner [1], and the idea of using all tree nodes to represent time periods was introduced by Canetti, Halevi, and Katz [10]. They used a full binary tree for private key update in forward-secure PKE schemes, but we use the full binary tree for ciphertext update.

In the full binary tree $\mathcal{BT}$, each node v (internal node or leaf node) is assigned a unique time value by using pre-order tree traversal that recursively visits the root node, the left subtree, and the right subtree. That is, the root node is associated with 0 value and the right most leaf node is associated with $2^{d_{max}+1} - 2$ value where d_{max} is the maximum depth of the tree. Note that we use breadth-first search for index assignment, but we use pre-order traversal for time assignment. Let $\mathbf{Path}(v)$ be the set of path nodes from the root node to a node v, $\mathbf{RightSibling}(\mathbf{Path}(v))$[1] be the set of right sibling nodes of $\mathbf{Path}(v)$, and $\mathbf{TimeNodes}(v)$ be the set of nodes that consists of v and $\mathbf{RightSibling}(\mathbf{Path}(v))$ excluding the parent's path nodes. That is, $\mathbf{TimeNodes}(v) = \{v\} \cup \mathbf{RightSibling}(\mathbf{Path}(v)) \setminus \mathbf{Path}(\mathbf{Parent}(v))$. Intuitively, if we consider all subtrees corresponding to all the times that are greater than and equal to the time T by pre-order traversal, then $\mathbf{TimeNode}(v)$ contains the root of each subtrees where v is associated with T. Pre-order traversal has the

[1]Note that we have $\mathbf{RightSibling}(\mathbf{Path}(v)) = \mathbf{RightChild}(\mathbf{Path}(\mathbf{Parent}(v)))$ where $\mathbf{RightChild}(\mathbf{Path}(v))$ be the set of right child nodes of $\mathbf{Path}(v)$ and $\mathbf{Parent}(v)$ be the parent node of v.

property such that if a node v is associated with time T and a node v' is associated with time T', then we have

$$\mathbf{TimeNodes}(v) \cap \mathbf{Path}(v') \neq \varnothing \text{ if and only if } T \leq T'^{2}.$$

Thus if a ciphertext has the delegation property such that it's association can be changed from a node to its child node, then a ciphertext for the time T can be easily delegated to a ciphertext for the time T' such that $T \leq T'$ by providing the ciphertexts of its own and right sibling nodes of path nodes excluding path nodes. In this case, the number of ciphertexts can be small since the number of right sibling nodes is the logarithm of tree nodes.

For the construction of an SUE scheme that uses a full binary tree, we need a CDE scheme that has the ciphertext delegation property in the tree such that a ciphertext associated with a node can be converted to another ciphertext associated with its child node. Hierarchical identity-based encryption (HIBE) has the similar delegation property in the tree, but the private keys of HIBE can be delegated [4, 16]. To construct a CDE scheme that supports the ciphertext delegation property, we start from the HIBE scheme of Boneh and Boyen [4, 5] and exchange the private key structure with the ciphertext structure of their HIBE scheme. To use the structure of HIBE, we associate each node with a unique label string $L \in \{0,1\}^*$. That is, the root node is assigned an empty string, and a node assigned with a label L has a left child node assigned with a label $L\|0$ and a right child node assigned with a label $L\|1$. The ciphertext delegation property in CDE is easily obtained from the private-key delegation property of HIBE.

To build an SUE scheme from the CDE scheme, we define a mapping function ψ that maps time T to a label L in the tree nodes since these two scheme uses the same full binary tree. The SUE ciphertext for time T consists of all CDE ciphertexts for all nodes in $\mathbf{TimeNodes}(v)$ where time T is associated with a node v. Although the ciphertext of SUE just consists of $O(\log T_{max})$ number of CDE ciphertexts, the ciphertext of SUE can be $O(\log^2 T_{max})$ group elements since the ciphertext of a naive CDE scheme from the HIBE scheme has $O(\log T_{max})$ number of group elements. To improve the efficiency of the ciphertext size, we use the randomness reuse technique for CDE ciphertexts. In this case, we obtain an SUE scheme with $O(\log T_{max})$ group elements in ciphertexts.

3.5 Construction

Our CDE scheme is described as follows:

CDE.Init(1^λ): This algorithm takes as input a security parameter 1^λ. It generates a bilinear group $\mathbb{G}$ of composite order $N = p_1 p_2 p_3$ where p_1, p_2, and p_3 are random primes. It chooses a random generator $g_1 \in \mathbb{G}_{p_1}$ and outputs a group description string as $GDS = ((N, \mathbb{G}, \mathbb{G}_T, e), g_1, p_1, p_2, p_3)$.

CDE.Setup(GDS, d_{max}): This algorithm takes as input the string GDS and the maximum length d_{max} of the label strings. Let $l = d_{max}$. It chooses random elements $w, \{u_{i,0}, u_{i,1}\}_{i=1}^{l}, \{h_{i,0}, h_{i,1}\}_{i=1}^{l} \in \mathbb{G}_{p_1}$, a random exponent $\beta \in \mathbb{Z}_N$, and a random element $Y \in \mathbb{G}_{p_3}$. We define $F_{i,b}(L) = u_{i,b}^{L} h_{i,b}$ where $i \in [l]$ and $b \in \{0,1\}$. It outputs the master secret key $MK = (\beta, Y)$ and the public parameters as

$$PP = \Big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_1,\ w,\ \{u_{i,0}, u_{i,1}\}_{i=1}^{l},\ \{h_{i,0}, h_{i,1}\}_{i=1}^{l},\ \Lambda = e(g,g)^{\beta}\Big).$$

[2] This property of pre-order traversal is implicitly given by Canetti et al. [10], and it is formally stated in the Theorem 4 of Sahai et al. [40].

CDE.GenKey(L, MK, PP)**:** This algorithm takes as input a label string $L \in \{0,1\}^n$ such that $n \leq l$, the master secret key MK, and the public parameters PP. It first selects a random exponent $r \in \mathbb{Z}_N$ and random elements $Y_0, Y_1, Y_{2,1}, \ldots, Y_{2,n} \in \mathbb{G}_{p_3}$. It outputs a private key that implicitly includes L as

$$SK_L = \Big(K_0 = g^{\beta} w^{-r} Y_0,\ K_1 = g^r Y_1,\ K_{2,1} = F_{1,L[1]}(L|_1)^r Y_{2,1},\ \ldots,\ K_{2,n} = F_{n,L[n]}(L|_n)^r Y_{2,n}\Big).$$

CDE.Encrypt($L, s, \vec{s}, PP$)**:** This algorithm takes as input a label string $L \in \{0,1\}^d$ such that $d \leq l$, a random exponent $s \in \mathbb{Z}_N$, a vector $\vec{s} = (s_1, \ldots, s_d) \in \mathbb{Z}_N^d$ of random exponents, and the public parameters PP. It outputs a ciphertext header that implicitly includes L as

$$CH_L = \Big(C_0 = g^s,\ C_1 = w^s \prod_{i=1}^{d} F_{i,L[i]}(L|_i)^{s_i},\ C_{2,1} = g^{-s_1},\ \ldots,\ C_{2,d} = g^{-s_d}\Big)$$

and a session key as $EK = \Lambda^s$.

CDE.DelegateCT(CH_L, c, PP)**:** This algorithm takes as input a ciphertext header $CH_L = (C_0, \ldots, C_{2,d})$ for a label string $L \in \{0,1\}^d$ such that $d < l$, a bit value $c \in \{0,1\}$, and the public parameters PP. It selects a random exponent $s_{d+1} \in \mathbb{Z}_N$ and outputs a delegated ciphertext header for the new label string $L' = L \| c$ as

$$CH_{L'} = \Big(C_0,\ C'_1 = C_1 \cdot F_{d+1,c}(L')^{s_{d+1}},\ C_{2,1},\ \ldots,\ C_{2,d},\ C'_{2,d+1} = g^{-s_{d+1}}\Big).$$

CDE.RandCT($CH_L, s', \vec{s}', PP$)**:** This algorithm takes as input a ciphertext header $CH_L = (C_0, \ldots, C_{2,d})$ for a label string $L \in \{0,1\}^d$ such that $d \leq l$, a new random exponent $s' \in \mathbb{Z}_N$, a new vector $\vec{s}' = (s'_1, \ldots, s'_d) \in \mathbb{Z}_N^d$, and the public parameters PP. It outputs a re-randomized ciphertext header as

$$CH'_L = \Big(C'_0 = C_0 \cdot g^{s'},\ C'_1 = C_1 \cdot w^{s'} \prod_{i=1}^{d} F_{i,L[i]}(L|_i)^{s'_i},\ C'_{2,1} = C_{2,1} \cdot g^{-s'_1},\ \ldots,\ C'_{2,d} = C_{2,d} \cdot g^{-s'_d}\Big).$$

and a partial session key $EK' = \Lambda^{s'}$ that will be multiplied with the session key EK of CH_L to produce a re-randomized session key.

CDE.Decrypt($CH_L, SK_{L'}, PP$)**:** This algorithm takes as input a ciphertext header CH_L for a label string $L \in \{0,1\}^d$, a private key $SK_{L'}$ for a label string $L' \in \{0,1\}^n$ such that $d \leq n \leq l$, and the public parameters PP. If L is a prefix of L', then it computes a delegated ciphertext header $CH'_{L'} = (C'_0, \ldots, C'_{2,n})$ by running **DelegateCT** and outputs a session key as

$$EK = e(C'_0, K_0) \cdot e(C'_1, K_1) \cdot \prod_{i=1}^{n} e(C'_{2,i}, K_{2,i}).$$

Otherwise, it outputs $\perp$.

Remark 3.8. *The syntax of* ***Encrypt*** *and* ***RandCT*** *is different with the definition of CDE since these algorithms additionally take input random values instead of selecting its own randomness. This difference is essential for the improvement of ciphertext efficiency in SUE. Because of this difference, we cannot show that other scheme that uses the CDE scheme as a building block is secure if the underlying CDE scheme is secure.*

Let ψ be a mapping from time T to a label L[3]. Our SUE scheme that uses our CDE scheme as a building block is described as follows:

SUE.Init(1^λ)**:** This algorithm outputs GDS by running **CDE.Init**(1^λ).

SUE.Setup(GDS, T_{max})**:** This algorithm outputs MK and PP by running **CDE.Setup**(GDS, d_{max}) where $T_{max} = 2^{d_{max}+1} - 1$.

SUE.GenKey(T, MK, PP)**:** This algorithm outputs SK_T by running **CDE.GenKey**$(\psi(T), MK, PP)$.

SUE.Encrypt(T, s, PP)**:** This algorithm takes as input a time T, a random exponent $s \in \mathbb{Z}_N$, and the public parameters PP. It proceeds as follows:

1. It first sets a label string $L \in \{0,1\}^d$ by computing $\psi(T)$. It sets an exponent vector $\vec{s} = (s_1, \ldots, s_d)$ by selecting random exponents $s_1, \ldots, s_d \in \mathbb{Z}_N$, and obtains $CH^{(0)} = (C_0, C_1, C_{2,1}, \ldots, C_{2,d})$ by running **CDE.Encrypt**$(L, s, \vec{s}, PP)$.
2. For $1 \leq j \leq d$, it sets $L^{(j)} = L|_{d-j} \| 1$ and proceeds the following steps:
   (a) If $L^{(j)} = L|_{d-j+1}$, then it sets $CH^{(j)}$ as an empty one since it is redundant or not included in **TimeNodes**(v) where v is associated with T. That is, if $(j = 1) \wedge (L^{(1)} = L)$ then $CH^{(1)}$ is equal with $CH^{(0)}$, and if $(j \geq 2) \wedge (L^{(j)} = L|_{d-j+1})$ then $v^{(j)} \in$ **Path**(**Parent**(v)) where $v^{(j)}$ is associated with $L^{(j)}$.
   (b) Otherwise, it sets a new exponent vector $\vec{s}' = (s'_1, \ldots, s'_{d-j+1})$ where $s'_1, \ldots s'_{d-j}$ are copied from $\vec{s}$ and s'_{d-j+1} is randomly selected in $\mathbb{Z}_N$ since $L^{(j)}$ and L have the same prefix string. It obtains $CH^{(j)} = (C'_0, C'_1, C'_{2,1}, \ldots, C'_{2,d-j+1})$ by running **CDE.Encrypt**$(L^{(j)}, s, \vec{s}', PP)$. It also prunes the redundant elements $C'_0, C'_{2,1}, \ldots, C'_{2,d-j}$ from $CH^{(j)}$, which are already contained in $CH^{(0)}$.
3. It removes all empty $CH^{(j)}$ and sets $CH_T = \left(CH^{(0)}, CH^{(1)}, \ldots, CH^{(d')}\right)$ for some $d' \leq d$ that consists of non-empty $CH^{(j)}$.
4. It outputs a ciphertext header that implicitly includes T as CH_T and a session key as $EK = \Lambda^s$. Note that $CH^{(i)}$ are ordered according to pre-order traversal.

SUE.UpdateCT$(CH_T, T+1, PP)$**:** This algorithm takes as input a ciphertext header $CH_T = (CH^{(0)}, \ldots, CH^{(d')})$ for a time T, a next time $T+1$, and the public parameters PP. Let $L^{(j)}$ be the label of $CH^{(j)}$. It proceeds as follows:

1. If the length d of $L^{(0)}$ is less than d_{max}, then it first obtains $CH_{L^{(0)}\|0}$ and $CH_{L^{(0)}\|1}$ by running **CDE.DelegateCT**$(CH^{(0)}, c, PP)$ for all $c \in \{0,1\}$ since $CH_{L^{(0)}\|0}$ is the ciphertext header for the next time $T+1$ by pre-order traversal. It also prunes the redundant elements in $CH_{L^{(0)}\|1}$. It outputs an updated ciphertext header as $CH_{T+1} = \left(CH'^{(0)} = CH_{L^{(0)}\|0}, CH'^{(1)} = CH_{L^{(0)}\|1}, CH'^{(2)} = CH^{(1)}, \ldots, CH'^{(d'+1)} = CH^{(d')}\right)$.
2. Otherwise, it copies the common elements in $CH^{(0)}$ to $CH^{(1)}$ and simply remove $CH^{(0)}$ since $CH^{(1)}$ is the ciphertext header for the next time $T+1$ by pre-order traversal. It outputs an updated ciphertext header as $CH_{T+1} = \left(CH'^{(0)} = CH^{(1)}, \ldots, CH'^{(d'-1)} = CH^{(d')}\right)$.

[3] In a full binary tree, each node is associated with a unique time T by pre-order traversal and a unique label L by the label assignment. Thus there exist a unique mapping function ψ from a time T to a label L.

SUE.RandCT(CH_T, s', PP): This algorithm takes as input a ciphertext header $CH_T = (CH^{(0)}, \dots, CH^{(d')})$ for a time T, a new random exponent $s' \in \mathbb{Z}_N$, and the public parameters PP. Let $L^{(j)}$ be the label of $CH^{(j)}$ and $d^{(j)}$ be the length of the label $L^{(j)}$. It proceeds as follows:

1. It first sets a vector $\vec{s}' = (s'_1, \dots, s'_{d^{(0)}})$ by selecting random exponents $s'_1, \dots, s'_{d^{(0)}} \in \mathbb{Z}_N$, and obtains $CH'^{(0)}$ by running **CDE.RandCT**($CH^{(0)}, s', \vec{s}', PP$).
2. For $1 \le j \le d'$, it sets a new vector $\vec{s}'' = (s'_1, \dots, s'_{d^{(j)}})$ where $s'_1, \dots s'_{d^{(j)}-1}$ are copied from $\vec{s}'$ and $s'_{d^{(j)}}$ is randomly chosen in $\mathbb{Z}_N$, and obtains $CH'^{(j)}$ by running **CDE.RandCT**($CH^{(j)}, s', \vec{s}'', PP$).
3. It outputs a re-randomized ciphertext header as $CH'_T = \big(CH'^{(0)}, \dots, CH'^{(d')}\big)$ and a partial session key as $EK' = \Lambda^{s'}$ that will be multiplied with the session key EK of CH_T to produce a re-randomized session key.

SUE.Decrypt($CH_T, SK_{T'}, PP$): This algorithm takes as input a ciphertext header CH_T, a private key $SK_{T'}$, and the public parameters PP. If $T \le T'$, then it finds $CH^{(j)}$ from CH_T such that $L^{(j)}$ is a prefix of $L' = \psi(T')$ and outputs EK by running **CDE.Decrypt**($CH^{(j)}, SK_{T'}, PP$). Otherwise, it outputs $\perp$.

Remark 3.9. *The ciphertext delegation (or update) algorithm of CDE (or SUE) just outputs a valid ciphertext header. However, we can easily modify it to output a ciphertext header that is identically distributed with that of the encrypt algorithm of CDE (or SUE) by applying the ciphertext randomization algorithm.*

3.6 Correctness

In CDE, if the label string L of a ciphertext is a prefix of the label string L' of a private key, then the ciphertext can be changed to a new ciphertext for the label string L' by using the ciphertext delegation algorithm. Thus the correctness of CDE is easily obtained from the following equation.

$$
\begin{aligned}
& e(C_0, K_0) \cdot e(C_1, K_1) \cdot \prod_{i=1}^{n} e(C_{2,i}, K_{2,i}) \\
&= e(g^s, g^{\beta} w^{-r} Y_0) \cdot e(w^s \prod_{i=1}^{n} F_{i,L[i]}(L|_i)^{s_i}, g^r Y_1) \cdot \prod_{i=1}^{n} e(g^{-s_i}, F_{i,L[i]}(L|_i)^r Y_{2,i}) \\
&= e(g^s, g^{\beta}) \cdot e(g^s, w^{-r}) \cdot e(w^s, g^r) = e(g,g)^{\beta s}
\end{aligned}
$$

The SUE ciphertext header of a time T consists of the CDE ciphertext headers $CH^{(0)}, CH^{(1)}, \dots, CH^{(d)}$ that are associated with the nodes in **TimeNodes**(v). If the SUE private key of a time T' associated with a node v' satisfies $T \le T'$, then we can find a unique node v'' such that **TimeNodes**(v) $\cap$ **Path**(v') $= v''$ since the property of pre-order tree traversal. Let CH'' be the CDE ciphertext header that is associated with the node v''. The correctness of SUE is easily obtained from the correctness of CDE since the label string L'' of CH'' is a prefix of the label string L' of the private key.

In CDE, the output of **CDE.DelegateCT** is a valid ciphertext header since the function $F_{d+1,c}(L')$ is used with a new random exponent s_{d+1} for the new label string L' with depth $d+1$. The output of **CDE.RandCT** is statistically indistinguishable from that of **CDE.Encrypt** since it has a random exponent $s'' = s + s'$ and a random vector $\vec{s}'' = (s_1 + s'_1, \dots, s_d + s'_d)$ where $s, s_1, \dots, s_d$ are original values in the ciphertext header and $s', s'_1, \dots, s'_d$ are newly selected random values.

In SUE, the output of **SUE.UpdateCT** is a valid ciphertext header since the output of **CDE.DelegateCT** is a valid ciphertext header and the CDE ciphertext headers $CH^{(0)}, \dots CH^{(d)}$ are still associated with the nodes in **TimeNodes**(v) where v is a node for the time $T+1$. The output of **SUE.RandCT** is statistically indistinguishable from that of the encryption algorithm since new random exponents $s', s'_1, \ldots, s'_{d^{(0)}}$ are chosen and these random exponents are reused among the CDE ciphertext headers.

3.7 Security Analysis

We prove the security of our SUE scheme under three static assumptions. Note that we don't prove the security of our SUE scheme under that of our CDE scheme since the SUE scheme reuses the randomness of CDE ciphertext headers to improve the ciphertext efficiency although it uses the CDE scheme as a building block.

Theorem 3.10. *The above SUE scheme is fully secure under a chosen plaintext attack if Assumptions 1, 2, and 3 hold. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{\mathcal{A}}^{SUE}(\lambda) \leq \boldsymbol{Adv}_{\mathcal{B}}^{A1}(\lambda) + 2q\boldsymbol{Adv}_{\mathcal{B}}^{A2}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}}^{A3}(\lambda)$ where q is the maximum number of private key queries of $\mathcal{A}$.*

Proof. To prove the security of our SUE scheme, we use the dual system encryption technique of Lewko and Waters [27, 45]. In dual system encryption, ciphertexts and private keys can be normal or semi-functional type. The normal type and the semi-functional type are indistinguishable and a semi-functional ciphertext can not be decrypted by a semi-functional private key. The whole security proof consists of hybrid games that change the normal challenge ciphertext and the normal private keys to the semi-functional challenge ciphertext and semi-functional private keys respectively. In the final game, the adversary given the semi-functional private keys and the semi-functional challenge ciphertext cannot distinguish a normal session key from a random session key.

We first define the semi-functional type of ciphertexts and private keys. For semi-functional private keys and ciphertexts, we let g_2 denote a fixed generator of the subgroup $\mathbb{G}_{p_2}$ and select random exponents $\{x_{i,0}, x_{i,1}\}_{i=1}^{l}, \{y_{i,0}, y_{i,1}\}_{i=1}^{l} \in \mathbb{Z}_N$ associated with group elements $\{u_{i,0}, u_{i,1}\}_{i=1}^{l}, \{h_{i,0}, h_{i,1}\}_{i=1}^{l} \in \mathbb{G}_{p_1}$ of PP.

SUE.GenKeySF1. This algorithm first creates a normal private key $SK'_L = (K'_0, \ldots, K'_{2,n})$ for $L = \psi(T)$ using the master key. Let $f_{i,c}(L) = x_{i,c}L + y_{i,c}$ where $c \in \{0,1\}$. It chooses random exponents $a, b \in \mathbb{Z}_N$ and outputs a semi-functional private key of type 1 as

$$SK_L = \Big(K_0 = K'_0 g_2^{-a},\ K_1 = K'_1 g_2^{b},\ K_{2,1} = K'_{2,1} g_2^{f_{1,L[1]}(L|_1)b},\ \ldots,\ K_{2,n} = K'_{2,n} g_2^{f_{n,L[n]}(L|_n)b}\Big).$$

SUE.GenKeySF2. This algorithm first creates a normal private key $SK'_L = (K'_0, \ldots, K'_{2,n})$ using the master key. It chooses a random exponent $a \in \mathbb{Z}_N$ and outputs a semi-functional private key of type 2 as

$$SK_L = \Big(K_0 = K'_0 g_2^{-a},\ K_1 = K'_1,\ K_{2,1} = K'_{2,1},\ \ldots,\ K_{2,n} = K'_{2,n}\Big).$$

SUE.EncryptSF. This algorithm first creates a normal ciphertext header $CH'_T = (CH'^{(0)}, \ldots, CH'^{(d)})$ and a session key EK' where $L^{(j)}$ is the label string of $CH'^{(j)}$ and $d^{(j)}$ is the length of $L^{(j)}$.

1. Let $CH'^{(0)} = \big(C'_0, C'_1, C'_{2,1}, \ldots, C'_{2,d^{(0)}}\big)$. It chooses random exponents $c, u \in \mathbb{Z}_N$ and sets an exponent vector $\vec{y} = (z_1, \ldots, z_{d^{(0)}})$ by selecting random exponents $z_1, \ldots, z_{d^{(0)}} \in \mathbb{Z}_N$, and creates semi-functional components $CH^{(0)}$ as

$$\Big(C_0 = C'_0 g_2^{c},\ C_1 = C'_1 g_2^{u + \sum_{i=1}^{d^{(0)}} f_{i,L^{(0)}[i]}(L^{(0)}|_i) z_i},\ C_{2,1} = C'_{2,1} g_2^{-z_1},\ \ldots,\ C_{2,d^{(0)}} = C'_{2,d^{(0)}} g_2^{-z_{d^{(0)}}}\Big).$$

2. For $1 \le j \le d$, it sets $L^{(j)} = L|_{d-j} \| 1$ and proceeds as follows: Let $CH'^{(j)} = \big(C'_1, C'_{2,d^{(j)}}\big)$. If $L^{(j)} = L|_{d-j+1}$, then it sets $CH^{(j)}$ as an empty one. Otherwise, it selects a random exponent $z'_{d^{(j)}} \in \mathbb{Z}_N$ and creates semi-functional components $CH^{(j)}$ as
$$\Big(C_1 = C'_1 g_2^{u+\sum_{i=1}^{d^{(j)}-1} f_{i,L^{(j)}[i]}(L^{(j)}|_i) z_i + f_{d^{(j)},L^{(j)}[d^{(j)}]}(L^{(j)}|_{d^{(j)}}) z'_{d^{(j)}}},\ C_{2,d^{(j)}} = C'_{2,d^{(j)}} g_2^{-z'_{d^{(j)}}}\Big).$$
Note that the elements $C'_0, C'_{2,1}, \ldots, C'_{2,d^{(j)}-1}$ were pruned in the normal ciphertext header.
3. It removes all empty $CH^{(j)}$ and sets $CH_T = (CH^{(0)}, \ldots, CH^{(d')})$ for some d' that consists of non-empty $CH^{(j)}$.
4. It outputs a semi-functional ciphertext header as $CH_T = (CH^{(0)}, \ldots, CH^{(d')})$ and a session key as $EK = EK'$.

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

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

Game $\mathbf{G}_0$. This game is the original security game. In this game, all private keys and the challenge ciphertext header are normal.

Game $\mathbf{G}_1$. In the next game, all private keys are normal, but the challenge ciphertext header is semi-functional.

Game $\mathbf{G}_2$. Next, we define a new game $\mathbf{G}_2$. In this game, all private keys are semi-functional of type 2 and the challenge ciphertext header is semi-functional. Suppose that an adversary makes at most q private key queries. For the security proof, we additionally define a sequence of games $\mathbf{G}_{1,1,1}, \mathbf{G}_{1,1,2}, \ldots, \mathbf{G}_{1,k,1}, \mathbf{G}_{1,k,2}, \ldots, \mathbf{G}_{1,q,1}, \mathbf{G}_{1,q,2}$ where $G_1 = G_{1,0,2}$. In the game $\mathbf{G}_{1,k,1}$ for $1 \le k \le q$, the challenge ciphertext header is semi-functional, the first $k-1$ private keys are semi-functional of type 2, the kth private key is semi-functional of type 1, and the remaining private keys are normal. In the game $\mathbf{G}_{1,k,2}$ for $1 \le k \le q$, the challenge ciphertext header is semi-functional, the first k private keys are semi-functional of type 2, and the remaining private keys are normal. It is obvious that $\mathbf{G}_{1,q,2} = \mathbf{G}_2$.

Game $\mathbf{G}_3$. In the final game $\mathbf{G}_3$, all private keys are semi-functional of type 2 and the ciphertext header is semi-functional, but the session key is random.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_j}$ be the advantage of $\mathcal{A}$ in the game $\mathbf{G}_j$. We easily obtain that $\mathbf{Adv}_{\mathcal{A}}^{SUE}(\lambda) = \mathbf{Adv}_{\mathcal{A}}^{G_0}$, $\mathbf{Adv}_{\mathcal{A}}^{G_1} = \mathbf{Adv}_{\mathcal{A}}^{G_{1,0,2}}$, $\mathbf{Adv}_{\mathcal{A}}^{G_2} = \mathbf{Adv}_{\mathcal{A}}^{G_{1,q,2}}$, and $\mathbf{Adv}_{\mathcal{A}}^{G_3} = 0$. Through the following four lemmas, we can obtain the following equation

$$\begin{aligned}
&\mathbf{Adv}_{\mathcal{A}}^{SUE}(\lambda) \\
&= \mathbf{Adv}_{\mathcal{A}}^{G_0} + (\mathbf{Adv}_{\mathcal{A}}^{G_1} - \mathbf{Adv}_{\mathcal{A}}^{G_1}) + \sum_{k=1}^{q} \big(\mathbf{Adv}_{\mathcal{A}}^{G_{1,k,1}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,k,1}} + \mathbf{Adv}_{\mathcal{A}}^{G_{1,k,2}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,k,2}}\big) - \mathbf{Adv}_{\mathcal{A}}^{G_3} \\
&\le \big|\mathbf{Adv}_{\mathcal{A}}^{G_0} - \mathbf{Adv}_{\mathcal{A}}^{G_1}\big| + \sum_{k=1}^{q} \big(\big|\mathbf{Adv}_{\mathcal{A}}^{G_{1,k-1,2}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,k,1}}\big| + \big|\mathbf{Adv}_{\mathcal{A}}^{G_{1,k,1}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,k,2}}\big|\big) + \big|\mathbf{Adv}_{\mathcal{A}}^{G_2} - \mathbf{Adv}_{\mathcal{A}}^{G_3}\big| \\
&\le \mathbf{Adv}_{\mathcal{B}}^{A1}(\lambda) + 2q\mathbf{Adv}_{\mathcal{B}}^{A2}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{A3}(\lambda).
\end{aligned}$$

This completes our proof. $\square$

Lemma 3.11. *If the Assumption 1 holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_0$ *and* $\mathbf{G}_1$ *with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the Assumption 1 using $\mathcal{A}$ is given: a challenge tuple $D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3})$ and Z where $Z = Z_0 = X_1 \in \mathbb{G}_{p_1}$ or $Z = Z_1 = X_1 R_1 \in \mathbb{G}_{p_1 p_2}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ first chooses random exponents $w', \{u'_{i,0}, u'_{i,1}\}_{i=1}^{l}, \{h'_{i,0}, h'_{i,1}\}_{i=1}^{l}, \beta \in \mathbb{Z}_N$. Let $f_{i,b}(L) = u'_{i,b} L + h'_{i,b}$ where $b \in \{0,1\}$. It sets the master secret key as $MK = (\beta, Y = g_{p_3})$ and publishes the public parameters PP as

$$g = g_{p_1}, w = g_{p_1}^{w'}, \{u_{i,0} = g_{p_1}^{u'_{i,0}}, u_{i,1} = g_{p_1}^{u'_{i,1}}\}_{i=1}^{l}, \{h_{i,0} = g_{p_1}^{h'_{i,0}}, h_{i,1} = g_{p_1}^{h'_{i,1}}\}_{i=1}^{l}, \Lambda = e(g_{p_1}, g_{p_1})^{\beta}.$$

Query 1: $\mathcal{A}$ adaptively request a private key for a time T. $\mathcal{B}$ creates a normal private key by running **SUE.GenKey** since it knows the master secret key. Note that it cannot create a semi-functional private key since it does not know g_{p_2}.

Challenge: In the challenge step, $\mathcal{A}$ outputs a challenge time T^*. $\mathcal{B}$ proceeds as follows:

1. It first sets a label string $L^* \in \{0,1\}^d$ by computing $\psi(T^*)$. It chooses random exponents $s'_1, \ldots, s'_d \in \mathbb{Z}_N$. It implicitly sets g^s to be the $\mathbb{G}_{p_1}$ part of Z and creates ciphertext components $CH^{(0)}$ as

$$\Big(C_0 = Z,\ C_1 = Z^{w'} \prod_{i=1}^{d} Z^{f_{i,L^*[i]}(L^*|_i) s'_i},\ C_{2,1} = Z^{-s'_1},\ \ldots,\ C_{2,d} = Z^{-s'_d}\Big).$$

2. For $1 \le j \le d$, it first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: If $L^{(j)} = L|_{d-j+1}$, it sets $CH^{(j)}$ as an empty one. Otherwise, it selects $s'_{d-j+1} \in \mathbb{Z}_N$ and creates ciphertext components $CH^{(j)}$ as

$$\Big(C_1 = Z^{w'} \prod_{i=1}^{d-j} Z^{f_{i,L^{(j)}[i]}(L^{(j)}|_i) s'_i} \cdot Z^{f_{d-j+1,L^{(j)}[d-j+1]}(L^{(j)}|_{d-j+1}) s'_{d-j+1}},\ C_{2,d-j+1} = Z^{-s'_{d-j+1}}\Big).$$

3. It removes all empty $CH^{(j)}$ and sets $CH_T = \big(CH^{(0)}, \ldots, CH^{(d')}\big)$ for some d' that consists of non-empty $CH^{(j)}$.

4. It sets the challenger ciphertext header as $CH_{T^*} = CH_T$ and the session keys $EK_0 = e(Z, g)^{\beta}$. It flips a random coin $b \in \{0,1\}$ and gives CH_{T^*} and EK_0 to $\mathcal{A}$ if $b = 0$. Otherwise, it gives CH_{T^*} and a random session key to $\mathcal{A}$.

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

To finish the proof, we should show that the distribution of the simulation is correct. We first show that the distribution using $Z_0 = X_1$ is the same as $\mathbf{G}_0$. Let $X_1 = g_{p_1}^{s}$. The ciphertext header is also correctly distributed since it implicitly sets $s_1 \equiv s s'_1 \mod p_1, \ldots, s_j \equiv s s'_j \mod p_1$. We next show that the distribution using $Z_1 = X_1 R_1$ is the same as $\mathbf{G}_1$. Let $X_1 R_1 = g_{p_1}^{s} g_{p_2}^{c}$. The semi-functional ciphertext header is generated by implicitly setting $u \equiv c w' \mod p_2, z_1 \equiv c s'_1 \mod p_2, \ldots, z_j \equiv c s'_j \mod p_2$. The values $c, u, z_1, \ldots, z_j$ modulo p_2 are not correlated with their values modulo p_1 by the Chinese Remainder Theorem (CRT). Thus this is a properly distributed semi-functional ciphertext header. This completes our proof. $\square$

Lemma 3.12. *If the Assumption 2 holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_{1,k-1,2}$ *and* $\mathbf{G}_{1,k,1}$ *with a non-negligible advantage.*

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

Setup: $\mathcal{B}$ first chooses random exponents $w', \{u'_{i,0}, u'_{i,1}\}_{i=1}^{l}, \{h'_{i,0}, h'_{i,1}\}_{i=1}^{l}, \beta \in \mathbb{Z}_N$. Let $f_{i,b}(L) = u'_{i,b}L + h'_{i,b}$ where $b \in \{0,1\}$. It sets the master secret key as $MK = (\beta, Y = g_{p_3})$ and publishes the public parameters PP as

$$g = g_{p_1}, w = g_{p_1}^{w'}, \{u_{i,0} = g_{p_1}^{u'_{i,0}}, u_{i,1} = g_{p_1}^{u'_{i,1}}\}_{i=1}^{l}, \{h_{i,0} = g_{p_1}^{h'_{i,0}}, h_{i,1} = g_{p_1}^{h'_{i,1}}\}_{i=1}^{l}, \Lambda = e(g_{p_1}, g_{p_1})^{\beta}.$$

Query 1: $\mathcal{A}$ adaptively requests a private key for a time T such that $L = \psi(T) \in \{0,1\}^n$. If this is a jth private key query, then $\mathcal{B}$ handles this query as follows: If $j < k$, then it chooses random exponents $r, a' \in \mathbb{Z}_N$ and random elements $Y'_0, Y'_1, Y'_{2,1}, \ldots, Y'_{2,n} \in \mathbb{G}_{p_3}$, and builds a semi-functional private key of type 2 as

$$K_0 = g^{\beta} w^{-r} (R_2Y_1)^{-a'} Y'_0,\ K_1 = g^r Y'_1,\ K_{2,1} = (u_{1,L[1]}^{L|_1} h_{1,L[1]})^r Y'_{2,1},\ \ldots,\ K_{2,n} = (u_{n,L[n]}^{L|_n} h_{n,L[n]})^r Y'_{2,n}.$$

If $j = k$, then it chooses random elements $Y'_0, Y'_1, Y'_{2,1}, \ldots, Y'_{2,n} \in \mathbb{G}_{p_3}$ and builds a private key as

$$K_0 = g^{\beta} (Z)^{-w'} Y'_0,\ K_1 = ZY'_1,\ K_{2,1} = (Z)^{f_{1,L[1]}(L|_1)} Y'_{2,1},\ \ldots,\ K_{2,n} = (Z)^{f_{n,L[n]}(L|_n)} Y'_{2,n}.$$

If $j > k$, then it builds a normal private key by running **SUE.GenKey** since it knows the master secret key.

Challenge: In the challenge step, $\mathcal{A}$ outputs a challenge time T^*. $\mathcal{B}$ proceeds as follows:

1. It first sets a label string $L^* \in \{0,1\}^d$ by computing $\psi(T^*)$. It chooses random exponents $s'_1, \ldots, s'_d \in \mathbb{Z}_N$. It creates ciphertext components $CH^{(0)}$ by implicitly setting $g^s = X_1$ as

$$\Big(C_0 = X_1R_1,\ C_1 = (X_1R_1)^{w'} \prod_{i=1}^{d} (X_1R_1)^{f_{i,L^*[i]}(L^*|_i)s'_i},\ C_{2,1} = (X_1R_1)^{-s'_1},\ \ldots,\ C_{2,d} = (X_1R_1)^{-s'_d}\Big).$$

2. For $1 \le j \le d$, it first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: If $L^{(j)} = L|_{d-j+1}$, it sets $CH^{(j)}$ an an empty one. Otherwise, it selects $s''_{d-j+1} \in \mathbb{Z}_N$ and creates ciphertext components $CH^{(j)}$ as

$$\Big(C_1 = (X_1R_1)^{w'} \prod_{i=1}^{d-j} (X_1R_1)^{f_{i,L^{(j)}[i]}(L^{(j)}|_i)s_i} \cdot (X_1R_1)^{f_{d-j+1,L^{(j)}[d-j+1]}(L^{(j)}|_{d-j+1})s''_{d-j+1}},\ C_{2,d-j+1} = (X_1R_1)^{-s''_{d-j+1}}\Big).$$

3. It removes all empty $CH^{(j)}$ and sets $CH_T = \big(CH^{(0)}, \ldots, CH^{(d')}\big)$ for some d' that consists of non-empty $CH^{(j)}$.

4. It sets the challenger ciphertext header as $CH_{T^*} = CH_T$ and the session keys $EK_0 = e(X_1R_1, g)^{\beta}$. It flips a random coin $b \in \{0,1\}$ and gives CH_{T^*} and EK_0 to $\mathcal{A}$ if $b = 0$. Otherwise, it gives CH_{T^*} and a random session key to $\mathcal{A}$.

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

To finish this proof, we should show that the distribution of the simulation is correct. We first show that the distribution using $Z_0 = X_2Y_2$ is the same as $\mathbf{G}_{1,k-1,2}$. Let $X_2Y_2 = g_{p_1}^r Y_2$ and $X_1R_1 = g_{p_1}^s g_{p_2}^c$. The kth private key is correctly distributed as

$$K_0 = g^\beta (X_2Y_2)^{-w'} Y_0' = g^\beta (g_{p_1}^{-w'})^r Y_2^{-w'} Y_0' = g^\beta w^{-r} Y_0'',$$
$$K_1 = (X_2Y_2)Y_1' = g_{p_1}^r Y_2 Y_1' = g^r Y_1'',$$
$$K_{2,i} = (X_2Y_2)^{f_{i,L[i]}(L|_i)} Y_{2,i}' = g_{p_1}^{(u'_{i,L[i]} L|_i + h'_{i,L[i]})r} Y_2^{f_{i,L[i]}(L|_i)} Y_{2,i}' = (u_{i,L[i]}^{L|_i} h_{i,L[i]})^r Y_{2,i}''.$$

The semi-functional challenge ciphertext header is also correctly distributed by implicitly setting $s_i \equiv ss_i' \mod p_1,\ u \equiv cw' \mod p_2,\ z_i \equiv cs_i' \mod p_2$ as

$$C_0 = X_1R_1 = g_{p_1}^s g_{p_2}^c,$$
$$C_1 = (X_1R_1)^{w'} \prod_{i=1}^{d} (X_1R_1)^{f_{i,L^*[i]}(L^*|_i)s_i'} = (g_{p_1}^{w'})^s (g_{p_2}^c)^{w'} \prod_{i=1}^{d} (g_{p_1}^{(u'_{i,L^*[i]} L^*|_i + h'_{i,L^*[i]})})^{ss_i'} (g_{p_2}^c)^{f_{i,L^*[i]}(L^*|_i)s_i'}$$
$$= w^s \prod_{i=1}^{d} (u_{i,L^*[i]}^{L^*|_i} h_{i,L^*[i]})^{s_i} g_{p_2}^{u + \sum_{i=1}^{d} f_{i,L^*|_i}(L^*|_i) z_i},$$
$$C_{2,i} = (X_1R_1)^{-s_i'} = g_{p_1}^{-ss_i'} g_{p_2}^{-cs_i'} = g^{-s_i} g_{p_2}^{-z_i}.$$

We next show that the distribution using $Z_1 = X_2R_3Y_2$ is the same as $\mathbf{G}_{1,k,1}$. Let $X_2R_3Y_2 = X_2 g_{p_2}^b Y_2$. The only difference between Z_0 and Z_1 is that Z_1 additionally has R_3. The kth private key components that have Z additionally have $R_3^{-w'}, R_3, R_3^{f_{1,L[1]}(L|_1)}, \dots, R_3^{f_{n,L[n]}(L|_n)}$ respectively. If we implicitly sets $a \equiv bw' \mod p_2$, then the kth private key is correctly distributed as the same as $\mathbf{G}_{1,k,1}$ except that it is nominally semi-functional since $ac \equiv (bw')c \equiv bu \mod p_2$.

Finally, we show that the adversary cannot distinguish the nominally semi-functional private key from the semi-functional private key of type 1. To argue this, we use the restriction of the security model such that the adversary cannot request a private key for a time T such that $T^* \leq T$. Recall that L is the label string of T, $L^{(j)}$ is the label string of $CH^{(j)}$ in $CH_{T^*} = (CH^{(0)}, \dots, CH^{(d')})$, and $L^{(0)} = L^*$. For easy analysis, we first consider $CH^{(0)}$ since the challenge SUE ciphertext header CH_{T^*} consists of many CDE ciphertext headers. From the restriction of the SUE security model, we have that the adversary cannot request a private key for a label string L such that L^* is a prefix of L. Suppose there exists an unbounded adversary. Then the adversary can gather the values $-w'b \mod p_2, b \mod p_2, \{f_{i,L[i]}(L|_i) \mod p_2\}$ from the kth private key for a label string $L \in \{0,1\}^n$ and $c \mod p_2, w'c + \sum_{i=1}^{d} f_{i,L^*[i]}(L^*|_i)s_i' \mod p_2, \{s_i' \mod p_2\}$ from $CH^{(0)}$ for a label string $L^* \in \{0,1\}^d$.

- If $(n < d)$ or $((d \leq n) \wedge (L[d] \neq L^*[d]))$, then $f_{d,L^*[d]}(L^*) \mod p_2$ looks random to the unbounded adversary since $u'_{d,L^*[d]} \mod p_2, h'_{d,L^*[d]} \mod p_2$ are information theoretically hidden to the adversary by the Chinese Remainder Theorem (CRT).

- If $((d \leq n) \wedge (L[d] = L^*[d]))$, then $f_{d,L^*[d]}(L|_d) \mod p_2$ and $f_{d,L^*[d]}(L^*) \mod p_2$ look random to the adversary since $f_{i,b}(L) = u'_{i,b}L + h'_{i,b}$ is a pair-wise independent function, $L|_d \neq L^*$ by the restriction of the security model, and $u'_{d,L^*[d]} \mod p_2, h'_{d,L^*[d]} \mod p_2$ are information theoretically hidden to the adversary.

Thus it is easy to show that $w'c + \sum_{i=1}^{d} f_{i,L^*[i]}(L^*|_i) \mod p_2$ looks random to the unbounded adversary since $f_{d,L^*[d]}(L^*) \mod p_2$ looks random to the adversary. We can apply the similar argument to other CDE ciphertext headers. This completes our proof. $\square$

Lemma 3.13. *If the Assumption 2 holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_{1,k,1}$ *and* $\mathbf{G}_{1,k,2}$ *with a non-negligible advantage.*

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

Setup: $\mathcal{B}$ first chooses random exponents $w', \{u'_{i,0}, u'_{i,1}\}_{i=1}^{l}, \{h'_{i,0}, h'_{i,1}\}_{i=1}^{l}, \beta \in \mathbb{Z}_N$. Let $f_{i,b}(L) = u'_{i,b}L + h'_{i,b}$ where $b \in \{0,1\}$. It sets the master secret key as $MK = (\beta, Y = g_{p_3})$ and publishes the public parameters PP as

$$g = g_{p_1}, w = g_{p_1}^{w'}, \{u_{i,0} = g_{p_1}^{u'_{i,0}}, u_{i,1} = g_{p_1}^{u'_{i,1}}\}_{i=1}^{l}, \{h_{i,0} = g_{p_1}^{h'_{i,0}}, h_{i,1} = g_{p_1}^{h'_{i,1}}\}_{i=1}^{l}, \Lambda = e(g_{p_1}, g_{p_1})^{\beta}.$$

Query 1: $\mathcal{A}$ adaptively requests a private key for a time T such that $L = \psi(T) \in \{0,1\}^n$. If this is a jth private key query, then $\mathcal{B}$ handles this query as follows: If $j < k$, then it chooses random exponents $r, a' \in \mathbb{Z}_N$ and random elements $Y'_0, Y'_1, Y'_{2,1}, \ldots, Y'_{2,n} \in \mathbb{G}_{p_3}$, and builds a semi-functional private key of type 2 as

$$K_0 = g^{\beta} w^{-r} (R_2Y_1)^{-a'} Y'_0,\ K_1 = g^r Y'_1,\ K_{2,1} = (u_{1,L[1]}^{L|_1} h_{1,L[1]})^r Y'_{2,1},\ \ldots,\ K_{2,n} = (u_{n,L[n]}^{L|_n} h_{n,L[n]})^r Y'_{2,n}.$$

If $j = k$, then it chooses random elements $Y'_0, Y'_1, Y'_{2,1}, \ldots, Y'_{2,n} \in \mathbb{G}_{p_3}$ and builds a private key as

$$K_0 = g^{\beta} (Z)^{-w'} (R_2Y_1)^{-a'} Y'_0,\ K_1 = Z Y'_1,\ K_{2,1} = (Z)^{f_{1,L[1]}(L|_1)} Y'_{2,1},\ \ldots,\ K_{2,n} = (Z)^{f_{n,L[n]}(L|_n)} Y'_{2,n}.$$

If $j > k$, then it builds a normal private key by running **SUE.GenKey** since it knows the master secret key.

Challenge: In the challenge step, $\mathcal{A}$ outputs a challenge time E^*. $\mathcal{B}$ creates the challenge ciphertext header CH_{T^*} and the session key EK_0 the same as the proof of Lemma 3.12. $\mathcal{B}$ flips a random coin $b \in \{0,1\}$ and gives CH_{T^*} and EK_0 to $\mathcal{A}$ if $b = 0$. Otherwise, it gives CH_{T^*} and a random session key to $\mathcal{A}$.

Query 2: Same as Query 1.

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

To finish this proof, we should show that the distribution of the simulation is correct. We first show that the distribution using $Z_0 = X_2Y_2$ is the same as $\mathbf{G}_{1,k,2}$. Let $X_2Y_2 = g_{p_1}^{r} Y_2$. The kth private key is correctly distributed as

$$\begin{aligned}
K_0 &= g^{\beta} (X_2Y_2)^{-w'} (R_2Y_1)^{a'} Y'_0 = g^{\beta} g_{p_1}^{-w'r} R_2^{a'} Y_2^{-w'} Y_1^{a'} Y'_0 = g^{\beta} w^{-r} R_2^{a'} Y''_0, \\
K_1 &= (X_2Y_2) Y'_1 = g_{p_1}^{r} Y_2 Y'_1 = g^r Y''_1, \\
K_{2,i} &= (X_2Y_2)^{f_{i,L[i]}(L|_i)} Y'_{2,i} = g_{p_1}^{(u'_{i,L[i]} L|_i + h'_{i,L[i]}) r} Y_2^{f_{i,L[i]}(L|_i)} Y'_{2,i} = (u_{i,L[i]}^{L|_i} h_{i,L[i]})^r Y''_{2,i}.
\end{aligned}$$

We next show that the distribution using $Z_1 = X_2R_3Y_2$ is the same as $\mathbf{G}_{1,k,1}$. The only difference between Z_0 and Z_1 is that Z_1 additionally has R_3. It is easy to check the distribution of kth private key, but it is no longer nominally semi-functional since an additional $(R_2Y_1)^{a'}$ term is added to randomize the $\mathbb{G}_{p_2}$ part of K_0. Note that we no longer have the relation $ac \equiv bu \mod p_2$. This completes our proof. $\square$

Lemma 3.14. *If the Assumption 3 holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_2$ *and* $\mathbf{G}_3$ *with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguish $\mathbf{G}_2$ from $\mathbf{G}_3$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the Assumption 3 using $\mathcal{A}$ is given: a challenge tuple $D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_2}, g_{p_3}, g_{p_1}^a R_1, g_{p_1}^b R_2)$ and Z where $Z = Z_0 = e(g_{p_1}, g_{p_1})^{ab}$ or $Z = Z_1 = e(g_{p_1}, g_{p_1})^c$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ first chooses a random exponent $w' \in \mathbb{Z}_N$ and random elements $\{u_{i,0}, u_{i,1}\}_{i=1}^l, \{h_{i,0}, h_{i,1}\}_{i=1}^l \in \mathbb{G}_{p_1}$. It implicitly sets $\beta = a$ and publishes the public parameters PP as

$$g = g_{p_1}, w = g_{p_1}^{w'}, \{u_{i,0}, u_{i,1}\}_{i=1}^l, \{h_{i,0}, h_{i,1}\}_{i=1}^l, \Lambda = e(g_{p_1}, g_{p_1}^a R_1).$$

Query 1: $\mathcal{A}$ adaptively requests a private key for a time T such that $L = \psi(T) \in \{0,1\}^n$. To response this query, $\mathcal{B}$ selects a random exponent $r \in \mathbb{Z}_N$ and random elements $R_0' \in \mathbb{G}_{p_2}, Y_0', Y_1', Y_{2,1}', \dots, Y_{2,n}' \in \mathbb{G}_{p_3}$, and creates a semi-functional private key of type 2 as

$$K_0 = (g_{p_1}^a R_1) w^{-r} R_0' Y_0',\ K_1 = g_{p_1}^r Y_1',\ K_{2,1} = F_{1,L[1]}(L|_1)^r Y_{2,1}',\ \dots,\ K_{2,n} = F_{n,L[n]}(L|_n)^r Y_{2,n}'.$$

Challenge: In the challenge step, $\mathcal{A}$ outputs a challenge time T^*. $\mathcal{B}$ proceeds as follows:

1. It first sets a label string $L^* \in \{0,1\}^d$ by computing $\psi(T^*)$. It chooses random exponents $s_1, \dots, s_d \in \mathbb{Z}_N$ and random elements $R_1', R_{2,1}', \dots, R_{2,d}' \in \mathbb{G}_{p_2}$. It creates ciphertext components $CH^{(0)}$ by implicitly setting $g^s = g^b$ as
$$\Big(C_0 = g_{p_1}^b R_2,\ C_1 = (g_{p_1}^b R_2)^{w'} \prod_{i=1}^{d} F_{i,L^{(0)}[i]}(L^{(0)}|_i)^{s_i} R_1',\ C_{2,1} = g^{-s_1} R_{2,1}',\ \dots,\ C_{2,d} = g^{-s_d} R_{2,d}'\Big).$$

2. For $1 \le j \le d$, it first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: If $L^{(j)} = L|_{d-j+1}$, it sets $CH^{(j)}$ as an empty one. Otherwise, it selects $s_{d-j+1}' \in \mathbb{Z}_N$, $R_1', R_{2,d-j+1}' \in \mathbb{G}_{p_2}$ and creates ciphertext components $CH^{(j)}$ as
$$\Big(C_1 = (g_{p_1}^b R_2)^{w'} \prod_{i=1}^{d-j} F_{i,L^{(j)}[i]}(L^{(j)}|_i)^{s_i} \cdot F_{d-j+1,L^{(j)}[d_j]}(L^{(j)}|_{d-j+1})^{s_{d-j+1}'} R_1',\ C_{2,d-j+1} = g^{-s_{d-j+1}'} R_{2,d-j+1}'\Big).$$

3. It removes all empty $CH^{(j)}$ and sets $CH_T = \big(CH^{(0)}, \dots, CH^{(d')}\big)$ for some d' that consists of non-empty $CH^{(j)}$.

4. It sets the challenger ciphertext header as $CH_{T^*} = CH_T$ and the session keys $EK = Z$. It gives CH_{T^*} and EK to $\mathcal{A}$.

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

To finish the proof, we show that the distribution of the simulation is correct. The semi-functional private keys of type 2 and the semi-functional challenge ciphertext header are correctly distributed. If $Z = Z_0 = e(g_{p_1}, g_{p_1})^{ab}$, then the session key is properly distributed as the same as $\mathbf{G}_2$. Otherwise, the session key is a random element as the same as $\mathbf{G}_3$. This completes our proof. □

Corollary 3.15. *The above CDE scheme is fully secure under a chosen plaintext attack if Assumptions 1, 2, and 3 hold. That is, for any PPT adversary* $\mathcal{A}$, *we have that* $\boldsymbol{Adv}_{\mathcal{A}}^{CDE}(\lambda) \le \boldsymbol{Adv}_{\mathcal{B}}^{A1}(\lambda) + 2q\boldsymbol{Adv}_{\mathcal{B}}^{A2}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}}^{A3}(\lambda)$ *where* q *is the maximum number of private key queries of* $\mathcal{A}$.

We omit the proof of this since it can be easily derived from that of Theorem 3.10.

3.8 Discussions

Efficiency. Suppose that an SUE scheme supports the maximum time T_{max}. In this case, the number of group elements in a private key is $\log T_{max}$ since the private key of SUE is the same as that of CDE, and the number of group elements in a ciphertext is at most $3 \log T_{max}$ since the SUE ciphertext reuses the randomness of other ciphertexts of CDE. The decryption algorithm of SUE requires $\log T_{max}$ pairing operations since it just runs the decryption algorithm of CDE at once after finding the matching ciphertext of CDE. For example, if we set $T_{max} = 2^{20}$ in SUE, then the private key consists of at most 20 group elements, the ciphertext consists of at most 60 group elements, and the decryption algorithm requires at most 20 pairing operations.

Supporting an Exponential Number of Time Periods. In our SUE scheme, the setup algorithm takes the maximum (polynomial) number of time periods T_{max} as an input. We can modify our SUE scheme to support an exponential number of time periods by setting the maximum depth of CDE as $d_{max} = \lambda$ since the security loss of the reduction is independent of T_{max}. In this case, this scheme can support $T_{max} = 2^{\lambda+1} - 1$.

Supporting a Time Interval. In SUE, a ciphertext with a time T_L can be updatable by anyone just using a public key to any time T such that $T_L \leq T$. However, one may want to limit the lifetime of a ciphertext by specifying the time interval $[T_L, T_R]$ of a ciphertext for some applications where it is necessary to ensure that a ciphertext is only decrypted within a specific time interval. If we use the technique of Kasamatsu et al. [21] that was used to build a time-specific encryption from FSE, we expect that an SUE scheme for a time interval $[T_L, T_R]$ can be built by carefully combining two SUE schemes to prevent collusion attacks: one is a future SUE scheme with ciphertext updating for future times such that $T_L \leq T$ and another one is a past SUE scheme with ciphertext updating for past times $T \leq T_R$.

Different Constructions. We use the HIBE structure of Boneh and Boyen [4,5] for our SUE scheme with private keys of $O(\log T_{max})$ group elements and ciphertexts of $O(\log T_{max})$ group elements by reusing the randomness of ciphertexts. If we use other HIBE structures for SUE, then they will give an SUE scheme with different efficiency tradeoffs. For example, if we build an SUE scheme by using the HIBE structure of Boneh, Boyen, and Goh [6], then this SUE scheme will have private keys of $O(1)$ group elements and ciphertexts of $O(\log^2 T_{max})$ group elements. If we use the hybrid HIBE structure that combines the Boneh-Boyen-Goh HIBE and the Boneh-Boyen HIBE, then this SUE scheme will have private keys of $O(\log^{1/2} T_{max})$ group elements and ciphertexts of $O(\log^{3/2} T_{max})$ group elements.

Prime Order Groups. Our SUE scheme is fully secure under a chosen plaintext attack, but it is based on composite order bilinear groups since the proof of its security uses the dual system encryption technique. The group order of composite order bilinear groups should be at least 1024 bit to prevent the integer factorization attack. To improve the efficiency of the SUE scheme, we can use prime order bilinear groups instead of composite order bilinear groups. In Appendix A, we proposed an SUE scheme in prime order bilinear groups, but it is only selectively secure under a chosen plaintext attack where an adversary submits the challenge time T^* before he receives the public parameters.

4 Revocable-Storage Attribute-Based Encryption

In this section, we define RS-ABE and propose an RS-ABE scheme with shorter ciphertexts by combining the ABE scheme of Lewko et al. [26] and our SUE scheme in the previous section.

4.1 Definitions

Revocable-storage attribute-based encryption (RS-ABE) is attribute-based encryption (ABE) that additionally supports the revocation functionality and the ciphertext update functionality. The revocation functionality is that a user whose private key is revealed can be revoked by preventing for the user to decrypt ciphertexts that will be created in the future. The ciphertext update functionality is that a user who is revoked cannot access to ciphertexts that are created in the past by updating the time of the ciphertexts. Boldyreva, Goyal, and Kumar introduced the concept of revocable ABE (R-ABE) that provides the revocation functionality [3], and Sahai, Seyalioglu, and Waters introduced the concept of RS-ABE that provides the ciphertext update functionality in R-ABE [40].

In RS-ABE, each user receives a private key that is associated with a set of attributes S and a user index u from a key generation center. After that, the key generation center periodically broadcasts an update key for a set of non-revoked users where the update key is associated with an update time T' and a set of revoked users R. To send a message to a receiver, a sender creates a ciphertext that is associated with an access structure $\mathbb{A}$ and a time T. A receiver who has a private key for a set of attributes S and a user index u and an update key for an update time T' and a set of revoked users R can decrypt the ciphertext by combining the private key and the update key if the set of attributes S satisfies the access structure $\mathbb{A}$, the user index u is not contained in the set of revoked users R, and the time T in the ciphertext is less than the time T' in the update key. The following is the syntax of RS-ABE.

Definition 4.1 (Revocable-Storage Attribute-Based Encryption)**.** *A revocable-storage (ciphertext-policy) attribute-based encryption (RS-ABE) scheme consists of seven PPT algorithms* ***Setup****,* ***GenKey****,* ***UpdateKey****,* ***Encrypt****,* ***UpdateCT****,* ***RandCT****, and* ***Decrypt****, which are defined as follows:*

Setup$(1^\lambda, \mathcal{U}, T_{max}, N_{max})$*. The setup algorithm takes as input a security parameter* 1^λ*, the universe of attributes* $\mathcal{U}$*, the maximum time* T_{max}*, and the maximum number of users* N_{max}*, and it outputs public parameters* PP *and a master secret key* MK*.*

GenKey(S, u, MK, PP)*. The key generation algorithm takes as input a set of attributes* $S \subseteq \mathcal{U}$*, a user index* $u \in \mathcal{N}$*, the master secret key* MK*, and the public parameters* PP*, and it outputs a private key* $SK_{S,u}$*.*

UpdateKey(T, R, MK, PP)*. The key update algorithm takes as input a time* $T \leq T_{max}$*, a set of revoked users* $R \subseteq \mathcal{N}$*, the master secret key* MK*, and the public parameters* PP*, and it outputs an update key* $UK_{T,R}$*.*

Encrypt$(\mathbb{A}, T, M, PP)$*. The encryption algorithm takes as input an access structure* $\mathbb{A}$*, a time* $T \leq T_{max}$*, a message* M*, and the public parameters* PP*, and it outputs a ciphertext* $CT_{\mathbb{A},T}$*.*

UpdateCT$(CT_{\mathbb{A},T}, T+1, PP)$*. The ciphertext update algorithm takes as input a ciphertext* $CT_{\mathbb{A},T}$ *for an access structure* $\mathbb{A}$ *and a time* T*, a new time* $T+1$ *such that* $T+1 \leq T_{max}$*, and the public parameters* PP*, and it outputs an updated ciphertext* $CT_{\mathbb{A},T+1}$*.*

RandCT$(CT_{\mathbb{A},T}, PP)$*. The ciphertext randomization algorithm takes as input a ciphertext* $CT_{\mathbb{A},T}$ *for an access structure* $\mathbb{A}$ *and a time* T*, and the public parameters* PP*, and it outputs a re-randomized ciphertext* $CT'_{\mathbb{A},T}$*.*

Decrypt$(CT_{\mathbb{A},T}, SK_{S,u}, UK_{T',R}, PP)$*. The decryption algorithm takes as input a ciphertext* $CT_{\mathbb{A},T}$*, a private key* $SK_{S,u}$*, an update key* $UK_{T',R}$*, and the public parameters* PP*, and it outputs a message* M *or the distinguished symbol* $\perp$*.*

The correctness property of RS-ABE is defined as follows: For all PP, MK generated by ***Setup****, all S and u, any $SK_{S,u}$ generated by* ***GenKey****, all $\mathbb{A}$, T, and M, any $CT_{\mathbb{A},T}$ generated by* ***Encrypt*** *or* ***UpdateCT****, all T' and R, any $UK_{T',R}$ generated by* ***UpdateKey****, it is required that:*

- *If $(S \in \mathbb{A}) \wedge (u \notin R) \wedge (T \leq T')$, then* $\textbf{Decrypt}(CT_{\mathbb{A},T}, SK_{S,u}, UK_{T',R}, PP) = M$.
- *If $(S \notin \mathbb{A}) \vee (u \in R) \vee (T' < T)$, then* $\textbf{Decrypt}(CT_{\mathbb{A},T}, SK_{S,u}, UK_{T',R}, PP) = \perp$ *with all but negligible probability.*

Additionally, it requires that the ciphertext distribution of ***RandCT*** *is statistically equal to that of* ***Encrypt****.*

The security property of RS-ABE can be defined by following the security property of revocable ABE that is defined by Boldyreva et al. [3] and by Sahai et al. [40]. In the security game of this security property, the adversary is first given public parameters, and then he can adaptively obtain many private keys for a set of attributes S and a user index u and many update keys for a set of non-revoked users in an update time T'. In the challenge step, the adversary submits a challenge access structure $\mathbb{A}$, a challenge time T^*, and two challenge message M_0^*, M_1^*, and then he receives a challenge ciphertext CT^* that is an encryption of the challenge message M_b^* where b is a random coin used to create the challenge ciphertext. The adversary may continue to request private keys and update keys. Finally, the adversary outputs a guess for the random coin b. If the private keys and the update keys satisfy the non-trivial conditions of the security game and the guess is correct, then the adversary wins the game. The following is the formal definition of the security.

Definition 4.2 (Security)**.** *The security property for RS-ABE is defined in terms of the indistinguishability under a chosen plaintext attack (IND-CPA). The security game for this property is defined as the following game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:*

1. ***Setup****: $\mathcal{C}$ runs* ***Setup*** *to generate the public parameters PP and the master secret key MK, and it gives PP to $\mathcal{A}$.*
2. ***Query 1****: $\mathcal{A}$ may adaptively request a polynomial number of private keys and update keys. $\mathcal{C}$ proceeds as follows:*
    - *If this is a private key query for a set of attributes S and a user index u, then it gives the corresponding private key $SK_{S,u}$ to $\mathcal{A}$ by running* $\textbf{GenKey}(S, u, MK, PP)$*. Note that the adversary is allowed to query only one private key for each user u.*
    - *If this is an update key query for an update time T and a set of revoked users R, then it gives the corresponding update key $UK_{T,R}$ to $\mathcal{A}$ by running* $\textbf{UpdateKey}(T, R, MK, PP)$*. Note that the adversary is allowed to query only one update key for each time T.*
3. ***Challenge****: $\mathcal{A}$ outputs a challenge access structure $\mathbb{A}^*$, a challenge time T^*, and challenge messages $M_0^*, M_1^* \in \mathcal{M}$ of equal length subject to the following restriction:*
    - *It is required that $(S_i \notin \mathbb{A}^*) \vee (u_i \in R_j) \vee (T_j < T^*)$ for all $\{(S_i, u_i)\}$ of private key queries and all $\{(T_j, R_j)\}$ of update key queries.*

    $\mathcal{C}$ chooses a random bit b and gives the ciphertext CT^* to $\mathcal{A}$ by running $\textbf{Encrypt}(\mathbb{A}^*, T^*, M_b^*, PP)$.
4. ***Query 2****: $\mathcal{A}$ may continue to request private keys and update keys subject to the same restrictions as before, and $\mathcal{C}$ gives the corresponding private keys and update keys to $\mathcal{A}$.*

5. ***Guess***: *Finally $\mathcal{A}$ outputs a bit b'.*

The advantage of $\mathcal{A}$ is defined as $\boldsymbol{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the game. A RS-ABE scheme is fully secure under a chosen plaintext attack if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.

Remark 4.3. *In the above security game, it is not needed to explicitly describe **UpdateCT** since the adversary can run **UpdateCT** to the challenge ciphertext by just using PP. Note that the use of **UpdateCT** does not violate the security game because of the restrictions in the game.*

Remark 4.4. *The restriction in the challenge step of the above security game implies that all private keys for a set of attributes S and a user index u that satisfy $S \in \mathbb{A}^*$ should be revoked for all update keys with $T \geq T^*$.*

4.2 Design Principle

In our construction, we take the ciphertext-policy ABE (CP-ABE) scheme of Lewko et al. [26] as the primary encryption scheme, and combine it with our SUE scheme and a revocation mechanism. The revocation mechanism follows the design principle of Boldyreva, Goyal, and Kumar [3] that uses the complete subtree method to securely update the keys of the non-revoked users.

The main challenge in combining all the components is protecting the scheme against a collusion attack, e.g., a non-revoked user with a few attributes should not decrypt more ciphertexts than he is allowed to, given the help of a revoked user with many attributes. To prevent the collusion attack, we use the secret sharing technique in the generation of private keys for ABE and SUE. That is, the master secret key of RS-ABE is separated by using the secret sharing technique, and these separate secrets are served as the master secret key for ABE and SUE respectively. The idea of using the secret sharing technique to prevent a collusion attack and using the CS scheme to revoke a user were introduced by Boldyreva et al. [3], and this idea was widely employed by other schemes that require the revocation functionality [31,40,42][4].

Our RS-ABE scheme uses one full binary tree in the SUE scheme to efficiently manage time and another full binary tree $\mathcal{BT}$ to efficiently manage users in the system. Each user in the system is first assigned to a leaf node in $\mathcal{BT}$ as arbitrary, and he receives private keys for ABE that are related with the path from the root node to the leaf node in $\mathcal{BT}$ from the key generation center. The private keys for ABE are generated by using a random g^{γ_i} as the master key of ABE instead of the original master key g^{α} where the random γ_i is assigned to a node v_i in $\mathcal{BT}$. After that, the key generation center periodically broadcast private keys for SUE as an update key to non-revoked users. The private keys of SUE are also generated by using a random $g^{\alpha-\gamma_i}$ as the master key of SUE instead of the original master key. Note that the master secret key of ABE and the master secret key of SUE in a fixed node v_i in $\mathcal{BT}$ are separated by using the secret sharing technique, and these master secret key can be combined as $g^{\alpha} = g^{\gamma_i} \cdot g^{\alpha-\gamma_i}$.

To create a ciphertext, a sender creates a ciphertext for ABE that is associated with an access structure $\mathbb{A}$ and a ciphertext for SUE that is associated with a time T by using the same random exponent s to recover the secret that are shared among private keys for ABE and private keys for SUE. A receiver who has a private key associated with a set of attributes S and an update key associated with an update time T' and a set of revoked users R can find the private key of ABE and the private key of SUE that corresponds to a common node v in BT if he is not contained in R. Thus the receiver can decrypt the ciphertext by using the private

[4]The secret sharing technique of Boldyreva et al. [3] is different with that of other schemes since the scheme of Boldyreva et al. uses a polynomial secret sharing whereas other schemes use a $(2,2)$-additive secret sharing. Note that we also use this simple $(2,2)$-additive secret sharing.

key of ABE and the private key of SUE if $S \in \mathbb{A}$ and $T \leq T'$. That is, the secret sharing in the generation of private keys and update keys and the same random exponent in the ciphertext prevent the collusion attack.

4.3 Construction

Before presenting our RS-ABE scheme, we first describe the (ciphertext-policy) ABE scheme of Lewko et al. [26]. Note that their ABE scheme has the restriction such that the attributes are only used once in the access structure and the size of the total attributes in the system is small, but they showed that these restrictions can be removed.

ABE.Setup($GDS, \mathcal{U}$**):** This algorithm takes as input a group description string GDS and the universe of attributes $\mathcal{U}$. It chooses a random exponent $a \in \mathbb{Z}_N$ and random elements $\{T_j\}_{j \in \mathcal{U}} \in \mathbb{G}_{p_1}$, $Y \in \mathbb{G}_{p_3}$, and a random exponent $\gamma \in \mathbb{Z}_N$. It outputs the master secret key $MK = (\gamma, Y)$ and the public parameters as

$$PP = \Big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_1,\ g^a,\ \{T_j\}_{j \in \mathcal{U}},\ \Lambda = e(g,g)^{\gamma} \Big).$$

ABE.GenKey(S, MK, PP**):** This algorithm takes as input a set of attributes S, the master secret key MK, and the public parameters PP. It chooses a random exponent $r \in \mathbb{Z}_N$ and random elements $Y_0, Y_1, \{Y_{2,j}\}_{j \in S} \in \mathbb{G}_{p_3}$. It outputs a private key that implicitly includes S as

$$SK_S = \Big(K_0 = g^{\gamma} g^{ar} Y_0,\ K_1 = g^r Y_1,\ \big\{ K_{2,j} = T_j^r Y_{2,j} \big\}_{j \in S} \Big).$$

ABE.Encrypt($\mathbb{A}, s, PP$**):** This algorithm takes as input an LSSS access structure $\mathbb{A} = (A, \rho)$ where A is an $l \times n$ matrix and ρ is a map from each row A_j of A to an attribute $\rho(j)$, a random exponent $s \in \mathbb{Z}_N$, and the public parameters PP. It first sets a random vector $\vec{v} = (s, v_2, \ldots, v_n)$ by selecting random exponents $v_2, \ldots, v_n \in \mathbb{Z}_N$. It selects random exponents $s_1, \ldots, s_l \in \mathbb{Z}_N$ and outputs a ciphertext header that implicitly includes $\mathbb{A}$ as

$$CH_{\mathbb{A}} = \Big(C_0 = g^s,\ \big\{ C_{1,j} = g^{aA_j \cdot \vec{v}} T_{\rho(j)}^{s_j},\ C_{2,j} = g^{-s_j} \big\}_{1 \leq j \leq l} \Big)$$

and a session key $EK = \Lambda^s$.

ABE.RandCT($CH_{\mathbb{A}}, s', PP$**):** This algorithm takes as input a ciphertext header $CH_{\mathbb{A}}$ for an LSSS access structure $\mathbb{A} = (A, \rho)$, a new random exponent $s' \in \mathbb{Z}_N$, and the public parameters PP. It first sets a new vector $\vec{v}' = (s', v'_2, \ldots, v'_n)$ by selecting random exponents $v'_2, \ldots, v'_n \in \mathbb{Z}_N$. It selects random exponents $s'_1, \ldots, s'_l \in \mathbb{Z}_N$ and outputs a ciphertext header as

$$CH'_{\mathbb{A}} = \Big(C'_0 = C_0 \cdot g^{s'},\ \big\{ C'_{1,j} = C_{1,j} \cdot g^{aA_j \cdot \vec{v}'} T_{\rho(j)}^{s'_j},\ C'_{2,j} = C_{2,j} \cdot g^{-s'_j} \big\}_{1 \leq j \leq l} \Big)$$

and a partial session key $EK' = \Lambda^{s'}$ that will be multiplied with the session of $CH_{\mathbb{A}}$.

ABE.Decrypt($CH_{\mathbb{A}}, SK_S, PP$**):** This algorithm takes as input a ciphertext header $CH_{\mathbb{A}}$ for an LSSS access structure $\mathbb{A} = (A, \rho)$, a private key SK_S for a set of attributes S, and the public parameters PP. If S satisfies $\mathbb{A}$, then it computes constants $\omega_j \in \mathbb{Z}_p$ such that $\sum_{\rho(j) \in S} \omega_j A_j = (1, 0, \ldots, 0)$ and outputs a session key as

$$EK = e(C_0, K_0) / \prod_{\rho(j) \in S} \big(e(C_{1,j}, K_{1,j}) \cdot e(C_{2,j}, K_{2,j}) \big)^{\omega_j}.$$

Otherwise, it outputs $\perp$.

Let $\mathcal{M}$ be $\mathbb{G}_T$. Our RS-ABE scheme that uses the above ABE scheme as a building block is described as follows:

RS-ABE.Setup($1^\lambda, \mathcal{U}, T_{max}, N_{max}$)**:** This algorithm takes as input a security parameter 1^λ, the universe of attributes $\mathcal{U}$, the maximum time T_{max}, and the maximum number of users N_{max}.

1. It first generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of composite order $N = p_1 p_2 p_3$ where p_1, p_2, and p_3 are random primes. Let g_1 be the generator of $\mathbb{G}_{p_1}$. It sets $GDS = ((N, \mathbb{G}, \mathbb{G}_T, e), g_1, p_1, p_2, p_3)$.
2. It obtains MK_{ABE}, PP_{ABE} and MK_{SUE}, PP_{SUE} by running **ABE.Setup**($GDS, \mathcal{U}$) and **SUE.Setup** (GDS, T_{max}) respectively. It also obtains $\mathcal{BT}$ by running **CS.Setup**(N_{max}) and assigns a random exponent $\gamma_i \in \mathbb{Z}_N$ to each node v_i in $\mathcal{BT}$.
3. It selects a random exponent $\alpha \in \mathbb{Z}_N$, and then it outputs the master secret key $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$ and the public parameters as $PP = \big(PP_{ABE}, PP_{SUE}, g = g_1, \Omega = e(g,g)^\alpha\big)$.

RS-ABE.GenKey(S, u, MK, PP)**:** This algorithm takes as input a set of attributes S, a user index u, the master secret key $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$, and the public parameters PP.

1. It first obtains a private set $PV_u = \{S'_{j_0}, \ldots, S'_{j_d}\}$ by running **CS.Assign**($\mathcal{BT}, u$) and retrieves $\{\gamma_{j_0}, \ldots, \gamma_{j_d}\}$ from $\mathcal{BT}$ where S'_{j_k} is associated with a node v_{j_k} and γ_{j_k} is assigned to the node v_{j_k}.
2. For $0 \le k \le d$, it sets $MK'_{ABE} = (\gamma_{j_k}, Y)$ and obtains $SK_{ABE,k}$ by running **ABE.GenKey**(S, MK'_{ABE}, PP_{ABE}).
3. It outputs a private key as $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d}\big)$.

RS-ABE.UpdateKey(T, R, MK, PP)**:** This algorithm takes as input an update time T, a set of revoked users R, the master secret key MK, and the public parameters PP.

1. It first obtains a covering set $CV_R = \{S'_{i_1}, \ldots, S'_{i_m}\}$ by running **CS.Cover**($\mathcal{BT}, R$) and retrieves $\{\gamma_{i_1}, \ldots, \gamma_{i_m}\}$ from $\mathcal{BT}$ where S'_{i_k} is associated with a node v_{i_k} and γ_{i_k} is assigned to the node v_{i_k}.
2. For $1 \le k \le m$, it sets $MK'_{SUE} = (\alpha - \gamma_{i_k}, Y)$ and obtains $SK_{SUE,k}$ by running **SUE.GenKey**(T, MK'_{SUE}, PP_{SUE}).
3. It outputs an update key that implicitly includes T and R as $UK_{T,R} = \big(CV_R, SK_{SUE,1}, \ldots, SK_{SUE,m}\big)$.

RS-ABE.Encrypt($\mathbb{A}, T, M, PP$)**:** This algorithm takes as input an LSSS access structure $\mathbb{A}$, a time T, a message $M \in \mathcal{M}$, and the public parameters PP.

1. It selects a random exponent $s \in \mathbb{Z}_N$ and obtains CH_{ABE} and CH_{SUE} by running **ABE.Encrypt** ($\mathbb{A}, s, PP_{ABE}$) and **SUE.Encrypt**(T, s, PP_{SUE}) respectively. Note that it ignores two partial session keys that are returned by **ABE.Encrypt** and **SUE.Encrypt**.
2. It outputs a ciphertext that implicitly includes T as $CT_{\mathbb{A},T} = \big(CH_{ABE}, CH_{SUE}, C = \Omega^s \cdot M\big)$.

RS-ABE.UpdateCT($CT_{\mathbb{A},T}, T+1, PP$)**:** This algorithm takes as input a ciphertext $CT_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE}, C)$ for an LSSS access structure $\mathbb{A}$ and a time T, a new time $T+1$, and the public parameters PP.

1. It first obtains CH'_{SUE} by running **SUE.UpdateCT**($CH_{SUE}, T+1, PP_{SUE}$).
2. It outputs an updated ciphertext that implicitly includes $T+1$ as $CT_{\mathbb{A},T+1} = \big(CH_{ABE}, CH'_{SUE}, C\big)$.

RS-ABE.RandCT($CT_{\mathbb{A},T}, PP$): This algorithm takes as input a ciphertext $CT_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE}, C)$ and the public parameters PP.

1. It selects a random exponent $s' \in \mathbb{Z}_N$ and obtains CH'_{ABE} and CH'_{SUE} by running **ABE.RandCT** (CH_{ABE}, s', PP_{ABE}) and **SUE.RandCT**(CH_{SUE}, s', PP_{SUE}), respectively.
2. It outputs a re-randomized ciphertext as $CT'_{\mathbb{A},T} = \left(CH'_{ABE}, CH'_{SUE}, C' = C \cdot \Omega^{s'}\right)$.

RS-ABE.Decrypt($CT_{\mathbb{A},T}, SK_{S,u}, UK_{T',R}, PP$): This algorithm takes as input a ciphertext $CT_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE}, C)$ for an LSSS access structure $\mathbb{A}$ and a time T, a private key $SK_{S,u} = (PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d})$ for a set of attributes S and a user index u, an update key $UK_{T',R} = (CV_R, SK_{SUE,1}, \ldots, SK_{SUE,m})$ for an update time T' and a set of revoked users R, and the public parameters PP.

1. If $u \notin R$, then it obtains (S_i, S_j) by running **CS.Match**(CV_R, PV_u). Otherwise, it outputs $\perp$.
2. If $S \in \mathbb{A}$ and $T \leq T'$, then it obtains EK_{ABE} and EK_{SUE} by running **ABE.Decrypt**$(CH_{ABE}, SK_{ABE,j}, PP_{ABE})$ and **SUE.Decrypt**$(CH_{SUE}, SK_{SUE,i}, PP_{SUE})$ respectively and outputs a message M by computing $C \cdot \left(EK_{ABE} \cdot EK_{SUE}\right)^{-1}$. Otherwise, it outputs $\perp$.

Remark 4.5. *The ciphertext update algorithm of our scheme just outputs a valid updated ciphertext since a past ciphertext will be erased in most applications. However, the definition of Sahai et al. [40] requires that the output of **UpdateCT** should be equally distributed with that of **Encrypt**. Our scheme also can meet this strong requirement by applying **RandCT** to the output of **UpdateCT**.*

4.4 Correctness

The correctness of RS-ABE easily follows from the correctness of ABE, SUE, and SC. Suppose that the private key of a user is associated with a set of attributes S and a user index u, the update key of the user is associated with an update time T' and a set of revoked users R, and the ciphertext is associated with an access structure $\mathbb{A}$ and a time T. If $u \notin R$, then the user can find a private key for ABE and a private key for SUE from the correctness of the CS scheme by running **CS.Match**. Next, if $S \in \mathbb{A}$, then he can obtain a session key for ABE by running **ABE.Decrypt** from the correctness of ABE. If $T \leq T'$, he also can obtain a session key for SUE by running **SUE.Decrypt** from the correctness of SUE. Therefore he can obtain the correct session key from two session key since the same random exponent s was used.

The output of the ciphertext update algorithm is a valid ciphertext for RS-ABE since the output of **SUE.UpdateCT** is also a valid ciphertext for SUE. The ciphertext distribution of the ciphertext randomization algorithm is statistically equal to that of the encryption algorithm since **ABE.RandCT** and **SUE.RandCT** outputs statistically equivalent ciphertext headers for ABE and SUE respectively.

4.5 Security Analysis

We prove the security of our RS-ABE scheme under three static assumptions. Note that we don't prove the security of our RS-ABE scheme based on that of SUE and ABE schemes in a black-box manner, since our scheme merges two schemes in a non-black-box way by using secret-sharing for key generation.

Theorem 4.6. *The above RS-ABE scheme is fully secure under a chosen plaintext attack if Assumptions 1, 2, and 3 hold. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) \leq \boldsymbol{Adv}_{\mathcal{B}}^{A1}(\lambda) + O(q^2) \cdot \boldsymbol{Adv}_{\mathcal{B}}^{A2}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}}^{A3}(\lambda)$ where q is the maximum number of private key and update key queries of $\mathcal{A}$.*

Proof. For semi-functional types, we let g_2 denote a fixed generator of the subgroup $\mathbb{G}_{p_2}$. The semi-functional types of private keys, update keys, and ciphertexts are defined as follows:

RS-ABE.GenKeySF. This algorithm first creates a normal private key $SK'_{S,u} = (PV_u, SK'_{ABE,0}, \dots, SK'_{ABE,d})$ using the master secret key where $PV_u = \{S_{j_0}, \dots, S_{j_d}\}$ and $SK'_{ABE,k} = (K'_0, K'_1, \{K'_{2,j}\})$. For $0 \le k \le d$, it generates a random exponent $\delta_{k,0} \in \mathbb{Z}_N$ once for the node $v_k \in \mathcal{BT}$ that is related with $S_{j_k} \in PV_u$ and builds a semi-functional ABE private key as

$$SK_{ABE,k} = \Big(K_0 = K'_0 g_2^{\delta_{k,0}},\ K_1 = K'_1,\ \{K_{2,j} = K'_{2,j}\}_{j \in S} \Big).$$

It outputs a semi-functional private key as $SK_{S,u} = (PV_u, SK_{ABE,0}, \dots, SK_{ABE,d})$. Note that it uses the same $\delta_{k,0}$ for the node v_k in $\mathcal{BT}$.

RS-ABE.UpdateKeySF. This algorithm first creates a normal update key $UK'_{T,R} = (CV_R, SK'_{SUE,1}, \dots, SK'_{SUE,m})$ using the master secret key where $CV_R = \{S_{i_1}, \dots, S_{i_n}\}$ and $SK'_{SUE,k} = (K'_0, K'_1, K'_{2,1}, \dots, K'_{2,n})$. For $1 \le k \le n$, it generates a random exponent $\delta_{k,1} \in \mathbb{Z}_N$ once for the node $v_k \in \mathcal{BT}$ that is related with $S_{i_k} \in CV_R$ and builds a semi-functional SUE private key as

$$SK_{SUE,k} = \Big(K_0 = K'_0 g_2^{\delta_{k,1}},\ K_1 = K'_1,\ K_{2,1} = K'_{2,1},\ \dots,\ K_{2,n} = K'_{2,n} \Big).$$

It outputs a semi-functional update key as $UK_{T,R} = (CV_R, SK_{SUE,1}, \dots, SK_{SUE,n})$. Note that it uses the same $\delta_{v_k,1}$ for the node v_k in $\mathcal{BT}$.

RS-ABE.EncryptSF. This algorithm first creates a normal ciphertext $CT' = (CH'_{ABE}, CH'_{SUE}, C')$ where $CH'_{ABE} = (C'_0, \{C'_{1,j}, C'_{2,j}\})$ and $CH'_{SUE} = (CH'^{(0)}, \dots, CH'^{(d)})$. It chooses a random exponent $c \in \mathbb{Z}_N$, a random vector $\vec{u} = (u_1, \dots, u_n) \in \mathbb{Z}_N^n$, and random exponents $y_1, \dots, y_d \in \mathbb{Z}_N$. It builds a semi-functional ciphertext header for ABE as

$$CH_{ABE} = \Big(C_0 = C'_0 g_2^c,\ \{C_{1,j} = C'_{1,j} g_2^{A_j \cdot \vec{u} + z_{\rho(j)} y_i},\ C_{2,j} = C'_{2,j} g_2^{-y_j}\}_{1 \le j \le l} \Big).$$

It builds a semi-functional ciphertext header CH_{SUE} for SUE as the same as Theorem 3.10 except that the same exponent c that is used for ABE is also used here. It outputs a semi-functional ciphertext as $CT = (CH_{ABE}, CH_{SUE}, C')$.

Note that if we use a semi-functional private key and a semi-functional update key to decrypt a semi-functional ciphertext, then we fail to decrypt the ciphertext since an additional random element $e(g_2, g_2)^{c(\delta_{k,0} + \delta_{k,1})}$ is left.

In the general security proof of dual system encryption, the normal challenge ciphertext is first changed to a semi-functional one, and then the private keys are also changed to semi-functional ones one by one through hybrid games [26, 27, 45]. However, we cannot use this proof strategy directly since a private key SK consists of many ABE private keys $\{SK_{ABE,i}\}$ and an update key UK also consists of many SUE private keys $\{SK_{SUE,j}\}$. Instead we use different hybrid games that change ABE private keys (or SUE private keys) that are related with a specified node v in $\mathcal{BT}$ from normal ones to semi-functional ones one by one through hybrid games. Note that this proof strategy was introduced by Sahai et al. [40].

For the hybrid games that change ABE private keys (or SUE private keys) that are related with a node v from normal ones to semi-functional ones, we need to state additional information of a node v in $\mathcal{BT}$ that is related with an ABE private key (or an SUE private key). Note that ABE private keys in a private key and

SUE private keys of an update key are related with some nodes in $\mathcal{BT}$ since the private key and the update key of our RS-ABE scheme are related with a private set PV_u and a covering set CV_R respectively. Thus we associate an ABE private key (or an SUE private key) with a tuple of indexes (i_n, i_c) to state additional information about the node v that is related with the ABE private key (or the SUE private key) where i_n is a node index and i_c is a counter index.

Suppose that an ABE private key (or an SUE private key) is related with a node v in $\mathcal{BT}$. The node index i_n for ABE private keys (or SUE private keys) is assigned as follows: If the node v appears first time in queries, then we set i_n as the number of distinct node v' in previous queries plus one. If the node v already appeared before in queries, then we set i_n as the value i'_n of previous ABE private key (or SUE private key) with the same node v. The counter index i_c for ABE private keys is assigned as follows: If the node v appears first time in ABE private key queries, then we set i_c as one. If the node v appeared before in ABE private key queries, then we set i_c as the number of ABE private keys with the node v that appeared before plus one. Similarly, we assigns the counter index i_c for SUE private keys.

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

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

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

Game $\mathbf{G}_2$. Next, we define a new game $\mathbf{G}_2$. In this game, the challenge ciphertext, all private keys, and all update keys are semi-functional. For the security proof, we define a sequence of games $\mathbf{G}_{1,1}, \dots, \mathbf{G}_{1,h}, \dots, \mathbf{G}_{1,q_n}$ where $\mathbf{G}_1 = \mathbf{G}_{1,0}$ and q_n is the number of nodes in $\mathcal{BT}$ that are used in ABE private keys of private key queries and SUE private keys of update key queries. In the game $\mathbf{G}_{1,h}$ for $1 \leq h \leq q_n$, the challenge ciphertext is semi-functional, ABE private keys in private keys and SUE private keys in update keys with i_n index such that $i_n \leq h$ are semi-functional, and the remaining ABE private keys and SUE private keys with a node index i_n such that $h < i_n$ are normal. It is obvious that $\mathbf{G}_{1,q_n} = \mathbf{G}_2$. Note that $q_n < q_{sk} \cdot \log N_{max} + q_{uk} \cdot r_{max} \log N_{max}$ from Lemma 2.2 where q_{sk} is the number of private key queries, q_{uk} is the number of update key queries, N_{max} is the number of leaf nodes, and r_{max} is the maximum number of revoked set.

Game $\mathbf{G}_3$. In the final game $\mathbf{G}_3$, all private keys, all update keys, and the ciphertext are semi-functional, but the session key is random.

Let $\mathbf{Adv}_{\mathcal{A}}^{G_j}$ be the advantage of $\mathcal{A}$ in the game $\mathbf{G}_j$. We easily obtain that $\mathbf{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) = \mathbf{Adv}_{\mathcal{A}}^{G_0}$, $\mathbf{Adv}_{\mathcal{A}}^{G_1} = \mathbf{Adv}_{\mathcal{A}}^{G_{1,0}}$, and $\mathbf{Adv}_{\mathcal{A}}^{G_3} = 0$. From the following Lemmas 4.7, 4.8, and 4.19, we obtain the equation

$$\begin{aligned}
\mathbf{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) &= \mathbf{Adv}_{\mathcal{A}}^{G_0} + (\mathbf{Adv}_{\mathcal{A}}^{G_1} - \mathbf{Adv}_{\mathcal{A}}^{G_1}) + \sum_{h=1}^{q_n} \left(\mathbf{Adv}_{\mathcal{A}}^{G_{1,h}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,h}}\right) - \mathbf{Adv}_{\mathcal{A}}^{G_3} \\
&\leq \left|\mathbf{Adv}_{\mathcal{A}}^{G_0} - \mathbf{Adv}_{\mathcal{A}}^{G_1}\right| + \sum_{h=1}^{q_n} \left|\mathbf{Adv}_{\mathcal{A}}^{G_{1,h-1}} - \mathbf{Adv}_{\mathcal{A}}^{G_{1,h}}\right| + \left|\mathbf{Adv}_{\mathcal{A}}^{G_2} - \mathbf{Adv}_{\mathcal{A}}^{G_3}\right| \\
&\leq \mathbf{Adv}_{\mathcal{B}}^{A1}(\lambda) + (4\sum_{h=1}^{q_n}(q_{h,abe} + q_{h,sue}) + 2q_n) \cdot \mathbf{Adv}_{\mathcal{B}}^{A2}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{A3}(\lambda).
\end{aligned}$$

This completes our proof. □

Lemma 4.7. *If the Assumption 1 holds, then no polynomial-time adversary can distinguish between* $\mathbf{G}_0$ *and* $\mathbf{G}_1$ *with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguishes between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the Assumption 1 using $\mathcal{A}$ is given: a challenge tuple $D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_3})$ and T where $Z = Z_0 = X_1 \in \mathbb{G}_{p_1}$ or $Z = Z_1 = X_1 R_1 \in \mathbb{G}_{p_1 p_2}$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ first select random exponents $a, \{t_i'\}, \gamma \in \mathbb{Z}_N$ and sets $MK_{ABE} = (\gamma, Y = g_{p_3})$ and the public parameters of ABE as

$$PP_{ABE} = \Big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_{p_1}, g^a, \{T_i = g^{t_i'}\}, \Lambda = e(g,g)^{\gamma}\Big).$$

It sets MK_{SUE} and PP_{SUE} as the same as Lemma 3.11. It also obtains $\mathcal{BT}$ by running **CS.Setup** and assigns a random exponent γ_i to each node v_i in $\mathcal{BT}$. It selects a random exponent $\alpha \in \mathbb{Z}_N$ and sets $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$ and $PP = \big(PP_{ABE}, PP_{SUE}, g = g_{p_1}, \Omega = e(g,g)^{\alpha}\big)$.

Query 1: In the query step, $\mathcal{A}$ adaptively request private keys and update keys. If this is a private key query, $\mathcal{B}$ creates a normal private key by running **RS-ABE.GenKey** since it knows the master secret key. If this is an update key query, $\mathcal{B}$ creates a normal update key by running **RS-ABE.UpdateKey** since it knows the master secret key. Note that it cannot create a semi-functional one since it does not know g_{p_2}.

Challenge: In the challenge step, $\mathcal{A}$ submits a challenge access structure $\mathbb{A}^*$, a challenge time T^*, and challenge messages M_0^*, M_1^*. To make the challenge ciphertext, $\mathcal{B}$ implicitly sets g^s to be the $\mathbb{G}_{p_1}$ part of Z. It selects random exponents $v_2', \ldots, v_n', s_1', \ldots, s_l' \in \mathbb{Z}_N$ and sets a random vector $\vec{v}' = (1, v_2', \ldots, v_n')$. Next, it creates a ciphertext header for ABE by implicitly setting $\vec{v} = (s, sv_2', \ldots, sv_n')$ and $s_j = ss_j'$ as

$$CH_{ABE} = \Big(C_0 = Z,\ \{C_{1,j} = (Z)^{aA_j \cdot \vec{v}'} (Z)^{t'_{\rho(j)} s'_j},\ C_{2,j} = (Z)^{-s'_j}\}_{1 \le j \le l}\Big).$$

It creates a ciphertext header CH_{SUE} as the same as Lemma 3.11 by using the same element Z of the assumption. Next, it flips a random coin $b \in \{0,1\}$ and sets the challenger ciphertext as $CT^*_{\mathbb{A}^*, T^*} = (CH_{ABE}, CH_{SUE}, C = e(Z,g)^{\alpha} \cdot M_b^*)$. It gives the challenge ciphertext to $\mathcal{A}$.

Query 2: Same as Query 1.

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

If $Z \in \mathbb{G}_{p_1}$, then the above challenge ciphertext is normal. If $Z \in \mathbb{G}_{p_1 p_2}$, the above challenge ciphertext is semi-functional by the CRT. We omit the detailed analysis since it is the same as Lemma 7 in [26] and Lemma 3.11. This completes our proof. □

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

Proof. We first divide the behavior of an adversary as two types: Type-I and Type-II. We next show that this lemma holds for two types of the adversary. The two types of adversaries are formally defined as follows:

Type-I. An adversary is Type-I if for the ABE private key with a node index h it queries an attribute set S such that $S \notin \mathbb{A}^*$ where $\mathbb{A}^*$ is the challenge access structure of the adversary.

Type-II. An adversary is Type-II if for the SUE private key with a node index h it queries an update time T such that $T < T^*$ where T^* is the challenge time of the adversary.

Note that these two types of an adversary are not a partition since an adversary can be both Type-I and Type-II, but these two types cover all adversaries because of the restrictions in the security model. That is, if an adversary is neither Type-I nor Type-II, then he can easily decrypt the challenge ciphertext by querying both an ABE private key for $S \in \mathbb{A}^*$ with the node index h and an SUE private key for $T \geq T^*$ with the same node index h. However, this is not allowed by the restrictions of the security model in Definition 4.2.

For the Type-I adversary $\mathcal{A}_I$, we define hybrid games $\mathbf{H}_{0,2}, \mathbf{H}_{1,1}, \mathbf{H}_{1,2}, \ldots, \mathbf{H}_{q_{h,abe},1}, \mathbf{H}_{q_{h,abe},2} = \mathbf{H}'_{q_{h,abe},2}$, $\mathbf{H}'_{q_{h,abe},1}, \ldots, \mathbf{H}'_{1,2}, \mathbf{H}'_{1,1}, \mathbf{H}'_{0,2}, \mathbf{H}''$ where $\mathbf{H}_{0,2} = \mathbf{G}_{1,h-1}$, $\mathbf{H}'' = \mathbf{G}_{1,h}$, and $q_{h,abe}$ is the number of ABE private keys with a node index h.

Game $\mathbf{H}_{h_c,1}$. This game $\mathbf{H}_{h_c,1}$ for $1 \leq h_c \leq q_{h,abe}$ is almost the same as $\mathbf{G}_{1,h-1}$ except the generation of ABE private keys with a node index h. That is, ABE private keys and SUE private keys with a node index i_n such that $i_n < h$ are generated as semi-functional type, ABE private keys and SUE private keys with a node index i_n such that $h < i_n$ are generated as normal type, and SUE private keys with a node index $i_n = h$ are generated as normal type. ABE private keys with a node index $i_n = h$ and a counter index i_c are generated as follows:

- $i_c < h_c$: It first generate a normal ABE private key $SK' = (K'_0, K'_1, \{K'_{2,j}\})$. It selects a new random exponent $\delta \in \mathbb{Z}_N$ and outputs the ABE private key as
$$SK_{ABE} = \Big(K_0 = K'_0 g_2^{\delta},\ K_1 = K'_1,\ \{K_{2,j} = K'_{2,j}\}\Big).$$
Note that this ABE private key is the same as the semi-functional private key of type 2 in [26], but it is different with the ABE private keys in a semi-functional RS-ABE private key since a different random δ is selected for the same node v.
- $i_c = h_c$: It first generates a normal ABE private key $SK' = (K'_0, K'_1, \{K'_{2,j}\})$. It selects new random exponents $\delta, b \in \mathbb{Z}_N$ and outputs the ABE private keys as
$$SK_{ABE} = \Big(K_0 = K'_0 g_2^{\delta},\ K_1 = K'_1 g_2^{b},\ \{K_{2,j} = K'_{2,j} g_2^{z_{\rho(j)} b}\}\Big).$$
Note that this ABE private key is the same as the semi-functional private key of type 1 in [26].
- $i_c > h_c$: It simply builds a normal ABE private key.

Game $\mathbf{H}_{h_c,2}$. This game $\mathbf{H}_{h_c,2}$ is almost the same as $H_{h_c,1}$ except that the ABE private key with a counter index i_c such that $i_c = h_c$ is generated with $b = 0$. This corresponds to generating first h_c ABE private keys for the node index h as semi-functional type 2 in [26] and remaining ABE private keys for the node index h as normal type.

Game $\mathbf{H}'_{h_c,1}$. This game $\mathbf{H}'_{h_c,1}$ is almost the same as $\mathbf{H}_{h_c,1}$ except the generation of ABE private keys with a counter index h_c. Let $SK''_{ABE} = (K''_0, K''_1, \{K''_{2,j}\})$ be an ABE private key with a counter index h_c that is generated from the game $\mathbf{H}_{h_c,1}$. It generates a random exponent $\delta_{k,0}$ once for the node v_k that is related with this ABE private key and builds a new ABE private key as
$$SK_{ABE} = \Big(K_0 = K''_0 g_2^{\delta_{k,0}},\ K_1 = K''_1,\ \{K_{2,j} = K''_{2,j}\}\Big).$$

Game $\mathbf{H}'_{h_c,2}$. This game $\mathbf{H}'_{h_c,2}$ is almost the same as $\mathbf{H}_{h_c,2}$ except ABE private keys with a counter index h_c. The modification is similar to the game $\mathbf{H}'_{h_c,1}$. We should note that $\mathbf{H}_{q_{h,abe},2} = \mathbf{H}'_{q_{h,abe},2}$ since the random element g_2^{δ} in each ABE private keys subsumes the additional fixed element $g_2^{\delta_{k,0}}$. In the game $\mathbf{H}'_{0,2}$, all ABE private keys with a node index h are semi-functional of type 2 where $\delta_{k,0}$ is fixed for the node v_k that is related with the node index h, but all SUE private keys with a node index h are still normal.

Game $\mathbf{H}''$. This game $\mathbf{H}''$ is the same as $\mathbf{G}_{1,h}$. Compared to the game $\mathbf{H}'_{0,2}$, all SUE private keys with a node index h are changed to be semi-functional of type 2 where $\delta_{k,1}$ is fixed for the node v_k.

Let $\mathbf{Adv}^{H}_{\mathcal{A}_I}$ be the advantage of $\mathcal{A}_I$ in a game $\mathbf{H}$. From the following Claims 4.9, 4.10, 4.11, 4.12, and 4.13, we can obtain the following equation

$$\begin{aligned}
\mathbf{Adv}^{H_{0,2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H''}_{\mathcal{A}_I} &= \mathbf{Adv}^{H_{0,2}}_{\mathcal{A}_I} + \sum_{h_c=1}^{q_{h,abe}} \left(\mathbf{Adv}^{H_{h_c,1}}_{\mathcal{A}_I} - \mathbf{Adv}^{H_{h_c,1}}_{\mathcal{A}_I}\right) + \sum_{h_c=1}^{q_{h,abe}} \left(\mathbf{Adv}^{H_{h_c,2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H_{h_c,2}}_{\mathcal{A}_I}\right) + \\
&\quad \sum_{h_c=1}^{q_{h,abe}} \left(\mathbf{Adv}^{H'_{h_c,1}}_{\mathcal{A}_I} - \mathbf{Adv}^{H'_{h_c,1}}_{\mathcal{A}_I}\right) + \sum_{h_c=0}^{q_{h,abe}-1} \left(\mathbf{Adv}^{H'_{h_c,2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H'_{h_c,2}}_{\mathcal{A}_I}\right) - \mathbf{Adv}^{H''}_{\mathcal{A}_I} \\
&\leq \sum_{h_c=1}^{q_{h,abe}} \left|\mathbf{Adv}^{H_{h_c-1,2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H_{h_c,1}}_{\mathcal{A}_I}\right| + \sum_{h_c=1}^{q_{h,abe}} \left|\mathbf{Adv}^{H_{h_c,1}}_{\mathcal{A}_I} - \mathbf{Adv}^{H_{h_c,2}}_{\mathcal{A}_I}\right| + \\
&\quad \sum_{h_c=1}^{q_{h,abe}} \left|\mathbf{Adv}^{H'_{h_c,2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H'_{h_c,1}}_{\mathcal{A}_I}\right| + \sum_{h_c=1}^{q_{h,abe}} \left|\mathbf{Adv}^{H'_{h_c,1}}_{\mathcal{A}_I} - \mathbf{Adv}^{H'_{h_c-1,2}}_{\mathcal{A}_I}\right| + \left|\mathbf{Adv}^{H'_{0,2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H''}_{\mathcal{A}_I}\right| \\
&\leq (4q_{h,abe}+1) \cdot \mathbf{Adv}^{A2}_{\mathcal{B}}(\lambda).
\end{aligned}$$

For the Type-II adversary $\mathcal{A}_{II}$, we also define hybrid games $\mathbf{I}_{0,2}, \mathbf{I}_{1,1}, \mathbf{I}_{1,2}, \ldots, \mathbf{I}_{q_{h,sue},1}, \mathbf{I}_{q_{h,sue},2} = \mathbf{I}'_{q_{h,sue},2}, \mathbf{I}'_{q_{h,sue},1}, \ldots, \mathbf{I}'_{1,2}, \mathbf{I}'_{1,1}, \mathbf{I}'_{0,2}, \mathbf{I}''$ where $\mathbf{I}_{0,2} = \mathbf{G}_{1,h-1}, \mathbf{I}'' = \mathbf{G}_{1,h}$, and $q_{h,sue}$ is the number of SUE private keys with a node index h.

Game $\mathbf{I}_{h_c,1}$. This game $\mathbf{I}_{h_c,1}$ is almost the same as $\mathbf{G}_{1,h-1}$ except the generation of SUE private keys with a node index h. That is, ABE private keys and SUE private keys with a node index i_n such that $i_n < h$ are generated as semi-functional type, ABE private keys and SUE private keys with a node index i_n such that $h < i_n$ are generated as normal type, and ABE private keys with a node index $i_n = h$ are generated as normal type. SUE private keys with a node index $i_n = h$ and a counter index i_c are generated as follows:

- $i_c < h_c$: It first generate a normal SUE private key $SK' = (K'_0, K'_1, K'_{2,1}, \ldots, K'_{2,n})$. It selects a new random exponent $\delta \in \mathbb{Z}_N$ and outputs a semi-functional SUE private key of type 2 as

$$SK_{SUE} = \Big(K_0 = K'_0 g_2^{\delta},\ K_1 = K'_1,\ K_{2,1} = K'_{2,1},\ \ldots,\ K_{2,n} = K'_{2,n}\Big).$$

  Note that this ABE private key is different with the SUE private keys in a semi-functional RS-ABE update key since a different random δ is selected for the same node v.

- $i_c = h_c$: It first generates a normal SUE private key $SK' = (K'_0, K'_1, K'_{2,1}, \ldots, K'_{2,n})$. It selects new random exponents $\delta, b \in \mathbb{Z}_N$ and outputs a semi-functional SUE private keys of type 1 as

$$SK_{SUE} = \Big(K_0 = K'_0 g_2^{\delta},\ K_1 = K'_1 g_2^{b},\ K_{2,1} = K'_{2,1} g_2^{f_{1,L[1]}(L|_1)b},\ \ldots,\ K_{2,1} = K'_{2,1} g_2^{f_{n,L[n]}(L|_n)b},\ \Big).$$

- $i_c > h_c$: It simply builds a normal SUE private key.

Game $\mathbf{I}_{h_c,2}$. This game $\mathbf{I}_{h_c,2}$ is almost the same as $I_{h_c,1}$ except that the SUE private key with a counter index i_c such that $i_c = h_c$ is generated with $b = 0$. This corresponds to generating first h_c SUE private keys for the node index h as semi-functional type 2 and remaining ABE private keys for the node index h as normal type.

Game $\mathbf{I}'_{h_c,1}$. This game $\mathbf{I}'_{h_c,1}$ is almost the same as $\mathbf{I}_{h_c,1}$ except that SUE private keys with a counter index h_c. Let $SK''_{SUE} = (K''_0, K''_1, K''_{2,1}, \ldots, K''_{2,n})$ be an SUE private key with a counter index h_c that is generated from the game $\mathbf{I}_{h_c,1}$. It generates a random exponent $\delta_{k,1}$ once for the node v_k that is related with this SUE private key and builds a new SUE private key as

$$SK_{SUE} = \Big(K_0 = K''_0 g_2^{\delta_{k,1}},\ K_1 = K''_1,\ K_{2,1} = K''_{2,1},\ \ldots, K_{2,n} = K''_{2,n}\Big).$$

Game $\mathbf{I}'_{h_c,2}$. This game $\mathbf{I}'_{h_c,2}$ is almost the same as $\mathbf{I}_{h_c,2}$ except SUE private keys with a counter index h_c. The modification is similar to the game $\mathbf{I}'_{h_c,1}$. We should note that $\mathbf{I}_{q_{h,sue},2} = \mathbf{I}'_{q_{h,sue},2}$ since the random element g_2^{δ} in each SUE private keys subsumes the additional fixed element $g_2^{\delta_{k,1}}$. In the game $\mathbf{I}'_{0,2}$, all SUE private keys with a node index h are semi-functional of type 2 where $\delta_{k,1}$ is fixed for the node v_k that is related with the node index h, but all ABE private keys with a node index h are still normal.

Game $\mathbf{I}''$. This game $\mathbf{I}''$ is the same as $\mathbf{G}_{1,h}$. Compared to the game $\mathbf{I}'_{0,2}$, all ABE private keys with a node index h are changed to be semi-functional of type 2 where $\delta_{k,0}$ is fixed for the node v_k.

Let $\mathbf{Adv}^{I}_{\mathcal{A}_{II}}$ be the advantage of $\mathcal{A}_{II}$ in a game $\mathbf{I}$. From the following Claims 4.14, 4.15, 4.16, 4.17, and 4.18, we can easily obtain the equation $\mathbf{Adv}^{I_{0,2}}_{\mathcal{A}_{II}} - \mathbf{Adv}^{I''}_{\mathcal{A}_{II}} \leq (4q_{h,sue}+1)\cdot \mathbf{Adv}^{A2}_{\mathcal{B}}(\lambda)$.

Let E_I, E_{II} be the event such that an adversary behave like the Type-I, Type-II adversary respectively. If an adversary can be both of types, then it just selects one type arbitrary. From the above two inequalities for two types of adversaries, we have the following inequality as

$$\begin{aligned}
\mathbf{Adv}^{G_{1,h-1}}_{\mathcal{A}} - \mathbf{Adv}^{G_{1,h}}_{\mathcal{A}} &\leq \Pr[E_I]\cdot(\mathbf{Adv}^{G_{1,h-1}}_{\mathcal{A}_I} - \mathbf{Adv}^{G_{1,h}}_{\mathcal{A}_I}) + \Pr[E_{II}]\cdot(\mathbf{Adv}^{G_{1,h-1}}_{\mathcal{A}_{II}} - \mathbf{Adv}^{G_{1,h}}_{\mathcal{A}_{II}}) \\
&\leq \Pr[E_I]\cdot(\mathbf{Adv}^{H_{0,2}}_{\mathcal{A}_I} - \mathbf{Adv}^{H''}_{\mathcal{A}_I}) + \Pr[E_{II}]\cdot(\mathbf{Adv}^{I_{0,2}}_{\mathcal{A}_{II}} - \mathbf{Adv}^{I''}_{\mathcal{A}_{II}}) \\
&\leq \Pr[E_I]\cdot(4q_{h,abe}+1)\cdot\mathbf{Adv}^{A2}_{\mathcal{B}}(\lambda) + (1-\Pr[E_I])\cdot(4q_{h,sue}+1)\cdot\mathbf{Adv}^{A2}_{\mathcal{B}}(\lambda) \\
&\leq (4(q_{h,abe}+q_{h,sue})+2)\cdot\mathbf{Adv}^{A2}_{\mathcal{B}}(\lambda).
\end{aligned}$$

This completes our proof. $\square$

Claim 4.9. *If the Assumption 2 holds, then no polynomial-time Type-I adversary can distinguish between $\boldsymbol{H}_{h_c-1,2}$ and $\boldsymbol{H}_{h_c,1}$ with a non-negligible advantage.*

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

Setup: $\mathcal{B}$ first select random exponents $a, \{t'_i\}, \gamma \in \mathbb{Z}_N$ and sets $MK_{ABE} = (\gamma, Y = g_{p_3})$ and the public parameters of ABE as

$$PP_{ABE} = \Big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_{p_1}, g^a, \{T_i = g^{t'_i}\}, \Lambda = e(g,g)^{\gamma}\Big).$$

It sets MK_{SUE} and PP_{SUE} as the same as Lemma 3.12. It also obtains $\mathcal{BT}$ by running **CS.Setup** and assigns random exponents $\gamma_i, \delta_{i,0}, \delta_{i,1} \in \mathbb{Z}_N$ to each node v_i in $\mathcal{BT}$. It selects a random exponent $\alpha \in \mathbb{Z}_N$ and sets $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$ and $PP = \big(PP_{ABE}, PP_{SUE}, g = g_{p_1}, \Omega = e(g,g)^{\alpha}\big)$.
Query 1: In the query step, $\mathcal{A}_I$ adaptively requests ABE private keys for private keys and SUE private keys for update keys. If this is an ABE private key (or an SUE private key) query with indexes (i_n, i_c), then $\mathcal{B}$ handles this query as follows:

- **Case** $i_n < h$: It first builds a normal ABE private key (or a normal SUE private key) since it knows the master secret key and converts it to a semi-functional one by using $\delta_{k,0}$ (or $\delta_{k,1}$) from $\mathcal{BT}$ and R_2Y_1 from the assumption.

- **Case** $i_n = h$: If this is an SUE private key query, then it creates a normal SUE private key since it knows the master secret key. If this is an ABE private key query, then it proceeds as follows:

  If $i_c < h_c$, then it first builds a normal ABE private key and converts it to semi-functional one of type 2 by selecting a new random exponent $\delta \in \mathbb{Z}_N$ as
  $$SK_{ABE} = \Big(K_0 = g^{\gamma_{j_k}} g^{ar} Y_0' \cdot (R_2Y_1)^{\delta},\ K_1 = g^r Y_1',\ \{K_{2,j} = T_j^r Y_{2,j}'\}_{j\in S}\Big).$$
  If $i_c = h_c$, then it chooses random elements $Y_0', Y_1', \{Y_{2,j}'\} \in \mathbb{G}_{p_3}$ and builds an ABE private key as
  $$SK_{ABE} = \Big(K_0 = g^{\gamma_{j_k}} (Z)^a Y_0',\ K_1 = ZY_1',\ \{K_{2,j} = (Z)^{t'_{\rho(j)}} Y_{2,j}'\}_{j\in S}\Big).$$
  If $i_c > h_c$, it builds a normal ABE private key since it knows the master secret key.

- **Case** $i_n > h$: It creates a normal ABE private key (or a normal SUE private key) since it knows the master secret key.

Challenge: In the challenge step, $\mathcal{A}$ submits a challenge access structure $\mathbb{A}^*$, a challenge time T^*, and challenge messages M_0^*, M_1^*. To make the semi-functional challenge ciphertext, $\mathcal{B}$ implicitly sets $g^s = X_1$ and $g_2^c = R_1$. It selects random exponents $v_2', \dots, v_n', s_1', \dots, s_l' \in \mathbb{Z}_N$ and sets a random vector $\vec{v}' = (a, v_2', \dots, v_n')$. Next it builds a semi-functional ciphertext header CH_{ABE} by implicitly setting $\vec{v} = sa^{-1}\vec{v}'$, $\vec{u} = c\vec{v}'$, and $s_j = ss_j', y_j = ss_j'$ as
$$CH_{ABE} = \Big(C_0 = X_1R_1,\ \{C_{1,j} = (X_1R_1)^{A_j\vec{v}'}(X_1R_1)^{t'_{\rho(j)}s'_j},\ C_{2,j} = (X_1R_1)^{-s'_j}\}\Big).$$
It builds a ciphertext header CH_{SUE} as the same as Lemma 3.12 by using the same element X_1R_1 of the assumption. Next, it flips a random coin $b \in \{0,1\}$ and creates the challenger ciphertext as $CT^*_{\mathbb{A}^*,T^*} = (CH_{ABE}, CH_{SUE}, C = e(X_1R_1, g)^{\alpha} \cdot M_b^*)$.
Query 2: Same as Query 1.
Guess: $\mathcal{A}$ outputs a guess b'. If $b = b'$, then $\mathcal{B}$ outputs 1. Otherwise, it outputs 0.

If $Z \in \mathbb{G}_{p_1p_3}$, the above simulation is properly distributed. If $Z \in \mathbb{G}$, the above simulation is almost properly distributed except that the $C_{1,j}$ components of the challenge ABE ciphertext header are correlated with the ABE private key with indexes (h, h_c) because $\vec{v}' = (a, \dots)$ is used in $C_{1,j}$ to solve the paradox of dual system encryption. Though these are correlated, we must argue that this is information theoretically hidden to the Type-I adversary. To argue this, we can use the restriction of the Type-I adversary that he can only request any ABE private key with a node index h for the set of attributes S such that $S \notin \mathbb{A}^*$, and the restrictions of attributes such that the attributes are only used once in the ciphertext header for ABE. This argument is the same as that of Lemma 8 in [26]. □

Claim 4.10. *If the Assumption 2 holds, then no polynomial-time Type-I adversary can distinguish between* $\boldsymbol{H}_{h_c,1}$ *and* $\boldsymbol{H}_{h_c,2}$ *with a non-negligible advantage.*

Proof. The proof of this claim is almost the same as that of Claim 4.9. The only difference is the generation of an ABE private key for $i_n = h$ and $i_c = h_c$. This ABE private key is generated as follows:

- If $i_n = h$ and $i_c = h_c$, it chooses a new random exponent $\delta \in \mathbb{Z}_N$ and random elements $Y'_0, Y'_1, \{Y'_{2,j}\} \in \mathbb{G}_{p_3}$, and then builds an ABE private key as

$$SK_{ABE} = \Big(K_0 = g^{\gamma_{j_k}}(Z)^{w'}Y'_0(R_2Y_1)^{\delta},\ K_1 = ZY'_1,\ \{K_{2,j} = (Z)^{t'_{\rho(j)}}Y'_{2,j}\}_{j\in S}\Big).$$

Note that the ABE private key with indexes (h, h_c) is no longer correlated with the challenge ABE ciphertext header since the K_0 component of the ABE private key is re-randomized by $(R_2Y_1)^{\delta}$. □

Claim 4.11. *If the Assumption 2 holds, then no polynomial-time Type-I adversary can distinguish between* $\boldsymbol{H}'_{h_c,1}$ *and* $\boldsymbol{H}'_{h_c,2}$ *with a non-negligible advantage.*

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

The proofs of Claim 4.11 and Claim 4.12 are almost the same as that of Claim 4.9 and Claim 4.10 respectively. The only difference is that each element K_0 of ABE private keys with a node index h that is generated in Claim 4.9 and Claim 4.10 respectively is additionally multiplied by $(R_2Y_1)^{\delta'_{k,0}}$ where $\delta'_{k,0}$ is a fixed exponent that is related with the node v_k of the ABE private key. This modification is possible since R_2Y_1 is given in the assumption. In this case, ABE private keys with the node index h additionally contain $R_2^{\delta'_{k,0}}$. We omit the detailed proofs of these claims.

Claim 4.13. *If the Assumption 2 holds, then no polynomial-time Type-I adversary can distinguish between* $\boldsymbol{H}'_{0,2}$ *and* $\boldsymbol{H}''$ *with a non-negligible advantage.*

Proof. The proof of this claim is almost the same as that of Claim 4.9 except that the generation of ABE private keys and SUE private keys for $i_n = h$. In the query step, ABE private keys and SUE private keys for $i_n = h$ are generated as follows:

- **Case** $i_n = h$: If this is an ABE private key query, then it selects random $r \in \mathbb{Z}_N, Y'_0, Y'_1, \{Y'_{2,j}\} \in \mathbb{G}_{p_3}$ and builds an ABE private key by selecting $\delta'_{k,0} \in \mathbb{Z}_N$ once for this node index h as

$$SK_{ABE} = \Big(K_0 = Zg^{ar}Y'_0 \cdot (R_2Y_1)^{\delta'_{k,0}},\ K_1 = g^rY'_1,\ \{K_{2,j} = T_j^rY'_{2,j}\}_{j\in S}\Big).$$

  If this is an SUE private key query, then it selects random $r \in \mathbb{Z}_N, Y'_0, Y'_1, Y'_{2,1}, \ldots, Y'_{2,n} \in \mathbb{G}_{p_3}$ and builds an SUE private key as

$$SK_{SUE} = \Big(K_0 = g^{\alpha}Z^{-1}w^{-r}Y'_0,\ K_1 = g^rY'_1,\ \{K_{2,j} = (u_{j,L[j]}^{L|_j}h_{j,L[j]})^rY'_{2,j}\}_{1\le j\le n}\Big).$$

Let g^{γ_h} be the $\mathbb{G}_{p_1}$ part of Z and $g_2^{\delta_h}$ be the $\mathbb{G}_{p_2}$ part of Z. If $Z \in \mathbb{G}_{p_1p_3}$, the simulation is the same as $\mathbf{H}'_{0,2}$ since the SUE private key has $K_0 = g^{\alpha-\gamma_h}w^rY_0$. If $Z \in \mathbb{G}$, the simulation is the same as $\mathbf{H}''$ since the ABE private key has $K_0 = g^{\gamma_h}g^{ar}Y_0 \cdot g_2^{\delta_h}R_2^{\delta'_{k,0}}$ and the SUE private key has $K_0 = g^{\alpha-\gamma_h}w^rY_0g_2^{-\delta_h}$. Note that the $\mathbb{G}_{p_2}$ part of the ABE private key and that of the SUE private key are reused across all queries with the same node index h. This completes our proof. □

Claim 4.14. *If the Assumption 2 holds, then no polynomial-time Type-II adversary can distinguish between $\boldsymbol{I}_{h_c-1,2}$ and $\boldsymbol{I}_{h_c,1}$ with a non-negligible advantage.*

Proof. The public parameters, ABE private keys and SUE private keys with a node index i_n such that $i_n \neq h$, and the challenge ciphertext are generated as the same as Claim 4.9. The ABE private keys and SUE private keys for the node index $i_n = h$ are generated as follows: If this is an ABE private key (or an SUE private key) query with indexes (i_n, i_c), then $\mathcal{B}$ handles this query as follows:

- **Case** $i_n = h$: If this is an ABE private key query, then it creates a normal ABE private key since it knows the master secret key. If this is an SUE private key query, then it proceeds as follows:

  If $i_c < h_c$, then it first builds a normal SUE private key and converts it to a semi-functional one of type 2 by selecting a new random exponent $\delta \in \mathbb{Z}_N$ as
  $$SK_{SUE} = \Big(K_0 = g^{\alpha - \gamma_{j_k}} w^{-r} Y_0' \cdot (R_2 Y_1)^{\delta},\ K_1 = g^r Y_1',\ \{K_{2,j} = (u_{j,L[j]}^{L|_j} h_{j,L[j]})^r Y_{2,j}'\}_{1 \le j \le n}\Big).$$

  If $i_c = h_c$, then it chooses random elements $Y_0', Y_1', Y_{2,1}', \ldots, Y_{2,n}' \in \mathbb{G}_{p_3}$ and builds an SUE private key as
  $$SK_{SUE} = \Big(K_0 = g^{\alpha - \gamma_{j_k}} (Z)^{-w'} Y_0',\ K_1 = Z Y_1',\ \{K_{2,j} = (Z)^{f_{j,L[j]}(L|_j)} Y_{2,j}'\}_{1 \le j \le n}\Big).$$

  If $i_c > h_c$, it builds a normal SUE private key since it knows the master secret key.

If $Z \in \mathbb{G}_{p_1 p_3}$, the above simulation is properly distributed. If $Z \in \mathbb{G}$, the above simulation is almost properly distributed except that the C_1 component of the challenge SUE ciphertext header is correlated with the SUE private key with indexes (h, h_c) because w' is used in C_1 to solve the paradox of dual system encryption. Though these are correlated, we must argue that this is information theoretically hidden to the Type-II adversary. To argue this, we can use the restriction of the Type-II adversary that he can only request any SUE private key with a node index h for the update time T such that $T < T^*$. This argument is the same as that of Lemma 3.12. □

Claim 4.15. *If the Assumption 2 holds, then no polynomial-time Type-II adversary can distinguish between $\boldsymbol{I}_{h_c,1}$ and $\boldsymbol{I}_{h_c,2}$ with a non-negligible advantage.*

Proof. The proof of this claim is almost the same as that of Claim 4.14. The only difference is the generation of an SUE private key for $i_n = h$ and $i_c = h_c$. This SUE private key is generated as follows:

- If $i_n = h$ and $i_c = h_c$, it chooses random elements $\delta \in \mathbb{Z}_N, Y_0', Y_1', Y_{2,1}', \ldots, Y_{2,n}' \in \mathbb{G}_{p_3}$, and then builds an SUE private key as
  $$SK_{SUE} = \Big(K_0 = g^{\alpha - \gamma_{j_k}} (Z)^{-w'} Y_0' (R_2 Y_1)^{\delta},\ K_1 = Z Y_1',\ \{K_{2,j} = (Z)^{f_{j,L[j]}(L|_j)} Y_{2,j}'\}_{1 \le j \le n}\Big).$$

Note that the SUE private key with indexes (h, h_c) is no longer correlated with the challenge SUE ciphertext header since the K_0 component of the SUE private key is re-randomized by $(R_2 Y_1)^{\delta}$. □

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

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

The proofs of Claim 4.16 and Claim 4.17 are almost the same as that of Claim 4.14 and Claim 4.15 respectively except that each element K_0 of SUE private keys with a node index h that are generated in Claim 4.14 and Claim 4.15 respectively is additionally multiplied by $(R_2Y_1)^{\delta'_{k,1}}$ where $\delta'_{k,1}$ is a fixed exponent that is related with the node v_k of the SUE private key. We omit the detailed proofs of these claims.

Claim 4.18. *If the Assumption 2 holds, then no polynomial-time Type-II adversary can distinguish between $\boldsymbol{I}'_{0,2}$ and $\boldsymbol{I}''$ with a non-negligible advantage.*

Proof. The proof of this claim is almost the same as that of Claim 4.14 except that the generation of ABE private keys and SUE private keys for $i_n = h$. In the query step, ABE private keys and SUE private keys for $i_n = h$ are generated as follows:

- **Case** $i_n = h$: If this is an SUE private key query, then it selects random $r \in \mathbb{Z}_N, Y'_0, Y'_1, \{Y'_{2,j}\} \in \mathbb{G}_{p_3}$ and builds an SUE private key by selecting $\delta'_{k,1} \in \mathbb{Z}_N$ once for this node index h as

$$SK_{SUE} = \Big(K_0 = g^{\alpha} Z^{-1} w^{-r} Y'_0 \cdot (R_2Y_1)^{\delta'_{k,1}},\ K_1 = g^r Y'_1,\ \{K_{2,j} = (u_{j,L[j]}^{L|_j} h_{j,L[j]})^r Y'_{2,j}\}_{1 \le j \le n}\Big).$$

  If this is an ABE private key query, then it selects random $r \in \mathbb{Z}_N, Y'_0, Y'_1, Y'_{2,1}, \ldots, Y'_{2,n} \in \mathbb{G}_{p_3}$ and builds an ABE private key as

$$SK_{ABE} = \Big(K_0 = Z g^{ar} Y'_0,\ K_1 = g^r Y'_1,\ \{K_{2,j} = T_j^r Y'_{2,j}\}_{j \in S}\Big).$$

Let g^{γ_h} be the $\mathbb{G}_{p_1}$ part of Z and $g_2^{\delta_h}$ be the $\mathbb{G}_{p_2}$ part of Z. If $Z \in \mathbb{G}_{p_1p_3}$, the simulation is the same as $\mathbf{I}'_{0,2}$ since the ABE private key has $K_0 = g^{\gamma_h} g^{ar} Y_0$. If $Z \in \mathbb{G}$, the simulation is the same as $\mathbf{I}''$ since the SUE private key has $K_0 = g^{\alpha-\gamma_h} w^{-r} Y_0 \cdot g_2^{-\delta_h} R_2^{\delta'_{k,1}}$ and the ABE private key has $K_0 = g^{\gamma_h} g^{ar} Y_0 g_2^{\delta_h}$. Note that the $\mathbb{G}_{p_2}$ part of the ABE private key and that of the SUE private key are reused across all queries with the same node index h. $\square$

Lemma 4.19. *If the Assumption 3 holds, then no polynomial-time adversary can distinguish between $\boldsymbol{G}_2$ and $\boldsymbol{G}_3$ with a non-negligible advantage.*

Proof. Suppose there exists an adversary $\mathcal{A}$ that distinguish $\mathbf{G}_2$ from $\mathbf{G}_3$ with a non-negligible advantage. A simulator $\mathcal{B}$ that solves the Assumption 3 using $\mathcal{A}$ is given: a challenge tuple $D = ((N, \mathbb{G}, \mathbb{G}_T, e), g_{p_1}, g_{p_2}, g_{p_3}, g_{p_1}^a R_1, g_{p_1}^b R_2)$ and Z where $Z = Z_0 = e(g_{p_1}, g_{p_1})^{ab}$ or $Z = Z_1 = e(g_{p_1}, g_{p_1})^c$. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Setup: $\mathcal{B}$ chooses random exponents $a, \{t'_i\}, \gamma \in \mathbb{Z}_N$ and sets $MK_{ABE} = (\gamma, Y = g_{p_3})$ and the public parameters of ABE as

$$PP_{ABE} = \big((N, \mathbb{G}, \mathbb{G}_T, e), g = g_{p_1}, g^a, \{T_i = g^{t'_i}\}, \Lambda = e(g,g)^{\gamma}\big).$$

It sets MK_{SUE} and PP_{SUE} as the same as Lemma 3.14. It also obtains $\mathcal{BT}$ by running **CS.Setup** and assigns random exponents $\gamma_i, \delta_{i,0}, \delta_{i,1} \in \mathbb{Z}_N$ to each node v_i in $\mathcal{BT}$. It implicitly sets $\alpha = a$ from the term $g_{p_1}^a R_1$ and creates $PP = (PP_{ABE}, PP_{SUE}, g = g_1, \Omega = e(g, g_{p_1}^a R_1))$.

Query 1: In the query step, $\mathcal{A}$ adaptively requests private keys and update keys. If this is a private key query, then $\mathcal{B}$ creates a semi-functional private key since it knows $\{g^{\gamma_i}, g_2^{\delta_{i,0}}\}$ from $\mathcal{BT}$ and g_{p_2} from the assumption. If this is an update key query, then $\mathcal{B}$ creates a semi-functional update key since it knows $g_{p_1}^a R_1, g_{p_2}$ from the assumption and $\{g^{\gamma_i}, g_2^{\delta_{i,1}}\}$ from $\mathcal{BT}$. Note that it cannot create a normal update key since it does not know $g_{p_1}^a$.

Challenge: In the challenge step, $\mathcal{B}$ first selects random exponents $y'_2, \ldots, y'_n$ and sets a random vector $\vec{v}' = (a, y'_2, \ldots, y'_n)$ and creates a ciphertext header for ABE by selecting random exponents $s_1, \ldots, s_l \in \mathbb{Z}_N$ as

$$CH_{ABE} = \Big(C_0 = g_{p_1}^b R_2,\ \{C_{1,j} = (g_{p_1}^b R_2)^{A_j \cdot \vec{v}'} (g_{p_1}^b R_2)^{t'_{\rho(j)} s_j},\ C_{2,j} = (g_{p_1}^b R_2)^{-s_j}\}_{1 \le j \le l} \Big).$$

It creates a ciphertext header CH_{SUE} as the same as Lemma 3.14. Next, it flips a random coin $b \in \{0,1\}$ and sets the challenger ciphertext as $CT^*_{\mathbb{A}^*, T^*} = (CH_{ABE}, CH_{SUE}, C = Z \cdot M_b^*)$.

Query 2: Same as Query 1.

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

If $Z = e(g_{p_1}, g_{p_1})^{ab}$, the challenge ciphertext is properly distributed as a semi-functional one. Otherwise, C is an encryption of a random message. This completes our proof. □

4.6 Discussions

Efficiency. In our RS-ABE scheme, the number of group elements in a private key, an update key, and a ciphertext is $\log N_{max} \cdot |S|$, $r \log(N_{max}/r) \cdot \log T_{max}$, and $2l + 3 \log T_{max}$ respectively where S is the set of attributes, r is the size of a revoked set R, and l is the row size of an access structure. The decryption algorithm of our RS-ABE scheme requires at most $2l + \log T_{max}$ pairing operations. In the RS-ABE scheme of Sahai et al. [40], the number of group elements in a ciphertext is $2 \log T_{max} \cdot (l + 2 \log T_{max})$ since a piecewise CP-ABE scheme was used. For instance, if we set $T_{max} = 2^{20}$ and $l = 20$, then the ciphertext of our scheme consists of just 100 group elements whereas the ciphertext of their scheme consists of 2400 group elements.

Removing Stored Exponents. In the setup algorithm of our RS-ABE scheme, each node v_i in $\mathcal{BT}$ keeps a random exponent γ_i for later use. If we use the idea of Libert and Vergnaud [31] that uses a pseudo-random function (PRF), then we can remove this requirement. That is, the authority keeps a seed z for PRF and deterministically generates a random exponent γ_i for a node $v_i \in \mathcal{BT}$ when it is needed as $\gamma_i = PRF_z(L_i)$ where L_i is a label string that is associated with v_i. The security of this modified scheme easily follows from the security of PRF.

Supporting an Exponential Number of Users. The setup algorithm of our RS-ABE scheme takes the maximum (polynomial) number of users N_{max}. One problem of our RS-ABE scheme is that it cannot expand the capacity of users if the number of users exceeds N_{max}. We may try to use the idea of Boldyreva et al. [3] that combines an old binary tree and an empty binary tree into a new bigger binary tree. However, it is not easy to broadcast the new private key components of users for the root node of the new binary tree in RS-ABE since an attribute in a private key is shared among different users. If we consider the modified RS-ABE scheme that uses a PRF to remove the stored exponents, then it can support an exponential number of users by simply setting $N_{max} = 2^\lambda$. In this case, the setup algorithm does not need to keep the binary tree $\mathcal{BT}$ in MK.

Deriving Decryption Keys. The decryption algorithm of our RS-ABE scheme uses a (long-term) private key SK and an update key UK instead of using a (short-term) decryption key DK that can be derived from SK and UK to directly decrypt a ciphertext. The use of decryption keys in R-IBE was introduced by Boldyreva et al. [3], and this can be useful to protect SK by storing it in a temper-proof hardware and DK in a weakly secure memory area. However, we should consider *decryption key exposure* attacks in this scenario as pointed by Seo and Emura [43]. We can modify our RS-ABE scheme to support the derivation of DK by tying the element K_0 of $SK_{ABE,j} \in SK$ and the element K_0 of $SK_{SUE,i} \in UK$ that match the same node in $\mathcal{BT}$. This modified scheme can be secure against decryption key exposure attacks since SK cannot be extracted from DK by key re-randomization[5].

Key-Policy ABE. For our RS-ABE scheme, we use the CP-ABE scheme of Lewko et al. [26] as a primary encryption scheme where CP-ABE supports an access structure f in a ciphertext and a set of attributes S in a private key. We expect that an RS-ABE scheme that has an access structure f in a private key and a set of attributes S in a ciphertext can be built by using the KP-ABE scheme of Lewko et al. [26] as the primary encryption scheme. Sahai et al. [40] also proposed an RS-ABE scheme that supports a key-policy by employing their piecewise KP-ABE scheme.

5 Revocable-Storage Predicate Encryption

In this section, we introduce the concept of RS-PE and propose an RS-PE scheme by combining the predicate encryption scheme of Park [36] and our SUE scheme.

5.1 Definitions

Revocable-storage predicate encryption (RS-PE) is predicate encryption (PE) that supports additionally the revocation functionality and the ciphertext update functionality. That is, RS-PE can prevent the access of a user whose private key is revealed to ciphertexts that will be created in the future time by revoking the user, and it also can prevent the access of the user to ciphertexts that were created in the past time by updating the ciphertexts. Additionally, RS-PE can hide the attribute information in ciphertexts compared to RS-ABE. The following is the syntax of RS-PE.

Definition 5.1 (Revocable-Storage Predicate Encryption)**.** *A revocable-storage predicate encryption (RS-PE) scheme for the class of predicates $\mathcal{F}$ over the set of attributes Σ consists of seven PPT algorithms **Setup**, **GenKey**, **UpdateKey**, **Encrypt**, **UpdateCT**, **RandCT**, and **Decrypt**, which are defined as follows:*

***Setup**$(1^\lambda, T_{max}, N_{max})$. The setup algorithm takes as input a security parameter 1^λ, the maximum time T_{max}, and the maximum number of users N_{max}, and it outputs public parameters PP and a master secret key MK.*

***GenKey**(f, u, MK, PP). The key generation algorithm takes as input a predicate $f \in \mathcal{F}$, a user index $u \in \mathcal{N}$, the master secret key MK, and the public parameters PP, and it outputs a private key $SK_{f,u}$.*

***UpdateKey**(T, R, MK, PP). The key update algorithm takes as input a time $T \leq T_{max}$, a set of revoked users $R \subseteq \mathcal{N}$, the master secret key MK, and the public parameters PP, and it outputs an update key $UK_{T,R}$.*

***Encrypt**(I, T, M, PP). The encryption algorithm takes as input an attribute $I \in \Sigma$, a time $T \leq T_{max}$, a message $M \in \mathcal{M}$, and the public parameters PP, and it outputs a ciphertext CT_T.*

***UpdateCT**$(CT_T, T+1, PP)$. The ciphertext update algorithm takes as input a ciphertext CT_T for an attribute I and a time T, a new time $T+1$, and the public parameters PP, and it outputs an updated ciphertext $CT_{T'}$ for the attribute I and the new time $T+1$.*

[5]To support perfect re-randomization, we can move the element $Y \in \mathbb{G}_{p_3}$ in MK of ABE (or SUE) into PP since the static assumptions always contain $g_{p_3} \in \mathbb{G}_{p_3}$.

RandCT(CT_T, PP). *The ciphertext randomization algorithm takes as input a ciphertext CT_T for an attribute I and a time T, and the public parameters PP, and it outputs a re-randomized ciphertext CT'_T.*

Decrypt$(CT_T, SK_{f,u}, UK_{T',R}, PP)$. *The decryption algorithm takes as input a ciphertext CT_T, a private key $SK_{f,u}$, an update key $UK_{T',R}$, and the public parameters PP, and it outputs a message M or the distinguished symbol $\perp$.*

*The correctness property of RS-PE is defined as follows: For all PP, MK generated by **Setup**, all f and u, any $SK_{f,u}$ generated by **GenKey**, all I, T, and M, any CT_T generated by **Encrypt** or **UpdateCT**, all T' and R, any $UK_{T',R}$ generated by **UpdateKey**, it is required that:*

- *If $(f(I) = 1) \wedge (u \notin R) \wedge (T \leq T')$, then **Decrypt**$(CT_T, SK_{f,u}, UK_{T',R}, PP) = M$.*
- *If $(f(I) = 0) \vee (u \in R) \vee (T' < T)$, then **Decrypt**$(CT_T, SK_{f,u}, UK_{T',R}, PP) = \perp$ with all but negligible probability.*

The security property of RS-PE can be defined by combining the security property of RS-ABE in Section 4 and that of PE [22]. However, we should be careful to define the security property of RS-PE since the security property of PE considers not only the security of messages, but also the security of attributes. In this paper, we define a selective (revocation list) security for RS-PE such that an adversary should submits challenge attributes I_0^*, I_1^*, a challenge time T^*, and the set of revoked users R^* at the time T^* before he receives the public parameters. The following is the formal definition of the security.

Definition 5.2 (Selective Revocation List Security)**.** *The security property for RS-PE is defined in terms of the indistinguishability under a chosen plaintext attack (IND-CPA). The selective security game for this property is defined as the following game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:*

1. ***Init**: $\mathcal{A}$ first submits two challenge attributes $I_0^*, I_1^* \in \Sigma$, a challenge time $T^* \leq T_{max}$, and the set of revoked users $R^* \subseteq \mathcal{N}$ at the time T^*.*
2. ***Setup**: $\mathcal{C}$ generates the public parameters PP and the master secret key MK by running **Setup**(1^λ), and it gives PP to $\mathcal{A}$.*
3. ***Query 1**: $\mathcal{A}$ may adaptively request a polynomial number of private keys and update keys. $\mathcal{C}$ proceeds as follows:*
   - *If this is a private key query for a predicate $f \in \mathcal{F}$ and a user index $u \in \mathcal{N}$, then it gives the corresponding private key $SK_{f,u}$ to $\mathcal{A}$ by running **GenKey**(f, u, MK, PP). Note that the adversary is allowed to query only one private key for each user u.*
   - *If this is an update key query for an update time $T \leq T_{max}$ and a set of revoked users $R \subseteq \mathcal{N}$, then it gives the corresponding update key $UK_{T,R}$ to $\mathcal{A}$ by running **UpdateKey**(T, R, MK, PP). Note that the adversary is allowed to query only one update key for each time T.*

   We also require that $R_i \subseteq R_j$ if $T_i < T_j$ where R_i and R_j are the sets of revoked users at the time T_i and T_j respectively.
4. ***Challenge**: $\mathcal{A}$ submits two challenge messages $M_0^*, M_1^* \in \mathcal{M}$ of equal length subject to the following restrictions:*
   - *If $M_0^* \neq M_1^*$, it is required that $(f_i(I_0^*) = 0) \vee (u_i \in R_j) \vee (T_j < T^*)$ and $(f_i(I_1^*) = 0) \vee (u_i \in R_j) \vee (T_j < T^*)$ for all $\{(f_i, u_i)\}$ of private key queries and all $\{(T_j, R_j)\}$ of update key queries.*

- *If $M_0^* = M_1^*$, it is required that $f_i(I_0^*) = f_i(I_1^*)$ for all $\{(f_i, u_i)\}$ of private key queries.*

*$\mathcal{C}$ chooses a random bit b and gives the ciphertext CT^* to $\mathcal{A}$ by running **Encrypt**(I_b^*, T^*, M_b^*, PP).*

5. ***Query 2**: $\mathcal{A}$ may continue to request private keys and update keys subject to the same restrictions as before, and $\mathcal{C}$ gives the corresponding private keys and update keys to $\mathcal{A}$.*
6. ***Guess**: Finally $\mathcal{A}$ outputs a bit b'.*

The advantage of $\mathcal{A}$ is defined as $\textbf{Adv}_{\mathcal{A}}^{RS\text{-}PE}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|$ where the probability is taken over all the randomness of the game. A RS-PE scheme is selectively secure under a chosen plaintext attack if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.

Remark 5.3. *In the above security model, the adversary additionally submits the set of revoked users R^* at the time T^*. Thus this model is weaker than the usual selective security model that the adversary only submits the challenge attributes and a challenge time. However, Boldyreva et al. [3] already used this weaker selective model to prove their revocable ABE scheme.*

Remark 5.4. *If the adversary submits M_0^*, M_1^* such that $M_0^* \neq M_1^*$ in the challenge step, then the restrictions in the challenge step implies that all private keys for a predicate f and a user index u that satisfy $f(I_b^*) = 1$ should be revoked for all update keys with $T \geq T^*$. That is, $u \in R^*$ for all private keys with f and u such that $f(I_b^*) = 1$. Note that we easily have that $R^* \subseteq R_j$ for all R_j with $T_j \geq T^*$ since it is required that $R_i \subseteq R_j$ if $T_i < T_j$ in the security model.*

5.2 Construction

We construct an RS-PE scheme in prime order groups by combining a PE scheme and the SUE scheme in Appendix A. The basic idea of constructing an RS-PE scheme by combining the PE and SUE schemes are the same as that of the RS-ABE scheme in Section 4.

Before presenting our RS-PE scheme, we first describe the PE scheme of Park [36] in prime order groups. This PE scheme supports supports the class of predicates $\mathcal{F} = \{f_{\vec{y}} \mid \vec{y} = (y_1, \ldots, y_n) \in \mathbb{Z}_p^n\}$ such that

$$f_{\vec{y}}(\vec{x}) = \begin{cases} 1 & \text{if } \langle \vec{x}, \vec{y} \rangle = 0 \mod p, \\ 0 & \text{otherwise} \end{cases}$$

and secure under standard assumptions. The PE scheme in prime order groups is described as follows:

PE.Setup(GDS, n)**:** This algorithm takes as input a group description string $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$ and a parameter n. It chooses random exponents $\omega, \{w_{1,i}\}_{i=1}^n, \{f_{1,i}, f_{2,i}\}_{i=1}^n, \{t_{1,i}\}_{i=1}^n, \{h_{1,i}, h_{2,i}\}_{i=1}^n, u_1, u_2, v_1, v_2, \gamma \in \mathbb{Z}_p$. Next, it computes $\{w_{2,i}\}_{i=1}^n, \{t_{2,i}\}_{i=1}^n$ to satisfy the constraints $\omega = u_1 w_{2,i} - u_2 w_{1,i}$ and $\omega = v_1 t_{2,i} - v_2 t_{1,i}$. It outputs the master secret key $MK = (\gamma, \{f_{1,i}, f_{2,i}, h_{1,i}, h_{2,i}\}_{i=1}^n)$ and the public parameters as

$$PP = \Big((p, \mathbb{G}, \mathbb{G}_T, e), g,\ g^{\omega}, \{W_{1,i} = g^{w_{1,i}}, W_{2,i} = g^{w_{2,i}}, F_{1,i} = g^{f_{1,i}}, F_{2,i} = g^{f_{2,i}}\}_{i=1}^n,$$
$$\{T_{1,i} = g^{t_{1,i}}, T_{2,i} = g^{t_{2,i}}, H_{1,i} = g^{h_{1,i}}, H_{2,i} = g^{h_{2,i}}\}_{i=1}^n, \{U_i = g^{u_i}, V_i = g^{v_i}\}_{i=1}^2, \Lambda = e(g,g)^{\gamma}\Big).$$

PE.GenKey($\vec{y}, MK, PP$): This algorithm takes as input a vector $\vec{y} = (y_1, \ldots, y_n)$, the master secret key MK, and the public parameters PP. It first selects random exponents $\lambda_1, \lambda_2, \{r_i\}, \{\phi_i\} \in \mathbb{Z}_p$. It outputs a private key that implicitly includes $\vec{y}$ as

$$SK_{\vec{y}} = \Big(K_A = g^{\gamma} \prod_{i=1}^{n} K_{1,i}^{-f_{1,i}} K_{2,i}^{-f_{2,i}} K_{3,i}^{-h_{1,i}} K_{4,i}^{-h_{2,i}},\ K_B = \prod_{i=1}^{n} g^{-(r_i+\phi_i)},$$
$$\big\{ K_{1,i} = U_2^{-r_i} W_{2,i}^{\lambda_1 y_i},\ K_{2,i} = U_1^{r_i} W_{1,i}^{-\lambda_1 y_i} \big\}_{i=1}^{n},\ \big\{ K_{3,i} = V_2^{-\phi_i} T_{2,i}^{\lambda_2 y_i},\ K_{4,i} = V_1^{\phi_i} T_{1,i}^{-\lambda_2 y_i} \big\}_{i=1}^{n} \Big).$$

PE.Encrypt($\vec{x}, s, PP$): This algorithm takes as input a vector $\vec{x} = (x_1, \ldots, x_n)$, a random exponent $s \in \mathbb{Z}_p$, and the public parameters PP. It selects random exponents $s_1, s_3, s_4 \in \mathbb{Z}_p$ and outputs a ciphertext header as

$$CH = \Big(C_A = g^s, C_B = g^{\omega s_1},\ \big\{ C_{1,j} = W_{1,i}^{s_1} F_{1,i}^{s} U_1^{x_i s_3},\ C_{2,i} = W_{2,i}^{s_1} F_{2,i}^{s} U_2^{x_i s_3} \big\}_{i=1}^{n},$$
$$\big\{ C_{3,i} = T_{1,i}^{s_1} H_{1,i}^{s} V_1^{x_i s_4},\ C_{4,i} = T_{2,i}^{s_1} H_{2,i}^{s} V_2^{x_i s_4} \big\}_{i=1}^{n} \Big)$$

and a session key $EK = \Lambda^s$.

PE.RandCT(CH, s', PP): This algorithm takes as input a ciphertext CH, a random exponent $s' \in \mathbb{Z}_p$, and the public parameters PP. It selects a random exponent $s'_1 \in \mathbb{Z}_p$ and outputs a partially re-randomized ciphertext header as

$$CH' = \Big(C'_A = C_A \cdot g^{s'}, C'_B = C_B \cdot g^{\omega s'_1},\ \big\{ C'_{1,i} = C_{1,i} \cdot W_{1,i}^{s'_1} F_{1,i}^{s'},\ C'_{2,i} = C_{2,i} \cdot W_{2,i}^{s'_1} F_{2,i}^{s'} \big\}_{i=1}^{n},$$
$$\big\{ C'_{3,i} = C_{3,i} \cdot T_{1,i}^{s'_1} H_{1,i}^{s'},\ C'_{4,i} = C_{4,i} \cdot T_{2,i}^{s'_1} H_{2,i}^{s'} \big\}_{i=1}^{n} \Big)$$

and a partial session key $EK' = \Lambda^{s'}$ that will be multiplied with the session key EK of CH.

PE.Decrypt($CH, SK_{\vec{y}}, PP$): This algorithm takes as input a ciphertext header CH, a private key $SK_{\vec{y}}$ for a vector $\vec{y}$, and the public parameters PP. It outputs a session key as

$$EK = e(C_A, K_A) \cdot e(C_B, K_B) \cdot \prod_{i=1}^{n} \prod_{j=1}^{4} e(C_{j,i}, K_{j,i}).$$

Note that it never returns $\perp$ even when $\langle \vec{x}, \vec{y} \rangle \neq 0$.

Let $\mathcal{M}$ be a subset of $\mathbb{G}_T$ such that $|\mathcal{M}| \leq p^{1/2}$. Our RS-PE scheme that uses the above PE scheme as a building block is described as follows:

RS-PE.Setup($1^\lambda, n, T_{max}, N_{max}$): This algorithm takes as input a security parameter 1^λ, the size of vectors n, the maximum time T_{max}, and the maximum number of users N_{max}.

1. It first generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p. Let g be the generator of $\mathbb{G}$. It sets $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$.
2. It obtains MK_{PE}, PP_{PE} and MK_{SUE}, PP_{SUE} by running **PE.Setup**(GDS, n) and **SUE.Setup**(GDS, T_{max}) respectively. It also obtains $\mathcal{BT}$ by running **CS.Setup**(N_{max}) and assigns a random exponent $\gamma_i \in \mathbb{Z}_p$ to each node v_i in $\mathcal{BT}$.

3. It selects a random exponent $\alpha \in \mathbb{Z}_p$, and the it outputs the master secret key $MK = (MK_{PE}, MK_{SUE}, \alpha, \mathcal{BT})$ and the public parameters as $PP = \big(PP_{PE}, PP_{SUE}, g, \Omega = e(g,g)^{\alpha}\big)$.

RS-PE.GenKey($\vec{y}, u, MK, PP$)**:** This algorithm takes as input a vector $\vec{y}$, a user index u, the master secret key $MK = (MK_{PE}, MK_{SUE}, \alpha, \mathcal{BT})$, and the public parameters PP.

1. It first obtains a private set $PV_u = \{S_{j_0}, \dots, S_{j_d}\}$ by running **CS.Assign**$(\mathcal{BT}, u)$ and retrieves random exponents $\{\gamma_{j_0}, \dots, \gamma_{j_d}\}$ from $\mathcal{BT}$ where γ_{j_k} is assigned to the node v_{j_k}.
2. For $0 \le k \le d$, it sets $MK'_{PE} = (\gamma_{j_k}, \{\dots\})$ by just replacing the first component γ to γ_{j_k} and obtains $SK_{PE,k}$ by running **PE.GenKey**$(\vec{y}, MK'_{PE}, PP_{PE})$.
3. It outputs a private key as $SK_{\vec{y},u} = \big(PV_u, SK_{PE,0}, \dots, SK_{PE,d}\big)$.

RS-PE.UpdateKey(T, R, MK, PP)**:** This algorithm takes as input an update time T, a set of revoked users R, the master secret key MK, and the public parameters PP.

1. It first obtains a covering set $CV_R = \{S_{i_1}, \dots, S_{i_m}\}$ by running **CS.Cover**$(\mathcal{BT}, R)$.
2. For $1 \le k \le m$, it sets $MK'_{SUE} = (\alpha - \gamma_{i_k})$ and obtains $SK_{SUE,k}$ by running **SUE.GenKey**(T, MK'_{SUE}, PP_{SUE}).
3. It outputs an update key that implicitly includes T and R as $UK_{T,R} = \big(CV_R, SK_{SUE,1}, \dots, SK_{SUE,m}\big)$.

RS-PE.Encrypt($\vec{x}, T, M, PP$)**:** This algorithm takes as input a vector $\vec{x}$, a time T, a message $M \in \mathcal{M}$, and the public parameters PP.

1. It selects a random exponent $s \in \mathbb{Z}_p$ and obtains CH_{PE} and CH_{SUE} by running **PE.Encrypt**$(\vec{x}, s, PP_{PE})$ and **SUE.Encrypt**(T, s, PP_{SUE}), respectively. Note that it ignores two partial session keys that are returned by **PE.Encrypt** and **SUE.Encrypt**.
2. It outputs a ciphertext that implicitly includes T as $CT_T = \big(CH_{PE}, CH_{SUE}, C = \Omega^s \cdot M\big)$.

RS-PE.UpdateCT($CT_T, T+1, PP$)**:** This algorithm takes as input a ciphertext $CT_T = (CH_{PE}, CH_{SUE}, C)$ for a vector $\vec{x}$ and a time T, a new time $T+1$, and the public parameters PP.

1. It first obtains CH'_{SUE} by running **SUE.UpdateCT**$(CH_{SUE}, T+1, PP_{SUE})$.
2. It outputs an updated ciphertext that implicitly includes $T+1$ as $CT_{T+1} = \big(CH_{PE}, CH'_{SUE}, C\big)$.

RS-PE.RandCT(CT_T, PP)**:** This algorithm takes as input a ciphertext $CT_T = (CH_{PE}, CH_{SUE}, C)$ and the public parameters PP.

1. It selects a random exponent $s' \in \mathbb{Z}_p$ and obtains CH'_{PE} and CH'_{SUE} by running **PE.RandCT**(CH_{PE}, s', PP_{PE}) and **SUE.RandCT**(CH_{SUE}, s', PP_{SUE}), respectively.
2. It outputs a re-randomized ciphertext as $CT'_T = \big(CH'_{PE}, CH'_{SUE}, C' = C \cdot \Omega^{s'}\big)$.

RS-PE.Decrypt($CT_T, SK_{\vec{y},u}, UK_{T',R}, PP$)**:** This algorithm takes as input a ciphertext $CT_T = (CH_{PE}, CH_{SUE}, C)$ for a vector $\vec{x}$ and a time T, a private key $SK_{\vec{y},u} = (PV_u, SK_{PE,0}, \dots, SK_{PE,d})$ for a vector $\vec{y}$ and a user index u, an update key $UK_{T',R} = (CV_R, SK_{SUE,1}, \dots, SK_{SUE,m})$ for an update time T' and a revoked set of users R, and the public parameters PP.

1. If $u \notin R$, then it obtains (S_i, S_j) by running **CS.Match**(CV_R, PV_u). Otherwise, it outputs $\perp$.

2. If $T \leq T'$, then it obtains EK_{PE} and EK_{SUE} by running **PE.Decrypt**$(CH_{PE}, SK_{PE,j}, PP_{PE})$ and **SUE.Decrypt**$(CH_{SUE}, SK_{SUE,i}, PP_{SUE})$ respectively and obtains a message M by computing $C \cdot \big(EK_{PE} \cdot EK_{SUE}\big)^{-1}$. If $M \in \mathcal{M}$, then it outputs M. Otherwise, it outputs $\perp$.

The correctness of RS-PE easily follows from the correctness of PE, SUE, and SC. The proof of correctness is almost similar to that of RS-ABE except that this scheme uses a limited message space $\mathcal{M}$ to check $\langle \vec{x}, \vec{y} \rangle = 0$. We omit the proof of this.

In the PE scheme of Park [36], a ciphertext cannot be perfectly re-randomized without knowing the vector $\vec{x}$ of the ciphertext since $\vec{x}$ is hidden by the attribute-hiding property of PE. However, it is possible to partially re-randomize a ciphertext, and this re-randomized ciphertext is computationally indistinguishable from that of the encryption algorithm since the randomness that is not associated with $\vec{x}$ is perfectly randomized and the randomness associated with $\vec{x}$ is computationally hidden by the attribute-hiding property. Because of this, the output of **RS-PE.UpdateCT** may not be statistically close to that of **RS-PE.Encrypt**.

5.3 Security Analysis

One variant of PE is predicate-only PE (poPE) [22]. In poPE, a ciphertext is an encryption on a vector $\vec{x}$ instead of a vector $\vec{x}$ and a message M. The above PE scheme can be easily converted to a poPE scheme by removing g^{γ} from K_A in the private key and removing the session key from the ciphertext. Park proved that the poPE scheme is secure under the DLIN assumption.

Theorem 5.5 ([36]). *The predicate-only PE (poPE) scheme that is derived from the above PE scheme is selectively attribute-hiding if the DLIN assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{\mathcal{A}}^{poPE}(\lambda) \leq 1/2 \cdot 4\boldsymbol{Adv}_{\mathcal{B}}^{DLIN}(\lambda)$.*

Theorem 5.6. *The above RS-PE scheme is selectively secure under a chosen plaintext attack if the DBDH and DLIN assumptions hold. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{\mathcal{A}}^{RS\text{-}PE}(\lambda) \leq 2\boldsymbol{Adv}_{\mathcal{B}}^{DLIN}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}}^{DBDH}(\lambda)$.*

Proof. We first divide the behavior of an adversary as two types: Type-A and Type-B. The two types of adversaries are formally defined as follows:

Type-A. An adversary is Type-A if it submits two challenge messages M_0^*, M_1^* such that $M_0^* = M_1^*$.

Type-B. An adversary is Type-B if it submits two challenge messages M_0^*, M_1^* such that $M_0^* \neq M_1^*$.

The security proof consists of a sequence of hybrid games $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2$, and $\mathbf{G}_3$. We define the games as follows:

Game $\mathbf{G}_0$. In this game, the challenge ciphertext $CT^* = (CH_{PE}^*, CH_{SUE}^*, C^*)$ is an encryption on the vector $\vec{x}_0^*$, the time T^*, and the message M_0^*. That is, the ciphertext header CH_{PE}^* is an encryption on the vector $\vec{x}_0^*$ and the ciphertext header CH_{SUE}^* is an encryption on the time T^*, and the component C^* is an encryption on the message M_0^*. Note that this game is the original security game except that the challenge bit b is fixed to 0.

Game $\mathbf{G}_1$. This game is almost the same as the game $\mathbf{G}_0$ except that the ciphertext header CH_{PE}^* is an encryption on the vector $\vec{x}_1^*$. That is, the challenge ciphertext is an encryption on $\vec{x}_1^*$, T^*, and M_0^*.

Game $\mathbf{G}_2$. Next, we define a new game $\mathbf{G}_2$. This game is almost identical to the game G_1 except that the component C^* of the challenge ciphertext is replaced by an random element in $\mathbb{G}_T$.

Game $\mathbf{G}_3$. In the final game $\mathbf{G}_3$, the challenge ciphertext is an encryption on the vector $\vec{x}_1^*$, the time T^*, and the message M_1^*. Note that this game is the original security game except that the challenge bit b is fixed to 1.

Let $E_{\mathcal{A}}^{G_j}$ be an event that $\mathcal{A}$ outputs 0 in the game $\mathbf{G}_j$. We easily obtain $\Pr[E_{\mathcal{A}_A}^{G_1}] = \Pr[E_{\mathcal{A}_A}^{G_3}]$ since $M_0^* = M_1^*$ for the Type-A adversary. From Lemma 5.7, we obtain the equation for the Type-A adversary as

$$\Pr[E_{\mathcal{A}_A}^{G_0}] - \Pr[E_{\mathcal{A}_A}^{G_3}] = \Pr[E_{\mathcal{A}_A}^{G_0}] - \Pr[E_{\mathcal{A}_A}^{G_1}] \leq \big|\Pr[E_{\mathcal{A}_A}^{G_0}] - \Pr[E_{\mathcal{A}_A}^{G_1}]\big| \leq 4\mathbf{Adv}_{\mathcal{B}}^{DLIN}(\lambda).$$

From Lemma 5.8, 5.9, and 5.10, we obtain the equation for the Type-B adversary as

$$\begin{aligned}\Pr[E_{\mathcal{A}_B}^{G_0}] - \Pr[E_{\mathcal{A}_B}^{G_3}] &= \Pr[E_{\mathcal{A}_B}^{G_0}] + (\Pr[E_{\mathcal{A}_B}^{G_1}] - \Pr[E_{\mathcal{A}_B}^{G_1}]) + (\Pr[E_{\mathcal{A}_B}^{G_2}] - \Pr[E_{\mathcal{A}_B}^{G_2}]) - \Pr[E_{\mathcal{A}_B}^{G_3}] \\ &\leq \big|\Pr[E_{\mathcal{A}_B}^{G_0}] - \Pr[E_{\mathcal{A}_B}^{G_1}]\big| + \big|\Pr[E_{\mathcal{A}_B}^{G_1}] - \Pr[E_{\mathcal{A}_B}^{G_2}]\big| + \big|\Pr[E_{\mathcal{A}_B}^{G_2}] - \Pr[E_{\mathcal{A}_B}^{G_3}]\big| \\ &\leq 4\mathbf{Adv}_{\mathcal{B}}^{DLIN}(\lambda) + 2\mathbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda).\end{aligned}$$

Let E_A, E_B be the event that an adversary behaves like the Type-A and Type-B adversary respectively. Then we obtain the following inequality relation as

$$\begin{aligned}\mathbf{Adv}_{\mathcal{A}}^{RS\text{-}PE}(\lambda) &= \big|\Pr[b=0]\cdot\Pr[b=b'|b=0] + \Pr[b=1]\cdot\Pr[b=b'|b=1] - \frac{1}{2}\big| \\ &= \big|\frac{1}{2}\cdot\Pr[b'=0|b=0] + \frac{1}{2}\cdot(1-\Pr[b'=0|b=1]) - \frac{1}{2}\big| \\ &= \frac{1}{2}\cdot\big|\Pr[b'=0|b=0] - \Pr[b'=0|b=1]\big| \\ &\leq \frac{1}{2}\cdot\big|\Pr[E_{\mathcal{A}}^{G_0}] - \Pr[E_{\mathcal{A}}^{G_3}]\big| \\ &\leq \frac{1}{2}\cdot\big(\Pr[E_A]\cdot\big|\Pr[E_{\mathcal{A}_A}^{G_0}] - \Pr[E_{\mathcal{A}_A}^{G_3}]\big| + \Pr[E_B]\cdot\big|\Pr[E_{\mathcal{A}_B}^{G_0}] - \Pr[E_{\mathcal{A}_B}^{G_3}]\big|\big) \\ &\leq 2\mathbf{Adv}_{\mathcal{B}}^{DLIN}(\lambda) + \mathbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda).\end{aligned}$$

This completes our proof. □

Lemma 5.7. *If the DLIN assumption holds, then no polynomial-time Type-A adversary can distinguish between $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage.*

Proof. Suppose there exists a Type-A adversary $\mathcal{A}_A$ that attacks the above RS-PE scheme with a non-negligible advantage. A simulator $\mathcal{B}$ that attacks the attribute-hiding property of the predicate-only PE (poPE) scheme is given: a challenge public parameters PP_{poPE} of the poPE scheme. Then $\mathcal{B}$ that interacts with $\mathcal{A}_A$ is described as follows:

Init: $\mathcal{A}_A$ initially submits two challenge vectors $\vec{x}_0^*, \vec{x}_1^*$, a challenge time T^*, and a revoked set R^* at the time T^*. $\mathcal{B}$ also submits $\vec{x}_0^*, \vec{x}_1^*$ and receives PP_{poPE}.

Setup: $\mathcal{B}$ first derives $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$ from PP_{poPE}. It chooses a random exponent $\gamma \in \mathbb{Z}_p$ and sets $PP_{PE} = (PP_{poPE}, \Lambda = e(g,g)^\gamma)$. It obtains MK_{SUE}, PP_{SUE} by running **SUE.Setup**(GDS, T_{max}). It also obtains $\mathcal{BT}$ by running **CS.Setup** and assigns a random exponent γ_i to each node v_i in $\mathcal{BT}$. It selects a random exponent $\alpha \in \mathbb{Z}_p$ and publishes the public parameters $PP = (PP_{PE}, PP_{SUE}, g, \Omega = e(g,g)^\alpha)$.

Query 1: $\mathcal{A}_A$ adaptively request private keys or update keys. $\mathcal{B}$ proceeds as follows:

- If this is a private key query for a vector $\vec{y}$ and a user u, then it first obtains a private set PV_u by running **CS.Assign**$(\mathcal{BT}, u)$ and retrieves random exponents $\{\gamma_{j_0}, \ldots, \gamma_{j_d}\}$ from $\mathcal{BT}$. For $0 \le k \le d$, it request a private key for the poPE scheme and obtains $SK_{poPE} = (K_A, K_B, \ldots)$ from his private key oracle, and then it sets $SK_{PE,k} = (g^{\gamma_{j_k}} K_A, K_B, \ldots)$. Notice that K_A does not have g^{γ} in poPE and $g^{\gamma_{j_k}}$ is multiplied for secret sharing. It creates the private key $SK_{\vec{y},u} = (PV_u, SK_{PE,0}, \ldots, SK_{PE,d})$.
- If this is an update key query for a time T and a revoked set R, then it creates the update key $UK_{T,R}$ by running **RS-PE.UpdateKey**(T, R, MK, PP) since it knows α and MK_{SUE}.

Challenge: $\mathcal{A}_A$ submits two challenge messages M_0^*, M_1^* such that $M_0^* = M_1^* = M^*$. To create the challenge ciphertext, $\mathcal{B}$ obtains a challenge ciphertext header $CH_{poPE}^* = (C_A, C_B, \ldots)$. Next, it creates a ciphertext header CH_{SUE}^* as follows:

1. It first sets a label $L^* \in \{0,1\}^d$ by computing $\psi(T^*)$. It chooses random exponents $s_1, \ldots, s_d \in \mathbb{Z}_p$. It implicitly sets $s = c$ and builds ciphertext components $CH^{(0)}$ as
$$C_0 = C_A,\ C_1 = (C_A)^{w'} \prod_{i=1}^{d} (V_{ID^{-1}(L^{(0)}|_i)})^{s_i},\ C_{2,1} = g^{-s_1},\ \ldots,\ C_{2,d} = g^{-s_d}.$$
2. For $1 \le j \le d$, it first sets $L^{(j)} = L^*|_{d-j} \| 1$, and proceeds as follows: If $L^{(j)} = L|_{d-j+1}$, it sets $CH^{(j)}$ as an empty one. Otherwise, it selects $s_{d_j} \in \mathbb{Z}_p$ and builds ciphertext components $CH^{(j)}$ as
$$C_1 = (C_A)^{w'} \prod_{i=1}^{d_j - 1} (V_{ID^{-1}(L^{(j)}|_i)})^{s_i} (V_{ID^{-1}(L^{(j)})})^{s_{d_j}},\ C_{2,d_j} = g^{s_{d_j}}.$$
3. It removes all empty $CH^{(j)}$ and sets $CH_{SUE} = \big(CH^{(0)}, \ldots, CH^{(d')}\big)$ for some d' that consists of non-empty $CH^{(j)}$.
4. It sets the ciphertext header as $CH_{SUE}^* = CH_{SUE}$.

It sets a challenge ciphertext $CT^* = (CH_{poPE}^*, CH_{SUE}^*, C^* = e(C_A, g)^{\alpha} \cdot M^*)$ and gives CT^* to $\mathcal{A}$.
Query 2: Same as Query 1.
Guess: $\mathcal{A}$ outputs a guess b'. $\mathcal{B}$ also outputs b'.

To finish the proof, we first show that the distribution of the simulation is correct. The private keys are correctly distributed since SK_{poPE} returned from the private key oracle of poPE is correct and a random element $g^{\gamma_{j_k}}$ is correctly multiplied. The update keys are correctly distributed since it just runs **RS-PE.UpdateKey** by using the correct master secret key. The challenge ciphertext is also correctly distributed since CH_{poPE}^* is correctly distributed and the same element $C_A = g^s$ is used for CH_{SUE}^*. Therefore an adversary for this game can be used to attack the poPE scheme of Park and this leads to attack the DLIN assumption by Theorem 5.5. This completes our proof. □

Lemma 5.8. *If the DLIN assumption holds, then no polynomial-time Type-B adversary can distinguish between* $\boldsymbol{G}_0$ *and* $\boldsymbol{G}_1$ *with a non-negligible advantage.*

The proof of this lemma is the same as that of Lemma 5.7 since the challenge ciphertext of two games is an encryption on the same message M_0^*. We omit the proof of this lemma.

Lemma 5.9. *If the DBDH assumption holds, then no polynomial-time Type-B adversary can distinguish between $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ with a non-negligible advantage.*

Proof. To prove this lemma, we may try to combine the partitioning strategy of PE [36] and that of SUE in Theorem A.1. These partitioning strategies only work if $\langle \vec{x}^*, \vec{y} \rangle \neq 0$ and $T < T^*$ by the security definitions of PE and SUE respectively. However, this simple combination does not work in RS-PE since an adversary may request a private key for $\vec{y}$ such that $\langle \vec{x}^*, \vec{y} \rangle = 0$ and an update key for T such that $T^* \leq T$ if the user is revoked where $\vec{x}^*$ is a challenge vector and T^* is a challenge time. To solve this problem, we use the fact that the adversary first submits a set of revoked user R^* before he receives PP and the secret key α is split into γ_i for PE and $\alpha - \gamma_i$ for SUE by using $\mathcal{BT}$ of the CS scheme. That is, if a simulator cannot use the partitioning strategy of PE or SUE for some nodes in $\mathcal{BT}$, then it just uses γ_i for the master secret key. Additionally, to maintain the consistency of this secret splitting, it can use R^* that was given by the adversary.

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

Init: $\mathcal{A}_B$ initially submits two challenge vectors $\vec{x}_0^*, \vec{x}_1^*$, a challenge time T^*, and a set of revoked users R^* at the time T^*. It obtains $\mathcal{BT}$ by running **CS.Setup** and assigns a random exponent $\gamma_i \in \mathbb{Z}_p$ to each node v_i in $\mathcal{BT}$. For each user $u_i \in R^*$, it randomly assigns the user u_i to a leaf node $v_{u_i} \in \mathcal{BT}$. Let RV^* be the set of leaf nodes that are randomly assigned for R^*. Recall that $\mathbf{Path}(v)$ is the set of path nodes from the root node to the leaf node v. That is, $\mathbf{Path}(v) = \{v_{j_0}, \ldots, v_{j_d}\}$ where v_{j_0} is the root node and v_{j_d} is the leaf node such that $v_{j_d} = v$. Let $\mathbf{RevTree}(RV^*)$ be the minimal subtree that connects the root node to all leaf nodes in RV^*. That is, $\mathbf{RevTree}(RV^*) = \bigcup_{v_u \in RV^*} \mathbf{Path}(v_u)$.

Setup: $\mathcal{B}$ first chooses random exponents $\omega, \{w_{1,i}, f'_{1,i}, f'_{2,i}, t_{1,i}, h'_{1,i}, h'_{2,i}\}_{i=1}^n, u_1, u_2, v_1, v_2 \in \mathbb{Z}_p$, and $\gamma \in \mathbb{Z}_p$. It computes $\{w_{2,i}, t_{2,i}\}$ to satisfy the constrains $\omega = u_1 w_{2,i} - u_2 w_{1,i}$ and $\omega = v_1 t_{2,i} - v_2 t_{1,i}$. Let $\vec{x}_1^* = (x_1, \ldots, x_n)$. It implicitly sets $f_{1,i} = b x_i u_1 + f'_{1,i}, f_{2,i} = b x_i u_2 + f'_{2,i}, h_{1,i} = b x_i v_1 + h'_{1,i}, h_{2,i} = b x_i v_2 + h'_{2,i}$ and builds the public parameters PP_{PE} as

$$g,\ g^{\omega},\ \{W_{1,i} = g^{w_{1,i}}, W_{2,i} = g^{w_{2,i}}, F_{1,i} = (g^b)^{x_i u_1} g^{f'_{1,i}}, F_{2,i} = (g^b)^{x_i u_2} g^{f'_{2,i}}\}_{i=1}^n,$$
$$\{T_{1,i} = g^{t_{1,i}}, T_{2,i} = g^{t_{2,i}}, H_{1,i} = (g^b)^{x_i v_1} g^{h'_{1,i}}, H_{2,i} = (g^b)^{x_i v_2} g^{h'_{2,i}}\}_{i=1}^n,\ \{U_i = g^{u_i}, V_i = g^{v_i}\}_{i=1}^2,$$
$$\Lambda = e(g,g)^{\gamma}.$$

Next, it builds the public parameters PP_{SUE} as the same as Theorem A.1 except that $\Lambda = e(g,g)^{\beta}$ by selecting a random exponent $\beta \in \mathbb{Z}_p$. It implicitly sets $\alpha = ab$ and gives the public parameters $PP = \big(PP_{PE},\ PP_{SUE},\ g,\ \Omega = e(g^a, g^b)\big)$ to $\mathcal{A}$.

Query 1: $\mathcal{A}$ adaptively request private keys or update keys. If this is a private key query for a vector $\vec{y} = (y_1, \ldots, y_n)$ and a user u, then $\mathcal{B}$ proceeds as follows:

- **Case** $u \in R^*$: In this case, the simulator can simply creates PE private keys for path nodes since it will use γ_i from $\mathcal{BT}$ for the master key of PE.
    1. It first retrieves the leaf node $v_u \in RV^*$ that is assigned to the user u.
    2. Next, it obtains $\mathbf{Path}(v_u) = \{v_{j_0}, \ldots, v_{j_d}\}$ where $v_{j_d} = v_u$ and retrieves exponents $\{\gamma_{j_0}, \ldots, \gamma_{j_d}\}$ from $\mathcal{BT}$ that are associated with $\mathbf{Path}(v_u)$.

3. For all $v_{j_k} \in \mathbf{Path}(v_u)$, it selects random exponents $\lambda_1, \lambda_2, \{r_i, \phi_i\}_{i+1}^n \in \mathbb{Z}_p$ and builds a PE private key $SK_{PE,k}$ as

$$K_A = g^{\gamma_{j_k}} \cdot \prod_{i=1}^n F_{1,i}^{u_2 r_i - w_{2,i}\lambda_1 y_i} F_{2,i}^{-u_1 r_i + w_{1,i}\lambda_1 y_i} H_{1,i}^{v_2\phi_i - t_{2,i}\lambda_2 y_i} H_{2,i}^{-v_1\phi_i + t_{1,i}\lambda_2 y_i},\ K_B = \prod_{i=1}^n g^{-(r_i+\phi_i)},$$
$$\big\{K_{1,i} = U_2^{-r_i} W_{2,i}^{\lambda_1 y_i},\ K_{2,i} = U_1^{r_i} W_{1,i}^{-\lambda_1 y_i}\big\}_{i=1}^n,\ \big\{K_{3,i} = V_2^{-\phi_i} T_{2,i}^{\lambda_2 y_i},\ K_{4,i} = V_1^{\phi_i} T_{1,i}^{-\lambda_2 y_i}\big\}_{i=1}^n.$$

4. It creates the private key $SK_{\vec{y},u} = \big(PV_u, SK_{PE,0}, \dots, SK_{PE,d}\big)$.

- **Case** $u \notin R^*$: In this case, the simulator can use the partitioning strategy of PE [36] for some nodes since the adversary can only request $\vec{y}$ such that $\langle \vec{x}, \vec{y}\rangle \neq 0$.

  1. It first randomly assigns the user u to a leaf node v_u in $\mathcal{BT}$ such that $v_u \notin RV^*$.
  2. Next, it obtains $\mathbf{Path}(v_u) = \{v_{j_0}, \dots, v_{j_d}\}$ where $v_{j_d} = v_u$ and retrieves exponents $\{\gamma_{j_0}, \dots, \gamma_{j_d}\}$ from $\mathcal{BT}$ that are associated with $\mathbf{Path}(v_u)$.
  3. For all $v_{j_k} \in \mathbf{RevTree}(RV^*) \cap \mathbf{Path}(v_u)$, it builds a PE private key $SK_{PE,k}$ as

$$K_A = g^{\gamma_{j_k}} \cdot \prod_{i=1}^n F_{1,i}^{u_2 r_i - w_{2,i}\lambda_1 y_i} F_{2,i}^{-u_1 r_i + w_{1,i}\lambda_1 y_i} H_{1,i}^{v_2\phi_i - t_{2,i}\lambda_2 y_i} H_{2,i}^{-v_1\phi_i + t_{1,i}\lambda_2 y_i},\ K_B = \prod_{i=1}^n g^{-(r_i+\phi_i)},$$
$$\big\{K_{1,i} = U_2^{-r_i} W_{2,i}^{\lambda_1 y_i},\ K_{2,i} = U_1^{r_i} W_{1,i}^{-\lambda_1 y_i}\big\}_{i=1}^n,\ \big\{K_{3,i} = V_2^{-\phi_i} T_{2,i}^{\lambda_2 y_i},\ K_{4,i} = V_1^{\phi_i} T_{1,i}^{-\lambda_2 y_i}\big\}_{i=1}^n.$$

  4. For all $v_{j_k} \in \mathbf{Path}(v_u) \setminus (\mathbf{RevTree}(RV^*) \cap \mathbf{Path}(v_u))$, it selects random exponents $\lambda_1', \lambda_2', \{r_i, \phi_i\}_{i=1}^n \in \mathbb{Z}_p$ and builds a PE private key $SK_{PE,k}$ by implicitly setting $\lambda_1 = a/(2\omega z) + \lambda_1', \lambda_2 = a/(2\omega z) + \lambda_2'$ as

$$K_A = g^{-\gamma_{j_k}} \cdot (g^b)^{-\omega(\lambda_1'+\lambda_2')z} \prod_{i=1}^n K_{1,i}^{-f_{1,i}'} K_{2,i}^{-f_{2,i}'} K_{3,i}^{-h_{1,i}'} K_{4,i}^{-h_{2,i}'},\ K_B = \prod_{i=1}^n g^{-(r_i+\phi_i)},$$
$$\big\{K_{1,i} = U_2^{-r_i}(g^a)^{w_{2,i}y_i/(2\omega z)} W_{2,i}^{\lambda_1' y_i},\ K_{2,i} = U_1^{r_i}(g^a)^{-w_{1,i}y_i/(2\omega z)} W_{1,i}^{-\lambda_1' y_i}\big\}_{i=1}^n,$$
$$\big\{K_{3,i} = V_2^{-\phi_i}(g^a)^{t_{2,i}y_i/(2\omega z)} T_{2,i}^{\lambda_2 y_i},\ K_{4,i} = V_1^{\phi_i}(g^a)^{-t_{1,i}y_i/(2\omega z)} T_{1,i}^{-\lambda_2 y_i}\big\}_{i=1}^n$$

  where $\langle \vec{x}_1^*, \vec{y}\rangle = z \neq 0$ since $u \notin R^*$ from Remark 5.4.
  5. It creates the private key $SK_{\vec{y},u} = \big(PV_u, SK_{PE,0}, \dots, SK_{PE,d}\big)$.

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

- **Case** $T < T^*$: In this case, the simulator can use the partitioning strategy of SUE in Theorem A.1 for some nodes since the adversary can only request T such that $T < T^*$.

  1. It first obtains a label $L \in \{0,1\}^n$ by computing $\psi(T)$.
  2. Next, it obtains a covering set CV_R by running $\mathbf{CS.Cover}(\mathcal{BT}, R)$. Let $\mathbf{Cover}(R) = \{v_{i_1}, \dots, v_{i_m}\}$ be the set of nodes that are associated with CV_R.
  3. For all $v_{i_k} \in \mathbf{RevTree}(RV^*) \cap \mathbf{Cover}(R)$, it selects a random exponent $r' \in \mathbb{Z}_p$ and builds an SUE private key $SK_{SUE,k}$ by implicitly setting $r = -b + r'$ as

$$K_0 = (g^b)^{-w'} w^{r'},\ K_1 = (g^b)^{-1} g^{r'},$$
$$K_{2,1} = (g^b)^{-v'_{ID^{-1}(L|_1)}} \big(V_{ID^{-1}(L|_1)}\big)^{r'},\ \dots,\ K_{2,n} = (g^b)^{-v'_{ID^{-1}(L|_n)}} \big(V_{ID^{-1}(L|_n)}\big)^{r'}.$$

4. For all $v_{i_k} \in \mathbf{Cover}(R) \setminus (\mathbf{RevTree}(RV^*) \cap \mathbf{Cover}(R))$, it selects a random exponent $r \in \mathbb{Z}_p$ and builds an SUE private key $SK_{SUE,k}$ as
$$K_0 = g^{\gamma_{i_k}} \cdot w^{-r},\ K_1 = g^r,\ K_{2,1} = (V_{ID^{-1}(L|_1)})^r,\ \ldots,\ K_{2,n} = (V_{ID^{-1}(L|_n)})^r.$$
5. It creates the update key $UK_{T,R} = (CV_R, SK_{SUE,1}, \ldots, SK_{SUE,m})$.

- **Case** $T \geq T^*$: In this case, the simulator can simply create SUE private keys since it will use γ_i from $\mathcal{BT}$ for the master key of SUE. Note that if $T \geq T^*$, then $\mathbf{RevTree}(RV^*) \cap \mathbf{Cover}(R) = \emptyset$ since $R^* \subseteq R$ from the definition of the security model.

1. It first obtains a label $L \in \{0,1\}^n$ by computing $\psi(T)$.
2. Next, it obtains a covering set CV_R by running $\mathbf{CS.Cover}(\mathcal{BT}, R)$. Let $\mathbf{Cover}(R) = \{v_{i_1}, \ldots, v_{i_m}\}$ be the set of nodes that are associated with CV_R.
3. For all $v_{i_k} \in \mathbf{Cover}(R)$, it selects a random exponent $r \in \mathbb{Z}_p$ and builds an SUE private key $SK_{SUE,k}$ as
$$K_0 = g^{\gamma_{i_k}} \cdot w^{-r},\ K_1 = g^r,\ K_{2,1} = (V_{ID^{-1}(L|_1)})^r,\ \ldots,\ K_{2,n} = (V_{ID^{-1}(L|_n)})^r.$$
4. It creates the update key $UK_{T,R} = (CV_R, SK_{SUE,1}, \ldots, SK_{SUE,m})$.

Challenge: $\mathcal{A}$ submits two challenge messages M_0^*, M_1^*. $\mathcal{B}$ first selects random exponents $s_1, s_3', s_4' \in \mathbb{Z}_p$, and builds the ciphertext header CH_{PE}^* by implicitly setting $s = c, s_3 = -bc + s_3', s_4 = -bc + s_4'$ as
$$C_A = g^c,\ C_B = (g^{\omega})^{s_1},\ \big\{C_{1,i} = W_{1,i}^{s_1}(g^c)^{f_{1,i}'}U_1^{x_i s_3'},\ C_{2,i} = W_{2,i}^{s_1}(g^c)^{f_{2,i}'}U_2^{x_i s_3'}\big\}_{i=1}^n,$$
$$\big\{C_{3,i} = T_{1,i}^{s_1}(g^c)^{h_{1,i}'}V_1^{x_i s_4'},\ C_{4,i} = T_{2,i}^{s_1}(g^c)^{h_{2,i}'}V_2^{x_i s_4'}\big\}_{i=1}^n.$$
Next, it builds the ciphertext header CH_{SUE}^* for SUE as the same as Theorem A.1 by using the element g^c given in the assumption. It finally creates the challenger ciphertext $CT^* = (CH_{PE}^*, CH_{SUE}^*, C^* = Z \cdot M_0^*)$ and gives it to $\mathcal{A}$.
Query 2: Same as Query 1.
Guess: $\mathcal{A}$ outputs a guess b'. $\mathcal{B}$ also outputs b'.

To finish the proof, we should show that the distribution of the simulation is correct. We first show that private keys are correctly distributed. The PE private key for $v_{j_k} \in \mathbf{RevTree}(RV^*) \cap \mathbf{Path}(v_u)$ is correctly distributed as
$$\begin{aligned} K_A &= g^{\gamma_{j_k}} \cdot \prod_{i=1}^n F_{1,i}^{u_2 r_i - w_{2,i}\lambda_1 y_i} F_{2,i}^{-u_1 r_i + w_{1,i}\lambda_1 y_i} H_{3,i}^{v_2\phi_i - t_{2,i}\lambda_2 y_i} H_{4,i}^{-v_1\phi_i + t_{1,i}\lambda_2 y_i} \\ &= g^{\gamma_{j_k}} \cdot \prod_{i=1}^n g^{-(-u_2 r_i + w_{2,i}\lambda_1 y_i) f_{1,i}} g^{-(u_1 r_i - w_{1,i}\lambda_1 y_i) f_{2,i}} g^{-(-v_2\phi_i + t_{2,i}\lambda_2 y_i) h_{1,i}} g^{-(v_1\phi_i - t_{1,i}\lambda_2 y_i) h_{2,i}} \\ &= g^{\gamma_{j_k}} \cdot \prod_{i=1}^n K_{1,i}^{-f_{1,i}} K_{2,i}^{-f_{2,i}} K_{3,i}^{-h_{1,i}} K_{4,i}^{-h_{2,i}}. \end{aligned}$$
We next analyze the distribution of the PE private key for $v_{j_k} \in \mathbf{Path}(v_u) \setminus (\mathbf{RevTree}(RV^*) \cap \mathbf{Path}(v_u))$. The components $K_{1,i}, K_{2,i}$ of the PE private key are correctly distributed as
$$\begin{aligned} K_{1,i} &= U_2^{-r_i}(g^a)^{w_{2,i} y_i/(2\omega z)} W_{2,i}^{\lambda_1' y_i} = U_2^{-r_i} W_{2,i}^{(a/(2\omega z) + \lambda_1') y_i}, \\ K_{2,i} &= U_1^{r_i}(g^a)^{-w_{1,i} y_i/(2\omega z)} W_{1,i}^{-\lambda_1' y_i} = U_1^{r_i} W_{1,i}^{-(a/(2\omega z) + \lambda_1') y_i}. \end{aligned}$$

To analyze the distribution of K_A, we first obtain the following two equations

$$\begin{aligned}
K_{1,i}^{-(bx_iu_1+f'_{1,i})} K_{2,i}^{-(bx_iu_2+f'_{2,i})} &= (U_2^{-r_i} W_{2,i}^{(a/(2\omega z)+\lambda'_1)y_i})^{-(bx_iu_1)} K_{1,i}^{-f'_{1,i}} \cdot (U_1^{r_i} W_{1,i}^{-(a/(2\omega z)+\lambda'_1)y_i})^{-(bx_iu_2)} K_{2,i}^{-f'_{2,i}} \\
&= (g^{ab})^{(-u_1w_{2,i}+u_2w_{1,i})\cdot x_iy_i/(2\omega z)} (g^b)^{(-u_1w_{2,i}+u_2w_{1,i})\lambda'_1 \cdot x_iy_i} \cdot K_{1,i}^{-f'_{1,i}} K_{2,i}^{-f'_{2,i}} \\
&= (g^{ab})^{-x_iy_i/(2z)} (g^b)^{-\omega\lambda'_1 \cdot x_iy_i} \cdot K_{1,i}^{-f'_{1,i}} K_{2,i}^{-f'_{2,i}}, \\
K_{3,i}^{-(bx_iv_1+h'_{1,i})} K_{4,i}^{-(bx_iv_2+h'_{2,i})} &= (g^{ab})^{-x_iy_i/(2z)} (g^b)^{-\omega\lambda'_2 \cdot x_iy_i} \cdot K_{3,i}^{-h'_{1,i}} K_{4,i}^{-h'_{2,i}}.
\end{aligned}$$

Using the above two equations and the relation $\langle \vec{x}, \vec{y} \rangle = z$, we have that the component K_A is correctly distributed as

$$\begin{aligned}
K_A &= g^{-\gamma_{j_k}} \cdot \prod_{i=1}^{n} (g^b)^{-\omega(\lambda'_1+\lambda'_2)\cdot x_iy_i} K_{1,i}^{-f'_{1,i}} K_{2,i}^{-f'_{2,i}} K_{3,i}^{-h'_{1,i}} K_{4,i}^{-h'_{2,i}} \\
&= g^{ab} g^{-\gamma_{j_k}} \cdot (g^{ab})^{-2\sum_{i=1}^{n} x_iy_i/(2z)} \prod_{i=1}^{n} (g^b)^{-\omega(\lambda'_1+\lambda'_2)\cdot x_iy_i} K_{1,i}^{-f'_{1,i}} K_{2,i}^{-f'_{2,i}} K_{3,i}^{-h'_{1,i}} K_{4,i}^{-h'_{2,i}} \\
&= g^{ab} g^{-\gamma_{j_k}} \cdot \prod_{i=1}^{n} (g^{ab})^{-2x_iy_i/(2z)} (g^b)^{-\omega(\lambda'_1+\lambda'_2)\cdot x_iy_i} K_{1,i}^{-f'_{1,i}} K_{2,i}^{-f'_{2,i}} K_{3,i}^{-h'_{1,i}} K_{4,i}^{-h'_{2,i}} \\
&= g^{\alpha-\gamma_{j_k}} \cdot \prod_{i=1}^{n} K_{1,i}^{-(bx_iu_1+f'_{1,i})} K_{2,i}^{-(bx_iu_2+f'_{2,i})} K_{3,i}^{-(bx_iv_1+h'_{1,i})} K_{4,i}^{-(bx_iv_2+h'_{2,i})}.
\end{aligned}$$

The update keys are correctly distributed since it is the same as Theorem A.1. The challenge PE ciphertext header is correctly distributed as

$$\begin{aligned}
C_{1,i} &= W_{1,i}^{s_1} (g^c)^{f'_{1,i}} U_1^{x_is'_3} = W_{1,i}^{s_1} ((g^b)^{x_iu_1} g^{f'_{1,i}})^c U_1^{x_i(-bc+s'_3)} = W_{1,i}^{s_1} F_{1,i}^c U_1^{x_i(-bc+s'_3)}, \\
C_{3,i} &= T_{1,i}^{s_1} (g^c)^{h'_{1,i}} V_1^{x_is'_4} = T_{1,i}^{s_1} ((g^b)^{x_iv_1} g^{h'_{1,i}})^c V_1^{x_is'_4} = T_{1,i}^{s_1} H_{1,i}^c V_1^{x_i(-bc+s'_4)}.
\end{aligned}$$

This completes our proof. □

Lemma 5.10. *If the DBDH assumption holds, then no polynomial-time Type-B adversary can distinguish between* $\boldsymbol{G}_2$ *and* $\boldsymbol{G}_3$ *with a non-negligible advantage.*

We omit the proof of this lemma since it is symmetric to the proof of Lemma 5.9.

5.4 Discussions

Achieving Full Security. Our RS-PE scheme is only selectively secure under standard assumptions since we use the selectively secure PE scheme of Park [36] for our primary encryption scheme. To achieve full security, we may use the fully secure PE scheme of Okamoto and Takashima [34] in the dual pairing vector space setting. However, it is unclear whether we can obtain a fully secure SUE scheme with shorter ciphertexts in the dual pairing vector space by applying the translation method of Lewko [25] to our composite order SUE scheme.

Acknowledgements

We thank the anonymous reviewers of ASIACRYPT 2013 for their helpful comments.

References

[1] Mihir Bellare and Sara K. Miner. A forward-secure digital signature scheme. In Michael J. Wiener, editor, *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 431–448. Springer, 1999.

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

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

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

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

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

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

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

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

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

[11] Melissa Chase. Multi-authority attribute based encryption. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 515–534. Springer, 2007.

[12] Sherman S. M. Chow, Volker Roth, and Eleanor G. Rieffel. General certificateless encryption and timed-release encryption. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN*, volume 5229 of *Lecture Notes in Computer Science*, pages 126–143. Springer, 2008.

[13] Yevgeniy Dodis, Matthew K. Franklin, Jonathan Katz, Atsuko Miyaji, and Moti Yung. Intrusion-resilient public-key encryption. In Marc Joye, editor, *CT-RSA*, volume 2612 of *Lecture Notes in Computer Science*, pages 19–32. Springer, 2003.

[14] Yevgeniy Dodis, Jonathan Katz, Shouhuai Xu, and Moti Yung. Key-insulated public key cryptosystems. In Lars R. Knudsen, editor, *EUROCRYPT*, volume 2332 of *Lecture Notes in Computer Science*, pages 65–82. Springer, 2002.

[15] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 479–499. Springer, 2013.

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

[17] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *STOC*, pages 555–564. ACM, 2013.

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

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

[20] Gene Itkis and Leonid Reyzin. Sibir: Signer-base intrusion-resilient signatures. In Moti Yung, editor, *CRYPTO*, volume 2442 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2002.

[21] Kohei Kasamatsu, Takahiro Matsuda, Keita Emura, Nuttapong Attrapadung, Goichiro Hanaoka, and Hideki Imai. Time-specific encryption from forward-secure encryption. In Ivan Visconti and Roberto De Prisco, editors, *SCN*, volume 7485 of *Lecture Notes in Computer Science*, pages 184–204. Springer, 2012.

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

[23] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *J. Cryptology*, 26(2):191–224, 2013.

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

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

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

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

[28] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 568–588. Springer, 2011.

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

[30] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 180–198. Springer, 2012.

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

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

[33] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.

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

[35] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM Conference on Computer and Communications Security*, pages 195–203. ACM, 2007.

[36] Jong Hwan Park. Inner-product encryption under standard assumptions. *Des. Codes Cryptography*, 58(3):235–257, 2011.

[37] Jong Hwan Park, Kwangsu Lee, Willy Susilo, and Dong Hoon Lee. Fully secure hidden vector encryption under standard assumptions. *Inf. Sci.*, 232:188–207, 2013.

[38] Kenneth G. Paterson and Elizabeth A. Quaglia. Time-specific encryption. In Juan A. Garay and Roberto De Prisco, editors, *SCN*, volume 6280 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2010.

[39] Ronald L. Rivest, Adi Shamir, and David A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684, 1996.

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

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

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

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

[44] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 560–578. Springer, 2008.

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

[46] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.

A Self-Updatable Encryption in Prime Order Groups

In this section, we propose an SUE scheme in prime order bilinear groups and prove its selective security.

A.1 Bilinear Groups of Prime Order

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

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

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

A.2 Complexity Assumptions

Decisional Linear (DLIN) Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of prime order p. Let g be generators of subgroups $\mathbb{G}$. The DLIN assumption is that if the challenge tuple

$$D = ((p, \mathbb{G}, \mathbb{G}_T, e), g, g^x, g^y, g^{xa}, g^{yb}) \text{ and } Z,$$

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

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

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

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

A.3 Construction

Our CDE scheme in prime order groups is described as follows:

CDE.Init(1^λ): This algorithm takes as input a security parameter 1^λ. It generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p. Let g be the generator of $\mathbb{G}$. It outputs a group description string as $GDS = \big((p, \mathbb{G}, \mathbb{G}_T, e),\ g\big)$.

CDE.Setup(GDS, d_{max}): This algorithm takes as input the string GDS and the maximum depth d_{max} of a full binary tree $\mathcal{BT}$. The total number X of nodes in the tree is $2^{d_{max}+1} - 1$. It chooses random elements $w, V_0, \ldots, V_{X-1} \in \mathbb{G}$ and a random exponent $\beta \in \mathbb{Z}_p$ where a node v_i is associated with the element V_i. It outputs the master secret key $MK = (\beta)$ and the public parameters as

$$PP = \Big((p, \mathbb{G}, \mathbb{G}_T, e),\ g,\ w,\ V_0, \ldots, V_{X-1},\ \Omega = e(g,g)^\beta\Big).$$

CDE.GenKey(L, MK, PP): This algorithm takes as input a label $L \in \{0,1\}^n$, the master secret key MK, and the public parameters PP. Let v be the node with the label L in $\mathcal{BT}$. It selects a random exponent $r \in \mathbb{Z}_p$ and outputs a private key for the node v that implicitly includes L as

$$SK_L = \Big(K_0 = g^\beta w^{-r},\ K_1 = g^r,\ K_{2,1} = (V_{ID^{-1}(L|_1)})^r,\ \ldots,\ K_{2,n} = (V_{ID^{-1}(L|_n)})^r\Big).$$

CDE.Encrypt$(L, s, \vec{s}, PP)$: This algorithm takes as input a label $L \in \{0,1\}^d$, a random exponent $s \in \mathbb{Z}_p$, a vector $\vec{s} = (s_1, \ldots, s_d) \in \mathbb{Z}_p^d$ of random exponents, and the public parameters PP. Let v be the node with the label L in $\mathcal{BT}$. It outputs a ciphertext header for the node v that implicitly includes L as

$$CH_L = \Big(C_0 = g^s,\ C_1 = w^s \prod_{i=1}^{d} (V_{ID^{-1}(L|_i)})^{s_i},\ C_{2,1} = g^{-s_1},\ \ldots,\ C_{2,d} = g^{-s_d}\Big).$$

and a session key as $EK = \Omega^s$.

CDE.DelegateCT(CH_L, c, PP): This algorithm takes as input a ciphertext header $CH_L = (C_0, \ldots, C_{2,d})$ for a label $L \in \{0,1\}^d$, a bit value $c \in \{0,1\}$, and the public parameters PP. It selects a random exponent $s_{d+1} \in \mathbb{Z}_p$ and outputs a delegated ciphertext header for the node v' with the label $L' = L \| c$ as

$$CH_{L'} = \Big(C_0,\ C'_1 = C_1 \cdot (V_{ID^{-1}(L')})^{s_{d+1}},\ C_{2,1},\ \ldots,\ C_{2,d},\ C'_{2,d+1} = g^{-s_{d+1}}\Big).$$

CDE.RandCT($CH_L, s', \vec{s}', PP$): This algorithm takes as input a ciphertext header CH_L, a random exponent $s' \in \mathbb{Z}_p$, a vector $\vec{s}' = (s'_1, \ldots, s'_d) \in \mathbb{Z}_p^d$ of random exponents, and the public parameters PP. It outputs a re-randomized ciphertext header as

$$CH'_L = \Big(C'_0 = C_0 \cdot g^{s'},\ C'_1 = C_1 \cdot w^{s'} \prod_{i=1}^{d} (V_{ID^{-1}(L|_i)})^{s'_i},\ C'_{2,1} = C_{2,1} \cdot g^{-s'_1},\ \ldots,\ C'_{2,d} = C_{2,d} \cdot g^{-s'_d} \Big).$$

and a partial session key as $EK' = \Omega^{s'}$ that will be multiplied with the session key EK of CH_L.

CDE.Decrypt($CH_L, SK_{L'}, PP$): This algorithm takes as input a ciphertext header CH_L for a label $L \in \{0,1\}^d$, a private key $SK_{L'}$ for a label $L' \in \{0,1\}^n$, and the public parameters PP. If L is a prefix of L', then it computes a delegated ciphertext header $CH'_{L'} = (C'_0, \ldots, C'_{2,n})$ by running **DelegateCT** and outputs a session key as

$$EK = e(C'_0, K_0) \cdot e(C'_1, K_1) \cdot \prod_{i=1}^{n} e(C'_{2,i}, K_{2,i})$$

Otherwise, it outputs $\perp$.

Our SUE scheme in prime order groups is almost the same as the SUE scheme in composite order groups described in Section 3 except that it uses the above CDE scheme in prime order groups and selects random exponents in $\mathbb{Z}_p$ instead of $\mathbb{Z}_N$. We omit the description of our SUE scheme in prime order groups.

A.4 Security Analysis

Theorem A.1. *The above SUE scheme is selectively secure under a chosen plaintext attack if the DBDH assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that $\textbf{Adv}_{\mathcal{A}}^{SUE}(\lambda) \le \frac{1}{2}\textbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda)$.*

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

Init: $\mathcal{A}$ initially submits a challenge time T^*. $\mathcal{B}$ first obtains a challenge node $v^* \in \mathcal{BT}$ that is associated with the challenge time T^* by computing $v^* = ID^{-1}(\psi(T^*))$. Recall that **TimeNodes**(v) is the set of nodes that consists of v and the right siblings of path nodes of v that are not in the parent's path. That is, **TimeNodes**(v) $= \{v\} \cup$ **RightSibling**(**Path**(v)) $\setminus$ **Path**(**Parent**(v)).

Setup: $\mathcal{B}$ first chooses random exponents $w', v'_0, \ldots, v'_{X-1} \in \mathbb{Z}_p$. It implicitly sets $\beta = ab$ and publishes the public parameters PP as

$$g,\ w = g^a g^{w'},\ \{V_i = g^a g^{v'_i}\}_{v_i \in \textbf{TimeNodes}(v^*)}, \{V_i = g^{v'_i}\}_{v_i \in \mathcal{BT} \setminus \textbf{TimeNodes}(v^*)},\ \Lambda = e(g^a, g^b).$$

Query 1: $\mathcal{A}$ adaptively request a private key for a time T such that $T < T^*$. $\mathcal{B}$ first obtains a label $L \in \{0,1\}^n$ by computing $\psi(T)$. Next, it selects a random exponent $r' \in \mathbb{Z}_p$ and creates a private key by implicitly setting $r = -b + r'$ as

$$K_0 = (g^b)^{-w'} w^{r'},\ K_1 = (g^b)^{-1} g^{r'},$$
$$K_{2,1} = (g^b)^{-v'_{ID^{-1}(L|_1)}} (V_{ID^{-1}(L|_1)})^{r'},\ \ldots,\ K_{2,n} = (g^b)^{-v'_{ID^{-1}(L|_n)}} (V_{ID^{-1}(L|_n)})^{r'}.$$

Note that if $T < T^*$, then it can create a private key since $\textbf{Path}(v) \cap \textbf{TimeNodes}(v^*) = \emptyset$ where $v = ID^{-1}(\psi(T))$.
Challenge: To create the challenge ciphertext for the challenge time T^*, $\mathcal{B}$ proceeds as follows:

1. It first sets a label $L^* \in \{0,1\}^d$ by computing $\psi(T^*)$. It chooses random exponents $s_1, \dots, s_{d-1}, s'_d \in \mathbb{Z}_p$. It implicitly sets $s = c, s_d = -c + s'_d$ and creates ciphertext components $CH^{(0)}$ as

$$C_0 = g^c,\ C_1 = (g^c)^{w'} \cdot \prod_{i=1}^{d-1} (V_{ID^{-1}(L^{(0)}|_i)})^{s_i} \cdot (g^c)^{-v'_{ID^{-1}(L^{(0)})}} (V_{ID^{-1}(L^{(0)})})^{s'_d},$$
$$C_{2,1} = g^{-s_1},\ \dots,\ C_{2,d-1} = g^{-s_{d-1}},\ C_{2,d} = (g^c)^{-1} g^{s'_d}.$$

2. For $1 \le j \le d$, it first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: If $L^{(j)} = L|_{d-j+1}$, it sets $CH^{(j)}$ as an empty one. Otherwise, it selects $s'_{d_j} \in \mathbb{Z}_p$ and creates ciphertext components $CH^{(j)}$ as

$$C_1 = (g^c)^{w'} \cdot \prod_{i=1}^{d-j} (V_{ID^{-1}(L^{(j)}|_i)})^{s_i} \cdot (g^c)^{-v'_{ID^{-1}(L^{(j)})}} (V_{ID^{-1}(L^{(j)})})^{s'_{d-j+1}},\ C_{2,d-j+1} = (g^c)^{-1} g^{s'_{d-j+1}}.$$

3. It removes all empty $CH^{(j)}$ and sets $CH_T = \big(CH^{(0)}, \dots, CH^{(d')}\big)$ for some d' that consists of non-empty $CH^{(j)}$.

4. It sets the challenger ciphertext header as $CH_{T^*} = CH_T$ and the session key $EK = Z$. It gives CH_{T^*} and EK to $\mathcal{A}$.

Note that it can create the challenge ciphertext for T^* since for all labels $L^{(j)}$ in the challenge ciphertext, $v_j \in \textbf{TimeNodes}(v^*)$ where $v_j = ID^{-1}(\psi(L^{(j)}))$.
Query 2: Same as Query 1.
Guess: $\mathcal{A}$ outputs a guess b'. If $b = b'$, then $\mathcal{B}$ outputs 1. Otherwise, it outputs 0.

To finish the proof, we should show that the distribution of the simulation is correct. The private key is correctly distributed as

$$K_0 = (g^b)^{-w'} w^{r'} = g^{ab} (g^a g^{w'})^{-b+r'},\ K_1 = (g^b)^{-1} g^{r'} = g^{-b+r'},$$
$$K_{2,i} = (g^b)^{-v'_{ID^{-1}(L|_i)}} (V_{ID^{-1}(L|_i)})^{r'} = (V_{ID^{-1}(L|_i)})^{-b+r'}.$$

The challenge ciphertext header is also correctly distributed as

$$C_1 = (g^c)^{w'} \cdot \prod_{i=1}^{d-1} (V_{ID^{-1}(L^{(0)}|_i)})^{s_i} \cdot (g^c)^{-v'_{ID^{-1}(L^{(0)})}} (V_{ID^{-1}(L^{(0)})})^{s'_d}$$
$$= (g^a g^{w'})^c \cdot \prod_{i=1}^{d-1} (V_{ID^{-1}(L^{(0)}|_i)})^{s_i} \cdot (g^a g^{v'_{ID^{-1}(L^{(0)})}})^{-c+s'_d},$$
$$C_{2,d} = (g^c)^{-1} g^{s'_d} = g^{-c+s'_d}.$$

This completes our proof. □