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

Kwangsu Lee[1], Seung Geol Choi[2], Dong Hoon Lee[1], Jong Hwan Park[1,3], and Moti Yung[4]

[1] CIST, Korea University, Korea
[2] US Naval Academy, USA
[3] Sangmyung University, Korea
[4] Google Inc. and Columbia University, USA

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. 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.

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 [14, 18, 30]. 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 [29], 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. [29] 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 [19, 31].

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. [18] 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. [29], 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 [26] 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. [8] 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. [22]. 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 $\alpha - \gamma_j$).

1.3 Other Applications

Timed-Release Encryption. One application of SUE is timed-release encryption (TRE) and its variants [27, 28]. 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) [10, 27]. 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) [12]. 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 [11].

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 [30] introduced fuzzy IBE (F-IBE) that is a special type of ABE. Goyal et al. [14] 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 [9, 18, 20, 25, 32].

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. Boneh and Waters [7] introduced the concept of PE and proposed a hidden vector encryption (HVE) scheme that supports conjunctive queries on encrypted data. Katz et al. [15] proposed a PE scheme that supports inner-product queries on encrypted data. After that, many PE schemes with different properties were proposed [17, 23, 24, 26]. Boneh, Sahai, and Waters [6] formalized the concept of functional encryption (FE) by generalizing ABE and PE.

Revocation. Boneh and Franklin [5] 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 [21] proposed a fully secure R-IBE scheme.

2 Preliminaries

2.1 Notation

We let λ be a security parameter. Let $[n]$ denote the set $\{1,\ldots,n\}$ 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. 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. 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.

2.3 Subset Cover Framework

The subset cover (SC) framework introduced by Naor, Naor, and Lotspiech [22] is a general methodology for the construction of efficient revocation systems. 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. The formal definition of SC is given in the full version of this paper [16].

We use the complete subset (CS) scheme proposed by Naor et al. [22] 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 $(v_{j_0}, v_{j_1}, \ldots, v_{j_d})$ be the path from the root node $v_{j_0} = v_0$ to the leaf node $v_{j_n} = v_u$. It sets $PV_u = \{S_{j_0}, \ldots, S_{j_d}\}$, 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 $\mathcal{T}_{i_1}, \ldots \mathcal{T}_{i_m}$ be all the subtrees of $\mathcal{BT}$ that hang off $ST(R)$, that is all subtrees whose roots $v_{i_1}, \ldots v_{i_m}$ are not in $ST(R)$ but adjacent to nodes of outdegree 1 in $ST(R)$. It outputs a covering set $CV_R = \{S_{i_1}, \ldots, S_{i_m}\}$.

CS.Match(CV_R, PV_u): This algorithm takes input as a covering set $CV_R = \{S_{i_1}, \ldots, S_{i_m}\}$ and a private set $PV_u = \{S_{j_0}, \ldots, S_{j_d}\}$. 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 1 ([22]). *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

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. The following is the syntax of CDE.

Definition 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$(L, s, \mathbf{s}, PP)$*. The encryption algorithm takes as input a label string* $L \in \{0,1\}^d$ *with* $d \leq d_{max}$*, a random exponent s, an exponent vector* $\mathbf{s}$*, 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$(CH_L, s', \mathbf{s}, PP)$*. The ciphertext randomization algorithm takes as input a ciphertext header* CH_L *for a label string* $L \in \{0,1\}^d$ *with* $d < d_{max}$*, a new random exponent* s'*, a new vector* $\mathbf{s}$*, 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****.*

Remark 1. The syntax of CDE is different with the usual syntax of encryption since the encryption algorithm additionally takes input random values instead of selecting its own randomness. Because of this difference, we cannot show the security of SUE under the security of CDE, but this syntax difference is essential for the ciphertext efficiency of SUE.

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 2 (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, s, PP)*. The encryption algorithm takes as input a time T, a random value s, 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, s', PP)*. The ciphertext randomization algorithm takes as input a ciphertext header* CH_T *for a time T, a new random exponent* s'*, 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* $T \leq T'$*, then* $\textbf{Decrypt}(CH_T, SK_{T'}, PP) = EK$*.*
– *If* $T > T'$*, then* $\textbf{Decrypt}(CH_T, SK_{T'}, PP) = \perp$ *with all but negligible probability.*

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

Remark 2. 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.

Definition 3 (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 $T_1, \dots, T_{q'}$, and $\mathcal{C}$ gives the corresponding private keys $SK_{T_1}, \dots, SK_{T_{q'}}$ 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^*, s, 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 $T_{q'+1}, \dots, T_q$ 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 $\boldsymbol{Adv}_{\mathcal{A}}^{SUE}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|$ 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. 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 $T_i < T^*$.

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 $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ has the following properties:

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

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

3.3 Complexity Assumptions

We give three static assumptions in bilinear groups of composite order that were introduced by Lewko and Waters [19]. The Assumption 1 (Subgroup Decision), the Assumption 2 (General Subgroup Decision), and the Assumption 3 (Composite Diffie-Hellman) are described in the the full version of this paper [16].

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 [8]. 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 the pre-order tree traversal that recursively visits the root node, the left subtree, and the right subtree. 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, $\mathbf{TimeNodes}(v) = \{v\} \cup \mathbf{RightSibling}(\mathbf{Path}(v)) \setminus \mathbf{Path}(\mathbf{Parent}(v))$. Pre-order traversal has the property such that if a node v is associated with time T and a node v' is associated with time T', then we have

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

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.

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, 13]. To construct a CDE scheme that supports the ciphertext delegation property, we start from the HIBE scheme of Boneh and Boyen [4] and interchange 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 $L \in \{0,1\}^*$. 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.

[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.

3.5 Construction

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

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

CDE.GenKey(L, MK, PP): This algorithm takes as input a label string $L \in \{0,1\}^n$, 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 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($L, s, \boldsymbol{s}, PP$): This algorithm takes as input a label string $L \in \{0,1\}^d$, a random exponent $s \in \mathbb{Z}_N$, a vector $\boldsymbol{s} = (s_1, \ldots, s_d) \in \mathbb{Z}_N^d$ of random exponents, and PP. It outputs a ciphertext header 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 $L \in \{0,1\}^d$, a bit value $c \in \{0,1\}$, and 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($CH_L, s', \boldsymbol{s}', PP$): This algorithm takes as input a ciphertext header $CH_L = (C_0, \ldots, C_{2,d})$ for a label string $L \in \{0,1\}^d$, a new random exponent $s' \in \mathbb{Z}_N$, a new vector $\boldsymbol{s}' = (s_1', \ldots, s_d') \in \mathbb{Z}_N^d$, and 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 $L' \in \{0,1\}^n$, and PP. If L is a prefix of L', then it computes $CH'_{L'} = (C'_0, \dots, C'_{2,n})$ 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 $\perp$.

Let ψ be a mapping from time T to a label L[2]. 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 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 $\mathbf{s} = (s_1, \dots, s_d)$ by selecting random exponents $s_1, \dots, s_d \in \mathbb{Z}_N$, and obtains $CH^{(0)}$ by running **CDE.Encrypt**$(L, s, \mathbf{s}, PP)$.
2. For $1 \le j \le d$, 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.
   (b) Otherwise, it sets a new exponent vector $\mathbf{s}' = (s'_1, \dots, s'_{d-j+1})$ where $s'_1, \dots s'_{d-j}$ are copied from $\mathbf{s}$ and s'_{d-j+1} is randomly selected in $\mathbb{Z}_N$ since $L^{(j)}$ and L have the same prefix string. It obtains $CH^{(j)} = (C'_0, \dots, C'_{2,d-j+1})$ by running **CDE.Encrypt**$(L^{(j)}, s, \mathbf{s}', PP)$. It also prunes the redundant elements $C'_0, C'_{2,1}, \dots, C'_{2,d-j}$ from $CH^{(j)}$, which are already contained in $CH^{(0)}$.
3. It removes all empty $CH^{(j)}$ and sets $CH_T = \big(CH^{(0)}, CH^{(1)}, \dots, CH^{(d')}\big)$ for some $d' \le d$ that consists of non-empty $CH^{(j)}$.
4. It outputs a ciphertext header as CH_T and a session key as $EK = \Lambda^s$.

SUE.UpdateCT($CH_T, T+1, PP$): This algorithm takes as input a ciphertext header $CH_T = (CH^{(0)}, \dots, CH^{(d)})$ for a time T, a next time $T+1$, and 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 $CH_{T+1} = \big(CH'^{(0)} = CH_{L^{(0)}||0}, CH'^{(1)} = CH_{L^{(0)}||1}, CH'^{(2)} = CH^{(1)}, \dots, CH'^{(d+1)} = CH^{(d)}\big)$.

[2] In a full binary tree, each node is associated with a unique time T by the 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.

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 $CH_{T+1} = \big(CH'^{(0)} = CH^{(1)}, \ldots, CH'^{(d-1)} = CH^{(d)}\big)$.

SUE.RandCT(CH_T, s', PP): This algorithm takes as input a ciphertext header $CH_T = (CH^{(0)}, \ldots, CH^{(d)})$ for a time T, a new random exponent $s' \in \mathbb{Z}_N$, and 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 $\vec{s}' = (s'_1, \ldots, s'_{d^{(0)}})$ by selecting random exponents $s'_1, \ldots, s'_{d^{(0)}} \in \mathbb{Z}_N$, and obtains $CH'^{(0)}$ by running $\mathbf{CDE.RandCT}(CH^{(0)}, s', \vec{s}', PP)$.
2. For $1 \le j \le d$, it sets a new vector $\vec{s}'' = (s'_1, \ldots, s'_{d^{(j)}})$ where $s'_1, \ldots s'_{d^{(j)}-1}$ are copied from $\vec{s}'$ and $s'_{d^{(j)}}$ is randomly chosen in $\mathbb{Z}_N$, and obtains $CH'^{(j)}$ by running $\mathbf{CDE.RandCT}(CH^{(j)}, s', \vec{s}'', PP)$.
3. It outputs a re-randomized ciphertext header as $CH'_T = \big(CH'^{(0)}, \ldots, CH'^{(d)}\big)$ 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 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 $\mathbf{CDE.Decrypt}(CH^{(j)}, SK_{T'}, PP)$. Otherwise, it outputs $\perp$.

Remark 4. 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 $CH^{(0)}, CH^{(1)}, \ldots, CH^{(d)}$ that are associated with the nodes in $\mathbf{TimeNodes}(v)$. 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 $\mathbf{TimeNodes}(v) \cap \mathbf{Path}(v') = v''$ since the property of the 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 $s'' = (s_1 + s'_1, \dots, s_d + s'_d)$ 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 $s', s'_1, \dots, s'_{d^{(0)}}$ are chosen and these random exponents are reused among the CDE ciphertext headers.

3.7 Security Analysis

Theorem 1. *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 $\boldsymbol{Adv}^{SUE}_{\mathcal{A}}(\lambda) \leq \boldsymbol{Adv}^{A1}_{\mathcal{B}}(\lambda) + 2q\boldsymbol{Adv}^{A2}_{\mathcal{B}}(\lambda) + \boldsymbol{Adv}^{A3}_{\mathcal{B}}(\lambda)$ where q is the maximum number of private key queries of $\mathcal{A}$.*

The proof of this theorem is given in the full version of this paper [16].

4 Revocable-Storage Attribute-Based Encryption

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. 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 [29].

Definition 4 (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* ***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* ***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***.

Definition 5 (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* ***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* ***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 **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 $\mathbf{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) = \left|\Pr[b = b'] - \frac{1}{2}\right|$ 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 5. 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.

4.2 Construction

For our RS-ABE scheme, we use the (ciphertext-policy) ABE scheme of Lewko et al. [18] as a primary encryption scheme with slight modifications. That is, we use the key encapsulation mechanism version of CP-ABE and the encryption algorithm additionally takes input a random exponent for a session key. The detailed description of CP-ABE is given in the full version of this paper [16]. Our RS-ABE scheme 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 $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$ and $PP = \big(PP_{ABE}, PP_{SUE}, g = g_1, \Omega = e(g,g)^\alpha\big)$.

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

1. It first obtains $PV_u = \{S_{j_0}, \ldots, S_{j_d}\}$ by running **CS.Assign**$(\mathcal{BT}, u)$ and retrieves $\{\gamma_{j_0}, \ldots, \gamma_{j_d}\}$ from $\mathcal{BT}$ where γ_i is assigned to the node v_i.
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 $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d}\big)$.

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

1. It first obtains $CV_R = \{S_{i_1}, \dots, S_{i_m}\}$ by running **CS.Cover**$(\mathcal{BT}, R)$ and retrieves $\{\gamma_{i_1}, \dots, \gamma_{i_m}\}$ from $\mathcal{BT}$.
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 $UK_{T,R} = \big(CV_R, SK_{SUE,1}, \dots, SK_{SUE,m}\big)$.

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, and PP. 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. It outputs 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 PP. It obtains CH'_{SUE} by running **SUE.UpdateCT**$(CH_{SUE}, T+1, PP_{SUE})$. It outputs $CT_{\mathbb{A},T+1} = \big(CH_{ABE}, CH'_{SUE}, C\big)$.

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 PP. It first selects a random exponent $s' \in \mathbb{Z}_N$. It obtains CH'_{ABE} and CH'_{SUE} by running **ABE.RandCT**(CH_{ABE}, s', PP_{ABE}) and **SUE.RandCT** (CH_{SUE}, s', PP_{SUE}), respectively. It outputs $CT'_{\mathbb{A},T} = \big(CH'_{ABE}, CH'_{SUE}, C' = C \cdot \Omega^{s'}\big)$.

RS-ABE.Decrypt($CT_{\mathbb{A},T}, SK_{S,u}, UK_{T',R}, PP$): This algorithm takes as input a ciphertext $CT_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE}, C)$, a private key $SK_{S,u} = (PV_u, SK_{ABE,0}, \dots, SK_{ABE,d})$, an update key $UK_{T',R} = (CV_R, SK_{SUE,1}, \dots, SK_{SUE,m})$, and 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 $T \le T'$, then it can obtain 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 M by computing $C \cdot \big(EK_{ABE} \cdot EK_{SUE}\big)^{-1}$. Otherwise, it outputs $\perp$.

Remark 6. 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. [29] 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**.

Theorem 2. *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 $\boldsymbol{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) \le \boldsymbol{Adv}_{\mathcal{B}}^{A1}(\lambda) + O(q) \cdot \boldsymbol{Adv}_{\mathcal{B}}^{A2}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}}^{A3}(\lambda)$ where q is the maximum number of private key and update key queries of $\mathcal{A}$.*

The proof of this theorem is given in the full version of this paper [16].

4.3 Discussions and RS-PE Results

Efficiency. In our RS-ABE scheme, the number of group elements in a ciphertext is $2l + 3\log T_{max}$ where l is the row size of an access structure. In the RS-ABE scheme of Sahai et al. [29], 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.

Revocable-Storage Predicate Encryption. If we use the PE scheme of Park [26] as a primary encryption scheme, then we can build an RS-PE scheme in prime order bilinear groups that additionally supports attribute-hiding property. The definition, construction, and proof of RS-PE are given in the full version of this paper [16].

Theorem 3. *The RS-PE scheme is selectively secure under a chosen plaintext attack if the DBDH and the DLIN assumptions hold. That is, for any PPT adversary $\mathcal{A}$, we have that* $\boldsymbol{Adv}_{\mathcal{A}}^{RS\text{-}PE}(\lambda) \leq 2\boldsymbol{Adv}_{\mathcal{B}}^{DLIN}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}}^{DBDH}(\lambda)$.

Acknowledgements. We thank the anonymous reviewers of ASIACRYPT 2013 for their helpful comments. This work was partially done while Kwangsu Lee and Seung Geol Choi were at Columbia University. Kwangsu Lee was supported by Basic Science Research Program through NRF funded by the Ministry of Education (2013R1A1A2008394). Dong Hoon Lee was supported by Mid-career Researcher Program through NRF grant funded by the MEST (2010-0029121). Jong Hwan Park was supported by Basic Science Research Program through NRF funded by the Ministry of Education (2013R1A1A2009524).

References

1. Bellare, M., Miner, S.K.: A forward-secure digital signature scheme. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 431–448. Springer, Heidelberg (1999)
2. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Computer Society (2007)
3. Boldyreva, A., Goyal, V., Kumar, V.: Identity-based encryption with efficient revocation. In: Ning, P., Syverson, P.F., Jha, S. (eds.) ACM Conference on Computer and Communications Security, pp. 417–426. ACM (2008)
4. Boneh, D., Boyen, X.: Efficient selective-ID secure identity-based encryption without random oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
5. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
6. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 253–273. Springer, Heidelberg (2011)
7. Boneh, D., Waters, B.: Conjunctive, subset, and range queries on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
8. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 255–271. Springer, Heidelberg (2003)
9. Chase, M.: Multi-authority attribute based encryption. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 515–534. Springer, Heidelberg (2007)
10. Chow, S.S.M., Roth, V., Rieffel, E.G.: General certificateless encryption and timed-release encryption. In: Ostrovsky, R., De Prisco, R., Visconti, I. (eds.) SCN 2008. LNCS, vol. 5229, pp. 126–143. Springer, Heidelberg (2008)
11. Dodis, Y., Franklin, M., Katz, J., Miyaji, A.: Intrusion-resilient public-key encryption. In: Joye, M. (ed.) CT-RSA 2003. LNCS, vol. 2612, pp. 19–32. Springer, Heidelberg (2003)
12. Dodis, Y., Katz, J., Xu, S., Yung, M.: Key-insulated public key cryptosystems. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 65–82. Springer, Heidelberg (2002)

13. Gentry, C., Silverberg, A.: Hierarchical ID-based cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501, pp. 548–566. Springer, Heidelberg (2002)
14. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Juels, A., Wright, R.N., di Vimercati, S.D.C. (eds.) ACM Conference on Computer and Communications Security, pp. 89–98. ACM (2006)
15. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
16. Lee, K., Choi, S.G., Lee, D.H., Park, J.H., Yung, M.: Self-updatable encryption: Time constrained access control with hidden attributes and better efficiency. Cryptology ePrint Archive (2013), `http://eprint.iacr.org/2013/`
17. Lee, K., Lee, D.H.: Improved hidden vector encryption with short ciphertexts and tokens. Des. Codes Cryptography 58(3), 297–319 (2011)
18. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (Hierarchical) inner product encryption. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 62–91. Springer, Heidelberg (2010)
19. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 455–479. Springer, Heidelberg (2010)
20. Lewko, A., Waters, B.: Decentralizing attribute-based encryption. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 568–588. Springer, Heidelberg (2011)
21. Libert, B., Vergnaud, D.: Adaptive-ID secure revocable identity-based encryption. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 1–15. Springer, Heidelberg (2009)
22. Naor, D., Naor, M., Lotspiech, J.: Revocation and tracing schemes for stateless receivers. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 41–62. Springer, Heidelberg (2001)
23. Okamoto, T., Takashima, K.: Hierarchical predicate encryption for inner-products. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 214–231. Springer, Heidelberg (2009)
24. Okamoto, T., Takashima, K.: Adaptively attribute-hiding (Hierarchical) inner product encryption. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 591–608. Springer, Heidelberg (2012)
25. Ostrovsky, R., Sahai, A., Waters, B.: Attribute-based encryption with non-monotonic access structures. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) ACM Conference on Computer and Communications Security, pp. 195–203. ACM (2007)
26. Park, J.H.: Inner-product encryption under standard assumptions. Des. Codes Cryptography 58(3), 235–257 (2011)
27. Paterson, K.G., Quaglia, E.A.: Time-specific encryption. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 1–16. Springer, Heidelberg (2010)
28. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684 (1996)
29. Sahai, A., Seyalioglu, H., Waters, B.: Dynamic credentials and ciphertext delegation for attribute-based encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 199–217. Springer, Heidelberg (2012)
30. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
31. Waters, B.: Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In: Halevi, S. (ed.) CRYPTO 2009. LNCS, vol. 5677, pp. 619–636. Springer, Heidelberg (2009)
32. Waters, B.: Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 53–70. Springer, Heidelberg (2011)