# Fully Secure Self-Updatable Encryption in Prime Order Bilinear Groups[*]

Pratish Datta, Ratna Dutta, and Sourav Mukhopadhyay

Department of Mathematics
Indian Institute of Technology Kharagpur
Kharagpur-721302, India
{pratishdatta,ratna,sourav}@maths.iitkgp.ernet.in

**Abstract.** In CRYPTO 2012, Sahai et al. raised the concern that in a cloud control system revocation of past keys should also be accompanied by updation of previously generated ciphertexts in order to prevent unread ciphertexts from being read by revoked users. Self-updatable encryption (SUE), introduced by Lee et al. in ASIACRYPT 2013, is a newly developed cryptographic primitive that realizes ciphertext update as an inbuilt functionality and thus improves the efficiency of key revocation and time evolution in cloud management. In SUE, a user can decrypt a ciphertext associated with a specific time if and only if the user possesses a private key corresponding to either the same time as that of the ciphertext or some future time. Furthermore, a ciphertext attached to a certain time can be updated to a new one attached to a future time using only public information. The SUE schemes available in the literature are either (a) fully secure but developed in a composite order bilinear group setting under highly non-standard assumptions or (b) designed in prime order bilinear groups but only selectively secure. This paper presents the *first fully secure* SUE scheme in *prime order bilinear groups* under *standard assumptions*, namely, the Decisional Linear and the Decisional Bilinear Diffie-Hellman assumptions. As pointed out by Freeman (EUROCRYPT 2010) and Lewko (EUROCRYPT 2012), the communication and storage, as well as, computational efficiency of prime order bilinear groups are much higher compared to that of composite order bilinear groups with an equivalent level of security. Consequently, our SUE scheme is highly cost-effective than the existing fully secure SUE.

**Keywords:** public-key encryption, self-updatable encryption, ciphertext update, prime order bilinear groups, cloud storage.

## 1 Introduction

Cloud storage is gaining popularity very rapidly in recent years due to its lower cost of service, easier data management facility and, most importantly, the accessibility of stored data through Internet from any geographic location. However, since these databases are often filled with oversensitive information, these are prime targets of attackers and security breaches in such systems are not uncommon, especially by insiders of the organizations maintaining the cloud servers. In particular, access control is one of the greatest concerns, i.e., the sensitive data items have to be protected from any illegal access, whether it comes from outsiders or even from insiders without proper access rights. Additionally, organizations storing extremely sensitive data to an external cloud server might not want to give the server any access to their information at all. Similar problems can easily arise when dealing with centralized storage within an organization, where different users in different departments have access to varying levels of sensitive data.

One possible approach for this problem is to use attribute-based encryption (ABE) that provides cryptographically enhanced access control functionality in encrypted data [12]. However, in a cloud storage data access is not static. To deal with the change of users' credentials that takes place over time, revocable ABE (R-ABE) [1] has been suggested in which a user's private key can be revoked. In R-ABE, a revoked user is restricted from learning any partial information about the messages encrypted when the ciphertext is created after the time of revocation. Sahai

[*] This is the full version of the paper that appeared in Proceedings of the 17th Information Security Conference (ISC 2014), LNCS 8783, pp. 1–18, Springer.

et al. [15] highlighted the fact that R-ABE alone does not suffice in managing dynamic credentials for cloud storage. In fact, R-ABE cannot prevent a revoked user from accessing ciphertexts that were created before the revocation, since the old private key of the revoked user is enough to decrypt those ciphertexts. Thus a complete solution has to support not only the revocation functionality but also the ciphertext update functionality such that a ciphertext at any arbitrary time can be updated to a new ciphertext at a future time by the cloud server just using publicly available information and thereby making previously stored data inaccessible to revoked users.

*Self-updatable encryption* (*SUE*), introduced by Lee et al. [10], is a newly developed cryptographic primitive that realizes ciphertext update as an inbuilt functionality and thus improves the efficiency of key revocation and time evolution in cloud management. In SUE, a ciphertext and a private key are associated with specific times. A user who has a private key with time $T'$ can decrypt a ciphertext with time $T$ if and only if $T \leq T'$. Additionally, a ciphertext attached to a particular time can be updated to a new ciphertext attached to a future time just using public values.

SUE is related to the notion of forward secure encryption (FSE) [5]. In FSE, a private key associated with a time is evolved to another private key with the immediate subsequent time, and then the past private key is erased. Forward security ensures that an exposed private key with a time cannot be used to decrypt a past ciphertext. In addition to satisfying a similar objective, SUE offers more flexibility in the sense that SUE supports one-to-many communication and thus is suitable for constructing more complex cryptographic primitives by combining with ABE or predicate encryption (PE) [10]. We may view SUE as the dual concept of FSE since the role of private keys and that of the ciphertexts are reversed.

Being an encryption scheme with enhanced time control mechanism, SUE can be applied in various other cryptographic primitives such as time-release encryption [14] and key-insulated encryption [6] to provide better security guarantees.

It is worth observing that there are simple inefficient ways (without the independence guarantee) to realize SUE assuming a vanilla identity based encryption (IBE) scheme [16]. For instance, we can view time periods as identities. An SUE ciphertext for a particular time $T$ is constructed as the collection of IBE ciphertexts for the message corresponding to all times $T'$ in the range $[T, T_{max}]$, where $T_{max}$ is the maximum time bound in the system. An SUE private key attached to a specific time $T''$ consists of IBE private keys for all time periods upto $T''$. The ciphertext update procedure simply deletes the IBE ciphertext for the least time period contained in an SUE ciphertext. However, in this approach the ciphertext and key sizes depend polynomially on $T_{max}$. Constructing an SUE scheme in which the ciphertext and private key sizes are *polylogarithmic* in terms of the maximum time period in the system is not straightforward. In particular, this requires managing the time structure in some suitable manner and design the decryption policy, as well as, the ciphertext update functionality accordingly.

**Related Works**: Lee et al. [10] constructed a fully secure SUE scheme in composite order bilinear group under non-standard assumptions. To the best of our knowledge, this scheme is so far the only SUE scheme in the literature that achieves full security. However, as pointed out by Freeman [7] and Lewko [11], group operations and pairing computations are prohibitively slow in composite order bilinear groups as compared to prime order bilinear groups with an equivalent level of security. Although, SUE in prime order bilinear groups are proposed in [10], [9] under standard assumptions those are only selectively secure. We note that the work of Sahai et al. [15] can be viewed as a restricted form of SUE but their constructions are highly customized for ABE. In particular, they first added ciphertext delegation to ABE, and then, represented time as a set of attributes, and by doing so reduced ciphertext update to ciphertext delegation. Also in their system all legitimate users are given keys corresponding to the same time and they can decrypt ciphertexts attached to the same time period as their keys. Further, their systems have composite order bilinear group setting.

**Our Contribution**: We present the *first* fully secure cost effective SUE scheme in prime order bilinear groups under standard assumptions. As in [10], [9], in order to make the ciphertext and private key sizes of our SUE scheme polylogarithmic in terms of the maximum time period supported by the system, we employ a full binary tree to efficiently manage the time structure in the system. We assign a time value to each node of the full binary tree. The ciphertext associated with a particular time consists of a set of components corresponding to some specific nodes in the tree. When a user with a private key associated with a particular time $T'$ attempts to decrypt a ciphertext attached to a certain time $T$, it can encounter a node associated with a component in the target ciphertext on the path from the root node to the node corresponding to the time $T'$ if and only if $T \leq T'$ and hence the decryption is successful if and only if $T \leq T'$. To design our SUE scheme, we start from the hierarchical identity based encryption (HIBE) scheme of Waters [16] and exchange the private key structure with the ciphertext structure of this HIBE scheme. This is not a trivial task. Unlike [10], [9], the decryption algorithm of our SUE scheme does not involve repeated application of the ciphertext update algorithm to convert a ciphertext associated with a previous time to a ciphertext corresponding to the time associated with the private key used for decryption, resulting in a much faster decryption.

Specifically, the salient features of our SUE scheme lie in the following two aspects:

- *Firstly*, our scheme is proven fully secure under two well-studied assumptions in prime order bilinear groups, namely, the Decisional Linear assumption and the Decisional Bilinear Diffie-Hellman assumption. We employ the dual system encryption methodology introduced by Lewko and Waters [13], [16] for proving security of our SUE scheme. In a dual system, there are two kinds of keys and ciphertexts: normal and semi-functional. Normal keys and ciphertexts are used in the real system, while their semi-functional counter parts are only invoked in the proof. These objects must be constructed satisfying certain relationships. Two crucial properties of the underlying bilinear group that are exploited in proofs employing the dual system encryption technique are *canceling* [7] and *parameter hiding* [11]. These features are achieved naturally when the underlying bilinear group is of composite order by the orthogonality property of the subgroups and the Chinese Remainder theorem. Replicating these properties in prime order groups is quite challenging [7], [11], which makes the construction of SUE in prime order bilinear groups rather difficult. To overcome the difficulty, we construct the SUE ciphertexts and private keys in such a way that we can associate additional random spaces to the semi-functional form of one that is orthogonal to the normal form of the other and vice versa. This approach of achieving the required relations between the normal ciphertexts and private keys with their semi-functional counter parts is quite different from the generic tools developed in [7], [11] and is one of the major contribution of this work.
- *Secondly*, our SUE scheme outperforms the existing fully secure SUE scheme [10] in terms of both communication and computation. In our SUE construction, a ciphertext consists of at most $5 \log T_{max} + 7$ group elements together with $2 \log T_{max}$ elements of $\mathbb{Z}_p$ and a private key involves at most $\log T_{max} + 7$ group elements together with $\log T_{max}$ elements of $\mathbb{Z}_p$, where $T_{max}$ is the maximum time period in the system and $p$ is the order of the bilinear group. At a first glance, these values seem worse than the corresponding values of $3 \log T_{max} + 2$ and $\log T_{max} + 2$ group elements respectively for [10]. However, the important point to observe here is that the order of a composite order bilinear group used in [10] must be at least 1024 bits in order to prevent factorization of the group order, whereas, the size of a prime order bilinear group used in our design that provides an equivalent level of security is only 160 bits [7] which is more than six times smaller. Thus, when compared in terms of bit length, both the ciphertext and key sizes of our SUE scheme are roughly one-third of the corresponding sizes for [10]. The prime order bilinear group has a similar advantage over composite order group in terms of computation as well [7], [11]. For this reason, all the algorithms in our

scheme are much faster than [10]. For instance, the decryption algorithm of our SUE scheme is roughly 25 times faster than that of [10] and the ciphertext update algorithm is more than 3 times faster than that of [10].

## 2 Preliminaries

A function $\epsilon$ is *negligible* if, for every integer $c$, there exists an integer $K$ such that for all $k > K$, $|\epsilon(k)| < 1/k^c$. A problem is said to be *computationally hard* (or *intractable*) if there exists no probabilistic polynomial time ($\mathsf{PPT}$) algorithm that solves it with non-negligible probability (in the size of the input or the security parameter).

### 2.1 Self-Updatable Encryption

As introduced in [10], self-updatable encryption (SUE) is a variant of public key encryption (PKE) with the ciphertext updating property such that a time is associated with private keys and ciphertexts, and a ciphertext with a time can be easily updated to a new ciphertext with a future time. In SUE, the private key of a user is associated with a time $T'$ and a ciphertext is also associated with a time $T$. If $T \leq T'$, then a user who has a private key with a time $T'$ can decrypt a ciphertext with a time $T$, i.e., a user who has a private key for a time $T'$ can decrypt any ciphertext attached a past time $T$ such that $T \leq T'$, but it cannot decrypt a ciphertext attached a future time $T$ such that $T' < T$. Additionally, the SUE scheme has the ciphertext update algorithm through which an semi-trusted third party, employed for managing users' access on ciphertexts by incrementing the time component, updates a ciphertext attached the time $T$ to a new ciphertext attached the time $T+1$ by using only public parameters.

**Syntax of Self-Updatable Encryption**: A self-updatable encryption (SUE) scheme consists of the following $\mathsf{PPT}$ algorithms:

**SUE.Setup**$(1^\lambda, T_{max})$: The key generation center takes as input a security parameter $1^\lambda$ and the maximum time $T_{max}$. It publishes public parameters $\mathsf{PP}$ and generates a master secret key $\mathsf{MK}$ for itself.

**SUE.GenKey**$(T', \mathsf{MK}, \mathsf{PP})$: On input a time $T'$, the master secret key $\mathsf{MK}$ together with the public parameters $\mathsf{PP}$, the key generation center outputs a private key $\mathsf{SK}_{T'}$ for a user.

**SUE.Encrypt**$(T, \mathsf{PP})$: Taking as input a time $T$ and the public parameters $\mathsf{PP}$, the encryptor creates a ciphertext header $\mathsf{CH}_T$ and a session key $\mathsf{EK}$. It sends $\mathsf{CH}_T$ to the ciphertext storage and keeps $\mathsf{EK}$ secret to itself.

**SUE.ReEncrypt**$(\mathsf{CH}_T, T+1, \mathsf{PP})$: On input a ciphertext header $\mathsf{CH}_T$ for a time $T$, the next time $T+1$ and the public parameters $\mathsf{PP}$, the semi-trusted third party, employed for managing users' access on ciphertext by updating the time component, publishes a ciphertext header for time $T+1$ and a partial session key that will be combined with the session key of $\mathsf{CH}_T$ to produce a new session key. The re-encryption procedure may be realized by sequentially executing the following two algorithms although it is not mandatary.

**SUE.UpdateCT**$(\mathsf{CH}_T, T+1, \mathsf{PP})$: The semi-trusted third party managing users' access on ciphertexts takes as input a ciphertext header $\mathsf{CH}_T$ for a time $T$, a next time $T+1$ together with the public parameters $\mathsf{PP}$, and it outputs an updated ciphertext header $\mathsf{CH}_{T+1}$.

**SUE.RandCT**$(\mathsf{CH}_T, \mathsf{PP})$: Taking input a ciphertext header $\mathsf{CH}_T$ for a time $T$ and the public parameters $\mathsf{PP}$, the semi-trusted ciphertext access managing third party outputs a re-randomized ciphertext header $\widetilde{\mathsf{CH}}_T$ and a partial session key $\widetilde{\mathsf{EK}}$ that will be combined with the session key $\mathsf{EK}$ of $\mathsf{CH}_T$ to produce a re-randomized session key.

**SUE.Decrypt**$(\mathsf{CH}_T, \mathsf{SK}_{T'}, \mathsf{PP})$: A user takes as input a ciphertext header $\mathsf{CH}_T$, its private key $\mathsf{SK}_{T'}$ together with the public parameters $\mathsf{PP}$, and computes either a session key $\mathsf{EK}$ or the distinguished symbol $\perp$.

**Correctness**: The correctness property of SUE is defined as follows: For all $\mathsf{PP}, \mathsf{MK}$ generated by **SUE.Setup**, all $T, T'$, any $\mathsf{SK}_{T'}$ output by **SUE.GenKey**, and any $\mathsf{CH}_T, \mathsf{EK}$ returned by **SUE.Encrypt** or **SUE.ReEncrypt**, it is required that:

- If $T \leq T'$, then **SUE.Decrypt**$(\mathsf{CH}_T, \mathsf{SK}_{T'}, \mathsf{PP}) = \mathsf{EK}$.
- If $T > T'$, then **SUE.Decrypt**$(\mathsf{CH}_T, \mathsf{SK}_{T'}, \mathsf{PP}) = \perp$ with all but negligible probability.

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

**Setup**: $\mathcal{B}$ runs **SUE.Setup**$(1^\lambda, T_{max})$ to generate the public parameters $\mathsf{PP}$ and the master secret key $\mathsf{MK}$, and it gives $\mathsf{PP}$ to $\mathcal{A}$.

**Query 1**: $\mathcal{A}$ may adaptively request a polynomial number of private keys for times $T'_1, \ldots, T'_{q'} \in [0, T_{max}]$, and $\mathcal{B}$ gives the corresponding private keys $\mathsf{SK}_{T'_1}, \ldots, \mathsf{SK}_{T'_{q'}}$ to $\mathcal{A}$ by executing **SUE.GenKey**$(T'_i, \mathsf{MK}, \mathsf{PP})$.

**Challenge**: $\mathcal{A}$ outputs a challenge time $T^* \in [0, T_{max}]$ subject to the following restriction: For all times $\{T'_i\}$ of private key queries, it is required that $T'_i < T^*$. $\mathcal{B}$ chooses a random bit $\beta \in \{0,1\}$ and computes a ciphertext header $\mathsf{CH}_{T^*}$ and a session key $\mathsf{EK}^*$ by performing **SUE.Encrypt**$(T^*, \mathsf{PP})$. If $\beta = 0$, then it gives $\mathsf{CH}_{T^*}$ and $\mathsf{EK}^*$ to $\mathcal{A}$. Otherwise, it gives $\mathsf{CH}_{T^*}$ and a random session key to $\mathcal{A}$.

**Query 2**: $\mathcal{A}$ may continue to request private keys for additional times $T'_{q'+1}, \ldots, T'_q \in [0, T_{max}]$ subject to the same restriction as before, and $\mathcal{B}$ gives the corresponding private keys to $\mathcal{A}$.

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

The advantage of $\mathcal{A}$ is defined as $\mathsf{Adv}^{\mathrm{SUE}}_{\mathcal{A}}(\lambda) = |\Pr[\beta = \beta'] - 1/2|$ where the probability is taken over all the randomness of the game.

**Definition 1 (Security of SUE).** *An SUE scheme is fully secure under a chosen plaintext attack* (CPA) *if for all* PPT *adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in the above game, i.e.,* $\mathsf{Adv}^{\mathrm{SUE}}_{\mathcal{A}}(\lambda)$ *is negligible in the security parameter $\lambda$.*

*Remark 1.* In the above security game, it is not needed to explicitly describe **SUE.ReEncrypt** since the adversary can run **SUE.UpdateCT** to the challenge ciphertext header by just using PP. Note that the use of **SUE.UpdateCT** does not violate the security game since the adversary only can request a private key query for $T'_i$ such that $T'_i < T^*$.

*Remark 2.* Note that, like [10], we consider only CPA security in this paper. The re-randomizability guarantee of the SUE ciphertexts makes the security against adaptive chosen ciphertext attack (CCA2) unattainable as the challenge ciphertext can simply be re-randomized and sent to the decryption oracle. An analysis for the possible relaxations of CCA2 security in the above security model is an interesting future direction of research.

### 2.2 Bilinear Groups of Prime Order and Complexity Assumptions

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

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

We say that $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, e)$ is a bilinear group if the group operation in $\mathbb{G}$ and the bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ are both efficiently computable.

**Definition 2. [Decisional Bilinear Diffie-Hellman (DBDH)]**: *Given $(\mathcal{D}_1 = (g, g^{c_1}, g^{c_2}, g^{c_3}), z) \in \mathbb{G}^4 \times \mathbb{G}_T$ as input for random generator $g \in \mathbb{G}$ and random exponents $c_1, c_2, c_3 \in \mathbb{Z}_p$, the* DBDH *assumption holds if it is computationally hard to decide whether $z = e(g, g)^{c_1 c_2 c_3}$. Formally, we define the advantage of an algorithm $\mathcal{B}$ in solving the* DBDH *problem as follows:*

$$\mathsf{Adv}_{\mathcal{B}}^{\mathrm{DBDH}}(\lambda) = |\mathsf{Pr}[\mathcal{B}(\mathcal{D}_1, z = e(g, g)^{c_1 c_2 c_3}) = 1] - \mathsf{Pr}[\mathcal{B}(\mathcal{D}_1, z \text{ random in } \mathbb{G}_T) = 1]|$$

*The* DBDH *assumption holds if $\mathsf{Adv}_{\mathcal{B}}^{\mathrm{DBDH}}(\lambda)$ is negligible for any* PPT *$\mathcal{B}$.*

**Definition 3. [Decisional Linear (DLIN)]**: *Given $(\mathcal{D}_2 = (g, f, \nu, g^{c_1}, f^{c_2}), z) \in \mathbb{G}^6$ as input for random generators $g, f, \nu \in \mathbb{G}$ and random exponents $c_1, c_2 \in \mathbb{Z}_p$, the* DLIN *assumption holds if it is computationally hard to decide whether $z = \nu^{c_1 + c_2}$. Formally, the advantage of an algorithm $\mathcal{B}$ in solving the* DLIN *problem is defined as follows:*

$$\mathsf{Adv}_{\mathcal{B}}^{\mathrm{DLIN}}(\lambda) = |\mathsf{Pr}[\mathcal{B}(\mathcal{D}_2, z = \nu^{c_1 + c_2}) = 1] - \mathsf{Pr}[\mathcal{B}(\mathcal{D}_2, z \text{ random in } \mathbb{G}) = 1]|$$

*The* DLIN *assumption holds if $\mathsf{Adv}_{\mathcal{B}}^{\mathrm{DLIN}}(\lambda)$ is negligible for any* PPT *$\mathcal{B}$.*

Both of the above assumptions are well-studied. The DBDH assumption has been introduced by Boneh and Franklin [3]. On the other hand, the validity of DLIN assumption in the generic group model has been established by Boneh et al. [2].

## 3 Our Self-Updatable Encryption

In this section we present our fully secure SUE scheme in prime order bilinear group.

### 3.1 Managing the Time Structure

As in [10], [9], we use a full binary tree $\mathcal{T}$ to represent time in our SUE scheme by assigning time periods to all tree nodes. In the full binary tree $\mathcal{T}$, each node (internal node or leaf node) is assigned a unique time value by using pre-order tree traversal that recursively visits the root node, the left subtree, and the right subtree; i.e., the root node of $\mathcal{T}$ is associated with 0 time value and the right most leaf node of $\mathcal{T}$ is associated with $2^{d_{max}+1} - 2$ time value where $d_{max}$ is the maximum depth of the tree. We fix some notation at this point.

- $\nu_T$ = The node associated with time $T$.
- $\mathbf{Parent}(\nu_T)$ = The parent node of $\nu_T$ in the tree.
- $\mathbf{Path}(\nu_T)$ = The set of path nodes from the root node to the node $\nu_T$.
- $\mathbf{RightSibling}(\mathbf{Path}(\nu_T))$ = The set of right sibling nodes of $\mathbf{Path}(\nu_T)$.
- $\mathbf{TimeNodes}(\nu_T) = \{\nu_T\} \cup \mathbf{RightSibling}(\mathbf{Path}(\nu_T)) \setminus \mathbf{Path}(\mathbf{Parent}(\nu_T))$.

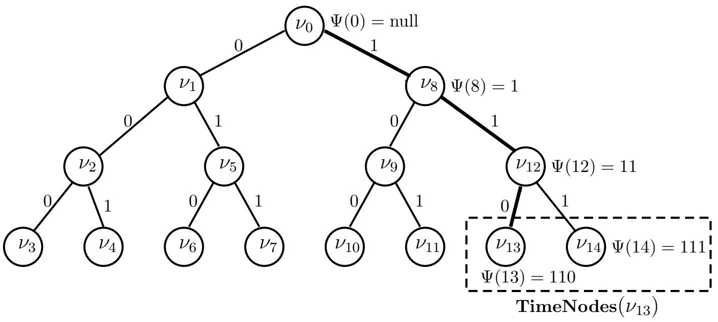

Fig. 1: The tree-based time structure for SUE

We mention that, if a node on **Path**$(\nu_T)$ is itself a right sibling of some node in the tree $\mathcal{T}$, then we include it in **RightSibling**(**Path**$(\nu_T)$).

We instantiate the above notations using Figure 1. Let us consider the node $\nu_{13}$, the node associated with the time $T = 13$ according to pre-order traversal. Here, **Parent**$(\nu_{13}) = \nu_{12}$, **Path**$(\nu_{13}) = \{\nu_0, \nu_8, \nu_{12}, \nu_{13}\}$, **RightSibling**(**Path**$(\nu_{13})) = \{\nu_8, \nu_{12}, \nu_{14}\}$, and **TimeNodes**$(\nu_{13})$ $= \{\nu_{13}, \nu_{14}\}$ (marked with doted rectangle).

Intuitively, if we consider all subtrees corresponding to all the times that are greater than and equal to the time $T$ by pre-order traversal, then **TimeNodes**$(\nu_T)$ contains the root of each such subtree. More precisely, we have the following lemma, the proof of which is straightforward.

**Lemma 1.** ***TimeNodes***$(\nu_T)$ $\cap$ ***Path***$(\nu_{T'}) \neq \varnothing$ *if and only if* $T \leq T'$.

To construct our SUE scheme in prime order bilinear group, we start from the hierarchical identity based encryption (HIBE) scheme of Waters [16] and exchange the private key structure with the ciphertext structure of this HIBE scheme. To use the structure of HIBE, we associate each node with a unique label string $L \in \{0,1\}^*$. The label of each node in the tree is assigned as follows: Each edge in the tree is assigned with label 0 or 1 depending on whether the edge connects a node to its left or right child node respectively. The label $L$ of a node $\nu_T$ is defined as the bit string obtained by reading all the labels of edges in **Path**$(\nu_T)$. We assign a special empty string 'null' to the root node as a label. Note that the length of the label string associated with a node increases with the depth of the node. For example, in Figure 1, the label of $\nu_8$ is 1, that of $\nu_{12}$ is 11, $\nu_{13}$ is 110 etc. For a label string $L \in \{0,1\}^*$ of length $d$, we define the following notations:

- $L[i]$ = the $i$th bit of $L$.
- $L|_i$ = the prefix of $L$ with $i$-bit length, where $i = 0$ means that $L|_i$ is the empty string.
- $L\|L'$ = concatenation of the string $L$ and another string $L' \in \{0,1\}^*$.
- $L^{(j)} = L|_{d-j}\|1$, $1 \leq j \leq d$. $L^{(0)}$ is defined to be the label string $L$ itself.
- $d^{(j)}$ = the length of $L^{(j)}$, $0 \leq j \leq d$. Note that $d^{(0)} = d$.

A simple observation would guarantee the validity of the following lemma.

**Lemma 2.** *If a node* $\nu_T$ *has label string* $L$ *of length* $d$*, then* $\{L^{(j)} | 1 \leq j \leq d\}$ *contains the label strings of all the nodes in* ***RightSibling(Path***$(\nu_T)$***)***. *Further, if* $L^{(j)} = L|_{d-j+1}$, *for* $1 < j \leq d$, *then* $L^{(j)}$ *falls in* ***Path***(***Parent***$(\nu_T)$) *and hence they are excluded from* ***TimeNodes***$(\nu_T)$.*On the other hand, if* $L^{(j)} \neq L|_{d-j+1}$, $1 \leq j \leq d$, *then* $L^{(j)}$ *does not correspond to a node in* ***Path***(***Parent***$(\nu_T)$), *rather, it is contained in* ***TimeNodes***$(\nu_T)$.

For example, if $L = 110$, then $L[1] = 1$, $L[2] = 1$, $L[3] = 0$, and $L|_1 = 1$, $L|_2 = 11$, $L|_3 = 110$. In Figure 1, $\nu_{13}$ has label string $L = 110$ and the set $\{L^{(1)}, L^{(2)}, L^{(3)}\} = \{111, 11, 1\}$ consists of the label strings of $\{\nu_{14}, \nu_{12}, \nu_8\}$ = **RightSibling**(**Path**$(\nu_{13})$). Also note that $L^{(2)} = 11 = L|_{3-2+1}$

corresponds to the node $\nu_{12}$ which is in **Path**(**Parent**$(\nu_{13})$) and similarly for $L^{(3)}$, whereas $L^{(1)} = 111 \neq 110 = L|_{3-1+1}$ and the node with label string $L^{(1)}$, i.e., $\nu_{14}$ is not contained in **Path**(**Parent**$(\nu_{13})$), rather, it is included in **TimeNodes**$(\nu_{13})$.

We define a mapping $\Psi$ that maps time $T$ corresponding to a tree node $\nu_T$ to a unique label $L$ associated with $\nu_T$. Specifically, we define $\Psi(T) = L$ where the node $\nu_T$, according to preorder traversal, has label $L$ assigned using the method discussed above. For instance, in Figure 1, $\Psi(8) = 1$, $\Psi(12) = 11$, $\Psi(13) = 110$ etc. The SUE ciphertext header for time $T$ consists of a ciphertext component for each node in **TimeNodes**$(\nu_T)$.

### 3.2 Construction

**SUE.Setup**$(1^\lambda, T_{max})$: The key generation center takes as input a security parameter $1^\lambda$ and a maximum time $T_{max}$ where $T_{max} = 2^{d_{max}+1} - 2$. It proceeds as follows:

- It considers a full binary tree of depth $d_{max}$ so that $d_{max}$ is the maximum length of label strings associated with the nodes in the tree.
- It generates a bilinear group $\mathcal{G} = (p, \mathbb{G}, \mathbb{G}_T, e)$ of prime order $p$ where $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ is the bilinear map.
- It chooses random generators $g, v, v_1, v_2, w, u_1, \ldots, u_{d_{max}}, h_1, \ldots, h_{d_{max}} \in \mathbb{G}$ and exponents $a_1, a_2, b, \alpha \in \mathbb{Z}_p$. Let $\tau_1 = vv_1^{a_1}$, $\tau_2 = vv_2^{a_2}$.
- It publishes the public parameters

$$\mathsf{PP} = (\mathbb{G}, \mathbb{G}_T, e, g^b, g^{a_1}, g^{a_2}, g^{ba_1}, g^{ba_2}, \tau_1, \tau_2, \tau_1^b, \tau_2^b, w^{a_1}, \{u_i^{a_1}, h_i^{a_1}\}_{i=1}^{d_{max}}, e(g,g)^{\alpha a_1 b})$$

and sets the master secret key

$$\mathsf{MK} = (g, g^\alpha, g^{\alpha a_1}, v, v_1, v_2, w, \{u_i, h_i\}_{i=1}^{d_{max}})$$

for itself.

**SUE.GenKey**$(T', \mathsf{MK}, \mathsf{PP})$: Taking as input a time $T'$, the master secret key $\mathsf{MK}$ and the public parameters $\mathsf{PP}$, the key generation center generates a secret key $\mathsf{SK}_{T'}$ for a user as follows:

- It computes the label string $L' = \Psi(T') \in \{0,1\}^n$ (say).
- It selects random $s_1, s_2, z_1, z_2, \mathsf{ktag}_1, \ldots, \mathsf{ktag}_n \in \mathbb{Z}_p$ and sets $s = s_1 + s_2$.
- It outputs the private key $\mathsf{SK}_{T'}$ that implicitly includes $T'$ as

$$\mathsf{SK}_{T'} = \begin{pmatrix} D_1 = g^{\alpha a_1} v^s, D_2 = g^{-\alpha} v_1^s g^{z_1}, D_3 = (g^b)^{-z_1}, D_4 = v_2^s g^{z_2}, \\ D_5 = (g^b)^{-z_2}, D_6 = (g^b)^{s_2}, D_7 = g^{s_1}, \\ \{K_i = (u_i^{L'|_i} w^{\mathsf{ktag}_i} h_i)^{s_1}\}_{i=1}^n, \{\mathsf{ktag}_i\}_{i=1}^n \end{pmatrix} \qquad (1)$$

where for any $z \in \mathbb{G}$ and any label string $L \in \{0,1\}^*$, $z^L = y \in \mathbb{G}$ such that $\log_z y = \sum_{i=1}^{|L|} L[i] 2^{i-1} \pmod p$, $|L|$ being the length of $L$.

**SUE.Encrypt**$(T, \mathsf{PP})$: Taking as input a time $T$ and the public parameters $\mathsf{PP}$, the encryptor prepares a ciphertext header consisting of components corresponding to all the nodes in **TimeNodes**$(\nu_T)$ and a session key by performing the following operations:

1. It first computes $L^{(0)} = \psi(T) \in \{0,1\}^d$ (say). It generates the ciphertext component $\mathsf{CH}^{(0)}$ corresponding to the node $\nu_T$ as follows:
   - It selects random $\mu_1, \ldots, \mu_d, r_1, r_2, \mathsf{ctag}_1, \ldots, \mathsf{ctag}_d \in \mathbb{Z}_p$ and sets $t = \sum_{i=1}^{d} \mu_i$.

– It sets

$$\mathsf{CH}^{(0)} = \begin{pmatrix} C_1 = (g^b)^{r_1+r_2}, C_2 = (g^{ba_1})^{r_1}, C_3 = (g^{a_1})^{r_1}, C_4 = (g^{ba_2})^{r_2}, \\ C_5 = (g^{a_2})^{r_2}, C_6 = \tau_1^{r_1}\tau_2^{r_2}, C_7 = (\tau_1^b)^{r_1}(\tau_2^b)^{r_2}(w^{a_1})^{-t}, \\ \{E_{i,1} = [(u_i^{a_1})^{L^{(0)}|_i}(w^{a_1})^{\mathsf{ctag}_i}h_i^{a_1}]^{\mu_i}, E_{i,2} = (g^{a_1})^{\mu_i}\}_{i=1}^d, \{\mathsf{ctag}_i\}_{i=1}^d \end{pmatrix} \quad (2)$$

2. For $1 \le j \le d$, it sets $L^{(j)} = L^{(0)}|_{d-j}\|1$, i.e., it sets $L^{(j)}$ to be the label of the nodes belonging to $\mathbf{RightSibling}(\mathbf{Path}(\nu_T))$, and proceeds the following steps:

   **Case**(a): $(L^{(j)} = L^{(0)}|_{d-j+1})$ In this case, $L^{(j)}$ is the label of a node in $\mathbf{Path}(\nu_T)$. The encryptor sets $\mathsf{CH}^{(j)}$ as an empty one since it either corresponds to the node $\nu_T$ $(j = 1)$, for which the ciphertext component has already been computed, or it corresponds to a node belonging to $\mathbf{Path}(\mathbf{Parent}\ (\nu_T))$ which is not in $\mathbf{TimeNodes}(\nu_T)$ (as explained in Figure 1) and hence for which the ciphertext component is not needed.

   **Case**(b): $(L^{(j)} \neq L^{(0)}|_{d-j+1})$ In this case, $L^{(j)}$ is the label of a node in $\mathbf{TimeNodes}(\nu_T)$. The encryptor obtains $\mathsf{CH}^{(j)}$ as follows:

   – It chooses fresh random $\mu'_{d-j+1}, \mathsf{ctag}'_{d-j+1} \in \mathbb{Z}_p$, sets $\mu'_i = \mu_i$, $\mathsf{ctag}'_i = \mathsf{ctag}_i$ for $1 \le i \le d-j$ and defines $t' = \sum_{i=1}^{d-j+1} \mu'_i$.

   – It computes

$$\mathsf{CH}^{(j)} = \begin{pmatrix} C'_1 = (g^b)^{r_1+r_2}, C'_2 = (g^{ba_1})^{r_1}, C'_3 = (g^{a_1})^{r_1}, C'_4 = (g^{ba_2})^{r_2}, \\ C'_5 = (g^{a_2})^{r_2}, C'_6 = \tau_1^{r_1}\tau_2^{r_2}, C'_7 = (\tau_1^b)^{r_1}(\tau_2^b)^{r_2}(w^{a_1})^{-t'}, \\ \{E'_{i,1} = [(u_i^{a_1})^{L^{(j)}|_i}(w^{a_1})^{\mathsf{ctag}'_i}h_i^{a_1}]^{\mu'_i}, E'_{i,2} = (g^{a_1})^{\mu'_i}\}_{i=1}^{d-j+1}, \\ \{\mathsf{ctag}'_i\}_{i=1}^{d-j+1} \end{pmatrix} \quad (3)$$

   Note that $L^{(j)}$ and $L^{(0)}$ have the same prefix string of length $d-j$.

   – It also prunes redundant elements, namely, $C'_1, \ldots, C'_6, \{E'_{i,1}, E'_{i,2}\}_{i=1}^{d-j}, \{\mathsf{ctag}'_i\}_{i=1}^{d-j}$ from $\mathsf{CH}^{(j)}$ which are already contained in $\mathsf{CH}^{(0)}$.

3. It removes all empty $\mathsf{CH}^{(j)}$'s and sets $\mathsf{CH}_T = (\mathsf{CH}^{(0)}, \ldots, \mathsf{CH}^{(d')})$ for some $d' < d$ that consists of nonempty $\mathsf{CH}^{(j)}$'s.
4. It sends the ciphertext header $\mathsf{CH}_T$ that implicitly includes $T$ to the ciphertext storage and keeps a session key as $\mathsf{EK} = (e(g,g)^{\alpha a_1 b})^{r_2}$ private to itself. Note that $\mathsf{CH}^{(j)}$'s are ordered according to pre-order traversal.

**SUE.UpdateCT**$(\mathsf{CH}_T, T+1, \mathsf{PP})$: On input a ciphertext header $\mathsf{CH}_T = (\mathsf{CH}^{(0)}, \ldots, \mathsf{CH}^{(d')})$ for a time $T$, a next time $T+1$ and the public parameters $\mathsf{PP}$, the semi-trusted third party, employed for managing users' access on ciphertexts by updating a ciphertext header associated with some time to a ciphertext header corresponding to a future time, performs the following steps, where $L^{(j)}$ is the label of $\mathsf{CH}^{(j)}$.

**Case**(a): $(L^{(0)} = \Psi(T)$ is the label of an internal node in the tree)

– It first obtains $\mathsf{CH}^{L^{(0)}\|0}$ and $\mathsf{CH}^{L^{(0)}\|1}$ as follows:
  - Let $\mathsf{CH}^{(0)} = (C_1, \ldots, C_7, \{E_{i,1}, E_{i,2}\}_{i=1}^d, \{\mathsf{ctag}_i\}_{i=1}^d)$.
    It selects a random $\mu_{d+1}, \mathsf{ctag}_{d+1} \in \mathbb{Z}_p$.

- It computes

$$\mathsf{CH}^{L^{(0)}\|0} = \begin{pmatrix} C^{\dagger}_1 = C_1, C^{\dagger}_2 = C_2, C^{\dagger}_3 = C_3, C^{\dagger}_4 = C_4, \\ C^{\dagger}_5 = C_5, C^{\dagger}_6 = C_6, C^{\dagger}_7 = C_7 (w^{a_1})^{-\mu_{d+1}}, \\ \{E^{\dagger}_{i,1} = E_{i,1}, E^{\dagger}_{i,2} = E_{i,2}\}_{i=1}^{d}, \\ E^{\dagger}_{d+1,1} = [(u^{a_1}_{d+1})^{L^{(0)}\|0} (w^{a_1})^{\mathsf{ctag}_{d+1}} h^{a_1}_{d+1}]^{\mu_{d+1}}, \\ E^{\dagger}_{d+1,2} = (g^{a_1})^{\mu_{d+1}}, \{\mathsf{ctag}^{\dagger}_i = \mathsf{ctag}_i\}_{i=1}^{d+1} \end{pmatrix}$$

- In a similar fashion, $\mathsf{CH}^{L^{(0)}\|1}$ is computed from $\mathsf{CH}^{(0)}$ by selecting fresh random $\mu'_{d+1}, \mathsf{ctag}'_{d+1} \in \mathbb{Z}_p$.

Note that in this case, $L^{(0)}\|0$ corresponds to the node $\nu_{T+1}$ and $L^{(0)}\|1$ corresponds to a node in **TimeNodes**$(\nu_{T+1})$ according to pre-order traversal. See Figure 1 for a pictorial understanding.

– It also prunes redundant elements in $\mathsf{CH}^{L^{(0)}\|1}$ which are contained in $\mathsf{CH}^{L^{(0)}\|0}$ in the same way as in **SUE.Encrypt**.

– It outputs an updated ciphertext header as $\mathsf{CH}_{T+1} = (\mathsf{CH}^{\dagger(0)} = \mathsf{CH}^{L^{(0)}\|0}, \mathsf{CH}^{\dagger(1)} = \mathsf{CH}^{L^{(0)}\|1}, \ldots, \mathsf{CH}^{\dagger(d'+1)} = \mathsf{CH}^{(d')})$. Note that all the nodes in **TimeNodes**$(\nu_T)$ except the node $\nu_T$ itself are also included in **TimeNodes**$(\nu_{T+1})$ due to pre-order traversal as can be seen from Figure 1.

**Case**(b): ($L^{(0)} = \Psi(T)$ is the label of a leaf node) It copies common elements in $\mathsf{CH}^{(0)}$ to $\mathsf{CH}^{(1)}$ and simply removes $\mathsf{CH}^{(0)}$, since $\mathsf{CH}^{(1)}$ is the ciphertext component corresponding to $\nu_{T+1}$ in this case and the node $\nu_T$ corresponding to the ciphertext component $\mathsf{CH}^{(0)}$ is excluded from **TimeNodes**$(\nu_{T+1})$ by pre-order traversal. It outputs an updated ciphertext header as $\mathsf{CH}_{T+1} = (\mathsf{CH}^{\dagger(0)} = \mathsf{CH}^{(1)}, \ldots, \mathsf{CH}^{\dagger(d'-1)} = \mathsf{CH}^{(d')})$. Observe that in this case also, the nodes included in **TimeNodes**$(\nu_T)$ except $\nu_T$ itself are also included in **TimeNodes**$(\nu_{T+1})$. See Figure 1 for an instantiation.

**SUE.RandCT**$(\mathsf{CH}_T, \mathsf{PP})$: The semi-trusted third party managing the users' access on ciphertexts takes as input a ciphertext header $\mathsf{CH}_T = (\mathsf{CH}^{(0)}, \ldots, \mathsf{CH}^{(d')})$ for a time $T$, and public parameters $\mathsf{PP}$. Let $L^{(j)}$ be the label of $\mathsf{CH}^{(j)}$ and $d^{(j)}$ be the length of the label $L^{(j)}$. It proceeds as follows:

1. It first obtains a re-randomized ciphertext component $\widetilde{\mathsf{CH}}^{(0)}$ from $\mathsf{CH}^{(0)} = (C_1, \ldots, C_7, \{E_{i,1}, E_{i,2}\}_{i=1}^{d^{(0)}}, \{\mathsf{ctag}_i\}_{i=1}^{d^{(0)}})$ as follows:

   – It chooses random exponents $\mu'_1, \ldots, \mu'_{d^{(0)}}, r'_1, r'_2 \in \mathbb{Z}_p$ and sets $t' = \sum_{i=1}^{d^{(0)}} \mu'_i$.

   – It obtains

$$\widetilde{\mathsf{CH}}^{(0)} = \begin{pmatrix} \widetilde{C}_1 = C_1 (g^b)^{r'_1 + r'_2}, \widetilde{C}_2 = C_2 (g^{ba_1})^{r'_1}, \widetilde{C}_3 = C_3 (g^{a_1})^{r'_1}, \widetilde{C}_4 = C_4 (g^{ba_2})^{r'_2}, \\ \widetilde{C}_5 = C_5 (g^{a_2})^{r'_2}, \widetilde{C}_6 = C_6 (\tau_1^{r'_1} \tau_2^{r'_2}), \widetilde{C}_7 = C_7 (\tau_1^b)^{r'_1} (\tau_2^b)^{r'_2} (w^{a_1})^{-t'}, \\ \{\widetilde{E}_{i,1} = E_{i,1} [(u_i^{a_1})^{L^{(0)}|i} (w^{a_1})^{\mathsf{ctag}_i} h_i^{a_1}]^{\mu'_i}, \widetilde{E}_{i,2} = E_{i,2} (g^{a_1})^{\mu'_i}\}_{i=1}^{d^{(0)}}, \\ \{\widetilde{\mathsf{ctag}}_i = \mathsf{ctag}_i\}_{i=1}^{d^{(0)}} \end{pmatrix}$$

2. For $1 \leq j \leq d'$, it computes $\widetilde{\mathsf{CH}}^{(j)}$ from pruned $\mathsf{CH}^{(j)} = (C'_7, \{E'_{d^{(j)},1}, E'_{d^{(j)},2}\}, \{\mathsf{ctag}'_{d^{(j)}}\})$ as follows:

   – It selects random $\mu''_{d^{(j)}} \in \mathbb{Z}_p$ and defines $t'' = \sum_{i=1}^{d^{(j)}-1} \mu'_i + \mu''_{d^{(j)}}$. Note that $L^{(j)}$ has the same prefix string of length $d^{(j)} - 1$ as $L^{(0)}$.

- It computes

$$\widetilde{\mathsf{CH}}^{(j)} = \begin{pmatrix} \widetilde{C}'_7 = C'_7(\tau_1^b)^{r'_1}(\tau_2^b)^{r'_2}(w^{a_1})^{-t''}, \\ \{\widetilde{E}'_{d^{(j)},1} = E'_{d^{(j)},1}[(u_{d^{(j)}}^{a_1})^{L^{(j)}}(w^{a_1})^{\mathsf{ctag}'_{d^{(j)}}} h_{d^{(j)}}^{a_1}]^{\mu''_{d^{(j)}}}, \\ \widetilde{E}'_{d^{(j)},2} = E'_{d^{(j)},2}(g^{a_1})^{\mu''_{d^{(j)}}}\}, \{\widetilde{\mathsf{ctag}}'_{d^{(j)}} = \mathsf{ctag}'_{d^{(j)}}\} \end{pmatrix}$$

Note that the other components of $\widetilde{\mathsf{CH}}^{(j)}$ have been pruned.

3. It publishes a re-randomized ciphertext header as $\widetilde{\mathsf{CH}}_T = (\widetilde{\mathsf{CH}}^{(0)}, \ldots, \widetilde{\mathsf{CH}}^{(d')})$ and a partial session key as $\widetilde{\mathsf{EK}} = (e(g,g)^{\alpha a_1 b})^{r'_2}$ that will be multiplied with the session key $\mathsf{EK}$ of $\mathsf{CH}_T$ to produce a re-randomized session key.

**SUE.Decrypt**$(\mathsf{CH}_T, \mathsf{SK}_{T'}, \mathsf{PP})$: A user takes as input a ciphertext header $\mathsf{CH}_T = (\mathsf{CH}^{(0)}, \ldots, \mathsf{CH}^{(d')})$, its own private key $\mathsf{SK}_{T'} = (D_1, \ldots, D_7, \{K_i\}_{i=1}^n, \{\mathsf{ktag}_i\}_{i=1}^n)$ such that $L' = \Psi(T') \in \{0,1\}^n$ together with the public parameters $\mathsf{PP}$.

1. If the user can find $\mathsf{CH}^{(j)} = (C_1, \ldots, C_7, \{E_{i,1}, E_{i,2}\}_{i=1}^{d^{(j)}}, \{\mathsf{ctag}_i\}_{i=1}^{d^{(j)}})$ from $\mathsf{CH}_T$ such that its corresponding label string $L^{(j)} \in \{0,1\}^{d^{(j)}}$ is a prefix of $L'$ and for $1 \le i \le d^{(j)}$, it holds that $\mathsf{ctag}_i \neq \mathsf{ktag}_i$, then the user retrieves $\mathsf{EK}$ by the following computation:
$$A_1 = \prod_{i=1}^{5} e(C_i, D_i),\ A_2 = e(C_6, D_6)e(C_7, D_7),\ A_3 = A_1/A_2,\ A_4 = \prod_{i=1}^{d^{(j)}} \left[\frac{e(E_{i,1}, D_7)}{e(E_{i,2}, K_i)}\right]^{\frac{1}{\mathsf{ctag}_i - \mathsf{ktag}_i}}$$
and finally, $\mathsf{EK} = A_3/A_4$.
2. Otherwise, the user obtains $\perp$.

### 3.3 Correctness

The SUE ciphertext header $\mathsf{CH}_T$ of a time $T$ consists of the ciphertext components $\mathsf{CH}^{(0)}, \ldots, \mathsf{CH}^{(d')}$ each of which is associated with a node in **TimeNodes**$(\nu_T)$. If the SUE private key $\mathsf{SK}_{T'}$ for a time $T'$ satisfies $T \le T'$, then **TimeNodes**$(\nu_T)$ $\cap$ **Path**$(\nu_{T'}) = \{\nu_{\widehat{T}}\}$, where $T \le \widehat{T} \le T'$, for a unique node $\nu_{\widehat{T}}$, by the property of pre-order tree traversal as stated in Lemma 1. Let $\mathsf{CH}^{(j)}$ be the ciphertext component that is associated with the node $\nu_{\widehat{T}}$. Now as $\nu_{\widehat{T}} \in$ **Path**$(\nu_{T'})$, the label string $L^{(j)}$ of $\nu_{\widehat{T}}$ is a prefix of the label string $L' = \Psi(T')$. Assume that $L^{(j)}$ and $L'$ are of length $d^{(j)}$ and $n$ respectively. According to the scheme description, we have $\mathsf{SK}_{T'}$ and $\mathsf{CH}^{(j)}$ are of the forms as defined in equations (1) and (2) or (3) respectively. Thus we obtain

$$\left.\begin{array}{l} A_1 = \displaystyle\prod_{i=1}^{5} e(C_i, D_i) = e(g,g)^{\alpha a_1 b r_2} e(v,g)^{b(r_1+r_2)s} e(v_1,g)^{b a_1 r_1 s} e(v_2,g)^{b a_2 r_2 s} \\ A_2 = e(C_6, D_6)e(C_7, D_7) = e(v,g)^{b(r_1+r_2)s} e(v_1,g)^{b a_1 r_1 s} e(v_2 g)^{b a_2 r_2 s} e(g,w)^{-a_1 t s_1} \\ A_3 = A_1/A_2 = e(g,g)^{\alpha a_1 b r_2} e(g,w)^{a_1 t s_1} \\ A_4 = \displaystyle\prod_{i=1}^{d^{(j)}} \left[\frac{e(E_{i,1}, D_7)}{e(E_{i,2}, K_i)}\right]^{\frac{1}{\mathsf{ctag}_i - \mathsf{ktag}_i}} = e(g,w)^{a_1 s_1 \sum_{i=1}^{d^{(j)}} \mu_i},\ \text{provided } \mathsf{ctag}_i \neq \mathsf{ktag}_i,\ 1 \le i \le d^{(j)} \\ \mathsf{EK} = A_3/A_4 = e(g,g)^{\alpha a_1 b r_2},\ \text{since } t = \displaystyle\sum_{i=1}^{d^{(j)}} \mu_i \end{array}\right] \quad (4)$$

Further, one can easily verify that **SUE.UpdateCT** outputs a valid ciphertext header and **SUE.RandCT** re-randomizes the input ciphertext header. These procedures are sequentially executed to implement the procedure **SUE.ReEncrypt**, whereby any untrusted third party, managing users' access on ciphertexts, can update a ciphertext header attached a certain time to a new ciphertext header attached a future time using only public information.

## 4 Security Analysis

**Theorem 1.** *For any* $\mathsf{PPT}$ $\mathsf{CPA}$ *adversary* $\mathcal{A}$ *for the SUE scheme, introduced in Section* 3*, we have* $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{SUE}}(\lambda) \leq \mathsf{Adv}_{\mathcal{B}_1}^{\mathrm{DLIN}}(\lambda) + q\mathsf{Adv}_{\mathcal{B}_2}^{\mathrm{DLIN}}(\lambda) + \mathsf{Adv}_{\mathcal{B}_3}^{\mathrm{DBDH}}(\lambda)$*, where* $\mathsf{Adv}_{\mathcal{B}_i}^{\mathrm{DLIN}}(\lambda)$*, for* $i \in \{1, 2\}$*, denotes the advantage of an algorithm* $\mathcal{B}_i$ *in solving the DLIN problem;* $\mathsf{Adv}_{\mathcal{B}_3}^{\mathrm{DBDH}}(\lambda)$ *denotes that of an algorithm* $\mathcal{B}_3$ *in solving the DBDH problem; and* $q$ *is the maximum number of private key queries made by* $\mathcal{A}$*. Thus, the SUE scheme of Section* 3 *is fully secure under* $\mathsf{CPA}$ *if the DBDH and DLIN assumptions hold.*

*Proof.* To prove the security of our SUE scheme, we use the dual system encryption technique of Lewko and Waters [13], [16]. In dual system encryption, ciphertexts and private keys can be normal or semi-functional type. The normal type and the semi-functional type are indistinguishable and a semi-functional ciphertext cannot be decrypted by a semi-functional private key. The whole security proof consists of hybrid games that change the normal challenge ciphertext and the normal private keys to the semi-functional challenge ciphertext and semi-functional private keys respectively. In the final game, the adversary given the semi-functional private keys and the semi-functional challenge ciphertext cannot distinguish a proper session key from a random session key.

We first define the semi-functional type of ciphertexts and private keys.

**SUE.GenKeySF**: This algorithm generates a semi-functional private key $\widehat{\mathsf{SK}}_{T'}$ for a time $T'$ as follows:

- It first computes the label string $L' = \Psi(T') \in \{0,1\}^n$ (say).
- Next, it creates a normal private key

$$\mathsf{SK}_{T'} = \begin{pmatrix} D_1 = g^{\alpha a_1} v^s, D_2 = g^{-\alpha} v_1^s g^{z_1}, D_3 = (g^b)^{-z_1}, D_4 = v_2^s g^{z_2}, \\ D_5 = (g^b)^{-z_2}, D_6 = (g^b)^{s_2}, D_7 = g^{s_1}, \\ \{K_i = (u_i^{L'|_i} w^{\mathsf{ktag}_i} h_i)^{s_1}\}_{i=1}^n, \{\mathsf{ktag}_i\}_{i=1}^n \end{pmatrix}$$

  for the time $T'$ under random exponents $s_1, s_2, z_1, z_2 \in \mathbb{Z}_p$ using the master key $\mathsf{MK}$ along with the public parameters $\mathsf{PP}$ by running the algorithm **SUE.GenKey**$(T', \mathsf{MK}, \mathsf{PP})$.
- Then, it chooses a random $\gamma \in \mathbb{Z}_p$.
- It sets

$$\widehat{\mathsf{SK}}_{T'} = \begin{pmatrix} \widehat{D}_1 = D_1 g^{-a_1 a_2 \gamma}, \widehat{D}_2 = D_2 g^{a_2 \gamma}, \widehat{D}_3 = D_3, \widehat{D}_4 = D_4 g^{a_1 \gamma}, \\ \widehat{D}_5 = D_5, \widehat{D}_6 = D_6, \widehat{D}_7 = D_7, \\ \{\widehat{K}_i = K_i\}_{i=1}^n, \{\widehat{\mathsf{ktag}}_i = \mathsf{ktag}_i\}_{i=1}^n \end{pmatrix} \tag{5}$$

- It outputs

$$\widehat{\mathsf{SK}}_{T'} = (\widehat{D}_1, \ldots, \widehat{D}_7, \{\widehat{K}_i\}_{i=1}^n, \{\widehat{\mathsf{ktag}}_i\}_{i=1}^n).$$

**SUE.EncryptSF**: This algorithm computes a semi-functional ciphertext header $\widehat{\mathsf{CH}}_T$ for a time $T$ as follows:

- It first computes the label string $L^{(0)} = \Psi(T) \in \{0,1\}^d$ (say).
- Next, it obtains a normal ciphertext header $\mathsf{CH}_T = (\mathsf{CH}^{(0)}, \ldots, \mathsf{CH}^{(d')})$ for some $d' < d$ that consists of nonempty $\mathsf{CH}^{(j)}$, $1 \leq j \leq d$, and a session key $\mathsf{EK}$ for the time $T$ by running **SUE.Encrypt**$(T, \mathsf{PP})$, where $L^{(j)}$ is the label string of $\mathsf{CH}^{(j)}$ and $d^{(j)}$ is the length of $L^{(j)}$. Note that $d^{(0)} = d$.
- Then, it modifies the components of $\mathsf{CH}_T$ as follows:
  1. Let

$$\mathsf{CH}^{(0)} = \begin{pmatrix} C_1 = (g^b)^{r_1 + r_2}, C_2 = (g^{ba_1})^{r_1}, C_3 = (g^{a_1})^{r_1}, C_4 = (g^{ba_2})^{r_2}, \\ C_5 = (g^{a_2})^{r_2}, C_6 = \tau_1^{r_1} \tau_2^{r_2}, C_7 = (\tau_1^b)^{r_1} (\tau_2^b)^{r_2} (w^{a_1})^{-t}, \\ \{E_{i,1} = [(u_i^{a_1})^{L^{(0)}|_i} (w^{a_1})^{\mathsf{ctag}_i} h_i^{a_1}]^{\mu_i}, E_{i,2} = (g^{a_1})^{\mu_i}\}_{i=1}^{d^{(0)}}, \{\mathsf{ctag}_i\}_{i=1}^{d^{(0)}} \end{pmatrix}$$

- It chooses a random $x \in \mathbb{Z}_p$.
- It sets

$$\widehat{\mathsf{CH}}^{(0)} = \begin{pmatrix} \widehat{C}_1 = C_1, \widehat{C}_2 = C_2, \widehat{C}_3 = C_3, \widehat{C}_4 = C_4 g^{ba_2x}, \\ \widehat{C}_5 = C_5 g^{a_2x}, \widehat{C}_6 = C_6 v_2^{a_2x}, \widehat{C}_7 = C_7 v_2^{a_2bx}, \\ \{\widehat{E}_{i,1} = E_{i,1}, \widehat{E}_{i,2} = E_{i,2}\}_{i=1}^{d^{(0)}}, \{\widehat{\mathsf{ctag}}_i = \mathsf{ctag}_i\}_{i=1}^{d^{(0)}} \end{pmatrix} \tag{6}$$

2. For $1 \leq j \leq d^{(0)}$, it proceeds as follows: If $\mathsf{CH}^{(j)}$ is not present in $\mathsf{CH}_T$, then it sets $\widehat{\mathsf{CH}}^{(j)}$ as an empty one. Otherwise, let

$$\mathsf{CH}^{(j)} = \begin{pmatrix} C'_7 = (\tau_1^b)^{r_1}(\tau_2^b)^{r_2}(w^{a_1})^{-t'}, \{E'_{d(j),1} = [(u_{d(j)}^{a_1})^{L(j)}(w^{a_1})^{\mathsf{ctag}'_{d(j)}} h_{d(j)}^{a_1}]^{\mu'_{d(j)}}, \\ E'_{d(j),2} = (g^{a_1})^{\mu'_{d(j)}}\}, \{\mathsf{ctag}'_{d(j)}\} \end{pmatrix}$$

It creates the semi-functional ciphertext component $\widehat{\mathsf{CH}}^{(j)}$ as

$$\widehat{\mathsf{CH}}^{(j)} = \begin{pmatrix} \widehat{C}'_7 = C'_7 v_2^{a_2bx}, \{\widehat{E}'_{d(j),1} = E'_{d(j),1}, \widehat{E}'_{d(j),2} = E'_{d(j),2}\}, \\ \{\widehat{\mathsf{ctag}}'_{d(j)} = \mathsf{ctag}'_{d(j)}\} \end{pmatrix} \tag{7}$$

Note that the elements $C'_1, C'_2, C'_3, C'_4, C'_5, C'_6, \{E'_{i,1}, E'_{i,2}\}_{i=1}^{d(j)-1}, \{\mathsf{ctag}'_i\}_{i=1}^{d(j)-1}$ have been pruned in the normal ciphertext component $\mathsf{CH}^{(j)}$ and so are in $\widehat{\mathsf{CH}}^{(j)}$.

3. It removes all empty $\widehat{\mathsf{CH}}^{(j)}$ and sets $\widehat{\mathsf{CH}}_T = (\widehat{\mathsf{CH}}^{(0)}, \ldots, \widehat{\mathsf{CH}}^{(d')})$.
4. It outputs a semi-functional ciphertext header as $\widehat{\mathsf{CH}}_T$ and a session key as $\widehat{\mathsf{EK}} = \mathsf{EK} = (e(g,g)^{\alpha a_1 b})^{r_2}$.

We make a few remarks about the nature of the semi-functional keys and ciphertexts.

- First, note that if one attempts to decrypt a semi-functional ciphertext $\widehat{\mathsf{CH}}_T$ with a normal key $\mathsf{SK}_{T'}$ satisfying $T \leq T'$, then the decryption would succeed. This follows from the fact that the semi-functional ciphertext header $\widehat{\mathsf{CH}}_T$ differs from the normal ciphertext header $\mathsf{CH}_T$ in the components $\widehat{C}_4, \widehat{C}_5, \widehat{C}_6, \widehat{C}_7$. Since,

$$\frac{e(g^{ba_2x}, D_4)e(g^{a_2x}, D_5)}{e(v_2^{a_2x}, D_6)e(v_2^{a_2bx}, D_7)} = 1$$

when $D_4, D_5, D_6, D_7$ come from $\mathsf{SK}_{T'}$, from equation (4), we see that $A_3$ remains invariant and so does $\mathsf{EK}$.
- Similarly, a semi-functional private key $\widehat{\mathsf{SK}}_{T'}$ is different from a normal private key $\mathsf{SK}_{T'}$ in the components $\widehat{D}_1, \widehat{D}_2, \widehat{D}_4$ and since

$$e(C_1, g^{-a_1a_2\gamma})e(C_2, g^{a_2\gamma})e(C_4, g^{a_1\gamma}) = 1$$

when $C_1, C_2, C_4$ come from a normally generated ciphertext header $\mathsf{CH}_T$, from equation (4) we see that $A_1$ remains unchanged. As a result, the semi-functional components of $\widehat{\mathsf{SK}}_{T'}$ (with $T \leq T'$) will not impede decryption when applied on $\mathsf{CH}_T$.
- However, when a semi-functional private key $\widehat{\mathsf{SK}}_{T'}$ is used to decrypt a semi-functional ciphertext header $\widehat{\mathsf{CH}}_T$ with $T \leq T'$, then additionally random element $e(g,g)^{ba_1a_2x\gamma}$ will be generated in the computation of $A_1$ in equation (4) and will carry over through the subsequent computations to finally get multiplied by the intended session key. Hence, the decryption process does not succeed.

We highlight that in order to generate semi-functional ciphertext and private keys (according to the defined procedures) one respectively needs $v_2^{a_2 b}$ and $g^{a_1 a_2}$ – neither of which is available from the public parameters or the master key, as well as, none of which can be computed because none of $a_1, a_2, b$ is explicitly available.

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

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

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

**Game $\mathbf{G}_2$**: Next, we define a new game $\mathbf{G}_2$. In this game all private keys are semi-functional and the challenge ciphertext header is semi-functional. Suppose that an adversary makes at most $q$ private key queries. For the security proof, we additionally define a sequence of games $\mathbf{G}_{1,1}, \ldots, \mathbf{G}_{1,k}, \ldots, \mathbf{G}_{1,q}$ where $\mathbf{G}_1 = \mathbf{G}_{1,0}$. In the game $\mathbf{G}_{1,k}$, for $1 \le k \le q$, the challenge ciphertext header is semi-functional, the first $k$ private keys are semi-functional, and the remaining private keys are normal.

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

Let $\mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_j}$ be the advantage of the adversary $\mathcal{A}$ in the game $\mathbf{G}_j$. We easily obtain that

$$\mathsf{Adv}_{\mathcal{A}}^{\text{SUE}}(\lambda) = \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_0},\ \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_1} = \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_{1,0}},\ \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_2} = \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_{1,q}},\ \text{and } \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_3} = 0.$$

From the subsequent Lemmas we can obtain the following result

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A}}^{\text{SUE}}(\lambda) &= \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_0} + (\mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_1} - \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_1}) + \sum_{k=1}^{q} (\mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_{1,k}} - \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_{1,k}}) - \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_3} \\
&\le |\mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_0} - \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_1}| + \sum_{k=1}^{q} |\mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_{1,k-1}} - \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_{1,k}}| + |\mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_2} - \mathsf{Adv}_{\mathcal{A}}^{\mathbf{G}_3}| \\
&\le \mathsf{Adv}_{\mathcal{B}_1}^{\text{DLIN}}(\lambda) + q\mathsf{Adv}_{\mathcal{B}_2}^{\text{DLIN}}(\lambda) + \mathsf{Adv}_{\mathcal{B}_3}^{\text{DBDH}}(\lambda)
\end{aligned}$$

This completes the proof. □

**Lemma 3.** *If there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}_{\mathcal{A}}^{\boldsymbol{G}_0} - \mathsf{Adv}_{\mathcal{A}}^{\boldsymbol{G}_1}| = \epsilon(\lambda)$, then there would exist an algorithm $\mathcal{B}_1$ such that $\mathsf{Adv}_{\mathcal{B}_1}^{\text{DLIN}}(\lambda) = \epsilon(\lambda)$. Thus, if the DLIN assumption holds, then no* PPT *adversary $\mathcal{A}$ can distinguish between the games $\boldsymbol{G}_0$ and $\boldsymbol{G}_1$ with a non-negligible advantage.*

*Proof.* Suppose there exists an adversary $\mathcal{A}$ that distinguishes between the games $\mathbf{G}_0$ and $\mathbf{G}_1$ with a non-negligible advantage. We construct a simulator $\mathcal{B}_1$ that attempts to solve the DLIN problem using $\mathcal{A}$. $\mathcal{B}_1$ is given a challenge tuple $(g, f, \nu, g^{c_1}, f^{c_2}, z) \in \mathbb{G}^6$ where $z$ is either $\nu^{c_1+c_2}$ or random. Then $\mathcal{B}_1$ that interacts with $\mathcal{A}$ is described as follows:

**Setup**:

- $\mathcal{B}_1$ chooses random exponents $b, \alpha, y_v, y_{v_1}, y_{v_2}, y_w, y_{u_1}, \ldots, y_{u_{d_{max}}}, y_{h_1}, \ldots, y_{h_{d_{max}}} \in \mathbb{Z}_p$.
- It then sets the public parameters

$$\mathsf{PP} = \begin{pmatrix} g^b, g^{a_1} = f, g^{a_2} = \nu, g^{ba_1} = f^b, g^{ba_2} = \nu^b, \tau_1 = g^{y_v} f^{y_{v_1}}, \tau_2 = g^{y_v} \nu^{y_{v_2}}, \\ \tau_1^b, \tau_2^b, w^{a_1} = f^{y_w}, \{u_i^{a_1} = f^{y_{u_i}}, h_i^{a_1} = f^{y_{h_i}}\}_{i=1}^{d_{max}}, e(g,g)^{\alpha a_1 b} = e(g,f)^{\alpha b} \end{pmatrix}$$

and the master key

$$\mathsf{MK} = (g, g^{\alpha}, g^{\alpha a_1} = f^{\alpha}, v = g^{y_v}, v_1 = g^{y_{v_1}}, v_2 = g^{y_{v_2}}, w = g^{y_w}, \{u_i = g^{y_{u_i}}, h_i = g^{y_{h_i}}\}_{i=1}^{d_{max}}).$$

- $\mathcal{B}_1$ publishes $\mathsf{PP}$ and keeps $\mathsf{MK}$ to itself.

**Query 1**: $\mathcal{A}$ adaptively requests a private key for time $T'$. $\mathcal{B}_1$ creates a normal private key by running **SUE.GenKey** by virtue of knowing the master secret key. $\mathcal{B}_1$ returns the created private key to $\mathcal{A}$. Note that $\mathcal{B}_1$ cannot generate a semi-functional private key since it can not compute $g^{a_1 a_2}$.

**Challenge**: In the challenge step, $\mathcal{A}$ submits a challenge time $T^*$ subject to the condition that for all of its private key queries for times $T'$ in Query 1 it holds that $T' < T^*$. $\mathcal{B}_1$ proceeds as follows:

1. It first obtains the label string $L^* \in \{0,1\}^d$ (say) by computing $\Psi(T^*)$. It creates the ciphertext component $\overline{\mathsf{CH}}^{(0)}$ in two steps.
   - It starts by computing a normal ciphertext component
     $$\mathsf{CH}^{(0)} = \left( \begin{array}{l} C_1 = (g^b)^{r_1+r_2}, C_2 = (f^b)^{r_1}, C_3 = f^{r_1}, C_4 = (\nu^b)^{r_2}, \\ C_5 = \nu^{r_2}, C_6 = \tau_1^{r_1}\tau_2^{r_2}, C_7 = (\tau_1^b)^{r_1}(\tau_2^b)^{r_2}(f^{y_w})^{-t}, \\ \{E_{i,1} = [(f^{y_{u_i}})^{L^*|_i}(f^{y_w})^{\mathsf{ctag}_i} f^{y_{h_i}}]^{\mu_i}, E_{i,2} = f^{\mu_i}\}_{i=1}^{d}, \{\mathsf{ctag}_i\}_{i=1}^{d} \end{array} \right)$$
     where $\mu_1, \ldots, \mu_d, r_1, r_2, \mathsf{ctag}_1, \ldots, \mathsf{ctag}_d$ are random elements of $\mathbb{Z}_p$ and $t = \sum_{i=1}^{d} \mu_i$.
   - Then it modifies components of $\mathsf{CH}^{(0)}$ to obtain $\overline{\mathsf{CH}}^{(0)}$ as follows:
     $$\overline{\mathsf{CH}}^{(0)} = \left( \begin{array}{l} \overline{C}_1 = C_1(g^{c_1})^b, \overline{C}_2 = C_2(f^{c_2})^{-b}, \overline{C}_3 = C_3 f^{-c_2}, \overline{C}_4 = C_4 z^b, \overline{C}_5 = C_5 z, \\ \overline{C}_6 = C_6(g^{c_1})^{y_v}(f^{c_2})^{-y_{v_1}} z^{y_{v_2}}, \overline{C}_7 = C_7[(g^{c_1})^{y_v}(f^{c_2})^{-y_{v_1}} z^{y_{v_2}}]^b (f^{c_2})^{y_w}, \\ \{\overline{E}_{i,1} = E_{i,1}, \overline{E}_{i,2} = E_{i,2}\}_{i=1}^{d-1}, \{\overline{E}_{d,1} = E_{d,1}(f^{c_2})^{-(y_{u_d} L^* + y_w \mathsf{ctag}_d + y_{h_d})}, \\ \overline{E}_{d,2} = E_{d,2}(f^{c_2})^{-1}\}, \{\overline{\mathsf{ctag}}_i = \mathsf{ctag}_i\}_{i=1}^{d} \end{array} \right)$$
     Intuitively, $\mathcal{B}_1$ implicitly replaces $r_1, r_2, t$ in $\mathsf{CH}^{(0)}$ by $\overline{r}_1, \overline{r}_2, \overline{t}$ respectively $\overline{\mathsf{CH}}^{(0)}$, where $\overline{r}_1 = -c_2 + r_1$, $\overline{r}_2 = r_2 + c_1 + c_2$ and $\overline{t} = -c_2 + t$.
2. For $1 \le j \le d$, $\mathcal{B}_1$ first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: If $L^{(j)} = L^*|_{d-j+1}$, it sets $\overline{\mathsf{CH}}^{(j)}$ as an empty one. Otherwise, it obtains $\overline{\mathsf{CH}}^{(j)}$ as follows:
   - It first computes
     $$\mathsf{CH}^{(j)} = \left( \begin{array}{l} C'_1 = (g^b)^{r_1+r_2}, C'_2 = (f^b)^{r_1}, C'_3 = f^{r_1}, C'_4 = (\nu^b)^{r_2}, \\ C'_5 = \nu^{r_2}, C'_6 = \tau_1^{r_1}\tau_2^{r_2}, C'_7 = (\tau_1^b)^{r_1}(\tau_2^b)^{r_2}(f^{y_w})^{-t'}, \\ \{E'_{i,1} = [(f^{y_{u_i}})^{L^{(j)}|_i}(f^{y_w})^{\mathsf{ctag}'_i} f^{y_{h_i}}]^{\mu'_i}, E'_{i,2} = f^{\mu'_i}\}_{i=1}^{d-j+1}, \{\mathsf{ctag}'_i\}_{i=1}^{d-j+1} \end{array} \right)$$
     where it uses $\mu'_i = \mu_i$, $\mathsf{ctag}'_i = \mathsf{ctag}_i$ for $1 \le i \le d-j$, $r_1$, $r_2$ that were used for computing $\mathsf{CH}^{(0)}$ together with fresh random $\mu'_{d-j+1}, \mathsf{ctag}'_{d-j+1} \in \mathbb{Z}_p$ and sets $t' = \sum_{r=1}^{d-j+1} \mu'_i$.
   - Next it modifies the components of $\mathsf{CH}^{(j)}$ the same way as for $\overline{\mathsf{CH}}^{(0)}$ obtaining
     $$\overline{\mathsf{CH}}^{(j)} = \left( \begin{array}{l} \overline{C}'_1 = C'_1(g^{c_1})^b, \overline{C}'_2 = C'_2(f^{c_2})^{-b}, \overline{C}'_3 = C'_3 f^{-c_2}, \overline{C}'_4 = C'_4 z^b, \overline{C}'_5 = C'_5 z, \\ \overline{C}'_6 = C'_6(g^{c_1})^{y_v}(f^{c_2})^{-y_{v_1}} z^{y_{v_2}}, \overline{C}'_7 = C'_7[(g^{c_1})^{y_v}(f^{c_2})^{-y_{v_1}} z^{y_{v_2}}]^b (f^{c_2})^{y_w}, \\ \{\overline{E}'_{i,1} = E'_{i,1}, \overline{E}'_{i,2} = E'_{i,2}\}_{i=1}^{d-j}, \\ \{\overline{E}'_{d-j+1,1} = E'_{d-j+1,1}(f^{c_2})^{-(y_{u_{d-j+1}} L^{(j)} + y_w \mathsf{ctag}'_{d-j+1} + y_{h_{d-j+1}})}, \\ \overline{E}'_{d-j+1,2} = E'_{d-j+1,2}(f^{c_2})^{-1}\}, \{\overline{\mathsf{ctag}}'_i = \mathsf{ctag}'_i\}_{i=1}^{d-j+1} \end{array} \right)$$
     by implicitly replacing $r_1, r_2, t'$ in $\mathsf{CH}^{(j)}$ by $\overline{r}_1 = -c_2 + r_1, \overline{r}_2 = r_2 + c_1 + c_2, \overline{t}' = -c_2 + t'$ respectively.

  - It also prunes redundant elements from $\overline{\mathsf{CH}}^{(j)}$.

3. It removes all empty $\overline{\mathsf{CH}}^{(j)}$ and sets $\overline{\mathsf{CH}}_{T^*} = (\overline{\mathsf{CH}}^{(0)}, \ldots, \overline{\mathsf{CH}}^{(d')})$ for some $d' < d$ that consists of nonempty $\overline{\mathsf{CH}}^{(j)}$.
4. $\mathcal{B}_1$ sets the challenge ciphertext header as $\mathsf{CH}_{T^*} = \overline{\mathsf{CH}}_{T^*}$ and the session key $\mathsf{EK}^* = e(g, f)^{\alpha b r_2}[e(g^{c_1}, f)e(g, f^{c_2})]^{\alpha b}$.
5. It flips a random coin $\beta \in \{0, 1\}$ and gives $\mathsf{CH}_{T^*}$ and $\mathsf{EK}^*$ to $\mathcal{A}$ if $\beta = 0$. Otherwise, it gives $\mathsf{CH}_{T^*}$ and a random session key to $\mathcal{A}$.

**Query 2**: Same as Query 1 subject to the same condition that for all the private key queries of $\mathcal{A}$ for times $T'$, it hold that $T' < T^*$.

**Guess**: $\mathcal{A}$ outputs a guess $\beta'$. If $\beta = \beta'$ then $\mathcal{B}_1$ outputs 1. Otherwise, it outputs 0.

If $z = \nu^{c_1+c_2}$, then the simulation is the same as the game $\mathbf{G}_0$ since the challenge ciphertext header is generated as normal as in equations (2) and (3) by implicitly setting $\overline{t} = -c_2 + \sum_{i=1}^{d} \mu_i$ for $\overline{\mathsf{CH}}^{(0)}$ and $\overline{t}' = -c_2 + \sum_{i=1}^{d-j+1} \mu'_i$ for all other nonempty $\overline{\mathsf{CH}}^{(j)}$'s together with $\overline{r}_1 = -c_2 + r_1$, $\overline{r}_2 = r_2 + c_1 + c_2$. On the other hand, if $z = \nu^r$ for some random $r \in \mathbb{Z}_p$, then the simulation is same as the game $\mathbf{G}_1$ since the ciphertext header is generated as semi-functional as in equations (6) and (7) by implicitly setting $x \equiv r - c_1 - c_2 \mod p$. This completes the proof. □

**Lemma 4.** *If there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}_{\mathcal{A}}^{\boldsymbol{G}_{1,k-1}} - \mathsf{Adv}_{\mathcal{A}}^{\boldsymbol{G}_{1,k}}| = \epsilon(\lambda)$, then there would exist an algorithm $\mathcal{B}_2$ such that $\mathsf{Adv}_{\mathcal{B}_2}^{\mathrm{DLIN}}(\lambda) = \epsilon(\lambda)$. Thus, if the DLIN assumption holds, then no* PPT *adversary $\mathcal{A}$ can distinguish between the games $\boldsymbol{G}_{1,k-1}$ and $\boldsymbol{G}_{1,k}$ with a non-negligible advantage.*

*Proof.* Suppose there exists an adversary $\mathcal{A}$ that distinguishes between the games $\mathbf{G}_{1,k-1}$ and $\mathbf{G}_{1,k}$ with a non-negligible advantage. We build a simulator $\mathcal{B}_2$ that uses $\mathcal{A}$ as a subroutine to solve the DLIN problem. $\mathcal{B}_2$ is given a challenge tuple $(g, f, \nu, g^{c_1}, f^{c_2}, z) \in \mathbb{G}^6$ where $z$ is either $\nu^{c_1+c_2}$ or random. Then $\mathcal{B}_2$ interacts with $\mathcal{A}$ as follows.

**Setup**:

- $\mathcal{B}_2$ first chooses random exponents $\alpha, a_1, a_2, y_{v_1}, y_{v_2}, y_w, y_{u_1}, \ldots, y_{u_{d_{max}}}, y_{h_1}, \ldots, y_{h_{d_{max}}}, (A_1, B_1), \ldots, (A_{d_{max}}, B_{d_{max}}) \in \mathbb{Z}_p$.
- It then sets the public parameters

$$\mathsf{PP} = \begin{pmatrix} g^b = f, g^{a_1}, g^{a_2}, g^{ba_1} = f^{a_1}, g^{ba_2} = f^{a_2}, \tau_1 = vv_1^{a_1} = g^{y_{v_1}a_1}, \tau_2 = vv_2^{a_2} = g^{y_{v_2}a_2}, \\ \tau_1^b = f^{y_{v_1}a_1}, \tau_2^b = f^{y_{v_2}a_2}, w^{a_1} = (fg^{y_w})^{a_1}, \\ \{u_i^{a_1} = (f^{-A_i}g^{y_{u_i}})^{a_1}, h_i^{a_1} = (f^{-B_i}g^{y_{h_i}})^{a_1}\}_{i=1}^{d_{max}}, e(g,g)^{\alpha a_1 b} = e(f,g)^{\alpha a_1} \end{pmatrix}$$

  and the master key

$$\mathsf{MK} = \begin{pmatrix} g, g^{\alpha}, g^{\alpha a_1}, v = \nu^{-a_1 a_2}, v_1 = \nu^{a_2}g^{y_{v_1}}, v_2 = \nu^{a_1}g^{y_{v_2}}, w = fg^{y_w}, \\ \{u_i = f^{-A_i}g^{y_{u_i}}, h_i = f^{-B_i}g^{y_{h_i}}\}_{i=1}^{d_{max}} \end{pmatrix}$$

- $\mathcal{B}_2$ gives $\mathsf{PP}$ to $\mathcal{A}$ and keeps $\mathsf{MK}$ to itself.

**Query 1**: $\mathcal{A}$ adaptively requests a private key for a time $T'$. If this is a $j$th query, then $\mathcal{B}_2$ handles this query as follows:

- If $j > k$, $\mathcal{B}_2$ will generate a normal private key by running **SUE.GenKey**. Note that $\mathcal{B}_2$ can do that since it knows the master secret key $\mathsf{MK}$.

- If $j < k$, $\mathcal{B}_2$ will prepare a semi-functional private key by calling **SUE.GenKeySF**. It can run this procedure since it can compute $g^{a_1 a_2}$.
- If $j = k$, $\mathcal{B}_2$ generates the private key as follows:
  - It first obtains a normal private key

$$\mathsf{SK}_{T'} = \left( \begin{array}{l} D_1 = g^{\alpha a_1} v^s, D_2 = g^{-\alpha} v_1^s g^{z_1}, D_3 = f^{-z_1}, D_4 = v_2^s g^{z_2}, \\ D_5 = f^{-z_2}, D_6 = f^{s_2}, D_7 = g^{s_1}, \\ \{K_i = (u_i^{L'|_i} w^{\mathsf{ktag}_i} h_i)^{s_1}\}_{i=1}^n, \{\mathsf{ktag}_i\}_{i=1}^n \end{array} \right)$$

  such that $\Psi(T') = L' \in \{0,1\}^n$, where it uses $\mathsf{ktag}_i = A_i L'|_i + B_i,\ i = 1, \ldots, n$ and the random exponents $s_1, s_2, z_1, z_2 \in \mathbb{Z}_p$.
  - It then sets

$$\overline{\mathsf{SK}}_{T'} = \left( \begin{array}{l} \overline{D}_1 = D_1 z^{-a_1 a_2}, \overline{D}_2 = D_2 z^{a_2} (g^{c_1})^{y_{v_1}}, \overline{D}_3 = D_3 (f^{c_2})^{y_{v_1}}, \\ \overline{D}_4 = D_4 z^{a_1} (g^{c_1})^{y_{v_2}}, \overline{D}_5 = D_5 (f^{c_2})^{y_{v_2}}, \overline{D}_6 = D_6 f^{c_2}, \overline{D}_7 = D_7 g^{c_1}, \\ \{\overline{K}_i = K_i (g^{c_1})^{y_{u_i} L'|_i + y_{h_i} + \mathsf{ktag}_i y_w}\}_{i=1}^n, \{\overline{\mathsf{ktag}}_i = \mathsf{ktag}_i\}_{i=1}^n \end{array} \right)$$

  We emphasize that, the fact that $\mathsf{ktag}_i = A_i L'|_i + B_i$ allows $\mathcal{B}_2$ to create the components $K_i$. In addition, we note that $\mathcal{B}_2$ implicitly changes $s_1, s_2, z_1, z_2$ used in $\mathsf{SK}_{T'}$ by $\overline{s}_1, \overline{s}_2, \overline{z}_1, \overline{z}_2$ to compute $\overline{\mathsf{SK}}_{T'}$ by setting $\overline{s}_1 = s_1 + c_1,\ \overline{s}_2 = s_2 + c_2,\ \overline{z}_1 = z_1 - y_{v_1} c_2,\ \overline{z}_2 = z_2 - y_{v_2} c_2$.
  - $\mathcal{B}_2$ returns $\overline{\mathsf{SK}}_{T'}$ to $\mathcal{A}$ as the query response.

**Challenge**: $\mathcal{A}$ submits a challenge time $T^*$ such that for all of its private key queries for times $T'$ in Query 1, it holds that $T' < T^*$. $\mathcal{B}_2$ proceeds as follows:

1. It first computes the label string $L^* = \Psi(T^*) \in \{0,1\}^d$ (say). Then it prepares the semi-functional ciphertext component $\widehat{\mathsf{CH}}^{(0)}$ as follows:
   - $\mathcal{B}_2$ computes a normal ciphertext component

$$\mathsf{CH}^{(0)} = \left( \begin{array}{l} C_1 = f^{r_1 + r_2}, C_2 = (f^{a_1})^{r_1}, C_3 = (g^{a_1})^{r_1}, C_4 = (f^{a_2})^{r_2}, \\ C_5 = (g^{a_2})^{r_2}, C_6 = \tau_1^{r_1} \tau_2^{r_2}, C_7 = (\tau_1^b)^{r_1} (\tau_2^b)^{r_2} (w^{a_1})^{-t'}, \\ \{E_{i,1} = [(u_i^{a_1})^{L^*|_i} (w^{a_1})^{\mathsf{ctag}_i} h_i^{a_1}]^{\mu_i}, E_{i,2} = (g^{a_1})^{\mu_i}\}_{i=1}^{d-1}, \\ \{E_{d,1} = [(u_d^{a_1})^{L^*} (w^{a_1})^{\mathsf{ctag}_d} h_d^{a_1}]^{\mu'_d}, E_{d,2} = (g^{a_1})^{\mu'_d}\}, \{\mathsf{ctag}_i\}_{i=1}^d \end{array} \right)$$

   under random exponents $\mu_1, \ldots, \mu_{d-1}, \mu'_d, r_1, r_2, t' = \sum_{i=1}^{d-1} \mu_i + \mu'_d \in \mathbb{Z}_p$. During this run it uses $\mathsf{ctag}_d = A_d L^* + B_d$ and other $\mathsf{ctag}_i$'s as random elements in $\mathbb{Z}_p$.
   - To make the components of $\mathsf{CH}^{(0)}$ semi-functional, $\mathcal{B}_2$ selects a random $x \in \mathbb{Z}_p$, implicitly sets $g^{\mu_d} = g^{\mu'_d} \nu^{a_2 x}$ which in turn implies $g^t = g^{t'} \nu^{a_2 x}$, and sets

$$\widehat{\mathsf{CH}}^{(0)} = \left( \begin{array}{l} \widehat{C}_1 = C_1, \widehat{C}_2 = C_2, \widehat{C}_3 = C_3, \widehat{C}_4 = C_4 f^{a_2 x}, \widehat{C}_5 = C_5 g^{a_2 x}, \\ \widehat{C}_6 = C_6 v_2^{a_2 x}, \widehat{C}_7 = C_7 f^{y_{v_2} a_2 x} \nu^{-a_1 a_2 y_w x}, \{\widehat{E}_{i,1} = E_{i,1}, \widehat{E}_{i,2} = E_{i,2}\}_{i=1}^{d-1}, \\ \{\widehat{E}_{d,1} = E_{d,1} \nu^{(y_{u_d} L^* + y_{h_d} + \mathsf{ctag}_d y_w) a_1 a_2 x}, \widehat{E}_{d,2} = E_{d,2} \nu^{a_1 a_2 x}\}, \\ \{\widehat{\mathsf{ctag}}_i = \mathsf{ctag}_i\}_{i=1}^d \end{array} \right)$$

2. For $1 \leq j \leq d$, $\mathcal{B}_2$ first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: If $L^{(j)} = L^*|_{d-j+1}$, it sets $\widehat{\mathsf{CH}}^{(j)}$ as an empty one. Otherwise, it constructs $\widehat{\mathsf{CH}}^{(j)}$ as discussed below.
   - $\mathcal{B}_2$ first obtains

$$\mathsf{CH}^{(j)} = \left( \begin{array}{l} C'_1 = f^{r_1 + r_2}, C'_2 = (f^{a_1})^{r_1}, C'_3 = (g^{a_1})^{r_1}, C'_4 = (f^{a_2})^{r_2}, \\ C'_5 = (g^{a_2})^{r_2}, C'_6 = \tau_1^{r_1} \tau_2^{r_2}, C'_7 = (\tau_1^b)^{r_1} (\tau_2^b)^{r_2} (w^{a_1})^{-t'''}, \\ \{E'_{i,1} = [(u_i^{a_1})^{L^{(j)}|_i} (w^{a_1})^{\mathsf{ctag}'_i} h_i^{a_1}]^{\mu'_i}, E'_{i,2} = (g^{a_1})^{\mu'_i}\}_{i=1}^{d-j}, \\ \{E'_{d-j+1,1} = [(u_{d-j+1}^{a_1})^{L^{(j)}} (w^{a_1})^{\mathsf{ctag}'_{d-j+1}} h_{d-j+1}^{a_1}]^{\mu''_{d-j+1}}, \\ E'_{d-j+1,2} = (g^{a_1})^{\mu''_{d-j+1}}\}, \{\mathsf{ctag}'_i\}_{i=1}^{d-j+1} \end{array} \right)$$

where it uses $\mu'_1 = \mu_1, \ldots, \mu'_{d-j} = \mu_{d-j}, r_1, r_2, \mathsf{ctag}'_1 = \mathsf{ctag}_1, \ldots, \mathsf{ctag}'_{d-j} = \mathsf{ctag}_{d-j}$ together with fresh random $\mu''_{d-j+1} \in \mathbb{Z}_p$, $\mathsf{ctag}'_{d-j+1} = A_{d-j+1}L^{(j)} + B_{d-j+1}$ and defines $t''' = \sum_{i=1}^{d-j} \mu'_i + \mu''_{d-j+1}$.

- Next, $\mathcal{B}_2$ modifies $\mathsf{CH}^{(j)}$ in the same way as in the case of $\widehat{\mathsf{CH}}^{(0)}$ using the same $x \in \mathbb{Z}_p$ and implicitly setting $g^{\mu'_{d-j+1}} = g^{\mu''_{d-j+1}}\nu^{a_2 x}$, which in turn sets $g^{t''} = g^{t'''}\nu^{a_2 x}$, obtaining

$$\widehat{\mathsf{CH}}^{(j)} = \begin{pmatrix} \widehat{C}'_1 = C'_1, \widehat{C}'_2 = C'_2, \widehat{C}'_3 = C'_3, \widehat{C}'_4 = C'_4 f^{a_2 x}, \widehat{C}'_5 = C'_5 g^{a_2 x}, \\ \widehat{C}'_6 = C'_6 v_2^{a_2 x}, \widehat{C}'_7 = C'_7 f^{y_{v_2} a_2 x} \nu^{-a_1 a_2 y_w x}, \{\widehat{E}'_{i,1} = E'_{i,1}, \widehat{E}'_{i,2} = E'_{i,2}\}_{i=1}^{d-j}, \\ \{\widehat{E}'_{d-j+1,1} = E'_{d-j+1,1}\nu^{(y_{u_{d-j+1}}L^{(j)} + y_{h_{d-j+1}} + \mathsf{ctag}'_{d-j+1} y_w) a_1 a_2 x}, \\ \widehat{E}'_{d-j+1,2} = E'_{d-j+1,2}\nu^{a_1 a_2 x}\}, \{\widehat{\mathsf{ctag}}'_i = \mathsf{ctag}'_i\}_{i=1}^{d-j+1} \end{pmatrix}$$

- It also prunes redundant elements in $\widehat{\mathsf{CH}}^{(j)}$.

3. $\mathcal{B}_2$ removes all empty $\widehat{\mathsf{CH}}^{(j)}$ and sets $\widehat{\mathsf{CH}}_{T^*} = (\widehat{\mathsf{CH}}^{(0)}, \ldots, \widehat{\mathsf{CH}}^{(d')})$ for some $d' < d$ that consists of nonempty $\widehat{\mathsf{CH}}^{(j)}$.
4. It sets the challenge ciphertext header as $\mathsf{CH}_{T^*} = \widehat{\mathsf{CH}}_{T^*}$ and the session key $\mathsf{EK}^* = e(f, g)^{\alpha a_1 r_2}$.
5. $\mathcal{B}_2$ flips a random coin $\beta \in \{0, 1\}$ and gives $\mathsf{CH}_{T^*}$ and $\mathsf{EK}^*$ to $\mathcal{A}$ if $\beta = 0$. Otherwise, it gives $\mathsf{CH}_{T^*}$ and a random session key to $\mathcal{A}$.

**Query 2**: Same as Query 1 with the same restriction that $T' < T^*$ for all of $\mathcal{A}$'s private key queries for times $T'$.

**Guess**: $\mathcal{A}$ outputs a guess $\beta'$. If $\beta = \beta'$, then $\mathcal{B}_2$ outputs 1. Otherwise, it outputs 0.

If $z = \nu^{c_1 + c_2}$, we are in game $\mathbf{G}_{1,k-1}$ since in this case the $k$th query results in a normal private key as in equation (1) under randomness $\overline{s}_1 = s_1 + c_1$, $\overline{s}_2 = s_2 + c_2$. Otherwise, if $z$ is a random group element, then we can write $z = \nu^{c_1 + c_2} g^{\gamma}$ for random $\gamma \in \mathbb{Z}_p$. This makes the $k$th queried key semi-functional as in equation (5), where $\gamma$ is the added randomness to make it semi-functional and thus we are in game $\mathbf{G}_{1,k}$. This completes the proof. □

**Lemma 5.** *If there exists an adversary $\mathcal{A}$ such that $|\mathsf{Adv}_{\mathcal{A}}^{G_2} - \mathsf{Adv}_{\mathcal{A}}^{G_3}| = \epsilon(\lambda)$, then there would exist an algorithm $\mathcal{B}_3$ such that $\mathsf{Adv}_{\mathcal{B}_3}^{\mathrm{DBDH}}(\lambda) = \epsilon(\lambda)$. Thus, if the DBDH assumption hold, then no* PPT *adversary can distinguish between the games $\mathbf{G}_2$ and $\mathbf{G}_3$ with a non-negligible advantage.*

*Proof.* Suppose there exists an adversary $\mathcal{A}$ that distinguishes the game $\mathbf{G}_2$ from the game $\mathbf{G}_3$ with a non-negligible advantage. A simulator $\mathcal{B}_3$ that attempts to solve the DBDH problem using $\mathcal{A}$ as a subroutine is constructed below. $\mathcal{B}_3$ is given a challenge tuple $(g, g^{c_1}, g^{c_2}, g^{c_3}, z) \in \mathbb{G}^4 \times \mathbb{G}_T$ where $z$ is either $e(g, g)^{c_1 c_2 c_3}$ or random. $\mathcal{B}_3$ interacts with $\mathcal{A}$ as follows:

**Setup**:

- $\mathcal{B}_3$ begins by selecting random exponents $a_1, b, y_v, y_{v_1}, y_{v_2}, y_w, y_{u_1}, \ldots, y_{u_{d_{max}}}, y_{h_1}, \ldots, y_{h_{d_{max}}} \in \mathbb{Z}_p$.
- It then sets

$$\mathsf{PP} = \begin{pmatrix} g^b, g^{a_1}, g^{a_2} = g^{c_2}, g^{ba_1}, g^{ba_2} = (g^{c_2})^b, \tau_1 = g^{y_v + y_{v_1} a_1}, \tau_2 = g^{y_v}(g^{c_2})^{y_{v_2}}, \\ \tau_1^b, \tau_2^b, w^{a_1} = g^{y_w a_1}, \{u_i^{a_1} = g^{y_{u_i} a_1}, h_i^{a_1} = g^{y_{h_i} a_1}\}_{i=1}^{d_{max}}, e(g, g)^{\alpha a_1 b} = e(g^{c_1}, g^{c_2})^{a_1 b} \end{pmatrix}$$

We point out that $\mathcal{B}_3$ implicitly sets $\alpha = c_1 c_2$ and $a_2 = c_2$. Due to this setting, the master key components $g^{\alpha}$ and $g^{\alpha a_1}$ will not be directly available to $\mathcal{B}_3$. However, it sets the other components of the master secret key $\mathsf{MK}$ as

$$g, v = g^{y_v}, v_1 = g^{y_{v_1}}, v_2 = g^{y_{v_2}}, w = g^{y_w}, \{u_i = g^{y_{u_i}}, h_i = g^{y_{h_i}}\}_{i=1}^{d_{max}}.$$

- It publishes the public parameters $\mathsf{PP}$.

**Query 1**: $\mathcal{A}$ adaptively requests a private key for a time $T'$. To answer, $\mathcal{B}_3$ computes $L' = \Psi(T') \in \{0,1\}^n$ (say) and proceeds as follows to generate a semi-functional private key.

- It chooses random $s_1, s_2, z_1, z_2, \gamma', \mathsf{ktag}_1, \ldots, \mathsf{ktag}_n \in \mathbb{Z}_p$ and defines $s = s_1 + s_2$. It aims to implicitly set the variable $\gamma = c_1 + \gamma'$.
- It creates the semi-functional private key

$$\widehat{\mathsf{SK}}_{T'} = \begin{pmatrix} \widehat{D}_1 = (g^{c_2})^{-\gamma' a_1} v^s, \widehat{D}_2 = (g^{c_2})^{\gamma'} v_1^s g^{z_1}, \widehat{D}_3 = (g^b)^{-z_1}, \widehat{D}_4 = (g^{c_1})^{a_1} g^{a_1 \gamma'} v_2^s g^{z_2}, \\ \widehat{D}_5 = (g^b)^{-z_2}, \widehat{D}_6 = g^{b s_2}, \widehat{D}_7 = g^{s_1}, \\ \{\widehat{K}_i = [u_i^{L'|_i} w^{\mathsf{ktag}_i} h_i]^{s_1}\}_{i=1}^n, \{\widehat{\mathsf{ktag}}_i = \mathsf{ktag}_i\}_{i=1}^n \end{pmatrix}$$

**Challenge**: In the challenge step $\mathcal{A}$ submits a challenge time $T^*$ such that for all of its private key queries for times $T'$ in Query 1, it holds that $T' < T^*$. $\mathcal{B}_3$ proceeds as follows:

1. At first, $\mathcal{B}_3$ computes the label string $L^* = \Psi(T^*) \in \{0,1\}^d$ (say) and generates semi-functional ciphertext component $\widehat{\mathsf{CH}}^{(0)}$ as follows:
   - It selects random $\mu_1, \ldots, \mu_d, r_1, \mathsf{ctag}_1, \ldots, \mathsf{ctag}_d, x' \in \mathbb{Z}_p$ and sets $t = \sum_{i=1}^{d} \mu_i$. It will implicitly let $r_2 = c_3$ and $x = -c_3 + x'$.
   - Next it computes

$$\widehat{\mathsf{CH}}^{(0)} = \begin{pmatrix} \widehat{C}_1 = g^{b r_1} (g^{c_3})^b, \widehat{C}_2 = g^{b a_1 r_1}, \widehat{C}_3 = g^{a_1 r_1}, \widehat{C}_4 = (g^{c_2})^{b x'}, \widehat{C}_5 = (g^{c_2})^{x'}, \\ \widehat{C}_6 = \tau_1^{r_1} (g^{c_3})^{y_v} (g^{c_2})^{y_{v_2} x'}, \widehat{C}_7 = (\tau_1^b)^{r_1} (g^{c_3})^{y_v b} (g^{c_2})^{y_{v_2} b x'} w^{-a_1 t}, \\ \{\widehat{E}_{i,1} = [u_i^{L^*|_i} w^{\mathsf{ctag}_i} h_i]^{a_1 \mu_i}, \widehat{E}_{i,2} = g^{a_1 \mu_i}\}_{i=1}^d, \{\widehat{\mathsf{ctag}}_i = \mathsf{ctag}_i\}_{i=1}^d \end{pmatrix}$$

2. For $1 \leq j \leq d$, $\mathcal{B}_3$ first sets $L^{(j)} = L^*|_{d-j} \| 1$ and proceeds as follows: If $L^{(j)} = L^*|_{d-j+1}$, it sets $\widehat{\mathsf{CH}}^{(j)}$ as an empty one. Otherwise, it computes $\widehat{\mathsf{CH}}^{(j)}$ in the same way as $\widehat{\mathsf{CH}}^{(0)}$ by performing the following steps.
   - It selects fresh random $\mu'_{d-j+1}, \mathsf{ctag}'_{d-j+1} \in \mathbb{Z}_p$.
   - It computes the semi-functional ciphertext component

$$\widehat{\mathsf{CH}}^{(j)} = \begin{pmatrix} \widehat{C}'_1 = g^{b r_1} (g^{c_3})^b, \widehat{C}'_2 = g^{b a_1 r_1}, \widehat{C}'_3 = g^{a_1 r_1}, \widehat{C}'_4 = (g^{c_2})^{b x'}, \widehat{C}'_5 = (g^{c_2})^{x'}, \\ \widehat{C}'_6 = \tau_1^{r_1} (g^{c_3})^{y_v} (g^{c_2})^{y_{v_2} x'}, \widehat{C}'_7 = (\tau_1^b)^{r_1} (g^{c_3})^{y_v b} (g^{c_2})^{y_{v_2} b x'} w^{-a_1 t'}, \\ \{\widehat{E}'_{i,1} = [u_i^{L^{(j)}|_i} w^{\mathsf{ctag}'_i} h_i]^{a_1 \mu'_i}, \widehat{E}'_{i,2} = g^{a_1 \mu'_i}\}_{i=1}^{d-j+1}, \{\widehat{\mathsf{ctag}}'_i = \mathsf{ctag}'_i\}_{i=1}^{d-j+1} \end{pmatrix}$$

     where it uses $\mu'_1 = \mu_1, \ldots, \mu'_{d-j} = \mu_{d-j}, \mathsf{ctag}'_1 = \mathsf{ctag}_1, \ldots, \mathsf{ctag}'_{d-j} = \mathsf{ctag}_{d-j}$ together with the same $x'$ that were used for computing $\widehat{\mathsf{CH}}^{(0)}$, defines $t' = \sum_{i=1}^{d-j+1} \mu'_i$, and implicitly sets $r_2 = c_3,\ x = -c_3 + x'$.
   - It also prunes redundant components from $\widehat{\mathsf{CH}}^{(j)}$.
3. $\mathcal{B}_3$ removes all empty $\widehat{\mathsf{CH}}^{(j)}$ and sets $\widehat{\mathsf{CH}}_{T^*} = (\widehat{\mathsf{CH}}^{(0)}, \ldots, \widehat{\mathsf{CH}}^{(d')})$ for some $d' < d$ that consists of nonempty $\widehat{\mathsf{CH}}^{(j)}$.
4. $\mathcal{B}_3$ sets the challenge ciphertext header as $\mathsf{CH}_{T^*} = \widehat{\mathsf{CH}}_{T^*}$ and the challenge session key $\mathsf{EK}^* = z^{a_1 b}$.
5. Finally $\mathcal{B}_3$ gives $\mathsf{CH}_{T^*}$ and $\mathsf{EK}^*$ to $\mathcal{A}$.

**Query 2**: Same as Query 1 with the same restriction that for all the private key queries for times $T'$ in this phase also $T' < T^*$.

**Guess**: $\mathcal{A}$ outputs a guess $\beta' \in \{0,1\}$. $\mathcal{B}_3$ also outputs $\beta'$.

One can verify that the semi-functional private keys and the semi-functional ciphertext header are correctly distributed as in equations (5), (6) and (7). If $z = e(g,g)^{c_1 c_2 c_3}$, then the session key is properly distributed as in the game $\mathbf{G}_2$. Otherwise, the session key is random element as in the game $\mathbf{G}_3$. This completes the proof. □

## 5 Efficiency

Table 1 and 2 present the comparison between our fully secure SUE scheme and that of [10], the only SUE scheme with full security available in the literature, in terms of communication together with storage and computation respectively. The other previously known SUE schemes, viz., the prime order constructions of [10], [9] are only selectively secure and hence we do not consider them for efficiency comparison. Let us first concentrate on the communication and storage efficiency. Just by looking at Table 1, it may appear that the communication efficiency of our scheme is worse than that of [10]. However, our scheme is built on prime order bilinear group as opposed to [10] which is built on composite order bilinear group and this difference in the group order results in a complete turn-around of the situation. As noted in [7], the only known instantiations of composite order bilinear group use elliptic curves (or more generally, abelian varieties) over finite fields. Since the elliptic curve group order must be infeasible to factor, it must be at least 1024 bits. On the other hand, the size of a prime order elliptic curve group that provides an equivalent level of security is only 160 bits which is more than six times smaller. Thus, when compared in terms of bit length, we can readily infer from Table 1 that both the ciphertext and key sizes in our SUE scheme are roughly one-third of the corresponding sizes in [10]. Our public parameters contain almost half the number of group elements in the public parameters of [10]. Regarding the master key size of our SUE scheme, note that, although it is not constant, it grows moderately with the maximum time bound in the system. For instance, if the time periods are taken to be months, then the maximum time bound $T_{max} = 2^{17}$ is more than 850 years, and in this case our master key would involve only 41 group elements.

Table 1: Communication and Storage Comparison

| SUE | $\vert\mathcal{G}\vert$ | Complexity Assumptions | $\vert\mathsf{PP}\vert$ | $\vert\mathsf{MK}\vert$ | $\vert\mathsf{SK}_T\vert$ | $\vert\mathsf{CH}_T\vert$ |
|---|---|---|---|---|---|---|
| [10] | $n$ (composite) | Variants of SD,<br>Composite DH | $4k+2$ in $\mathbb{G}$,<br>1 in $\mathbb{G}_T$ | 1 in $\mathbb{G}$,<br>1 in $\mathbb{Z}_n$ | $k+2$ in $\mathbb{G}$ | $3k+2$ in $\mathbb{G}$ |
| Ours | $p$ (prime) | DLIN, DBDH | $2k+10$ in $\mathbb{G}$,<br>1 in $\mathbb{G}_T$ | $2k+7$ in $\mathbb{G}$ | $k+7$ in $\mathbb{G}$,<br>$k$ in $\mathbb{Z}_p$ | $5k+7$ in $\mathbb{G}$,<br>$2k$ in $\mathbb{Z}_p$ |

Here, $k = \log T_{max}$ where $T_{max}$ is the maximum time in the system.

Table 2: Computation Comparison

| SUE | $\vert\mathcal{G}\vert$ | **SUE.Setup** | **SUE.GenKey** | **SUE.Encrypt** | **SUE.ReEncrypt** | **SUE.Decrypt** |
|---|---|---|---|---|---|---|
| [10] | $n$<br>(composite) | 1 in $\mathbb{G}_T$; 1 | $2k+3$ in $\mathbb{G}$ | $4k+2$ in $\mathbb{G}$,<br>1 in $\mathbb{G}_T$ | $4k+6$ in $\mathbb{G}$,<br>1 in $\mathbb{G}_T$ | $2k-2$ in $\mathbb{G}$;<br>$k+2$ |
| Ours | $p$ (prime) | $2k+12$ in $\mathbb{G}$,<br>1 in $\mathbb{G}_T$; 1 | $3k+9$ in $\mathbb{G}$ | $9k+10$ in $\mathbb{G}$,<br>1 in $\mathbb{G}_T$ | $9k+20$ in $\mathbb{G}$,<br>1 in $\mathbb{G}_T$ | $k$ in $\mathbb{G}$;<br>$2k+7$ |

Here, $k = \log T_{max}$ where $T_{max}$ is the maximum time in the system.
In this table, '$x; y$' signifies '$x$ many exponentiations and $y$ many pairings'.

Table 2 displays the computational cost of our SUE scheme in comparison with that of [10]. Here also our scheme is benefited from the use of prime order bilinear group. Due to the excessive bit length of the group order, group operations and pairing computations are prohibitively slow on composite order elliptic curves [7], [11]. In particular, an exponentiation is at least six times slower and a pairing computation is roughly 50 times slower on a 1024 bit composite order elliptic curve than the corresponding operations on a comparable prime order curve [7]. In this light, we can rapidly obtain from Table 2 that the **SUE.GenKey** algorithm of our SUE scheme is almost

4 times faster, the **SUE.Encrypt** and the **SUE.ReEncrypt** algorithms are almost 3 times faster, as well as, the **SUE.Decrypt** algorithm is roughly 25 times faster than the respective algorithms in [10]. Finally, following [8], if we assume that in prime order elliptic curve groups, a pairing is equivalent to six exponentiations, then our **SUE.Setup** algorithm would have lower computational complexity compared to that of [10] even for $T_{max} = 2^{200}$. However, we note that in last few years there has been significant improvement in the efficiency of pairing computation. Despite this we may conclude that our **SUE.Setup** algorithm would still have lower complexity than [10] for sufficiently large value of $T_{max}$.

*Remark 3.* Note that, Freeman [7] and Lewko [11] have developed abstract frameworks to simulate the key properties of bilinear groups of composite order that are required to construct secure pairing based cryptosystems in prime order groups. Their tools are not generic scheme conversions in the sense that they do not transform any composite order cryptographic protocol to a prime order protocol. On the contrary, their techniques serve in replicating certain tricks in the security proofs of independent prime order protocol designs in a manner analogous to that of their composite order variants. In the security proof of our SUE scheme, we have incorporated the required features from the underlying prime order bilinear groups in a different fashion. Further, we have developed a *tag based approach* to overcome certain subtility which they addressed using *nominal semi-functionality* [11].

*Remark 4.* As mentioned earlier in the paper, the work of Sahai et al. [15] can be viewed as a restricted SUE. However, Lee et al. [10] have already shown that their SUE scheme is more efficient than [15] in terms of both communication and storage, as well as, computation. Thus, our scheme having better performance guarantee compared to [10], naturally outperforms [15]. On the other hand, although achieving a similar goal, SUE should not be confused with the notion of updatable encryption introduced by Boneh et al. [4]. An updatable encryption scheme supports the functionality of re-encrypting a certain data encrypted under some symmetric key to a fresh encryption of the same data under another key. On the contrary, SUE associates ciphertexts with time and not with keys and thus it has more flexibility. Furthermore, in case of updatable encryption re-encrypting a ciphertext requires a re-encryption key derived from the current encryption key and the target one, whereas, in SUE ciphertexts are publicly updatable without requiring any re-encryption key.

## 6 Conclusion

In this paper, we have designed an SUE scheme in prime order bilinear groups and proved its full CPA security under standard assumptions, namely, the DLIN and the DBDH assumptions. To the best of our knowledge, our SUE scheme is the *first* to achieve full security in prime order bilinear group setting. We have employed the dual system encryption technique of Lewko and Waters [13], [16]. In our SUE scheme, the master key size is not constant. Also the ciphertext and key sizes are not very short because the tag values could not be compressed. Developing an SUE scheme that overcomes these difficulties could be an interesting future work. As noted earlier, an SUE scheme cannot be CCA2 secure. However, one may attempt to analyse the possible relaxations of CCA2 security for SUE and design an SUE scheme, secure in that relaxed CCA2 security model, in prime order bilinear groups under standard assumptions.

## References

1. Boldyreva, A., Goyal, V., Kumar, V.: Identity-based encryption with efficient revocation. In: Proceedings of the 15th ACM conference on Computer and communications security. pp. 417–426. ACM (2008)
2. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. In: Advances in Cryptology–CRYPTO 2004. pp. 41–55. Springer (2004)

3. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Advances in CryptologyCRYPTO 2001. pp. 213–229. Springer (2001)
4. Boneh, D., Lewi, K., Montgomery, H., Raghunathan, A.: Key homomorphic prfs and their applications. In: Advances in Cryptology–CRYPTO 2013, pp. 410–428. Springer (2013)
5. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: Advances in CryptologyEurocrypt 2003, pp. 255–271. Springer (2003)
6. Dodis, Y., Katz, J., Xu, S., Yung, M.: Key-insulated public key cryptosystems. In: Advances in CryptologyEUROCRYPT 2002. pp. 65–82. Springer (2002)
7. Freeman, D.M.: Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In: Advances in Cryptology–EUROCRYPT 2010, pp. 44–61. Springer (2010)
8. Hohenberger, S.: Advances in signatures, encryption, and e-cash from bilinear groups. Ph.D. thesis, Citeseer (2006)
9. Lee, K.: Self-updatable encryption with short public parameters and its extensions Available at `http://eprint.iacr.org/2014/231.pdf`
10. Lee, K., Choi, S.G., Lee, D.H., Park, J.H., Yung, M.: Self-updatable encryption: Time constrained access control with hidden attributes and better efficiency. In: Advances in Cryptology-ASIACRYPT 2013, pp. 235–254. Springer (2013)
11. Lewko, A.: Tools for simulating features of composite order bilinear groups in the prime order setting. In: Advances in Cryptology–EUROCRYPT 2012, pp. 318–335. Springer (2012)
12. Lewko, A., Okamoto, T., Sahai, A., Takashima, K., Waters, B.: Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In: Advances in Cryptology–EUROCRYPT 2010, pp. 62–91. Springer (2010)
13. Lewko, A., Waters, B.: New techniques for dual system encryption and fully secure hibe with short ciphertexts. In: Theory of Cryptography, pp. 455–479. Springer (2010)
14. Paterson, K.G., Quaglia, E.A.: Time-specific encryption. In: Security and Cryptography for Networks, pp. 1–16. Springer (2010)
15. Sahai, A., Seyalioglu, H., Waters, B.: Dynamic credentials and ciphertext delegation for attribute-based encryption. In: Advances in Cryptology–CRYPTO 2012, pp. 199–217. Springer (2012)
16. Waters, B.: Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In: Advances in Cryptology-CRYPTO 2009, pp. 619–636. Springer (2009)