# Self-Updatable Encryption with Short Public Parameters and Its Extensions

Kwangsu Lee*

**Abstract**

Cloud storage is very popular since it has many advantages, but there is a new threat to cloud storage that was not considered before. *Self-updatable encryption* that updates a past ciphertext to a future ciphertext by using a public key is a new cryptographic primitive introduced by Lee, Choi, Lee, Park, and Yung (Asiacrypt 2013) to defeat this threat such that an adversary who obtained a past-time private key can still decrypt a (previously unread) past-time ciphertext stored in cloud storage. Additionally, an SUE scheme can be combined with an attribute-based encryption (ABE) scheme to construct a powerful revocable-storage ABE (RS-ABE) scheme introduced by Sahai, Seyalioglu, and Waters (Crypto 2012) that provides the key revocation and ciphertext updating functionality for cloud storage. In this paper, we propose an efficient SUE scheme and its extended schemes. First, we propose an SUE scheme with short public parameters in prime-order bilinear groups and prove its security under a $q$-type assumption. Next, we extend our SUE scheme to a time-interval SUE (TI-SUE) scheme that supports a time interval in ciphertexts. Our TI-SUE scheme has short public parameters and also secure under the $q$-type assumption. Finally, we propose the first large universe RS-ABE scheme with short public parameters in prime-order bilinear groups and prove its security in the selective revocation list model under a $q$-type assumption.

**Keywords:** Public-key encryption, Self-updatable encryption, Ciphertext delegation, Cloud storage, Bilinear maps.

*Korea University, Korea. Email: guspin@korea.ac.kr.

## Contents

## 1 Introduction

Cloud storage is very popular in these days since the cost of service is low, data management is easy, and data can be accessed through the internet without the restriction of a geographical location. If data that contains sensitive information is stored in cloud storage, then this data should be encrypted and stored as a ciphertext. To provide an access control on these ciphertexts, a ciphertext can be specified with a time $T$, and a user who has a master key can delegate his key to other delegator by giving a (delegated) private key with a time $T$ that only can be used to decrypt a ciphertext with the same time $T$. In a typical communication system, this simple method can reduce the damage caused by the exposure of the delegated private key with a time $T$ since a ciphertext is only used during a short time period and an adversary who obtained a private key with $T$ cannot decrypt a ciphertext with a time $T'$ such that $T \neq T'$. However, this adversary can cause a serious problem in cloud storage since ciphertexts are stored in cloud storage at all times and the adversary who has a private key with a past time can still decrypt a (previously unread) ciphertext with the past time. This problem is a new threat to cloud storage.

A naive approach that solves this problem is that cloud storage keeps the master key of a user and a ciphertext with a past time is decrypted and then it is encrypted again with a new time by cloud storage. However, this approach requires trusted cloud storage that we would like to avoid and the master keys in cloud storage can be an easy target of attackers. A better solution for this problem is that cloud storage only keeps the public key of a user and a ciphertext with a past time is updated to a new one with a new time by just using the public key. Self-updatable encryption (SUE) that was introduced by Lee *et al.* [12] is a new cryptographic primitive that provides this functionality. In SUE, a ciphertext is associated with a time $T$ and a private key is also associated with a time $T'$. If $T \leq T'$, then the ciphertext with $T$ can be decrypted by the private key with $T'$. Additionally, the ciphertext with $T$ can be updated to a new one with $T+1$ by just using public parameters. Lee *et al.* proposed an efficient SUE scheme in bilinear groups and also constructed an efficient revocable-storage attribute-based encryption (RS-ABE) scheme that provides the key revocation and ciphertext updating functionality for cloud storage. SUE also can be used for timed-release encryption (TRE) and key-insulated encryption (KIE) [12].

The previous SUE schemes of Lee *et al.* have $O(\log T_{max})$ number of group elements in public parameters if composite-order bilinear groups are used or $O(T_{max})$ number of group elements in public parameters if prime-order bilinear groups are used where $T_{max}$ is the maximum number of times. We ask whether there is a practical SUE construction that has short public parameters in prime-order bilinear groups.

### 1.1 Our Results

In this paper, we propose an efficient SUE scheme and its extended schemes by modifying the previous SUE scheme. The followings are our results:

**SUE with short public parameters**. We first propose an efficient SUE scheme in prime-order bilinear groups that has constant number of group elements in public parameters and prove its security under a $q$-type assumption. Compared with the previous SUE schemes of Lee *et al.* [12] that have $O(\log T_{max})$ (or $O(T_{max})$) number of group elements in public parameters, our SUE scheme just has $O(1)$ number of group elements in public parameters, and the number of group elements in a private key and a ciphertext is $O(\log T_{max})$ and $O(\log T_{max})$ respectively as the same as that of the previous SUE schemes. To devise an SUE scheme with short public parameters, we add additional layer that has a new randomness to the structure of a private key. This added new layer enables the simulation of a polynomial number of private keys, and the number of group elements in public parameters can be reduced by the power of a $q$-type assumption.

**Time-Interval SUE with short parameters**. One natural extension of SUE is to specify a time interval in a ciphertext. By extending our SUE scheme, we propose a time-interval SUE (TI-SUE) scheme with short public parameters such that a ciphertext is associated with a time interval. In TI-SUE, a ciphertext is specified with a time interval $[T_L, T_R]$ and a private key with a time $T$ can be used to decrypt this ciphertext if $T_L \leq T \leq T_R$. To devise a TI-SUE scheme, we combine two SUE scheme by using a simple secret sharing scheme to prevent collusion attacks, and prove its security under a $q$-type assumption. In the security proof, we introduce a meta-simulation technique that uses previous simulators of SUE as sub-simulators to simplify the security proof. This meta-simulation technique has an independent interest.

**RS-ABE with short parameters**. The main application of SUE is RS-ABE for cloud storage that provides the key revocation and ciphertext updating functionality if SUE is combined with ABE. Sahai, Seyalioglu, and Waters [22] introduced the concept of RS-ABE and Lee *et al.* [12] showed that an efficient RS-ABE scheme can be constructed from an SUE scheme. We propose a large universe RS-ABE scheme with short public parameters in prime-order bilinear groups by combining the ciphertext-policy ABE (CP-ABE) scheme of Rouselakist and Waters [21] and our SUE scheme, and prove its selective revocation list security under the $q$-type assumption introduced by Rouselakis and Waters. Our RS-ABE scheme is the first efficient RS-ABE scheme with short public parameters that supports a large universe of attributes.

## 1.2 Related Work

SUE is related with timed-release encryption (TRE) since private keys and ciphertexts are associated with times. In TRE, a ciphertext is specified with a releasing time $T$ and a private key with a time $T'$ is broadcasted to all users by a trusted-center periodically at each time $T'$. The ciphertext with $T$ can be decrypted by the private key with $T'$ if $T \leq T'$. The concept of TRE was introduced by May [16] and its extensions were proposed in [19, 20]. As mentioned, an SUE scheme can be easily converted to a TRE scheme by using the ciphertext updating property. One notable difference between TRE and SUE is that a past private key that was broadcasted is used to decrypt a past ciphertext in TRE, whereas a current private key is used to decrypt a past ciphertext by updating the ciphertext in SUE.

SUE is also related with key-insulated encryption (KIE) and forward-secure encryption (FSE) that mitigate the damage of key exposure. In KIE, a master key is stored in a physically secure device and a temporal key for a time $T'$ derived from the master key is used to decrypt a ciphertext with a time $T$ if $T = T'$. Note that the exposure of a temporal key with a time $T'$ does not damage the ciphertext security at a time $T$ such that $T \neq T'$. The concept of KIE was introduced by Dodis *et al.* [9] and its extension was presented in [8]. As mentioned before, SUE can be used to enhance the ciphertext security of KIE by updating ciphertexts if these are stored in cloud storage. In FSE, a private key with a time $T$ is evolved to another private key with a next time $T+1$ at each time period, and then the past private key is erased. The forward security ensures that an exposed private key with a time $T'$ cannot be used to decrypt a past ciphertext with a time $T$ if $T < T'$. The first FSE scheme was proposed by Canetti *et al.* [7] by using a hierarchical HIBE (HIBE) scheme and an efficient FSE scheme was presented in [3]. We may view SUE as the dual concept of FSE since the role of private keys and that of ciphertexts are reversed.

Identity-based encryption (IBE) that generates a delegated key for an identity is related with SUE since a delegated (or limited) private key for a time $T$ is generated for a user in SUE. In IBE, a ciphertext is associated with an identity $ID$ and a private key with $ID'$ can be used to decrypt this ciphertext if $ID = ID'$. The first IBE scheme was presented by Boneh and Franklin [4] by using bilinear maps and other IBE schemes were proposed in [2, 25]. IBE also can be extended to HIBE, ABE, predicate encryption (PE), and functional encryption (FE) [5, 6, 10, 11]. To minimize the damage of private key exposure in IBE, the revocation functionality that can revoked a user is required. An efficient revocable IBE (RIBE) scheme was proposed by Boldyreva *et al.* [1] and its improvement was presented in [13, 15, 18, 23, 24]. Recently, Sahai *et al.* [22] introduced the concept of RS-ABE for cloud storage and Lee *et al.* [12] presented an improved RS-ABE and RS-PE schemes by using an SUE scheme.

## 2 Preliminaries

In this section, we introduce the definition of self-updatable encryption and give the necessary background of bilinear groups and complexity assumptions.

### 2.1 Notation

We let $\lambda$ 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 $i$th 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 Self-Updatable Encryption

Self-updatable encryption (SUE) is a new type of public-key encryption such that a ciphertext associated with a time can be updated to a future time by using public parameters, and the concept of this primitive was introduced by Lee *et al.* [12]. We follow the SUE definition of Lee *et al.* In SUE, a ciphertext associated with a time $T$ is stored in a cloud storage. A user who has a private key associated with a time $T'$ such that $T \leq T'$ can decrypt a ciphertext with a time $T$ in the cloud storage. Additional feature of SUE is that a ciphertext with a time $T$ can be easily converted to a new time $T+1$ by the cloud storage to prevent a user who has a past private key from accessing the ciphertext in the cloud storage. The formal syntax of SUE is defined as follows:

**Definition 2.1** (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***$(1^\lambda)$*. The initialization algorithm takes as input a security parameter* $1^\lambda$*, 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 a master key MK and public parameters PP.*

***GenKey***$(T, MK, PP)$*. The key generation algorithm takes as input a time* $T$*, the master 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 a 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 of SUE is defined as follows: For all $MK, PP$ generated by **Setup**, 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 **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**.*

The security model of SUE was introduced by Lee *et al.* [12] and we use a selective security model such that an adversary initially submits a challenge time $T^*$. In the selective security game, an adversary initially submits a challenge time $T^*$ and receives public parameters. After that, he may request private keys for a time that is less than the challenge time. In a challenge step, he is given a ciphertext header and a challenge session key that is a well-formed one or a random one depending on a random coin. Additionally he can request private keys and he finally outputs a coin guess. If the guess is correct, then he wins the game. The formal definition is given as follows:

**Definition 2.2** (Selective Security)**.** *The selective security of SUE is defined in terms of the indistinguishability under chosen plaintext attacks (IND-CPA). The security game is defined as the following experiment between a challenger $\mathcal{C}$ and a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$:*

1. ***Init**: $\mathcal{A}$ initially submits a challenge time $T^*$.*
2. ***Setup**: $\mathcal{C}$ generates a master key $MK$ and public parameters $PP$ by running **Init** and **Setup**, and it gives $PP$ to $\mathcal{A}$.*
3. ***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**$(T_i, MK, PP)$ with the following restriction: For any time $T_i$ of private key queries, it is required that $T_i < T^*$.*
4. ***Challenge**: $\mathcal{C}$ chooses a random bit $\mu \in \{0,1\}$ and computes a ciphertext header $CH^*$ and a session key $EK^*$ by running **Encrypt**$(T^*, PP)$. If $\mu = 0$, then it gives $CH^*$ and $EK^*$ to $\mathcal{A}$. Otherwise, it gives $CH^*$ and a random session key to $\mathcal{A}$.*
5. ***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 $\mathcal{A}$.*
6. ***Guess**: Finally $\mathcal{A}$ outputs a bit $\mu'$.*

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

## 2.3 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 bit string 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 $L(i)$ 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 $L(v_i)$ as $L(i)$ if there is no ambiguity.

For the notational convenience, we define additional functions in a full binary tree. **Parent**$(L)$ is a function that returns the parent node's label $L'$ of the input node with the label $L$. **RightChild**$(L)$ is a function that returns the right child's label $L'$ of the input node with $L$. **RightSibling**$(L)$ is a function that returns the right sibling node's label $L'$ of the input node with $L$. That is, **RightSibling**$(L) =$ **RightChild**$($**Parent**$(L))$. Finally, **Path**$(L)$ is a function that returns a set of path node's labels from the root node to the input node with $L$.

## 2.4 Bilinear Groups

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 $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ 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.

## 2.5 Complexity Assumptions

In this subsection, we introduce a $q$-type assumption and the standard assumption for the proof of our schemes. Before introducing our $q$-type assumption, we first introduce the $q$-RW1 assumption that was introduced by Rouselakis and Waters [21] to prove the security of their ciphertext-policy ABE scheme. Our $q$-sRW assumption is a simplified version of the $q$-RW1 assumption. The DBDH assumption is the standard assumption that was extensively used in the proof of other schemes.

**Assumption 2.3** ($q$-RW1, [21])**.** *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 q-RW1 assumption is that if the challenge tuple*

$$D = \Big((p,\mathbb{G},\mathbb{G}_T,e), g, g^c, \big\{g^{a^i}, g^{d_j}, g^{cd_j}, g^{a^i d_j}, g^{a^i/d_j^2}\big\}_{\forall 1\le i,j\le q}, \big\{g^{a^i/d_j}\big\}_{\forall 1\le i\le 2q, i\neq q+1, \forall 1\le j\le q},$$
$$\big\{g^{a^i d_j/d_{j'}^2}\big\}_{\forall 1\le i\le 2q, \forall 1\le j,j'\le q, j'\neq j}, \big\{g^{a^i c d_j/d_{j'}}, g^{a^i c d_j/d_{j'}^2}\big\}_{\forall 1\le i,j,j'\le q, j'\neq j}\Big) \text{ and } Z,$$

*are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = e(g,g)^{a^{q+1}c}$ from $Z = Z_1 = e(g,g)^f$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\textbf{Adv}_{\mathcal{A}}^{q\text{-}RW1}(\lambda) = \big|\Pr[\mathcal{A}(D,Z_0)=0] - \Pr[\mathcal{A}(D,Z_1)=0]\big|$ where the probability is taken over random choices of $a, c, \{d_j\}_{1\le j\le q}, f \in \mathbb{Z}_p$.*

**Lemma 2.4** ( [21])**.** *The q-RW1 assumption is secure in the generic group model.*

**Assumption 2.5** ($q$-simplified RW, $q$-sRW)**.** *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 q-sRW assumption is that if the challenge tuple*

$$D = \Big((p,\mathbb{G},\mathbb{G}_T,e), g, g^a, g^b, g^c, \big\{g^{d_j}, g^{cd_j}, g^{ad_j}, g^{b/d_j^2}, g^{a/d_j}\big\}_{\forall 1\le j\le q},$$
$$\big\{g^{abd_j/d_{j'}^2}, g^{acd_j/d_{j'}}, g^{bcd_j/d_{j'}^2}\big\}_{\forall 1\le j,j'\le q, j'\neq j}\Big) \text{ and } Z,$$

*are given, no PPT algorithm $\mathcal{A}$ can distinguish $Z = Z_0 = e(g,g)^{abc}$ from $Z = Z_1 = e(g,g)^f$ with more than a negligible advantage. The advantage of $\mathcal{A}$ is defined as $\textbf{Adv}_{\mathcal{A}}^{q\text{-}sRW}(\lambda) = \big|\Pr[\mathcal{A}(D,Z_0)=0] - \Pr[\mathcal{A}(D,Z_1)=0]\big|$ where the probability is taken over random choices of $a, b, c, \{d_j\}_{1\le j\le q}, f \in \mathbb{Z}_p$.*

**Lemma 2.6.** *The q-sRW assumption is secure in the generic group model if the q-RW1 assumption is secure in the generic group model.*

The proof of this Lemma is easily obtained since all elements of the $q$-sRW assumption can be derived from the elements of the $q$-RW1 assumption by simply setting $b = a^q$ where $b$ is a random element in the $q$-sRW assumption and $q$ is a parameter in the $q$-RW1 assumption. That is, if there exists an adversary $\mathcal{A}$ that attacks the $q$-sRW assumption, then there exists an algorithm $\mathcal{B}$ that attacks the $q$-RW1 assumption by using $\mathcal{A}$ since $\mathcal{B}$ can derive all elements of the $q$-sRW assumption from that of the $q$-RW1 assumption.

**Assumption 2.7** (Decisional Bilinear Diffie-Hellman, DBDH, [4])**.** *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*

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

*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 $\textbf{Adv}_{\mathcal{A}}^{DBDH}(\lambda) = \big|\Pr[\mathcal{A}(D,Z_0)=0] - \Pr[\mathcal{A}(D,Z_1)=0]\big|$ where the probability is taken over random choices of $a, b, c, d \in \mathbb{Z}_p$.*

## 3 Self-Updatable Encryption with Short Parameters

In this section, we propose an SUE scheme with short public parameters in prime-order bilinear groups and prove its security under a $q$-type assumption.

## 3.1 Design Principle

To devise an SUE scheme that supports the ciphertext updating property, we follow the design principle of Lee *et al.* [12] that constructs an SUE scheme from a CDE scheme that supports the ciphertext delegation property by carefully reusing the randomness of the CDE ciphertexts. Lee *et al.* derived a CDE scheme from the HIBE scheme of Boneh and Boyen [2] by switching the structure of private keys and that of ciphertexts, but this CDE scheme can not be easily proven by using the partitioning method since a polynomial number of private keys cannot be simulated. To solve this problem, they constructed a CDE scheme in composite-order bilinear groups and proved its security by using the dual system encryption method of Waters [26]. Note that they also proposed a CDE scheme in prime-order groups and proved its security in the partitioning method, but the size of public parameters is depend on the number of tree nodes.

The main reason of this difficulty that prevents the security proof of their CDE scheme by using the partitioning method is that only one private key of CDE can be simulated since the structure of CDE private keys is derived from the structure of HIBE ciphertexts. Note that only one challenge ciphertext of HIBE is simulated in the security proof of HIBE. To solve this difficult problem, we add an additional layer of Boneh and Boyen's HIBE [2] to the structure of private keys. By adding this additional layer, we can easily simulate private keys since the randomness of a private key in Boneh and Boyen's HIBE can be used to cancel out an element that cannot be simulated. To achieve constant size of public parameters, we use a $q$-type assumption since multiple values can be programmed in shorter public parameters by the power of this $q$-type assumption.

## 3.2 Construction

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

**CDE.Init**($1^\lambda$)**:** This algorithm takes as input a security parameter $1^\lambda$. 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**($GDS, l$)**:** This algorithm takes as input the string $GDS$ and the maximum length $l$ of label strings. It chooses random elements $w, v, u, h \in \mathbb{G}$ and a random exponent $\beta \in \mathbb{Z}_p$. It outputs a master key $MK = \beta$ and public parameters as

$$PP = \Big((p, \mathbb{G}, \mathbb{G}_T, e),\ g,\ w,\ v,\ u,\ h,\ \Lambda = e(g,g)^\beta\Big).$$

**CDE.GenKey**($L, MK, PP$)**:** This algorithm takes as input a label string $L \in \{0,1\}^n$, the master key $MK$, and the public parameters $PP$. It selects random exponents $r, r_1, \ldots, r_n \in \mathbb{Z}_p$ and outputs a private key that implicitly includes $L$ as

$$SK_L = \Big(K_0 = g^\beta w^r,\ K_1 = g^{-r},\ \big\{K_{i,1} = v^r (u^{L|_i} h)^{r_i},\ K_{i,2} = g^{-r_i}\big\}_{i=1}^{n}\Big).$$

**CDE.RandKey**($SK_L, \delta, PP$)**:** This algorithm takes as input a private key $SK_L = (K_0, K_1, \{K_{i,1}, K_{i,2}\})$, an exponent $\delta \in \mathbb{Z}_p$, and the public parameters $PP$. It selects random exponents $r', r'_1, \ldots, r'_n \in \mathbb{Z}_p$ and outputs a re-randomized private key as

$$SK_L = \Big(K'_0 = K_0 \cdot g^\delta w^{r'},\ K'_1 = K_1 \cdot g^{-r'},\ \big\{K'_{i,1} = K_{i,1} \cdot v^{r'} (u^{L|_i} h)^{r'_i},\ K'_{i,2} = K_{i,2} \cdot g^{-r'_i}\big\}_{i=1}^{n}\Big).$$

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

$$CH_L = \Big(C_0 = g^t,\ C_1 = w^t \prod_{i=1}^{d} v^{s_i},\ \big\{C_{i,1} = g^{s_i},\ C_{i,2} = (u^{L|_i} h)^{s_i}\big\}_{i=1}^{d}\Big).$$

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

**CDE.DelegateCT($CH_L, c, PP$):** This algorithm takes as input a ciphertext header $CH_L = (C_0, C_1, \{C_{i,1}, C_{i,2}\})$ 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 $s_{d+1} \in \mathbb{Z}_p$ and outputs a delegated ciphertext header for a new label string $L' = L\|c$ as

$$CH_{L'} = \Big(C'_0 = C_0,\ C'_1 = C_1 \cdot v^{s_{d+1}},\ \big\{C'_{i,1} = C_{i,1},\ C'_{i,2} = C_{i,2}\big\}_{i=1}^{d},\ C'_{d+1,1} = g^{s_{d+1}},\ C'_{d+1,2} = (u^{L\|c} h)^{s_{d+1}}\Big).$$

**CDE.RandCT($CH_L, t', \vec{s}', PP$):** This algorithm takes as input a ciphertext header $CH_L = (C_0, C_1, \{C_{i,1}, C_{i,2}\})$, a random exponent $t' \in \mathbb{Z}_p$, an exponent vector $\vec{s}' = (s'_1, \ldots, s'_d) \in \mathbb{Z}_p^d$, and the public parameters $PP$. It outputs a re-randomized ciphertext header as

$$CH'_L = \Big(C'_0 = C_0 \cdot g^{t'},\ C'_1 = C_1 \cdot w^{t'} \prod_{i=1}^{d} v^{s'_i},\ \big\{C'_{i,1} = C_{i,1} \cdot g^{s'_i},\ C'_{i,2} = C_{i,2} \cdot (u^{L|_i} h)^{s'_i}\big\}_{i=1}^{d}\Big).$$

and a partial session key as $EK' = \Lambda^{t'}$ 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 $SK_{L'} = (K_0, K_1, \{K_{i,1}, K_{i,2}\}_{i=1}^n)$ for a label $L' \in \{0,1\}^n$, and the public parameters $PP$. If $L$ is a prefix of $L'$, then it derives $CH'_{L'} = (C'_0, C'_1, \{C'_{i,1}, C'_{i,2}\}_{i=1}^n)$ by iteratively running **DelegateCT** and outputs a session key as

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

Otherwise, it outputs $\perp$.

Let $\mathcal{BT}$ be a full binary tree with a depth $l$. For each node in $\mathcal{BT}$, a unique time is assigned by using the pre-order traversal. That is, the root node is assigned to 1 and the right most leaf node is assigned to $2^{l+1} - 1$. Let $\psi$ be a mapping from a time $T$ to a label $L$ by the pre-order traversal. We define $\textbf{TimeLabels}(L) = \{L\} \cup \textbf{RightSibling}(\textbf{Path}(L)) \setminus \textbf{Path}(\textbf{Parent}(L))$. Our SUE scheme is almost the same as that of Lee *et al.* [12] since a ciphertext with a label $L$ consists of CDE ciphertexts where each CDE ciphertext is associated with a label in $\textbf{TimeLabels}(L)$. Our SUE scheme that uses the above CDE scheme is described as follows:

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

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

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

**SUE.RandKey($SK_T, \delta, PP$):** This algorithm outputs $SK_T$ by running **CDE.RandKey**$(SK_T, \delta, PP)$.

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

1. It first sets a label $L \in \{0,1\}^d$ by computing $\psi(T)$. It sets an exponent vector $\vec{s} = (s_1, \dots, s_d)$ by selecting random exponents $s_1, \dots, s_d \in \mathbb{Z}_p$, and obtains $CH^{(0)} = (C_0, C_1, \{C_{i,1}, C_{i,2}\}_{i=1}^{d})$ by running **CDE.Encrypt**$(L, t, \vec{s}, PP)$.
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 needed.
   (b) Otherwise, it sets a new exponent vector $\vec{s}' = (s_1, \dots, s_{d-j}, s'_{d-j+1})$ where $s_1, \dots s_{d-j}$ are copied from $\vec{s}$ and $s'_{d-j+1}$ is randomly selected in $\mathbb{Z}_p$ since $L^{(j)}$ and $L$ have the same prefix string. It obtains $CH^{(j)} = (C'_0, C'_1, \{C'_{i,1}, C'_{i,2}\}_{i=1}^{d-j+1})$ by running **CDE.Encrypt**$(L^{(j)}, t, \vec{s}', PP)$. It also prunes redundant elements $C'_0, \{C'_{i,1}, C'_{i,2}\}_{i=1}^{d-j}$ 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 $d' \leq d$ 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 $EK = \Lambda^t$. Note that $CH^{(j)}$ are ordered according to pre-order traversal.

**SUE.UpdateCT($CH_T, T+1, PP$):** This algorithm takes as input a ciphertext header $CH_T = (CH^{(0)}, \dots, CH^{(d')})$ 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 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. Otherwise, it copies common elements in $CH^{(0)}$ to $CH^{(1)}$ and simply removes $CH^{(0)}$ since $CH^{(1)}$ is the ciphertext header for the next time $T+1$ by pre-order traversal. It outputs an updated ciphertext header as $CH_{T+1} = \big(CH'^{(0)} = CH^{(1)}, \dots, CH'^{(d'-1)} = CH^{(d')}\big)$.

**SUE.RandCT($CH_T, t', PP$):** This algorithm takes as input a ciphertext header $CH_T = (CH^{(0)}, \dots, CH^{(d')})$ for a time $T$, a new random exponent $t' \in \mathbb{Z}_p$, 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 $s'_1, \dots, s'_{d^{(0)}} \in \mathbb{Z}_p$, and obtains $CH'^{(0)}$ by running **CDE.RandCT**$(CH^{(0)}, t', \vec{s}', PP)$.
2. For $1 \leq j \leq d'$, it sets a new vector $\vec{s}'' = (s'_1, \dots, s'_{d^{(j)}-1}, s''_{d^{(j)}})$ where $s'_1, \dots s'_{d^{(j)}-1}$ are copied from $\vec{s}'$ and $s''_{d^{(j)}}$ is randomly chosen in $\mathbb{Z}_p$, and obtains $CH'^{(j)}$ by running **CDE.RandCT**$(CH^{(j)}, t', \vec{s}'', PP)$.
3. It outputs a re-randomized ciphertext header as $CH'_T = \big(CH'^{(0)}, \dots, CH'^{(d')}\big)$ and a partial session key as $EK' = \Lambda^{t'}$ 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 $T \leq T'$, 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 $\perp$.

## 3.3 Correctness

Let $SK_{L'}$ be a private key with a label $L'$ and $CH_L$ be a ciphertext header with a label $L$. If $L$ is a prefix of $L'$, then the ciphertext delegation algorithm can be used to derive a ciphertext header $CH_{L'}$. The correctness of CDE is satisfied by the following equation

$$\begin{aligned}
& e(C'_0, K_0) \cdot e(C'_1, K_1) \cdot \prod_{i=1}^{n} \big( e(C'_{i,1}, K_{i,1}) \cdot e(C'_{i,2}, K_{i,2}) \big) \\
&= e(g^t, g^\beta w^r) \cdot e(w^t \prod_{i=1}^{n} v^{s_i}, g^{-r}) \cdot \prod_{i=1}^{n} \big( e(g^{s_i}, v^r (u^{L|_i} h)^{r_i}) \cdot e((u^{L|_i} h)^{s_i}, g^{-r_i}) \big) \\
&= e(g^t, g^\beta) \cdot e(g^t, w^r) \cdot e(w^t, g^{-r}) \cdot e(\prod_{i=1}^{n} v^{s_i}, g^{-r}) \cdot \prod_{i=1}^{n} e(g^{s_i}, v^r) = e(g,g)^{\beta t}.
\end{aligned}$$

The correctness of SUE can be obtained from the property of the pre-order traversal and the correctness of CDE. Let $SK_{T'}$ be a private key with a time $T'$ and $CH_T$ be a ciphertext header with a time $T$. If $T \leq T'$, then there exists a CDE ciphertext $CH_{CDE,L}$ in the SUE ciphertext such that $L$ is a prefix of $L'$ where $L'$ is a label for the time $T'$ since **TimeLabels**$(L) \cap$ **Path**$(L') \neq \emptyset$ from the property of the pre-order traversal where $L$ is associated with $T$ and $L'$ is associated with $T'$. Therefore, a correct session key can be derived from the correctness of CDE by using the decryption algorithm of CDE since $L$ is a prefix of $L'$.

## 3.4 Security Analysis

To prove the security of our SUE scheme, we use the partitioning method that was widely used to prove other schemes. In SUE, a challenge ciphertext for the time $T^*$ is associated with labels in **TimeLabels**$(L^*)$ where $L^*$ is associated with $T^*$. In the security proof that uses the partitioning method, these labels should be programmed in short public parameters. To programming these labels in short public parameters, we use a $q$-type assumption. A $q$-type assumption was previously used for this purpose in [14, 21, 27]. The detailed security proof is described as follows:

**Theorem 3.1.** *The above SUE scheme is selectively secure under chosen plaintext attacks if the q-sRW1 assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that* $\mathbf{Adv}_{\mathcal{A}}^{SUE}(\lambda) \leq \frac{1}{2} \cdot \mathbf{Adv}_{\mathcal{B}}^{q\text{-}sRW}(\lambda)$.

*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 $q$-sRW assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \mathbb{G}_T, e), g, g^a, g^b, g^c, \{g^{d_j}, g^{cd_j}, g^{ad_j}, g^{b/d_j^2}, g^{a/d_j}\}_{\forall 1 \leq j \leq q}, \{g^{abd_j/d_{j'}^2}, g^{acd_j/d_{j'}}, g^{bcd_j/d_{j'}^2}\}_{\forall 1 \leq j, j' \leq q, j' \neq j})$ and $Z$ where $Z = Z_0 = e(g,g)^{abc}$ or $Z = Z_1 = e(g,g)^f$. 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 label string $L^*$ that is associated with $T^*$ by computing $L^* = \psi(T^*)$. Recall that **TimeLabels**$(L^*)$ is the set of label strings that consists of $L^*$ and the right sibling labels of path nodes of $L^*$ that are not in the parent's path. That is, **TimeLabels**$(L^*) = \{L^*\} \cup$ **RightSibling**$($**Path**$(L^*)) \setminus$ **Path**$($**Parent**$(L^*))$. We define **TL**$(L^*, j)$ be a function that returns the $j$th label string in **TimeLabels**$(L^*)$ where $l^*$ is the maximum number of label strings in **TimeLabels**$(L^*)$.

**Setup**: $\mathcal{B}$ first chooses random exponents $w', v', u', h' \in \mathbb{Z}_p$. It implicitly sets $\beta = ab$ and publishes the public parameters $PP$ as

$$g,\ w = g^{w'} g^{a},\ v = g^{v'} \prod_{j'=1}^{l^*} g^{a/d_{j'}},\ u = g^{u'} \prod_{j'=1}^{l^*} g^{b/d_{j'}^2},\ h = g^{h'} \prod_{j'=1}^{l^*} (g^{b/d_{j'}^2})^{-\mathbf{TL}(L^*,j')},\ \Lambda = e(g^a, g^b).$$

**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 random exponents $r', r'_1, \ldots, r'_n \in \mathbb{Z}_p$ and creates a private key by implicitly setting $r = -b + r'$, $\{r_i = \sum_{k=1}^{l^*} cd_k/(L|_i - \mathbf{TL}(L^*,k)) + r'_i\}_{i=1}^n$ as

$$\begin{aligned}
&K_0 = (g^b)^{-w'} w^{r'},\ K_1 = g^b g^{-r'},\\
&\Big\{K_{i,1} = (g^b)^{v'} v^{r'} \prod_{k=1}^{l^*} \big(g^{ad_k}\big)^{(u'L|_i + h')/(L|_i - \mathbf{TL}(L^*,k))}\\
&\qquad \cdot \prod_{j'=1}^{l^*} \prod_{k=1, k\neq j'}^{l^*} \big(g^{abd_k/d_{j'}^2}\big)^{(L|_i - \mathbf{TL}(L^*,j'))/(L|_i - \mathbf{TL}(L^*,k))} \big(u^{L|_i} h\big)^{r'_i},\\
&K_{i,2} = \prod_{k=1}^{l^*} \big(g^{ad_k}\big)^{-1/(L|_i - \mathbf{TL}(L^*,k))} g^{-r'_i}\Big\}_{i=1}^n.
\end{aligned}$$

Note that if $T < T^*$, then it can create a private key since $L|_i - \mathbf{TL}(L^*,k) \neq 0$ for all $1 \leq k \leq l^*$ from the fact that $\mathbf{Path}(L) \cap \mathbf{TimeLabels}(L^*) = \emptyset$ where $L = \psi(T)$.

**Challenge**: To create the challenge ciphertext for the 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 $s_1, \ldots, s_{d-1}, s'_d \in \mathbb{Z}_p$. Let $k$ be an index such that $L^* = \mathbf{TL}(L^*,k)$. It implicitly sets $s = c, s_d = -cd_k + s'_d$ and creates ciphertext components $CH^{(0)}$ as

$$\begin{aligned}
&C_0 = g^c,\ C_1 = (g^c)^{w'} \prod_{i=1}^{d-1} v^{s_i} \cdot \big(g^{cd_k}\big)^{-v'} \prod_{j'=1, j'\neq k}^{l^*} \big(g^{acd_k/d_{j'}}\big)^{-1} \cdot v^{s'_d},\ \big\{C_{i,1} = g^{s_i},\ C_{i,2} = (u^{L^*|_i} h)^{s_i}\big\}_{i=1}^{d-1},\\
&C_{d,1} = \big(g^{cd_k}\big)^{-1} g^{s'_d},\ C_{d,2} = \big(g^{cd_k}\big)^{-(u'L^* + h')} \prod_{j'=1, j\neq k}^{l^*} \big(g^{bcd_k/d_{j'}^2}\big)^{-(L^* - \mathbf{TL}(L^*,j'))} \cdot \big(u^{L^*} h\big)^{s'_d}.
\end{aligned}$$

2. For $1 \leq j \leq d$, it first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: Let $d^{(j)}$ be the length of $L^{(j)}$ and $k$ be an index such that $L^{(j)} = \mathbf{TL}(L^*,k)$. 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

$$\begin{aligned}
&C_1 = (g^c)^{w'} \prod_{i=1}^{d^{(j)}-1} v^{s_i} \cdot \big(g^{cd_k}\big)^{-v'} \prod_{j'=1, j'\neq k}^{l^*} \big(g^{acd_k/d_{j'}}\big)^{-1} \cdot v^{s'_{d^{(j)}}},\\
&C_{d^{(j)},1} = \big(g^{cd_k}\big)^{-1} g^{s'_{d^{(j)}}},\ C_{d^{(j)},2} = \big(g^{cd_k}\big)^{-(u'L^{(j)} + h')} \prod_{j'=1, j\neq k}^{l^*} \big(g^{bcd_k/d_{j'}^2}\big)^{-(L^{(j)} - \mathbf{TL}(L^*,j'))} \cdot \big(u^{L^{(j)}} h\big)^{s'_{d^{(j)}}}.
\end{aligned}$$

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 key $EK = Z$. It gives $CH_{T^*}$ and $EK$ to $\mathcal{A}$.

Note that it can create the challenge ciphertext for $T^*$ since $L^{(j)} \in \mathbf{TimeLabels}(L^*)$ for all label strings $L^{(j)}$ in the challenge ciphertext.

**Query 2**: Same as Query 1.

**Guess**: $\mathcal{A}$ outputs a guess $\mu'$. $\mathcal{B}$ also outputs $\mu'$.

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

$$\begin{aligned}
K_0 &= g^{\beta} w^r = g^{ab} (g^{w'} g^a)^{-b+r'} = (g^b)^{-w'} w^{r'}, \\
K_1 &= g^{-r} = g^{b-r'} = g^b g^{-r'}, \\
K_{i,1} &= v^r (u^{L|_i} h)^{r_i} = \Big(g^{v'} \prod_{j'=1}^{l^*} g^{a/d_{j'}}\Big)^{-b+r'} \Big(g^{u'L|_i+h'} \prod_{j'=1}^{l^*} g^{b(L|_i-\mathbf{TL}(L^*,j'))/d_{j'}^2}\Big)^{\sum_{k=1}^{l^*} ad_k/(L|_i-\mathbf{TL}(L^*,k))+r'_i} \\
&= (g^b)^{v'} v^{r'} \cdot \prod_{k=1}^{l^*} \big(g^{ad_k}\big)^{(u'L|_i+h')/(L|_i-\mathbf{TL}(L^*,k))} \\
&\quad \cdot \prod_{j'=1}^{l^*} \prod_{k=1,k\neq j'}^{l^*} \big(g^{abd_k/d_{j'}^2}\big)^{(L|_i-\mathbf{TL}(L^*,j'))/(L|_i-\mathbf{TL}(L^*,k))} \cdot \big(u^{L|_i} h\big)^{r'_i}, \\
K_{i,2} &= g^{-r_i} = g^{-\sum_{k=1}^{l^*} ad_k/(L|_i-\mathbf{TL}(L^*,k))-r'_i} = \prod_{k=1}^{l^*} \big(g^{ad_k}\big)^{-1/(L|_i-\mathbf{TL}(L^*,k))} g^{-r'_i}.
\end{aligned}$$

Note that the term $\prod_{j'=1}^{l^*} g^{ab/d_{j'}}$ of $K_{i,1}$ that is not given in the assumption is cancelled. The challenge ciphertext component $CH^{(j)}$ is also correctly distributed as

$$\begin{aligned}
C_1 &= w^t \prod_{i=1}^{d^{(j)}} v^{s_i} = \big(g^{w'} g^a\big)^c \prod_{i=1}^{d^{(j)}-1} v^{s_i} \Big(g^{v'} \prod_{j'=1}^{l^*} g^{a/d_{j'}}\Big)^{-cd_k+s'_{d^{(j)}}} \\
&= (g^c)^{w'} \prod_{i=1}^{d^{(j)}-1} v^{s_i} \cdot \big(g^{cd_k}\big)^{-v'} \prod_{j'=1,j'\neq k}^{l^*} \big(g^{acd_k/d_{j'}}\big)^{-1} \cdot v^{s'_{d^{(j)}}}, \\
C_{d^{(j)},1} &= g^{s_{d^{(j)}}} = g^{-cd_k+s'_{d^{(j)}}} = \big(g^{cd_k}\big)^{-1} g^{s'_{d^{(j)}}}, \\
C_{d^{(j)},2} &= \big(u^{L^{(j)}} h\big)^{s_{d^{(j)}}} = \Big(g^{u'L^{(j)}+h'} \prod_{j'=1}^{l^*} g^{b(L^{(j)}-\mathbf{TL}(L^*,j'))/d_j^2}\Big)^{-cd_k+s'_{d^{(j)}}} \\
&= \big(g^{cd_k}\big)^{-(u'L^{(j)}+h')} \prod_{j'=1,j'\neq k}^{l^*} \big(g^{bcd_k/d_{j'}^2}\big)^{-(L^{(j)}-\mathbf{TL}(L^*,j'))} \cdot \big(u^{L^{(j)}} h\big)^{s'_{d^{(j)}}}.
\end{aligned}$$

Note that the term $g^{ac}$ of $C_1$ is cancelled and the term $g^{bcd_k/d_k^2}$ of $C_{d^{(j)},2}$ is not needed since $L^{(j)} = \mathbf{TL}(L^*,k)$. This completes our proof. $\square$

## 3.5 Discussions

**Efficiency Analysis**. In our SUE scheme, the public parameters consist of 6 group elements, a private key consists of at most $2\log T_{max} + 2$ group elements, and a ciphertext header consists of at most $5\log T_{max} + 2$

group elements where $T_{max}$ is the maximum number of times. The decryption algorithm of SUE consists of at most $3 \log T_{max}$ exponentiations and $2 \log T_{max} + 2$ pairing operations.

**Full Model Security**. The security of our SUE scheme is proven in the selective model where an adversary should initially submit a challenge time before he receives public parameters. In the full security model, the adversary first receives public parameters and he submits the challenge time in the challenge step. If the maximum number of time $T_{max}$ is a polynomial value, then a selectively secure SUE scheme can be converted to a fully secure one with reduction loss $T_{max}$.

**Standard Assumption**. Our SUE scheme is secure under the $q$-sRW assumption where $q$ is dependent on the depth of a binary tree. This $q$-type assumption is a stronger one compared with the well-known standard assumption. Thus a SUE scheme that is secure under the the standard assumption is desired. If we increase the number of public parameters in our SUE scheme, then we can obtain a SUE scheme that is secure under the DBDH assumption. This SUE scheme is described in Appendix B. Note that this SUE scheme has $O(\log T_{max})$ number of group elements in public parameters, whereas the SUE scheme of Lee *et al.* [12] in prime-order groups has $O(T_{max})$ number of group elements in public parameters.

## 4 Self-Updatable Encryption for Time Intervals

In this section, we introduce the concept of time-interval SUE (TI-SUE) such that a ciphertext is associated with a time-interval and propose an efficient TI-SUE scheme by extending our SUE scheme. Note that the idea of TI-SUE was introduced by Lee *et al.* [12], but we give the formal definition and the security analysis of a TI-SUE scheme.

### 4.1 Definitions

Time-interval SUE (TI-SUE) is an interesting extension of SUE such that the time of a ciphertext can be specified by a time-interval. That is, the ciphertext of TI-SUE is associated with times $T_L$ and $T_R$ and a private key with a time $T$ such that $T_L \leq T \leq T_R$ can be used to decrypt this ciphertext. The formal syntax of TI-SUE is given as follows:

**Definition 4.1** (Time-Interval Self-Updatable Encryption, TI-SUE)**.** *A time-interval self-updatable encryption (TI-SUE) scheme consists of seven PPT algorithms **Init**, **Setup**, **GenKey**, **Encrypt**, **UpdateCT**, **RandCT**, and **Decrypt**, which are defined as follows:*

- ***Init**$(1^\lambda)$. The initialization algorithm takes as input a security parameter $1^\lambda$, 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 a master key MK and public parameters PP.*

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

- ***RandKey**$(SK_T, \delta, PP)$. The key randomization algorithm takes as input a private key $SK_T$, an exponent $\delta$, and the public parameters PP, and it outputs a re-randomized private key $SK'_T$.*

- ***Encrypt**$(T_L, T_R, PP)$. The encryption algorithm takes as input a left time $T_L$, a right time $T_R$, and the public parameters PP, and it outputs a ciphertext header $CH_{T_L,T_R}$ and a session key EK.*

***UpdateCT***$(CH_{T_L,T_R}, T'_L, T'_R, PP)$. *The ciphertext update algorithm takes as input a ciphertext header* $CH_{T_L,T_R}$*, a new left time* $T_L$*, a new right time* $T_R$*, and the public parameters* $PP$*, and it outputs an updated ciphertext header* $CH_{T'_L,T'_R}$.

***RandCT***$(CH_{T_L,T_R}, PP)$. *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***$(CH_{T_L,T_R}, SK_T, PP)$. *The decryption algorithm takes as input a ciphertext header* $CH_{T_L,T_R}$*, a private key* $SK_T$*, and the public parameters* $PP$*, and it outputs a session key* $EK$ *or the distinguished symbol* $\perp$.

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

- *If* $T_L \le T \le T_R$*, then* $\textbf{Decrypt}(CH_{T_L,T_R}, SK_T, PP) = EK$.
- *If* $(T < T_L) \vee (T_R < T)$*, then* $\textbf{Decrypt}(CH_{T_L,T_R}, SK_T, PP) = \perp$ *with all but negligible probability.*

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

The security of TI-SUE can be defined by following the security of SUE except that the challenge ciphertext is specified by a time interval. In this case, an adversary is allowed to request a polynomial number of private keys for times that are not in the challenge time interval. The formal definition of the selective security is given as follows:

**Definition 4.2** (Selective Security)**.** *The selective security of TI-SUE is defined in terms of the indistinguishability under chosen plaintext attacks (IND-CPA). The security game is defined as the following experiment between a challenger* $\mathcal{C}$ *and a PPT adversary* $\mathcal{A}$*:*

1. ***Init****:* $\mathcal{A}$ *initially submits challenge times* $T_L^*, T_R^*$.
2. ***Setup****:* $\mathcal{C}$ *generates a master key* $MK$ *and public parameters* $PP$ *by running* ***Init*** *and* ***Setup****, and it gives* $PP$ *to* $\mathcal{A}$.
3. ***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* $\textbf{GenKey}(T_i, MK, PP)$ *with the following restriction: For any time* $T_i$ *of private key queries, it is required that* $T_i < T_L^*$ *or* $T_R^* < T_i$.
4. ***Challenge****:* $\mathcal{C}$ *chooses a random bit* $\mu \in \{0,1\}$ *and computes a ciphertext header* $CH^*$ *and a session key* $EK^*$ *by running* $\textbf{Encrypt}(T_L^*, T_R^*, PP)$. *If* $\mu = 0$*, then it gives* $CH^*$ *and* $EK^*$ *to* $\mathcal{A}$. *Otherwise, it gives* $CH^*$ *and a random session key to* $\mathcal{A}$.
5. ***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* $\mathcal{A}$.
6. ***Guess****: Finally* $\mathcal{A}$ *outputs a bit* $\mu'$.

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

## 4.2 Construction

Before describing our TI-SUE scheme, we define a future SUE (F-SUE) scheme and a past SUE (P-SUE) scheme. F-SUE is the same as SUE that is defined in Section 2.2 such that the time $T$ of a ciphertext is updated to a future time $T+1$. P-SUE is similar to F-SUE except that the time $T$ of a ciphertext is updated to a past time $T-1$. A P-SUE scheme can be also constructed from a CDE scheme similarly to the F-SUE scheme if a time for a tree node is assigned in a decreasing order from the maximum time $T_{max}$ by following the pre-order traversal. The security model of P-SUE is also similarly defined and the security proof of P-SUE is easily obtained since the proof is almost the same as that of F-SUE except that the order of times is reversed.

The design idea of a TI-SUE scheme from SUE schemes is to combine an F-SUE scheme and a P-SUE scheme by sharing the master key to prevent collusion attacks. That is, if the ciphertext of F-SUE is associated with a left time $T_L$ and the ciphertext of P-SUE is associated with a right time $T_R$, then a private key with $T$ that satisfies $T_L \leq T \leq T_R$ only can be used to decrypt the ciphertext of F-SUE and that of P-SUE. This idea was presented by Lee *et al.* [12] without an actual scheme and the security proof.

Our TI-SUE scheme that uses F-SUE and P-SUE schemes is described as follows:

**TI-SUE.Init($1^\lambda$):** This algorithm outputs $GDS$ by running **SUE.Init**$(1^\lambda)$.

**TI-SUE.Setup($GDS, T_{max}$):** This algorithm first obtains $MK_{F\text{-}SUE}, PP_{F\text{-}SUE}$ and $MK_{P\text{-}SUE}, PP_{P\text{-}SUE}$ by running **F-SUE.Setup**$(GDS, T_{max})$ and **P-SUE.Setup**$(GDS, T_{max})$ respectively. It selects a random exponent $\beta \in \mathbb{Z}_p$ and outputs a master key $MK = \beta$ and public parameters $PP = \big(PP_{F\text{-}SUE}, PP_{P\text{-}SUE}, \Omega = e(g,g)^{\beta}\big)$.

**TI-SUE.GenKey($T, MK, PP$):** Let $MK = \beta$. It first selects a random exponent $\beta' \in \mathbb{Z}_p$ and obtains $SK_{F\text{-}SUE,T}$ and $SK_{P\text{-}SUE,T}$ by running **F-SUE.GenKey**$(T, \beta', PP_{F\text{-}SUE})$ and **P-SUE.GenKey**$(T, MK - \beta', PP_{P\text{-}SUE})$ respectively. It outputs a private key as $SK_T = \big(SK_{F\text{-}SUE,T}, SK_{P\text{-}SUE,T}\big)$.

**TI-SUE.RandKey($SK_T, \delta, PP$):** Let $SK_T = (SK_{F\text{-}SUE,T}, SK_{P\text{-}SUE,T})$. It selects a random exponent $\delta' \in \mathbb{Z}_p$ and obtains $SK'_{F\text{-}SUE,T}$ and $SK'_{P\text{-}SUE,T}$ by running **F-SUE.RandKey**$(SK_{F\text{-}SUE,T}, \delta', PP_{F\text{-}SUE})$ and **P-SUE.RandKey**$(SK_{P\text{-}SUE,T}, \delta - \delta', PP_{P\text{-}SUE})$ respectively. It outputs a re-randomized private key as $SK'_T = \big(SK'_{F\text{-}SUE,T}, SK'_{P\text{-}SUE,T}\big)$.

**TI-SUE.Encrypt($T_L, T_R, t, PP$):** It first obtains $CH_{F\text{-}SUE,T_L}, EK_{F\text{-}SUE}$ and $CH_{P\text{-}SUE,T_R}, EK_{P\text{-}SUE}$ by running **F-SUE.Encrypt**$(T_L, t, PP_{F\text{-}SUE})$ and **P-SUE.Encrypt**$(T_R, t, PP_{P\text{-}SUE})$ respectively. It outputs a ciphertext header as $CH_{T_L,T_R} = \big(CH_{F\text{-}SUE,T_L}, CH_{P\text{-}SUE,T_R}\big)$ an a session key as $EK = EK_{F\text{-}SUE} \cdot EK_{P\text{-}SUE}$.

**TI-SUE.UpdateCT($CH_{T_L,T_R}, T'_L, T'_R, PP$):** Let $CH_{T_L,T_R} = (CH_{F\text{-}SUE,T_L}, CH_{P\text{-}SUE,T_R})$. If $T_L < T'_L$, then it obtains $CH_{F\text{-}SUE,T'_L}$ by iteratively running **F-SUE.UpdateCT**$(CH_{F\text{-}SUE,T_L}, T_L+1, PP_{F\text{-}SUE})$ to the time $T'_L$; otherwise $(T_L = T'_L)$, it sets $CH_{F\text{-}SUE,T'_L} = CH_{F\text{-}SUE,T_L}$. If $T'_R < T_R$, then it obtains $CH_{P\text{-}SUE,T'_R}$ by iteratively running **P-SUE.UpdateCT**$(CH_{P\text{-}SUE,T_R}, T_R - 1, PP_{P\text{-}SUE})$ to the time $T'_R$; otherwise $(T'_R = T_R)$, it sets $CH_{P\text{-}SUE,T'_R} = CH_{P,T_R}$. It outputs an updated ciphertext header as $CH_{T'_L,T'_R} = \big(CH_{F\text{-}SUE,T'_L}, CH_{P\text{-}SUE,T'_R}\big)$.

**TI-SUE.RandCT($CH_{T_L,T_R}, t', PP$):** Let $CH_{T_L,T_R} = (CH_{F\text{-}SUE,T_L}, CH_{P\text{-}SUE,T_R})$. It obtains $CH'_{F\text{-}SUE,T_L}, EK'_{F\text{-}SUE}$ and $CH'_{P\text{-}SUE,T_R}, EK'_{P\text{-}SUE}$ by running **F-SUE.RandCT**$(CH_{F\text{-}SUE,T_L}, t', PP_{F\text{-}SUE})$ and **P-SUE.RandCT** $(CH_{P\text{-}SUE,T_R}, t', PP_{P\text{-}SUE})$ respectively. It outputs a re-randomized ciphertext header as $CH'_{T_L,T_R} = \big(CH'_{F\text{-}SUE,T_L}, CH'_{P\text{-}SUE,T_R}\big)$ and a partial session key as $EK' = EK'_{F\text{-}SUE} \cdot EK'_{P\text{-}SUE}$.

**TI-SUE.Decrypt**($CH_{T_L,T_R}, SK_T, PP$)**:** Let $CH_{T_L,T_R} = (CH_{F\text{-}SUE,T_L}, CH_{P\text{-}SUE,T_R})$ and $SK_T = (SK_{F\text{-}SUE,T}, SK_{P\text{-}SUE,T})$. If $T_L \leq T \leq T_R$, then it obtains $EK_{F\text{-}SUE}$ and $EK_{P\text{-}SUE}$ by running **F-SUE.Decrypt**($CH_{F\text{-}SUE,T_L}, SK_{F\text{-}SUE,T}, PP_{F\text{-}SUE}$) and **P-SUE.Decrypt**($CH_{P\text{-}SUE,T_R}, SK_{P\text{-}SUE,T}, PP_{P\text{-}SUE}$) respectively and outputs $EK = EK_{F\text{-}SUE} \cdot EK_{P\text{-}SUE}$. Otherwise, it outputs $\perp$.

## 4.3 Correctness

The correctness of TI-SUE is obtained from the correctness of F-SUE and P-SUE. Let $SK_T = (SK_{F-SUE,T}, SK_{P-SUE,T})$ be a private key with a time $T$ and $CH_{T_L,T_R} = (CH_{F-SUE,T_L}, CH_{P-SUE,T_R})$ be a ciphertext header with a time interval $[T_L, T_R]$. Note that $SK_{F-SUE,T}$ and $SK_{P-SUE}$ have $\beta'$ and $\beta - \beta'$ as a shared key respectively by using a simple addictive secret sharing scheme where $\beta$ is the master key. If $T_L \leq T \leq T_R$, then one partial session key $e(g,g)^{\beta' t}$ is correctly derived from the correctness of F-SUE and another partial session key $e(g,g)^{(\beta-\beta')t}$ is also derived from the correctness of P-SUE. The final session key $e(g,g)^{\beta t}$ is obtained by multiplying two partial session keys.

## 4.4 Security Analysis

To prove the security of our TI-SUE scheme, we can use the partitioning method as the same as the proof of our SUE scheme to show that private keys and the challenge ciphertext header are correctly simulated by using the $q$-sRW assumption. To simplify this security proof, we construct a meta-simulator that uses the simulators of F-SUE and P-SUE schemes in Theorem 3.1 as sub-simulators instead of constructing a simulator directly from the $q$-sRW assumption. This meta-simulator greatly simplifies the description of the security proof since private keys and the challenge ciphertext header can be generated by the sub-simulators. The detailed security proof is given as follows:

**Theorem 4.3.** *The above TI-SUE scheme is selectively secure under chosen plaintext attacks if the q-sRW assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that* $\boldsymbol{Adv}_{\mathcal{A}}^{TI\text{-}SUE}(\lambda) \leq \frac{1}{2} \cdot \boldsymbol{Adv}_{\mathcal{B}}^{q\text{-}sRW}(\lambda)$.

*Proof.* Suppose there exists an adversary $\mathcal{A}$ that attacks the above TI-SUE scheme with a non-negligible advantage. A meta-simulator $\mathcal{B}$ that solves the $q$-sRW assumption using $\mathcal{A}$ is given: a challenge tuple $D = ((p, \mathbb{G}, \mathbb{G}_T, e), g, g^a, g^b, g^c, \{g^{d_j}, g^{cd_j}, g^{ad_j}, g^{b/d_j^2}, g^{a/d_j}\}_{\forall 1 \leq j \leq q}, \{g^{abd_j/d_{j'}^2}, g^{acd_j/d_{j'}}, g^{bcd_j/d_{j'}^2}\}_{\forall 1 \leq j,j' \leq q, j' \neq j})$ and $Z$ where $Z = Z_0 = e(g,g)^{abc}$ or $Z = Z_1 = e(g,g)^f$. Let $\mathcal{B}_{F\text{-}SUE}$ be a simulator of F-SUE and $\mathcal{B}_{P\text{-}SUE}$ be a simulator of P-SUE. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ and internally runs two simulators $\mathcal{B}_{F\text{-}SUE}$ and $\mathcal{B}_{P\text{-}SUE}$ is described as follows:

**Init**: $\mathcal{A}$ initially submits challenge times $T_L^*, T_R^*$. $\mathcal{B}$ first runs $\mathcal{B}_{F\text{-}SUE}$ and $\mathcal{B}_{P\text{-}SUE}$ by giving the challenge tuple $D$ and $Z$.

**Setup**: $\mathcal{B}$ submits $T_L^*$ to $\mathcal{B}_{F\text{-}SUE}$ and receives $PP_{F\text{-}SUE}$, and it submits $T_R^*$ to $\mathcal{B}_{P\text{-}SUE}$ and receives $PP_{P\text{-}SUE}$. It implicitly sets $\beta = ab$ and publishes the public parameters $PP = \big(PP_{F\text{-}SUE}, PP_{P\text{-}SUE}, \Omega = e(g^a, g^b)\big)$.

**Query 1**: $\mathcal{A}$ adaptively request a private key for a time $T$ such that $T < T_L^*$ or $T_R^* < T$. $\mathcal{B}$ proceeds as follows:

- Case $T < T_L^*$: It first requests a private key for the time $T$ to $\mathcal{B}_{F\text{-}SUE}$ and obtains $SK'_{F\text{-}SUE,T}$. Next, it selects a random exponent $\beta' \in \mathbb{Z}_p$ and obtains $SK_{F\text{-}SUE,T}$ by running **F-SUE.RandKey**($SK'_{F\text{-}SUE,T}, -\beta', PP_{F\text{-}SUE}$). It also obtains $SK_{P\text{-}SUE,T}$ by running **P-SUE.GenKey**($T, \beta', PP_{P\text{-}SUE}$). It sets a private key $SK_T = \big(SK_{F\text{-}SUE,T}, SK_{P\text{-}SUE,T}\big)$.

- Case $T > T_R^*$: It first requests a private key for the time $T$ to $\mathcal{B}_{P\text{-}SUE}$ and obtains $SK'_{P\text{-}SUE,T}$. Next, it selects a random exponent $\beta' \in \mathbb{Z}_p$ and obtains $SK_{P\text{-}SUE,T}$ by running **P-SUE.RandKey**($SK'_{P\text{-}SUE,T}, -\beta'$,

$PP_{P\text{-}SUE})$. It also obtains $SK_{F\text{-}SUE,T}$ by running **F-SUE.GenKey**$(T, \beta', PP_{F\text{-}SUE})$. It sets a private key $SK_T = (SK_{F\text{-}SUE,T}, SK_{P\text{-}SUE,T})$.

**Challenge**: To create the challenge ciphertext for the challenge times $T_L^*, T_R^*$, $\mathcal{B}$ proceeds as follows: It first obtains $CH_{F\text{-}SUE,T_L^*}$ from $\mathcal{B}_{F\text{-}SUE}$ and obtains $CH_{P\text{-}SUE,T_R^*}$ from $\mathcal{B}_{P\text{-}SUE}$. It sets the challenge ciphertext header $CH_{T_L^*,T_R^*} = (CH_{F\text{-}SUE,T_L^*}, CH_{P\text{-}SUE,T_R^*})$ and the session key $EK = Z$.
**Query 2**: Same as Query 1.
**Guess**: $\mathcal{A}$ outputs a guess $\mu'$. $\mathcal{B}$ also outputs $\mu'$.

To finish the proof, we should show that the simulation is correct. The public parameters is correct since $PP_{F\text{-}SUE}$ and $PP_{P\text{-}SUE}$ share the same generator $g$ that is given in the assumption, $\mathcal{B}_{F\text{-}SUE}$ internally sets $\beta = ab$, and $\mathcal{B}_{P\text{-}SUE}$ internally sets $\beta = ab$. Now we show that private keys are correctly generated. In case of $T < T_L^*$, an F-SUE private key is generated with a master key $ab - \beta'$ is used by the help of $\mathcal{B}_{F\text{-}SUE}$ and a P-SUE private key is generated with a master key $\beta'$. In case of $T > T_R^*$, an F-SUE private key is generated with a master key $\beta'$ and a P-SUE private key is generated with a master key $ab - \beta'$ by the help of $\mathcal{B}_{P\text{-}SUE}$. Finally, we show that the challenge ciphertext header is correctly generated. The F-SUE challenge ciphertext header and the P-SUE challenge ciphertext header are correctly generated since the same element $g^c$ of the assumption is used in two simulators. If $Z = Z_0 = e(g,g)^{abc}$, then $EK$ is correctly distributed. Otherwise, $EK$ is a random element in $\mathbb{G}_T$. We omit the analysis of the probability. This completes our proof. $\square$

## 5 Revocable-Storage Attribute-Based Encryption

In this section, we propose an efficient RS-ABE scheme with short public parameters that supports the large universe of attributes, and prove its security under a $q$-type assumption.

### 5.1 Definitions

Revocable-storage attribute-based encryption (RS-ABE) is a new extension of ABE, introduced by Sahai *et al.* [22], that enhances the security of ciphertexts that are stored in cloud storage by providing user revocation and ciphertext updating functionalities. In RS-ABE, a ciphertext associated with an access structure $\mathbb{A}$ and a time $T$ is stored in cloud storage. A user has a private key with a set of attributes $S$ and he obtains an additional update key associated with an update time $T$ and a revoked user set $R$ from a center. If $S \in \mathbb{A}$ and the user is not revoked in $R$ at the time $T$, then he can decrypt the ciphertext in cloud storage by using his private key and update key. Additionally, the administrator of cloud storage can update the time $T$ of a ciphertext to a new time $T+1$ by using the public parameters to prevent a revoked users from accessing the past ciphertext. The formal syntax of RS-ABE is defined as follows:

**Definition 5.1** (Revocable-Storage Attribute-Based Encryption)**.** *A revocable-storage (ciphertext-policy) attribute-based encryption (RS-ABE) scheme for the universe of attributes $\mathcal{U}$ 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 $1^\lambda$, the maximum time $T_{max}$, and the maximum number of users $N_{max}$, and it outputs a master key $MK$ and public parameters $PP$.*

***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 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 key $MK$, and the public parameters $PP$, and it outputs an update key $UK_{T,R}$.*

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

***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 $CT_{\mathbb{A},T}$, 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 $CT_{\mathbb{A},T}$ and the public parameters $PP$, and it outputs a re-randomized ciphertext $CT'_{\mathbb{A},T}$.*

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

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

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

The security model of RS-ABE was defined by Sahai *et al.* [22]. We extend their security model to consider the decryption key exposure attacks, introduced by Seo and Emura [24], and define the selective revocation list model. In the security game of this model, an adversary initially submits a challenge access structure $\mathbb{A}^*$, a challenge time $T^*$, a set of revoked users $R^*$ at the time $T^*$. After that, he can adaptively request a private, an update, and decryption key that satisfy additional conditions. In the challenge step, he submits challenge messages $M_0^*, M_1^*$ and receives a challenge ciphertext $CT^*$ for $\mathbb{A}^*$, $T^*$, and $M_\mu^*$ where $\mu$ is a random coin. The adversary may request additional key queries and finally he outputs a guess. If the guess is correct, then the adversary wins the game. The formal security model is described as follows:

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

1. ***Init**: $\mathcal{A}$ first submits a challenge access structure $\mathbb{A}^*$, a challenge time $T^*$, and the set of revoked users $R^*$ at the time $T^*$.*
2. ***Setup**: $\mathcal{C}$ generates a master key $MK$ and public parameters $PP$ by running **Setup**$(1^\lambda, T_{max}, N_{max})$, and it gives $PP$ to $\mathcal{A}$.*

3. ***Query 1***: *$\mathcal{A}$ may adaptively request a polynomial number of private keys, update keys, and decryption 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 $\mathcal{A}$ is allowed to query only one private key for each user $u$.*
   - *If this is an update key query for a 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 $\mathcal{A}$ is allowed to query only one update key for each time $T$.*
   - *If this is a decryption key query for a set of attributes $S$ and a time $T$, then it gives the corresponding decryption key $DK_{S,T}$ to $\mathcal{A}$.*

   *We require restrictions on the queries of $\mathcal{A}$ as follows:*

   (a) *If an update key for $T$ and $R$ was queried, then $R \subseteq R_j$ for all update key queries on $T_j$ and $R_j$ such that $T < T_j$.*

   (b) *If a private key for $S$ and $u$ such that $S \in \mathbb{A}^*$ was queried, then an update key for $T_j$ and $R_j$ such that $u \in R_j$ and $T_j \leq T^*$ should be queried to revoke this user index $u$.*

   (c) *A decryption key for $S$ and $T$ such that $S \in \mathbb{A}^*$ and $T \geq T^*$ was not queried.*

4. ***Challenge***: *$\mathcal{A}$ submits challenge messages $M_0^*, M_1^* \in \mathcal{M}$ of equal length. $\mathcal{C}$ chooses a random bit $\mu \in \{0,1\}$ and gives a challenge ciphertext $CT^*$ to $\mathcal{A}$ by running* ***Encrypt***$(\mathbb{A}^*, T^*, M_\mu^*, PP)$.

5. ***Query 2***: *$\mathcal{A}$ may continue to request private keys, update keys, and decryption keys subject to the same restrictions as before, and $\mathcal{C}$ gives corresponding keys to $\mathcal{A}$.*

6. ***Guess***: *Finally $\mathcal{A}$ outputs a bit $\mu'$.*

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

## 5.2 Subset Cover Framework

The subset cover framework was introduced by Naor, Naor, and Lotspiech [17] as a general framework to construct a broadcast encryption scheme. The complete subset (CS) scheme is one instance of the subset cover framework, and we define this CS scheme as the same as that of Lee *et al.* [12] by excluding the key assigning part. The detailed description of the CS scheme is given 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 5.3** ( [17]). *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 $r \log(N_{max}/r)$.*

## 5.3 Construction

To construct an RS-ABE scheme with short public parameters, we combine the CP-ABE scheme of Rouselakis and Waters [21], our SUE scheme with short public parameters, and the CS scheme by following the design principle of Lee *et al.* [12]. To simplify the design and proof of our RS-ABE scheme, we additionally define private key randomization and ciphertext randomization algorithms for the CP-ABE scheme.

The key-encapsulation mechanism (KEM) version of the CP-ABE scheme of Rouselakis and Waters [21] is described as follows:

**ABE.Setup($GDS$):** This algorithm takes as input a group description string $GDS$. It chooses random elements $w_A, v_A, u_A, h_A \in \mathbb{G}$, and a random exponent $\gamma \in \mathbb{Z}_p$. It outputs the master key $MK = \gamma$ and the public parameters as $PP = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, w_A, v_A, u_A, h_A, \Lambda = e(g,g)^\gamma\big)$.

**ABE.GenKey($S, MK, PP$):** This algorithm takes as input a set of attributes $S = \{A_1, A_2, \ldots, A_k\}$, the master key $MK$, and the public parameters $PP$. It chooses random exponents $r, r_1, \ldots, r_n \in \mathbb{Z}_p$ and outputs a private key that implicitly includes $S$ as $SK_S = \big(K_0 = g^\gamma w_A^r, K_1 = g^{-r}, \{K_{i,1} = v_A^r (u_A^{A_i} h_A)^{r_i}, K_{i,2} = g^{-r_i}\}_{1 \le i \le n}\big)$.

**ABE.RandKey($SK_S, \delta, PP$):** This algorithm takes as input a private key $SK_S = (K_0, K_1, \{K_{i,1}, K_{i,2}\})$ for a set of attributes $S = \{A_1, A_2, \ldots, A_k\}$, an exponent $\delta \in \mathbb{Z}_p$, and the public parameters $PP$. It chooses random exponents $r', r'_1, \ldots, r'_n \in \mathbb{Z}_p$ and outputs a re-randomized private key as $SK_S = \big(K'_0 = K_0 \cdot g^\delta w_A^{r'}, K'_1 = K_1 \cdot g^{-r'}, \{K'_{i,1} = K_{i,1} \cdot v_A^{r'} (u_A^{A_i} h_A)^{r'_i}, K'_{i,2} = K_{i,2} \cdot g^{-r'_i}\}_{1 \le i \le n}\big)$.

**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 $\rho$ is a map from each row $A_j$ of $A$ to an attribute $\rho(j)$, a random exponent $s \in \mathbb{Z}_p$, and the public parameters $PP$. It first sets a random vector $\vec{v} = (s, v_2, \ldots, v_n)$ by selecting random exponents $v_2, \ldots, v_n \in \mathbb{Z}_p$. It selects random exponents $s_1, \ldots, s_l \in \mathbb{Z}_p$ and outputs a ciphertext header that implicitly includes $\mathbb{A}$ as $CH_\mathbb{A} = \big(C_0 = g^t, \{C_{j,1} = w_A^{\lambda_j} v_A^{s_j}, C_{j,2} = g^{s_j}, C_{j,3} = (u_A^{\rho(j)} h_A)^{s_j}\}_{1 \le j \le l}\big)$ 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 $s' \in \mathbb{Z}_p$, and the public parameters $PP$. It first sets a new vector $\vec{v}' = (s', v'_2, \ldots, v'_n)$ by selecting random exponents $v'_2, \ldots, v'_n \in \mathbb{Z}_p$. It selects random exponents $s'_1, \ldots, s'_l \in \mathbb{Z}_p$ and outputs a ciphertext header as $CH'_\mathbb{A} = \big(C'_0 = C_0 \cdot g^{s'}, \{C'_{1,j} = C_{1,j} \cdot g^{aA_j \cdot \vec{v}'} T_{\rho(j)}^{s'_j}, C'_{2,j} = C_{2,j} \cdot g^{-s'_j}\}_{1 \le j \le l}\big)$ 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 $S \in \mathbb{A}$, 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, our SUE scheme, and the CS scheme is described as follows:

**RS-ABE.Setup($1^\lambda, T_{max}, N_{max}$):** It first generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order $p$. Let $g$ be the generator of $\mathbb{G}$. It sets $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e), g)$. It obtains $MK_{ABE}, PP_{ABE}$ and $MK_{SUE}, PP_{SUE}$ by running **ABE.Setup**($GDS$) 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 $\mathcal{BT}$. It selects a random exponent $\alpha \in \mathbb{Z}_p$, 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($S, u, MK, PP$):** Let $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$. 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 $\gamma_{j_k}$ is assigned to the node $v_{j_k}$. For $0 \le k \le d$, it sets $MK'_{ABE} = (\gamma_{j_k}, Y)$ and obtains $SK_{ABE,k}$ by running **ABE.GenKey**($S, MK'_{ABE}, PP_{ABE}$). It outputs a private key as $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d}\big)$.

**RS-ABE.UpdateKey($T, R, MK, PP$):** 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 $\gamma_{i_k}$ is assigned to the node $v_{i_k}$. For $1 \le k \le m$, it sets $MK'_{SUE} = (\alpha - \gamma_{i_k}, Y)$ and obtains $SK_{SUE,k}$ by running **SUE.GenKey**($T, MK'_{SUE}, PP_{SUE}$). 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.DeriveKey($SK_{S,u}, UK_{T',R}, PP$):** Let $SK_{S,u} = (PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d})$ and $UK_{T',R} = (CV_R, SK_{SUE,1}, \ldots, SK_{SUE,m})$. If $u \notin R$, then it obtains $(S_i, S_j)$ by running **CS.Match**($CV_R, PV_u$). Otherwise, it outputs $\perp$. Next, it selects a random exponent $\delta \in \mathbb{Z}_p$ and obtains $SK_{ABE}$ and $SK_{SUE}$ by running **ABE.RandKey**($\delta, SK_{ABE,j}, PP_{ABE}$) and **SUE.RandKey**($-\delta, SK_{SUE,i}, PP_{SUE}$) respectively. It outputs a decryption key as $DK_{S,T'} = \big(SK_{ABE}, SK_{SUE}\big)$.

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

**RS-ABE.UpdateCT($CT_{\mathbb{A},T}, T+1, PP$):** Let $CT_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE}, C)$. It first obtains $CH'_{SUE}$ by running **SUE.UpdateCT**($CH_{SUE}, T+1, PP_{SUE}$). 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($CT_{\mathbb{A},T}, PP$):** Let $CT_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE}, C)$. It selects a random exponent $t' \in \mathbb{Z}_p$ and obtains $CH'_{ABE}$ and $CH'_{SUE}$ by running **ABE.RandCT**($CH_{ABE}, t', PP_{ABE}$) and **SUE.RandCT**($CH_{SUE}, t', PP_{SUE}$), respectively. It outputs a re-randomized ciphertext as $CT'_{\mathbb{A},T} = \big(CH'_{ABE}, CH'_{SUE}, C' = C \cdot \Omega^{s'}\big)$.

**RS-ABE.Decrypt($CT_{\mathbb{A},T}, DK_{S,T'}, PP$):** Let $CT_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE}, C)$ and $DK_{S,T'} = (SK_{ABE}, SK_{SUE})$. If $S \in \mathbb{A}$ and $T \le T'$, then it obtains $EK_{ABE}$ and $EK_{SUE}$ by running **ABE.Decrypt**($CH_{ABE}, SK_{ABE}, PP_{ABE}$)

and **SUE.Decrypt**$(CH_{SUE}, SK_{SUE}, PP_{SUE})$ respectively and outputs a message $M$ by computing $C \cdot \left(EK_{ABE} \cdot EK_{SUE}\right)^{-1}$. Otherwise, it outputs $\perp$.

## 5.4 Correctness

The correctness of RS-ABE can be obtained from the correctness of ABE, SUE, and CS schemes. Let $SK_{S,u}$ be a private key with a set of attributes $S$ and an index $u$ and $UK_{T',R}$ be an update key with an update time $T'$ and a set of revoked users $R$. If $u \notin R$, then the decryption key derivation algorithm can correctly derive a decryption key $DK_{S,T'} = (SK_{ABE,S}, SK_{SUE,T'})$ by the correctness of the CS scheme. Note that $SK_{ABE,S}$ has $\gamma_i + \delta$ as a master key and $SK_{SUE,T'}$ has $\alpha - \gamma_i - \delta$ as a master key. Let $CT_{\mathbb{A},T}$ be a ciphertext with an access structure $\mathbb{A}$ and a time $T$ and $DK_{S,T'}$ be a decryption key with a set of attributes $S$ and a time $T'$. If $S \in \mathbb{A}$ and $T \leq T'$, then one partial session key is derived from the correctness of the CP-ABE scheme and another partial session key also is derived from the correctness of the SUE scheme. Finally the session key is obtained by multiplying two partial session keys.

## 5.5 Security Analysis

To prove the security of our RS-ABE scheme, we cannot reduce the security of ABE or SUE to that of RS-ABE since our RS-ABE scheme uses a simple secret sharing scheme to share the master key to prevent collusion attacks instead of using ABE and SUE schemes as black-boxes. However, we can simplify the description of the security proof by constructing a meta-simulator that runs the simulators of ABE and SUE as sub-simulators. To construct a meta-simulator, we slightly modify the original simulator of the ABE scheme to meet additional conditions for the meta-simulation. The detailed description of the security proof is given as follows:

**Theorem 5.4** ( [21])**.** *The above ABE scheme is selectively secure under chosen plaintext attacks if the q-RW1 assumption holds.*

The original simulator of the above ABE scheme in [21] implicitly sets $\gamma = a^{q+1} + \gamma'$ by selecting a random exponent $\gamma' \in \mathbb{Z}_p$ and sets $w_A = g^a$ where $a$ and $q$ are set in the $q$-RW1 assumption. To use this simulator for the security proof of RS-ABE, we slightly modify this simulator to implicitly sets $\gamma = a^{q+1}$ and sets $w_A = g^a g^{w'}$ by selecting a random exponent $w' \in \mathbb{Z}_p$. The simulation of private keys and the challenge ciphertext also should be slightly modified, but this modification is easy. Note that this modified simulator sets $g$ of $PP$ as that in the assumption, implicitly sets $\gamma = a^{q+1}$, and sets $g^t$ of the challenge ciphertext header as $g^c$ of the assumption.

**Theorem 5.5.** *The above RS-ABE scheme is secure in the selective revocation list model under chosen plaintext attacks if the q-RW1 assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) \leq \boldsymbol{Adv}_{\mathcal{B}}^{q\text{-}RW1}(\lambda)$.*

*Proof.* Suppose there exists an adversary $\mathcal{A}$ that attacks the above RS-ABE scheme with a non-negligible advantage. A meta-simulator $\mathcal{B}$ that solves the $q$-RW1 assumption using $\mathcal{A}$ is given: a challenge tuple $D = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, g^c, \{g^{a^i}, g^{d_j}, g^{cd_j}, g^{a^i d_j}, g^{a^i/d_j^2}\}_{\forall 1 \leq i,j \leq q}, \{g^{a^i/d_j}\}_{\forall 1 \leq i \leq 2q, i \neq q+1, \forall 1 \leq j \leq q}, \{g^{a^i d_j / d_{j'}^2}\}_{\forall 1 \leq i \leq 2q, \forall 1 \leq j,j' \leq q, j' \neq j}, \{g^{a^i c d_j / d_{j'}}, g^{a^i c d_j / d_{j'}^2}\}_{\forall 1 \leq i,j,j' \leq q, j' \neq j}\big)$ and $Z$ where $Z = Z_0 = e(g,g)^{a^{q+1}c}$ or $Z = Z_1 = e(g,g)^f$. Note that a challenge tuple $D_{q\text{-}sRW}$ for the $q$-sRW assumption can be easily derived from the challenge tuple $D$ of the $q$-RW1 assumption by setting $b = a^q$. Let $B_{ABE}$ be a modified simulator in the security proof of Theorem 5.4 and

$B_{SUE}$ be a simulator in the security proof of Theorem 3.1. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

**Init**: $\mathcal{A}$ initially submits a challenge access structure $\mathbb{A}^*$, a challenge time $T^*$, and a set of revoked users $R^*$ at the time $T^*$. $\mathcal{B}$ first runs $\mathcal{B}_{ABE}$ by giving $D$ and $Z$, and it also runs $\mathcal{B}_{SUE}$ by giving $D_{q\text{-}sRW}$ and $Z$. 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**: $\mathcal{B}$ submits $\mathbb{A}^*$ to $\mathcal{B}_{ABE}$ and receives $PP_{ABE}$, and it submits $T^*$ to $\mathcal{B}_{SUE}$ and receives $PP_{SUE}$. It randomizes $\Lambda$ of $PP_{ABE}$ and $\Lambda$ of $PP_{SUE}$ by selecting random exponents $\gamma', \beta' \in \mathbb{Z}_p$. It implicitly sets $\alpha = a^{q+1}$ and gives the public parameters $PP = \big(PP_{ABE}, PP_{SUE}, \Omega = e(g^a, g^{a^q})\big)$ to $\mathcal{A}$.

**Query 1**: $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, and decryption key queries. If this is a private key query for a set of attributes $S$ and a user index $u$, then $\mathcal{B}$ proceeds as follows:

- **Case** $u \in R^*$: In this case, it can simply creates ABE private keys for path nodes if it uses $\gamma_i$ from $\mathcal{BT}$ for the master key of ABE.

  1. It first retrieves the leaf node $v_u \in RV^*$ that is assigned to the user $u$. 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 $\mathbf{Path}(v_u)$.
  2. For all $v_{j_k} \in \mathbf{Path}(v_u)$, it obtains $SK_{ABE,k}$ by running $\mathbf{ABE.GenKey}(S, \gamma_{j_k}, PP_{ABE})$
  3. It creates the private key $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d}\big)$.

- **Case** $u \notin R^*$: In this case, it can use $\mathcal{B}_{ABE}$ to generate ABE private keys since $\mathcal{A}$ can only request $S$ such that $S \notin \mathbb{A}^*$.

  1. It first queries an ABE private key for $S$ to $\mathcal{B}_{ABE}$ and receives $SK'_S$.
  2. Let $v_u$ be a leaf node in $\mathcal{BT}$ that is assigned to the user index $u$ such that $v_u \notin RV^*$. 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 $v_{j_k} \in \mathbf{RevTree}(RV^*) \cap \mathbf{Path}(v_u)$, it obtains $SK_{ABE,k}$ by running $\mathbf{ABE.GenKey}(S, \gamma_{j_k}, PP_{ABE})$.
  4. For all $v_{j_k} \in \mathbf{Path}(v_u) \setminus (\mathbf{RevTree}(RV^*) \cap \mathbf{Path}(v_u))$, it obtains $SK_{ABE,k}$ by running **ABE.RandKey** $(SK'_S, -\gamma_{j_k}, PP_{ABE})$.
  5. It creates the private key $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d}\big)$.

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, it can use $\mathcal{B}_{SUE}$ to generate SUE private keys since $\mathcal{A}$ can only request $T$ such that $T < T^*$.

  1. It first queries an SUE private key for $T$ to $\mathcal{B}_{SUE}$ and receives $SK'_{SUE}$.
  2. It obtains $CV_R$ 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 $v_{i_k} \in \mathbf{RevTree}(RV^*) \cap \mathbf{Cover}(R)$, it obtains $SK_{SUE,k}$ by running $\mathbf{SUE.RandKey}(SK'_{SUE}, -\gamma_{i_k}, PP_{SUE})$.
4. For all $v_{i_k} \in \mathbf{Cover}(R) \setminus (\mathbf{RevTree}(RV^*) \cap \mathbf{Cover}(R))$, it obtains $SK_{SUE,k}$ by running $\mathbf{SUE.GenKey}(T, \gamma_{i_k}, PP_{SUE})$.
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, it can simply create SUE private keys if it uses $\gamma_i$ 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 $CV_R$ 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 $CV_R$.
  2. For all $v_{i_k} \in \mathbf{Cover}(R)$, it obtains $SK_{SUE,k}$ by running $\mathbf{SUE.GenKey}(T, \gamma_{i_k}, PP_{SUE})$.
  3. It creates the update key $UK_{T,R} = (CV_R, SK_{SUE,1}, \ldots, SK_{SUE,m})$.

If this is a decryption key query for a set of attributes $S$ and a time $T$, then $\mathcal{B}$ proceeds as follows:

- **Case** $S \notin \mathbb{A}^*$: In this case, it can use $\mathcal{B}_{ABE}$ to generate an ABE private key since $S \notin \mathbb{A}^*$.

  1. It first queries an ABE private key for $S$ to $\mathcal{B}_{ABE}$ and receives $SK'_{ABE}$.
  2. It selects a random exponent $\delta \in \mathbb{Z}_p$ and obtains $SK_{ABE}$ and $SK_{SUE}$ by running $\mathbf{ABE.RandKey}(SK'_S, -\delta, PP_{ABE})$ and $\mathbf{SUE.GenKey}(T, \delta, PP_{SUE})$ respectively.
  3. It creates the decryption key $DK_{S,T} = \big(SK_{ABE}, SK_{SUE}\big)$.

- **Case** $S \in \mathbb{A}^*$: In this case, we have $T < T^*$ from the restriction of the security model. It uses $\mathcal{B}_{SUE}$ to generate an SUE private key since $T < T^*$.

  1. It first queries an SUE private key for $T$ to $\mathcal{B}_{SUE}$ and receives $SK'_{SUE}$.
  2. It selects a random exponent $\delta \in \mathbb{Z}_p$ and obtains $SK_{ABE}$ and $SK_{SUE}$ by running $\mathbf{ABE.GenKey}(S, \delta, PP_{ABE})$ and $\mathbf{SUE.RandKey}(SK'_{SUE}, -\delta, PP_{SUE})$ respectively.
  3. It creates the decryption key $DK_{S,T} = \big(SK_{ABE}, SK_{SUE}\big)$.

**Challenge**: $\mathcal{A}$ submits two challenge messages $M_0^*, M_1^*$. $\mathcal{B}$ queries an ABE challenge ciphertext header to $\mathcal{B}_{ABE}$ and receives $CH_{ABE}^*$. $\mathcal{B}$ also queries an SUE challenge ciphertext header to $\mathcal{B}_{SUE}$ and receives $CH_{SUE}^*$. Finally, it flips a random coin $\mu \in \{0,1\}$ and creates the challenger ciphertext $CT^* = (CH_{ABE}^*, CH_{SUE}^*, C^* = Z \cdot M_\mu^*)$ and gives it to $\mathcal{A}$.
**Query 2**: Same as Query 1.
**Guess**: $\mathcal{A}$ outputs a guess $\mu'$. If $\mu = \mu'$, then $\mathcal{B}$ outputs 0. Otherwise, it outputs 1.

To finish the proof, we should show that the simulation is correct. The public parameters is correct since $PP_{ABE}$ and $PP_{SUE}$ share the same generator $g$ that is given in the assumption, $\mathcal{B}_{ABE}$ internally sets $\gamma = a^{q+1}$, and $\mathcal{B}_{SUE}$ internally sets $\beta = a^{q+1}$. Now we show that private keys are correctly generated. In case of $u \in R^*$, an ABE private key with a node $v_{j_k} \in \mathbf{RevTree}(RV^*)$ is correctly generated with a master key $\gamma_{j_k}$ is used. In case of $u \notin R^*$, an ABE private key with a node $v_{j_k} \in \mathbf{RevTree}(RV^*)$ is generated with a master key $\gamma_{j_k}$ and an ABE private key with a node $v_{j_k} \notin \mathbf{RevTree}(RV^*)$ is generated with a master key $a^{q+1} - \gamma_{j_k}$ by the help of $\mathcal{B}_{ABE}$. Note that the master key assignment for each node is consistent in the simulation of private keys.

Next, we show that update keys are correctly generated. In case of $T < T^*$, an SUE private key with a node $v_{i_k} \in \mathbf{RevTree}(RV^*)$ is generated with a master key $a^{q+1} - \gamma_{i_k}$ by the help of $\mathcal{B}_{SUE}$ and an SUE private key with a node $v_{i_k} \notin \mathbf{RevTree}(RV^*)$ is generated with a master key $\gamma_{i_k}$. In case of $T \geq T^*$, an SUE private key with a node $v_{i_k} \notin \mathbf{RevTree}(RV^*)$ is generated with a master key $\gamma_{i_k}$ since $\mathbf{RevTree}(RV^*) \cap \mathbf{Cover}(R) = \emptyset$. Note that the master key assignment for each node in the simulation of update keys is consistent with that of private keys. We also show that decryption keys are correctly generated. In case of $S \notin \mathbb{A}^*$, an ABE private key is generated with a master key $a^{q+1} - \delta$ by the help of $\mathcal{B}_{ABE}$ and an SUE private key is generated with a master key $\delta$. In case of $(S \in \mathbb{A}^*) \wedge (T < T^*)$, an ABE private key is generated with a master key $\delta$ and an SUE private key is generated with a master key $a^{q+1} - \delta$ by the help of $\mathcal{B}_{SUE}$. Finally, we show that the challenge ciphertext is correctly generated. The ABE challenge ciphertext header and the SUE challenge ciphertext header are correctly generated since the same element $g^c$ of the assumption is used in two simulators. If $Z = Z_0 = e(g,g)^{a^{q+1}c}$, then $C^*$ is correctly distributed. Otherwise, $C^*$ is independent of $\mu$ since $Z_1$ is a random element in $\mathbb{G}_T$. We omit the analysis of the probability. This completes our proof. $\square$

### 5.6 Discussions

**Efficiency Analysis**. In our RS-ABE scheme, the public parameters consist of $O(1)$ group elements, a private key consists of $O(\log N_{max} * |S|)$ group elements, an update key consists of $O(r \log(N_{max}/r) * \log T_{max})$ group elements, a decryption key consists of $O(|S| + \log T_{max})$ group elements, and a ciphertext consists of $O(l + \log T_{max})$ group elements where $N_{max}$ is the maximum number users, $S$ is the set of attributes, $T_{max}$ is the maximum number of times, $r$ is the number of revoked users, and $l$ is the row size of an access structure. Compared with the efficient RS-ABE scheme of Lee *et al.* [12] such that the public parameters consist of $O(|\mathcal{U}| + \log T_{max})$ group elements where $\mathcal{U}$ is the universe of attributes, our RS-ABE scheme is very efficient since the public parameters consist of just $O(1)$ group elements.

**Supporting a Time Interval**. Our RS-ABE scheme does not support a time interval since it just uses the SUE scheme of Section 3. To construct an RS-ABE scheme that supports a time interval, we can combine the CP-ABE scheme, the TI-SUE scheme of Section 4, and the CS scheme. The security of this RS-ABE scheme can be proven under the $q$-RW1 assumption by using the similar meta-simulation technique of Theorem 5.5. We omit the detailed proof.

## 6 Conclusion

In this paper, we proposed the first SUE scheme with short public parameters in prime-order bilinear groups and proved its security under a $q$-type assumption. We also presented two extensions of our SUE scheme: a TI-SUE scheme with short public parameters that supports time intervals, and a large universe RS-ABE scheme with short public parameters.

There are many interesting problems that are left. The first one is to devise an SUE scheme with short public parameters under standard assumptions. We expect that a new proof technique is needed since our SUE scheme uses the power of a $q$-type assumption. One possible approach is to use the dual system encryption method of Waters [26]. The second one is to construct a large universe RS-ABE scheme with short public parameters that is fully secure. The RS-ABE scheme of Lee *et al.* [12] is fully secure, but the size of public parameters is not short and only a small universe of attributes is supported. The third one is to devise an RS-ABE scheme that uses the subset difference (SD) scheme for revocation instead of using the CS scheme. If the SD scheme can be used, then the size of update keys that should be broadcasted at each time period can be reduced. Recently, Lee *et al.* [13] proposed an RIBE scheme that uses the SD scheme, but they claimed that their technique cannot be directly applicable to construct an RS-ABE scheme.

## References

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

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

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

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

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

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

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

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

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

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

[11] 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.

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

[13] Kwangsu Lee, Dong Hoon Lee, and Jong Hwan Park. Efficient revocable identity-based encryption via subset difference methods. Cryptology ePrint Archive, Report 2014/132, 2014. `http://eprint.iacr.org/2014/132`.

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

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

[16] Timothy C. May. Timed-release crypto. Unpublished manuscript, 1993.

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

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

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

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

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

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

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

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

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

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

[27] 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, *PKC 2011*, volume 6571 of *Lecture Notes in Computer Science*, pages 53–70. Springer, 2011.

## A Definition of Ciphertext Delegatable Encryption

Ciphertext delegatable encryption (CDE) is a new type of PKE such that a ciphertext with a label $L$ can be delegated to a new ciphertext with a label $L'$ if $L$ is a prefix of $L'$. The concept of CDE was introduced by Lee *et al.* [12] to construct a SUE scheme. The formal syntax of CDE is described as follows:

**Definition A.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**($1^\lambda$). The initialization algorithm takes as input a security parameter $1^\lambda$, and it outputs a group description string GDS.*

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

***GenKey**($L, MK, PP$). The key generation algorithm takes as input a label string $L \in \{0,1\}^n$ with $n \leq l$, the master 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 $d \leq l$ 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 $d < l$, 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 $d < l$ 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 of CDE is defined as follows: For all $MK, PP$ generated by **Setup**, 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 model of CDE was introduced by Lee *et al.* [12] and we follow the selective security model version of their security definition. The security is defined as follows:

**Definition A.2** (Selective Security)**.** *The selective security of CDE is defined in terms of the indistinguishability under chosen plaintext attacks (IND-CPA). The security game is defined as the following experiment between a challenger $\mathcal{C}$ and a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$:*

1. ***Init**: $\mathcal{A}$ initially submits a challenge label string $L^*$.*
2. ***Setup**: $\mathcal{C}$ generates a master key $MK$ and public parameters $PP$ by running **Init** and **Setup**, and it gives $PP$ to $\mathcal{A}$.*
3. ***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**$(L_i, MK, PP)$ with the following restriction: For any label string $L_i$ of private key queries, it is required that $L^*$ is not a prefix of $L_i$.*
4. ***Challenge**: $\mathcal{C}$ 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 $\mathcal{A}$.*
5. ***Query 2**: $\mathcal{A}$ may continue to request private keys for additional label strings $L_{q'+1}, \ldots, L_q$ subject to the same restriction as before, and $\mathcal{C}$ gives the corresponding private keys to $\mathcal{A}$.*
6. ***Guess**: Finally $\mathcal{A}$ outputs a bit $b'$.*

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

## B Self-Updatable Encryption under Standard Assumptions

In this section, we propose an SUE scheme with shorter public parameters and prove its selective security under the standard assumption.

### B.1 Construction

To devise an SUE scheme under the standard assumption, we modify the SUE scheme in Section 3 by slightly increasing the number of group elements in public parameters. In the security proof of Section 3, we can program multiple challenge labels to short public parameters by the help of the $q$-type assumption, but we cannot use this programming technique in the standard assumption. However, the number of group elements in public parameters is just proportional to the depth of a binary tree since the challenge labels is just proportional to the depth of the tree. Note that this SUE scheme has shorter public parameters compared with the SUE scheme of Lee *et al.* [12] in prime-order bilinear groups.

Our CDE scheme is described as follows:

**CDE.Init**($1^\lambda$)**:** This algorithm takes as input a security parameter $1^\lambda$. 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**($GDS, l$)**:** This algorithm takes as input the string $GDS$ and the maximum length $l$ of label strings. It chooses random elements $w, v, \{u_{i,0}, u_{i,1}, h_{i,0}, h_{i,1}\}_{i=1}^{l} \in \mathbb{G}$ and a random exponent $\beta \in \mathbb{Z}_p$. 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 key $MK = \beta$ and the public parameters as

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

**CDE.GenKey**($L, MK, PP$)**:** This algorithm takes as input a label string $L \in \{0,1\}^n$ such that $n \leq l$, the master key $MK$, and the public parameters $PP$. It selects random exponents $r, r_1, \ldots, r_n \in \mathbb{Z}_p$ and outputs a private key that implicitly includes $L$ as

$$SK_L = \Big(K_0 = g^\beta w^r,\ K_1 = g^{-r},\ \big\{K_{i,1} = v^r F_{i,L[i]}(L|_i)^{r_i},\ K_{i,2} = g^{-r_i}\big\}_{i=1}^{n}\Big).$$

**CDE.Encrypt**($L, t, \vec{s}, PP$)**:** This algorithm takes as input a label string $L \in \{0,1\}^d$ such that $d \leq l$, a random exponent $t \in \mathbb{Z}_p$, 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 ciphertext header that implicitly includes $L$ as

$$CH_L = \Big(C_0 = g^t,\ C_1 = w^t \prod_{i=1}^{d} v^{s_i},\ \big\{C_{i,1} = g^{s_i},\ C_{i,2} = F_{i,L[i]}(L|_i)^{s_i}\big\}_{i=1}^{d}\Big).$$

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

**CDE.DelegateCT**($CH_L, c, PP$)**:** This algorithm takes as input a ciphertext header $CH_L = (C_0, C_1, \{C_{i,1}, C_{i,2}\})$ 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 $s_{d+1} \in \mathbb{Z}_p$ and outputs a delegated ciphertext header for a new label string $L' = L \| c$ as

$$CH_{L'} = \Big(C'_0 = C_0,\ C'_1 = C_1 \cdot v^{s_{d+1}},\ \big\{C'_{i,1} = C_{i,1},\ C'_{i,2} = C_{i,2}\big\}_{i=1}^{d},\ C'_{d+1,1} = g^{s_{d+1}},\ C'_{d+1,2} = F_{d+1,c}(L')^{s_{d+1}}\Big).$$

**CDE.RandCT**($CH_L, t', \vec{s}', PP$)**:** This algorithm takes as input a ciphertext header $CH_L = (C_0, C_1, \{C_{i,1}, C_{i,2}\})$ for a label string $L \in \{0,1\}^d$ such that $d \leq l$, a random exponent $t' \in \mathbb{Z}_p$, 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

$$CH'_L = \Big(C'_0 = C_0 \cdot g^{t'},\ C'_1 = C_1 \cdot w^{t'} \prod_{i=1}^{d} v^{s'_i},\ \big\{C'_{i,1} = C_{i,1} \cdot g^{s'_i},\ C'_{i,2} = C_{i,2} \cdot F_{i,L[i]}(L|_i)^{s'_i}\big\}_{i=1}^{d}\Big).$$

and a partial session key as $EK' = \Lambda^{t'}$ 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 string $L \in \{0,1\}^d$, a private key $SK_{L'} = (K_0, K_1, \{K_{i,1}, K_{i,2}\}_{i=1}^{n})$ 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 derives $CH'_{L'} = (C'_0, C'_1, \{C'_{i,1}, C'_{i,2}\}_{i=1}^{n})$ by iteratively running **DelegateCT** and outputs a session key as

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

Otherwise, it outputs $\perp$.

The description of our SUE scheme is almost the same as that of Section 3. We omit the description of the SUE scheme.

## B.2 Security Analysis

To prove the security of above SUE scheme, we use the partitioning method. In the preparation of public parameters, a simulator can program only one challenge label to one element. The detailed description of the security proof is given as follows:

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

*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 label $L^*$ that is associated with the challenge time $T^*$ by computing $L^* = \psi(T^*)$. Recall that $\mathbf{TimeLabels}(L) = \{L\} \cup \mathbf{RightSibling}(\mathbf{Path}(L)) \setminus \mathbf{Path}(\mathbf{Parent}(L))$. We define $\mathbf{TL}(L^*, i, j)$ be a function that returns a label string $L$ in $\mathbf{TimeLabels}(L^*)$ such that the length of $L$ is $i$ and $L[i] = j$. Note that $\mathbf{TL}(L^*, i, j)$ return 0 if there is no label string for $i$ and $j$.

**Setup**: $\mathcal{B}$ first chooses random exponents $w', v', \{u'_{i,j}, h'_{i,j}\}_{\forall 1 \leq i \leq l, \forall j \in \{0,1\}} \in \mathbb{Z}_p$. It implicitly sets $\beta = ab$ and publishes the public parameters $PP$ as

$$g,\ w = g^a g^{w'},\ v = g^a g^{v'},\ \big\{u_{i,j} = g^a g^{u'_{i,j}},\ h_{i,j} = (g^a)^{-\mathbf{TL}(L^*,i,j)} g^{h'_{i,j}}\big\}_{\forall 1 \leq i \leq l, \forall j \in \{0,1\}},\ \Lambda = e(g^a, g^b).$$

**Query 1**: $\mathcal{A}$ adaptively request a private key for a time $T$ such that $T < T^*$. $\mathcal{B}$ first obtains a label string $L \in \{0,1\}^n$ by computing $\psi(T)$. Next, it selects random exponent $r', r'_1, \ldots, r'_n \in \mathbb{Z}_p$ and creates a private key by implicitly setting $r = -b + r'$, $\{r_i = b/(L|_i - \mathbf{TL}(L^*, i, L[i])) + r'_i\}_{i=1}^n$ as

$$\begin{aligned}
&K_0 = (g^b)^{-w'} w^{r'},\ K_1 = g^b g^{-r'},\\
&\big\{K_{i,1} = (g^b)^{-v'} v^{r'} (g^b)^{(u'_{i,L[i]} L|_i + h'_{i,L[i]})/(L|_i - \mathbf{TL}(L^*, i, L[i]))} F_{i,L[i]}(L|_i)^{r'_i},\\
&\quad K_{i,2} = (g^b)^{-1/(L|_i - \mathbf{TL}(L^*, i, L[i]))} g^{-r'_i}\big\}_{i=1}^n.
\end{aligned}$$

Note that if $T < T^*$, then it can create a private key since $\mathbf{Path}(L) \cap \mathbf{TimeLabels}(L^*) = \emptyset$ where $L = \psi(T)$.

**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 $s_1, \ldots, s_{d-1}, s'_d \in \mathbb{Z}_p$. It implicitly sets $t = c$, $s_d = -c + s'_d$ and creates ciphertext components $CH^{(0)}$ as

$$\begin{aligned}
&C_0 = g^c,\ C_1 = (g^c)^{w'} \prod_{i=1}^{d-1} v^{s_i} (g^c)^{-v'} v^{s'_d},\ \big\{C_{i,1} = g^{s_i},\ C_{i,2} = F_{i,L^*[i]}(L^*|_i)^{s_i}\big\}_{i=1}^{d-1},\\
&C_{d,1} = (g^c)^{-1} g^{s'_d},\ C_{d,2} = (g^c)^{-(u'_{i,L^*[i]} L^*|_i + h'_{i,L^*[i]})} F_{d,L^*[d]}(L^*|_i)^{s'_d}.
\end{aligned}$$

2. For $1 \leq j \leq d$, it first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: Let $d^{(j)}$ be the length of $L^{(j)}$. 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

$$C_1 = (g^c)^{w'} \prod_{i=1}^{d^{(j)}-1} v^{s_i} (g^c)^{-v'} v^{s'_{d^{(j)}}},$$
$$C_{d^{(j)},1} = (g^c)^{-1} g^{s'_{d^{(j)}}},\; C_{d^{(j)},2} = (g^c)^{-(u'_{d^{(j)},L[d^{(j)}]} L^{(j)} + h'_{d^{(j)},L[d^{(j)}]})} F_{d^{(j)},L^{(j)}}(L^{(j)})^{s'_{d^{(j)}}}.$$

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, $L^{(j)} \in \mathbf{TimeLabels}(L^*)$.
**Query 2**: Same as Query 1.
**Guess**: $\mathcal{A}$ outputs a guess $\mu'$. $\mathcal{B}$ also outputs $\mu'$.

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

$$\begin{aligned}
K_0 &= g^{\beta} w^r = g^{ab} (g^a g^{w'})^{-b+r'} = (g^b)^{-w'} w^{r'},\\
K_1 &= g^{-r} = g^{b-r'} = g^b g^{-r'},\\
K_{i,1} &= v^r F_{i,L[i]}(L|_i)^{r_i} = (g^a g^{v'})^{-b+r'} \big((g^a)^{L|_i - \mathbf{TL}(L^*,i,L[i])} g^{u'_{i,L[i]} L|_i + h'_{i,L[i]}}\big)^{b/(L|_i - \mathbf{TL}(L^*,i,L[i])) + r'_i}\\
&= (g^b)^{-v'} v^{r'} (g^b)^{(u'_{i,L[i]} L|_i + h'_{i,L[i]})/(L|_i - \mathbf{TL}(L^*,i,L[i]))} F_{i,L[i]}(L|_i)^{r'_i},\\
K_{i,2} &= g^{-r_i} = g^{-b/(L|_i - \mathbf{TL}(L^*,i,L[i])) - r'_i} = (g^b)^{-1/(L|_i - \mathbf{TL}(L^*,i,L[i]))} g^{-r'_i}.
\end{aligned}$$

Note that the term $g^{ab}$ of $K_{i,1}$ that is not given in the assumption is cancelled since $L|_i - \mathbf{TL}(L^*, i, L[i]) \neq 0$. The challenge ciphertext component $CH^{(j)}$ is also correctly distributed as

$$\begin{aligned}
C_1 &= w^t \prod_{i=1}^{d^{(j)}} v^{s_i} = (g^a g^{w'})^c \prod_{i=1}^{d^{(j)}-1} v^{s_i} \cdot (g^a g^{v'})^{-c+s'_{d^{(j)}}} = (g^c)^{w'} \prod_{i=1}^{d^{(j)}-1} v^{s_i} \cdot (g^c)^{-v'} v^{s'_{d^{(j)}}},\\
C_{d^{(j)},1} &= g^{s_{d^{(j)}}} = g^{-c+s'_{d^{(j)}}} = (g^c)^{-1} g^{s'_{d^{(j)}}},\\
C_{d^{(j)},2} &= F_{d^{(j)},L^*[d^{(j)}]}(L^{(j)})^{s_{d^{(j)}}} = \big((g^a)^{L^{(j)} - \mathbf{TL}(L^*,d^{(j)},L[d^{(j)}])} g^{u'_{d^{(j)},L[d^{(j)}]} L^{(j)} + h'_{d^{(j)},L[d^{(j)}]}}\big)^{-c+s'_{d^{(j)}}}\\
&= (g^c)^{-(u'_{d^{(j)},L[d^{(j)}]} L^{(j)} + h'_{d^{(j)},L[d^{(j)}]})} F_{d^{(j)},L^{(j)}}(L^{(j)})^{s'_{d^{(j)}}}.
\end{aligned}$$

Note that the term $g^{ac}$ of $C_1$ is cancelled and the term $g^{ac}$ of $C_{d^{(j)},2}$ is not needed since $L^{(j)} = \mathbf{TL}(L^*, d^{(j)}, L[d^{(j)}])$. This completes our proof. □