CCA Security for Self-Updatable Encryption: Protecting Cloud Data When Clients Read/Write Ciphertexts

Kwangsu Lee* Dong Hoon Lee† Jong Hwan Park‡ Moti Yung§

Abstract

Self-updatable encryption (SUE) is a new kind of public-key encryption, motivated by cloud computing, which enables anyone (i.e. cloud server with no access to private keys) to update a past ciphertext to a future ciphertext by using a public key. The main applications of SUE is revocable-storage attribute-based encryption (RS-ABE) that provides an efficient and secure access control to encrypted data stored in cloud storage. In this setting, there is a new threat such that a revoked user still can access past ciphertexts given to him by a storage server. RS-ABE solves this problem by combining user revocation and ciphertext updating functionalities. The mechanism was designed with semantic security (CPA).

We have noticed, however, that when anyone can contribute ciphertexts (as this is a public key setting), and when clients have access to some ciphertexts (encrypted data) at storage servers (since we do not exclude this possibility), then, when in order to retrieve plaintexts they employ decryption service (i.e., probe crypto servers in the cloud), this service may be sensitive to Chosen Ciphertext Attacks (CCA) when the adversary plays as a client. Next notice that when considering CCA, the RS-ABE functionality, by definition, allows certain malleability, namely, updating of messages by anyone (e.g., storage servers) over time. This seems, at first, anathema to this security notion, and this has to be dealt with!

Here, we propose the first SUE and RS-ABE schemes, secure against a relevant form of CCA, which allows ciphertexts submitted by attackers to decryption servers. Due to the fact that some ciphertexts are easily derived from others, we employ a different notion of CCA which avoids easy challenge related messages (we note that this type of idea was employed in other contexts before). Specifically, we define "time extended challenge" (TEC) CCA security for SUE which excludes ciphertexts that are easily derived from the challenge (over time periods) from being queried on (namely, once a challenge is decided by an adversary, no easy modification of this challenge to future and past time periods is allowed to be queried upon). We then propose an efficient SUE scheme with such CCA security, and we also define similar CCA security for RS-ABE and present an RS-ABE scheme with this CCA security.

Keywords: Public-key encryption, Self-updatable encryption, Chosen-ciphertext security, Cloud storage.

*Korea University, Korea. Email: kwangsu.lee@korea.ac.kr.
†Korea University, Korea. Email: donghlee@korea.ac.kr.
‡Sangmyung University, Korea. Email: jhpark@smu.ac.kr.
§Google Inc. and Columbia University, USA. Email: moti@cs.columbia.edu.

1 Introduction

In cloud storage, providing an efficient access control to encrypted data is very important issue, since it extends traditional data compartmentalization in organizations and services to the Internet based hosting paradigm. To provide such access control to encrypted data stored in cloud storage, revocable-storage attribute-based encryption (RS-ABE) was introduced by Sahai, Seyalioglu, and Waters [23]. Sensitive data may be kept in cloud storage which is controlled by some servers which do not have access to keys nor are necessarily fully trusted (and may be accessible at times to some clients). Then, there could be potential threats that are new to this setting, as nicely pointed out by Sahai et al. [23]. That is, a user who is revoked from cloud storage can still access old (before revocation) ciphertexts stored in cloud storage (e.g., an rachive), even if he cannot access ciphertexts encrypted after revocation. RS-ABE, which is a kind of ABE [12, 22, 24], can solve this problem by providing two functionalities: user revocation and ciphertext updating.

Self-updatable encryption (SUE) is public-key encryption (PKE) that allows anyone (and storage servers in particular) to update a past ciphertext to a future ciphertext by using public keys (or public parameters). Lee, Choi, Lee, Park, and Yung [14], building on [23], introduced the concept of SUE, and they showed that an efficient RS-ABE scheme can be built from an SUE scheme and an ABE scheme. That is, the initial RS-ABE scheme of Sahai et al. [23] contains $O(\log^2 T_{max})$ group elements in a ciphertext to support ciphertext updating where T_{max} is the maximum number of time units, while the modularly built RS-ABE scheme of Lee et al. [14] just contains $O(\log T_{max})$ group elements in a ciphertext to support ciphertext updating (due to the use of an SUE scheme). As pointed in [14], SUE can also be used to build time-released encryption (TIE) and key-insulated encryption (KIE) with certain properties.

The available RS-ABE schemes and SUE schemes only provide security against chosen-plaintext attacks (CPA security) [10, 13, 14, 23]. CPA is strong enough when assuming storage access is performed and decrypted by key holders. However, if storage access and decryption service are separated, encrypted data can be presented to a decryption server (with various capabilities and different revocation status). In this case the decryption keys need extra protection against adversaries who present (possibly arbitrary) ciphertexts. To deal with stronger attackers in this case, we should consider security against chosen-ciphertext attacks (CCA security), in which an adversary can adaptively request decryption queries on ciphertexts. CCA security for PKE and its extensions was intensively studied by many researchers and is a standard requirement by now e.g., [1, 8, 9, 18, 20, 27] and there are general methods to achieve it [4, 6]. However, constructing a CCA-secure RS-ABE scheme (or CCA-secure SUE scheme) is paradoxical by definition, since CCA means non-malleability, but these schemes need to support ciphertext updating functionality over time periods (and an attacker can update the ciphertext and query on it without changing the cleartext!), thus it seems an unusually hard requirement in this case. Here we, nevertheless, ask whether and to what extent it is possible to construct some (properly adapted) level of CCA-security for SUE and RS-ABE schemes.

1.1 Our Results

SUE with CCA Security. We first define time extended challenge (TEC) CCA security for SUE schemes allowing the ciphertext updating functionality. In a standard CCA security model given a challenge ciphertext the adversary is not allowed to probe on it when accessing the decryption oracle, we extend the restriction not allowing the adversary to ask a decryption query on any ciphertext that is the challenge or updated from the challenge ciphertext (this is a natural extension preventing the trivial attack due to the probing capabilities, but leaving the non trivial attack scenario in place; it is also in the spirit of similar limitations elsewhere). Then we propose an efficient SUE scheme and prove its TEC-CCA security under the decisional Bilinear Diffie-Hellman (DBDH) assumption. The design idea of our SUE scheme is given in the later part of this section. Note that another natural restriction can allow queries after the challenge phase to belong only to the time unit of the challenge; this however is more restricting and is a case covered by our definition.

RS-ABE with CCA Security. RS-ABE schemes support self updating as well, so for the above reasons we, similarly, define TEC-CCA security for them. We then propose an efficient RS-ABE scheme by combining a TEC-CCA secure SUE scheme, a CCA-secure ciphertext-policy ABE (CP-ABE) scheme, and the complete subtree (CS) scheme. We prove that our RS-ABE scheme is TEC-CCA secure in a selective security model described next. Our RS-ABE scheme is the first one that achieves security beyond CPA, and gives an answer to the question raised by Sahai et al. [23].

Proving Selective Security. The selective revocation list model, introduced by Boldyreva et al. [2], was used in proving the security of revocable ABE (R-ABE) and RS-ABE if the partitioning technique is employed in the proof. Note that the selective revocation list model is weaker than the well-known selective model that is used for the security proof of identity-based encryption (IBE) [3, 5] and attribute-based encryption (ABE) [12, 28]. In the security proof of our RS-ABE scheme, we show that the TEC-CCA security of our RS-ABE scheme can be proven in the selective TEC-CCA model instead of the selective revocation list TEC-CCA model although we use the partitioning technique. We note that as a result of independent interest, our new proof technique can also be used to prove the selective CPA security of the R-ABE scheme of Boldyreva et al. [2] instead of the selective revocation list CPA security.

1.2 Our Techniques

A first naive approach to building a CCA-secure SUE scheme is to use the CHK transformation of Canetti et al. [6]. That is, a CPA-secure SUE scheme augmented by the IBE scheme of Boneh and Boyen [3] can be converted to a CCA-secure one by adding a one-time signature to provide the ciphertext integrity. However, this CCA-secure SUE scheme cannot provide the ciphertext updating functionality any longer. To solve this problem, we divide the ciphertext components into two parts: one part is related to a session key and another part is related to ciphertext updating. Thus we apply the CHK transformation to the ciphertext component related to a session key. To check the validity of ciphertext components related to ciphertext updating, we observe that the well-formedness of these components can be checked by bilinear maps since these components consist of Diffie-Hellman (DH) tuples. By using these two techniques, we can build an SUE scheme with TEC-CCA security.

As mentioned before, we combine a TEC-CCA secure SUE scheme, a CCA-secure CP-ABE scheme, and the CS scheme to build a TEC-CCA secure RS-ABE scheme motivated by the design principle of Lee et al. [14]. To prove the security of our RS-ABE scheme, we use the well-known partitioning method. However, we need the selective revocation list model to prove the security of RS-ABE as pointed by Lee [13]. The reason is that an adversary can request many private key queries that match to the challenge ciphertext in RS-ABE and these (matching) private keys should be placed on (fixed) leaf nodes in a binary tree for consistency. The selective revocation list model, introduced by Boldyreva et al. [2], is weaker than the well-known selective model [5]. To prove the security of RS-ABE in the selective model instead of the selective revocation list model, we observe that if a simulator can predict the number of private key queries that match the challenge ciphertext then the problem can be solved by placing a user's private key in a random leaf node of the binary tree. That is, if $\tilde{q}$ is the number of private keys such as $S \in \mathbb{A}^*$ where S is the set of attributes in a private key and $\mathbb{A}^*$ is the access structure in the challenge ciphertext, then the simulator can fix the positions of these private keys by arbitrary selecting $\tilde{q}$ number of leaf nodes randomly. In this case, the consistency of private keys and update keys is preserved.

1.3 Related Work

Self-Updatable Encryption. Lee et al. [14] introduced the notion of SUE and proposed an efficient SUE scheme with CPA security by reversing the structure of a private key and a ciphertext of hierarchical identity-based encryption (HIBE) [3] combining with the design idea of forward-secure encryption (FSE) [5]. Then, a number of different SUE schemes were proposed in [10, 13]. The main application of SUE is RS-ABE for cloud storage [14]. However, SUE itself can be used for other interesting applications: timed-release encryption [21] and key-insulated encryption [11].

Revocable IBE and Its Extensions. Providing an efficient user revocation in identity-based encryption (IBE) and attribute-based encryption (ABE) is very important issue in real applications. A scalable and efficient revocable IBE (R-IBE) scheme was first proposed by Boldyreva et al. [2] by using a full binary tree. After that numerous R-IBE schemes were presented in [15, 16, 19, 25, 26]. An efficient revocable ABE (R-ABE) scheme also proposed in [2] and its security was claimed in the weaker selective revocation list model. Sahai et al. [23] introduced the concept of RS-ABE to provide efficient access control on encrypted data stored in cloud storage. After the introduction of RS-ABE, an efficient RS-ABE schemes with CPA security were presented in [13, 14].

Chosen-Ciphertext Security. Security against (adaptively) chosen ciphertext attacks (or CCA security) is the standard notion of PKE [20]. For some applications, more adversary constrained notions of CCA security are considered since CCA security is too strong and immediately un-achievable. Such constrained adversary notions have been used before: Shoup introduced benign malleability [27], An et al. introduced generalized CCA security (or gCCA security) [1], and Canetti et al. introduced Replayable CCA security (or RCCA security) [8]. Note that benign malleability, gCCA security, and publicly detectable RCCA security are in fact the same notion. This (more adversarially constrained) CCA security was used to prove the security of other cryptographic primitives [7, 29].

2 Preliminaries

In this section, we define full binary trees and bilinear groups, and then introduce complexity assumptions in bilinear groups.

2.1 Full Binary Tree

For binary trees, we follow the notation in [14]. 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 $1 \leq i \leq 2N-1$, we denote by v_i a node in $\mathcal{BT}$. 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 maximum 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 each node $v_i \in \mathcal{BT}$, we associated v_i with a unique label string $L \in \{0,1\}^*$. The label of each node 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. We assign a special empty string to the root node label. We define $L(i)$ be a mapping from the index i of a node v_i to a label L. We also use $L(v_i)$ as $L(i)$ if there is no ambiguity. For a label string $L \in \{0,1\}^n$, we define some notations: $L[i]$ is the ith bit of L, $L|_i$ is the prefix of L with i-bit length, and $L\|L'$ is the concatenation of two strings L and L'.

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

2.2 Bilinear Groups

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

1. Bilinearity: for all $u, v \in \mathbb{G}$ and all $a, b \in \mathbb{Z}_p$, we have $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: $e(g, g) \neq 1$.

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.

2.3 Complexity Assumptions

Assumption 2.1 (Decisional Bilinear Diffie-Hellman, DBDH)**.** *Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of prime order p. Let g be a random generator of $\mathbb{G}$. 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 $\mathbf{Adv}_{\mathcal{A}}^{DBDH}(\lambda) = \big|\Pr[\mathcal{A}(D, Z_0) = 0] - \Pr[\mathcal{A}(D, Z_1) = 0]\big|$ where the probability is taken over random choices of $a, b, c, d \in \mathbb{Z}_p$.

3 Self-Updatable Encryption

In this section, we first define the syntax and the chosen ciphertext security of SUE. After that we propose an efficient SUE scheme in bilinear groups and prove its CCA security under the standard assumption.

3.1 Definitions

Before we introduce SUE, we define ciphertext delegatable encryption (CDE) that is a building block of SUE. The concept of CDE was introduced by Lee et al. [14] and it is public-key encryption (PKE) that supports the delegation of ciphertexts. We follow the definition of CDE in [14]. In CDE, a ciphertext header is associated with a label string L and a private key is also associated with a label string L'. The ciphertext header with L can be delegated to a new ciphertext header with a new label string L' with the restriction that L is a prefix of L'. If L is a prefix of L', then the ciphertext header with L can be decrypted by the private key with L'. The syntax of CDE is given as follows:

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

Setup$(1^\lambda, l)$*. The setup algorithm takes as input a security parameter* 1^λ *and the maximum length* l *of the label strings. It outputs a master key* MK *and public parameters* PP*.*

GenKey(L, MK, PP)*. The key generation algorithm takes as input a label string* $L \in \{0,1\}^k$ *with* $k \leq l$*, the master key* MK*, and the public parameters* PP*. 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*. 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*. It outputs a delegated ciphertext header* $CH_{L'}$ *for the label string* $L' = L||c$*.*

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*. It outputs a session key* EK *or the distinguished symbol* $\perp$*.*

The correctness property of CDE is defined as follows: For all PP, MK *generated by* ***Setup****, 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 delegated ciphertext header of ***DelegateCT*** *is a valid ciphertext header under the new label string.*

SUE, introduced by Lee et al. [14], is new PKE that supports the updating of ciphertexts by using a public key (or public parameters). We follow the definition of SUE in [14]. In SUE, a ciphertext header is associated with a time T and a private key is associated with a time T'. If $T' \geq T$, then the ciphertext header with T can be decrypted by a private key with T'. That is, the ciphertext header with T can be updated to a new ciphertext header with T' and then the private key with T' can decrypt the updated ciphertext header. The syntax of SUE is given as follows:

Definition 3.2 (Self-Updatable Encryption)**.** *A self-updatable encryption (SUE) scheme consists of five PPT algorithms,* ***Setup****,* ***GenKey****,* ***Encrypt****,* ***UpdateCT****, and* ***Decrypt****, which are defined as follows:*

Setup$(1^\lambda, T_{max})$*. The setup algorithm takes as input a security parameter* 1^λ *and the maximum time* T_{max}*. 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*. It outputs a private key* SK_T*.*

Encrypt(T, PP)*. The encryption algorithm takes as input a time* T*, and the public parameters* PP*. 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*. It outputs an updated ciphertext header* CH_{T+1}*.*

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*. It outputs a session key* EK *or the distinguished symbol* $\perp$*.*

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

- *If $T \leq T'$, then **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 updated ciphertext header of **UpdateCT** is a valid ciphertext header under the new time.*

Remark 3.3. *The syntax of SUE in [14] additionally includes the ciphertext randomization algorithm **RandCT**, but we omit this algorithm in the above syntax of SUE. The **RandCT** algorithm ensures that the ciphertext distribution of **SUE.UpdateCT** is statistically equal to that of **Encrypt** by completely re-randomizing the output of **UpdateCT**. However, in this paper, we weaken this strong requirement by just requiring that the output of **UpdateCT** is just a valid ciphertext header because of the reason in Remark 3.7. In this case, we do not need to completely re-randomize the updated ciphertext header of **UpdateCT**.*

Remark 3.4. *If a ciphertext header CH_T with a time T is updated to multiple ciphertext headers CH_{T_1}, CH_{T_2}, and CH_{T_3} where $T_1, T_2, T_3 \geq T$, then those updated ciphertext headers should be re-randomized to remove the relationship between ciphertext headers. That is, an adversary who obtained the session key of a ciphertext header can use this session key to break other ciphertext headers if they are not re-randomized. However, in most applications we can ensure that a ciphertext header CH_T is updated to a new single ciphertext header CH_{T+1} and completely delete the previous one. In this case, we can prevent the previous attack since there is only one ciphertext header that is related to the original ciphertext header.*

Security against chosen plaintext attacks (CPA security) for SUE was introduced by Lee et al. [14]. We define security against chosen ciphertext attacks (CCA security) for SUE by modifying their definition of CPA security. To be precise, there are several notions of CCA security: security against lunchtime attacks (or CCA1 security) and security against adaptively chosen ciphertext attacks (or CCA2 security) [18, 20]. We simply use CCA security for CCA2 security. Although CCA security is regarded as the standard notion for security of encryption schemes, it is too strong for SUE since CCA security cannot be achievable in SUE. In CCA security, an adversary is allowed to request a decryption query with the restriction that the challenge ciphertext given to the adversary cannot be queried. However, an adversary attacking an SUE scheme can query an updated ciphertext to the decryption oracle after obtaining the updated ciphertext by updating the challenge ciphertext. Thus, the adversary can easily break CCA security of SUE. To solve this problem in security definition, we relax the CCA security by restricting that the adversary cannot query the decryption oracle on a ciphertext that is the challenge or updated from the challenge ciphertext. We may view these ciphertexts that are the challenge or updated from the challenge ciphertext as time extended challenge (TEC) ciphertexts. The TEC-CCA security of SUE is given as follows:

Definition 3.5 (Selective TEC-CCA Security)**.** *The selective TEC-CCA security for SUE schemes is defined in terms of the indistinguishability under time extended challenge chosen plaintext attacks (IND-TEC-CCA). The security game is defined as the following game between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:*

***Init**: $\mathcal{A}$ first submits a challenge time T^*.*

***Setup**: $\mathcal{C}$ runs the setup algorithm to generate MK and PP. It gives PP to $\mathcal{A}$.*

***Query 1**: $\mathcal{A}$ adaptively requests a polynomial number of private key and decryption queries. $\mathcal{C}$ handles the queries as follows:*

- *If this is a private key query for a time T subject to the restriction $T < T^*$, then it creates the private key SK_T for the time T by calling the key generation algorithm. It responses the query with SK_T to $\mathcal{A}$.*
- *If this is a decryption query for a ciphertext header CH_T, then it computes the decapsulated session key EK by calling the decryption algorithm and responses the query with EK to $\mathcal{A}$.*

Challenge: *$\mathcal{A}$ requests a challenge ciphertext header and a challenge session key. $\mathcal{C}$ creates a ciphertext header $CH^*_{T^*}$ and a session key EK^* by calling the encryption algorithm under the challenge time T^*. It flips a random bit $\gamma \in \{0,1\}$. If $\gamma = 0$, then it gives $CH^*_{T^*}$ and EK^* to $\mathcal{A}$. Otherwise, it gives $CH^*_{T^*}$ and a random session key EK' to $\mathcal{A}$.*

Query 2: *$\mathcal{A}$ continues to request private key and decryption queries. $\mathcal{C}$ handles the private key queries as the same as before. It handles the decryption queries as follows:*

- *If this is a decryption query for a ciphertext header CH_T subject to the restriction that CH_T is not updated from $CH^*_{T^*}$ in case of $T \geq T^*$, then it computes the decapsulated session key EK by calling the decryption algorithm. It responses the query with EK to $\mathcal{A}$.*

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

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

Remark 3.6. *The above TEC-CCA security for SUE is closely related to the relaxed CCA security notions: the benign malleability of Shoup [27], the generalized CCA (gCCA) security of An et al. [1], and the public detectable Replayable CCA (RCCA) security of Canetti et al. [8]. In these relaxed CCA security models, there exists an efficiently computable relation $R(-,-)$ on ciphertext headers and the decryption results of two ciphertext headers CH_1 and CH_2 are equal if $R(CH_1, CH_2) = True$. Thus, the decryption query on any ciphertext header CH' that satisfies $R(CH^*, CH') = True$ where CH^* is the challenge ciphertext header is not allowed in this model. In TEC-CCA, the relation $R(-,-)$ checks whether a ciphertext header is updated from another ciphertext header or not.*

Remark 3.7. *If an SUE scheme can re-randomize the output of the **UpdateCT** algorithm, then this re-randomizable SUE scheme cannot satisfy the TEC-CCA security. The reason is that there is no efficiently computable relation $R(-,-)$ that can check whether a ciphertext header is updated from the challenge ciphertext header or not. Therefore, the syntax of SUE for TEC-CCA security only requires for the **UpdateCT** algorithm to output a just valid ciphertext header.*

3.2 Managing the Time Structure

To efficiently manage the time structure, we use a full binary tree as in [5, 14, 23]. We define some useful notations for a binary tree. Let v be a node in $\mathcal{BT}$. $\mathbf{Parent}(v)$ is the parent node of the input node v. $\mathbf{Path}(v)$ is the set of path nodes from the root node to the input node v. $\mathbf{RightSibling}(v)$ is the right sibling node of v. That is, $\mathbf{RightSibling}(v) = \mathbf{RightChild}(\mathbf{Parent}(v))$ where $\mathbf{RightChild}(v)$ is the right child of v. We also define $\mathbf{TimeNodes}(v) = \{v\} \cup \mathbf{RightSibling}(\mathbf{Path}(v)) \setminus \mathbf{Path}(\mathbf{Parent}(v))$ where $\mathbf{RightSibling}(\mathbf{Path}(v))$ is the set of right sibling nodes of $\mathbf{Path}(v)$. Let L be the label string of a node v. We also define $\mathbf{Parent}(L)$,

Path(L), **RightSibling**(L), and **TimeNodes**(L) similarly except that these are defined by using the label string L of v.

For each node in $\mathcal{BT}$, we assign a unique time value $T \in \{1, \dots, T_{max}\}$ by using pre-order traversal that recursively visits the root node, the left subtree, and the right subtree. That is, the root node is assigned to 1, the left most leaf node is assigned to $d+1$, and the right most leaf node is assigned to $T_{max} = 2^{d+1} - 1$ where d is the depth of the tree. We let v_T be a node associated with a time T. We define a mapping ψ from a time T to a label L. That is, $\psi(T)$ returns the label L of a node v_T associated with a time T. The following theorem guarantees that we can handle the time components efficiently.

Theorem 3.8 ([23]). *Let $\mathcal{BT}$ be a full binary tree of depth d and v_T be a node associated with a time T by pre-order traversal. For any node $v_T \in \mathcal{BT}$, **TimeNodes**(v_T) satisfies the following properties:*

- *Property 1. **TimeNodes**$(v_T) \cap$ **Path**$(v_{T'}) \neq \emptyset$ if and only if $T \leq T'$*
- *Property 2. If $v \in$ **TimeNodes**(v_{T+1}), then there is an ancestor of v in **TimeNodes**(v_T).*
- *Property 3. $|$**TimeNodes**$(v_T)| \leq d+1$*

Remark 3.9. *In pre-order traversal, if v_T is an internal node, then $v_{T+1} =$ **LeftChild**(v_T). If v_T is a leaf node, then $v_{T+1} =$ **RightChild**$(v_{T'})$ where $v_{T'} \in$ **Path**(v_T) is a node with the largest depth such that **LeftChild**$(v_{T'}) \in$ **Path**(v_T).*

3.3 Construction

We use the CPA-secure CDE scheme of Lee [13] as the building block of our TEC-CCA-secure SUE scheme. The main challenge when devising a TEC-CCA-secure SUE scheme is providing the integrity of ciphertexts while the updating of ciphertexts is also provided. To solve this problem, we observe that ciphertext elements can be divided into two parts: one part is related to the ciphertext updating and another part is related to the session key of the ciphertext. The validity of ciphertext elements for the ciphertext updating can be easily checked by using bilinear maps since these elements are composed of Diffie-Hellman (DH) tuples. The integrity of ciphertext elements for the session key can be achieved by using the well-known transformation of Canetti et al. [6]. To be precise, we will use the direct method of Boyen et al. [4] to improve the efficiency. The CDE scheme of Lee [13] is described as follows:

CDE.Init(1^λ)**:** It first generates bilinear groups $\mathbb{G}, \mathbb{G}_T$ of prime order p. It chooses a random generator $g \in \mathbb{G}$ and outputs $GDS = ((p, \mathbb{G}, \mathbb{G}_T, e),\ g)$.

CDE.Setup(GDS, l)**:** It chooses a random exponent $\alpha \in \mathbb{Z}_p$ and random elements $w, v, u, \{h_{i,0}, h_{i,1}\}_{i=1}^{l} \in \mathbb{G}$. Let $F_{i,b}(L) = u^L h_{i,b}$ where $i \in [l]$ and $b \in \{0,1\}$. It outputs a master key $MK = \beta$ and public parameters $PP = \big((p, \mathbb{G}, \mathbb{G}_T, e),\ g,\ w,\ v,\ u,\ \{h_{i,0}, h_{i,1}\}_{i=1}^{l},\ \Lambda = e(g,g)^\beta\big)$.

CDE.GenKey(L, MK, PP)**:** Let L be an n-bit label string. It chooses random exponents $r, r_1, \dots, r_n \in \mathbb{Z}_p$ and outputs a private key $SK_L = \big(K_0 = g^\beta w^r,\ K_1 = g^{-r},\ \{K_{i,1} = v^r F_{i,L[i]}(L|_i)^{r_i},\ K_{i,2} = g^{-r_i}\}_{i=1}^{n}\big)$.

CDE.RandKey(SK_L, δ, PP)**:** Let $SK_L = (K_0, K_1, \{K_{i,1}, K_{i,2}\})$ and δ be an exponent in $\mathbb{Z}_p$. It selects random exponents $r', r'_1, \dots, r'_n \in \mathbb{Z}_p$ and outputs a re-randomized private key $SK_L = \big(K'_0 = K_0 \cdot g^\delta w^{r'},\ K'_1 = K_1 \cdot g^{-r'},\ \{K'_{i,1} = K_{i,1} \cdot v^{r'} F_{i,L[i]}(L|_i)^{r'_i},\ K'_{i,2} = K_{i,2} \cdot g^{-r'_i}\}_{i=1}^{n}\big)$.

CDE.Encrypt($L, t, \vec{s}, PP$): Let L be a d-bit label string. By using the given exponent $t \in \mathbb{Z}_p$ and the vector $\vec{s} = (s_1, \dots, s_d) \in \mathbb{Z}_p^d$, it outputs a ciphertext header $CH_L = \big(C_0 = g^t,\ C_1 = w^t \prod_{i=1}^{d} v^{s_i},\ \big\{C_{i,1} = g^{s_i},\ C_{i,2} = F_{i,L[i]}(L|_i)^{s_i}\big\}_{i=1}^{d}\big)$ and a session key $EK = \Lambda^t$.

CDE.DelegateCT(CH_L, c, PP): Let $CH_L = (C_0, C_1, \{C_{i,1}, C_{i,2}\})$ where $L \in \{0,1\}^d$. It first sets a new label string $L' = L \| c$ where $c \in \{0,1\}$. It selects a random exponent $s_{d+1} \in \mathbb{Z}_p$ and outputs a delegated ciphertext header $CH_{L'} = \big(C'_0 = C_0,\ C'_1 = C_1 \cdot v^{s_{d+1}},\ \{C'_{i,1} = C_{i,1},\ C'_{i,2} = C_{i,2}\}_{i=1}^{d},\ C'_{d+1,1} = g^{s_{d+1}},\ C'_{d+1,2} = F_{d+1,c}(L')^{s_{d+1}}\big)$.

CDE.VerifyCT(CH_L, L, PP): Let $CH_L = (C_0, C_1, \{C_{i,1}, C_{i,2}\})$ for a label string $L \in \{0,1\}^d$. It checks that $e(C_1, g) \stackrel{?}{=} e(C_0, w) \cdot e(\prod_{i=1}^{d} C_{i,1}, v)$ and $e(C_{i,2}, g) \stackrel{?}{=} e(C_{i,1}, F_{i,L[i]}(L|_i))$ for all $i \in \{1, \dots, d\}$. If all tests pass, then it outputs 1. Otherwise, it outputs 0.

CDE.Decrypt($CH_L, SK_{L'}, PP$): Let CH_L be a ciphertext header for a label string L. Let $SK_{L'} = (K_0, K_1, \{K_{i,1}, K_{i,2}\})$ for a label string $L' \in \{0,1\}^n$. If L is a prefix of L', then it obtains $CH'_{L'} = (C'_0, C'_1, \{C'_{i,1}, C'_{i,2}\})$ for L' by iteratively running **CDE.DelegateCT** and outputs a session key EK by computing $e(C'_0, K_0) \cdot e(C'_1, K_1) \cdot \prod_{i=1}^{n} \big(e(C'_{i,1}, K_{i,1}) \cdot e(C'_{i,2}, K_{i,2})\big)$. Otherwise, it outputs $\perp$.

Let $\mathcal{H}$ be a family of collision resistant hash functions H^z, indexed by some finite index set $\{z\}$. Our SUE scheme is described as follows:

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

SUE.Setup(GDS, T_{max}): It first chooses random elements $u_S, h_S \in \mathbb{G}$ and obtains MK_{CDE} and PP_{CDE} by running **CDE.Setup**(GDS, l) where $T_{max} = 2^{l+1} - 1$. It also chooses a random index z for a hash function $H^z \in \mathcal{H}$. It outputs a master key $MK = MK_{CDE}$ and public parameters $PP = \big(PP_{CDE},\ z,\ u_S, h_S\big)$.

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

SUE.RandKey(SK_T, PP): It outputs SK_T by running **CDE.RandKey**(SK_T, PP).

SUE.Encrypt(T, t, PP): It sets a label string $L = \psi(T)$ and proceeds as follows:

1. It first obtains $TL = (L^{(0)}, L^{(1)}, \dots, L^{(d)})$ by computing **TimeNodes**(L) where $L = L^{(0)}$. Let $d^{(j)}$ be the length of the label $L^{(j)}$. Note that the labels in TL are ordered according to pre-order traversal.
2. For each label $L^{(j)}$ in TL such that $0 \le j \le d$, it proceeds the following steps:
   (a) If $j = 0$, then it sets a vector $\vec{s} = (s_1, \dots, s_{d^{(0)}})$ by selecting random exponents $s_1, \dots, s_{d^{(0)}} \in \mathbb{Z}_p$. It obtains $CH^{(0)} = (C_0, C_1, \{C_{i,1}, C_{i,2}\}_{i=1}^{d^{(0)}})$ and EK by running **CDE.Encrypt**$(L^{(0)}, t, \vec{s}, PP)$.
   (b) If $j \ge 1$, then 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 selected in $\mathbb{Z}_p$. It obtains $CH^{(j)} = (C'_0, C'_1, \{C'_{i,1}, C'_{i,2}\}_{i=1}^{d^{(j)}})$ by running **CDE.Encrypt**$(L^{(j)}, t, \vec{s}', PP)$. Next, it removes redundant elements $C'_0, \{C'_{i,1}, C'_{i,2}\}_{i=1}^{d^{(j)}-1}$ from $CH^{(j)}$ since they are contained in $CH^{(0)}$.
3. It computes $\pi = H^z(C_0)$ and sets $C_3 = (u_S^{\pi} h_S)^t$.
4. It outputs a ciphertext header $CH_T = \big(CH^{(0)}, CH^{(1)}, \dots, CH^{(d)}, C_3\big)$ and EK.

SUE.UpdateCT($CH_T, T+1, PP$): Let $CH_T = (CH^{(0)}, \dots, CH^{(d)}, C_3)$ and $L^{(j)}$ be the label of $CH^{(j)}$. It proceeds as follows:

1. If the length d of $L^{(0)}$ is less than l, 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 for the next time $T+1$ by pre-order traversal. Next, it prunes redundant elements in $CH_{L^{(0)}||1}$. It outputs an updated ciphertext header $CH_{T+1} = \big(CH'^{(0)} = CH_{L^{(0)}||0}, CH'^{(1)} = CH_{L^{(0)}||1}, CH'^{(2)} = CH^{(1)}, \dots, CH'^{(d+1)} = CH^{(d)}, C_3\big)$.
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 $CH_{T+1} = \big(CH'^{(0)} = CH^{(1)}, \dots, CH'^{(d-1)} = CH^{(d)}, C_3\big)$.

SUE.VerifyCT(CH_T, T, PP): Let $CH_T = (CH^{(0)}, \dots, CH^{(d)}, C_3)$. It sets a label string $L = \psi(T)$ and proceeds as follows:

1. It first obtains $TL = (L^{(0)}, L^{(1)}, \dots, L^{(d)})$ by computing **TimeNodes**(L) where $L = L^{(0)}$. Note that the labels in TL are ordered according to pre-order traversal. It checks that the number of labels in TL is the same as the number of CDE ciphertext headers in CH_T.
2. For each label $L^{(j)}$ in TL, it checks $1 \stackrel{?}{=}$ **CDE.VerifyCT**$(CH^{(j)}, L^{(j)}, PP)$. Note that the common elements of $CH^{(0)}$ should be copied to $CH^{(j)}$ when $j > 0$.
3. Next, it computes $\pi = H^z(C_0)$ and checks $e(C_3, g) \stackrel{?}{=} e(C_0, u_S^{\pi} h_S)$ where C_0 is the element in $CH^{(0)}$.
4. It outputs 1 if all checking pass. Otherwise, it outputs 0.

SUE.Decrypt($CH_T, SK_{T'}, PP$): If $T > T'$, then it outputs $\perp$ since it cannot decrypt. Otherwise, it proceeds as follows:

1. It first checks $1 \stackrel{?}{=}$ **SUE.VerifyCT**(CH_T, T, PP). If the checking fails, then it outputs $\perp$.
2. 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**$(CH^{(j)}, SK_{T'}, PP)$.

Remark 3.10. *Compared to the CPA-secure SUE scheme of Lee [13], the main difference of our SUE scheme is the **Encrypt** and **Decrypt** algorithms. The **Encrypt** algorithm additionally generates a C_3 group element to provide the integrity of the C_1 group element by using the direct method of Boyen et al. [4]. The **Decrypt** algorithm checks the validity of the ciphertext header before it derives a session key by running the **VerifyCT** algorithm. The **VerifyCT** algorithm first checks the validity of CDE ciphertext headers by running **VerifyCT** of CDE and then checks the integrity of C_1 by using bilinear maps. To check the validity of CDE ciphertext headers, we additionally added the **VerifyCT** algorithm in CDE and it checks the validity of CDE ciphertext header by using bilinear maps since the group elements in a CDE ciphertext header consist of DDH tuples.*

3.4 Correctness

The correctness of the CDE scheme was shown by Lee [13]. Let CH_T be a ciphertext header with a time T generated by **SUE.Encrypt** and $SK_{T'}$ be a private key with a time T' generated by **SUE.GenKey**. To show the correctness of the above SUE scheme, we should show that **SUE.Decrypt** derives a valid session key EK from CH_T and $SK_{T'}$ if $T \leq T'$ and that CH_{T+1} generated by **SUE.UpdateCT** is a valid ciphertext header with a time $T+1$.

We first show that **SUE.Decrypt** can derive a valid session key by presenting that **SUE.VerifyCT** can check the validity of CH_T and **CDE.Decrypt** can be used to derive a session key. The algorithm **SUE.VerifyCT** checks that the number of CDE ciphertext headers in CH_T, the validity of each CDE ciphertext header by running **CDE.VerifyCT**, and the validity of C_3. The validity of CDE ciphertext headers can be easily checked by using bilinear maps since a CDE ciphertext header consists of elements composed of Diffie-Hellman (DH) tuples and bilinear maps can check the validity of DDH tuples. The validity of C_3 also can be easily checked by using bilinear maps since C_3 is also a DDH tuple.

The **CDE.Decrypt** algorithm only can derive a valid session key if the label of a CDE ciphertext header is a prefix of the label of a CDE private key. From the property 2 of Theorem 3.8, we have $\textbf{TimeNodes}(\psi(T)) \cap \textbf{Path}(\psi(T')) \neq \emptyset$ if $T \geq T'$ where T and T' are times associated to the SUE ciphertext header and the SUE private key respectively. Thus, **CDE.Decrypt** can derive a session key since the SUE ciphertext header consists of many CDE ciphertext header with labels in $\textbf{TimeNodes}(\psi(T))$ and the SUE private key is equal to the CDE private key.

We now show that the output of **SUE.UpdateCT** is a valid ciphertext header. Recall that an SUE ciphertext header CH_T with a time T consists of CDE ciphertext headers that are associated with labels in $\textbf{TimeNodes}(\psi(T))$. Let v_T be a node in $\mathcal{BT}$ associated with a time T by pre-order traversal. If v_T is an internal node, we have $v_{T+1} = \textbf{LeftChild}(v_T)$ from Remark 3.9. Thus we have $\textbf{TimeNodes}(\psi(T+1)) \setminus \textbf{TimeNodes}(\psi(T)) = \{\textbf{LeftChild}(v_T), \textbf{RightChild}(v_T)\}$ since $\textbf{RightSibling}(v_{T+1}) = \textbf{RightChild}(v_T)$ by the definition of **TimeNodes**. If v_T is a leaf node, $v_{T+1} = \textbf{RightChild}(v_{T'})$ where $v_{T'} \in \textbf{Path}(v_T)$ is a node with the largest depth such that $\textbf{LeftChild}(v_{T'}) \in \textbf{Path}(v_T)$ from Remark 3.9. Thus we have $\textbf{TimeNodes}(\psi(T)) \setminus \textbf{TimeNodes}(\psi(T+1)) = v_T$. Therefore, the correctness of **SUE.UpdateCT** is easily obtained if **CDE.DelegateCT** outputs a valid CDE ciphertext header.

3.5 Security Analysis

To prove the TEC-CCA security of the above SUE scheme, we use the well-known partitioning method. To simplify the security proof, we use the meta-simulation technique of Lee [13]. In the meta-simulation technique, a simulator (meta-simulator) for the TEC-CCA security proof uses the previous simulator for the CPA security proof as a sub-simulator. Recall that the original SUE scheme of the above SUE scheme was proven to be selectively secure under the DBDH assumption by Lee [13]. If the meta-simulator use the previous simulator as a sub-simulator, then it can use the power of the sub-simulator to generate private keys and some elements of a challenge ciphertext header.

Theorem 3.11 ([13]). *The original SUE scheme is selectively secure under chosen plaintext attacks if the DBDH assumption holds.*

The simulator of this proof sets the element g in the assumption (g, g^a, g^b, g^c) as the generator of public parameters, implicitly sets ab as the master key β, and sets g^c in the assumption as the element g^t in a challenge ciphertext header. Note that these three settings are essential for the correctness of the meta-simulation. Now we can prove the TEC-CCA security of our SUE scheme as follows:

Theorem 3.12. *The above SUE scheme is selectively secure under time extended challenge chosen ciphertext attacks if the DBDH assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that $\textbf{Adv}_{\mathcal{A}}^{SUE}(\lambda) \leq \textbf{Adv}_{\mathcal{B}}^{DBDH}(\lambda) + \textbf{negl}(\lambda)$.*

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 $Z = Z_1 \in \mathbb{G}_T$. Let $\mathcal{B}_{SUE}$ be a simulator in the security proof of Theorem 3.11. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits a challenge time T^*. $\mathcal{B}$ first runs $\mathcal{B}_{SUE}$ by giving D and Z.

Setup: $\mathcal{B}$ submits T^* to $\mathcal{B}_{SUE}$ and receives $PP_{SUE} = ((p,\mathbb{G},\mathbb{G}_T,e),g,w,v,u,\{h_{i,j}\},\Lambda = e(g,g)^{\beta})$. Note that $\mathcal{B}_{SUE}$ implicitly sets $\beta = ab$. It chooses random exponents $u'_S, h'_S \in \mathbb{Z}_p$. It also chooses a random index z for H^z and calculates $\pi^* = H^z(g^c)$. It implicitly sets $\beta = ab$ and publishes public parameters $PP = \big(PP_{SUE},\ z,\ u_S = g^a g^{u'_S},\ h_S = (g^a)^{-\pi^*} g^{h'_S}\big)$.

Query 1: $\mathcal{A}$ adaptively requests a polynomial number of private key queries and decryption queries. If $\mathcal{A}$ requests a private key query for a time T such that $T < T^*$, then $\mathcal{B}$ receives a private key SK_T by requesting a private key query to $\mathcal{B}_{SUE}$ and responses SK_T to $\mathcal{A}$. If $\mathcal{A}$ requests a decryption query for a ciphertext header CH_T with a time T, then $\mathcal{B}$ handles this query as follows:

1. Let $CH_T = (CH^{(0)},\dots,CH^{(d)},C_3)$ and $CH^{(0)} = (C_0,C_1,\{C_{i,1},C_{i,2}\})$. It first checks whether the ciphertext header CH_T is valid or not by running **SUE.VerifyCT**(CH_T,T,PP). If the ciphertext header is not valid, then it responds with $\perp$. Otherwise, the ciphertext header is well-formed such as $C_0 = g^t$ for some unknown $t \in \mathbb{Z}_p$.

2. If $T < T^*$, then it receives a private key SK_T by requesting a private key query to $\mathcal{B}_{SUE}$ and obtains a session key EK by running **SUE.Decrypt**(CH_T,SK_T,PP).

3. If $T \geq T^*$, then it calculates $\pi = H^z(C_0)$ and performs the following steps: If $\pi = \pi^*$, then it terminates the simulation with $\mathcal{A}$ and halts since it cannot response. If $\pi \neq \pi^*$, then it selects a random exponent $r' \in \mathbb{Z}_p$ and computes the session key by implicitly setting $r_3 = -(u'_S\pi + h'_S)/(\pi - \pi^*) + r'$ as

$$\begin{aligned} EK &= e\big(C_0, (g^b)^{-\frac{u'_S\pi+h'_S}{\pi-\pi^*}} (u_S^{\pi} h_S)^{r'}\big) \cdot e\big(C_3, (g^b)^{\frac{1}{\pi-\pi^*}} g^{-r'}\big) \\ &= e(g^t, g^{ab}(u_S^{\pi} h_S)^{r_3}) \cdot e((u_S^{\pi} h_S)^t, g^{-r_3}) = e(g^a,g^b)^t. \end{aligned}$$

4. It responds to the query with the session key EK.

Challenge: To create the challenge ciphertext header and the session key for the challenge time T^*, $\mathcal{B}$ proceeds as follows:

1. It first requests a challenge query to $\mathcal{B}_{SUE}$ and receives $CH^* = (CH^{(0)},CH^{(1)},\dots,CH^{(d)})$ and EK^*.

2. It sets $C_3 = (g^c)^{u'_S\pi^*+h'_S}$. Note that this component is valid since $(g^c)^{u'_S\pi^*+h'_S} = \big((g^a g^{u'_S})^{\pi^*} \cdot (g^a)^{-\pi^*} g^{h'_S}\big)^c = (u_S^{\pi^*} h_S)^c$ for $\pi^* = H^z(C_0) = H^z(g^c)$.

3. It sets the challenge ciphertext header $CH^*_{T^*} = \big(CH^{(0)},CH^{(1)},\dots,CH^{(d)},C_3\big)$ and gives $CH^*_{T^*}$ and EK^* to $\mathcal{A}$.

Query 2: Same as Query 1.

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

To finish the proof, we show that the meta-simulator $\mathcal{B}$ correctly handles the queries of $\mathcal{A}$. The public parameters PP is correct since PP_{SUE} is correct and u_S, h_S are properly randomized. The private key is also correct since it is generated by $\mathcal{B}_{SUE}$. Now, we will show that the decryption query is correctly handled.

From the correctness of the **SUE.VerifyCT** algorithm, we confirm that the ciphertext header CH_T is associated with a claimed time T. The decryption only fails when $T \geq T^*$ and $\pi = \pi^*$ where $\pi = H^z(C_0)$ and $\pi^* = H^z(g^c)$. The probability of this collision event is negligible since H^z is a collision resistant function and $C_0 \neq g^c$ from the restriction of the security model. Finally, the challenge ciphertext is also correct since C_3 can be easily calculated. This completes our proof. □

Corollary 3.13. *The above SUE scheme is fully secure under time extended challenge chosen ciphertext attacks if the DBDH assumption holds and T_{max} is a polynomial value. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{\mathcal{A}}^{SUE}(\lambda) \leq T_{max} \cdot \boldsymbol{Adv}_{\mathcal{B}}^{DBDH}(\lambda) + \boldsymbol{negl}(\lambda)$.*

3.6 Discussions

Efficiency Analysis. The proposed SUE scheme consists of $O(\log T_{max})$ group elements in public parameters, a private key, and a ciphertext header since it uses the CDE scheme of Lee [13]. Note that our SUE scheme additionally contains two group elements in public parameters and one group element in a ciphertext header to provide the TEC-CCA security. In contrast to the CPA secure SUE scheme of Lee, our TEC-CCA secure SUE scheme requires to check the validity of ciphertext headers in the decryption algorithm. If the **SUE.Decrypt** algorithm checks the validity of ciphertext headers by naively performing the **CDE.VerifyCT** algorithm, then it requires $O(\log^2 T_{max})$ pairing operations since each **CDE.VerifyCT** requires $O(\log T_{max})$ pairing operations and an SUE ciphertext header consists of $O(\log T_{max})$ CDE ciphertext headers. However, we can reduce the number of pairing operations from $O(\log^2 T_{max})$ to $7 \log T_{max}$ since pairing operations for redundant elements can be omitted. The details of this improvement are given below in this section. Thus, the decryption algorithm just requires $9 \log T_{max}$ pairing operations since it additionally requires $2 \log T_{max}$ pairings in the **CDE.Decrypt** algorithm.

Reducing Public Parameters. The public parameters of our SUE scheme consists of $O(\log T_{max})$ group elements since the CDE scheme of Lee [13], which is CPA secure under the DBDH assumption, is used as the building scheme for the SUE scheme. To reduce the size of public parameters, we can employ the CDE scheme of Lee with short public parameters. In this case, we have an SUE scheme with $O(1)$ group elements in public parameters. Note that the size of a private key and a ciphertext header remains the same as before. However, this SUE scheme only can be proven to be TEC-CCA secure under the q-type assumption instead of the standard DBDH assumption.

Time-Interval SUE. By combining two CPA secure SUE schemes, a CPA secure time-interval SUE (TI-SUE) scheme can be constructed as presented by Lee [13]. In TI-SUE, a ciphertext header is associated with a time range specified by two times T_L and T_R and a private key is associated with a time T'. If $T_L \leq T' \leq T_R$, then the ciphertext with T_L and T_R can be decrypted by a private key with T'. By following the design of Lee, we can also build a TEC-CCA secure TI-SUE scheme by combining two TEC-CCA secure SUE schemes. That is, the master key is simply shared between two SUE schemes to prevent collusion attacks, the ciphertext header of one SUE scheme is used for future-time SUE, and the ciphertext header of another SUE scheme is used for past-time SUE. This TI-SUE scheme also can be proven to be secure under the DBDH assumption. Note that we also can reduce the size of public parameters if the CDE scheme with short public parameters is used.

Improved Ciphertext Verification. The ciphertext header of our SUE scheme consists of at most $\log T_{max}$ CDE ciphertext headers and the verification of each CDE ciphertext header requires $2 \log T_{max} + 3$ pairing operations. Thus a simple verification of an SUE ciphertext header requires at most $2 \log^2 T_{max} + 2 \log T_{max}$ pairing operations. As mentioned before, the number of pairing operations in the **CDE.VerifyCT** algorithm can be reduced if we omit the checking of the redundant elements in a CDE ciphertext header. In this case, we require $2\log T_{max} + 3$ pairing operations to check $CH^{(0)}$ and 5 pairing operations to check $CH^{(i)}$. Therefore we only need at most $7\log T_{max} + 3$ pairing operations.

4 Revocable-Storage Attribute-Based Encryption

In this section, we define the syntax and the TEC-CCA security of RS-ABE. We also propose an RS-ABE scheme and prove its TEC-CCA security.

4.1 Definitions

Revocable-Storage ABE (RS-ABE) is ABE that supports user revocation and ciphertext updating. The concept of RS-ABE was introduced by Sahai et al. [23] to handle the access control problem on ciphertexts in cloud storage. In this paper, we follow the RS-ABE syntax of Lee et al. [14]. In RS-ABE with ciphertext-policy, a user's private key is associated with a set of attributes S and an index u and a ciphertext is associated with an access structure $\mathbb{A}$ and a time T. A center periodically broadcast an update key that excludes a set of revoked users R on time T. If a user is not revoked ($u \notin R$) and the set of attributes satisfies the access structure ($S \in \mathbb{A}$), then the user with a private key and an update key can decrypt the ciphertext. The syntax of RS-ABE is given as follows:

Definition 4.1 (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****,* ***DeriveKey****,* ***Encrypt****,* ***UpdateCT****, and* ***Decrypt****, which are defined as follows:*

Setup$(1^\lambda, T_{max}, N_{max})$*. The setup algorithm takes as input a security parameter 1^λ, the maximum time T_{max}, and the maximum number of users N_{max}, and it outputs 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. 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. 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. It outputs a decryption key $DK_{S,T}$ or the distinguished symbol $\perp$.*

Encrypt$(\mathbb{A}, T, PP)$*. The encryption algorithm takes as input an access structure $\mathbb{A}$, a time $T \leq T_{max}$, and the public parameters PP. It outputs a ciphertext header $CH_{\mathbb{A},T}$ and a session key EK.*

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

Decrypt$(CH_{\mathbb{A},T}, DK_{S,T'}, PP)$*. The decryption algorithm takes as input a ciphertext header $CH_{\mathbb{A},T}$, a decryption key $DK_{S,T}$, and the public parameters PP. It outputs a session key EK 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, any $CH_{\mathbb{A},T}, EK$ 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**$(CH_{\mathbb{A},T}, DK_{S,T'}, PP) = EK$.*
- *If $(S \notin \mathbb{A}) \vee (T' < T)$, then **Decrypt**$(CH_{\mathbb{A},T}, DK_{S,T'}, PP) = \perp$ with all but negligible probability.*

*Additionally, it requires that the updated ciphertext header of **UpdateCT** is a valid ciphertext header under the new time.*

Remark 4.2. *The original definition of CPA-secure RS-ABE additionally contains the **RandCT** algorithm that randomizes a ciphertext header [14, 23]. However, our definition of RS-ABE with TEC-CCA security omits the **RandCT** algorithm because of the reason in Remark 4.5. Thus, the output of the **UpdateCT** algorithm is a just valid ciphertext header, and the randomness of an updated ciphertext header may be correlated to that of the original ciphertext header. Because of this correlation, if a ciphertext header is updated from an original one by using **UpdateCT**, then the original one should be deleted.*

Remark 4.3. *We define RS-ABE as a key encapsulation mechanism (KEM) version, in which the **Encrypt** algorithm derives a session key, instead of a full encryption version. Note that if a KEM scheme is combined with a symmetric key encryption scheme, then a full encryption scheme can be easily derive by using the hybrid encryption technique.*

The CPA security of RS-ABE was introduced by Sahai et al. [23]. We define the TEC-CCA security of RS-ABE by modifying their CPA-security model. Similar to the CCA security of SUE, RS-ABE also cannot achieve the traditional CCA2 security since the ciphertexts of RS-ABE can be updated by anyone. Thus, we also relax the definition of CCA2 security by restricting that an adversary cannot request a decryption query on a ciphertext that is updated from the challenge ciphertext given to the adversary. In this paper, we define selective time extended challenge (TEC) CCA security of RS-ABE where the adversary should submit a challenge access structure and a challenge time before he receives public parameters. The TEC-CCA security of RS-ABE is given as follows:

Definition 4.4 (Selective TEC-CCA Security)**.** *The selective security of RS-ABE is defined in terms of the indistinguishability under time extended challenge chosen ciphertext attacks (IND-TEC-CCA). The security game is defined as the following experiment between a challenger $\mathcal{C}$ and a PPT adversary $\mathcal{A}$:*

***Init**: $\mathcal{A}$ first submits a challenge access structure $\mathbb{A}^*$ and a challenge time T^*.*

***Setup**: $\mathcal{C}$ generates a master key MK and public parameters PP by calling the setup algorithm, and then it gives PP to $\mathcal{A}$.*

***Query 1**: $\mathcal{A}$ may adaptively request a polynomial number of private key, update key, decryption key, and decryption queries. $\mathcal{C}$ handles the queries as follows:*

- *If this is a private key query for a set of attributes S and a user index u, then it creates a private key $SK_{S,u}$ by calling the key generation algorithm and gives $SK_{S,u}$ to $\mathcal{A}$. 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 creates an update key $UK_{T,R}$ by calling the key update algorithm and gives $UK_{T,R}$ to $\mathcal{A}$. 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 creates a decryption key $DK_{S,T}$ by calling the decryption key derivation algorithm and gives $DK_{S,T}$ to $\mathcal{A}$.*
- *If this is a decryption query for a ciphertext header $CH_{\mathbb{A},T}$, then it computes the decapsulated session key EK by calling the decryption algorithm and gives EK to $\mathcal{A}$.*

We require the following restrictions on the queries of $\mathcal{A}$:

1. *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 $T < T_j$.*
2. *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.*
3. *If a decryption key for S and T was queried, then it is required that $S \notin \mathbb{A}^*$ or $T < T^*$.*

***Challenge**: $\mathcal{C}$ creates a ciphertext header $CH^*_{\mathbb{A}^*,T^*}$ and a session key EK^* by calling the encryption algorithm under the challenge access structure $\mathbb{A}^*$ and the challenge time T^*. It then flips a random bit $\mu \in \{0,1\}$. If $\mu = 0$ it sets $EK^*_0 = EK^*$, otherwise it sets EK^*_1 to a random session key. It gives $CH^*_{\mathbb{A}^*,T^*}$ and EK^*_μ to $\mathcal{A}$.*

***Query 2**: $\mathcal{A}$ continues to request private key, update key, decryption key, and decryption queries. $\mathcal{C}$ handles the queries as the same as before. In addition to the restrictions in query 1 step, we require the following additional restriction on the queries of $\mathcal{A}$:*

4. *If a decryption for a ciphertext header $CH_{\mathbb{A}^*,T}$ was queried, then it is required that $T < T^*$ or $CH_{\mathbb{A}^*,T}$ is not updated from $CH^*_{\mathbb{A}^*,T^*}$ for $T \geq T^*$.*

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

The advantage of $\mathcal{A}$ is defined as $\boldsymbol{Adv}^{RS\text{-}ABE}_{\mathcal{A}}(\lambda) = \big|\Pr[\mu = \mu'] - \frac{1}{2}\big|$ where the probability is taken over all the randomness of the game. An RS-ABE scheme is secure in the selective model under time extended challenge chosen ciphertext attacks if for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game is negligible in the security parameter λ.

Remark 4.5. *The adversary of the above TEC-CCA security model cannot request a decryption on an updated ciphertext header that is updated from the challenge ciphertext header. Thus, there should be an efficiently computable relation $R(-,-)$ that checks whether a ciphertext header is derived from another one or not. If an RS-ABE scheme supports perfect ciphertext re-randomization, then the security of this scheme cannot be proven in the above model since there is no efficiently computable relation $R(-,-)$.*

Remark 4.6. *Selective TEC-CCA security can be weakened to selective revocation list TEC-CCA security, in which an adversary should additionally submits a set of revoked users on the challenge time. Selective revocation list CPA security was introduced by Boldyreva et al. [2] to prove the security of their RS-ABE scheme and this is employed in RS-ABE by Lee [13]. Note that selective TEC-CCA security is stronger than selective revocation list TEC-CCA security.*

4.2 Subset Cover Framework

The subset cover (SC) framework, introduced by Naor, Naor, and Lotspiech [17], is a general methodology to construct an efficient revocation system. The complete subtree (CS) scheme is one instance of the SC framework. We follow the definition of CS in [14]. 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 $\mathcal{S}$ is defined as $\{S_i : v_i \in \mathcal{BT}\}$. Recall that S_i is the set of all the leaves in a subtree $\mathcal{T}_i$. It outputs $\mathcal{BT}$.

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

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

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

Lemma 4.7 ([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 $\log N_{max}$ and the size of a covering set is at most $r \log(N_{max}/r)$.*

4.3 Construction

Before we construct an RS-ABE scheme, we present a CCA-secure CP-ABE scheme from the CPA-secure CP-ABE scheme of Rouselakis and Waters [22]. To convert the CPA-secure CP-ABE scheme to a CCA-secure one, we follow the transformation of Canetti et al. [6]. To improve the efficiency, we actually employ the direct conversion method of Boyen et al. [4]. The KEM version of the CP-ABE scheme is given as follows:

CP-ABE.Setup(GDS): This algorithm takes as input a group description string GDS. It chooses random elements $w_A, v_A, u_A, h_A, u_B, h_B \in \mathbb{G}$, and a random exponent $\gamma \in \mathbb{Z}_p$. It also chooses a random index z for a hash function $H^z \in \mathcal{H}$. It outputs a master key $MK = \gamma$ and public parameters $PP = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, w_A, v_A, u_A, h_A, z, u_B, h_B, \Lambda = e(g,g)^\gamma\big)$.

CP-ABE.GenKey(S, MK, PP): Let $S = \{A_1, A_2, \dots, A_k\}$ be a set of attributes. It chooses random exponents $r, r_1, \dots, r_k \in \mathbb{Z}_p$ and outputs a private key $SK_S = \big(K_0 = g^\gamma w_A^r, K_1 = g^{-r}, \{K_{i,2} = v_A^r (u_A^{A_i} h_A)^{r_i}, K_{i,3} = g^{-r_i}\}_{i=1}^k\big)$.

CP-ABE.RandKey(SK_S, δ, PP): Let $SK_S = (K_0, K_1, \{K_{i,2}, K_{i,3}\})$ for $S = \{A_1, A_2, \dots, A_k\}$. It chooses random exponents $r', r'_1, \dots, r'_k \in \mathbb{Z}_p$ and outputs a re-randomized private key $SK_S = \big(K'_0 = K_0 \cdot g^\delta w_A^{r'}, K'_1 = K_1 \cdot g^{-r'}, \{K'_{i,2} = K_{i,2} \cdot v_A^{r'} (u_A^{A_i} h_A)^{r'_i}, K'_{i,3} = K_{i,3} \cdot g^{-r'_i}\}_{i=1}^k\big)$.

CP-ABE.Encrypt($\mathbb{A}, t, PP$)**:** Let $\mathbb{A} = (M, \rho)$ be an LSSS access structure where M is an $l \times n$ matrix and ρ is a map from each row M_j of M to an attribute $\rho(j)$. It first sets a random vector $\vec{v} = (t, v_2, \ldots, v_n)$ 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 computes $C_0 = g^t, \{C_{j,1} = w_A^{M_j \cdot \vec{v}} v_A^{s_j}, C_{j,2} = g^{s_j}, C_{j,3} = (u_A^{\rho(j)} h_A)^{s_j}\}_{1 \le j \le l}$. Next, it calculates $\pi = H^z(C_0, C_{1,1}, \ldots, C_{l,3})$ and sets $C_3 = (u_B^{\pi} h_B)^t$. It outputs a ciphertext header $CH_{\mathbb{A}} = \big(C_0, \{C_{j,1}, C_{j,2}, C_{j,3}\}_{j=1}^{l}, C_3\big)$ and a session key $EK = \Lambda^t$.

CP-ABE.VerifyCT($CH_{\mathbb{A}}, PP$)**:** Let $CH_{\mathbb{A}} = (C_0, \{C_{j,1}, C_{j,2}, C_{j,3}\}, C_3)$. It first computes $\pi = H^z(C_0, C_{1,1}, \ldots, C_{l,3})$ and checks $e(C_3, g) \stackrel{?}{=} e(C_0, u_B^{\pi} h_B)$. It outputs 1 if the check passes. Otherwise, it outputs 0.

CP-ABE.Decrypt($CH_{\mathbb{A}}, SK_S, PP$)**:** Let $CH_{\mathbb{A}} = (C_0, \{C_{j,1}, C_{j,2}, C_{j,3}\}, C_3)$ and $SK_S = (K_0, K_1, \{K_{j,2}, K_{j,3}\})$. It first checks $1 \stackrel{?}{=}$ **CP-ABE.VerifyCT**($CH_{\mathbb{A}}, PP$). If the checking fails, then it outputs $\perp$. If $S \in \mathbb{A}$, then it computes constants $\omega_j \in \mathbb{Z}_p$ such that $\sum_{\rho(j) \in S} \omega_j M_j = (1, 0, \ldots, 0)$ and outputs a session key as $EK = e(C_0, K_0) / \prod_{\rho(j) \in S} \big(e(C_{j,1}, K_1) \cdot e(C_{j,2}, K_{j,2}) \cdot e(C_{j,3}, K_{j,3})\big)^{\omega_j}$. Otherwise, it outputs $\perp$.

Remark 4.8. *Compared to the CPA-secure CP-ABE scheme, the above CP-ABE scheme additionally contains a hash function index z and two group elements u_B, h_B in public parameters, and a group element C_3 in a ciphertext header for integrity. We also added a ciphertext verification algorithm **VerifyCT** to check the validity of ciphertext headers.*

To construct an RS-ABE scheme, we follow the design principle of Lee et al. [14]. Our RS-ABE scheme that uses the above CP-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 **CP-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 outputs a master key $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$ and public parameters $PP = \big(PP_{ABE}, PP_{SUE}, \Omega = e(g,g)^{\alpha}\big)$.

RS-ABE.GenKey(S, u, MK, PP)**:** Let $MK = (MK_{ABE}, MK_{SUE}, \alpha, \mathcal{BT})$. It first assigns the index u to a random leaf node $v_u \in \mathcal{BT}$. It obtains a private set $PV_u = \{S'_{j_0}, \ldots, S'_{j_d}\}$ by running **CS.Assign**($\mathcal{BT}, u$) and retrieves $\{\gamma_{j_0}, \ldots, \gamma_{j_d}\}$ from $\mathcal{BT}$ where S'_{j_k} is associated with a node v_{j_k} and γ_{j_k} is assigned to the node v_{j_k}. For $0 \le k \le d$, it sets $MK'_{ABE} = \gamma_{j_k}$ and obtains $SK_{ABE,k}$ by running **CP-ABE.GenKey**(S, MK'_{ABE}, PP_{ABE}). It outputs a private key $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d}\big)$.

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

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 $\perp$. It selects a random exponent $\delta \in \mathbb{Z}_p$ and obtains SK_{ABE} by running **CP-ABE.RandKey**($\delta, SK_{ABE,j}, PP_{ABE}$). It also obtains SK_{SUE} by running **SUE.RandKey**($-\delta, SK_{SUE,i}, PP_{SUE}$). It outputs a decryption key $DK_{S,T'} = \big(SK_{ABE}, SK_{SUE}\big)$.

RS-ABE.Encrypt($\mathbb{A}, T, PP$): It selects a random exponent $t \in \mathbb{Z}_p$ and obtains CH_{ABE} and CH_{SUE} by running **CP-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 **CP-ABE.Encrypt** and **SUE.Encrypt**. It outputs a ciphertext header $CH_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE})$ and a session key $EK = \Omega^t$.

RS-ABE.UpdateCT($CH_{\mathbb{A},T}, T+1, PP$): Let $CH_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE})$. It first obtains CH'_{SUE} by running **SUE.UpdateCT**$(CH_{SUE}, T+1, PP_{SUE})$. It outputs an updated ciphertext header $CH_{\mathbb{A},T+1} = (CH_{ABE}, CH'_{SUE})$.

RS-ABE.Decrypt($CH_{\mathbb{A},T}, DK_{S,T'}, PP$): Let $CH_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE})$ and $DK_{S,T'} = (SK_{ABE}, SK_{SUE})$. Let $CH_{ABE} = (C_0, \ldots)$ and $CH_{SUE} = (CH^{(0)}, \ldots)$ where $CH^{(0)} = (C'_0, \ldots)$. It first checks $C_0 = C'_0$. If the check fails, then it outputs $\perp$. If $S \in \mathbb{A}$ and $T \le T'$, then it obtains EK_{ABE} and EK_{SUE} by running **CP-ABE.Decrypt**$(CH_{ABE}, SK_{ABE}, PP_{ABE})$ and **SUE.Decrypt**$(CH_{SUE}, SK_{SUE}, PP_{SUE})$ respectively. If $EK_{ABE} \neq \perp$ and $EK_{SUE} \neq \perp$, then it outputs a session key $EK = EK_{ABE} \cdot EK_{SUE}$. Otherwise, it outputs $\perp$.

Remark 4.9. *In contrast to CPA-secure RS-ABE schemes [13, 14, 23], our RS-ABE scheme does not provide a ciphertext re-randomization algorithm **RandCT**. Because of this, the outputted (future) ciphertext header of **UpdateCT** maybe correlated to the original (past) ciphertext header. Thus, the (past) original ciphertext header should be deleted after running the **UpdateCT** algorithm to remove the correlation between ciphertext headers.*

4.4 Correctness

We first show the correctness of the above CP-ABE scheme. Compared to the original CP-ABE scheme of Rouselakis and Waters [22], the above CP-ABE scheme additionally contains elements u_B, h_B in public parameters and an element C_3 in a ciphertext header. The validity of C_3 can be easily checked by using bilinear maps since C_3 is a DDH tuple. Thus the above CP-ABE scheme is correct since the ciphertext header can pass **CP-ABE.VerifyCT** and the original CP-ABE scheme is correct.

The correctness of the above RS-ABE scheme can be shown by using the correctness of the CP-ABE scheme, SUE scheme, and CS scheme. Let $SK_{S,u}$ be a private key and $UK_{T',R}$ be an update key. If $u \notin R$, then there are SK_{ABE} in $SK_{S,u}$ and SK_{SUE} in $UK_{T,R}$ that are associated with the same node v_i by the correctness of the CS scheme. The decryption key $DK_{S,T'} = (SK'_{ABE}, SK'_{SUE})$ is obtained from SK_{ABE} and SK_{SUE} after additional randomization. Note that the master key part of SK'_{ABE} is $\gamma_i + \delta$ and the master key part of SK'_{SUE} is $\alpha - \gamma_i - \delta$. Let $CH_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE})$ be a ciphertext header. If $S \in \mathbb{A}$, then **CP-ABE.Decrypt** can derive a partial session key EK_{ABE} from the correctness of the CP-ABE scheme. If $T \le T'$, then **SUE.Decrypt** can derive a partial session key EK_{SUE} from the correctness of the SUE scheme. By multiplying two partial session keys, we obtain a valid session key since the original master key α can be derived from the master key parts of ABE and SUE.

4.5 Security Analysis

To prove the TEC-CCA security of the above RS-ABE scheme, we use the n-RW1 assumption introduced by Rouselakis and Waters [22]. Rouselakis and Waters proposed an efficient CP-ABE scheme and prove its CPA security under the n-RW1 assumption. The definition of n-RW1 assumption and the security of the original CP-ABE scheme are given as follows:

Assumption 4.10 (n-RW1, [22])**.** *Let $(p, \mathbb{G}, \mathbb{G}_T, e)$ be a description of the bilinear group of prime order p. Let g be a random generator of $\mathbb{G}$. The n-RW1 assumption is that if the challenge tuple*

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

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

Lemma 4.11 ([22])**.** *The n-RW1 assumption holds in the generic bilinear group model.*

Theorem 4.12 ([22])**.** *The original CP-ABE scheme is selectively secure under chosen plaintext attacks if the n-RW1 assumption holds where n is the number of columns in the challenge matrix.*

To prove the TEC-CCA security of the above CP-ABE scheme, we use the meta-simulation technique that uses the previous CPA simulator in Theorem 4.12 as a sub-simulator. As pointed by Lee [13], the simulator in Theorem 4.12 cannot be directly used as a sub-simulator in meta-simulation since it sets $\gamma = a^{n+1} + \gamma'$ and $w_A = g^a$. To use this simulator in meta-simulation, we modify the simulator to set $\gamma = a^{n+1}$ and $w_A = g^a g^{w'}$ by selecting a random exponent w'. Note that this modification is easy. To handle decryption queries of an adversary, we use a variation of the CHK transformation, in which a CPA secure IBE scheme can be converted to a CCA secure PKE scheme [4, 6]. The CCA security of the above CP-ABE scheme is given as follows:

Theorem 4.13. *The above CP-ABE scheme is selectively secure under chosen ciphertext attacks if the n-RW1 assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that $\boldsymbol{Adv}_{\mathcal{A}}^{ABE}(\lambda) \le \boldsymbol{Adv}_{\mathcal{B}}^{n\text{-}RW1}(\lambda) + \boldsymbol{negl}(\lambda)$ where n is the number of columns in the challenge matrix.*

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 n-RW1 assumption using $\mathcal{A}$ is given: a challenge tuple $D = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, g^c, \{g^{a^i}, g^{d_j}, g^{cd_j}, g^{a^i d_j}, g^{a^i/d_j^2}\}, \{g^{a^i/d_j}\}, \{g^{a^i d_j/d_{j'}^2}\}, \{g^{a^i cd_j/d_{j'}}, g^{a^i cd_j/d_{j'}^2}\}\big)$ and Z where $Z = Z_0 = e(g,g)^{a^{n+1}c}$ or $Z = Z_1 \in \mathbb{G}_T$. Let $\mathcal{B}_{ABE}$ be a modified simulator in the security proof of Theorem 4.12. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits a challenge access structure $\mathbb{A}^*$. $\mathcal{B}$ first runs $\mathcal{B}_{ABE}$ by giving D and Z.

Setup: $\mathcal{B}$ submits $\mathbb{A}^*$ to $\mathcal{B}_{ABE}$ and receives $PP_{ABE} = ((p, \mathbb{G}, \mathbb{G}_T, e), g, w_A, v_A, u_A, h_A, \Lambda = e(g,g)^{a^{n+1}})$. It also requests a challenge ciphertext to $\mathcal{B}_{ABE}$ and receives a challenge ciphertext header $CH_{\mathbb{A}^*} = (C_0^*, \{C_{j,1}^*, C_{j,2}^*, C_{j,3}^*\})$ and a challenge session key EK^* where $C_0^* = g^c$ and $EK^* = Z$. It selects a random index z for a hash function H^z. It computes $\pi^* = H^z(C_0, C_{1,1}, \ldots, C_{l,3})$ and sets $u_B = g^{a^q} g^{u'_B}, h_B = (g^{a^q})^{-\pi^*} g^{h'_B}$ by selecting random exponents $u'_B, h'_B \in \mathbb{Z}_p$. It implicitly sets $\gamma = a^{n+1}$ and gives the public parameters $PP = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, w_A, v_A, u_A, h_A, z, u_B, h_B, \Lambda\big)$ to $\mathcal{A}$.

Query 1: $\mathcal{A}$ adaptively requests a polynomial number of private key and decryption queries. If this is a private key query for a set of attributes S, then $\mathcal{B}$ receives a private key SK_S from $\mathcal{B}_{ABE}$ by requesting a private key query and gives SK_S to $\mathcal{A}$.

If this is a decryption query for a ciphertext header $CH_{\mathbb{A}}$, then $\mathcal{B}$ proceeds as follows:

1. Let $CH_{\mathbb{A}} = (C_0, \{C_{j,1}, C_{j,2}, C_{j,3}\}, C_3)$. It first checks the validity of $CH_{\mathbb{A}}$ by running **CP-ABE.VerifyCT** $(CH_{\mathbb{A}}, PP)$. If the ciphertext header is not valid, then it responds the query with $\perp$. Otherwise, the ciphertext header is valid and formed as $CH_{\mathbb{A}} = \big(C_0 = g^t, \{C_{j,1}, C_{j,2}, C_{j,3}\}, C_3 = (u_B^{\pi} h_B)^t\big)$ for some unknown $t \in \mathbb{Z}_p$.

2. It calculates $\pi = H^z(C_0, C_{1,1}, \ldots, C_{l,3})$. If $\pi = \pi^*$, then it terminates the simulation with $\mathcal{A}$ and outputs a random bit since it cannot response. If $\pi \neq \pi^*$, the it can use the IBE technique of Boneh and Boyen [3] to decrypt the ciphertext header. It sets $D_0 = (g^a)^{-(u'_B \pi + h'_B)/(\pi - \pi^*)} (u_B^{\pi} h_B)^{r'}$ and $D_3 = (g^a)^{-1/(\pi-\pi^*)} g^{r'}$ by selecting a random exponent $r' \in \mathbb{Z}_p$ and it computes the session key as

$$EK = e(C_0, D_0) \cdot e(C_3, D_3) = e\big(g^t, g^{a^{n+1}} (u_B^{\pi} h_B)^r\big) \cdot e\big((u_B^{\pi} h_B)^t, g^{-r}\big) = e(g,g)^{a^{n+1} t}.$$

3. It responses the query with EK as a decapsulated session key.

Challenge: $\mathcal{A}$ requests a challenge ciphertext header and a challenge session key. $\mathcal{B}$ computes $C_3^* = (g^c)^{u'_B \pi^* + h'_B}$ since $(u_B^{\pi^*} h_B)^c = (g^{a^n})^{(\pi^* - \pi^*)c} (g)^{(u'_B \pi^* + h'_B)c}$. It sets $CH^* = (C_0^*, \{C_{j,1}^*, C_{j,2}^*, C_{j,3}^*\}, C_3^*)$ and EK^* and gives the challenge tuples to $\mathcal{A}$.

Query 2: Same as Query 1. Note that $\mathcal{A}$ cannot request the decryption query on the challenge ciphertext header CH^*.

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

To finish the proof, we show that $\mathcal{B}$ can handle decryption queries correctly. The decryption of $\mathcal{B}$ only fails when $\pi = \pi^*$ even if $CH_{\mathbb{A}} \neq CH^*$. However, the probability of this collision event is negligible since H^z is a collision resistant hash function. This completes our proof. □

To prove the TEC-CCA security of the above RS-ABE scheme, we also apply the partitioning method by using the meta-simulation technique. As mentioned before, we use the CCA simulator of the CP-ABE scheme and the TEC-CCA simulator of the SUE scheme as sub-simulators to simplify the description of a reduction algorithm (meta-simulator). Compared with the security proof of Lee's RS-ABE scheme in [13], the security proof of our RS-ABE scheme shows the TEC-CCA security instead of the CPA security and proves the security in the selective model instead of the selective revocation list model. An adversary should submit a challenge access structure $\mathbb{A}^*$ and a challenge time T^* before he receives the public parameters in the selective model, whereas the adversary additionally submits a set of revoked users RL^* on the challenge time before he receives the public parameters in the selective revocation list model. The selective revocation list model was introduced by Boldyreva et al. [2] to prove the security of their revocable ABE scheme and used in other systems in [13, 14, 19].

The main idea of proving the security in the selective model instead of the selective revocation list model is to assigning a user index to a random leaf node in a binary tree and to predicting the number of the adversary's private key queries with the condition $S \in \mathbb{A}^*$ where S is a set of attributes in a private key. The meta-simulator can easily generate a private key with $S \notin \mathbb{A}^*$ since it can use the CP-ABE simulator by creating an ABE private key that contains the master key α. However, it simply cannot generate a private key with $S \in \mathbb{A}^*$ by using the CP-ABE simulator because of the restriction in the CP-ABE security model. Thus it creates a private key by setting a random γ_i in a binary tree as the master key part of CP-ABE. To preserve the consistency of private key and update key generations, the meta-simulator should know the positions of user's private keys with $S \in \mathbb{A}^*$ in a binary tree. If the number of private key with $S \in \mathbb{A}^*$ is known, then the simulator can handle the private key queries by assigning user indexes to random leaf nodes. The TEC-CCA security proof of our RS-ABE scheme is described as follows:

Theorem 4.14. *The above RS-ABE scheme is selectively secure under time extended challenge chosen ciphertext attacks if the n-RW1 assumption holds. That is, for any PPT adversary $\mathcal{A}$, we have that* $\boldsymbol{Adv}_{\mathcal{A}}^{RS\text{-}ABE}(\lambda) \leq q \cdot \boldsymbol{Adv}_{\mathcal{B}}^{n\text{-}RW1}(\lambda) + \boldsymbol{negl}(\lambda)$ *where n is the number of columns in the challenge matrix and q is the number of private key queries.*

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 n-RW1 assumption using $\mathcal{A}$ is given: a challenge tuple $D = \big((p, \mathbb{G}, \mathbb{G}_T, e), g, g^c, \{g^{a^i}, g^{d_j}, g^{cd_j}, g^{a^i d_j}, g^{a^i/d_j^2}\}, \{g^{a^i/d_j}\}, \{g^{a^i d_j/d_{j'}^2}\}, \{g^{a^i cd_j/d_{j'}}, g^{a^i cd_j/d_{j'}^2}\}\big)$ and Z where $Z = Z_0 = e(g,g)^{a^{n+1}c}$ or $Z = Z_1 \in \mathbb{G}_T$. Note that a challenge tuple $D_{DBDH} = (g, g^a, g^{a^n}, g^c)$ for the DBDH assumption can be easily derived from the challenge tuple D of the n-RW1 assumption by setting $b = a^n$. Let $\mathcal{B}_{ABE}$ be a modified simulator in the security proof of Theorem 4.13 and $\mathcal{B}_{SUE}$ be a simulator in the security proof of Theorem 3.12. Then $\mathcal{B}$ that interacts with $\mathcal{A}$ is described as follows:

Init: $\mathcal{A}$ initially submits a challenge access structure $\mathbb{A}^*$ and a challenge time T^*. $\mathcal{B}$ first runs $\mathcal{B}_{ABE}$ by giving D and Z, and it also runs $\mathcal{B}_{SUE}$ by giving D_{DBDH} and Z. Let q be the maximum number of private key queries of $\mathcal{A}$. Let $\tilde{q}$ be the number of private key queries for a set of attributes S and a user index u that satisfy $S \in \mathbb{A}^*$. $\mathcal{B}$ randomly guesses $\tilde{q}$ by selecting a random integer in $\{0, \ldots, q\}$. Note that it can correctly guess $\tilde{q}$ with $1/(q+1)$ probability. Next, it obtains $\mathcal{BT}$ by running **CS.Setup**(N_{max}) where $N_{max} \geq q$ and assigns a random exponent $\gamma_i \in \mathbb{Z}_p$ to each node $v_i \in \mathcal{BT}$. Let SN^* be a set of random leaf nodes in $\mathcal{BT}$ with $|SN^*| = \tilde{q}$. Recall that **Path**(v) is the set of path nodes from the root node to the leaf node v. That is, **Path**$(v) = \{v_{j_0}, \ldots, v_{j_d}\}$ where v_{j_0} is the root node and $v_{j_d} = v$. Let **SteinerTree**(SN^*) be the minimal subtree that connects the root node to all leaf nodes in SN^*. That is, **SteinerTree**$(SN^*) = \bigcup_{v_j \in SN^*}$ **Path**(v_j).
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 queries an ABE challenge ciphertext header to $\mathcal{B}_{ABE}$ and receives CH^*_{ABE} and EK^*_{ABE}. It also queries an SUE challenge ciphertext header to $\mathcal{B}_{SUE}$ and receives CH^*_{SUE} and EK^*_{SUE}. It randomizes Λ of PP_{ABE} and Λ of PP_{SUE} by selecting random exponents $\gamma', \beta' \in \mathbb{Z}_p$. It implicitly sets $\alpha = a^{n+1}$ and gives the public parameters $PP = \big(PP_{ABE}, PP_{SUE}, \Omega = e(g^a, g^{a^n})\big)$ to $\mathcal{A}$.
Query 1: $\mathcal{A}$ adaptively requests a polynomial number of private key, update key, decryption key, and decryption 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** $S \in \mathbb{A}^*$: In this case, it can creates ABE private keys for path nodes by using γ_i of $\mathcal{BT}$ for the master key of ABE.

  1. If there is an unassigned leaf node in SN^*, then it randomly assigns a leaf node $v_u \in SN^*$ to u. Otherwise, it aborts the simulation since it failed to guess $\tilde{q}$.
  2. It obtains PV_u by running **CS.Assign**$(\mathcal{BT}, u)$. Let **Path**$(v_u) = \{v_{j_0}, \ldots, v_{j_d}\}$ be the set of nodes that is associated with PV_u where v_u is the leaf node assigned to u and $v_{j_d} = v_u$. It retrieves exponents $\{\gamma_{j_0}, \ldots, \gamma_{j_d}\}$ from $\mathcal{BT}$ that are associated with **Path**(v_u).
  3. For all $v_{j_k} \in$ **Path**(v_u), it obtains $SK_{ABE,k}$ by running **CP-ABE.GenKey**$(S, \gamma_{j_k}, PP_{ABE})$
  4. It responses the query with the private key $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \ldots, SK_{ABE,d}\big)$.

- **Case** $S \notin \mathbb{A}^*$: 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 randomly assigns a leaf node $v_u \notin SN^*$ to u.

2. It obtains PV_u by running **CS.Assign**$(\mathcal{BT}, u)$. Let **Path**$(v_u) = \{v_{j_0}, \dots, v_{j_d}\}$ be the set of nodes that is associated with PV_u where v_u is the leaf node assigned to u and $v_{j_d} = v_u$. It retrieves exponents $\{\gamma_{j_0}, \dots, \gamma_{j_d}\}$ from $\mathcal{BT}$ that are associated with **Path**(v_u).
3. It queries an ABE private key for S to $\mathcal{B}_{ABE}$ and receives SK'_S.
4. For each $v_{j_k} \in$ **Path**(v_u), it performs the following steps: If $v_{j_k} \in$ **SteinerTree**(SN^*), then it obtains $SK_{ABE,k}$ by running **CP-ABE.GenKey**$(S, \gamma_{j_k}, PP_{ABE})$. Otherwise, it obtains $SK_{ABE,k}$ by running **CP-ABE.RandKey**$(SK'_S, -\gamma_{j_k}, PP_{ABE})$.
5. It responses the query with the private key $SK_{S,u} = \big(PV_u, SK_{ABE,0}, \dots, SK_{ABE,d}\big)$.

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 $T < T^*$.

  1. It obtains CV_R by running **CS.Cover**$(\mathcal{BT}, R)$. Let **Cover**$(R) = \{v_{i_1}, \dots, v_{i_m}\}$ be the set of nodes that is associated with CV_R. It retrieves exponents $\{\gamma_{i_1}, \dots, \gamma_{i_m}\}$ from $\mathcal{BT}$ that are associated with **Cover**(R).
  2. It queries an SUE private key for T to $\mathcal{B}_{SUE}$ and receives SK'_{SUE}.
  3. For each $v_{i_k} \in$ **Cover**(R), it performs the following steps: If $v_{i_k} \in$ **SteinerTree**(SN^*), then it obtains $SK_{SUE,k}$ by running **SUE.RandKey**$(SK'_{SUE}, -\gamma_{i_k}, PP_{SUE})$. Otherwise, it obtains $SK_{SUE,k}$ by running **SUE.GenKey**$(T, \gamma_{i_k}, PP_{SUE})$.
  4. It responses the query with the update key $UK_{T,R} = (CV_R, SK_{SUE,1}, \dots, SK_{SUE,m})$.

- **Case** $T \geq T^*$: In this case, it can create SUE private keys by using γ_i for the master key of SUE. Let R^* be the set of revoked users on the time T^* and RN^* be the set of revoked leaf nodes on the time T^*. We first have $SN^* \subseteq RN^*$ since a revealed private key for $S \in \mathbb{A}^*$ should be revoked at some time $T \leq T^*$. We also have **SteinerTree**$(RN^*) \cap$ **Cover**$(R) = \emptyset$ since $R^* \subseteq R$ if $T \geq T^*$. Thus, we have **SteinerTree**$(SN^*) \cap$ **Cover**$(R) = \emptyset$ if $T \geq T^*$.

  1. If $T = T^*$, then it counts the number of leaf nodes q' in RN^* that satisfy $S \in \mathbb{A}^*$ and stops the simulation if $q' \neq \tilde{q}$ since it failed to guess q'.
  2. It obtains CV_R by running **CS.Cover**$(\mathcal{BT}, R)$. Let **Cover**$(R) = \{v_{i_1}, \dots, v_{i_m}\}$ be the set of nodes that is associated with CV_R. It retrieves exponents $\{\gamma_{i_1}, \dots, \gamma_{i_m}\}$ from $\mathcal{BT}$ that are associated with **Cover**(R).
  3. For each $v_{i_k} \in$ **Cover**(R), it obtains $SK_{SUE,k}$ by running **SUE.GenKey**$(T, \gamma_{i_k}, PP_{SUE})$.
  4. It responses the query with the update key $UK_{T,R} = (CV_R, SK_{SUE,1}, \dots, SK_{SUE,m})$.

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 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 **CP-ABE.RandKey** $(SK'_S, -\delta, PP_{ABE})$ and **SUE.GenKey**(T, δ, PP_{SUE}) respectively.
  3. It responds the query with the decryption key $DK_{S,T} = \big(SK_{ABE}, SK_{SUE}\big)$.

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

    1. It 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 **CP-ABE.GenKey** (S, δ, PP_{ABE}) and **SUE.RandKey**$(SK'_{SUE}, -\delta, PP_{SUE})$ respectively.
    3. It responds the query with the decryption key $DK_{S,T} = \big(SK_{ABE}, SK_{SUE}\big)$.

If this is a decryption query for a ciphertext header $CH_{\mathbb{A},T} = (CH_{ABE}, CH_{SUE})$, then $\mathcal{B}$ proceeds as follows:

1. Let $CH_{ABE} = (C_0, \ldots)$ and $CH_{SUE} = (CH^{(0)}, \ldots)$ where $CH^{(0)} = (C'_0, \ldots)$. It first checks $C_0 = C'_0$. If the check fails, then it responds with $\perp$. It checks the validity of CH_{ABE} and CH_{SUE} by running **CP-ABE.VerifyCT**(CH_{ABE}, PP_{ABE}) and **SUE.VerifyCT**(CH_{SUE}, T, PP_{SUE}) respectively. If two ciphertext headers are not valid, then it responds with $\perp$.

2. If $CH_{ABE} \neq CH^*_{ABE}$, then it queries the decryption of CH_{ABE} to $\mathcal{B}_{ABE}$ and receives EK. Otherwise, it queries the decryption of CH_{SUE} to $\mathcal{B}_{SUE}$ and receives EK.

3. It responses the query with EK as a decapsulated session key.

Challenge: $\mathcal{A}$ requests a challenge ciphertext header and a challenge session key for $\mathbb{A}^*$ and T^*. It sets the challenger ciphertext header $CH^* = (CH^*_{ABE}, CH^*_{SUE})$ and the challenge session key $EK^* = Z$. Recall that CH^*_{ABE} and CH^*_{SUE} were received from sub-simulators $\mathcal{B}_{ABE}$ and $\mathcal{B}_{SUE}$ at the setup stage. It gives CH^* and EK^* to $\mathcal{A}$.
Query 2: Same as Query 1.
Guess: $\mathcal{A}$ outputs a guess μ'. $\mathcal{B}$ also outputs μ'.

To finish the proof, we show that the meta-simulator $\mathcal{B}$ correctly handles the queries of $\mathcal{A}$. We first show that private keys are correctly generated. A user with an index u is randomly assigned to a unique leaf node v_u in $\mathcal{BT}$ and the private key of the user consists of ABE private keys, in which each ABE private key is associated with a node v_{j_k} in path nodes from the root node to the leaf node. If $v_{j_k} \in$ **SteinerTree**(SN^*), an ABE private key for v_{j_k} is generated by setting γ_{j_k} as the master key of ABE. Otherwise, an ABE private key for v_{j_k} is generated by setting $\alpha - \gamma_{j_k}$ as the master key of ABE. If $S \in \mathbb{A}^*$ where S is the set of attributes in a private key, then $\mathcal{B}$ simply uses γ_{j_k} stored in $\mathcal{BT}$ although it cannot use $\mathcal{B}_{ABE}$. If $S \notin \mathbb{A}^*$, then $\mathcal{B}$ can use $\mathcal{B}_{ABE}$ to generate an ABE private key with the master key α and then it can later add $-\gamma_{j_k}$ in the master key part. Thus, ABE private keys for private key generation are consistently generated depending on the condition $v_{j_k} \in$ **SteinerTree**(SN^*).

We next show that update keys are correctly generated by presenting that the master key parts of SUE private keys in update keys are consistent with those of ABE private keys in private keys. An update key for T and R consists of SUE private keys, in which each SUE private key is associated with a node v_{i_k} in **Cover**(R). If $v_{i_k} \in$ **SteinerTree**(SN^*), then an SUE private key should be generated by setting $\alpha - \gamma_{i_k}$ as the master key of SUE. If $v_{i_k} \notin$ **SteinerTree**(SN^*), then an SUE private key should be generated by setting γ_{i_k} as the master key of SUE. In this case, the consistency of private keys and update keys is preserved since the master key α can be derived from the master keys $\alpha - \gamma_{j_k}$ and γ_{j_k} of ABE private key and SUE private key for the same node v_{j_k}. If $T < T^*$, then $\mathcal{B}$ can easily generate SUE private keys since it can use $\mathcal{B}_{SUE}$. If $T \geq T^*$, then $\mathcal{B}$ should generate SUE private keys by using γ_{i_k} stored in $\mathcal{BT}$ since it cannot use $\mathcal{B}_{SUE}$ from the restriction of SUE. However, there is no problem to generate SUE private keys from the fact that

Cover$(R) \cap$ **SteinerTree**$(SN^*) = \emptyset$ if $T \geq T^*$. Thus, SUE private keys are also consistent with ABE private keys.

Decryption keys are correctly generated since $\mathcal{B}$ can use $\mathcal{B}_{ABE}$ if $S \notin \mathbb{A}^*$ or $\mathcal{B}_{SUE}$ if $T < T^*$ by the restriction of the security model. The decryption queries are also correctly handled since both $\mathcal{B}_{ABE}$ and $\mathcal{B}_{SUE}$ can handle their own decryption queries with non-negligible probability. The challenge ciphertext header is correctly generated since CH^*_{ABE} and CH^*_{SUE} are generated by $\mathcal{B}_{ABE}$ and $\mathcal{B}_{SUE}$ respectively and two sub-simulators set $C_0 = g^c$. This completes our proof. □

4.6 Discussions

Efficiency Analysis. Our RS-ABE scheme is similar to the RS-ABE scheme of Lee [13] in terms of efficiency except that the size of public parameters is increased and pairing operations are added to check the validity of the ciphertext header in our RS-ABE scheme. That is, our RS-ABE scheme has $O(\log T_{max})$ group elements in public parameters, $O(\log N_{max} * |S|)$ group elements in a private key, $O(r \log(N_{max}/r) * \log T_{max})$ group elements in an update key, and $O(l + \log T_{max})$ group elements in a ciphertext header. The decryption algorithm of SUE requires $O(|S| + \log T_{max})$ pairing operations since the SUE ciphertext verification can be done in $O(\log T_{max})$ pairing operations.

Key-Policy ABE. Our RS-ABE scheme combines the CP-ABE scheme of Rouselakis and Waters [22] and our SUE scheme. There is another kind of ABE, called key-policy ABE (KP-ABE) in which a private key is associated with an access structure and a ciphertext is associated with a set of attributes [12]. We can build a TEC-CCA secure RS-ABE scheme with key-policy by using a CCA secure KP-ABE scheme instead of using a CCA secure CP-ABE scheme. For instance, a CCA secure KP-ABE scheme can be derived from the KP-ABE scheme of Rouselakis and Waters [22]. This RS-ABE scheme with key-policy can be proven to be selectively TEC-CCA secure under a q-type assumption. We omit the details of the construction and the security proof.

Revocable ABE. A revocable ABE (R-ABE) scheme is an ABE scheme with user revocation. Boldyreva et al. [2] introduced the concept of R-ABE and they presented an R-ABE with key-policy and claimed it security in the selective revocation list model. R-ABE is a special type of RS-ABE since R-ABE only supports user revocation whereas RS-ABE supports both user revocation and ciphertext updating. Thus an R-ABE scheme can be built by combining an ABE scheme, an IBE scheme, and the CS scheme. Note that an SUE scheme for RS-ABE is replaced by an IBE scheme for R-ABE. Boldyreva et al. [2] originally claimed that their R-ABE scheme can be secure in the selective model, but they later corrected it by claiming that their R-ABE scheme is secure in the selective revocation list model, in which an adversary should submit a challenge set of attributes, a challenge time, and a set of revoked users on the challenge time [2]. If we use the proof technique of our RS-ABE scheme, we can prove the security of the R-ABE scheme in the selective model instead of the selective revocation list model.

5 Conclusion

In this paper, we focused on the CCA security of SUE and RS-ABE since previous SUE and RS-ABE schemes only provide CPA security. In the first part of this work, we defined TEC-CCA security for SUE, and then we proposed an efficient SUE scheme and proved its TEC-CCA security under the DBDH assumption. In the second part of this work, we also defined TEC-CCA security for RS-ABE and proposed a TEC-CCA-secure RS-ABE scheme in the selective model instead of the selective revocation list model. Our SUE and RS-ABE schemes are the first constructions that achieve (relaxed) CCA security.

There are many interesting problems. The first one is to construct SUE and RS-ABE schemes that support (perfect) ciphertext re-randomization. As mentioned before, encryption schemes with ciphertext re-randomization cannot be proven in TEC-CCA security model since there is no relation between a re-randomized ciphertext and the original one. The second one is to build an RS-ABE scheme that supports a designated entity who can update ciphertexts by using his private key. Note that the designated entity can update the ciphertexts, but he cannot decrypt the ciphertexts. In cloud storage, we may require the cloud sever only to update the ciphertexts instead anyone to update the ciphertexts.

References

[1] Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In Lars R. Knudsen, editor, *Advances in Cryptology - EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 83–107. Springer, 2002.

[2] 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. Full version available at `http://eprint.iacr.org/2012/052`.

[3] 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.

[4] Xavier Boyen, Qixiang Mei, and Brent Waters. Direct chosen ciphertext security from identity-based techniques. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *ACM Conference on Computer and Communications Security*, pages 320–329. ACM, 2005.

[5] 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.

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

[7] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *CCS 2007*, pages 185–194. ACM, 2007.

[8] Ran Canetti, Hugo Krawczyk, and Jesper Buus Nielsen. Relaxing chosen-ciphertext security. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 565–582. Springer, 2003.

[9] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, 1998.

[10] Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay. Fully secure self-updatable encryption in prime order bilinear groups. In Sherman S. M. Chow, Jan Camenisch, Lucas Chi Kwong Hui, and Siu-Ming Yiu, editors, *ISC 2014*, volume 8783 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2014.

[11] 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.

[12] 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.

[13] Kwangsu Lee. Self-updatable encryption with short public parameters and its extensions. *Designs Codes Cryptogr.*, 2015. `http://dx.doi.org/10.1007/s10623-015-0039-9`.

[14] 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.

[15] 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`.

[16] 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.

[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] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, pages 427–437. ACM, 1990.

[19] Seunghwan Park, Kwangsu Lee, and Dong Hoon Lee. New constructions of revocable identity-based encryption from multilinear maps. *IEEE Trans. Inf. Forensic Secur.*, 10(8):1564–1577, 2015.

[20] Charles Rackoff and Daniel R. Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In Joan Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 433–444. Springer, 1991.

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

[22] 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.

[23] 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.

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

[25] 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.

[26] 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.

[27] Victor Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. `http://eprint.iacr.org/2001/112`.

[28] 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.

[29] Rui Zhang, Goichiro Hanaoka, Junji Shikata, and Hideki Imai. On the security of multiple encryption or cca-security+cca-security=cca-security? In Feng Bao, Robert H. Deng, and Jianying Zhou, editors, *PKC 2004*, volume 2947 of *Lecture Notes in Computer Science*, pages 360–374. Springer, 2004.